虚拟形象 SDK

SDK 集成指引

腾讯云

文档目录

SDK 集成指引
快速开始
Android

最近更新时间：2023-05-19 11:06:22

集成方式

手动集成（资源内置）

下载SDK

下载 SDK，并解压。不同套餐包内的资源文件略有差异，详细说明见 附件。
建议您将包内的 demo 工程导入 AndroidStudio 跑起来，以便快速熟悉接口使用方法。请参考快速跑通 Demo。

集成

- 添加下载的 xmagic-xxxx `.aar` 文件到 app 工程 `libs` 目录下。
- 将 SDK 包内的 assets/ 目录下的全部资源拷贝到 `../src/main/assets` 目录下，如果 SDK 包中的 MotionRes 文件夹内有资源，将此文件夹也拷贝到 `../src/main/assets` 目录下 。
- 将 jniLibs 文件夹拷贝到工程的 `../src/main/jniLibs` 目录下。

导入方法

打开 app 模块的 `build.gradle` 添加依赖引用：

```
android{
    ...
    defaultConfig {
        applicationId "修改成与授权lic绑定的包名"
        ....
    }
    packagingOptions {
        pickFirst '**/libc++_shared.so'
    }
}

dependencies{
    ...
    compile fileTree(dir: 'libs', include: ['*.jar','*.aar'])//添加 *.aar
}
```

> **注意**
> 如果项目中没有集成 Google 的 Gson 库则还需添加如下依赖：

```
dependencies{
    implementation 'com.google.code.gson:gson:2.8.2'
    //2.6.0 版本及之后需要添加
    implementation 'androidx.exifinterface:exifinterface:1.3.3'
}
```

手动集成（资源动态下载）

动态下载 assets、so、动效资源指引

为了减少包大小，您可以将 SDK 所需的 assets 资源、so 库、以及动效资源 MotionRes（部分基础版 SDK 无动效资源）改为联网下载。在下载成功后，将上述文件的路径设置给 SDK。
我们建议您复用 Demo 的下载逻辑，当然，也可以使用您已有的下载服务。动态下载的详细指引，请参见 SDK 包体瘦身（Android）。

Maven 集成

腾讯特效 SDK 已经发布到 mavenCentral 库，您可以通过配置 gradle 自动下载更新。
1. 在 dependencies 中添加腾讯特效 SDK 的依赖。

```
dependencies {
 //例如：S1-04套餐如下
 implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

2. 在 defaultConfig 中，指定 App 使用的 CPU 架构。

```
defaultConfig {
 ndk {
     abiFilters "armeabi-v7a", "arm64-v8a"
 }
}
```

> **说明**
> 目前 特效 SDK 支持 armeabi-v7a 和 arm64-v8a。

3. 单击 Sync Now，自动下载 SDK 并集成到工程里。
4. 如果您的套餐包含动效和滤镜功能，那么需要在 SDK 下载 下载对应的套餐包，将包内免费的动效和滤镜素材放置在您工程下的如下目录：
   - 动效：`../assets/MotionRes`
   - 滤镜：`../assets/lut`

各套餐对应的 Maven 地址

版本	Maven 地址
A1 – 01	implementation 'com.tencent.mediacloud:TencentEffect_A1-01:latest.release'
A1 – 02	implementation 'com.tencent.mediacloud:TencentEffect_A1-02:latest.release'
A1 – 03	implementation 'com.tencent.mediacloud:TencentEffect_A1-03:latest.release'
A1 – 04	implementation 'com.tencent.mediacloud:TencentEffect_A1-04:latest.release'
A1 – 05	implementation 'com.tencent.mediacloud:TencentEffect_A1-05:latest.release'
A1 – 06	implementation 'com.tencent.mediacloud:TencentEffect_A1-06:latest.release'
S1 – 00	implementation 'com.tencent.mediacloud:TencentEffect_S1-00:latest.release'
S1 – 01	implementation 'com.tencent.mediacloud:TencentEffect_S1-01:latest.release'
S1 – 02	implementation 'com.tencent.mediacloud:TencentEffect_S1-02:latest.release'
S1 – 03	implementation 'com.tencent.mediacloud:TencentEffect_S1-03:latest.release'
S1 – 04	implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'

SDK使用流程

步骤一：鉴权

1. 申请授权，得到 License URL 和 License KEY，请参见 License 指引 。

> **注意**
> 正常情况下，只要 App 成功联网一次，就能完成鉴权流程，因此您**不需要**把 License 文件放到工程的 assets 目录里。但是如果您的 App 在从未联网的情况下也需要使用 SDK 相关功能，那么您可以把 License 文件下载下来放到 assets 目录，作为保底方案，此时 License 文件名必须是 `v_cube.license` 。

2. 在相关业务模块的初始化代码中设置 URL 和 KEY，触发 License 下载，避免在使用前才临时去下载。也可以在 Application 的 onCreate 方法里触发下载，但不建议，因为此时可能没有网络权限或联网失败率较高。

```
//如果仅仅是为了触发下载或更新license，而不关心鉴权结果，则第4个参数传入null。
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, null);
```

3. 然后在真正要使用美颜功能前(例如 Demo 的 TEMenuActivity `.java`)，再去做鉴权：

```
// 如果您的so库是从网络下载的，那么请在调用TELicenseCheck.getInstance().setTELicense之前，先设置so的路径，否则鉴权会失败。
// XmagicApi.setLibPathAndLoad(validLibsDirectory);
// 如果您的so内置在apk包内，则无需调用上面的方法。
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, new TELicenseCheckListener() {

        @Override
        public void onLicenseCheckFinish(int errorCode, String msg) {
            //注意：此回调不一定在调用线程
            if (errorCode == TELicenseCheck.ERROR_OK) {
                //鉴权成功
            } else {
                //鉴权失败
            }
        }
    });
```

鉴权 errorCode 说明：

错误码	说明
0	成功。Success
-1	输入参数无效，例如 URL 或 KEY 为空
-3	下载环节失败，请检查网络设置
-4	从本地读取的 TE 授权信息为空，可能是 IO 失败引起
-5	读取 VCUBE TEMP License文件内容为空，可能是 IO 失败引起
-6	v_cube.license 文件 JSON 字段不对。请联系腾讯云团队处理
-7	签名校验失败。请联系腾讯云团队处理
-8	解密失败。请联系腾讯云团队处理
-9	TELicense 字段里的 JSON 字段不对。请联系腾讯云团队处理
-10	从网络解析的 TE 授权信息为空。请联系腾讯云团队处理

-11	把TE授权信息写到本地文件时失败，可能是 IO 失败引起
-12	下载失败，解析本地 asset 也失败
-13	鉴权失败，请检查 so 是否在包里，或者已正确设置 so 路径
3004/3005	无效授权。请联系腾讯云团队处理
3015	Bundle Id / Package Name 不匹配。检查您的 App 使用的 Bundle Id / Package Name 和申请的是否一致，检查是否使用了正确的授权文件
3018	授权文件已过期，需要向腾讯云申请续期
其他	请联系腾讯云团队处理

步骤二：资源拷贝

1. 如果您的资源文件是内置在 assets 目录的，那么使用前需要 copy 到 app 的私有目录。您可以提前 copy 好，或者在上一步鉴权成功的回调里执行拷贝操作。示例代码在 Demo 的 `TEMenuActivity.java` 。

```
XmagicResParser.setResPath(new File(getFilesDir(), "xmagic").getAbsolutePath());
//loading

//copy资源文件到私有目录，只需要做一次
XmagicResParser.copyRes(getApplicationContext());
```

2. 如果您的资源文件是从 网络动态下载 的，下载成功后，需要设置资源文件路径。示例代码在 Demo 的 TEMenuActivity `.java` 。

```
XmagicResParser.setResPath(下载的资源文件本地路径);
```

步骤三：SDK 初始化及使用方法

使用腾讯特效 SDK 生命周期大致如下：

1. 构造美颜 UI 数据，可参考 Demo 工程（com.tencent.demo.beauty.provider包下代码）。
2. 预览布局中添加 Demo 中的 GLCameraXView。

```
<com.tencent.demo.camera.camerax.GLCameraXView
   android:id="@+id/te_camera_layout_camerax_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:back_camera="false"
   app:surface_view="false"
   app:transparent="true" />
```

3. （可选）快速实现相机。
   将 Demo 工程中的 com.tencent.demo.camera 目录拷贝到工程中。利用 GLCameraXView 类快速实现相机功能。详细实现可参考 Demo 工程的 TECameraActivity `.java` 。

```
@Override
public int onCustomProcessTexture(int textureId, int textureWidth, int textureHeight) {
    int resultTextureId = 0;
    if (!isUseTencentEffect) {
         resultTextureId = textureId;
     } else {
         resultTextureId = mBeautyImpl.process(textureId, textureWidth, textureHeight);
       }
    return resultTextureId;
  }
```

4. 初始化美颜 SDK，建议放在 Activity 的 `onResume()` 方法中。

```
mXmagicApi = new XmagicApi(this, XmagicResParser.getResPath(),new
XmagicApi.OnXmagicPropertyErrorListener());
```

参数

参数	含义
Context context	上下文
String resDir	资源文件目录，详见请参见 步骤二
OnXmagicPropertyErrorListener errorListener	回调函数实现类

返回

错误码含义请参见 API 文档。

5. 添加素材提示语回调函数（方法回调有可能运行在子线程），部分素材会提示用户：点点头、伸出手掌、比心，这个回调就是用于展示类似的提示语。

```
mXmagicApi.setTipsListener(new XmagicTipsListener() {
 final XmagicToast mToast = new XmagicToast();
 @Override
 public void tipsNeedShow(String tips, String tipsIcon, int type, int duration) {
     mToast.show(MainActivity.this, tips, duration);
 }

 @Override
 public void tipsNeedHide(String tips, String tipsIcon, int type) {
     mToast.dismiss();
 }
});
```

6. 美颜 SDK 处理每帧数据并返回相应处理结果。

```
int outTexture = mXmagicApi.process(textureId, textureWidth, textureHeight);
```

7. 更新指定类型的美颜特效数值。

```
// 可用的入参属性可以从 XmagicResParser.parseRes() 获得
mXmagicApi.updateProperty(XmagicProperty<?> p);
```

8. Pause美颜 SDK，建议与 Activity 的 `onPause()` 生命周期绑定。

```
//在 Activity 的 onPause 时调用，需要在 OpenGL 线程调用
mXmagicApi.onPause();
```

9. 释放美颜 SDK，建议与 Activity 的 `onDestroy()` 生命周期绑定。

```
//注意，此方法需要在GL线程中调用
mXmagicApi.onDestroy();
```

步骤四：混淆配置

- 如果您在打 release 包时，启用了编译优化（把 minifyEnabled 设置为 true），会裁掉一些未在 java 层调用的代码，而这些代码有可能会被 native 层调用，从而引起 `no xxx method` 的异常。
- 如果您启用了这样的编译优化，那就要添加这些 keep 规则，防止 xmagic 的代码被裁掉：

```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
-keep class androidx.exifinterface.** { *;}
```

附件（SDK文件结构）：

注意：
此表格列出了SDK用到的所有文件，可能您的套餐中没有某些文件，但并不影响该套餐功能的使用。

文件类型			说明
assets	audio2exp		avatar 虚拟人语音驱动模型，如果不使用该功能，则无需该模型
	benchmark		机型适配使用
	Light3DPlugin		3D 贴纸使用
	LightBodyPlugin	LightBody3DModel.bundle	人体 3D 骨骼点位使用
		LightBodyModel.bundle	美体功能使用
	LightCore		SDK 核心模型资源
	LightHandPlugin		手势贴纸、手部点位能力需要
	LightSegmentPlugin		背景分割能力需要使用
	lut		免费的滤镜资源
demo_xxx_android_xxxx			demo 工程
jniLibs	libace_zplan.so		3D 引擎库
	libaudio2exp.so		avatar 虚拟人语音驱动库，如果不使用该功能，则无需该库
	libc++_shared.so		libc++_shared.so 是一个 C++ 标准库的共享库，它提供了一组C++ 标准库函数和类，用于支持 C++ 程序的开发和运行。它在 Android 系统中被广泛使用，是 C++ 应用程序和库的重要组成部分。如果您的工程中已有 C++ 共享库，可以只保留一份
	liblight-sdk.so		light sdk 核心库
	libpag.so		light sdk 依赖的动画文件库
	libtecodec.so		light sdk 依赖的编解码库
	libv8jni.so		light sdk 依赖的用于解析 JavaScript 的库
	libYTCommonXMagic.so		license 鉴权使用
libs	xmagic-xxxx.aar		美颜 SDK 的 aar 文件
MotionRes	2dMotionRes		2D 贴纸
	3dMotionRes		3D 贴纸

	avatarRes	Avatar素材
	ganMotionRes	童趣贴纸
	handMotionRes	手势贴纸
	makeupRes	美妆贴纸
	segmentMotionRes	背景分割贴纸
unity	aar	unity 项目需要使用的桥接 aar
	module	桥接 aar 的原工程

iOS

最近更新时间：2023-02-20 11:01:53

集成准备

开发者环境要求

- 开发工具 XCode 11 及以上：App Store 或单击 下载地址。
- 建议运行环境：
- 设备要求：iPhone 5 及以上；iPhone 6 及以下前置摄像头最多支持到 720p，不支持 1080p。
- 系统要求：iOS 10.0 及以上。

导入 SDK

您可以选择使用 CocoaPods 方案，或者先将 SDK 下载到本地，再将其手动导入到您当前的项目中。

使用 CocoaPods

1. **安装 CocoaPods**
   在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：

```
sudo gem install cocoapods
```

2. **创建 Podfile 文件**
   进入项目所在路径，输入以下命令行之后项目路径下会出现一个 Podfile 文件。

```
pod init
```

3. **编辑 Podfile 文件**
   根据您的项目需要选择合适的版本，并编辑 Podfile 文件：
   - **XMagic 普通版**
     请按如下方式编辑 Podfile文件：

```
platform :ios, '8.0'

target 'App' do
pod 'XMagic'
end
```

   - **XMagic 精简版**
     安装包体积比普通版小，但仅支持基础版 A1-00、基础版 A1-01、高级版 S1-00，请按如下方式编辑 Podfile 文件：

```
platform :ios, '8.0'

target 'App' do
pod 'XMagic_Smart'
end
```

4. **更新并安装 SDK**
   在终端窗口中输入如下命令以更新本地库文件，并安装 SDK：

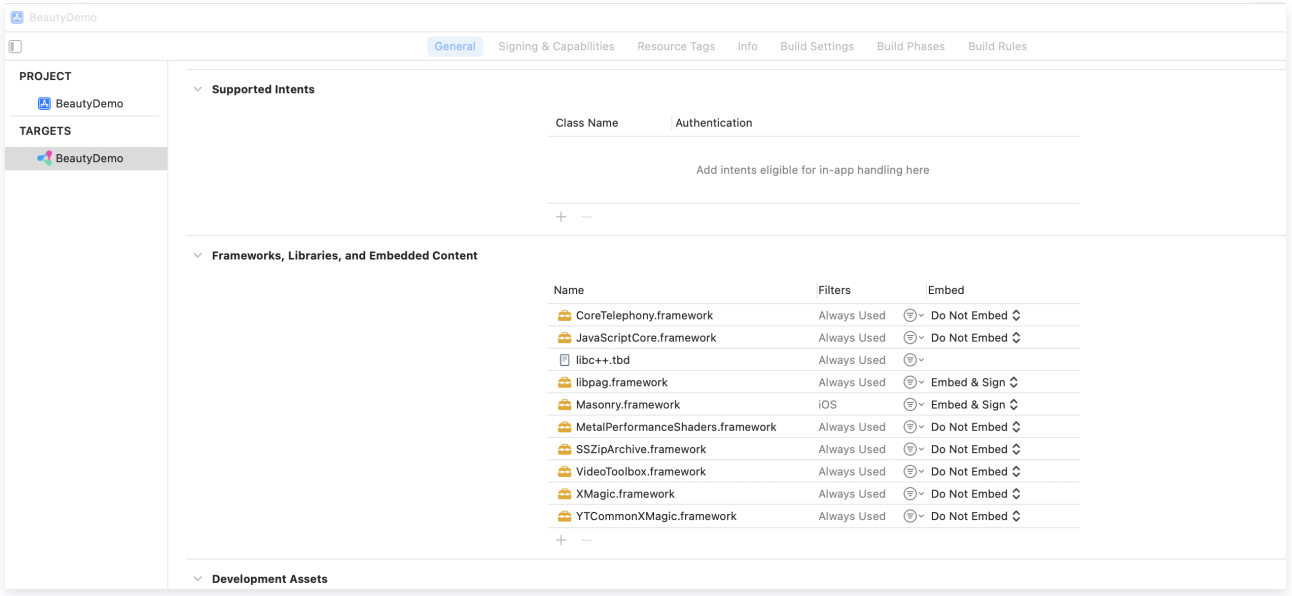
```
pod install
```

   pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。

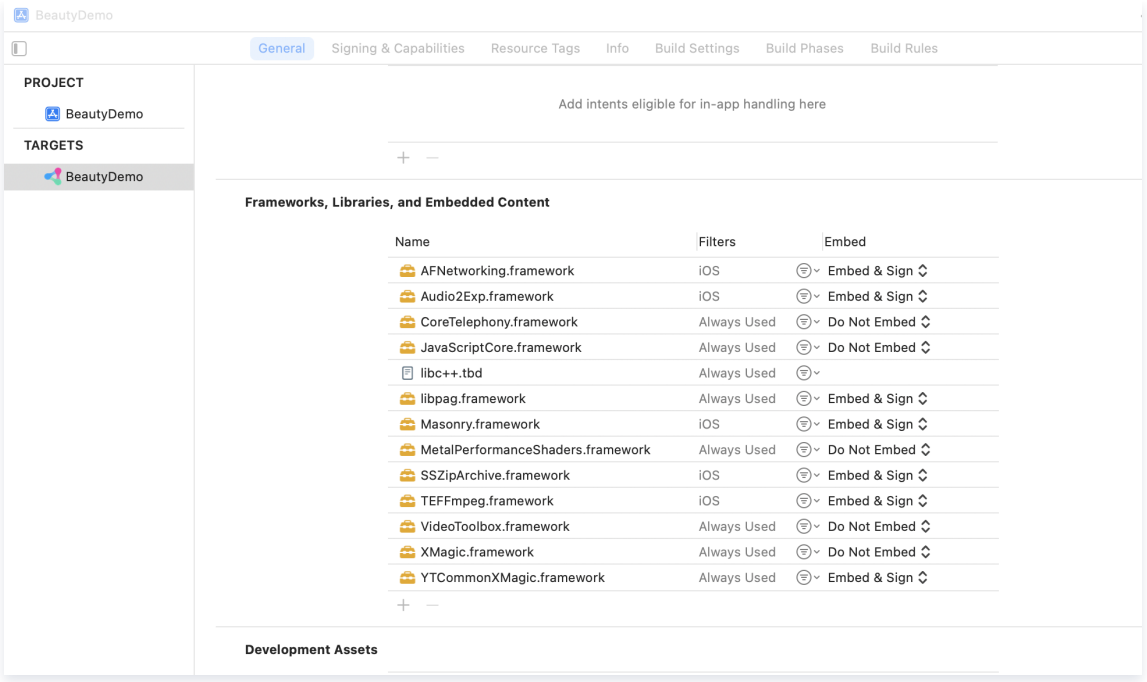
5. **添加资源到实际项目工程中**
   下载并解压 SDK，将 **resources** 文件夹下的除 **LightCore.bundle**、**Light3DPlugin.bundle**、**LightBodyPlugin.bundle**、**LightHandPlugin.bundle**、**LightSegmentPlugin.bundle**、**audio2exp.bundle** 以外的**其它 bundle 资源**添加到实际工程中。
   在 Build Settings 中的Other Linker Flags添加 `-ObjC` 。
6. 将 Bundle Identifier 修改成与申请的测试授权一致。

下载 SDK 并手动导入

1. 下载并解压 SDK，frameworks 文件夹里面是 sdk、resources 文件夹里面是美颜的 bundle 资源。
   - **SDK 版本在2.5.1以前：**
     打开您的 Xcode 工程项目，把 frameworks 文件夹里面的 framework 添加到实际工程中，选择要运行的 target , 选中 **General** 项，单击 **Frameworks,Libraries,and Embedded Content** 项展开，单击底下的“+”号图标去添加依赖库。依次添加下载的 `XMagic.framework` 、 `YTCommonXMagic.framework` 、 `libpag.framework` 及其所需依赖库 `MetalPerformanceShaders.framework` 、 `CoreTelephony.framework` 、 `JavaScriptCore.framework` 、 `VideoToolbox.framework` 、 `libc++.tbd` ，根据需要添加其它工具库 `Masonry.framework` （控件布局库）、 `SSZipArchive` （文件解压库）。



   - **SDK 版本在2.5.1及以后：**
     打开您的 Xcode 工程项目，把 frameworks 文件夹里面的 framework 添加到实际工程中，选择要运行的 target , 选中 **General** 项，单击 **Frameworks,Libraries,and Embedded Content** 项展开，单击底下的“+”号图标去添加依赖库。依次添加下载的 `XMagic.framework` 、 `YTCommonXMagic.framework` 、 `libpag.framework` 、 `Audio2Exp.framework` 、 `TEFFmpeg.framework` 及其所需依赖库 `MetalPerformanceShaders.framework` 、 `CoreTelephony.framework` 、 `JavaScriptCore.framework` 、 `VideoToolbox.framework` 、 `libc++.tbd` ，根据需要添加其它工具库 `Masonry.framework` （控件布局库）、 `SSZipArchive` （文件解压库）。



2. 把 resources 夹里面的美颜资源添加到实际工程中。
3. 在 Build Settings 中的Other Linker Flags添加 `-ObjC` 。
4. 将 Bundle Identifier 修改成与申请的测试授权一致。

动态下载集成

为了减少包大小，您可以将 SDK 所需的模型资源和动效资源 MotionRes（部分基础版 SDK 无动效资源）改为联网下载。在下载成功后，将上述文件的路径设置给 SDK。
我们建议您复用 Demo 的下载逻辑，当然，也可以使用您已有的下载服务。动态下载的详细指引，请参见 SDK 包体瘦身（iOS）。

配置权限

在 Info.plist 文件中添加相应权限的说明，否则程序在 iOS 10 系统上会出现崩溃。请在 Privacy - Camera Usage Description 中开启相机权限，允许 App 使用相机。

集成步骤

步骤一：鉴权

1. 申请授权，得到 LicenseURL 和 LicenseKEY。

> **注意**
> 正常情况下，只要 App 成功联网一次，就能完成鉴权流程，因此您**不需要**把 License 文件放到工程的工程目录里。但是如果您的 App 在从未联网的情况下也需要使用 SDK 相关功能，那么您可以把 License 文件下载下来放到工程目录，作为保底方案，此时 License 文件名必须是 `v_cube.license` 。

2. 在相关业务模块的初始化代码中设置 URL 和 KEY，触发 license 下载，避免在使用前才临时去下载。也可以在 AppDelegate 的 didFinishLaunchingWithOptions 方法里触发下载。其中，LicenseURL 和 LicenseKey 是控制台绑定 License 时生成的授权信息。

SDK版本在2.5.1以前，`TELicenseCheck.h` 在 `XMagic.framework` 里面；SDK版本在2.5.1及以后，`TELicenseCheck.h` 在 `YTCommonXMagic.framework` 里面。

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authresult, NSString *
_Nonnull errorMsg) {
 if (authresult == TELicenseCheckOk) {
     NSLog(@"鉴权成功");
```

```
    } else {
        NSLog(@"鉴权失败");
    }
}];
```

鉴权 errorCode 说明：

错误码	说明
0	成功。Success
-1	输入参数无效，例如 URL 或 KEY 为空
-3	下载环节失败，请检查网络设置
-4	从本地读取的 TE 授权信息为空，可能是 IO 失败引起
-5	读取 VCUBE TEMP License文件内容为空，可能是 IO 失败引起
-6	v_cube.license 文件 JSON 字段不对。请联系腾讯云团队处理
-7	签名校验失败。请联系腾讯云团队处理
-8	解密失败。请联系腾讯云团队处理
-9	TELicense 字段里的 JSON 字段不对。请联系腾讯云团队处理
-10	从网络解析的TE授权信息为空。请联系腾讯云团队处理
-11	把 TE 授权信息写到本地文件时失败，可能是IO失败引起
-12	下载失败，解析本地 asset 也失败
-13	鉴权失败
其他	请联系腾讯云团队处理

步骤二：加载 SDK（XMagic.framework）

使用虚拟形象 SDK 生命周期大致如下：

1. **加载相关资源。**

```
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
 @"root_path":[[NSBundle mainBundle] bundlePath]
};
```

2. **初始化虚拟形象 SDK。**

```
initWithRenderSize:assetsDict: (XMagic)
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDict];
```

3. **虚拟形象 SDK 处理每帧数据并返回相应处理结果。**

```
process: (XMagic)
```

```
// 在摄像头回调传入帧数据
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:
(CMSampleBufferRef)sampleBuffer fromConnection:(AVCaptureConnection *)connection;

// 获取原始数据，处理每帧的渲染信息
- (void)mycaptureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:
(CMSampleBufferRef)inputSampleBuffer fromConnection:(AVCaptureConnection *)connection
```

```
originImageProcess:(BOOL)originImageProcess;

// 使用CPU处理数据
- (YTProcessOutput*)processDataWithCpuFuc:(CMSampleBufferRef)inputSampleBuffer;

// 使用GPU处理数据
- (YTProcessOutput*)processDataWithGpuFuc:(CMSampleBufferRef)inputSampleBuffer;

// 腾讯特效 SDK处理数据接口
/// @param input 输入处理数据信息
/// @return 输出处理后的数据信息
- (YTProcessOutput* _Nonnull)process:(YTProcessInput * _Nonnull)input;
```

4. 释放虚拟形象 SDK。

```
deinit {XMagic}
// 在需要释放SDK资源的地方调用
[self.beautyKit deinit]
```

> **说明**
> 完成上述步骤后，用户即可根据自己的实际需求控制展示时机以及其他设备相关环境。

常见问题

问题1：编译报错：“unexpected service error: build aborted due to an internal error: unable to write manifest to-xxxx-manifest.xcbuild': mkdir(/data, S_IRWXU | S_IRWXG | S_IRWXO): Read-only file system (30):”？

1. 前往 File > Project settings > Build System 选择 Legacy Build System。
2. Xcode 13.0++ 需要在 File > Workspace Settings 勾选 Do not show a diagnostic issue about build system deprecation。

问题2：iOS 导入资源运行后报错：“Xcode 12.X 版本编译提示 Building for iOS Simulator, but the linked and embedded framework '.framework'...”？

将 Build Settings > Build Options > Validate Workspace 改为 Yes，再单击运行。

> **说明**
> Validate Workspace 改为 Yes 之后编译完成，再改回 No，也可以正常运行，所这里有这个问题注意下即可。

问题3：滤镜设置没反应?

检查下设置的值是否正确，范围为 0-100，可能值太小了效果不明显。

问题4：iOS Demo 编译，生成 dSYM 时报错?

```
PhaseScriptExecution CMake\ PostBuild\ Rules build/XMagicDemo.build/Debug-
iphoneos/XMagicDemo.build/Script-81731F743E244CF2B089C1BF.sh
    cd /Users/zhenli/Downloads/xmagic_s106
    /bin/sh -c /Users/zhenli/Downloads/xmagic_s106/build/XMagicDemo.build/Debug-
iphoneos/XMagicDemo.build/Script-81731F743E244CF2B089C1BF.sh

Command /bin/sh failed with exit code 1
```

- 问题原因： `libpag.framework`和`Masonary.framework` 重签名失败。
- 解决方法：

  1.1 打开 `demo/copy_framework.sh` 。

  1.2 用下述命令查看本机 cmake 的路径，将 `$(which cmake)` 改为本地 cmake 绝对路径。

```
which cmake
```

1.3 用自己的签名替换所有 `Apple Development: ......` 。

虚拟形象 SDK 集成指引

iOS
接入虚拟形象 SDK

最近更新时间：2025-06-05 14:46:22

基础集成请参考：快速开始 - iOS，如您已完成 License 校验和 SDK 初始化，请参考如下步骤继续。

步骤1：准备 Avatar 素材

1. 集成虚拟形象 SDK。
2. 在官网下载对应的 Demo 工程，并解压。
3. 将 Demo 中的 `BeautyDemo/bundle/avatarMotionRes.bundle` 素材文件复制到您的工程中。

步骤2：接入 Demo 界面

接入方法

1. 在项目中使用与 BeautyDemo 一样的 Avatar 操作界面。
2. 复制 Demo 中 `BeautyDemo/Avatar` 文件夹下的所有类到您的工程中，添加如下代码即可：

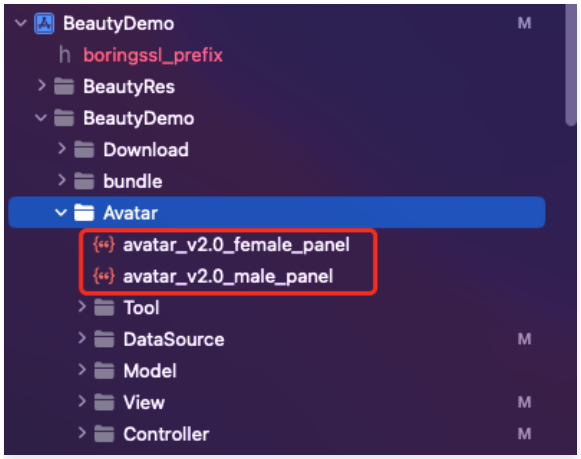
```
AvatarViewController *avatarVC = [[AvatarViewController alloc] init];
avatarVC.modalPresentationStyle = UIModalPresentationFullScreen;
avatarVC.currentDebugProcessType = AvatarPixelData; // 图像或者纹理Id方式
[self presentViewController:avatarVC animated:YES completion:nil];
```

Demo 界面说明

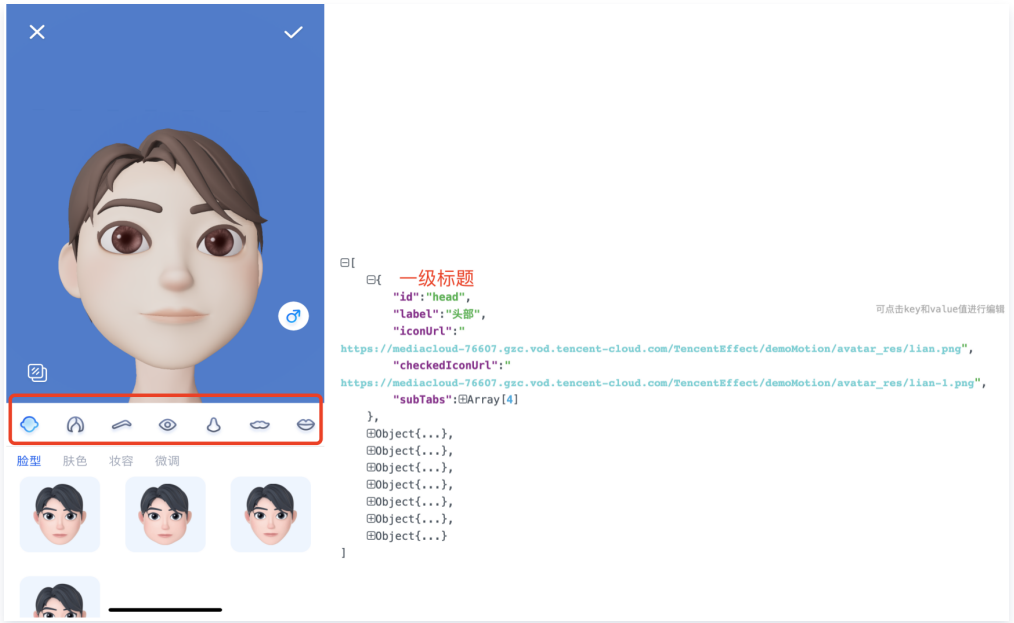
1. Demo UI 界面

2. 实现方案

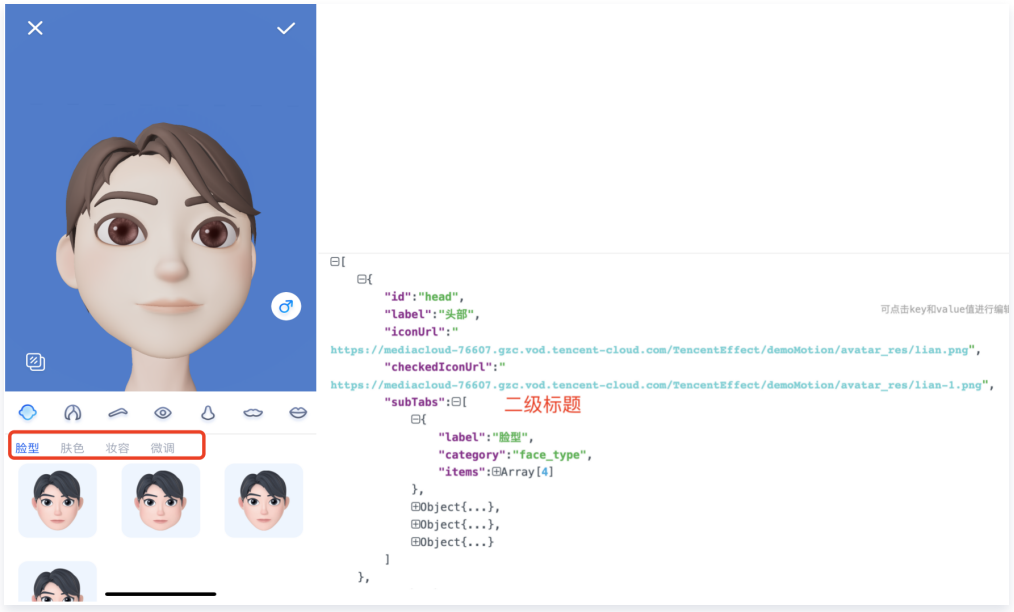
操作面板数据是解析一份 JSON 文件获得的，Demo 中的这份文件放在 `BeautyDemo/Avatar/` 目录。



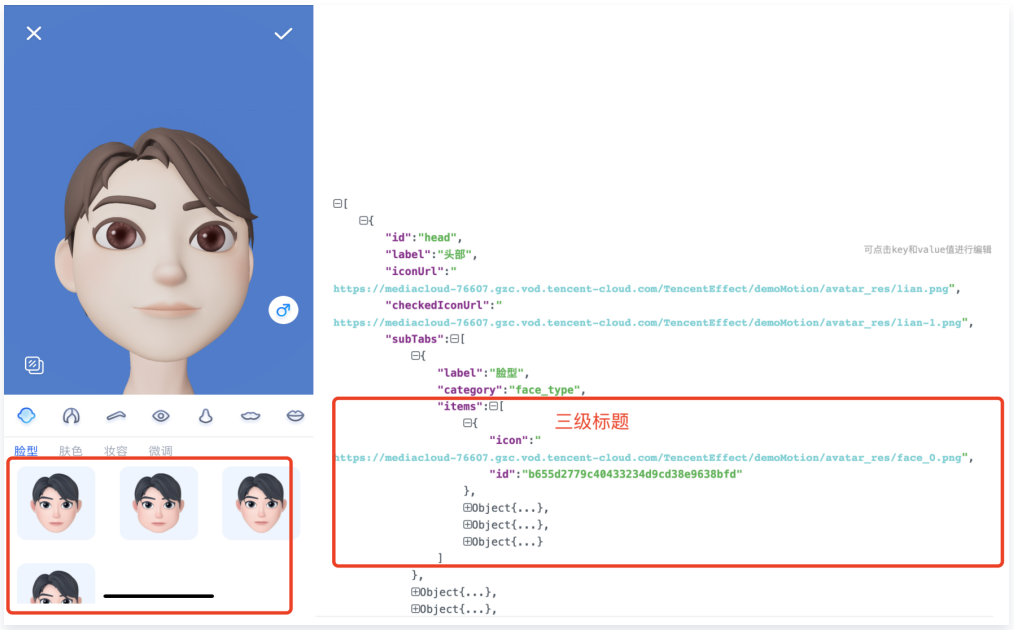
- **JSON 结构与 UI 面板对应的关系**
  - head 为第一个 icon 选中的内容：



  - subTabs 对应右侧二级菜单：

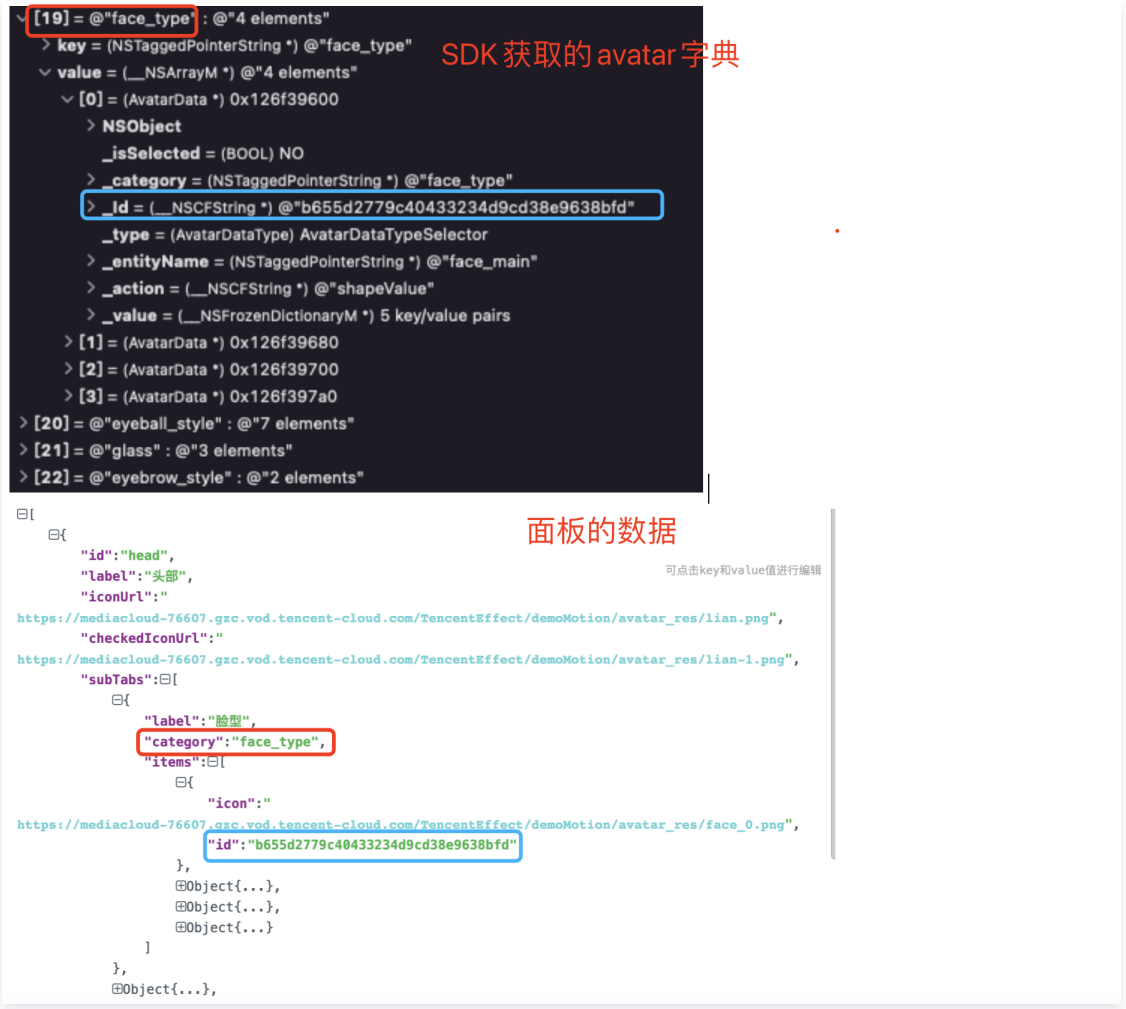


- items 对应右侧三级菜单：



- **面板解析出来的数据关联 SDK 接口获取的 Avatar 对象数据**

下图中上半部分为 SDK 获取的 avatar 字典（key 为 category，value 为 avatar 数组），下半部分为面板的数据。当点击面板的选项时，从面板二级标题获取 category（**红框标记**），通过该 category 可以在 SDK 返回的 avatar 字典内获取到对应的 avatarData 数组。从面板三级标题处获取 ID（**蓝框标记**），通过该 ID 可以在 avatarData 数组匹配到对应的 avatarData 对象，将此对象传入 SDK 的 updateAvatar 接口即可完全捏脸。



3. 修改标题的图标/文字

修改 Demo UI 图片所示 JSON 文件，例如修改头部展示的 icon，则修改下图红框中的 iconUrl 与 checkedIconUrl 即可。

步骤3：自定义捏脸功能

可参考 `BeautyDemo/Avatar/Controller` 中的 AvatarViewController 相关代码。

> **说明**
> 接口说明请参见 Avatar SDK 说明 。

1. 创建 xmagic 对象，设置 Avatar 默认模板。

```
- (void)buildBeautySDK {

    CGSize previewSize = CGSizeMake(kPreviewWidth, kPreviewHeight);
    NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) lastObject];
    beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_config.json"];
    NSFileManager *localFileManager=[[NSFileManager alloc] init];
    BOOL isDir = YES;
    NSDictionary * beautyConfigJson = @{};
    if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !isDir) {
            NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfigPath
encoding:NSUTF8StringEncoding error:nil];
            NSError *jsonError;
            NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8StringEncoding];
            beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
options:NSJSONReadingMutableContainers

error:&jsonError];
    }
    NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                                                        @"root_path":[[NSBundle mainBundle]
bundlePath],
                                                        @"tnn_"
                                                        @"beauty_config":beautyConfigJson
    };
    // Init beauty kit
    self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDict];
    // Register log
    [self.beautyKit registerSDKEventListener:self];
    [self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL];

    // 传入素材文件对应的路径即可加载avatar默认形象
```

```
    AvatarGender gender = self.genderBtn.isSelected ? AvatarGenderFemale : AvatarGenderMale;
    NSString *bundlePath = [self.resManager avatarResPath:gender];
    [self.beautyKit loadAvatar:bundlePath exportedAvatar:nil];

}
```

2. 获取素材的 Avatar 源数据。

```
@implementation AvatarViewController
_resManager = [[AvatarResManager alloc] init];
NSDictionary *avatarDict = self.resManager.getMaleAvatarData;
@end


@implementation AvatarResManager

- (NSDictionary *)getMaleAvatarData
{
    if (!_maleAvatarDict) {
            NSString *resDir = [self avatarResPath:AvatarGenderFemale];
            NSString *savedConfig = [self getSavedAvatarConfigs:AvatarGenderMale];
            // 通过sdk接口解析出素材源数据
            _maleAvatarDict = [XMagic getAvatarConfig:resDir exportedAvatar:savedConfig];
    }
    return _maleAvatarDict;
}
@end
```

3. 捏脸操作。

```
// 从sdk接口解析出来的素材源数据中拿到想要形象的avatar对象，传入sdk
NSMutableArray *avatars = [NSMutableArray array];
// avatarConfig是从sdk接口getAvatarConfig:exportedAvatar:获取的其中一个avatar对象
[avatars addObject:avatarConfig];
// 捏脸/换装接口，调用后实时更新当前素材呈现出的形象
[self.beautyKit updateAvatar:avatars];
```

4. 导出捏脸字符串：
将当前 Avatar 配置的对象导出为字符串，可自定义存储。

```
- (BOOL)saveSelectedAvatarConfigs:(AvatarGender)gender
{
    NSMutableArray *avatarArr = [NSMutableArray array];
    NSDictionary *avatarDict = gender == AvatarGenderMale ? _maleAvatarDict : _femaleAvatarDict;
    // 1、遍历找出选中的avatar对象
    for (NSArray *arr in avatarDict.allValues) {
            for (AvatarData *config in arr) {
                    if (config.type == AvatarDataTypeSelector) {
                            if (config.isSelected) {
                                    [avatarArr addObject:config];
                            }
                    } else {
                            [avatarArr addObject:config];
                    }
            }
    }
    // 2、调用sdk接口将选中的avatar对象导出为字符串
    NSString *savedConfig = [XMagic exportAvatar:avatarArr.copy];
    if (savedConfig.length <= 0) {
```

```
            return NO;
    }
    NSError *error;
    NSString *fileName = [self getSaveNameWithGender:gender];
    NSString *savePath = [_saveDir stringByAppendingPathComponent:fileName];
    // 判断目录是否存在，不存在则创建目录
    BOOL isDir;
    if (![[NSFileManager defaultManager] fileExistsAtPath:_saveDir isDirectory:&isDir]) {
            [[NSFileManager defaultManager] createDirectoryAtPath:_saveDir
withIntermediateDirectories:YES attributes:nil error:nil];
    }
    // 3、将导出的字符串写入沙盒，下次取出来可用
    [savedConfig writeToFile:savePath atomically:YES encoding:NSUTF8StringEncoding error:&error];
    if (error) {
            return NO;
    }
    return YES;
}
```

5. 切换虚拟与真实背景。

```
- (void)bgExchangeClick:(UIButton *)btn
{
    btn.selected = !btn.isSelected;
    NSDictionary *avatarDict = self.resManager.getFemaleAvatarData;
    NSArray *array = avatarDict[@"background_plane"];
    AvatarData *selConfig;
    // 背景实际上也是一个avatar对象，category为background_plane，替换背景就是设置不同的avatar对象
    for (AvatarData *config in array) {
            if ([config.Id isEqual:@"none"]) {
                    if (btn.selected) {
                            selConfig = config;
                            break;
                    }
            } else {
                    selConfig = config;
            }
    }
    [self.beautyKit updateAvatar:@[selConfig]];
}
```

虚拟形象 SDK 说明

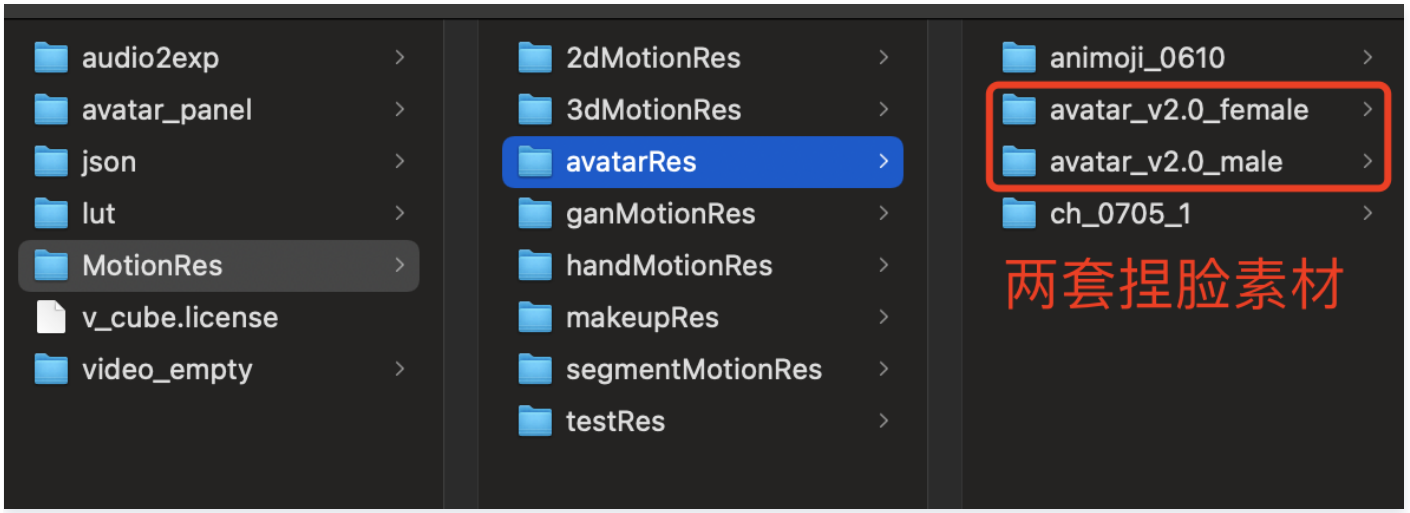
最近更新时间：2023-04-25 18:03:12

SDK 接入

SDK 的下载、接入、鉴权、跑通 Demo 请参见 独立集成腾讯特效 。

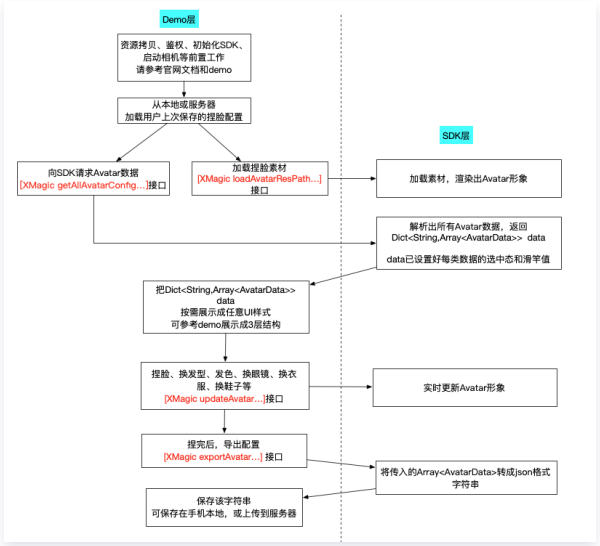
准备捏脸素材

目前我们随 SDK 提供了若干套捏脸、换装素材，素材在 SDK 解压后的 `MotionRes/avatarRes` 目录中，与其他的动效素材一样，您需要把它 copy 到工程的 assets 目录：

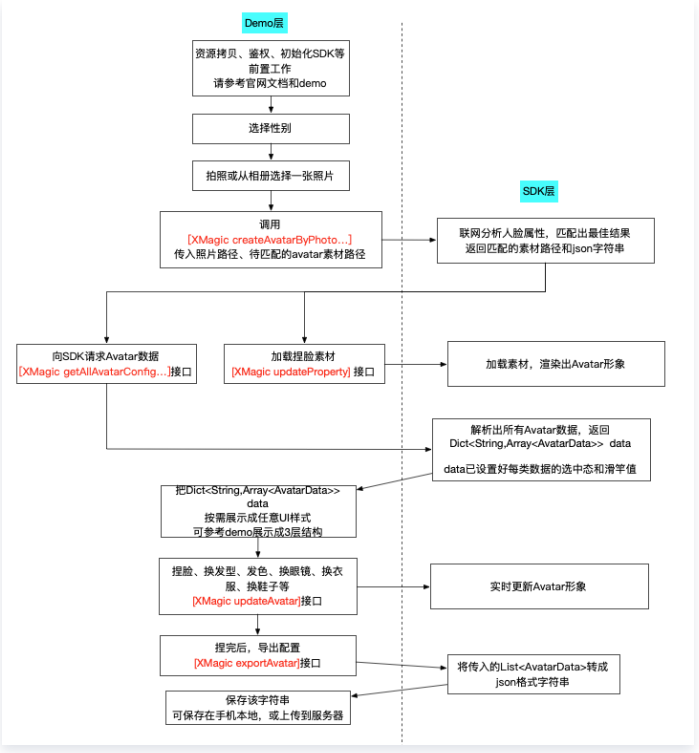


捏脸流程与 SDK 接口

捏脸流程



拍照捏脸流程



XMagicApi 的加载数据、捏脸、导出配置、拍照捏脸接口详情如下：

1. 获取 Avatar 源数据接口（getAvatarConfig）

```
+ (NSDictionary <NSString *, NSArray *>* _Nullable)getAvatarConfig:(NSString * _Nullable)resPath
exportedAvatar:(NSString *_Nullable)exportedAvatar;
```

- **输入参数：**
  - resPath：Avatar 素材在手机上的绝对路径，例如：
    ```
    /var/mobile/Containers/Data/Application/C82F1F7A-01A1-4404-8CF6-
    131B26B4DA1A/Library/Caches/avatarMotionRes.bundle/avatar_v2.0_male
    ```
  - exportedAvatar：用户上次捏脸之后保存的数据，是 JSON 格式的字符串。首次使用或用户之前没有保存过的话，这个值为 nil
- **输出参数：**
  以 NSDictionary 的形式返回，dictionary 的 key 是数据的 category，详见 TEDefine 类，dictionary 的 value 是这个 category 下全部数据。应用层拿到这份 dictionary 后，按需展示成自己想要的 UI 样式

2. 加载 Avatar 素材接口（loadAvatar）

```
- (void)loadAvatar:(NSString * _Nullable)resPath exportedAvatar:(NSString * _Nullable)exportedAvatar;
```

- **输入参数：**
  - resPath：avatar素材在手机上的绝对路径，例如：
    ```
    /var/mobile/Containers/Data/Application/C82F1F7A-01A1-4404-8CF6-
    131B26B4DA1A/Library/Caches/avatarMotionRes.bundle/avatar_v2.0_male
    ```
  - exportedAvatar：用户上次捏脸之后保存的数据，是json格式的字符串。首次使用或用户之前没有保存过的话，这个值为 nil
- **输出参数：**
  以 NSDictionary 的形式返回，dictionary 的 key 是数据的 category，详见 TEDefine 类，dictionary 的 value 是这个 category 下的全部数据。应用层拿到这份 dictionary 后，按需展示成自己想要的 UI 样式

3. 捏脸、换装接口（updateAvatar）

```
- (void)updateAvatar:(NSArray<AvatarData *> *_Nonnull)avatarDataList;
```

调用后实时更新当前素材的预览形象，一个 AvatarData 对象是一个原子配置（如换发型），一次可以传入多个原子配置（比如既换发型，又换发色）。该接口会检查传入的 AvatarData 的有效性，有效的设置给SDK，无效的数据会 callback 回去。

- 比如要求修改发型，但是头发模型文件（配置在 AvatarData 的 value 字段里）在本地找不到，就认为该 AvatarData 无效。
- 再比如要求修改瞳孔贴图，但是贴图文件（配置在 AvatarData 的 value 字段里）在本地找不到，就认为该 AvatarData 无效。

4. 导出捏脸配置接口（exportAvatar）

```
+ (NSString *_Nullable)exportAvatar:(NSArray <AvatarData *>*_Nullable)avatarDataList;
```

用户捏脸时，会修改 AvatarData 中的 selected 状态或形变值。捏完后，传入新的全量 AvatarData 列表，即可导出一份 JSON 字符串。这份字符串您可以存在本地，也可以上传到服务器。
导出的这份字符串有两个用途：

- 当下次再通过 XMagic 的 **loadAvatar** 接口加载这份 Avatar 素材时，把这份 JSON 字符串设置给 **exportedAvatar**，这样才能在预览中呈现出用户上次捏脸的形象。
- 如上文所述，调用 getAllAvatarData 时需要传入这个参数，以便修正 Avatar 源数据中的选中态和形变值。

5. 拍照捏脸接口（createAvatarByPhoto）

该接口需要联网。

```
+ (void)createAvatarByPhoto:(NSString * _Nullable)photoPath avatarResPaths:(NSArray <NSString *>*
_Nullable)avatarResPaths isMale:(BOOL)isMale success:(nullable void (^)(NSString *_Nullable
matchedResPath, NSString *_Nullable srcData))success failure:(nullable void (^)(NSInteger code, NSString
*_Nullable msg))failure;
```

- **photoPath**：照片路径，请确保人脸位于画面中间。建议画面中只包含一个人脸，如果有多个人脸，SDK 会随机选择一个。建议照片的短边大于等于500px，否则可能影响识别效果。
- **avatarResPaths**：您可以传入多套 Avatar 素材，SDK 会根据照片分析的结果，选择一套最合适的素材进行自动捏脸。

> ⚠ 注意
> 目前只支持一套，如果传入多套，SDK 只会使用第一套。

- **isMale**：是否是男性。暂未用到该属性，但建议准确传入，SDK 后续会优化。
- **success**：成功回调。 matchedResPath—匹配的素材路径、srcData—匹配结果，与上文中的 exportAvatar 接口返回的是一样的含义。
- **failure**：失败回调。code—错误码，msg—错误信息。

6. 将下载好的配置文件放置到对应的文件夹中（addAvatarResource）

```
+ (void)addAvatarResource:(NSString * _Nullable)rootPath category:(NSString * _Nullable)category filePath:
(NSString * _Nullable)filePath completion:(nullable void (^)(NSError * _Nullable error, NSArray
<AvatarData *>* _Nullable avatarList))completion;
```

该接口主要用于动态下载 Avatar 配件的场景。举个例子，您的 Avatar 素材中有10种发型，后来想动态下发一种发型给客户端，下载完成后，得到一个压缩包，然后调用该接口，把压缩包路径传给 SDK，SDK 会解析这份压缩包，将它放到对应的 category 目录下。下次您在调用 getAllAvatarData 接口时，SDK就能解析出新添加的这份数据。
参数说明：

- **rootPath**：Avatar 素材的根目录，例如
  `/var/mobile/Containers/Data/Application/C82F1F7A-01A1-4404-8CF6-131B26B4DA1A/Library/Caches/avatarMotionRes.bundle/avatar_v2.0_male`
- **category**：下载的这份配置的分类
- **zipFilePath**：下载下来的 ZIP 包存放的本地地址
- **completion**：结果回调。error — 错误信息。avatarList — 解析的 avatar 数据数组

7. 发送自定义事件

发送自定义事件，如：开启无人脸时的闲置显示状态。

```
- (void)sendCustomEvent:(NSString * _Nullable)eventKey eventValue:(NSString * _Nullable)eventValue;
```

- **eventKey**：自定义事件 key，可参考 TEDefine 的 AvatarCustomEventKey。
- **eventValue**：自定义事件 value，为 JSON 字符串，例如 `@{@"enabel" : @(YES)}`转json字符串，或者直接写`@"{\\"enable\\" : true}"`。

8. 调用 AvatarData

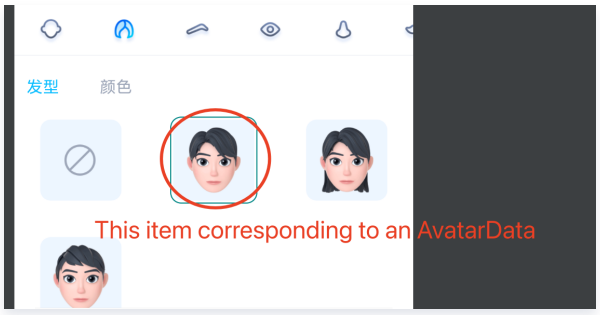
这几个接口的核心都是 AvatarData 类，其主要内容如下：

```
/// @brief 捏脸配置类型
@interface AvatarData : NSObject
/// 例如 脸型、眼睛微调 等。TEDefine中定义了标准的category，如果不满足需求，也可以自定义category字符串，跟已有的不冲突即
可，不能为空
@property (nonatomic, copy) NSString * _Nonnull category;
// 标识每一个具体item 或者 每一个微调项。比如每个眼镜都有自己的id。每一个微调项也有自己的id。不能为空
@property (nonatomic, copy) NSString * _Nonnull Id;
// selector选择类型或者AvatarDataTypeSlider值类型
@property (nonatomic, assign) AvatarDataType type;
/// 如果是selector类型，则它表示当前有无被选中
@property (nonatomic, assign) BOOL isSelected;

/// 捏身体某个部位 如：脸、眼睛、头发、上衣、鞋子等。如何设值参考官方文档
@property (nonatomic, copy) NSString * _Nonnull entityName;
/// 表示对entityName执行什么操作(action)。 规范参考官方文档
@property (nonatomic, copy) NSString * _Nonnull action;
```

```
/// 表示对entityName执行action操作的具体值。 规范参考官方文档
@property (nonatomic, copy) NSDictionary * _Nonnull value;
@end
```

一个 AvatarData 对象是一个原子配置，如换发型、调整脸颊等：

换发型	调整脸颊
	

- 捏脸时，如果数据是 selector类型，则修改 AvatarData 的 selected 字段。例如有4种眼镜 A、B、C、D，默认选中的是 A，那么 A 的 selected 是 true，B、C、D 为 false。如果用户选择了眼镜 B，则把 B 的 selected 为 true，A、C、D 为 false。
- 捏脸时，如果数据是 slider 类型，则修改 AvatarData 的 value 字段。value 字段是一个 JsonObject，里面是若干对 key-value，把 key-value 中的 value 修改为滑竿的值即可。

AvatarData 高级说明

AvatarData 是 SDK 自动从素材根目录的 custom_configs 目录中解析出来返回给应用层的。通常您不需要手动构造 AvataData。
AvatarData 中，对捏脸起关键作用的是 entityName 、action ， value 三个字段。这3个字段的值是 SDK 在解析素材配置时自动填入的。大多数情况下，您不需要了解这3个字段的含义，仅在 UI 层展示时，如果是滑竿类型，则需要解析 value 中的形变 key-value 与 UI 操作进行对应。

entityName 字段

捏脸时，需要明确指定捏哪个部位，比如脸、眼睛、头发、上衣、鞋子 等。entityName 字段就是描述这些身体部位名称的。

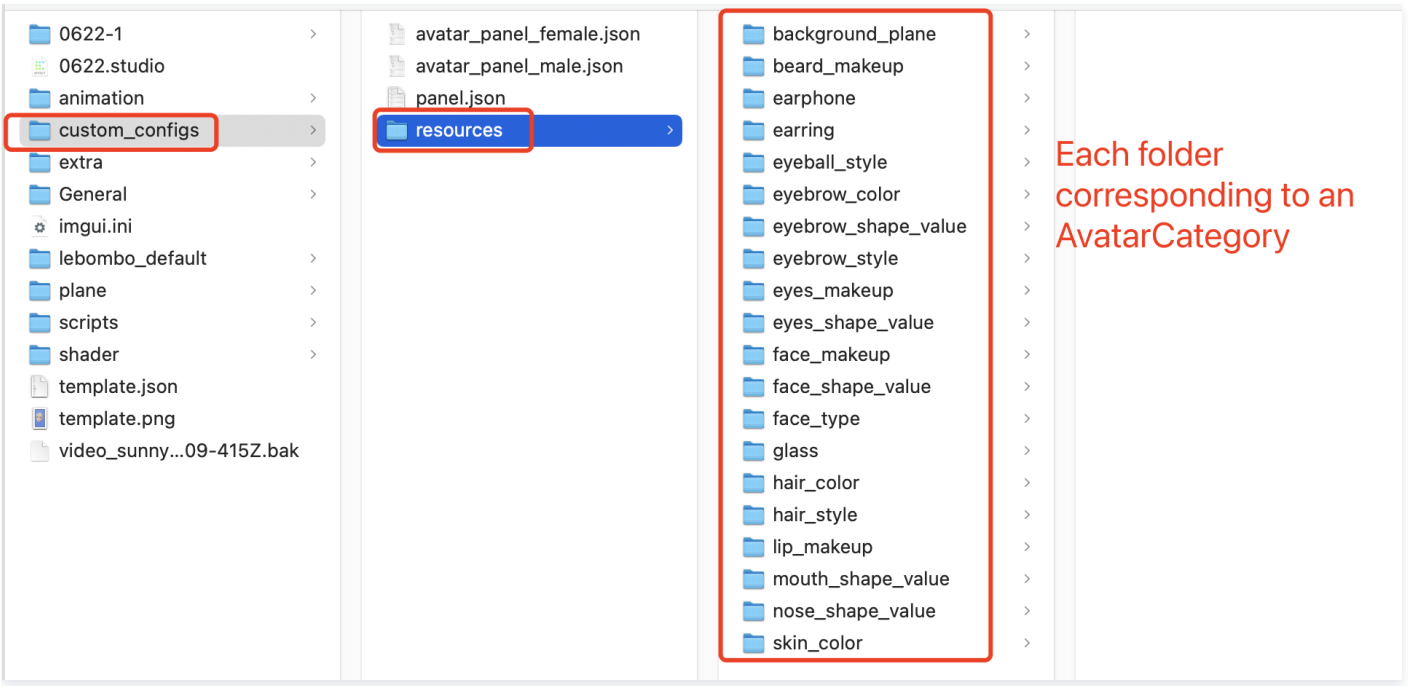
action 和 value字段

action 字段表示对 entityName 执行什么操作（action）。SDK 内定义了五种 action，每种 action 的含义及 value 要求如下：

action	含义	value 要求
changeColor	修改当前材质的颜色，包括基础色、自发光色等颜色属性	JsonObject 类型，必填。由素材制作工具自动生成。
changeTexture	修改当前材质的贴图，包括颜色纹理贴图、金属粗糙度纹理贴图、AO 纹理贴图、法线纹理贴图、自发光纹理贴图等	JsonObject 类型。必填。由素材制作工具自动生成。
shapeValue	修改blendShape形变值，一般用于面部细节形变微调	JsonObject 类型。里面的 key 是形变名称，value 是 float 类型的值。必填。由素材制作工具自动生成。
replace	替换子模型，例如替换眼镜、发型、衣服等	JsonObject 类型。里面描述了新的子模型的3D变换信息、模型路径、材质路径。如果要隐藏当前位置的子模型，则使用null。由素材制作工具自动生成。
basicTransform	调整位置、旋转、缩放。一般用于调整摄像机的远近、角度，从而实现模型全身和半身视角的切换	JsonObject 类型。必填。由素材制作工具自动生成。

配置 Avatar 捏脸换装数据

Avatar 属性配置存放在 resources 文件夹下（路径为：`素材/custom_configs/resources`）：

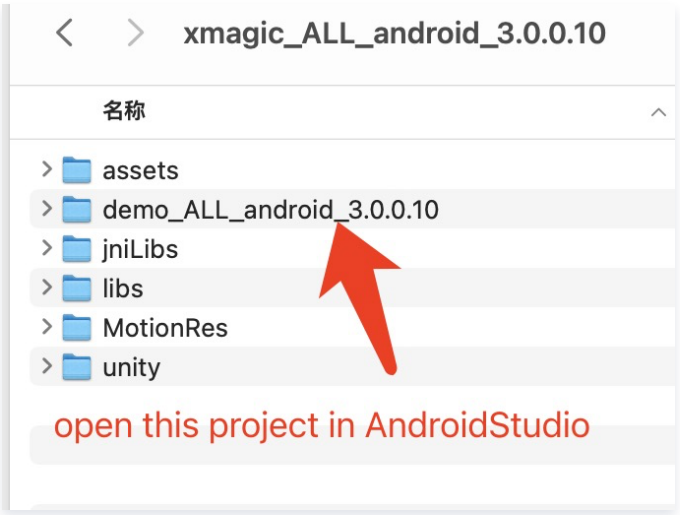


这些配置文件是自动生成的，通常不需要手动配置。自动生成的方式如下：
设计师用 TencentEffectStudio 设计好一套形象后，运行我们提供的 resource_generator_gui 这个 app（目前仅支持 MacOS 平台），即可自动生成这些配置，详情请参见 设计规范 。

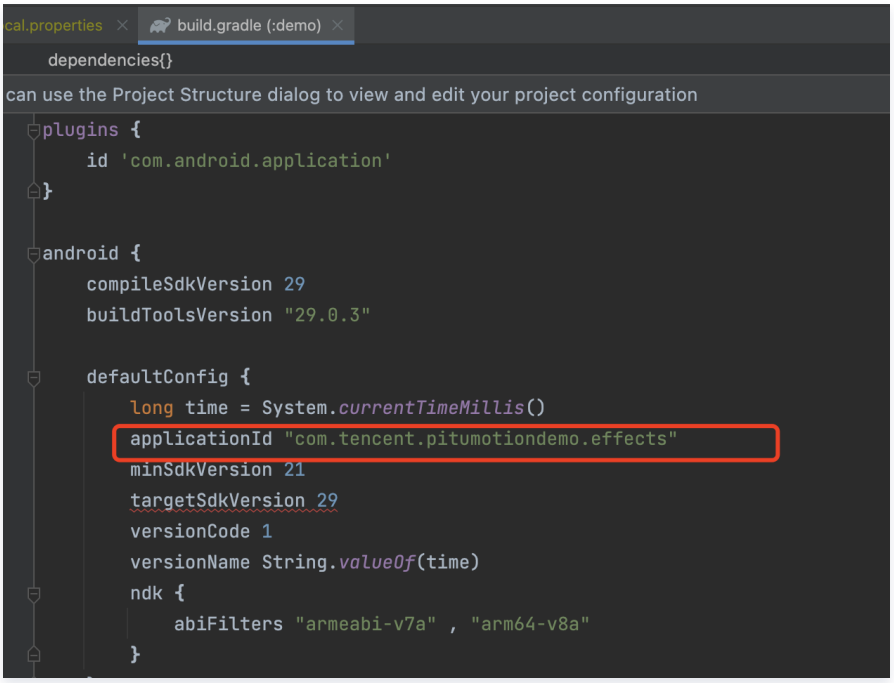
Android
快速跑通 Demo

最近更新时间：2023-05-18 15:00:31

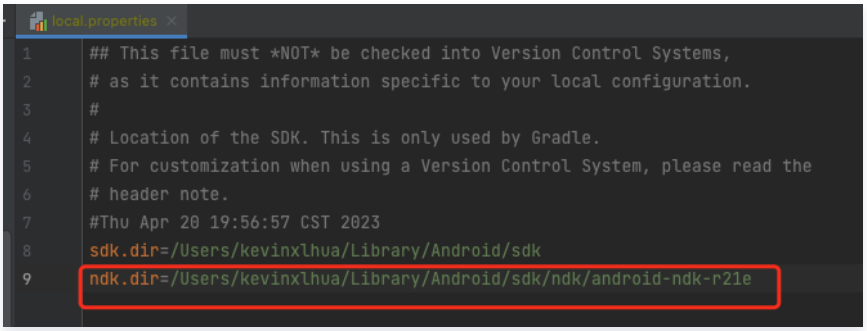
1. 使用 Android Studio 打开项目工程。



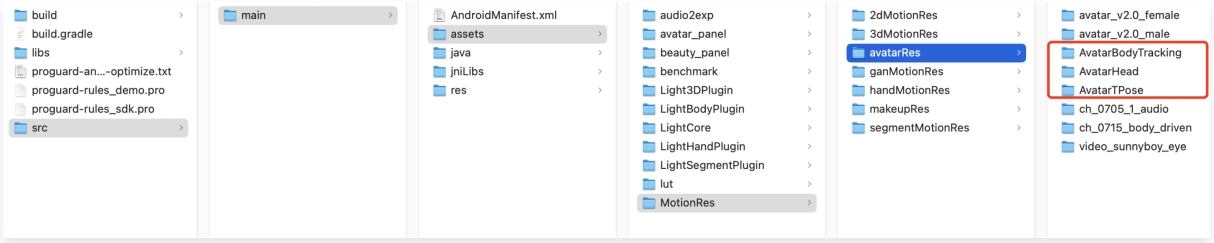
2. 在工程的 `com.tencent.demo.TEBeautyImpl.java` 类中设置自己申请的 `License` 信息。
3. 在工程下的 build.gradle 文件中修改包名为自己的包名。



4. 如果提示配置 NDK，则需要在工程的 local.properties 文件中配置 NDK 路径。



5. 替换授权的素材：demo 工程中的部分 Avatar 素材（如下图红框圈中的3个素材）需要签发授权才能使用，请 联系我们 获取授权素材。



6. 完成以上步骤，即可运行 demo。

接入虚拟形象 SDK

最近更新时间：2023-05-25 19:20:02

基础集成请参考：快速开始 - Android，如您已完成 License 校验和 SDK 初始化，请参考如下步骤继续。

步骤1：准备 Avatar 素材

1. 在官网下载对应的 demo 工程，并解压。
2. 添加 avatar 资源：联系我们进行 Avatar 资源签发，然后将签发的资源复制到工程中（和 Demo 位置保持一致 `demo/app/assets/MotionRes/avatarRes` 下）。由于 Demo 中的 `demo/app/assets/MotionRes/avatarRes` 下的 avatar 资源是加密授权过的，所以客户无法直接使用。
3. 复制 Demo 中 `com.tencent.demo.avatar` 文件夹下的所有类到您的工程中。

步骤2：接入 Demo 界面

参考 Demo 中的 `com.tencent.demo.avatar.AvatarEditActivity` 类，添加如下代码。

1. 在页面的 xml 文件中配置面板信息：

```
<com.tencent.demo.avatar.view.AvatarPanel
        android:id="@+id/avatar_panel"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        app:layout_constraintBottom_toBottomOf="parent" />
```

2. 在页面中获取面板对象并设置对应的数据回调接口：

```
avatarPanel.setAvatarPanelCallBack(new AvatarPanelCallBack() {

            @Override
            public void onReceiverBindData(List<AvatarData> avatarData) {
                mXMagicApi.updateAvatar(avatarData, AvatarEditActivity.this);
            }

            @Override
            public void onItemChecked(MainTab mainTab, AvatarItem avatarItem) {
                if (avatarItem.avatarData == null && URLUtil.isNetworkUrl(avatarItem.downloadUrl)) {
//此处表示要进行动态下载
                    downloadAvatarData(avatarItem, () -> updateConfig(avatarItem));
                } else {
                    updateConfig(avatarItem);
                    List<AvatarData> bindAvatarData =
AvatarResManager.getAvatarDataByBindData(avatarItem.bindData);
                    mXmagicApi.updateAvatar(bindAvatarData, AvatarActivity.this);
                }
            }

            @Override
            public void onItemValueChange(AvatarItem avatarItem) {
                updateConfig(avatarItem);
            }

            @Override
            public boolean onShowPage(AvatarPageInf avatarPageInf, SubTab subTab) {
                if (subTab != null && subTab.items != null && subTab.items.size() > 0) {
                    AvatarItem avatarItem = subTab.items.get(0);
                    if (avatarItem.type == AvatarData.TYPE_SLIDER && avatarItem.avatarData == null &&
URLUtil.isNetworkUrl(avatarItem.downloadUrl)) {  //此处表示要进行动态下载
                        downloadAvatarData(avatarItem, () -> {
                            if (avatarPageInf != null) {
```

```
                                avatarPageInf.refresh();
                            }
                        });
                        return false;
                    }
                }
                return true;
            }

            private void updateConfig(AvatarItem avatarItem) {
                if (mXmagicApi != null && avatarItem != null) {
                    List<AvatarData> avatarConfigList = new ArrayList<>();
                    avatarConfigList.add(avatarItem.avatarData);
                    mXmagicApi.updateAvatar(avatarConfigList, AvatarActivity.this);
                }
            }
        });
```

3. 获取面板数据，并设置给面板：

```
AvatarResManager.getInstance().getAvatarData(avatarResName, getAvatarConfig(), allData -> {
      avatarPanel.initView(allData);
 });
```

4. 创建 xmagicApi 对象，并加载捏脸资源：

```
 protected void initXMagicAndLoadAvatar(String avatarConfig, UpdatePropertyListener
updatePropertyListener) {
     if (mXMagicApi == null && !isFinishing() && !isDestroyed()) {
         WorkThread.getInstance().run(() - > {
             synchronized(lock) {
                 if (isXMagicApiDestroyed) {
                     return;
                 }
                 mXMagicApi = XmagicApiUtils.createXMagicApi(getApplicationContext(), null);
                 AvatarResManager.getInstance().loadAvatarRes(mXMagicApi, avatarResName, avatarConfig
== null ? getAvatarConfig() : avatarConfig, updatePropertyListener);
                 setAvatarPlaneType();
             }
         }, this.hashCode());
     }
 }
```

5. 保存 Avatar 属性，可参考 Demo 中的 saveAvatarConfigs 方法：

```
/**
* 保存用户设置的属性或者默认属性值
*/
    public void onSaveBtnClick() {
    //通过AvatarResManager的getUsedAvatarData获取用户配置的属性
        List < AvatarData > avatarDataList =
AvatarResManager.getUsedAvatarData(avatarPanel.getMainTabList());
       //通过XmagicApi.exportAvatar方法将配置的属性转换为字符串
         String content = XmagicApi.exportAvatar(avatarDataList);
         //保存此字符串即可，下载使用时将此字符串设置给SDK的loadAvatarRes方法，即可恢复此形象
        if (mXMagicApi != null) {
            mXMagicApi.exportCurrentTexture(bitmap - > saveAvatarModes(bitmap, content));
        }
```

```
    }
```

6. 切换背景参考 Demo 中的背景面板：

7. 设置模型动画参考demo 中的互动页面

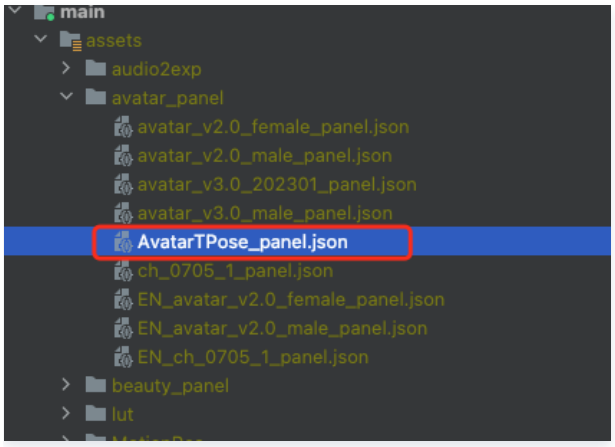
自定义 UI

最近更新时间：2023-05-18 15:00:33

Demo UI 说明

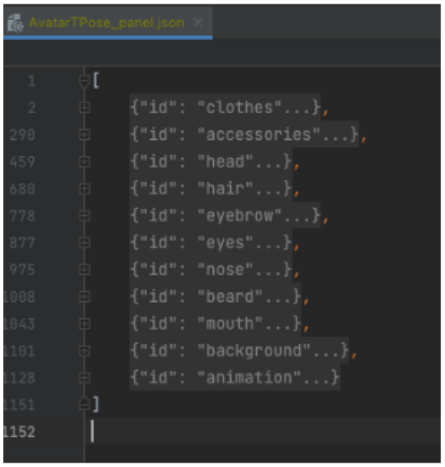
实现方式

面板配置信息可存放在任何路径， Demo 中存放在 assets 中，在 Demo 首次使用面板文件时会复制到安装目录下。



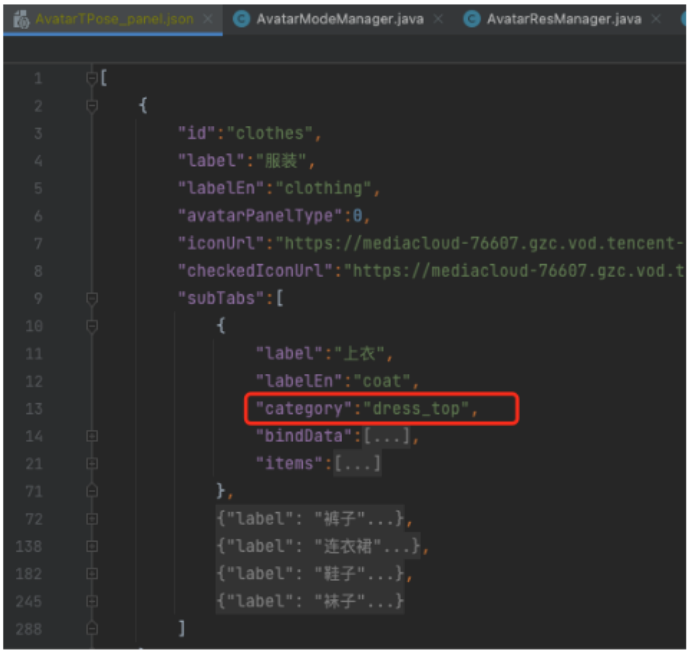
Json 结构和 UI 面板对应关系：

- 左侧 item 对应右侧页面一级菜单，clothes 为第一个 icon 选中的内容：



- 左侧红框 subTabs 对应右侧二级菜单：

- 左侧 icon 对应的 AvatarData 数据存储在素材下的 resources 文件夹中，右侧展示的是面板的配置数据，两者之间是通过面板数据中的 **category 进行关联**，SDK 会解析 resources 文件夹中的数据，放入对应的 map 中，map 的 key 是 category 的值，所以在 Demo 中解析完 `panel.json` 文件后，可通过 SDK 提供的方法获取数据进行关联。可以参考 demo 中的 AvatarResManager 的 getAvatarData 方法，此方法会解析面板文件并和 SDK 返回的属性进行关联。



Demo 重要类说明

路径：`com.tencent.demo.avater.AvatarResManager.java`

1. 加载 Avatar 资源

```
/**
     * 用于加载Avatar 资源
     *
     * @param xmagicApi      XmagicApi对象
     * @param avatarResName  名称
     * @param avatarSaveData 加载模型的默认配置，如果没有则传null
     */
    public void loadAvatarRes(XmagicApi xmagicApi, String avatarResName, String avatarSaveData)
```

2. 获取面板数据

```
 /**
     * 获取avatar面板数据，
     *
     * @param avatarResName      avatar素材名称
     * @param avatarDataCallBack 由于此方法会访问文件，所以会在子线程中进行文件操作，获取到数据后会在主线程回调
     *                           返回的数据是已经包含了resources文件夹下的数据
     */
    public void getAvatarData(String avatarResName, String avatarSaveData, LoadAvatarDataCallBack
avatarDataCallBack)
```

3. 从面板数据中解析出用户设置的属性或默认属性

```
//从面板的配置文件中解析出用户设置的属性或默认属性
public static List<AvatarData> getUsedAvatarData(List<MainTab> mainTabList)
```

4. 从 bindData 中解析出对应的 avatarData

```
    /**
     * 从bindData中解析出对应的avatarData
     *
     * @param bindDataList 绑定管理列表
     * @return 返回对应的avatarData数据列表
     */
    public static List < AvatarData > getAvatarDataByBindData(Map < String, MainTab > mainTabList, List <
BindData > bindDataList, boolean isFromSaveData)
```

附录

MainTab.java、SubTab.java、AvatarItem.java、BindData.java 中的字段介绍。

MainTab

字段	类型	是否必填	含义
id	String	是	主菜单唯一标识，用于区分主菜单，所以需要全局唯一
label	String	否	一级 tab 上展示的中文名称（Demo 中暂时不展示）
labelEn	String	否	一级 tab 上展示的英文名称（Demo 中暂时不展示）
avatarPanelType	int	是	面板类型 0：换装面板

			1：捏脸面板 2：背景面板 3：动作面板
iconUrl	String	是	图片地址，未选中时的图片地址
checkedIconUrl	String	是	图片地址，选中时的图片地址
subTabs	SubTab 类型列表	是	二级菜单列表

SubTab

字段	类型	是否必填	含义
label	String	是	二级菜单中文名称
labelEn	String	是	二级菜单英文名称
category	String	是	二级菜单的分类类型，在 SDK 中 `com.tencent.xmagic.avatar.AvatarCategory` 类中定义
type	int	是	页面展示类型： 0： 表示icon类型，默认值。 1：表示滑竿调节类型
bindData	BindData列表类型	否	被依赖的属性配置字段，此字段可在Subtab节点下，也可配置在AvatarItem节点下，配置在Subtab节点下表示Subtab节点下的所有item都依赖此binData中配置的属性，配置在AvatarItem节点下表示只有此item会依赖binData中的配置数据。 举例： 1. 发型和发色有依赖关系，发型修改的时还需使用上次用户设置过的发色，这个时候就需要在发型的中设置此字段。 2. 对于连衣裙和上衣、裤子的关系，当设置连衣裙的时候就需要将裤子和上衣设置为none,否则页面展示异常，所以可以在连衣裙的节点下配置此字段,具体参考demo中的AvatarTPose_panel.json。 3. 对于眼镜和镜片就存在这种依赖关系，眼镜依赖眼镜片，所以在每个眼镜的item下都配置了对应的bindData字段来关联镜片信息。
onNoneListener	BindData列表类型	否	字段解释：用于当items中没有任何选中的情况下使用此字段中的配置信息。 举例： 对于裤子、上衣 和连衣裙，在连衣裙中配置了此属性，当用户点击了连衣裙后，就不再选中上衣和裤子中的任何item，但是当客户再次点击上衣时，这个时候就需要给avatar形象设置一个默认的裤子（否则形象没有裤子），这时就可以解析此字段中配置的默认裤子，进行设置。具体参考demo中的AvatarTPose_panel.json。
items	AvatarItem列表类型	是	AvatarItem 类型列表数据

AvatarItem

字段	类型	是否必填	含义
id	String	是	每一个属性的 ID，和 SDK 返回的 AvatarData 数据中的 ID 相对应
icon	String	是	图片地址或者 ARGB 色值（”#FF0085CF“）
type	Int	是	UI 展示类型，`AvatarData.TYPE_SLIDER` 为滑竿类型，`AvatarData.TYPE_SELECTOR` 为 icon 类型
selected	boolean	是	如果 type 为 AvatarData.TYPE_SELECTOR 类型，此字段用于表示此item是否被选中
downloadUrl	String	否	配置文件的下载地址，用于动态下载配置文件
category	String	是	和 SubTab 中的 category 同义

labels	Map<String, String>	否	在 type 为 AvatarData.TYPE_SLIDER 类型时有值，存放面板左侧展示的中文 label
enLabels	Map<String, String>	否	在 type 为 AvatarData.TYPE_SLIDER 类型时有值，存放面板左侧展示的英文 label
bindData	BindData列表类型	否	被依赖的属性配置字段，此字段可在 Subtab 节点下，也可配置在 AvatarItem 节点下，配置在 Subtab 节点下表示 Subtab 节点下的所有item都依赖此 binData 中配置的属性，配置在 AvatarItem 节点下表示只有此 item 会依赖 binData 中的配置数据。 举例： 1. 发型和发色有依赖关系，发型修改的时还需使用上次用户设置过的发色，这个时候就需要在发型的中设置此字段。 2. 对于连衣裙和上衣、裤子的关系，当设置连衣裙的时候就需要将裤子和上衣设置为none,否则页面展示异常，所以可以在连衣裙的节点下配置此字段,具体参考 demo 中的AvatarTPose_panel.json。 3. 对于眼镜和镜片就存在这种依赖关系，眼镜依赖眼镜片，所以在每个眼镜的 item下都配置了对应的 bindData 字段来关联镜片信息。
avatarData	AvatarData	否	SDK 定义了属性操作类
animation	AvatarAnimation	否	SDK 定义了动作属性操作类

BindData

字段	类型	是否必填	含义
category	String	是	和 SubTab 中的 category 同义
id	String	是	每一个属性的 ID，和 SDK 返回的 AvatarData 数据中的 ID 相对应
firstLevelId	String	是	此数据所属的一级分类 ID
avatarData	AvatarData	是	SDK 定义了捏脸属性操作类
isTraverseOnSave	Boolean	是	在保存的时候是否遍历次 bindData 的数据，正常都需要遍历，除了帽子中配置的发型和发色，发型中配置的帽子和发色 不需要遍历

虚拟形象 SDK 说明

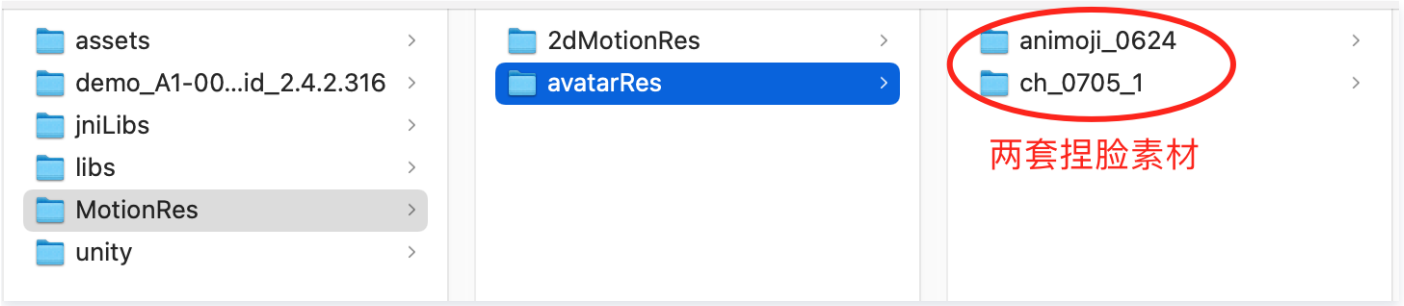
最近更新时间：2025-06-05 14:46:22

SDK 接入

SDK 的下载、接入、鉴权、跑通 Demo 请参见 独立集成腾讯特效 。

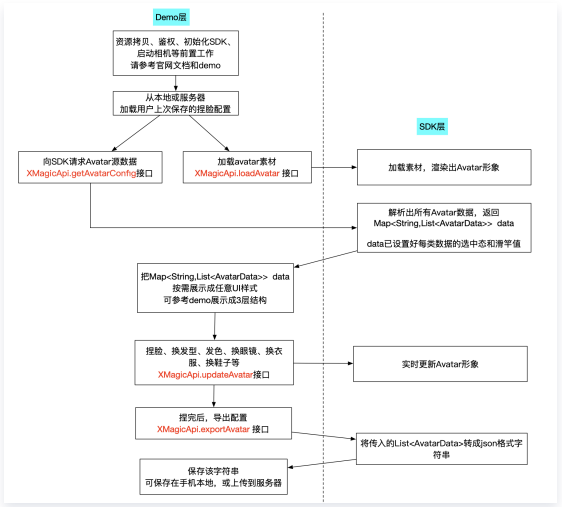
准备捏脸素材

目前我们随 SDK 提供了若干套捏脸、换装素材，素材在 SDK 解压后的 `MotionRes/avatarRes` 目录中，与其他的动效素材一样，您需要把它 copy 到工程的 assets 目录：

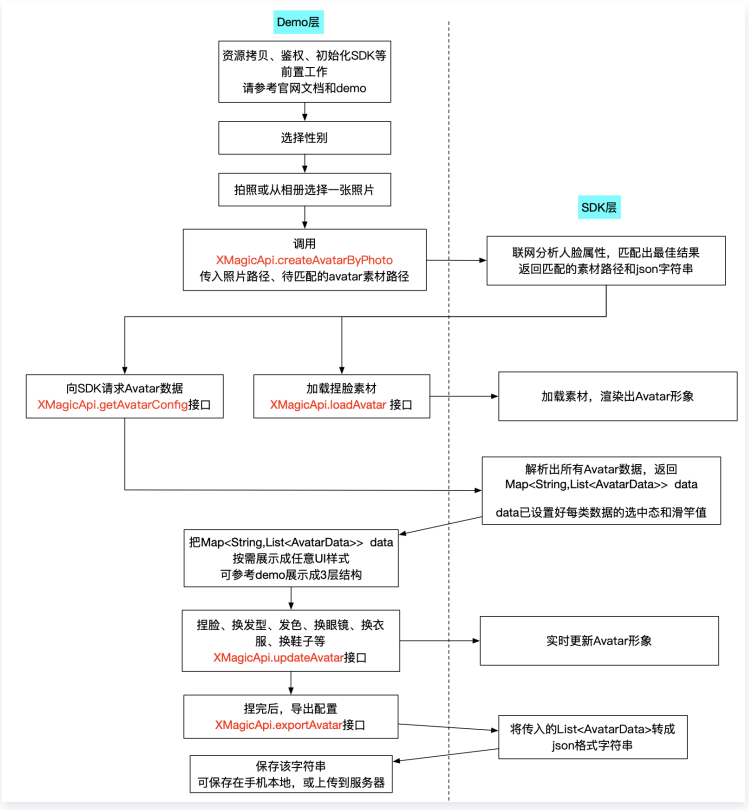


捏脸流程与 SDK 接口

捏脸流程



拍照捏脸流程



XMagicApi 的加载数据、捏脸、导出配置、拍照捏脸接口详情如下：

1. 加载 Avatar 素材（loadAvatar）

```
public void loadAvatar(XmagicProperty<?> property, UpdatePropertyListener updatePropertyListener)
```

Avatar 素材与普通的动效素材的加载方式是类似的，loadAvatar 接口与 SDK 的 updateProperty 接口是等价的，因此请参考 updateProperty 接口说明和 Demo 代码 。

2. 加载 Avatar 源数据（getAvatarConfig）

```
public static Map<String,List<AvatarData>> getAvatarConfig(String avatarResPath, String
savedAvatarConfigs)
```

- 输入参数：
  - **avatarResPath**：avatar 素材在手机上的绝对路径，例如 `/data/data/com.tencent.pitumotiondemo.effects/files/xmagic/MotionRes/avatarRes/animoji_0624` 。
  - **savedAvatarConfigs**：用户上次捏脸之后保存的数据，是 JSON 格式的字符串。首次使用或用户之前没有保存过的话，这个值为 null。
- 输出参数：
  以 map 的形式返回，map 的 key 是数据的 category，详见 AvatarCategory 类，map 的 value 是这个 category 下的全部数据。应用层拿到这份 map 后，按需展示成自己想要的 UI 样式。

3. 捏脸、换装（updateAvatar）

```
public void updateAvatar(List<AvatarData> avatarDataList, UpdateAvatarConfigListener
upDataAvatarConfigListener)
```

调用后实时更新当前素材的预览形象，一个 AvatarData 对象是一个原子配置（如换发型），一次可以传入多个原子配置（例如既换发型，又换发色）。该接口会检查传入 AvatarData 的有效性，有效的设置给 SDK，无效的数据会 callback 回去。

- 例如要求修改发型，但是头发模型文件（配置在 AvatarData 的 value 字段里）在本地找不到，就认为该 AvatarData 无效。
- 再例如要求修改瞳孔贴图，但是贴图文件（配置在 AvatarData 的 value 字段里）在本地找不到，就认为该 AvatarData 无效。

4. 导出捏脸配置（exportAvatar）

```
public static String exportAvatar(List<AvatarData> avatarDataList)
```

用户捏脸时，会修改 AvatarData 中的 selected 状态或形变值。捏完后，传入新的全量 AvatarData 列表，即可导出一份json字符串。这份字符串您可以存在本地，也可以上传到服务器。
导出的这份字符串有两个用途：

- 当下次再通过 XMagicApi 的 loadAvatar 接口加载这份 Avatar 素材时，您需要把这份 JSON 字符串设置给 XMagicProperty的customConfigs 字段，这样才能在预览中呈现出用户上次捏脸的形象。
- 如上文所述，调用 getAllAvatarData 时需要传入这个参数，以便修正Avatar源数据中的选中态和形变值。

5. 拍照、捏脸（createAvatarByPhoto）

该接口需要联网。

```
public void createAvatarByPhoto(String photoPath, List<String> avatarResPaths, boolean isMale,
            final FaceAttributeListener faceAttributeListener)
```

- **photoPath**：照片路径，请确保人脸位于画面中间。建议画面中只包含一个人脸，如果有多个人脸，SDK会随机选择一个。建议照片的短边大于等于500px，否则可能影响识别效果。
- **avatarResPaths**：您可以传入多套 Avatar 素材，SDK 会根据照片分析的结果，选择一套最合适的素材进行自动捏脸。

> ⚠ **注意：**
> 目前只支持一套，如果传入多套，SDK只会使用第一套。

- **isMale**：是否是男性，男性设置 true，女性设置 false。
- **faceAttributeListener**：如果失败，会回调 `void onError(int errCode, String msg)` 。如果成功，会回调 `void onFinish(String matchedResPath, String srcData)` ，第一个参数是匹配到的 Avatar 素材路径，第二个参数是匹配结果，与上文中的 exportAvatar 接口的返回值是一样的含义。

- onError 的错误码定义在 `FaceAttributeHelper.java`，具体如下：

```
public static final int ERROR_NO_AUTH = 1;//没有权限
public static final int ERROR_RES_INVALID = 5;//传入的Avatar素材路径无效
public static final int ERROR_PHOTO_INVALID = 10;//读取照片失败
public static final int ERROR_NETWORK_REQUEST_FAILED = 20;//网络请求失败
public static final int ERROR_DATA_PARSE_FAILED = 30;//网络返回数据解析失败
public static final int ERROR_ANALYZE_FAILED = 40;//人脸分析失败
public static final int ERROR_AVATAR_SOURCE_DATA_EMPTY = 50;//加载Avatar源数据失败
```

6. 将下载好的配置文件放置到对应的文件夹中（addAvatarResource）

```
public static Pair<Integer, List<AvatarData>> addAvatarResource(String resourceRootPath, String category,
String zipFilePath)
```

该接口主要用于动态下载Avatar配件的场景。举个例子，您的Avatar素材中有10种发型，后来想动态下发一种发型给客户端，下载完成后，得到一个压缩包，然后调用该接口，把压缩包路径传给SDK，SDK会解析这份压缩包，将它放到对应的 category 目录下。下次您在调用 getAvatarConfig 接口时，SDK就能解析出新添加的这份数据。

参数说明：

- **resourceRootPath**：Avatar 素材的根目录，例如 `/data/data/com.tencent.pitumotiondemo.effects/files/xmagic/MotionRes/avatarRes/animoji_0624`
- **category**：下载的这份配件的分类
- **zipFilePath**：下载的 zip 包地址
  - 接口返回 `Pair<Integer, List<AvatarData>>`，pair.first是错误码，pair.second是新添加的数据集合。
  - 错误码如下：

```
public interface AvatarActionErrorCode {
    int OK = 0;
    int ADD_AVATAR_RES_INVALID_PARAMS = 1000;
    int ADD_AVATAR_RES_ROOT_RESOURCE_NOT_EXIST = 1001;
    int ADD_AVATAR_RES_ZIP_FILE_NOT_EXIST = 1002;
    int ADD_AVATAR_RES_UNZIP_FILE_FAILED = 1003;
    int ADD_AVATAR_RES_COPY_FILE_FAILED = 1004;
}
```

7. 调用 AvatarData

上述接口的核心都是 AvatarData 类，其主要内容如下：

```
public class AvatarData {
    /**
     * 选择型数据。例如眼镜，有很多种眼镜，使用时只能从中选择一个。
     */
    public static final int TYPE_SELECTOR = 0;

    /**
     * 滑竿调节型数据。例如调整脸颊宽度。
     */
    public static final int TYPE_SLIDER = 1;

    //例如 脸型、眼睛微调 等。AvatarCategory.java中定义了标准的category，如果不满足需求，也可以自定义category字符串，跟
已有的不冲突即可
    //不能为空。
    public String category;

    //标识每一个具体item 或者 每一组微调项。
    //例如每个眼镜都有自己的id。每一组微调项也有自己的id。
```

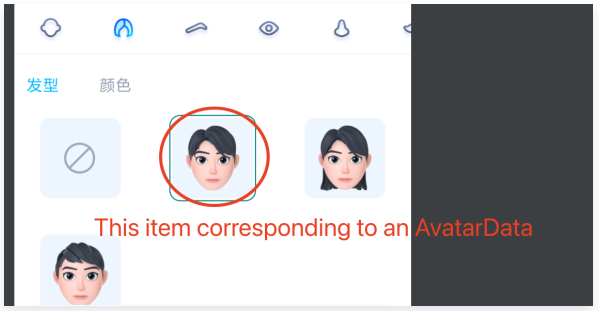
```
    //不能为空。
    public String id;

    //TYPE_SELECTOR 或者 TYPE_SLIDER
    public int type;

    //如果是selector类型，则它表示当前有无被选中
    public boolean selected = false;

    //每一个图标 或 每一组微调项 背后都对应着具体的配置详情，即下面这三要素。
    public String entityName;
    public String action;
    public JsonObject value;
}
```

一个 AvatarData 对象是一个原子配置，如换发型、调整脸颊等：

换发型	调整脸颊
	

- 捏脸时，如果数据是 selector 类型，则修改 AvatarData 的 selected 字段。例如有4种眼镜 A、B、C、D，默认选中的是 A，那么 A 的 selected 为 true，B、C、D 为 false。如果用户选择了眼镜 B，则把 B 的 selected 为 true，A、C、D 为 false。
- 捏脸时，如果数据是 slider 类型，则修改 AvatarData 的 value 字段。value 字段是一个 JsonObject，里面是若干对 key-value，把 key-value 中的 value 修改为滑竿的值即可。

8. 获取Avatar动画数据（getAvatarAnimations）

```
public static List<AvatarAnimation> getAvatarAnimations(String avatarResPath)
```

- **输入参数：avatarResPath**

  avatar 素材在手机上的绝对路径，例如

  `/data/data/com.tencent.pitumotiondemo.effects/files/xmagic/MotionRes/avatarRes/animoji_0624`
- **输出参数：**以 List 的形式返回所有的动画资源数据，详情可见 AvatarAnimation 类。

9. 将下载好的动画配置文件放置到对应的文件夹中（addAvatarAnimation）

```
public static Pair<Integer, List<AvatarAnimation>> addAvatarAnimation(String avatarResPath, String
zipFilePath)
```

该接口主要用于动态下载 Avatar 动画配件的场景。举个例子，您的 Avatar 素材中有1种动画，后来想动态下发一种动画给客户端，下载完成后，得到一个压缩包，然后调用该接口，把压缩包路径传给 SDK，SDK 会解析这份压缩包，将它放到对应的动画资源目录下。下次您在调用 getAvatarAnimations 接口时，SDK就能解析出新添加的这份数据。

参数说明：

- **avatarResPath：**Avatar 素材的根目录，例如

  `/data/data/com.tencent.pitumotiondemo.effects/files/xmagic/MotionRes/avatarRes/animoji_0624`
- **zipFilePath：**下载的 zip 包地址
  - 接口返回 `Pair<Integer, List<` AvatarAnimation `>>`，pair.first 是错误码，pair.second 是新添加的数据集合。

- 错误码如下：

```
public interface AvatarActionErrorCode {
    int OK = 0;
    int ADD_AVATAR_RES_INVALID_PARAMS = 1000;
    int ADD_AVATAR_RES_ROOT_RESOURCE_NOT_EXIST = 1001;
    int ADD_AVATAR_RES_ZIP_FILE_NOT_EXIST = 1002;
    int ADD_AVATAR_RES_UNZIP_FILE_FAILED = 1003;
    int ADD_AVATAR_RES_COPY_FILE_FAILED = 1004;
    int ADD_AVATAR_RES_PARSE_JSON_FILE_FAILED = 1005;
}
```

10. 播放/暂停 Avatar 动画（playAvatarAnimation）

```
public void playAvatarAnimation(AnimationPlayConfig animationConfig)
```

输入参数：
AnimationPlayConfig：动画播放信息描述类。此类主要包含以下信息。

```
public class AnimationPlayConfig {
    //action 描述，播放还是停止 动画
    public static final String ACTION_PLAY = "play";
    public static final String ACTION_PAUSE = "pause";
    public static final String ACTION_RESUME = "resume";
    public static final String ACTION_STOP = "stop";

    public String entityName;
    /** * 动画文件夹的路径，可以是相对于素材根目录的相对路径（例如 custom_configs/animations/Waving")
    /** 也可以是在手机上的绝对路径（例如 /data/data/xxx/xxx/Waving) */
    public String animPath;
    //值使用 ACTION_PLAY、ACTION_PAUSE、ACTION_RESUME、ACTION_STOP
    public String action;
    /** * 动画的名称 */
    public String animName;
    /** * 循环次数，-1表示无限 */
    public int loopCount = -1;
    /** * 动画的起始播放位置，单位是微秒 */
    public long startPositionUs = 0;
}
```

AvatarData 高级说明

AvatarData 中，对捏脸起关键作用的是 entityName、action、value 三个字段。这三个字段的值是 SDK 在解析素材配置时自动填入的。大多数情况下，您不需要了解这三个字段的含义，仅在 UI 层展示时，如果是滑竿类型，则需要解析 value 中的形变 key-value 与 UI 操作进行对应。
其中，AvatarData 要素分为：entityName 、action 和 value 字段。

entityName 字段

捏脸时，需要明确指定捏哪个部位，例如脸、眼睛、头发、上衣、鞋子 等。entityName 字段就是描述这些身体部位名称的。

action 和 value 字段

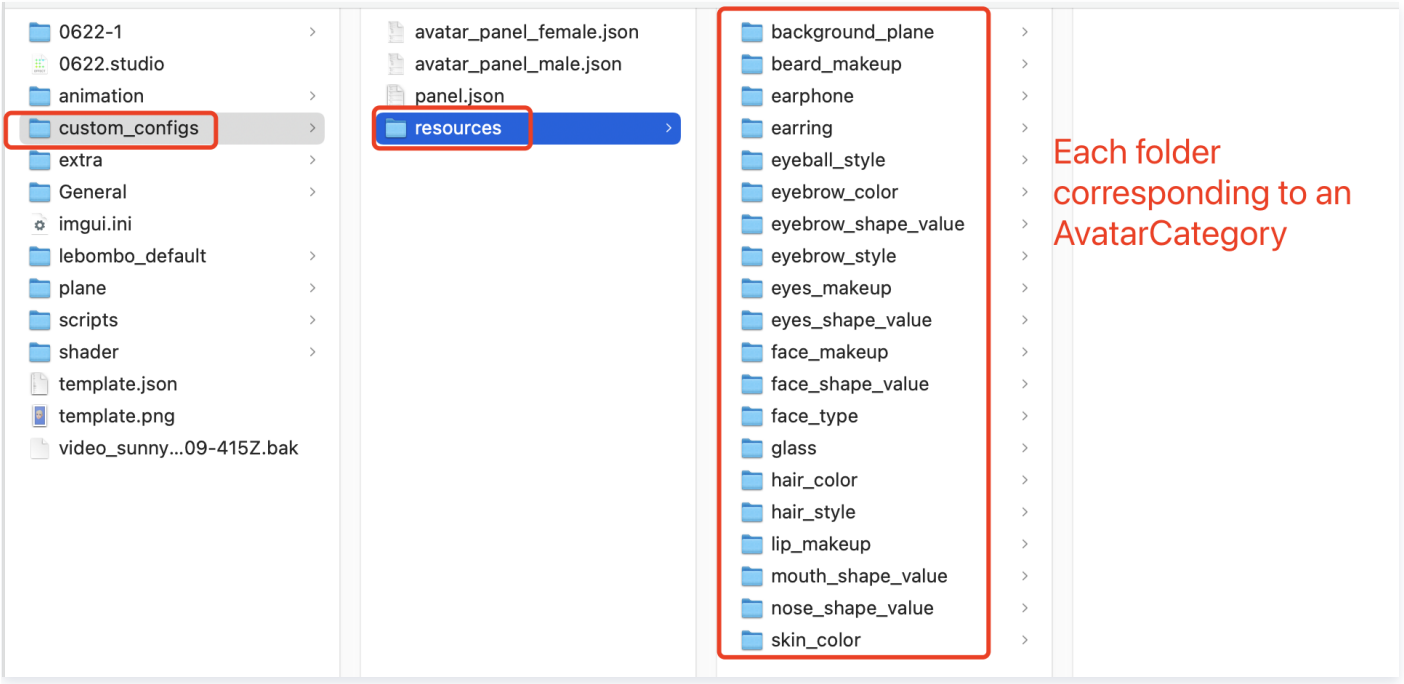
action 字段表示对 entityName 执行什么操作（action）。SDK 内定义了五种 action，均定义在 `AvatarAction.java` 中，每种 action 的含义及 value 要求如下：

action	含义	value要求
changeColor	修改当前材质的颜色，包括基础色、自发光色等颜色属性	JsonObject 类型，必填。由素材制作工具自动生成。

changeTexture	修改当前材质的贴图，包括颜色纹理贴图、金属粗糙度纹理贴图、AO 纹理贴图、法线纹理贴图、自发光纹理贴图等	JsonObject 类型。必填。由素材制作工具自动生成。
shapeValue	修改blendShape形变值，一般用于面部细节形变微调	JsonObject 类型。里面的 key 是形变名称，value 是float 类型的值。必填。由素材制作工具自动生成。
replace	替换子模型，例如替换眼镜、发型、衣服等	JsonObject 类型。里面描述了新的子模型的 3D 变换信息、模型路径、材质路径。如果要隐藏当前位置的子模型，则使用 null。由素材制作工具自动生成。
basicTransform	调整位置、旋转、缩放。一般用于调整摄像机的远近、角度，从而实现模型全身和半身视角的切换	JsonObject 类型。必填。由素材制作工具自动生成。

配置 Avatar 捏脸换装数据

avatar 属性配置存放在 resources 文件夹下（路径为：`素材/custom_configs/resources`）：



这些配置文件是自动生成的，通常不需要手动配置。自动生成的方式如下：
设计师按照设计规范，用 TencentEffectStudio 设计好一套形象后，运行我们提供的 resource_generator_gui 这个 App（目前仅支持 MacOS 平台），即可自动生成这些配置，详情请参见 设计规范说明 。

原子能力集成指引
人脸属性
Android

最近更新时间：2023-02-20 11:01:54

功能说明

输入一张包含人脸的照片，输出人物面部特征信息，包括眼睛、眉毛、发型、肤色、性别、年龄等。该接口需要联网，SDK 会把照片上传到 Server端进行解析。

接口说明

XMagicApi.java

```
public void getFaceFeatureFromPhoto(Bitmap bitmap, FaceFeatureListener listener);
```

- 参数 bitmap：请尽量让人脸位于画面中间，建议画面中只包含一个人脸，如果有多个人脸，SDK会随机选择一个。建议照片的短边大于等于500px，尺寸过小会影响识别效果。
- 参数 FaceFeatureListener，返回识别的结果

```
public interface FaceFeatureListener {
    void onError(int errCode, String msg);
    void onFinish(FaceDetailAttributesInfo faceInfo);
}
```

- onError 回调：解析失败时会回调此接口，错误码如下。

```
public static final int ERROR_NO_AUTH = 1;//没有权限
public static final int ERROR_RES_INVALID = 5;//传入的Avatar素材路径无效
public static final int ERROR_PHOTO_INVALID = 10;//读取照片失败
public static final int ERROR_NETWORK_REQUEST_FAILED = 20;//网络请求失败
public static final int ERROR_DATA_PARSE_FAILED = 30;//网络返回数据解析失败
public static final int ERROR_ANALYZE_FAILED = 40;//人脸分析失败
public static final int ERROR_AVATAR_SOURCE_DATA_EMPTY = 50;//加载Avatar源数据失败
```

- onFinish 回调：解析成功时回调此接口，FaceDetailAttributesInfo 说明如下。

```
    public static class FaceDetailAttributesInfo {
        public int age;  //[0,100]
        public int emotion;     //0：自然，1：高兴，2：惊讶，3：生气，4：悲伤，5：厌恶，6：害怕
        public Eye eye;      // 眼睛信息
        public Eyebrow eyebrow; // 眉毛信息
        public int gender;   // 性别信息。-1:没识别，0：男性，1：女性。
        public Hair hair;    // 发型信息
        public Hat hat;      // 帽子信息
        public int mask;     // 是否有口罩 -1:没识别，0:无，1有。
        public int moustache;    // 胡子信息。-1:没识别，0：无胡子，1：有胡子。
        public int nose;     // 鼻子信息。-1:没识别，0：朝天鼻，1：鹰钩鼻，2：普通，3：圆鼻头。
        public int shape;    // 脸型信息。 -1:没识别，0：方脸，1：三角脸，2：鹅蛋脸，3：心形脸，4：圆脸。
        public int skin;     // 肤色信息。 -1:没识别，0：黄色皮肤，1：棕色皮肤，2：黑色皮肤，3：白色皮肤。
        public int smile;    //微笑程度，[0,100]。
    }

    public static class Eye {
        public int eyelidType;   // 识别是否双眼皮。-1:没识别，0：无，1：有。
```

```
        public int eyeSize;      // 眼睛大小。-1:没识别，0：小眼睛，1：普通眼睛，2：大眼睛。
        public int glass;        // 识别是否佩戴眼镜。-1:没识别，0：无眼镜，1：普通眼镜，2：墨镜
        public int eyeOpen;      // 识别眼睛的睁开、闭合状态。-1:没识别，0：睁开，1：闭眼
    }

    public static class Eyebrow {
        public int eyebrowLength;    //眉毛长短。0：短眉毛，1：长眉毛。
        public int eyebrowDensity;   //眉毛浓密。 0：淡眉，1：浓眉。
        public int eyebrowCurve;     // 眉毛弯曲。0：不弯，1：弯眉。
    }

    public static class Hair {
        public int length; //头发长度信息。 0：光头，1：短发，2：中发，3：长发，4：绑发。
        public int bang;     //刘海信息。 0：无刘海，1：有刘海。
        public int color;     //头发颜色信息。0：黑色，1：金色，2：棕色，3：灰白色。
    }

    public static class Hat {
        public int style;    //帽子佩戴状态信息。0：不戴帽子，1：普通帽子，2：头盔，3：保安帽。
        public int color;    //帽子颜色.0：不戴帽子，1：红色系，2：黄色系，3：蓝色系，4：黑色系，5：灰白色系，6：混色
系子。
    }
```

语音转表情
iOS

最近更新时间：2023-02-20 11:01:54

功能说明

输入音频数据，输出苹果 ARKit 标准的52表情数据，请参见 ARFaceAnchor 。您可以利用这些表情数据做一进步的开发，例如传到 Unity 中驱动您的模型。

接入方式

方式1：与腾讯特效 SDK 一起使用

1. 语音转表情sdk结合腾讯特效 SDK 中，因此第一步需要按照腾讯特效文档进行接入。
2. 下载 腾讯特效 SDK 完整版 。
3. 参考 独立集成腾讯特效 文档完成集成。
4. 将腾讯特效 SDK 完整版 内的Audio2Exp.framework 拉入项目，并且在项目的target->General->Frameworks,Libraries,and Embedded Content处设置为Embed & Sign。

方式2：通过独立的语音转表情 SDK 接入

如果您只需要语音转表情，不需要用到腾讯特效 SDK 的任何能力，则可以考虑使用独立的语音转表情 SDK，Audio2Exp.framework 包约7MB左右。项目引入Audio2Exp.framework、YTCommonXMagic.framework两个动态库，并且在项目的target->General->Frameworks,Libraries,and Embedded Content处设置为Embed & Sign

接入步骤

1. 设置 License，请参见 鉴权 。
2. 配置模型文件：请将必需的模型文件audio2exp.bundle拷贝到工程目录，然后在调用 Audio2ExpApi 的 initWithModelPath: 接口时，传入参数 audio2exp.bundle" 模型文件所在的路径。

接口说明

接口	说明
+ (int)initWithModelPath:(NSString*)modelPath;	初始化，传入模型路径，见上文说明。返回值为0表示成功
+ (NSArray *)parseAudio:(NSArray *)inputData;	输入的是音频数据，要求单通道，16K采样率，数组长度为267（即267个采样点），输出的数据是长度为52的float数组，表示52表情基，取值为0到1之间，顺序为 苹果标准顺序 {"eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft","eyeSquintLeft","eyeWideLeft","eyeBlinkRight","eyeLookDownRight","eyeLookInRight","eyeLookOutRight","eyeLookUpRight","eyeSquintRight","eyeWideRight","jawForward","jawLeft","jawRight","jawOpen","mouthClose","mouthFunnel","mouthPucker","mouthRight","mouthLeft","mouthSmileLeft","mouthSmileRight","mouthFrownRight","mouthFrownLeft","mouthDimpleLeft","mouthDimpleRight","mouthStretchLeft","mouthStretchRight","mouthRollLower","mouthRollUpper","mouthShrugLower","mouthShrugUpper","mouthPressLeft","mouthPressRight","mouthLowerDownLeft","mouthLowerDownRight","mouthUpperUpLeft","mouthUpperUpRight","browDownLeft","browDownRight","browInnerUp","browOuterUpLeft","browOuterUpRight","cheekPuff","cheekSquintLeft","cheekSquintRight","noseSneerLeft","noseSneerRight","tongueOut"}
+ (int)releaseSdk	使用完毕后调用，释放资源

集成代码示例

```
// 初始化语音转表情sdk
NSString *path = [[NSBundle mainBundle] pathForResource:@"audio2exp" ofType:@"bundle"];
int ret = [Audio2ExpApi initWithModelPath:path];
// 语音数据转52表情数据
```

```
NSArray *emotionArray = [Audio2ExpApi parseAudio:floatArr];
// 释放sdk
[Audio2ExpApi releaseSdk];

// 结合腾讯特效sdk xmgaic使用
// 使用对应的资源初始化美颜sdk
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDict];
// 加载avatar素材
[self.beautyKit loadAvatar:bundlePath exportedAvatar:nil completion:nil];
// 将52表情数据传入美颜sdk 就能看到效果
[self.beautyKit updateAvatarByExpression:emotionArray];
```

说明：
完整的示例代码请参考 美颜特效 SDK demo 工程 。

Android

最近更新时间：2023-02-20 11:01:54

功能说明

输入音频数据，输出苹果 ARKit 标准的52表情数据，请参见 ARFaceAnchor 。您可以利用这些表情数据做一进步的开发，例如传到 Unity 中驱动您的模型。

接入方式

方式1：通过虚拟形象 SDK 接入

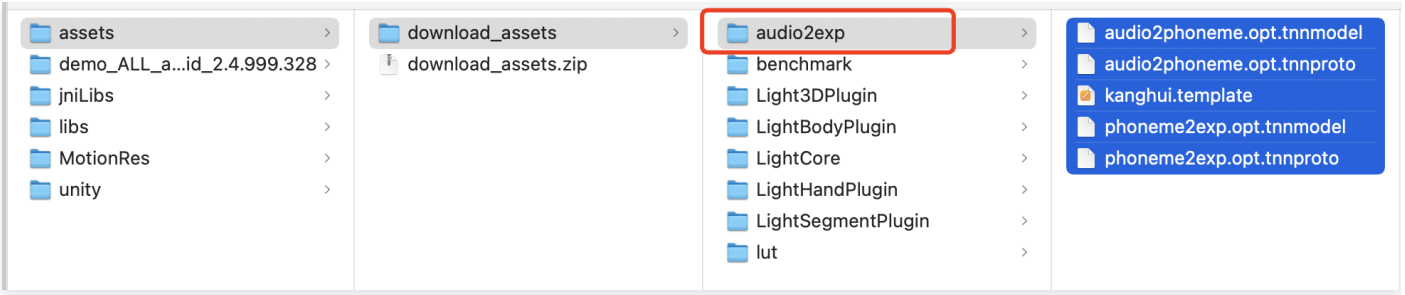
语音转表情集成在虚拟形象 SDK 中，因此第一步需要按照虚拟形象文档进行接入。

方式2：通过独立的语音转表情 SDK 接入

如果您只需要语音转表情的 AI 原子能力，不需要用到虚拟形象 SDK 的能力，则可以考虑使用独立的语音转表情 SDK，aar 包约6MB左右。请联系我们的架构师或销售获取此 SDK。

接入步骤

1. **设置 License**，请参见 鉴权 。
2. **配置模型文件**：请将必需的模型文件从 assets 拷贝到 app 的私有目录，例如：`context.getFilesDir() + "/my_models_dir/audio2exp"`，然后在调用 Audio2ExpApi 的 `init(String modelPath)` 接口时，传入参数 `context.getFilesDir() + "/my_models_dir"`
   模型文件在 SDK 包里，位置如下：



接口说明

接口	说明
public int Audio2ExpApi.init(String modelPath);	初始化，传入模型路径，见上文说明。返回值为0表示成功
public float[] Audio2ExpApi.parseAudio(float[] inputData);	输入的是音频数据，要求单通道，16K采样率，数组长度为267（即267个采样点），输出的数据是长度为52的float数组，表示52表情基，取值为0到1之间，顺序为 苹果标准顺序 `{"eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft","eyeSquintLeft","eyeWideLeft","eyeBlinkRight","eyeLookDownRight","eyeLookInRight","eyeLookOutRight","eyeLookUpRight","eyeSquintRight","eyeWideRight","jawForward","jawLeft","jawRight","jawOpen","mouthClose","mouthFunnel","mouthPucker","mouthRight","mouthLeft","mouthSmileLeft","mouthSmileRight","mouthFrownRight","mouthFrownLeft","mouthDimpleLeft","mouthDimpleRight","mouthStretchLeft","mouthStretchRight","mouthRollLower","mouthRollUpper","mouthShrugLower","mouthShrugUpper","mouthPressLeft","mouthPressRight","mouthLowerDownLeft","mouthLowerDownRight","mouthUpperUpLeft","mouthUpperUpRight","browDownLeft","browDownRight","browInnerUp","browOuterUpLeft","browOuterUpRight","cheekPuff","cheekSquintLeft","cheekSquintRight","noseSneerLeft","noseSneerRight","tongueOut"}`
public int Audio2ExpApi.release();	使用完毕后调用，释放资源

集成代码示例

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.button).setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View view) {
                        TELicenseCheck.getInstance().setTELicense(MainActivity.this, licenseUrl,
licenseKey, new TELicenseCheckListener() {
                                @Override
                                public void onLicenseCheckFinish(int errorCode, String s) {
                                        Log.d(TAG, "onLicenseCheckFinish: errorCode =
"+errorCode+",msg="+s);
                                        if (errorCode == TELicenseCheck.ERROR_OK) {
                                                //license check success
                                                Audio2ExpApi audio2ExpApi = new Audio2ExpApi();
                                                int err =
audio2ExpApi.init(MainActivity.this.getFilesDir() +"/models");
                                                Log.d(TAG, "onLicenseCheckFinish: err="+err);
                                                //TODO start record and parse audio data
                                        } else {
                                                // license check failed
                                        }
                                }
                        });
                }
        });
}
```

说明
完整的示例代码请参考 虚拟形象 SDK demo 工程 。

- 录音可以参考 `com.tencent.demo.avatar.audio.AudioCapturer` 。
- 接口使用可以参考： `com.tencent.demo.avatar.activity.Audio2ExpActivity` 及其相关类。