# 腾讯特效 SDK

# SDK 集成指引（含 UI）

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

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。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
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默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

# 文档目录

SDK 集成指引（含 UI）
- 通用集成腾讯特效
  - Android
  - iOS
- TRTC SDK 集成腾讯特效
  - iOS
  - Android
  - Flutter
  - uni-app
- 直播 SDK 集成腾讯特效
  - iOS
  - Android
  - Flutter
- 短视频 SDK 集成腾讯特效
  - iOS
  - Android
- Avatar 虚拟人集成指引
  - iOS
    - 快速跑通 Demo
    - 快速接入 Avatar
    - 自定义 Avatar UI
    - Avatar SDK 说明
  - Android
    - 快速跑通 demo
    - 快速接入 Avatar
    - 自定义 Avatar UI
    - Avatar SDK 说明

版权所有：腾讯云计算（北京）有限责任公司

# SDK 集成指引（含 UI）
# 通用集成腾讯特效
# Android

最近更新时间：2024-11-21 17:24:12

## TEBeautyKit 功能说明

为方便客户快速接入美颜，并简化 UI 面板相关的开发工作，我们提供了美颜特效UI组件：TEBeautyKit ，它包含了对 SDK 的进一步封装以及可定制化的 UI 面板，效果如下图。如果您不想使用这种 UI，可以参见 无UI集成腾讯特效 。

## 参考 Demo

从 github clone 出 demo 工程，按照 TEBeautyDemo/README 文档中的指引将 TEBeautyDemo 运行起来，然后结合本文了解含 UI 集成 SDK 的详细步骤。

## SDK集成

> **注意：**
> 此库只支持腾讯特效SDK V3.5.0及以上版本。

### 第一步：添加腾讯特效SDK

- 集成SDK
- 集成素材

### 第二步：添加 TEBeautyKit 依赖

源码集成（推荐）

请将 demo 工程 中的 tebeautykit module 拷贝到您的工程中，修改 `tebeautykit/build.gradle` 中依赖的SDK套餐类型和版本号，与“第一步”中 的套餐和版本号保持一致。

```
implementation 'com.tencent.mediacloud:TencentEffect_S1-04:SDK版本号'
```

然后在您 app 相关 module 中添加对 tebeautykit module 的依赖：

```
implementation project ( ':tebeautykit' )
```

### Maven 集成

在您 app 的 dependencies 中添加对 TEBeautyKit 库的依赖，“SDK版本号” 与 “第一步” 中的版本号保持一致。

```
dependencies{
    ...
    implementation 'com.tencent.mediacloud:TEBeautyKit:SDK版本号'
}
```

由于 tebeautykit 还依赖了 gson、okhttp 等组件，因此需要再添加如下依赖：

```
dependencies{
    implementation 'com.google.code.gson:gson:2.8.2'
    implementation 'com.squareup.okhttp3:okhttp:3.9.0'
    implementation 'com.github.bumptech.glide:glide:4.12.0'
    implementation 'androidx.appcompat:appcompat:1.0.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
}
```

### 下载 aar 集成

- 下载 tebeautykit(下载之后是一个 zip 文件，解压即可得到 aar 文件)
- 将 `tebeautykit-xxxx.aar` 文件拷贝到 app 工程 `libs` 目录下
- 打开 app 模块的 `build.gradle` 添加依赖引用：

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

由于 tebeautykit 还依赖了 gson、okhttp 等组件，因此需要再添加如下依赖：

```
dependencies{
    implementation 'com.google.code.gson:gson:2.8.2'
    implementation 'com.squareup.okhttp3:okhttp:3.9.0'
    implementation 'com.github.bumptech.glide:glide:4.12.0'
    implementation 'androidx.appcompat:appcompat:1.0.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
}
```

## 第三步：添加面板 json 文件

从 demo 工程 的 demo/src/main/assets/beauty_panel 获取面板配置文件，或者点击此处 下载 并解压。文件包含了 美颜、美体、滤镜、动效贴纸和分割属性的配置，请根据您的套餐类型选择一组 json 文件，放置在自己工程中的 `assets/beauty_panel` 文件夹下（**panel_icon 也需要放置在此文件夹中**）。

下载的压缩包中包含如上图文件，每一个套餐名下包含若干个 json 文件，下表对各个 json 文件进行说明：

| 文件 | 说明 |
| --- | --- |
| beauty.json | 美颜、美型、画面调整等配置文件 |
| beauty_body..js | 美体配置文件 |

| | |
|---|---|
| on | |
| lut.json | 滤镜配置文件。注意：由于不同客户使用的滤镜素材不一样，所以客户在下载之后，可以按照 json 结构进行自行配置（可参见 json 文件结构说明） |
| makeup.json | 风格整妆配置文件。 注意：由于不同客户使用的风格整妆素材不一样，所以客户在下载之后，可以按照 json 结构进行自行配置（可参见 json 文件结构说明） |
| motions.json | 动效贴纸配置文件。注意：由于不同客户使用的动效贴纸素材不一样，所以客户在下载之后，可以按照 json 结构进行自行配置（可参见 json 文件结构说明） |
| segmentation.json | 背景分割（虚拟背景）配置文件。注意：由于不同客户使用的分割素材不一样，所以客户在下载之后，可以按照 json 结构进行自行配置（可参见 json 文件结构说明） |
| panel_icon | 此文件夹中用于存放 json 文件中配置的图片，必须添加 |

## SDK 使用

强烈建议您参考 demo 工程 的 `TEMenuActivity.java` 和 `TECameraBaseActivity.java` 来了解如何接入 TEBeautyKit。

### 第一步：鉴权

1. 申请授权，得到 License URL 和 License KEY，请参见 License 指引 。
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**。例如我们的demo工程是在 Application 的 onCreate 方法里触发下载，但在您的项目中不建议在这里触发，因为此时可能没有网络权限或联网失败率较高，请选择更合适的时机触发 license 下载。

```
//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。
//TEApplication.java
TEBeautyKit.setTELicense(context, LicenseConstant.mXMagicLicenceUrl,
LicenseConstant.mXMagicKey, null);
```

3. 然后在真正要使用美颜功能前（例如启动相机前），再去做鉴权：

```
//TEMenuActivity.java
TEBeautyKit.setTELicense(context, LicenseConstant.mXMagicLicenceUrl,
LicenseConstant.mXMagicKey, 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 |

## 第二步：设置路径

```
  //TEMenuActivity.java
  String resPath = new File(getFilesDir(),
AppConfig.getInstance().getBeautyFileDirName()).getAbsolutePath();
  TEBeautyKit.setResPath(resPath);
```

## 第三步：资源拷贝

这里所指的资源文件包含两部分：

- SDK 的模型文件，位于 SDK 的 aar 包的 assets 目录。
- 滤镜和动效资源文件，位于 demo 工程的 assets 目录，命名分别是 lut 和 MotionRes。

使用美颜前需要将上述资源拷贝到 “第二步” 设置的 resPath。在未更新 SDK 版本的情况下，只需要拷贝一次。拷贝成功后，您可以在 App 的 SharedPreference 中记录下来，下次就不用再拷贝了。具体可以参见 demo 工程的 `TEMenuActivity.java` 的 `copyRes` 方法。

```
//TEMenuActivity.java
private void copyRes() {
    if (!isNeedCopyRes()) {
        return;
    }
    new Thread(() -> {
        TEBeautyKit.copyRes(getApplicationContext());
    }).start();
}
```

## 第四步：初始化 TEBeautyKit，并将 view 添加到页面中

```
//TECameraBaseActivity.java
TEBeautyKit.create(this.getApplicationContext(), beautyKit -> {
    mBeautyKit = beautyKit;
    initBeautyView(beautyKit);
});

public void initBeautyView(TEBeautyKit beautyKit){
    TEPanelViewResModel resModel = new TEPanelViewResModel();
    String combo = "S1_07";    //根据您的套餐进行设置，如果您的套餐没有包含某项功能，客户设置
为null
    resModel.beauty = "beauty_panel/"+combo+"/beauty.json";
    resModel.lut = "beauty_panel/"+combo+"/lut.json";
    resModel.beautyBody = "beauty_panel/"+combo+"/beauty_body.json";
    resModel.motion = "beauty_panel/"+combo+"/motions.json";
    resModel.lightMakeup = "beauty_panel/"+combo+"/light_makeup.json";
    resModel.segmentation = "beauty_panel/"+combo+"/segmentation.json";
    TEUIConfig.getInstance().setTEPanelViewRes(resModel);

    mTEPanelView = new TEPanelView(this);
    mTEPanelView.setTEPanelViewCallback(this);
    mTEPanelView.setupWithTEBeautyKit(beautyKit);
    mTEPanelView.showView(this);
    this.mPanelLayout.addView(mTEPanelView, new
LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
}
```

`TEUIConfig.getInstance().setTEPanelViewRes` 这个方法里的参数是上文中提到的 src/main/assets/beauty_panel 里的 json 文件，如果您不需要某一项，则在对应的位置传 `null` 即可。

## 第五步：使用美颜

我们假定您已经实现了相机应用，能正常启动相机，且能将相机的 SurfaceTexture 纹理信息回调到 Activity 用于美颜处理，如下所示：

```
//TECameraBaseActivity.java
@Override
public int onCustomProcessTexture(int textureId, int textureWidth, int
textureHeight) {
    //美颜SDK在这里处理textureId，为其添加美颜和特效，并返回处理后的新的textureID
}
```

如果您尚未实现相机应用，可以参考demo工程的 `TECameraBaseActivity.java` ，使用 `GLCameraXView` 这个组件，将它添加到您的 Activity 的 layout 中，以快速实现相机预览：

```
<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" />
```

美颜 SDK 处理每帧数据并返回相应处理结果。process 方法详细说明见 API 文档 。

```
//TECameraBaseActivity.java
@Override
public int onCustomProcessTexture(int textureId, int textureWidth, int
textureHeight) {
     return mBeautyKit.process(textureId, textureWidth, textureHeight);
}
```

## 第六步：生命周期管理

- 生命周期方法onResume，建议在 `Activity` 的 `onResume()` 方法中调用，调用后会恢复特效里的声音。

  ```
  mBeautyKit.onResume();
  ```

- 生命周期方法onPause，建议在 Activity 的 `onPause()` 方法调用，调用后会暂停特效里的声音。

  ```
  mBeautyKit.onPause();
  ```

- 释放美颜 SDK，在 OpenGL 环境销毁时调用，**需要在 GL 线程中调用，不能在主线程（Activity 的 onDestroy 里）调用**，否则可能造成资源泄露，多次进出后引起白屏、黑屏现象。

```
@Override
public void onGLContextDestroy() {
    mBeautyKit.onDestroy();
}
```

### 第七步：导出和导入美颜参数

导出当前美颜参数并保存在 SharedPreference 里：

```
String json = mBeautyKit.exportInUseSDKParam();
if (json != null) {
    getSharedPreferences("demo_settings", Context.MODE_PRIVATE).edit().
      putString("current_beauty_params", json).commit();
}
```

下次初始化TEBeautyKit时，导入这个字符串以修改默认美颜效果：

```
public void initBeautyView(TEBeautyKit beautyKit){
    TEUIConfig.getInstance().setTEPanelViewRes( beauty: "beauty_panel/S1_07/beauty.json",  beautyBody: "beauty_pane
            lut: "beauty_panel/S1_07/lut.json",  motion: "beauty_panel/S1_07/motions.json",
            makeup: "beauty_panel/S1_07/makeup.json",  segmentation: "beauty_panel/S1_07/segmentation.json");
    mTEPanelView = new TEPanelView( context: this);
    mTEPanelView.setTEPanelViewCallback(this);
    mTEPanelView.setupWithTEBeautyKit(beautyKit);

    SharedPreferences sp = getSharedPreferences( name: "demo_settings", Context.MODE_PRIVATE);
    String savedParams = sp.getString( s: "current_beauty_params",  s1: "");
    if (!savedParams.isEmpty()) {
        mTEPanelView.setLastParamList(savedParams);
    }

    mTEPanelView.showView( tePanelViewCallback: this);
    this.mPanelLayout.addView(mTEPanelView, new LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
}
```

## 附录

### 面板 JSON 文件说明

- 美颜、美体 。

```
ld.gradle (:tebeautykit)   beauty.json
{
    "displayName":"美颜",
    "displayNameEn":"Beauty effects",
    "propertyList":[
        {
            "displayName":"关闭",
            "displayNameEn":"None",
            "icon":"beauty_panel/panel_icon/beauty/none.png"
        },
        {
            "displayName":"美白",
            "displayNameEn":"Brighten",
            "icon":"beauty_panel/panel_icon/beauty/beauty_whiten.png",
            "propertyList":[...],
            "uiState":2
        },
        {
            "displayName":"磨皮",
            "displayNameEn":"Smoothskin",
            "icon":"beauty_panel/panel_icon/beauty/beauty_smooth.png",
            "sdkParam":{
                "effectName":"smooth.smooth",
                "effectValue":40
            },
            "uiState":1
        },
        {
            "displayName":"红润",
            "displayNameEn":"Rosyskin",
            "icon":"beauty_panel/panel_icon/beauty/beauty_ruddy.png",
            "sdkParam":{
                "effectName":"smooth.rosy",
                "effectValue":0
            }
        }
    ],
    "uiState":2
}
```

| 字段 | 说明 |
| --- | --- |
| displayName | 中文名称 |
| displayNameEn | 英文名称 |
| icon | 图片地址，支持设置本地图片和网络图片，本地图片支持 assets 资源和 SD 资源，assets 图片如上图所示，SD 卡图片设置图片全路径，网络图片设置对应的http链接 |
| sdkParam | 美颜 SDK 需要用到的属性，共包含四个属性，可参见美颜参数表 |
| effectName | 美颜属性 key，参见 属性参数表 |
| effectValue | 设置属性强度，参见 属性参数表 |
| resourcePath | 设置资源路径，参见 属性参数表 |
| extraInfo | 设置其他信息，参见 属性参数表 |

- **滤镜、动效贴纸、分割 。**

```
{
    "displayName":"滤镜",
    "displayNameEn":"Filters",
    "downloadPath":"light_material/lut/",
    "propertyList":[
        {
            "displayName":"无",
            "displayNameEn":"None",
            "icon":"beauty_panel/panel_icon/beauty/none.png"
        },
        {
            "displayName":"自然",
            "displayNameEn":"Natural",
            "icon":"beauty_panel/panel_icon/lut_icon/ziran_lf.png",
            "resourceUri":"light_material/lut/ziran_lf.png",
            "sdkParam":{
                "effectValue":60
            }
        },
        {
            "displayName":"自然-2",
            "displayNameEn":"Natural-2",
            "icon":"beauty_panel/panel_icon/lut_icon/ziran2_lf.png",
            "resourceUri":"https://mediacloud-76607.gzc.vod.tencent-cloud.com/TencentEffect/demoMotion/encrypted_lut/lut/ziran2_lf.png",
            "sdkParam":{
                "effectValue":60
            }
        },
```

由于**滤镜和动效贴纸、分割**的配置基本一致，所以此处用滤镜的JSON进行说明，这里新增了downloadPath和resourceUri字段。

| 字段 | 说明 |
| --- | --- |
| downloadPath | 如果您的滤镜素材是网络下载，那么这里配置的是您素材下载后在本地的存放位置，这里是**相对路径**，全路径是您在 `TEBeautyKit.setResPath` 时设置的路径 + **此处设置的路径** |
| resourceUri | 如果您的素材是需要通过网络下载的，那么这里配置网络地址，如上图第三个红框，如果您的滤镜素材在本地，则按照上图配置对应的本地地址。 |

- **风格整妆**

```
{
    "displayName":"风格整妆",
    "displayNameEn":"Makeup",
    "downloadPath":"MotionRes/makeupRes/",
    "propertyList":[
        {
            "displayName":"无",
            "displayNameEn":"None",
            "icon":"beauty_panel/panel_icon/beauty/none.png"
        },
        {
            "displayName":"微闪",
            "displayNameEn":"Glitter",
            "icon":"beauty_panel/panel_icon/motions_icon/video_makeup_weishan.png",
            "resourceUri":"https://mediacloud-76607.gzc.vod.tencent-cloud.com/TencentEffect/d
            "sdkParam":{
                "effectValue":80,
                "extraInfo":{
                    "makeupLutStrength":"60"
                }
            }
        },
```

在风格整妆中增加了 `extraInfo` 下的 `makeupLutStrength` 字段，此字段用于调节风格整妆素材中**滤镜的强度**（如果此风格整妆素材支持调节滤镜强度就进行配置），此字段可参考美颜参数表。

## TEBeautyKit 方法说明

```
//异步创建TEBeautyKit对象
public static void create(@NonNull Context context, @NonNull OnInitListener
initListener)

//isEnableHighPerformance 是否开启高性能模式，
//高性能模式开启后，美颜占用的系统CPU/GPU资源更少，可减少手机的发热和卡顿现象，更适合低端机长时间
使用。
//但请注意：开启高性能模式后，以下美颜项将不可用：
//1.眼部：眼宽、眼高、祛眼袋
//2.眉毛：角度、距离、高度、长度、粗细、眉峰
//3.嘴部：微笑唇
//4.面部：瘦脸（自然，女神，英俊），收下颌，祛皱、祛法令纹。建议用“脸型”实现综合大眼瘦脸效果
public static void create(@NonNull Context context, boolean
isEnableHighPerformance, @NonNull OnInitListener initListener)

//TEBeautyKit的构造方法，用于同步创建TEBeautyKit对象
public TEBeautyKit(Context context)

/**
 * @param context                 应用上下文
 * @param isEnableHighPerformance 是否开启高性能模式
 */
public TEBeautyKit(Context context, boolean isEnableHighPerformance)

/**
 * 设置静音
 *
 * @param isMute true 表示静音
 */
public void setMute(boolean isMute)

/**
 * 如果是对图片进行美颜处理，需要调用此方法设置数据源的类型，分别为相机数据源和图片数据源：
 * 相机数据源：XmagicApi.PROCESS_TYPE_CAMERA_STREAM  图片数据源：
XmagicApi.PROCESS_TYPE_PICTURE_DATA。
 * @param type 默认是视频流类型
 */
public void setBeautyStreamType(int type)

/**
 * 设置某个特性的开或关
 *
 * @param featureName 取值见 XmagicConstant.FeatureName
 * @param enable      true表示开启，false表示关闭
 */
public void setFeatureEnableDisable(String featureName, boolean enable)
```

```
/**
 * 对图片进行美颜处理
 *
 * @param bitmap
 * @param needReset
 * @return
 */
public Bitmap process(Bitmap bitmap, boolean needReset)

/**
 * 处理 视频/摄像头 每一帧数据
 *
 * @param textureId 纹理id，此纹理需要的是纹理类型为GL_TEXTURE_2D，纹理像素格式为RGBA
 * @param width     纹理宽度
 * @param height    纹理高度
 * @return 处理后的纹理ID
 */
public int process(int textureId, int width, int height)

/**
 * 更新美颜属性
 *
 * @param paramList
 */
public void setEffectList(List<TEUIProperty.TESDKParam> paramList)

/**
 * 更新美颜属性
 *
 * @param teParam
 */
public void setEffect(TEUIProperty.TESDKParam teParam)

/**
 * 开启或关闭增强模式
 *
 * @param enableEnhancedMode true 表示开启增强模式  false 表示关闭增强模式
 * @return 返回true表示状态发生改变了，false 表示状态没有改变
 */
public boolean enableEnhancedMode(boolean enableEnhancedMode)

/**
 * 获取当前生效的美颜属性列表字符串。
 * 客户可以将导出的字符串进行本地保存，在下次创建TEPanelView对象后调用setLastParamList方法进
行设置。
 * @return
 */
public String exportInUseSDKParam()
```

```
/**
 * 用于恢复贴纸中的声音
 * 恢复陀螺仪传感器
 */
public void onResume()

/**
 * 用于暂停贴纸中的声音
 * 暂停陀螺仪传感器
 */
public void onPause()

/**
 * 销毁美颜
 * 注意：必须在gl线程调用此方法
 */
public void onDestroy()

/**
 * 设置事件监听，用于监听 手机方向事件，用于adapter
 * @param listener 事件监听回调
 */
public void setEventListener(EventListener listener)

/**
 * 设置setAIDataListener 回调
 *
 * @param aiDataListener
 */
public void setAIDataListener(XmagicApi.XmagicAIDataListener aiDataListener)

/**
 * 设置动效提示语回调函数，用于将提示语展示到前端页面上。
 *
 * @param tipsListener
 */
public void setTipsListener(XmagicApi.XmagicTipsListener tipsListener)

/**
 * 截取当前纹理上的画面
 *
 * @param callback
 */
public void exportCurrentTexture(XmagicApi.ExportTextureCallback callback)

/**
 * 设置纹理逆时针旋转的度数。
```

```
 * 主要作用：用于SDK内部对纹理进行旋转，旋转对应角度之后，让人头朝上，这样SDK内部就可以识别人脸
 * 默认情况下SDK内部会使用sensor传感器获取需要旋转的角度
 *
 * @param orientation 取值只有0、90、180、270
 */
public void setImageOrientation(TEImageOrientation orientation)

/**
 * 检测当前设备是否支持此素材
 *
 * @param motionResPath 素材文件的路径
 * @return
 */
public boolean isDeviceSupport(String motionResPath)

/**
 * 获取美颜特效的开关状态
 *
 * @return EffectState
 */
public EffectState getEffectState()

/**
 * 设置是否开启美颜
 *
 * @param effectState ENABLED, 表示开启  DISABLED 表示关闭
 */
public void setEffectState(EffectState effectState)

/**
 * 设置增强模式的策略实现类，如果不设置，则使用默认的实现
 *
 * @param teParamEnhancingStrategy 增强模式处理类
 */
public void setParamEnhancingStrategy(TEParamEnhancingStrategy
teParamEnhancingStrategy)



//静态方法如下

/**
 * 设置美颜资源存放的路径
 *
 * @param resPath
 */
public static void setResPath(String resPath)
```

```
/**
 * 从 apk 的 assets 解压资源文件到指定路径，需要先设置路径：{@link #setResPath(String)}
<br>
 * 首次安装 App，或 App 升级后调用一次即可.
 * copy xmagic resource from assets to local path
 */
public static boolean copyRes(Context context)

/**
 * 进行美颜授权检验
 * 注意：在使用此方法时，如果不设置回调接口，将不进行鉴权（只会从网络下载鉴权信息），
 * 所以可以参考demo  在application中调用时不设置回调，但是在使用TEBeautyKit对象之前再次调用
此方法（设置回调接口）鉴权
 * @param context                应用上下文
 * @param licenseKey             在平台申请的licenseKey
 * @param licenseUrl             在平台申请的licenseUrl
 * @param teLicenseCheckListener 鉴权回调接口
 */
public static void setTELicense(Context context, String licenseUrl, String
licenseKey, TELicenseCheck.TELicenseCheckListener teLicenseCheckListener)
```

## TEUIConfig说明

- **修改面板颜色**
  - 全局修改：通过 `TEUIConfig.getInstance()` 获取到对象之后可以通过修改如下字段来调整面板颜色
  - 局部修改：通过 `new TEUIConfig` 对象，然后修改如下字段来调整面板颜色，使用 `TEPanelView.updateUIConfig` 方法更新面板样式。

```
@ColorInt
public int panelBackgroundColor = 0x66000000;  //默认背景色
@ColorInt
public int panelDividerColor = 0x19FFFFFF;  //分割线颜色
@ColorInt
public int panelItemCheckedColor = 0xFF006EFF;  //选中项颜色
@ColorInt
public int textColor = 0x99FFFFFF; //文本颜色
@ColorInt
public int textCheckedColor = 0xFFFFFFFF; //文本选中颜色
@ColorInt
public int seekBarProgressColor = 0xFF006EFF; //进度条颜色
```

- **配置面板的 JSON 文件**

```
/**
 * 设置美颜面板的JSON文件路径
 * @param beauty 美颜属性的JSON文件路径，如果没有则设置为null
 * @param beautyBody 美体属性JSON文件路径，如果没有则设置为null
```

```
 * @param lut 滤镜属性JSON文件路径，如果没有则设置为null
 * @param motion 动效贴纸属性JSON文件路径，如果没有则设置为null
 * @param makeup 风格整妆属性JSON文件路径，如果没有则设置为null
 * @param segmentation 分割属性JSON文件路径，如果没有则设置为null
 */
public void setTEPanelViewRes(String beauty, String beautyBody, String lut,
String motion, String makeup, String segmentation)
```

- **更新面板语言**

```
//当客户程序监听到系统字体修改后，可以调用此方法，目前面板只支持中文和英文。
public void setSystemLocal(Locale locale)
```

## TEPanelView 说明

```
/**
 * 用于设置美颜上次效果的数据，目的是将美颜面板还原到上次的状态，
 * 注意：此方法需要在 {@link #showView(TEPanelViewCallback tePanelViewCallback)}} 方法
之前使用
 *
 * @param lastParamList 美颜数据,可以通过 {@link TEBeautyKit#exportInUseSDKParam()}
()}}方法获取，然后存储，在下次启动美颜的时候传入此字符串即可
 */
void setLastParamList(String lastParamList);

/**
 * 展示美颜面板
 * @param tePanelViewCallback 面板事件回调接口
 */
void showView(TEPanelViewCallback tePanelViewCallback);


/**
 * 绑定TEBeautyKit对象，当用户点击item的时候，面板会直接调用TEBeautyKit的方法进行属性设置
 * @param beautyKit TEBeautyKit对象
 */
void setupWithTEBeautyKit(TEBeautyKit beautyKit);

/**
 * 设置选中 自定义分割或者绿幕的item ，因为绿幕或者自定义分割按钮在点击之后是跳转到相册，
 * 只有用户选择了图片或者视频之后才会选中，如果在这个过程中用户取消了操作就不能选中对应的item
 *
 * @param uiProperty
 */
void checkPanelViewItem(TEUIProperty uiProperty);
```

```
/**
 * 设置当前面板的UI配置
 * @param uiConfig
 */
void updateUIConfig(TEUIConfig uiConfig);
```

# iOS

最近更新时间：2024-06-25 10:40:21

## 功能说明

TEBeautyKit 是腾讯特效美颜模块的 UI 面板库，用于客户快速方便的使用和管理美颜功能，效果如下图：

## 集成步骤

1. 下载并解压 TEBeautyKit 。
2. 把 TEBeautyKit 文件夹拷贝到自己的工程中，和 podfile 同级目录。
3. 编辑 podfile 文件，添加下面的代码：

```
pod 'TEBeautyKit',:path => 'TEBeautyKit/TEBeautyKit.podspec'
```

4. 执行 `pod install` 。

## 使用指引

1. 美颜鉴权

app启动以后，需要进行一次美颜鉴权，才能正常使用美颜功能。
接口：

```
#import "TEBeautyKit.h"
```

```
+ (void)setTELicense:(NSString *)url key:(NSString *)key completion:(callback
_Nullable )completion;
```

**示例：**

```
[TEBeautyKit setTELicense:@"your license" key:@"your key" completion:^(NSInteger
authresult, NSString * _Nullable errorMsg) {
  NSLog(@"----------result: %zd  %@",authresult,errorMsg);
}];
```

### 2. 配置美颜素材路径

**美颜面板上面的美颜数据都是从** `TEBeautyKit/Assets/json/beauty_panel/` **目录下，根据不同美颜套餐中对应的 json 文件中解析出来。**

**接口：**

```
#import "TEUIConfig.h"
/// 根据美颜套餐设置美颜面板的数据
/// - Parameter panelLevel: 美颜套餐
/// 美颜套餐类型：
A1_00,A1_01,A1_02,A1_03,A1_04,A1_05,A1_06,S1_00,S1_01,S1_02,S1_03,S1_04,S1_05,S1_0
6,S1_07
-(void)setPanelLevel:(TEPanelLevel)panelLevel;
```

**示例：**

```
//如果
[[TEUIConfig shareInstance] setPanelLevel:S1_07];
```

### 3. 初始化并添加 TEPanelView

```
-(TEPanelView *)tePanelView{
    if (!_tePanelView) {
        _tePanelView = [[TEPanelView alloc] init:nil comboType:nil];
        _tePanelView.delegate = self;
    }
    return _tePanelView;
}
[self.view addSubview:self.tePanelView];
[self.tePanelView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.width.mas_equalTo(self.view);
    make.centerX.mas_equalTo(self.view);
    make.height.mas_equalTo(200);
    make.bottom.mas_equalTo(self.view.mas_bottom);
}];
```

### 4. 创建美颜对象

**接口：**

```
//创建TEBeautyKit对象,不开启高性能模式
+ (void)create:(OnInitListener _Nullable )onInitListener;
//创建TEBeautyKit对象，isEnableHighPerformance：是否开启高性能模式
+ (void)create:(BOOL)isEnableHighPerformance onInitListener:(OnInitListener
_Nullable )onInitListener;
```

**示例：**

```
-(void)initXMagic{
    __weak __typeof(self)weakSelf = self;
    [TEBeautyKit create:^(XMagic * _Nullable api) {
        __strong typeof(self) strongSelf = weakSelf;
        strongSelf.xMagicKit = api;
        [strongSelf.teBeautyKit setXMagicApi:api];
        strongSelf.tePanelView.teBeautyKit = strongSelf.teBeautyKit;
        [strongSelf.teBeautyKit setTePanelView:strongSelf.tePanelView];
        [strongSelf.teBeautyKit setLogLevel:YT_SDK_ERROR_LEVEL];
        strongSelf.tePanelView.beautyKitApi = api;
        [strongSelf.xMagicKit registerSDKEventListener:strongSelf];
    }];
}
```

### 5. 处理视频数据

**接口：**

```
/**
textureId：纹理id
textureWidth:纹理宽度
textureHeight：纹理高度
origin：枚举值（YtLightImageOriginTopLeft、YtLightImageOriginBottomLeft），设置成
YtLightImageOriginBottomLeft 时，图像上下镜像翻转
orientation：枚举值：图像旋转角度
*/
- (YTProcessOutput *)processTexture:(int)textureId
         textureWidth:(int)textureWidth
        textureHeight:(int)textureHeight
           withOrigin:(YtLightImageOrigin)origin
      withOrientation:(YtLightDeviceCameraOrientation)orientation
```

**示例：**

```
#pragma mark - TRTCVideoFrameDelegate
- (uint32_t)onProcessVideoFrame:(TRTCVideoFrame *_Nonnull)srcFrame dstFrame:
(TRTCVideoFrame *_Nonnull)dstFrame {
    if(!_xMagicKit){
        [self initXMagic];
    }
    YTProcessOutput *output = [self.teBeautyKit processTexture:srcFrame.textureId
    textureWidth:srcFrame.width textureHeight:srcFrame.height
    withOrigin:YtLightImageOriginTopLeft withOrientation:YtLightCameraRotation0];
    dstFrame.textureId = output.textureData.texture;
    return 0;
}
```

6. 销毁美颜

```
- (void)destroyXMagic{
    [self.xMagicKit clearListeners];
    [self.xMagicKit deinit];
    self.xMagicKit = nil;
}
```

# 附录

## 面板 JSON 文件说明

- **美颜、美体 。**

```
ld.gradle (:tebeautykit)   beauty.json
{
    "displayName":"美颜",
    "displayNameEn":"Beauty effects",
    "propertyList":[
        {
            "displayName":"关闭",
            "displayNameEn":"None",
            "icon":"beauty_panel/panel_icon/beauty/none.png"
        },
        {
            "displayName":"美白",
            "displayNameEn":"Brighten",
            "icon":"beauty_panel/panel_icon/beauty/beauty_whiten.png",
            "propertyList":[...],
            "uiState":2
        },
        {
            "displayName":"磨皮",
            "displayNameEn":"Smoothskin",
            "icon":"beauty_panel/panel_icon/beauty/beauty_smooth.png",
            "sdkParam":{
                "effectName":"smooth.smooth",
                "effectValue":40
            },
            "uiState":1
        },
        {
            "displayName":"红润",
            "displayNameEn":"Rosyskin",
            "icon":"beauty_panel/panel_icon/beauty/beauty_ruddy.png",
            "sdkParam":{
                "effectName":"smooth.rosy",
                "effectValue":0
            }
        }
    ],
    "uiState":2
}
```

| 字段 | 说明 |
| --- | --- |
| displayName | 中文名称 |
| displayNameEn | 英文名称 |
| icon | 图片地址，支持设置本地图片和网络图片，本地图片支持 assets 资源和 SD 资源，assets 图片如上图所示，SD 卡图片设置图片全路径，网络图片设置对应的 http 链接 |
| sdkParam | 美颜 SDK 需要用到的属性，共包含四个属性，可参考美颜参数表 |
| effectName | 美颜属性 key，参考属性参数表 |
| effectValue | 设置属性强度，参考属性参数表 |
| resourcePath | 设置资源路径，参考属性参数表 |
| extraInfo | 设置其他信息，参考属性参数表 |

- **滤镜、动效贴纸、分割 。**

```
{
    "displayName":"滤镜",
    "displayNameEn":"Filters",
    "downloadPath":"light_material/lut/",
    "propertyList":[
        {
            "displayName":"无",
            "displayNameEn":"None",
            "icon":"beauty_panel/panel_icon/beauty/none.png"
        },
        {
            "displayName":"自然",
            "displayNameEn":"Natural",
            "icon":"beauty_panel/panel_icon/lut_icon/ziran_lf.png",
            "resourceUri":"light_material/lut/ziran_lf.png",
            "sdkParam":{
                "effectValue":60
            }
        },
        {
            "displayName":"自然-2",
            "displayNameEn":"Natural-2",
            "icon":"beauty_panel/panel_icon/lut_icon/ziran2_lf.png",
            "resourceUri":"https://mediacloud-76607.gzc.vod.tencent-cloud.com/TencentEffect/demoMotion/encrypted_lut/lut/ziran2_lf.png",
            "sdkParam":{
                "effectValue":60
            }
        },
```

由于**滤镜和动效贴纸、分割**的配置基本一致，所以此处用滤镜的 JSON 进行说明，这里新增了 downloadPath 和 resourceUri 字段。

| 字段 | 说明 |
| --- | --- |
| downloadPath | 如果您的滤镜素材是网络下载，那么这里配置的是您素材下载后在本地的存放位置，这里是**相对路径**，全路径是 `TEDownloader.h` 中设置的 `basicPath` **+此处设置的路径** |
| resourceUri | 如果您的素材是需要通过网络下载的，那么这里配置网络地址，如上图第三个红框，如果您的滤镜素材在本地，则按照上图配置对应的本地地址。 |

- **风格整妆**

```
{
    "displayName":"风格整妆",
    "displayNameEn":"Makeup",
    "downloadPath":"MotionRes/makeupRes/",
    "propertyList":[
        {
            "displayName":"无",
            "displayNameEn":"None",
            "icon":"beauty_panel/panel_icon/beauty/none.png"
        },
        {
            "displayName":"微闪",
            "displayNameEn":"Glitter",
            "icon":"beauty_panel/panel_icon/motions_icon/video_makeup_weishan.png",
            "resourceUri":"https://mediacloud-76607.gzc.vod.tencent-cloud.com/TencentEffect/d
            "sdkParam":{
                "effectValue":80,
                "extraInfo":{
                    "makeupLutStrength":"60"
                }
            }
        },
```

在风格整妆中增加了 `extraInfo` 下的 `makeupLutStrength` 字段，此字段用于调节风格整妆素材中**滤镜的强度**（如果此风格整妆素材支持调节滤镜强度就进行配置），此字段可参考美颜参数表。

## TEBeautyKit 方法说明

```
//创建TEBeautyKit对象，不开启高性能模式
+ (void)create:(OnInitListener _Nullable )onInitListener;
/**
 创建TEBeautyKit对象
 isEnableHighPerformance：是否开启高性能模式
 高性能模式开启后，美颜占用的系统 CPU/GPU 资源更少，可减少手机的发热和卡顿现象，更适合低端机长时间用。
 注意：开启高性能模式后，以下美颜项将不可用：
  1. 眼部：眼宽、眼高、祛眼袋
  2. 眉毛：角度、距离、高度、长度、粗细、眉峰
  3. 嘴部：微笑唇
  4. 面部：瘦脸（自然，女神，英俊），收下颌，祛皱、祛法令纹。建议用“脸型”实现综合大眼瘦脸效果
 */
+ (void)create:(BOOL)isEnableHighPerformance onInitListener:(OnInitListener _Nullable )onInitListener;
//美颜鉴权
+ (void)setTELicense:(NSString *)url key:(NSString *)key completion:(callback _Nullable )completion;
//设置美颜对象
- (void)setXMagicApi:(XMagic *_Nullable)xmagicApi;
//设置美颜面板，用来实现tePanelView的delegate
- (void)setTePanelView:(id)tePanelView;
//美颜静音
- (void)setMute:(BOOL)isMute;
/**
 * 设置某个特性的开或关
 *
 * @param featureName 取值见 XmagicConstant.FeatureName
 * @param enable      true表示开启，false表示关闭
 */
- (void)setFeatureEnableDisable:(NSString *_Nullable)featureName enable:(BOOL)enable;
//处理图片美颜
- (UIImage *_Nullable)processUIImage:(UIImage *_Nullable)inputImage
                  imageWidth:(int)imageWidth
                 imageHeight:(int)imageHeight
                   needReset:(bool)needReset;
//处理texture
- (YTProcessOutput *_Nullable)processTexture:(int)textureId
          textureWidth:(int)textureWidth
         textureHeight:(int)textureHeight
            withOrigin:(YtLightImageOrigin)origin
       withOrientation:(YtLightDeviceCameraOrientation)orientation;
//处理CVPixelBufferRef
- (YTProcessOutput * _Nullable)processPixelData:(CVPixelBufferRef _Nullable )pixelData
```

```
                        pixelDataWidth:(int)pixelDataWidth
                       pixelDataHeight:(int)pixelDataHeight
                            withOrigin:(YtLightImageOrigin)origin
                       withOrientation:(YtLightDeviceCameraOrientation)orientation;
//设置美颜
- (void)setEffect:(TESDKParam *_Nullable)sdkParam;
//设置美颜list
- (void)setEffectList:(NSArray<TESDKParam *>*_Nullable)sdkParamList;
//是否开启美颜增强模式
- (void)enableEnhancedMode:(BOOL)enable;
//获取正在使用的美颜数据
- (NSString *_Nullable)exportInUseSDKParam;
//获取保存的美颜数据，下次进入美颜的时候，调用setEffectList，即可恢复相同的美颜效果
- (NSMutableArray<TESDKParam *> *_Nonnull)getInUseSDKParamList;
//恢复美颜
- (void)onResume;
//暂停美颜
- (void)onPause;
//销毁美颜
- (void)onDestroy;
//获取当前texture的图片
- (void)exportCurrentTexture:(void (^_Nullable)(UIImage * _Nullable
image))callback;
//设置log
- (void)setLogLevel:(YtSDKLoggerLevel)level;
//设置AIDataListener
- (void)setAIDataListener:(id<TEBeautyKitAIDataListener> _Nullable)listener;
//设置TipsListener
- (void)setTipsListener:(id<TEBeautyKitTipsListener> _Nullable)listener;
//保存设置的美颜数据
- (void)saveEffectParam:(TESDKParam *_Nonnull)sdkParam;
//删除某个保存的美颜数据
- (void)deleteEffectParam:(TESDKParam *_Nonnull)sdkParam;
//清空保存的美颜数据
- (void)clearEffectParam;
```

## TEUIConfig说明

```
//可在外部修改下列属性的颜色
//美颜面板背景色
@property(nonatomic,strong)UIColor *panelBackgroundColor;
//分割线颜色
@property(nonatomic,strong)UIColor *panelDividerColor;
//选中项颜色
@property(nonatomic,strong)UIColor *panelItemCheckedColor;
//文本颜色
@property(nonatomic,strong)UIColor *textColor;
//文本选中颜色
```

```
@property(nonatomic,strong)UIColor *textCheckedColor;
//进度条颜色
@property(nonatomic,strong)UIColor *seekBarProgressColor;

/// 根据美颜套餐设置美颜面板的数据，推荐使用
/// - Parameter panelLevel: 美颜套餐
/// 美颜套餐类型：
A1_00,A1_01,A1_02,A1_03,A1_04,A1_05,A1_06,S1_00,S1_01,S1_02,S1_03,S1_04,S1_05,S1_0
6,S1_07
-(void)setPanelLevel:(TEPanelLevel)panelLevel;
/**
 配置美颜面板数据
 beauty:美颜json路径
 beautyBody: 美体json路径
 lut: 滤镜json路径
 motion: 动效json路径
 makeup: 美妆json路径
 segmentation: 背景分割json路径
 */
-(void)setTEPanelViewRes:(NSString *)beauty
beautyBody:(NSString *)beautyBody
lut:(NSString *)lut
motion:(NSString *)motion
makeup:(NSString *)makeup
segmentation:(NSString *)segmentation;
```

## TEPanelView说明

```
//初始化美颜面板，abilityType和comboType都填nil即可
- (instancetype)init:(NSString *)abilityType comboType:(NSString *)comboType;

@protocol TEPanelViewDelegate <NSObject>
//设置了美颜以后回调
- (void)setEffect;
@end
```

# TRTC SDK 集成腾讯特效 iOS

最近更新时间：2024-10-11 17:41:41

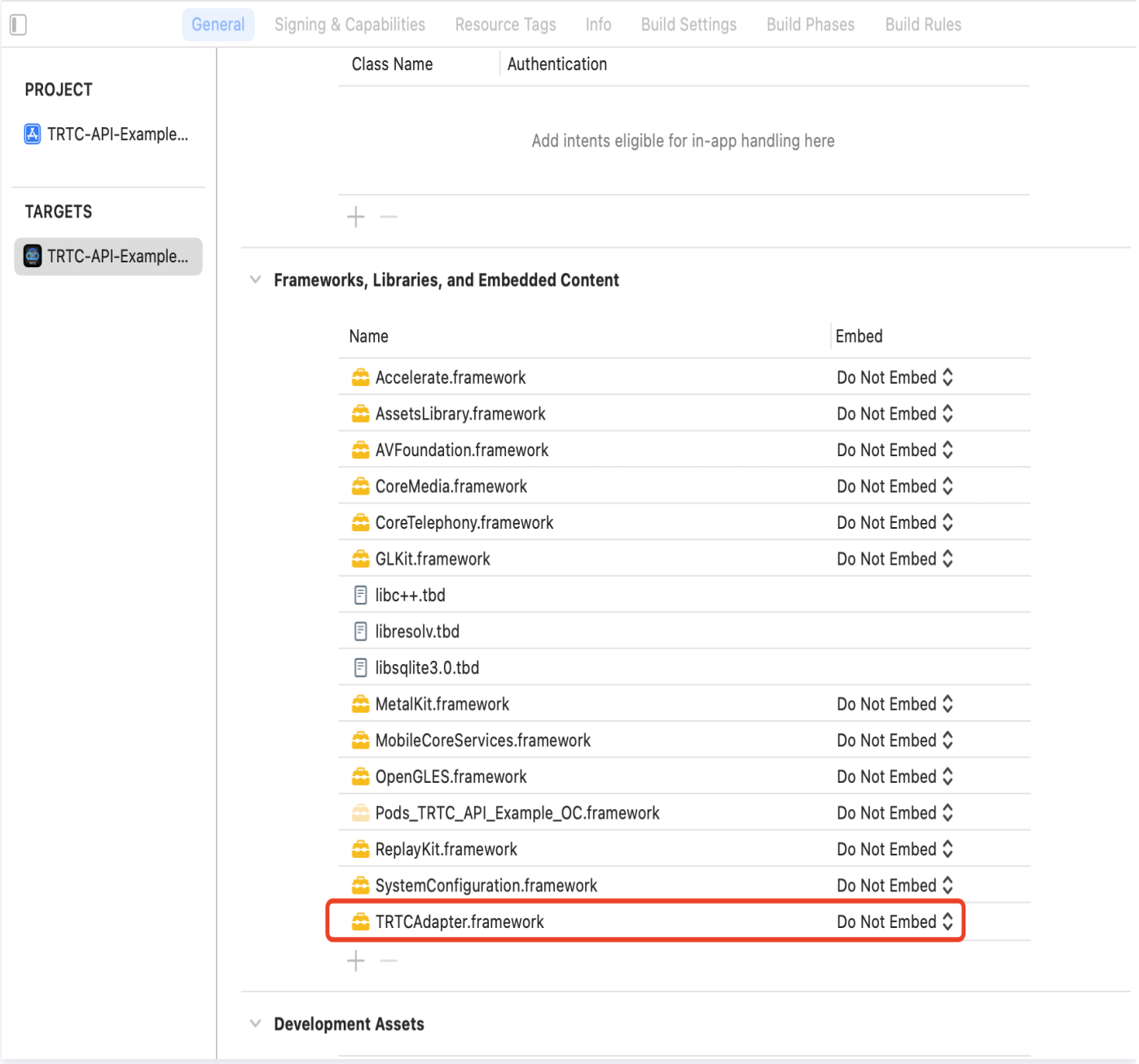
## SDK 集成

1. 本文档说明在 TRTC SDK 项目中集成和使用 TEBeautyKit 库。
2. 参见 demo(TRTC-XMagic-Adapter-Example) 。

## SDK 使用

### 步骤一：集成 TEBeautyKit

1. 下载并解压 TEBeautyKit 。
2. 下载并解压 TRTCAdapter 。
3. 把 TEBeautyKit 文件夹拷贝到自己的工程中，和 podfile 同级目录。
4. 把 `TRTCAdapter.framework` 添加到工程中。




5. 编辑 podfile 文件，添加下面的代码：

```
pod 'TEBeautyKit',:path => 'TEBeautyKit/TEBeautyKit.podspec'
```

6. 执行 `pod install` 。

## 步骤二：鉴权

```
[TEBeautyKit setTELicense:@"your license" key:@"your key" completion:^(NSInteger
authresult, NSString * _Nullable errorMsg) {
  NSLog(@"----------result: %zd  %@",authresult,errorMsg);
```

```
}];
```

## 步骤三：配置美颜素材路径

如果 json 文件中配置的素材是本地的，需要将美颜素材添加到工程中。

```
- (void)initBeautyJson{
  [[TEUIConfig shareInstance] setPanelLevel:S1_07];//根据美颜套餐选择
}
```

## 步骤四：初始化并添加 TEPanelView

```
-(TEPanelView *)tePanelView{
    if (!_tePanelView) {
        _tePanelView = [[TEPanelView alloc] init:nil comboType:nil];
        _tePanelView.delegate = self;
    }
    return _tePanelView;
}
[self.view addSubview:self.tePanelView];
[self.tePanelView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.width.mas_equalTo(self.view);
    make.centerX.mas_equalTo(self.view);
    make.height.mas_equalTo(220);
    make.bottom.mas_equalTo(self.view.mas_bottom);
}];
```

## 步骤五：adapter 绑定美颜

```
-(TEBeautyTRTCAdapter *)trtcAdapter{
  if (!_trtcAdapter) {
      _trtcAdapter = [[TEBeautyTRTCAdapter alloc] init];
  }
  return  _trtcAdapter;
}
__weak __typeof(self)weakSelf = self;
[self.trtcAdapter bind:self.trtcCloud  onCreatedTEBeautyKit:^(TEBeautyKit *
_Nullable beautyKit, XMagic * _Nullable xmagicApi) {
    __strong typeof(self) strongSelf = weakSelf;
    strongSelf.xMagicKit = xmagicApi;
    [strongSelf.teBeautyKit setXMagicApi:xmagicApi];
    strongSelf.tePanelView.teBeautyKit = strongSelf.teBeautyKit;
    [strongSelf.teBeautyKit setTePanelView:strongSelf.tePanelView];
    [strongSelf.teBeautyKit setLogLevel:YT_SDK_ERROR_LEVEL];
    strongSelf.tePanelView.beautyKitApi = xmagicApi;
} onDestroyTEBeautyKit:^{
```

```
}];
```

## 步骤六：参数变化通知 adapter

```
//通知adapter前后置摄像头，是否镜像
[self.trtcAdapter notifyCameraChanged:isFrontCamera mirrorType:mirrorType];
//通知adapter屏幕方向改变
[self.trtcAdapter setDeviceOrientation:orientation];
```

## 步骤七：解绑 adapter，销毁美颜

```
 [self.trtcAdapter unbind];
 self.trtcAdapter = nil;
```

# Android

最近更新时间：2024-11-21 17:24:12

## SDK 集成

1. 集成腾讯特效
   - 集成 SDK
   - 集成素材
2. 集成TEBeautyKit
   - 添加 TEBeautyKit 依赖
   - 添加面板 JSON 文件（如果不使用默认面板可以不添加）
3. 集成 `te_adapter_trtc`

### Maven依赖

在 dependencies 中添加 `te_adapter_trtc` 库的依赖

```
dependencies{
    ...
    implementation 'com.tencent.mediacloud:te_adapter_trtc:版本号'
}
```

### aar依赖

- 下载 aar 文件(下载的是一个 zip 文件，解压即可得到 aar 文件)。
- 添加下载的 te_adapter_trtc_xxxx `.aar` 文件到 app 工程 `libs` 目录下。
- 打开 app 模块的 `build.gradle` 添加依赖引用：

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

demo 工程请参考 TRTC demo 。

> **注意：**
> - 运行代码需要在 `com.tencent.thirdbeauty.xmagic.LicenseConstant.java` 文件中添加您申请的 `License` 信息，并将 `App module` 下的 `build.gradle` 中的 `applicationId` 修改为您申请 `License` 时填

写的包名。

- 由于依赖TRTC能力，所以需要在 `Debug module` 下找到 `com.tencent.trtc.debug.GenerateTestUserSig.java` 文件，并添加对应的 `APPID，SDKAPPID，SECRETKEY` 。
- 美颜集成的主要代码参见 `com.tencent.trtc.thirdbeauty.ThirdBeautyActivity.java` 文件。

# SDK 使用

## 第一步：设置面板 JSON 文件

请添加您在 TEBeautyKit集成中的第三步 中添加到您工程中的 JSON 文件的路径，没有的 JSON 文件则将路径设置为 null。

```
    TEPanelViewResModel resModel = new TEPanelViewResModel();
    String combo = "S1_07";    //根据您的套餐进行设置，如果您的套餐没有包含某项功能，客户设置
为null
    resModel.beauty = "beauty_panel/"+combo+"/beauty.json";
    resModel.lut = "beauty_panel/"+combo+"/lut.json";
    resModel.beautyBody = "beauty_panel/"+combo+"/beauty_body.json";
    resModel.motion = "beauty_panel/"+combo+"/motions.json";
    resModel.lightMakeup = "beauty_panel/"+combo+"/light_makeup.json";
    resModel.segmentation = "beauty_panel/"+combo+"/segmentation.json";
    TEUIConfig.getInstance().setTEPanelViewRes(resModel);
```

**注意：如果您不使用提供的美颜面板，请忽略这步操作。**

## 第二步：鉴权和复制资源

```
TEBeautyKit.setupSDK(this.getApplicationContext(),LicenseConstant.mXMagicLicenceUr
l,LicenseConstant.mXMagicKey, (i, s) -> {
    if (i == LicenseConstant.AUTH_STATE_SUCCEED) {
        runOnUiThread(() -> {
            Intent intent = new Intent(MainActivity.this,
ThirdBeautyActivity.class);
            startActivity(intent);
        }
    } else {
        Log.e(TAG, "te license check is failed,please checke  ");
    }
});
```

**注意：**
资源复制是根据SDK的版本号进行的，所以同一个版本号的SDK只会成功复制一次资源。

## 第三步：初始化 adapter 和添加面板

```
this.beautyAdapter = new TEBeautyTRTCAdapter();
//设置手机朝向
this.beautyAdapter.notifyScreenOrientationChanged(ActivityInfo.SCREEN_ORIENTATION_
PORTRAIT);
//设置相机是前置摄像头还是后置摄像头，以及是否编码镜像
this.beautyAdapter.notifyCameraChanged(isFront, this.isEncoderMirror);
//初始化面板代码
private void initBeautyPanelView() {
    RelativeLayout panelLayout = findViewById(R.id.live_pusher_bp_beauty_panel);
    this.tePanelView = new TEPanelView(this);
    if (lastParamList != null) {  //用于恢复美颜上次效果
        this.tePanelView.setLastParamList(lastParamList);
    }
    this.tePanelView.showView(this);
    panelLayout.addView(this.tePanelView);
}
```

**注意：**如果您不使用提供的面板，可以不创建 `TEPanelView` ，美颜属性调用 `TEBeautyKit` 的 `setEffect` 设置美颜属性。

## 第四步：绑定美颜

```
this.beautyAdapter.bind(this, this.mTRTCCloud, new ITEBeautyAdapter.CallBack() {
    @Override
    public void onCreatedTEBeautyKit(TEBeautyKit beautyKit) {
        mBeautyKit = beautyKit;
        //将mBeautyKit和美颜面板进行绑定
        tePanelView.setupWithTEBeautyKit(mBeautyKit);
        setTipListener(mBeautyKit);
        setLastParam(mBeautyKit);
        Log.e("beautyLiveAdapter", "onCreatedTEBeautyKit");
    }

    @Override
    public void onDestroyTEBeautyKit() {
        mBeautyKit = null;
        Log.e("beautyLiveAdapter", "onDestroyTEBeautyKit");
    }
});
```

## 第五步：参数变化通知 adapter

```
//当相机或画面镜像变化时需要调用notifyCameraChanged告诉adapter 最新的状态
 this.beautyAdapter.notifyCameraChanged(isFront, this.isEncoderMirror);

//当屏幕方向变化的时候 需要调用 notifyScreenOrientationChange方法
```

```
this.beautyAdapter.notifyScreenOrientationChanged(orientation);
```

### 第六步：销毁美颜

```
//当不再需要美颜时可以调用unbind方法解除绑定关系
this.beautyAdapter.unbind();
```

### 第七步：恢复声音

```
/**
 * 用于恢复贴纸中的声音
 * 恢复陀螺仪传感器，一般在Activity的onResume方法中调用
 */
this.mBeautyKit.onResume()
```

### 第八步：暂停声音

```
/**
 * 用于暂停贴纸中的声音
 * 暂停陀螺仪传感器，一般在Activity的onPause方法中调用
 */
this.mBeautyKit.onPause()
```

# Flutter

最近更新时间：2024-11-28 09:30:32

美颜 Flutter 版本 SDK 需要依赖 Android/iOS 端的美颜 SDK。通过 Flutter 提供的 Plugin 方式，可以将原生端的功能暴露给 Flutter 端。因此，在集成美颜功能时，需要手动集成原生端的 SDK。

## 跑通 Demo：

下载 Demo 工程，修改 `demo/lib` 下的 `main.dart` 文件，在此文件中添加您的 `licenseUrl` 和 `licenseKey`。TRTC 中使用美颜的示例代码主要在 `demo/lib/page/trtc_page.dart` 和 `demo/lib/main.dart` 中。

### Android

1. 在 demo/app下找到 build.gradle 文件，打开文件，将 `applicationId` 的值修改为您申请license信息时填写的包名。
2. 在 demo/lib 下执行 `flutter pub get` 。
3. 使用 Android studio 打开 demo 工程并运行。

### iOS

1. 在 demo 下执行 `flutter pub get` 。
2. 在 demo/ios 目录下执行 `pod install` 。
3. 使用 Xcode 工具打开 Runner.xcworkspace。
4. 修改工程的 `bundle ID` （需要和申请 license 时填写的 bundle ID 一致）。

## SDK 集成：

### 原生端集成：

### Android

1. 在 app 模块下找到 build.gradle 文件，添加您对应套餐的 maven 引用地址，例如您选择的是S1-04套餐，则添加如下：

```
dependencies {
   implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

各套餐对应的 maven 地址，请参见 文档，SDK的最新版本号可以在 版本历史 中查看。

2. 在您工程下的 `android/app` 模块下找到 `src/main/assets` 文件夹，将demo工程中 `demo/android/app/src/main/assets` 中的 `lut` 和 `MotionRes` 复制到您工程的 `android/app/ src/main/assets` 中，如果您的工程没有assets文件夹可以手动创建一个。
3. **如果您使用的Android版美颜SDK小于3.9版本**，您需要在 app 模块下找到 AndroidManifest.xml 文件，在 application 表填内添加如下标签：

```
    <uses-native-library
            android:name="libOpenCL.so"
            android:required="false" />
        //true 表示libOpenCL是当前app必需的。如果没有此库，系统将不允许app安装
        //false 表示libOpenCL不是当前app必需的。无论有没有此库，都可以正常安装app。如果设备有此库，腾讯特效SDK里的GAN类型特效能正常生效（例如童话脸、国漫脸）。如果设备没有此库，GAN类型不会生效，但也不影响SDK内其他功能的使用。
        //关于uses-native-library的说明，请参考Android 官网介绍：https://developer.android.com/guide/topics/manifest/uses-native-library-element
```

添加后如下图：

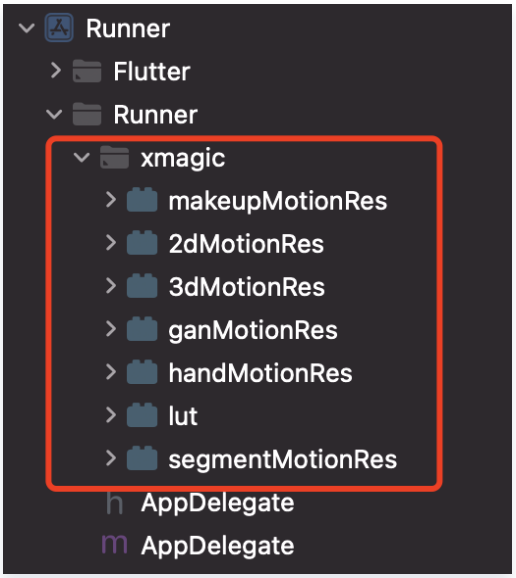
```
<application>
    <uses-native-library
        android:name="libOpenCL.so"
        android:required="false" />
</application>
```

4. 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.** { *;}
```

## iOS

1. 将demo工程中ios/Runner目录下的xmagic文件夹复制到您工程中ios/Runner目录下，添加后如下图：




> **注意：**
> 上述从 demo 工程中复制的素材是**测试素材**，正式素材需要您在购买套餐之后联系我们 工作人员 进行获取并重新添加。

## Flutter 端集成：

### 方式1：

远程依赖，在您工程的 pubspec.yaml 文件中添加如下引用：

```
tencent_effect_flutter:
 git:
   url: https://github.com/Tencent-RTC/TencentEffect_Flutter
```

### 方式2：

本地依赖，从 tencent_effect_flutter 下载最新版本的 tencent_effect_flutter，然后把文件夹 android、ios、lib 和文件 pubspec.yaml、tencent_effect_flutter.iml添加到工程目录下，然后在工程的 pubspec.yaml 文件中添加如下引用：（可参考 demo）

```
tencent_effect_flutter:
    path: ../
```

执行如下命令

```
flutter pub get
```

> **注意：**

tencent_effect_flutter 只是提供一个桥接，美颜功能依赖各个平台提供的美颜SDK，所以如果需要更新美颜SDK，您可以通过以下步骤进行 SDK 升级：

Android

在您的工程下的 app 模块下找到 build.gradle 文件，将 `implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'` 中的 `latest.release` 字段修改为最新的版本号。
如果您需要更换套餐，那么需要将 `TencentEffect_S1-04` 字段修改为您的套餐字段 。

iOS

**需要更换套餐类型和SDK版本：**

1. 美颜Flutter SDK的依赖方式需要修改为本地依赖
2. 在美颜Flutter SDK下找到 `ios/tencent_effect_flutter.podspec` 文件，打开找到 `TencentEffect_All` 修改为您需要的套餐，后边的版本号也可以修改为您需要的版本号。
3. 在您工程的ios目录下执行 pod update 命令即可

## SDK 使用：

### 1. 与 TRTC 关联

Android

在应用的 application 类的 oncreate 方法（或 FlutterActivity 的 onCreate 方法）中添加如下代码：

```
TRTCCloudPlugin.register(new XmagicProcesserFactory());
```

iOS

在应用的 AppDelegate 类中的 didFinishLaunchingWithOptions 方法里面中添加如下代码：

```
XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
[TencentTRTCCloud registerWithCustomBeautyProcesserFactory:instance];
```

添加后如下图：

```
#import "AppDelegate.h"
#import "GeneratedPluginRegistrant.h"
@import live_flutter_plugin;
@import tencent_effect_flutter;
@import tencent_trtc_cloud;

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  [GeneratedPluginRegistrant registerWithRegistry:self];
  // Override point for customization after application launch.
  XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
  [TXLivePluginManager registerWithCustomBeautyProcesserFactory:instance];
  [TencentTRTCCloud registerWithCustomBeautyProcesserFactory:instance];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}

@end
```

## 2. 调用资源初始化接口

v0.3.5.0及之后

```
void _initSettings(InitXmagicCallBack callBack) async {
  _setResourcePath();
  /// 复制资源只需要复制一次，在当前版本中如果成功复制了一次，以后就不需要再复制资源。
  /// Copying the resource only needs to be done once. Once it has been
successfully copied in the current version, there is no need to copy it again
in future versions.
  if (await isCopiedRes()) {
    callBack.call(true);
    return;
  } else {
    _copyRes(callBack);
  }
}

void _setResourcePath() async {
  String resourceDir = await ResPathManager.getResManager().getResPath();
  TXLog.printlog(
      '$TAG method is _initResource ,xmagic resource dir is $resourceDir');
  TencentEffectApi.getApi()?.setResourcePath(resourceDir);
}
```

```
void _copyRes(InitXmagicCallBack callBack) {
  _showDialog(context);
  TencentEffectApi.getApi()?.initXmagic((result) {
    if (result) {
      saveResCopied();
    }
    _dismissDialog(context);
    callBack.call(result);
    if (!result) {
      Fluttertoast.showToast(msg: "initialization failed");
    }
  });
}
```

**v0.3.1.1版本及之前**

```
String dir =  await BeautyDataManager.getInstance().getResDir();
 TXLog.printlog('文件路径为: $dir');
 TencentEffectApi.getApi()?.initXmagic(dir,(reslut) {
     _isInitResource = reslut;
     callBack.call(reslut);
     if (!reslut) {
         Fluttertoast.showToast(msg: "初始化资源失败");
     }
 });
```

## 3. 进行美颜授权

```
TencentEffectApi.getApi()?.setLicense(licenseKey, licenseUrl,
           (errorCode, msg) {
         TXLog.printlog("打印鉴权结果 errorCode = $errorCode   msg = $msg");
         if (errorCode == 0) {
            //鉴权成功
         }
       });
```

## 4. 开启美颜

```
///开启美颜操作
 var enableCustomVideo = await trtcCloud.enableCustomVideoProcess(open);
```

## 5. 设置美颜属性

v0.3.5.0及之后

具体美颜属性请参考 美颜属性表

```
TencentEffectApi.getApi()?.setEffect(sdkParam.effectName!,
    sdkParam.effectValue, sdkParam.resourcePath, sdkParam.extraInfo)
```

v0.3.1.1版本及之前

```
TencentEffectApi.getApi()?.updateProperty(_xmagicProperty!);
///_xmagicProperty 可通过 BeautyDataManager.getInstance().getAllPannelData();获取所有的属性，需要使用美颜属性的时候可通过updateProperty方法设置属性。
```

## 6. 设置其他属性

- **暂停美颜音效**

```
 TencentEffectApi.getApi()?.onPause();
```

- **恢复美颜音效**

```
TencentEffectApi.getApi()?.onResume();
```

- **监听美颜事件**

```
TencentEffectApi.getApi()
     ?.setOnCreateXmagicApiErrorListener((errorMsg, code) {
       TXLog.printlog("创建美颜对象出现错误 errorMsg = $errorMsg , code = $code");
 });    ///需要在创建美颜之前进行设置
```

- **设置人脸、手势、身体检测状态回调**

```
TencentEffectApi.getApi()?.setAIDataListener(XmagicAIDataListenerImp());
```

- **设置动效提示语回调函数**

```
TencentEffectApi.getApi()?.setTipsListener(XmagicTipsListenerImp());
```

- **设置人脸点位信息等数据回调（S1-05 和 S1-06 套餐才会有回调）**

```
TencentEffectApi.getApi()?.setYTDataListener((data) {
   TXLog.printlog("setYTDataListener  $data");
 });
```

- **移除所有回调**。在页面销毁的时候需要移除掉所有的回调：

```
 TencentEffectApi.getApi()?.setOnCreateXmagicApiErrorListener(null);
 TencentEffectApi.getApi()?.setAIDataListener(null);
 TencentEffectApi.getApi()?.setYTDataListener(null);
 TencentEffectApi.getApi()?.setTipsListener(null);
```

> **说明：**
> 接口详细可参考 API文档，其他可参考 Demo 工程。

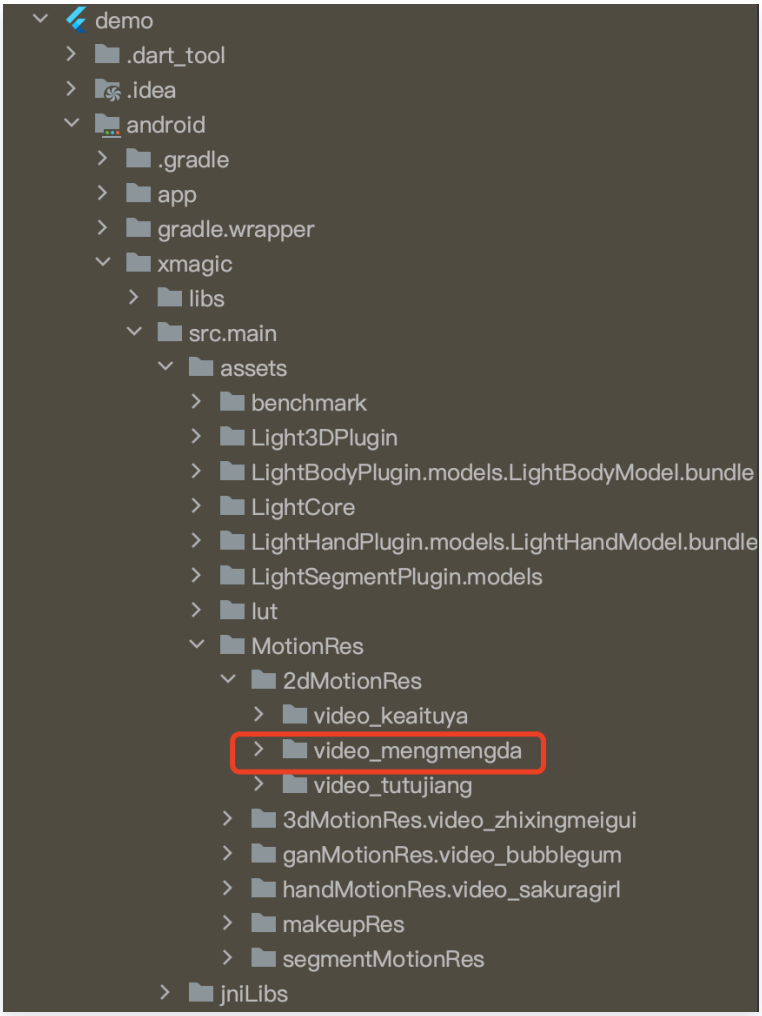
## 7. 添加和删除美颜面板上的美颜数据

### 添加美颜资源

把您的资源文件按照步骤一中的方法添加到对应的资源文件夹里面。例如您需要添加2D动效的资源：

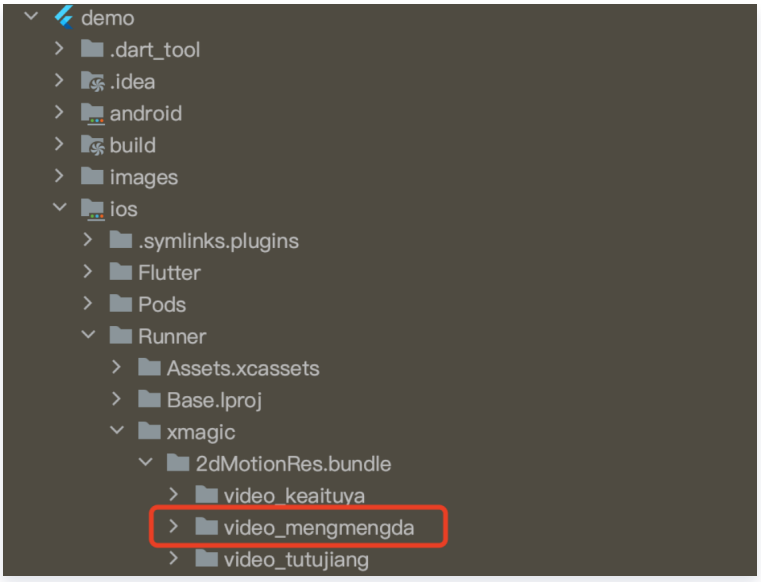
Android

1. 您应该把资源放在工程的 `android/xmagic/src.mian/assets/MotionRes/2dMotionRes` 目录下：




## iOS

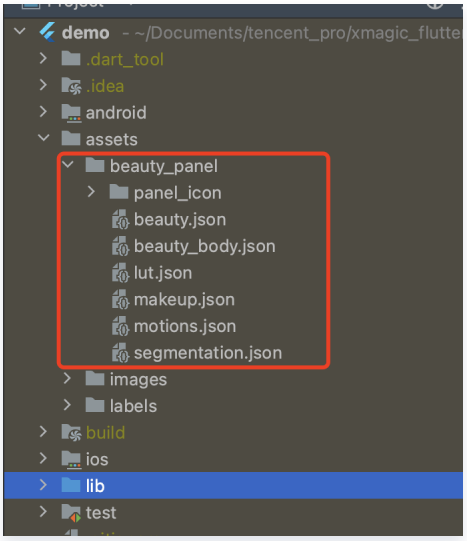
1. 并且把资源添加到工程的 `ios/Runner/xmagic/2dMotionRes.bundle` 目录下。



**美颜面板配置：**

V0.3.5.0及之后

在 demo 中提供了一个简单的美颜面板 UI，面板上的属性通过 JSON 文件进行配置，JSON 文件位置如下图。面板的实现可以参考 demo 工程。




JSON 文件说明

V0.3.1.1及之前

在 BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid 和 BeautyPropertyProducerIOS 这4个类中，您可以自主操作美颜面板数据的配置。

## 删除美颜资源

对于某些 License 没有授权美颜和美体的部分功能，美颜面板上不需要展示这部分功能，需要在美颜面板数据的配置中删除这部分功能的配置。
例如，删除口红特效：分别在 BeautyPropertyProducerAndroid 类和 BeautyPropertyProducerIOS 类中的 getBeautyData 方法中删除以下代码。

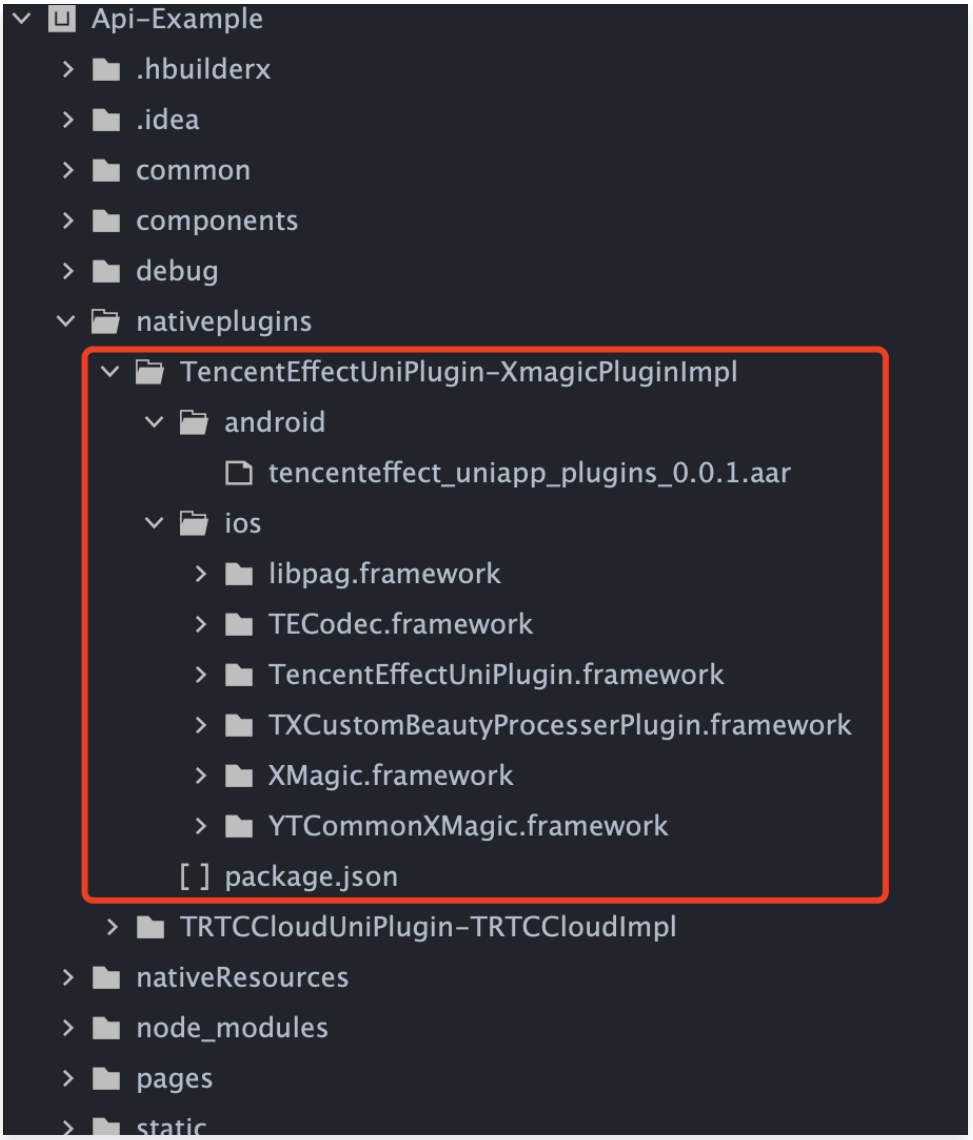
```
// Map<String, String> lipsResPathNames = {};
// lipsResPathNames["lips_fuguhong.png"] = "复古红";
// lipsResPathNames["lips_mitaose.png"] = "蜜桃色";
// lipsResPathNames["lips_shanhuju.png"] = "珊瑚橘";
// lipsResPathNames["lips_wenroufen.png"] = "温柔粉";
// lipsResPathNames["lips_huolicheng.png"] = "活力橙";
// List<XmagicUIProperty> itemLipsPropertys = [];
// String lipId = "beauty.lips.lipsMask";
// for (String ids in lipsResPathNames.keys) {
//   itemLipsPropertys.add(XmagicUIProperty(
//       uiCategory: Category.BEAUTY,
//       displayName: lipsResPathNames[ids]!,
//       id: lipId,
//       resPath: resPaths + ids,
//       thumbDrawableName: "beauty_lips",
//       effKey: BeautyConstant.BEAUTY_MOUTH_LIPSTICK,
//       effValue: XmagicPropertyValues(0, 100, 50, 0, 1),
//       rootDisplayName: "口红"));
// }
// XmagicUIProperty itemLips = XmagicUIProperty(
//     displayName: "口红",
//     thumbDrawableName: "beauty_lips",
//     uiCategory: Category.BEAUTY);
// itemLips.xmagicUIPropertyList = itemLipsPropertys;
// beautyList.add(itemLips);
```

# uni-app

最近更新时间：2024-08-08 17:55:41

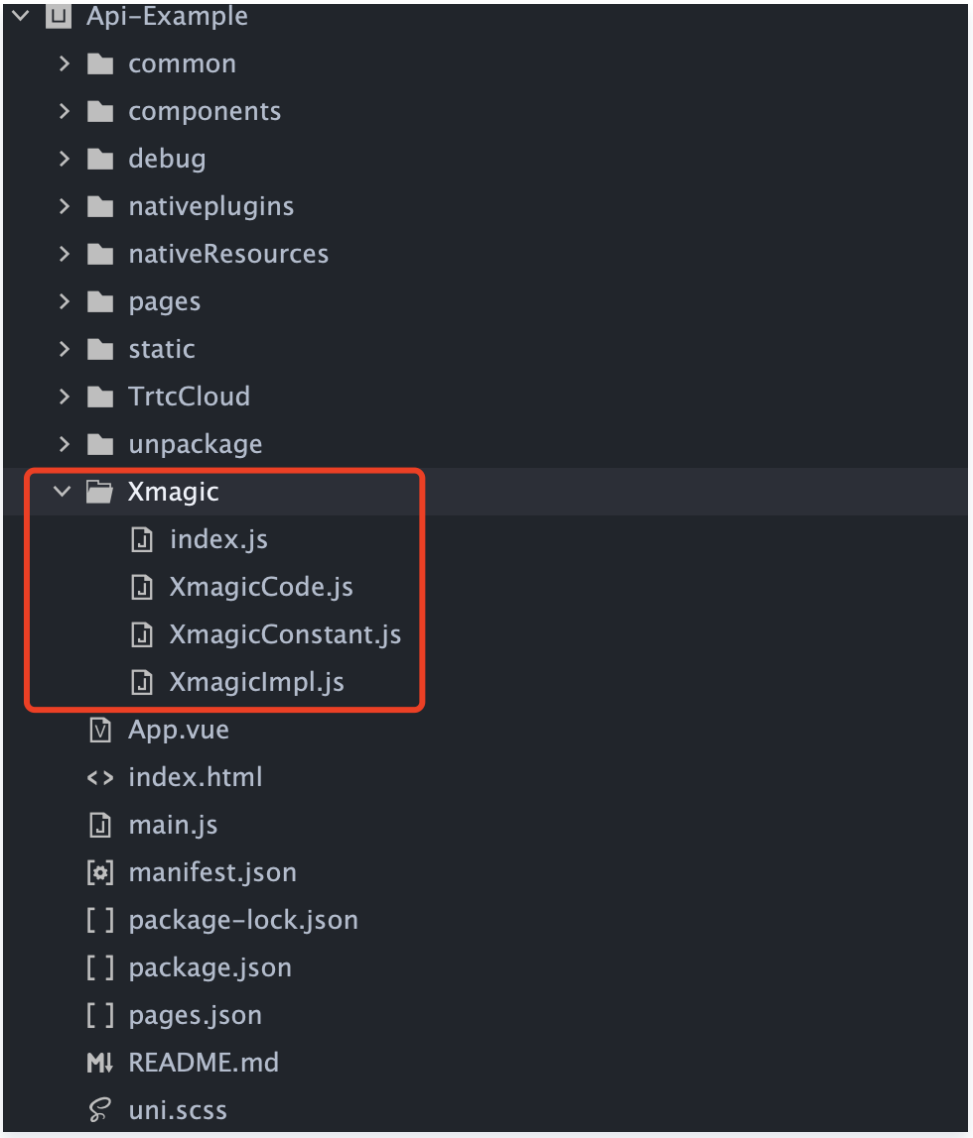
## 集成指引

1. 下载并解压 TencentEffect_UniApp 。
2. 找到 TencentEffect_UniApp/SDK/nativeplugins 目录下的 TencentEffectUniPlugin-XmagicPluginImpl 文件夹拷贝到自己项目工程的 nativeplugins 目录下。如下图所以位置：



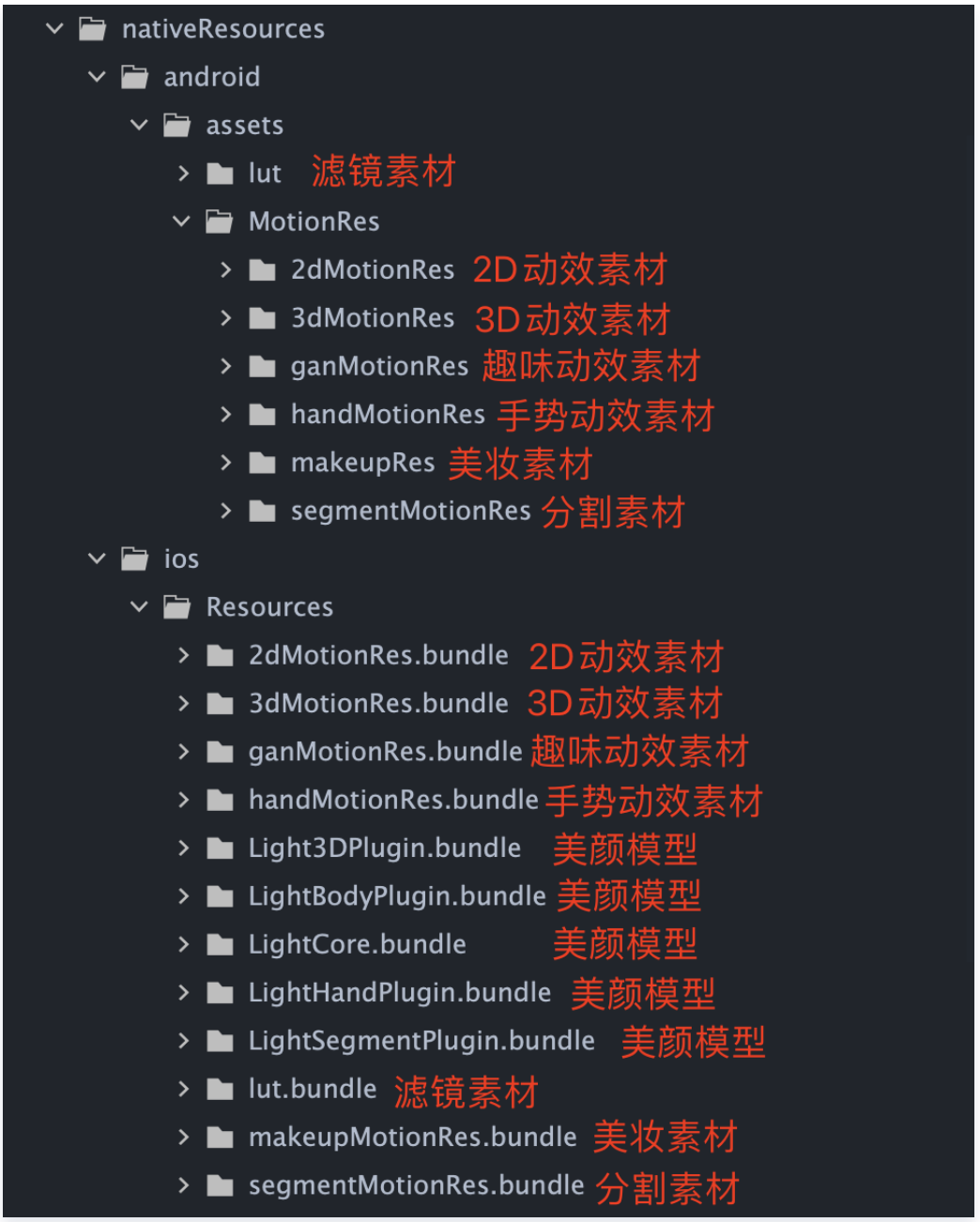

3. 在自己项目工程的 manifest.json 中配置原生插件，选择本地插件 TencentEffectUniPlugin-XmagicPluginImpl。

4. **把 SDK/js 目录下的 XMagic 文件夹拷贝到自己项目工程中。**




5. **添加美颜素材和模型：**

   **Android：**把在官网中下载的美颜素材拷贝到自己项目工程的 nativeResources/android/assets 目录下。
   **iOS：**把在官网中下载的美颜资源和美颜模型拷贝到自己项目工程的 nativeResources/ios/Resources 目录下。
   不同套餐，添加的美颜模型和美颜素材不一样。（详见 美颜官网 ）

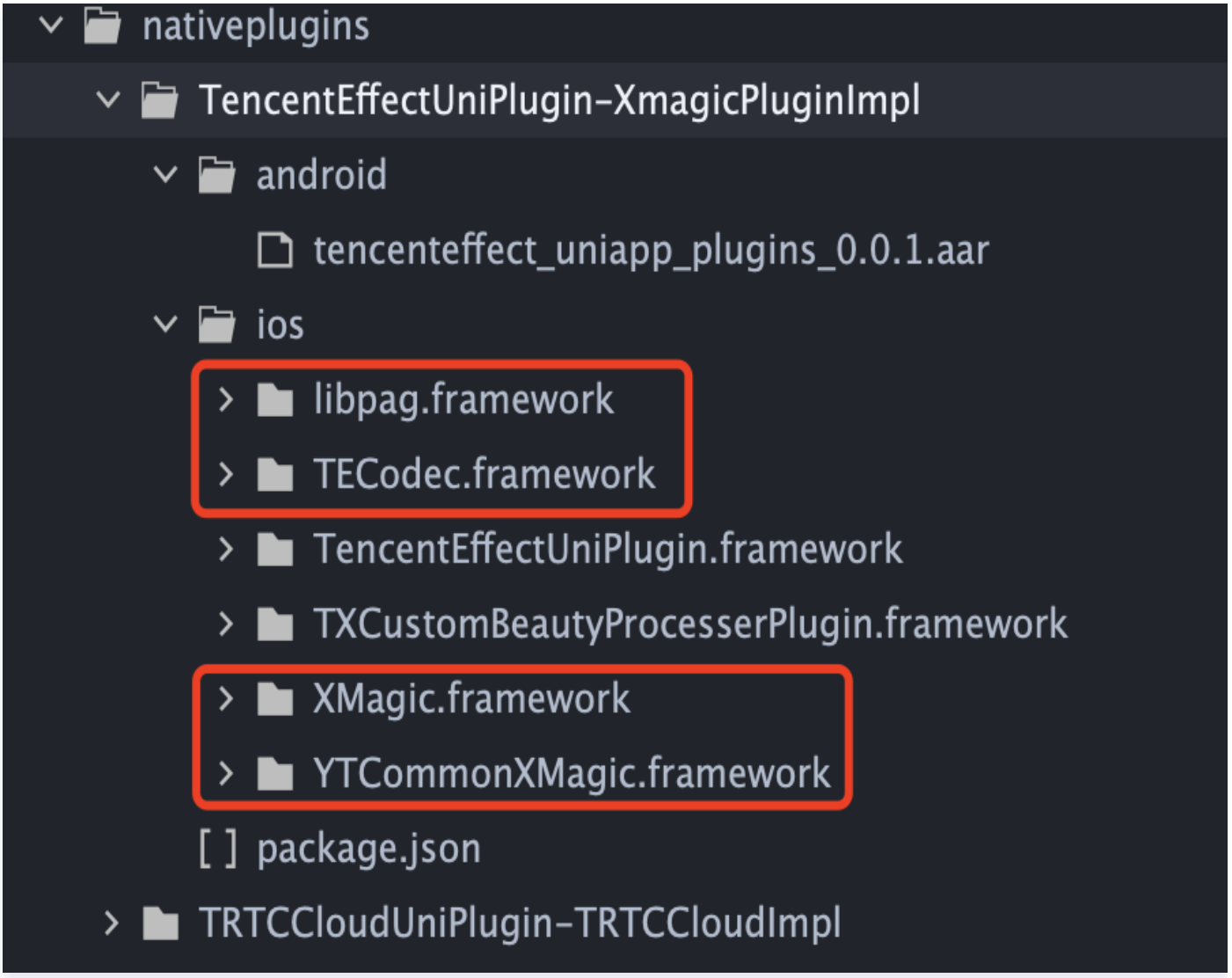



6. **添加腾讯特效 SDK**

   **Android：**通过 maven 依赖美颜特效 SDK，需要更改美颜套餐或者美颜版本，可手动修改 `nativeplugins/TencentEffectUniPlugin-XmagicPluginImpl/package.json` 中 `dependencies` 字段中腾讯特效 SDK 的 maven 地址。

```
    "android":{
        "plugins":[
            {
                "type":"module",
                "name":"TencentEffectUniPlugin-XmagicPluginImpl",
                "class":"com.tencent.effect.uniplugin.xmagic.XmagicPluginImp"
            }
        ],
        "integrateType":"aar",
        "validArchitectures":[
            "arm64",
            "armv7"
        ],
        "dependencies":[
            "com.tencent.liteav:custom-video-processor:latest.release",
            "com.tencent.mediacloud:TencentEffect_S1-04:3.3.0.2", //可选择美颜套餐和SDK版本
            "com.google.code.gson:gson:2.8.2",
            "androidx.exifinterface:exifinterface:1.3.3"
        ]
    }
```

**iOS：下载腾讯特效 SDK 后，把里面的 XMagic.framework、YTCommonXMagic.framework、TECodec.framework、libpag.framework 拷贝到** `nativeplugins/TencentEffectUniPlugin-XmagicPluginImpl/ios` **中。**




## 使用指引

**注：此美颜插件需要和 TRTC 插件结合使用,按照顺序调用对应方法。**

### 1. 复制美颜资源

接口：`copyXmaigcRes`
描述：复制美颜资源。
app 首次安装使用时，需要调用此接口把美颜资源复制到沙盒，**每个版本仅需要成功调用一次即可**。

### 2. 美颜鉴权

接口：`setLicense`
描述：使用 bundleId、licenseUrl、licenseKey 进行美颜鉴权。
app 启动以后，需要成功鉴权以后才能使用美颜功能。

### 3. 开启或关闭美颜

**接口：** `enableCustomVideoProcess`
**描述：调用此接口来开启或关闭美颜。**
**注意：关闭美颜需要在关闭TRTC 预览之前进行。**

## 4. 设置美颜属性

### V0.3.5.0及之后

**接口：** `setEffect`
**描述：设置美颜属性。**
**美颜属性包括美颜、美体、滤镜、动效（2D 动效、3D 动效、手势动效、趣味动效）、美妆、分割。设置不同的属性需要在调用此接口时传入不同的参数，**
**具体参见 美颜参数表 和 demo。**

### V0.3.3.0版本

**接口：** `updateProperty` **（已废弃）**
**描述：设置美颜属性。**
**美颜属性包括美颜、美体、滤镜、动效（2D 动效、3D 动效、手势动效、趣味动效）、美妆、分割。设置不同的属性需要在调用此接口时传入不同的参数，具体参数请参见下表和 demo。**

| 字段 | 美颜 | 滤镜 | 美体 | 动效贴纸 | 分割 | 美妆 |
| --- | --- | --- | --- | --- | --- | --- |
| category | Category.BEAUTY | Category.LUT | Category.BODY_BEAUTY | Category.MOTION | Category.SEGMENTATION | Category.MAKEUP |
| id | 参考 PropertyIds 类 | 滤镜资源名称 例如：baixi_lf.png | 无 | 贴纸资源的名称。例如：video_keaituya | 分割资源的名称 例如：video_segmentation_blur_45 | 美妆资源名称 例如：video_nvtuanzhuang |
| resPath | 参考 Android 参数表中的 单点妆容 中的 resPath 字段 | 滤镜资源的路径 例如：light_material/lut/baixi_lf.png | 无 | 贴纸资源名称。例如：MotionRes/2dMotionRes/video_keaituya | 分割资源路径。 例如：MotionRes/segmentMotionRes/video_segmentation_blur_45 | 美妆资源路径 例如：MotionRes/makeupRes/video_nvtuanzhuang |

| | | | | | | |
|---|---|---|---|---|---|---|
| effKey | 参考 EffectName 类 | 无 | 参考 EffectName 类 | 无 | 无 | 无 |
| effValue | 取值范围为0 – 1或者是–1 – 1<br>参考 Android 参数表，如果 displayMinValue 和 displayMaxValue 值为0 – 100，那么此处的范围是0 – 1，如果是–100 – 100，那么此处的值为–1 – 1 | 0 – 1 | 0 – 1 | 无 | 无 | 0 – 1 |

## 5. 暂停美颜

接口：`onPause`
描述：暂停美颜音频。

## 6. 恢复美颜

接口：`onResume`
描述：恢复美颜音频。

# 直播 SDK 集成腾讯特效
# iOS

最近更新时间：2024-10-21 15:21:52

## SDK 集成

1. 本文档说明在直播 SDK 项目中集成和使用 TEBeautyKit 库。
2. 参见 demo 。

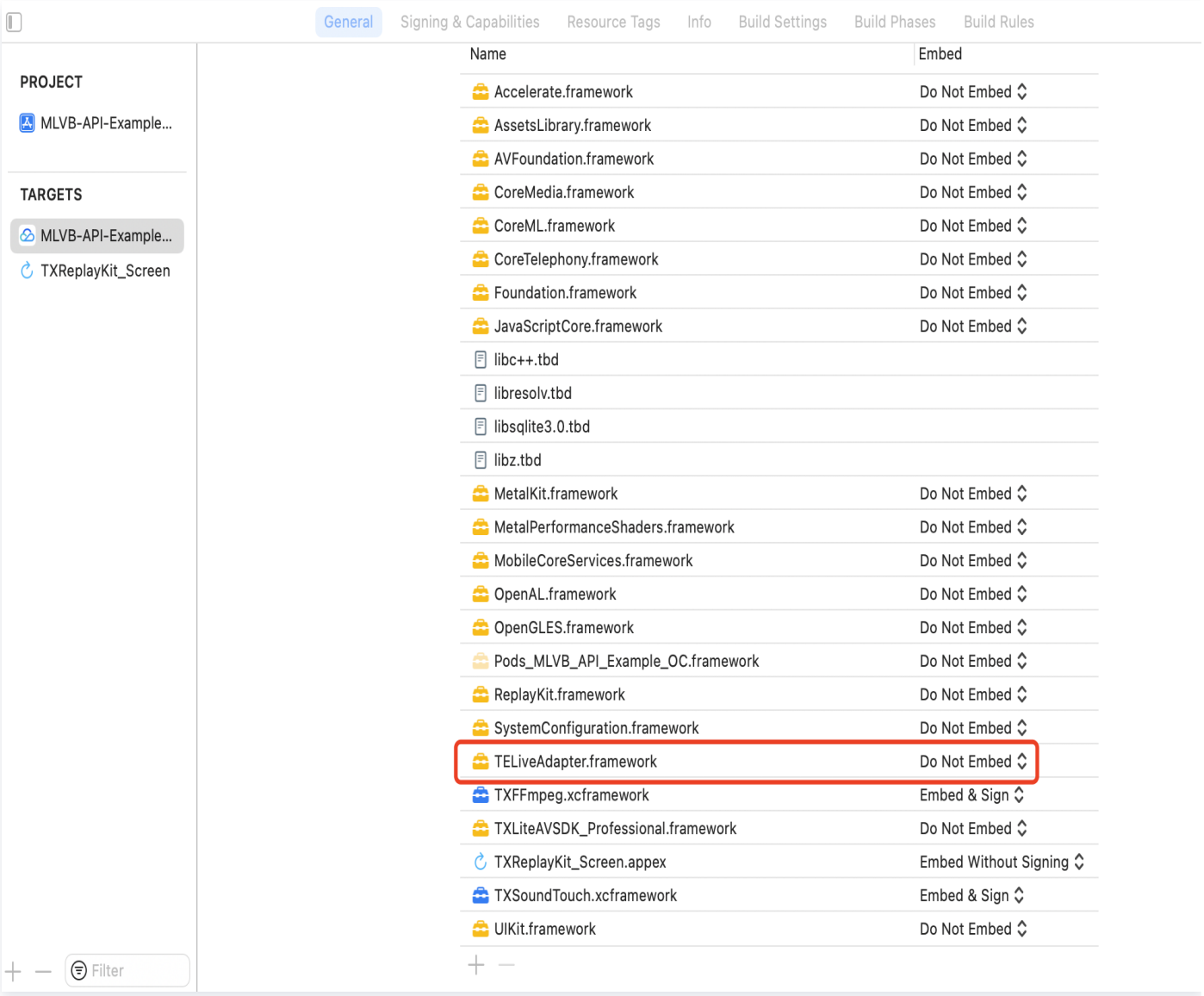
## SDK 使用

### 步骤一：集成 TEBeautyKit

1. 下载并解压 TEBeautyKit 。
2. 下载并解压 TELiveAdapter 。
3. 把 TEBeautyKit 文件夹拷贝到自己的工程中，和 podfile 同级目录。
4. 编辑 podfile 文件，添加下面的代码：

```
pod 'TEBeautyKit',:path => 'TEBeautyKit/TEBeautyKit.podspec'
```

5. 把 `TELiveAdapter.framework` 添加到工程中。




6. 执行 `pod install` 。

## 步骤二：鉴权

```
[TEBeautyKit setTELicense:@"your license" key:@"your key" completion:^(NSInteger authresult, NSString * _Nullable errorMsg) {
  NSLog(@"----------result: %zd  %@",authresult,errorMsg);
}];
```

## 步骤三：配置美颜素材路径

如果 json 文件中配置的素材是本地的，需要将美颜素材添加到工程中。

```
- (void)initBeautyJson{
  [[TEUIConfig shareInstance] setPanelLevel:S1_07];//根据美颜套餐选择
```

```
}
```

## 步骤四：初始化并添加 TEPanelView

```
-(TEPanelView *)tePanelView{
    if (!_tePanelView) {
        _tePanelView = [[TEPanelView alloc] init:nil comboType:nil];
        _tePanelView.delegate = self;
    }
    return _tePanelView;
}
[self.view addSubview:self.tePanelView];
[self.tePanelView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.width.mas_equalTo(self.view);
    make.centerX.mas_equalTo(self.view);
    make.height.mas_equalTo(220);
    make.bottom.mas_equalTo(self.view.mas_bottom);
}];
```

## 步骤五：adapter 绑定美颜

```
-(TEBeautyLiveAdapter *)liveAdapter{
    if (!_liveAdapter) {
        _liveAdapter = [[TEBeautyLiveAdapter alloc] init];
    }
    return _liveAdapter;
}
__weak __typeof(self)weakSelf = self;
[self.liveAdapter bind:self.livePusher  onCreatedTEBeautyKit:^(TEBeautyKit *
_Nullable beautyKit, XMagic * _Nullable xmagicApi) {
    __strong typeof(self) strongSelf = weakSelf;
    strongSelf.xMagicKit = xmagicApi;
    [strongSelf.teBeautyKit setXMagicApi:xmagicApi];
    strongSelf.tePanelView.teBeautyKit = strongSelf.teBeautyKit;
    [strongSelf.teBeautyKit setTePanelView:strongSelf.tePanelView];
    [strongSelf.teBeautyKit setLogLevel:YT_SDK_ERROR_LEVEL];
    strongSelf.tePanelView.beautyKitApi = xmagicApi;
    [strongSelf.liveAdapter setDeviceOrientation:UIDeviceOrientationPortrait];
    [strongSelf.xMagicKit registerSDKEventListener:strongSelf];
} onDestroyTEBeautyKit:^{
}];
```

## 步骤六：参数变化通知 adapter

```
//通知adapter前后置摄像头，是否镜像
[self.liveAdapter notifyCameraChanged:isFrontCamera mirrorType:mirrorType];
```

```
//通知adapter屏幕方向改变
[self.liveAdapter setDeviceOrientation:orientation];
```

## 步骤七：解绑 adapter

```
[self.liveAdapter unbind];
```

# Android

最近更新时间：2024-11-21 17:24:12

## SDK 集成

1. 集成腾讯特效
   - 集成 SDK
   - 集成素材
2. 集成TEBeautyKit
   - 添加 TEBeautyKit 依赖
   - 添加面板 JSON 文件（如果不使用默认面板可以不添加）
3. 集成 `te_adapter_live`

### Maven依赖

在 dependencies 中添加 `te_adapter_live` 库的依赖

```
dependencies{
    ...
    implementation 'com.tencent.mediacloud:te_adapter_live:版本号'
}
```

### aar依赖

- 下载 aar 文件（下载的是一个 zip 文件，解压即可得到 aar 文件）。
- 添加下载的 te_adapter_live_xxxx `.aar` 文件到 app 工程 `libs` 目录下。
- 打开 app 模块的 `build.gradle` 添加依赖引用：

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

demo 工程可参考 MLVB demo 。

> **注意：**
> - 运行代码需要在 `com.tencent.thirdbeauty.xmagic.LicenseConstant.java` 文件中添加您申请的 `License` 信息，并将 `App module` 下的 `build.gradle` 中的 `applicationId` 修改为您申请 `License` 时填

写的包名。

- 由于依赖 `MLVB` 能力，所以需要在 `Debug module` 下找到 `com.tencent.mlvb.debug.GenerateTestUserSig.java` 文件，并添加对应的 `LICENSEURL`，`LICENSEURLKEY`，`SDKAPPID`，`SECRETKEY`。
- 美颜集成的主要代码参见 `com.tencent.mlvb.thirdbeauty.ThirdBeautyActivity.java` 文件。

# SDK 使用

## 第一步：设置面板 JSON 文件

请添加您在 TEBeautyKit集成中的第三步 中添加到您工程中的 JSON 文件的路径，没有的 JSON 文件则将路径设置为 null。

```
    TEPanelViewResModel resModel = new TEPanelViewResModel();
    String combo = "S1_07";    //根据您的套餐进行设置，如果您的套餐没有包含某项功能，客户设置
为null
    resModel.beauty = "beauty_panel/"+combo+"/beauty.json";
    resModel.lut = "beauty_panel/"+combo+"/lut.json";
    resModel.beautyBody = "beauty_panel/"+combo+"/beauty_body.json";
    resModel.motion = "beauty_panel/"+combo+"/motions.json";
    resModel.lightMakeup = "beauty_panel/"+combo+"/light_makeup.json";
    resModel.segmentation = "beauty_panel/"+combo+"/segmentation.json";
    TEUIConfig.getInstance().setTEPanelViewRes(resModel);
```

**注意：如果您不使用提供的美颜面板，请忽略这步操作。**

## 第二步：鉴权和资源复制

```
TEBeautyKit.setupSDK(this.getApplicationContext(),LicenseConstant.mXMagicLicenceUr
l,LicenseConstant.mXMagicKey, (i, s) -> {
    if (i == LicenseConstant.AUTH_STATE_SUCCEED) {
        runOnUiThread(() -> {
            Intent intent = new Intent(MainActivity.this,
ThirdBeautyActivity.class);
            startActivity(intent);
        }
    } else {
        Log.e(TAG, "te license check is failed,please checke  ");
    }
});
```

**注意：**
资源复制是根据SDK的版本号进行的，所以同一个版本号的SDK只会成功复制一次资源。

## 第三步：初始化 adapter 和添加面板

```
this.beautyLiveAdapter = new TEBeautyLiveAdapter();
//设置手机朝向
this.beautyLiveAdapter.notifyScreenOrientationChanged(ActivityInfo.SCREEN_ORIENTAT
ION_PORTRAIT);
//设置相机是前置摄像头还是后置摄像头，以及是否编码镜像
this.beautyLiveAdapter.notifyCameraChanged(isFront, this.isEncoderMirror);


private void initBeautyPanelView() {
    RelativeLayout panelLayout = findViewById(R.id.live_pusher_bp_beauty_panel);
    this.tePanelView = new TEPanelView(this);
    if (lastParamList != null) {  //用于恢复美颜上次效果
        this.tePanelView.setLastParamList(lastParamList);
    }
    this.tePanelView.showView(this);
    panelLayout.addView(this.tePanelView);
}
```

**注意：如果您不想使用提供的面板，可以不创建 `TEPanelView` ，自己组织美颜属性，调用 `TEBeautyKit` 的 `setEffect` 设置美颜属性。**

## 第四步：绑定美颜

```
this.beautyLiveAdapter.bind(this, mLivePusher, new ITEBeautyAdapter.CallBack() {
    @Override
    public void onCreatedTEBeautyKit(TEBeautyKit beautyKit) {
        mBeautyKit = beautyKit;
        tePanelView.setupWithTEBeautyKit(mBeautyKit);
        setTipListener(mBeautyKit);
        setLastParam(mBeautyKit);
        Log.e("beautyLiveAdapter", "onCreatedTEBeautyKit");
    }

    @Override
    public void onDestroyTEBeautyKit() {
        mBeautyKit = null;
        Log.e("beautyLiveAdapter", "onDestroyTEBeautyKit");
    }
});
```

## 第五步：参数变化通知 adapter

```
//当相机或画面镜像变化时需要调用notifyCameraChanged告诉adapter 最新的状态
 this.beautyLiveAdapter.notifyCameraChanged(isFront, this.isEncoderMirror);
```

```
//当屏幕方向变化的时候 需要调用 notifyScreenOrientationChange方法
 this.beautyLiveAdapter.notifyScreenOrientationChanged(orientation);
```

### 第六步：销毁美颜

```
 //当不再需要美颜时可以调用unbind方法解除绑定关系
 this.beautyLiveAdapter.unbind();
```

### 第七步：恢复声音

```
/**
 * 用于恢复贴纸中的声音
 * 恢复陀螺仪传感器,一般在Activity的onResume方法中调用
 */
this.mBeautyKit.onResume()
```

### 第八步：暂停声音

```
/**
 * 用于暂停贴纸中的声音
 * 暂停陀螺仪传感器,一般在Activity的onPause方法中调用
 */
this.mBeautyKit.onPause()
```

# Flutter

最近更新时间：2024-11-21 17:24:12

美颜 Flutter 版本 SDK 需要依赖 Android/iOS 端的美颜 SDK。通过 Flutter 提供的 Plugin 方式，可以将原生端的功能暴露给 Flutter 端。因此，在集成美颜功能时，需要手动集成原生端的 SDK。

## 跑通 Demo：

下载 Demo 工程，修改 `demo/lib` 下的 `main.dart` 文件，在此文件中添加您的 `licenseUrl` 和 `licenseKey`。直播中使用美颜的示例代码主要在 `demo/lib/page/live_page.dart` 和 `demo/lib/main.dart` 中。

### Android

1. 在 demo/app下找到 build.gradle 文件，打开文件，将 `applicationId` 的值修改为您申请 license 信息时填写的包名。
2. 在 demo/lib 下执行 `flutter pub get` 。
3. 使用 Android studio 打开 demo 工程并运行。

### iOS

1. 在 demo 下执行 `flutter pub get` 。
2. 在 demo/ios 目录下执行 `pod install` 。
3. 使用 Xcode工具打开 Runner.xcworkspace。
4. 修改工程的 `bundle ID` （需要和申请 license 时填写的 bundle ID 一致）。

## SDK集成：

### 原生端集成：

### Android

1. 在 app 模块下找到 build.gradle 文件，添加您对应套餐的 maven 引用地址，例如您选择的是S1-04套餐，则添加如下：

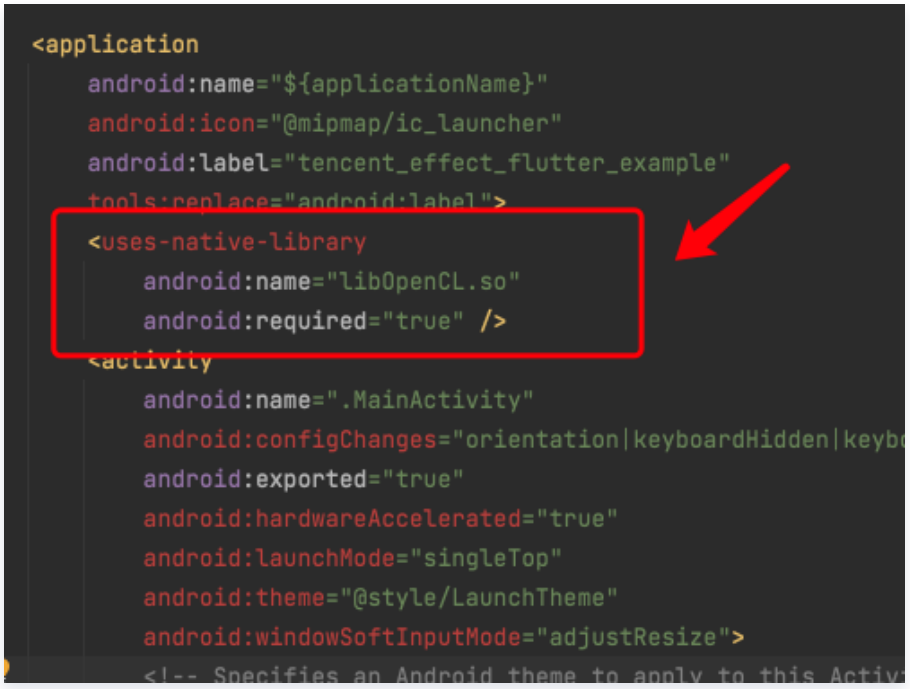
```
dependencies {
   implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

各套餐对应的 maven 地址，请参见文档，SDK的最新版本号可以在版本历史中查看。

2. 在您工程下的 `android/app` 模块下找到 `src/main/assets` 文件夹，将demo工程中 `demo/android/app/src/main/assets` 中的 `lut` 和 `MotionRes` 复制到您工程的 `android/app/ src/main/assets` 中，如果您的工程没有assets文件夹可以手动创建一个。
3. 在 app 模块下找到 AndroidManifest.xml 文件，在 application 表填内添加如下标签：

```
    <uses-native-library
            android:name="libOpenCL.so"
            android:required="false" />
        //true 表示libOpenCL是当前app必需的。如果没有此库，系统将不允许app安装
        //false 表示libOpenCL不是当前app必需的。无论有没有此库，都可以正常安装app。如果设备有此库，腾讯特效SDK里的GAN类型特效能正常生效（例如童话脸、国漫脸）。如果设备没有此库，GAN类型不会生效，但也不影响SDK内其他功能的使用。
        //关于uses-native-library的说明，请参考Android 官网介绍：https://developer.android.com/guide/topics/manifest/uses-native-library-element
```

添加后如下图：



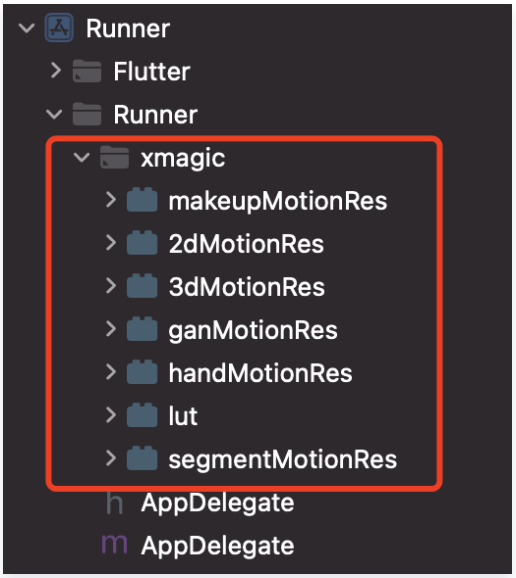

4. 混淆配置
   - 如果您在打 release 包时，启用了编译优化（把 minifyEnabled 设置为 true），会裁掉一些未在 java 层调用的代码，而这些代码有可能会被 native 层调用，从而引起 `no xxx method` 的异常。
   - 如果您启用了这样的编译优化，那就要添加这些 keep 规则，防止 xmagic 的代码被裁掉：

```
-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.** { *;}
```

iOS

1. 将demo工程中ios/Runner目录下的xmagic文件夹复制到您工程中ios/Runner目录下，添加后如下图：




> **注意：**
> 上述从demo工程中复制的素材是**测试素材**，正式素材需要您在购买套餐之后联系我们工作人员进行获取并重新添加。

## Flutter 端集成：

### 方式1：

远程依赖，在您工程的 pubspec.yaml 文件中添加如下引用：

```
tencent_effect_flutter:
 git:
   url: https://github.com/Tencent-RTC/TencentEffect_Flutter
```

### 方式2：

本地依赖，从 tencent_effect_flutter 下载最新版本的 tencent_effect_flutter，然后把文件夹 android、ios、lib 和文件 pubspec.yaml、tencent_effect_flutter.iml添加到工程目录下，然后在工程的 pubspec.yaml 文件中添加如下引用：（可参考 demo）

```
tencent_effect_flutter:
```

```
    path: ../
```

执行如下命令

```
flutter pub get
```

**注意：**
tencent_effect_flutter 只是提供一个桥接，美颜功能依赖各个平台提供的美颜SDK，所以如果需要更新美颜SDK，您可以通过以下步骤进行 SDK 升级：

**Android**

在您的工程下的 app 模块下找到 build.gradle 文件，将 `implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'` 中的 `latest.release` 字段修改为最新的版本号。
如果您需要更换套餐，那么需要将 `TencentEffect_S1-04` 字段修改为您的套餐字段 。

**iOS**

**需要更换套餐类型和 SDK 版本：**
1. 美颜 Flutter SDK 的依赖方式需要修改为**本地依赖**
2. 在美颜 Flutter SDK 下找到 `ios/tencent_effect_flutter.podspec` 文件，打开找到 `TencentEffect_All` 修改为您需要的套餐，后边的版本号也可以修改为您需要的版本号。
3. 在您工程的 ios 目录下执行 pod update 命令即可

# SDK 使用：

## 1. 与直播关联

**Android**

在应用的 application 类的 oncreate 方法（或 FlutterActivity 的 onCreate 方法）中添加如下代码：

```
TXLivePluginManager.register(new XmagicProcesserFactory());
```

**iOS**

在应用的 AppDelegate 类中的 didFinishLaunchingWithOptions 方法里面中添加如下代码：

```
XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
[TXLivePluginManager registerWithCustomBeautyProcesserFactory:instance];
```

添加后如下图：

```
#import "AppDelegate.h"
#import "GeneratedPluginRegistrant.h"
@import live_flutter_plugin;
@import tencent_effect_flutter;
@import tencent_trtc_cloud;

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  [GeneratedPluginRegistrant registerWithRegistry:self];
  // Override point for customization after application launch.
  XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
  [TXLivePluginManager registerWithCustomBeautyProcesserFactory:instance];
  [TencentTRTCCloud registerWithCustomBeautyProcesserFactory:instance];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}

@end
```

## 2. 调用资源初始化接口

v0.3.5.0及之后

```
void _initSettings(InitXmagicCallBack callBack) async {
  _setResourcePath();
  /// 复制资源只需要复制一次，在当前版本中如果成功复制了一次，以后就不需要再复制资源。
  /// Copying the resource only needs to be done once. Once it has been
successfully copied in the current version, there is no need to copy it again
in future versions.
  if (await isCopiedRes()) {
    callBack.call(true);
    return;
  } else {
    _copyRes(callBack);
  }
}
```

```
void _setResourcePath() async {
  String resourceDir = await ResPathManager.getResManager().getResPath();
  TXLog.printlog(
      '$TAG method is _initResource ,xmagic resource dir is $resourceDir');
  TencentEffectApi.getApi()?.setResourcePath(resourceDir);
}

void _copyRes(InitXmagicCallBack callBack) {
  _showDialog(context);
  TencentEffectApi.getApi()?.initXmagic((result) {
    if (result) {
      saveResCopied();
    }
    _dismissDialog(context);
    callBack.call(result);
    if (!result) {
      Fluttertoast.showToast(msg: "initialization failed");
    }
  });
}
```

**v0.3.1.1版本及之前**

```
String dir =  await BeautyDataManager.getInstance().getResDir();
 TXLog.printlog('文件路径为: $dir');
 TencentEffectApi.getApi()?.initXmagic(dir,(reslut) {
     _isInitResource = reslut;
     callBack.call(reslut);
     if (!reslut) {
         Fluttertoast.showToast(msg: "初始化资源失败");
     }
 });
```

## 3. 进行美颜授权

```
TencentEffectApi.getApi()?.setLicense(licenseKey, licenseUrl,
          (errorCode, msg) {
        TXLog.printlog("打印鉴权结果 errorCode = $errorCode   msg = $msg");
        if (errorCode == 0) {
           //鉴权成功
        }
      });
```

## 4. 开启美颜

```
///开启美颜操作
 var enableCustomVideo = await trtcCloud.enableCustomVideoProcess(open);
```

## 5. 设置美颜属性

v0.3.5.0及之后

具体美颜属性请参考 美颜属性表

```
TencentEffectApi.getApi()?.setEffect(sdkParam.effectName!,
    sdkParam.effectValue, sdkParam.resourcePath, sdkParam.extraInfo)
```

v0.3.1.1版本及之前

```
TencentEffectApi.getApi()?.updateProperty(_xmagicProperty!);
///_xmagicProperty 可通过 BeautyDataManager.getInstance().getAllPannelData();获
取所有的属性，需要使用美颜属性的时候可通过updateProperty方法设置属性。
```

## 6. 设置其他属性

- **暂停美颜音效**

```
 TencentEffectApi.getApi()?.onPause();
```

- **恢复美颜音效**

```
TencentEffectApi.getApi()?.onResume();
```

- **监听美颜事件**

```
TencentEffectApi.getApi()
     ?.setOnCreateXmagicApiErrorListener((errorMsg, code) {
       TXLog.printlog("创建美颜对象出现错误 errorMsg = $errorMsg , code = $code");
 });    ///需要在创建美颜之前进行设置
```

- **设置人脸、手势、身体检测状态回调**

```
TencentEffectApi.getApi()?.setAIDataListener(XmagicAIDataListenerImp());
```

- **设置动效提示语回调函数**

```
TencentEffectApi.getApi()?.setTipsListener(XmagicTipsListenerImp());
```

- **设置人脸点位信息等数据回调（S1-05 和 S1-06 套餐才会有回调）**

```
TencentEffectApi.getApi()?.setYTDataListener((data) {
   TXLog.printlog("setYTDataListener  $data");
 });
```

- **移除所有回调**。在页面销毁的时候需要移除掉所有的回调：

```
 TencentEffectApi.getApi()?.setOnCreateXmagicApiErrorListener(null);
 TencentEffectApi.getApi()?.setAIDataListener(null);
 TencentEffectApi.getApi()?.setYTDataListener(null);
 TencentEffectApi.getApi()?.setTipsListener(null);
```

> **说明：**
> 接口详细可参考 API文档，其他可参考 Demo 工程。

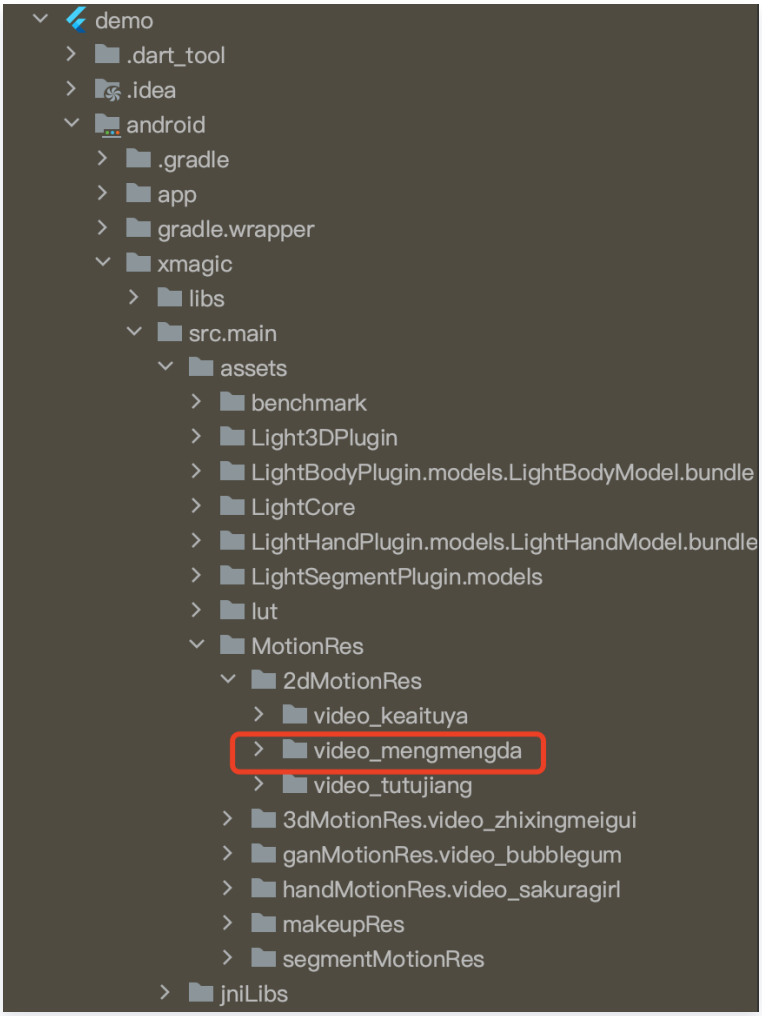
## 7. 添加和删除美颜面板上的美颜数据

### 添加美颜资源

把您的资源文件按照步骤一中的方法添加到对应的资源文件夹里面。例如您需要添加2D动效的资源：

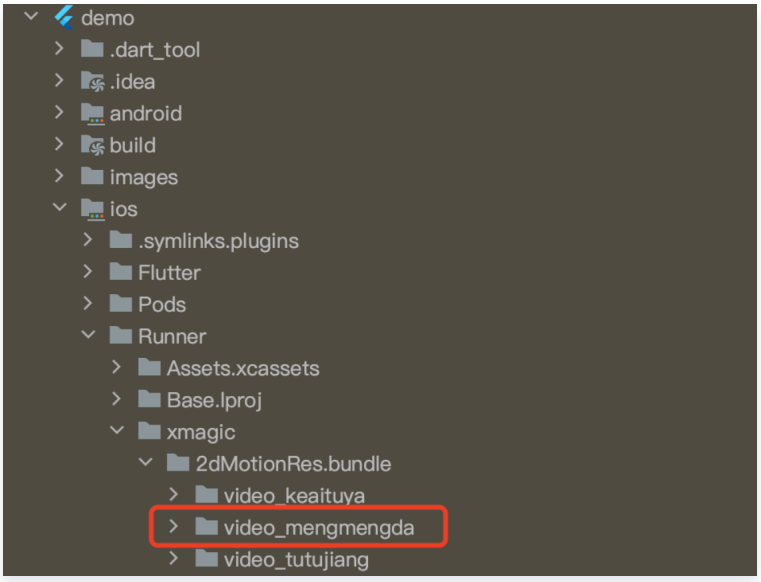
**Android**

1. 您应该把资源放在工程的 `android/xmagic/src.mian/assets/MotionRes/2dMotionRes` 目录下：




## iOS

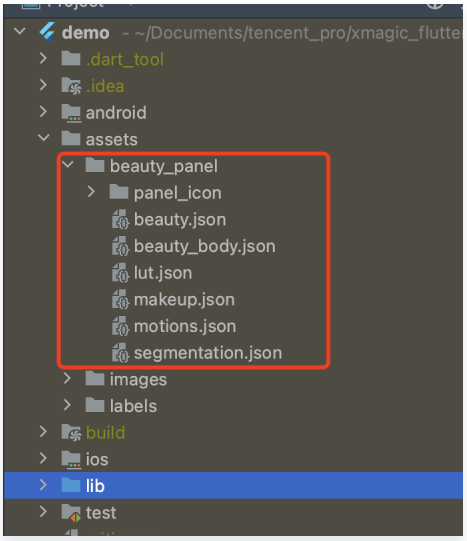
1. **并且把资源添加到工程的** `ios/Runner/xmagic/2dMotionRes.bundle` **目录下。**



**美颜面板配置：**

V0.3.5.0及之后

在 demo 中提供了一个简单的美颜面板 UI，面板上的属性通过 JSON 文件进行配置，JSON 文件位置如下图。面板的实现可以参考 demo 工程。




JSON 文件说明

V0.3.1.1及之前

在 BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid 和 BeautyPropertyProducerIOS 这4个类中，您可以自主操作美颜面板数据的配置。

## 删除美颜资源

对于某些 License 没有授权美颜和美体的部分功能，美颜面板上不需要展示这部分功能，需要在美颜面板数据的配置中删除这部分功能的配置。
例如，删除口红特效：分别在 BeautyPropertyProducerAndroid 类和 BeautyPropertyProducerIOS 类中的 getBeautyData 方法中删除以下代码。

```
// Map<String, String> lipsResPathNames = {};
// lipsResPathNames["lips_fuguhong.png"] = "复古红";
// lipsResPathNames["lips_mitaose.png"] = "蜜桃色";
// lipsResPathNames["lips_shanhuju.png"] = "珊瑚橘";
// lipsResPathNames["lips_wenroufen.png"] = "温柔粉";
// lipsResPathNames["lips_huolicheng.png"] = "活力橙";
// List<XmagicUIProperty> itemLipsPropertys = [];
// String lipId = "beauty.lips.lipsMask";
// for (String ids in lipsResPathNames.keys) {
//   itemLipsPropertys.add(XmagicUIProperty(
//       uiCategory: Category.BEAUTY,
//       displayName: lipsResPathNames[ids]!,
//       id: lipId,
//       resPath: resPaths + ids,
//       thumbDrawableName: "beauty_lips",
//       effKey: BeautyConstant.BEAUTY_MOUTH_LIPSTICK,
//       effValue: XmagicPropertyValues(0, 100, 50, 0, 1),
//       rootDisplayName: "口红"));
// }
// XmagicUIProperty itemLips = XmagicUIProperty(
//     displayName: "口红",
//     thumbDrawableName: "beauty_lips",
//     uiCategory: Category.BEAUTY);
// itemLips.xmagicUIPropertyList = itemLipsPropertys;
// beautyList.add(itemLips);
```

# 短视频 SDK 集成腾讯特效 iOS

最近更新时间：2022-12-13 10:12:23

## 集成准备

1. 下载并解压 Demo 包，将 Demo 工程中 `demo/XiaoShiPin/` 目录下的 xmagickit 文件夹拷贝到您的工程 podfile 文件的同一级目录下。
2. 在您的 Podfile 文件中添加以下依赖，之后执行 `pod install` 命令，完成导入。

```
pod 'xmagickit', :path => 'xmagickit/xmagickit.podspec'
```

3. 将 Bundle ID 修改成与申请的测试授权一致。

## 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 12.0 及以上。

## SDK 接口集成

### 步骤一：初始化授权

在工程 `AppDelegate` 的 `didFinishLaunchingWithOptions` 中添加如下代码，其中 `LicenseURL` ， `LicenseKey` 为腾讯云官网申请到授权信息，请参见 License 指引（XMagic SDK版本在2.5.1以前， `TELicenseCheck.h` 在 `XMagic.framework` 里面；XMagicSDK 版本在2.5.1及以后， `TELicenseCheck.h` 在 `YTCommonXMagic.framework` 里面）：

```
[TXUGCBase setLicenceURL:LicenseURL key:LicenseKey];

[TELicenseCheck setTELicense:LicenseURLkey: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：设置 SDK 素材资源路径

```
CGSize previewSize = [self
getPreviewSizeByResolution:self.currentPreviewResolution];
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];
```

## 步骤三：添加日志和事件监听

```
// Register log
[self.beautyKit registerSDKEventListener:self];
[self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL];
```

## 步骤四：配置美颜各种效果

```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString
*_Nonnull)propertyName withData:(NSString*_Nonnull)propertyValue withExtraInfo:(id
_Nullable)extraInfo;
```

## 步骤五：进行渲染处理

在短视频预处理帧回调接口，构造 YTProcessInput 将 textureId 传入到 SDK 内做渲染处理。

```
 [self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft
withOrientation:YtLightCameraRotation0]
```

## 步骤六：暂停/恢复 SDK

```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

## 步骤七：布局中添加 SDK 美颜面板

```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
if (@available(iOS 11.0, *)) {
    gSafeInset = [UIApplication sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
    {
```

```
        gSafeInset = UIEdgeInsetsZero;
    }

dispatch_async(dispatch_get_main_queue(), ^{
    //美颜选项界面
    _vBeauty = [[BeautyView alloc] init];
    [self.view addSubview:_vBeauty];
    [_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
        make.width.mas_equalTo(self.view);
        make.centerX.mas_equalTo(self.view);
        make.height.mas_equalTo(254);
        if(gSafeInset.bottom > 0.0){  // 适配全面屏
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
        } else {
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
        }
    }];
        _vBeauty.hidden = YES;
});
```

# Android

最近更新时间：2024-11-21 11:43:22

## 步骤一：解压 Demo 工程

1. 下载集成了腾讯特效 TE 的 UGSV Demo 工程。本 Demo 基于腾讯特效 SDK S1-04 套餐构建。
2. 替换资源：由于本 Demo 工程使用的 SDK 套餐未必与您实际的套餐一致，因此要将本 Demo 中的相关 SDK 文件替换为您实际使用的套餐的 SDK 文件。具体操作如下：
   - 在 `xmagickit module` 的 `build.gradle` 文件找到

     ```
     api 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
     ```

     替换为您购买的 套餐依赖包 。
   - 如果您的套餐包含动效和滤镜功能，那么需要在腾讯特效 SDK 下载页面 下载对应的资源，将动效和滤镜素材放置在 `xmagickit module` 下的如下目录：
     - 动效： `../assets/MotionRes`
     - 滤镜： `../assets/lut`
3. 将 Demo 程中的 xmagickit 模块引 到实际项 程中。

## 步骤二：打开 app 模块的 build.gradle

将 applicationId 修改成与申请的测试授权 致的包名。

## 步骤三：SDK 接口集成

可参考 Demo 程的 UGCKitVideoRecord 类。

1. 授权：

```
 //鉴权注意事项及错误码详情，请参考
https://cloud.tencent.com/document/product/616/65891#.E6.AD.A5.E9.AA.A4.E4.B8.80
.EF.BC.9A.E9.89.B4.E6.9D.83
 XMagicImpl.checkAuth(new TELicenseCheck.TELicenseCheckListener() {
         @Override
         public void onLicenseCheckFinish(int errorCode, String msg) {
             if (errorCode == TELicenseCheck.ERROR_OK) {
                 loadXmagicRes();
             } else {
                 Log.e("TAG", "auth fail , please check auth url and key" +
errorCode + " " + msg);
             }
         }
     });
```

2. 初始化素材：

```
private void loadXmagicRes() {
    if (XMagicImpl.isLoadedRes) {
        XmagicResParser.parseRes(mActivity.getApplicationContext());
        initXMagic();
        return;
    }
    new Thread(new Runnable() {
        @Override
        public void run() {
            XmagicResParser.copyRes(mActivity.getApplicationContext());
            XmagicResParser.parseRes(mActivity.getApplicationContext());
            XMagicImpl.isLoadedRes = true;
            new Handler(Looper.getMainLooper()).post(new Runnable() {
                @Override
                public void run() {
                    initXMagic();
                }
            });
        }
    }).start();
}
```

3. **短视频和美颜进行绑定：**

```
private void initBeauty() {
    TXUGCRecord instance = TXUGCRecord.getInstance(UGCKit.getAppContext());
    instance.setVideoProcessListener(new
TXUGCRecord.VideoCustomProcessListener() {
        @Override
        public int onTextureCustomProcess(int textureId, int width, int height)
{
            if (xmagicState == XMagicImpl.XmagicState.STARTED && mXMagic !=
null) {
                return mXMagic.process(textureId, width, height);
            }
            return textureId;
        }

        @Override
        public void onDetectFacePoints(float[] floats) {
        }

        @Override
        public void onTextureDestroyed() {
            if (Looper.getMainLooper() != Looper.myLooper()) {  //非主线程
                boolean stopped = xmagicState ==
XMagicImpl.XmagicState.STOPPED;
```

```
                if (stopped || xmagicState == XMagicImpl.XmagicState.DESTROYED)
{
                    if (mXMagic != null) {
                        mXMagic.onDestroy();
                    }
                }
                if (xmagicState == XMagicImpl.XmagicState.DESTROYED) {

TXUGCRecord.getInstance(UGCKit.getAppContext()).setVideoProcessListener(null);
                }
            }
        }
    });
 }
```

4. **暂停/销毁 SDK：** `onPause()` 用于暂停美颜效果，可以在 Activity/Fragment 生命周期方法中执行，onDestroy 方法需要在 GL 线程调用（可以在 onTextureDestroyed 方法中调用 XMagicImpl 对象的 `onDestroy()` ），更多使用请参考示例中 onTextureDestroyed 方法。

```
        @Override
        public void onTextureDestroyed() {
            if (Looper.getMainLooper() != Looper.myLooper()) {  //非主线程
                boolean stopped = xmagicState ==
XMagicImpl.XmagicState.STOPPED;
                if (stopped || xmagicState == XMagicImpl.XmagicState.DESTROYED)
{
                    if (mXMagic != null) {
                        mXMagic.onDestroy();
                    }
                }
                if (xmagicState == XMagicImpl.XmagicState.DESTROYED) {

TXUGCRecord.getInstance(UGCKit.getAppContext()).setVideoProcessListener(null);
                }
            }
        }
```

5. **布局中添加承载美颜面板的布局：**

```
 <RelativeLayout
     android:id="@+id/panel_layout"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:layout_alignParentBottom="true"
     android:visibility="gone"/>
```

6. **创建美颜对象并添加美颜面板：**

```
private void initXMagic() {
    if (mXMagic == null) {
        mXMagic = new XMagicImpl(mActivity, getBeautyPanel());
    } else {
        mXMagic.onResume();
    }
}
```

具体操作请参见 Demo 程的 UGCKitVideoRecord类。

# Avatar 虚拟人集成指引
# iOS
# 快速跑通 Demo

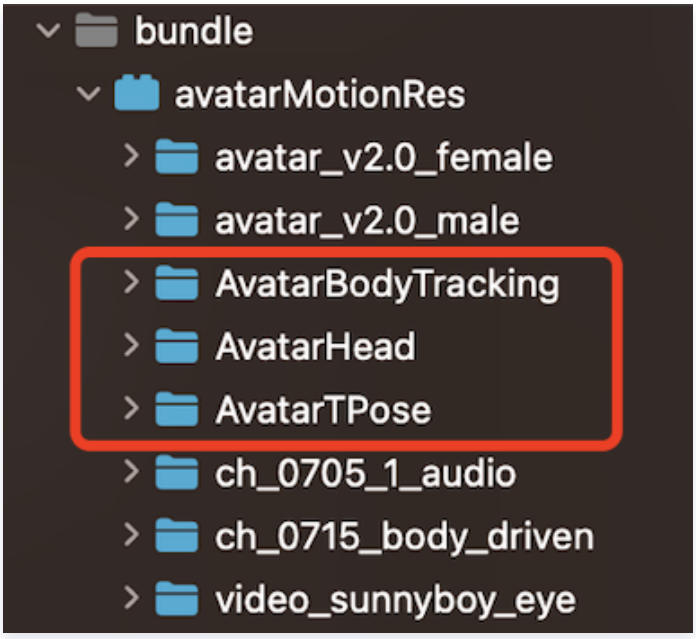
最近更新时间：2024-08-20 14:07:41

1. 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。
2. 在 `BeautyDefine.h` 中设置自己的 LicenseURL 和 LicenseKEY，并且在 Signing & Capabilities 中设置 license 对应的 Bundle Identifier。

   通过 `HomeViewController.m` 中的 `setTELicense` 判断是否正确设置 LicenseURL、LicenseKEY 和 Bundle Identifier。
3. 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。




4. 完成以上步骤，即可运行 Demo。

# 快速接入 Avatar

最近更新时间：2024-08-20 14:07:41

由于 Avatar 是腾讯特效的部分功能，所以需要先集成腾讯美颜特效 SDK，再加载 Avatar 素材即可。若未接入腾讯美颜特效 SDK，可参见 独立集成腾讯特效 进行了解与集成。

## 步骤1：准备 Avatar 素材

1. 集成腾讯特效 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];
```

## 代码参考：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);
    NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                                                                @"root_path":
[[NSBundle mainBundle] bundlePath]
    };
    // 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. 设置半身或全身形象。

```
// 设置半身
- (void)setupHalfOrWholeBody
{
    AvatarData *config = [self createAvatarData];
    [self.beautyKit updateAvatar:@[config] completion:^(BOOL success,
NSArray<AvatarData *> * _Nullable invalidAvatarList) {
```

```
        NSLog(@"---");
    }];
}
- (AvatarData *)createAvatarData
{
    AvatarData *config = [[AvatarData alloc] init];
    config.entityName = @"Camera3D";
    config.action = @"basicTransform";
    config.value = @{@"position" : _avatarViewValue[_viewIndex]};
    return config;
}
```

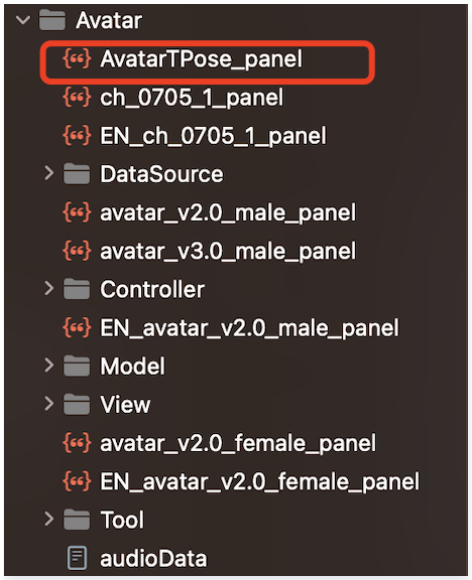
# 自定义 Avatar UI

最近更新时间：2024-08-20 14:07:41

## Demo UI 说明

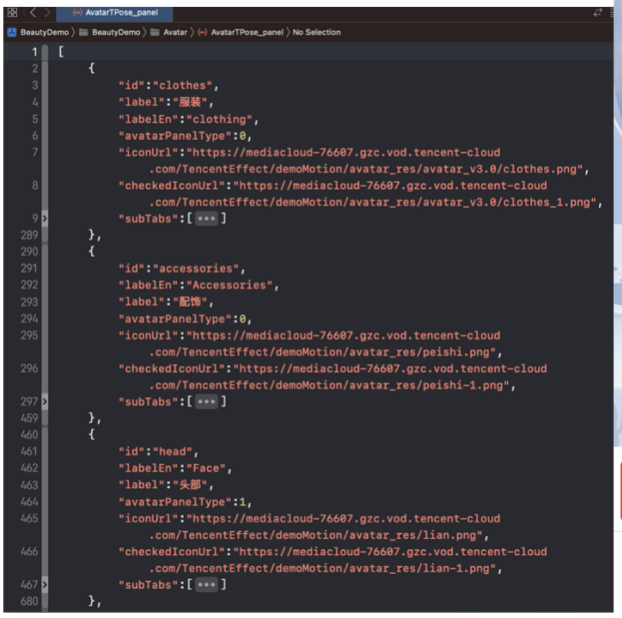
## 实现方式

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

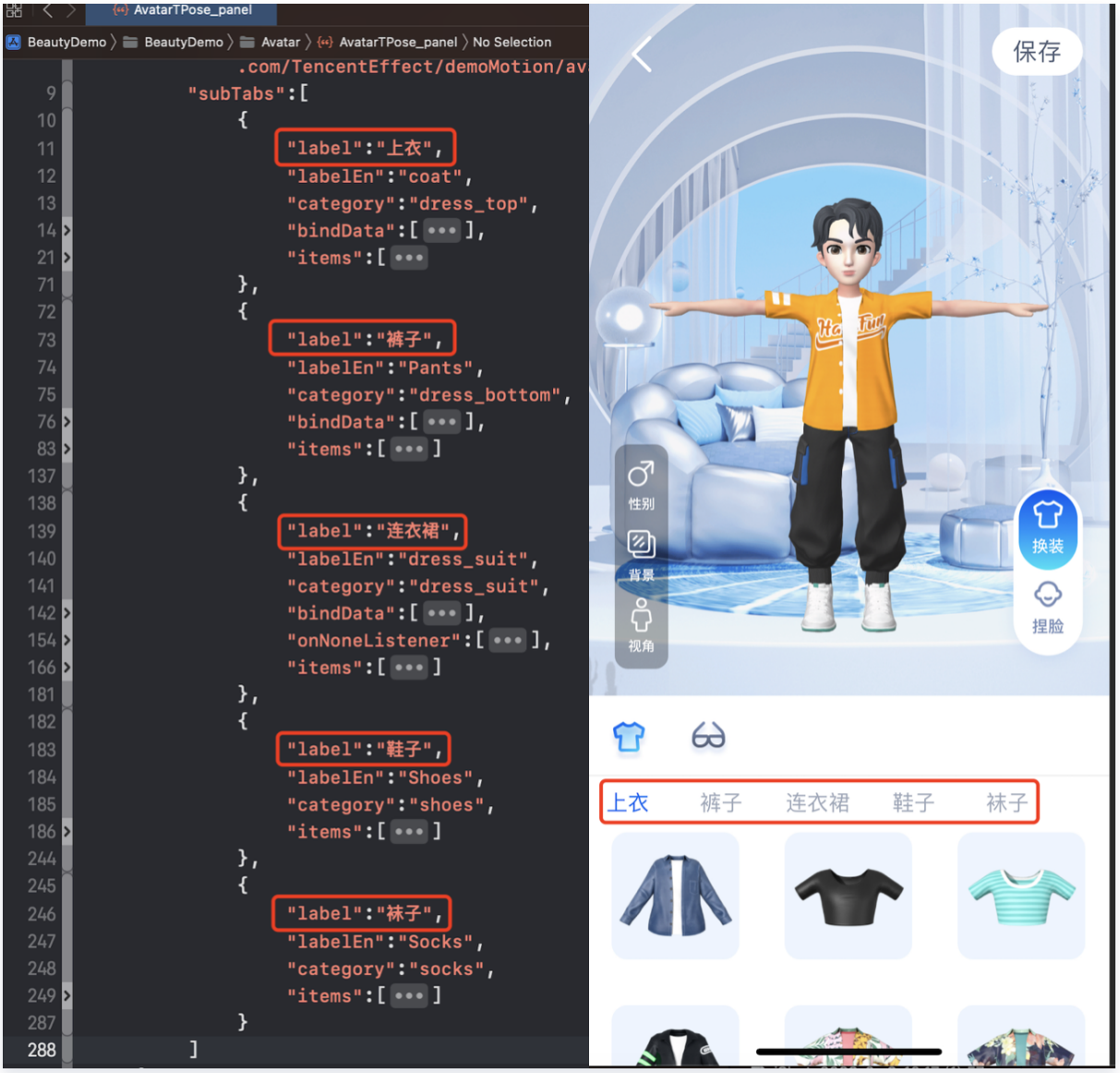



**Json 结构和 UI 面板对应关系：**

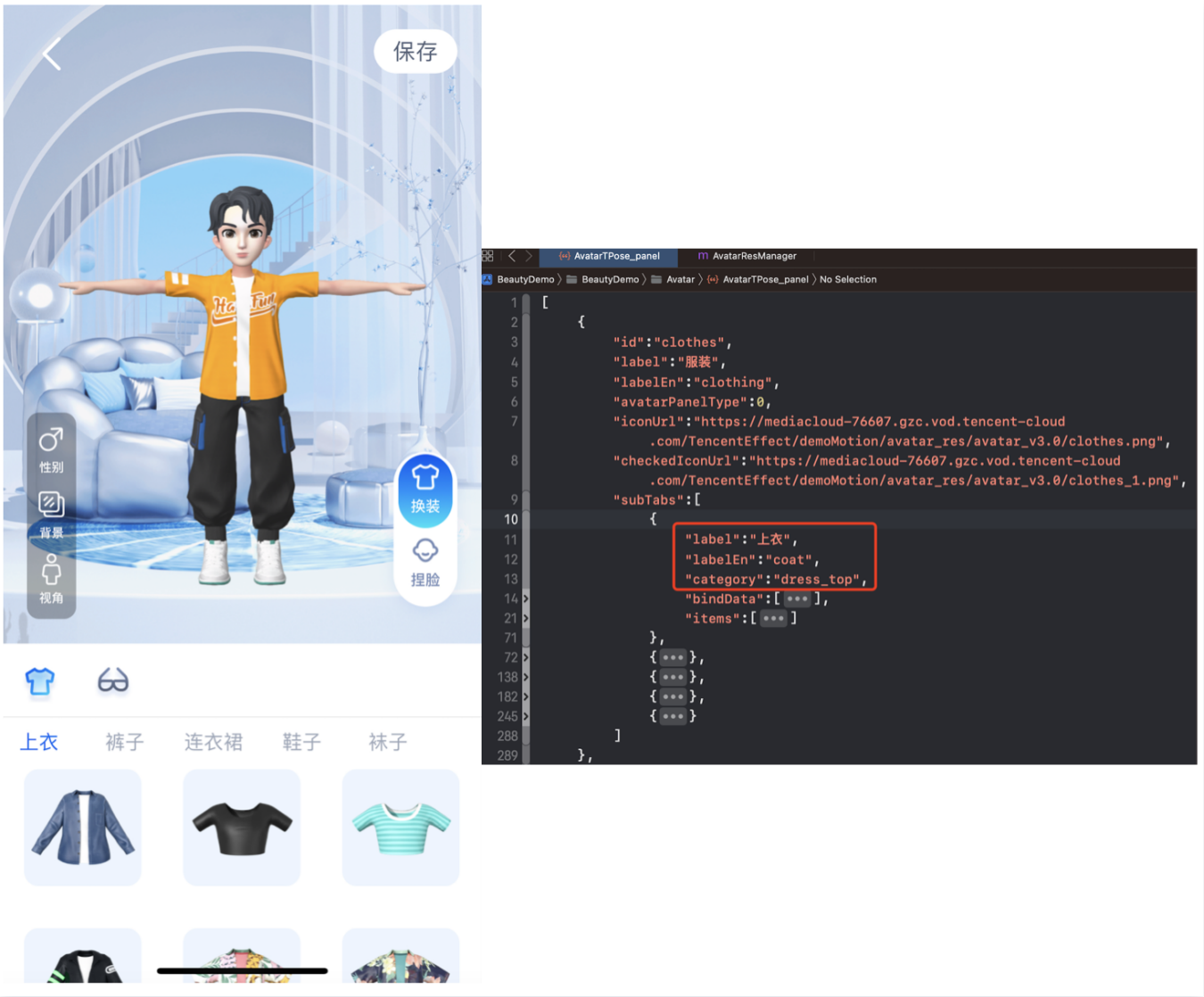
- 左侧 item 对应右侧页面一级菜单，clothes 为第一个 icon 选中的内容：



- 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：`BeautyDemo/Avatar/DataSource/AvatarResManager.m`

### 1. 拷贝 Avatar 素材

```
/**
 copy资源到沙盒，因为有下载某单个素材需求，在mainBundle里面的资源文件无法修改，
 需要拷贝到沙盒，才可以往资源添加文件
 */
- (void)copyBundleToSanboxIfNeed
```

### 2. 获取面板数据

```
// 获取UI面板数据
- (NSArray *)getPanelData:(AvatarGender)gender
```

### 3. 保存捏脸形象数据

```
// 保存捏脸形象数据
- (BOOL)saveSelectedAvatarConfigs:(AvatarGender)gender image:(NSData *)image
```

### 4.获取保存的捏脸形象数据

```
//获取保存的全身形象数据
- (void)getSavedAvatarConfigs
```

### 5. 获取默认的全身形象数据

```
// 获取默认的全身形象数据
- (NSString *)getDefaultAvatarConfigs:(AvatarGender)gender
```

# Avatar SDK 说明

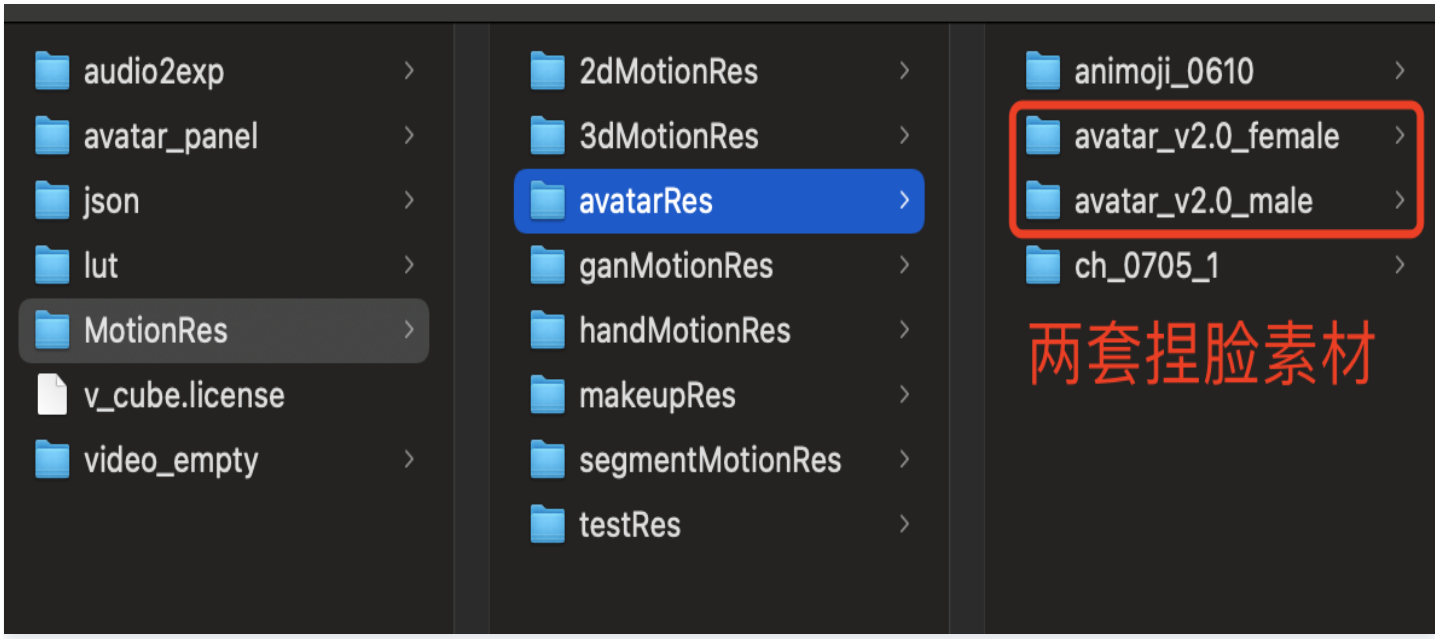
最近更新时间：2024-08-20 14:07:41

## 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 。

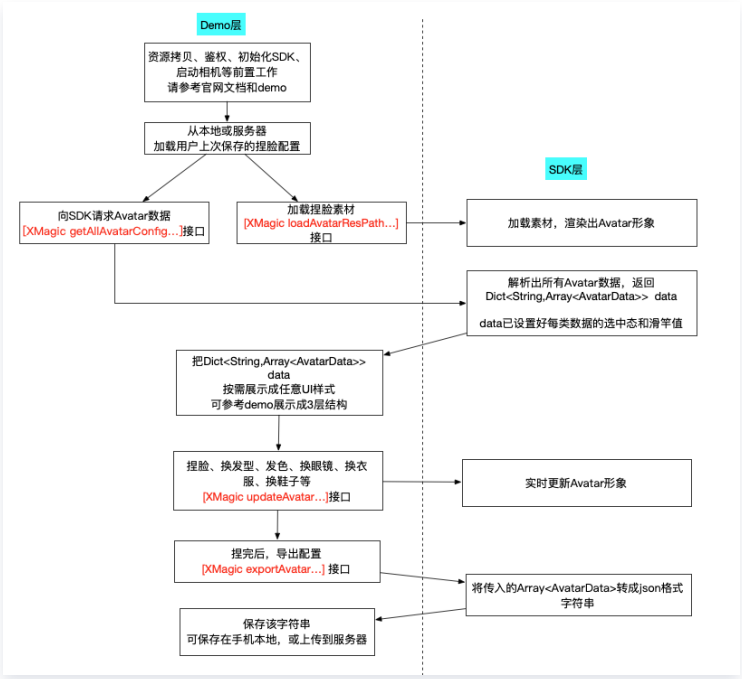
## 准备捏脸素材

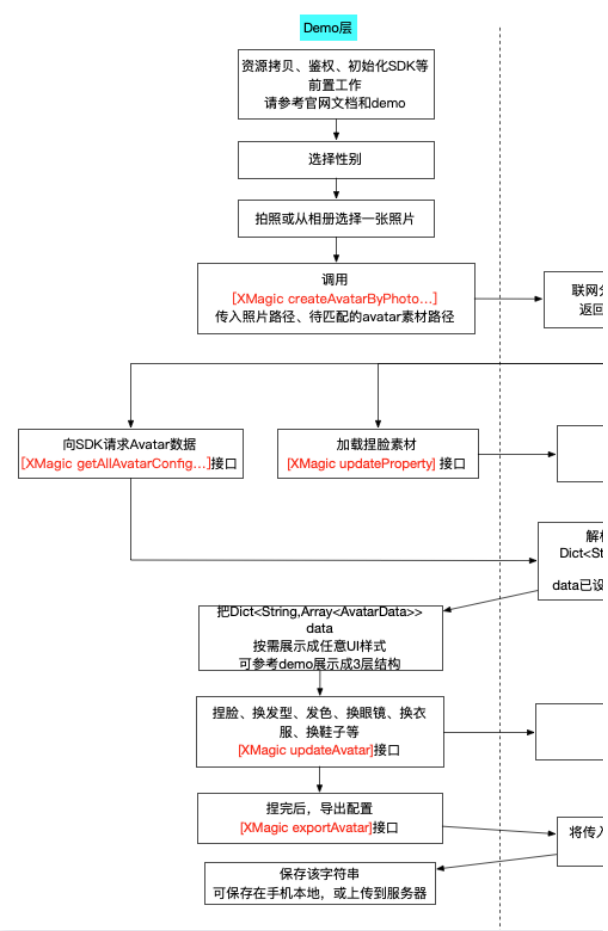
目前我们随 SDK 提供了若干套捏脸、换装素材，素材在 SDK 解压后的 `MotionRes/avatarRes` 目录中，与其他的动效素材一样，您需要把它 copy 到工程的 assets 目录：




## 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;
```

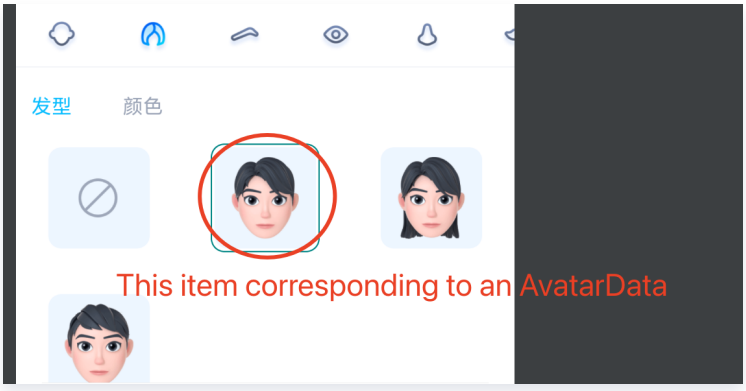
- **eventKey**：自定义事件 key，可参考 TEDefine 的 AvatarCustomEventKey。
- **eventValue**：自定义事件 value，为 JSON 字符串，例如
  `@{@"enabel" : @(YES)}`转json字符串，或者直接写`@"{\\"enable\\" : true}"` 。

## 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。

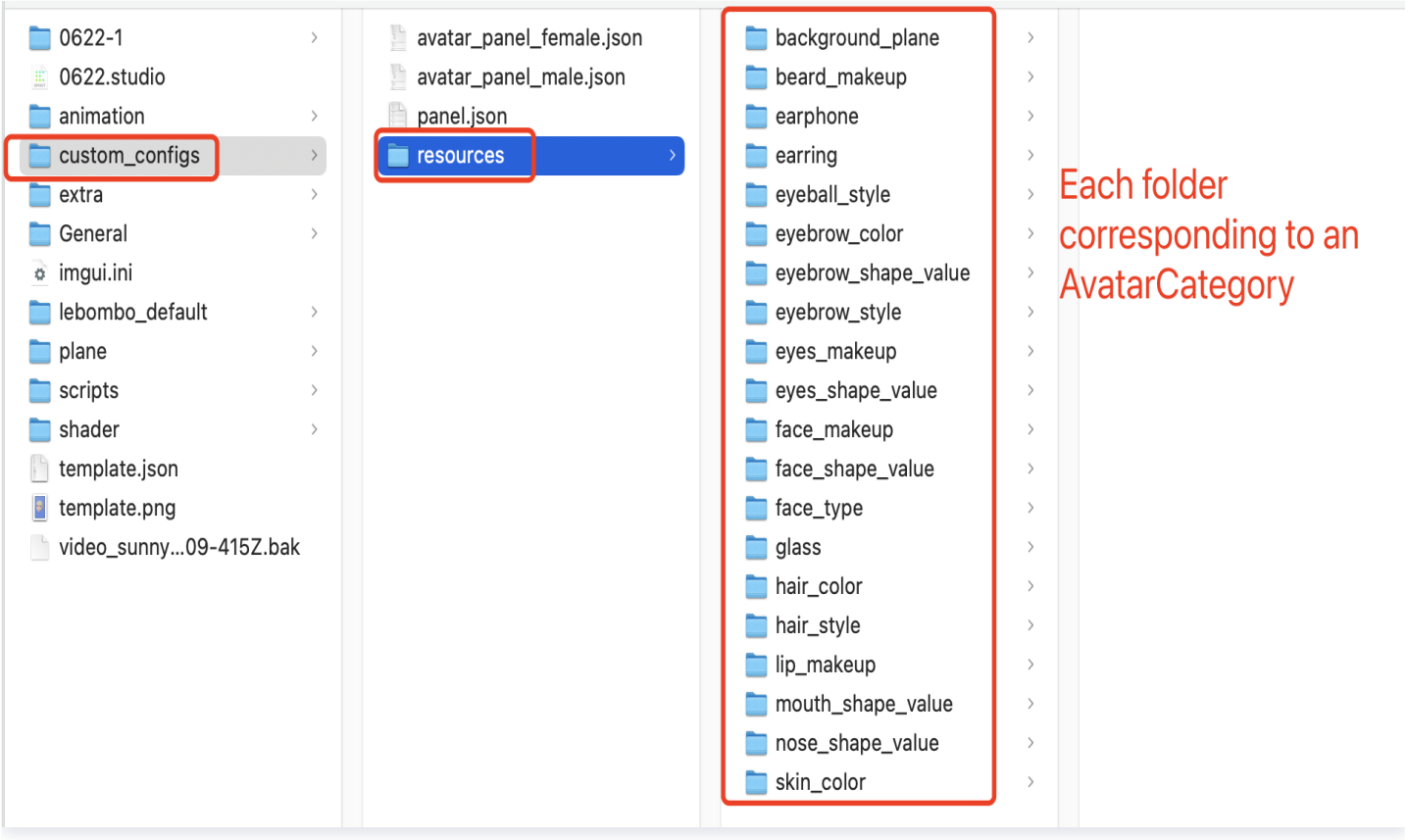
## action 和 value字段

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`）：

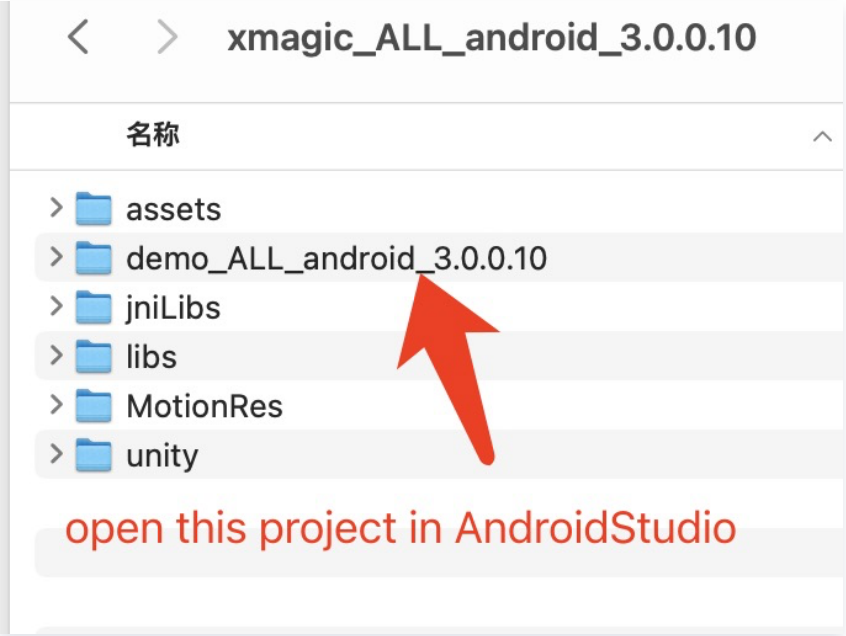



这些配置文件是自动生成的，通常不需要手动配置。自动生成的方式如下：
设计师用 TencentEffectStudio 设计好一套形象后，运行我们提供的 resource_generator_gui 这个 app（目前仅支持 MacOS 平台），即可自动生成这些配置，详情请参见 设计规范 。

# Android
# 快速跑通 demo

最近更新时间：2024-02-19 11:42:11

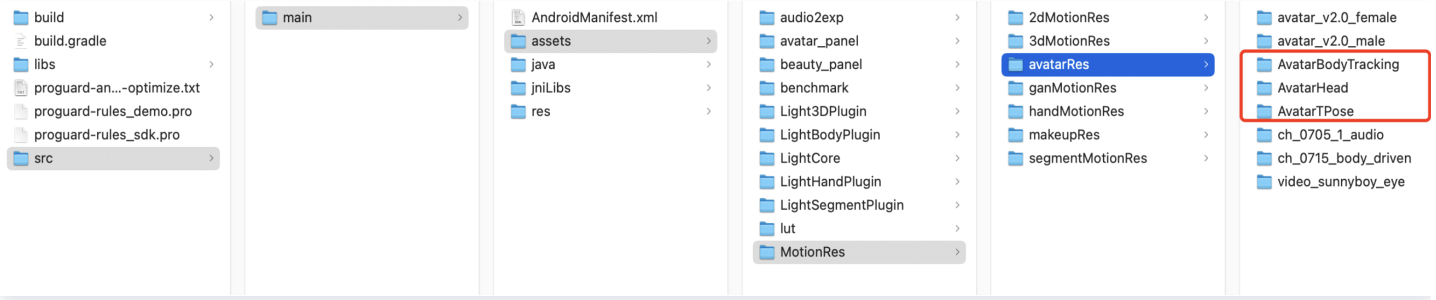
1. 使用 Android Studio 打开项目工程。




2. 在工程的 `com.tencent.demo.constant.LicenseConstant.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。

```
cal.properties   build.gradle (:demo)
dependencies{}
can use the Project Structure dialog to view and edit your project configuration
plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        long time = System.currentTimeMillis()
        applicationId "com.tencent.pitumotiondemo.effects"
        minSdkVersion 21
        targetSdkVersion 29
        versionCode 1
        versionName String.valueOf(time)
        ndk {
            abiFilters "armeabi-v7a" , "arm64-v8a"
        }
```

4. 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。




5. 完成以上步骤，即可运行 demo。

# 快速接入 Avatar

最近更新时间：2023-05-25 15:43:55

由于 Avatar 只是腾讯特效的部分功能，所以在接入时需参考 腾讯特效接入文档，将 SDK 集成到您的项目之中。然后按照下文的方法添加 Avatar UI、加载 Avatar 素材。

1. 按照腾讯特效文档进行接入，请参见 独立集成腾讯特效 。
2. 按照如下方法加载 Avatar 素材。

## 使用 Avatar 功能具体步骤

### 步骤1：复制 Demo 中的文件

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：添加关键代码

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

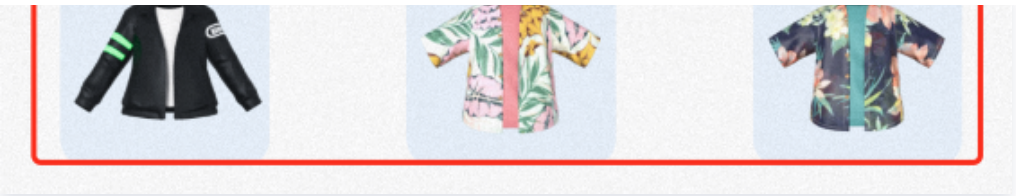
动作

# 自定义 Avatar UI

最近更新时间：2023-05-17 17:08:14

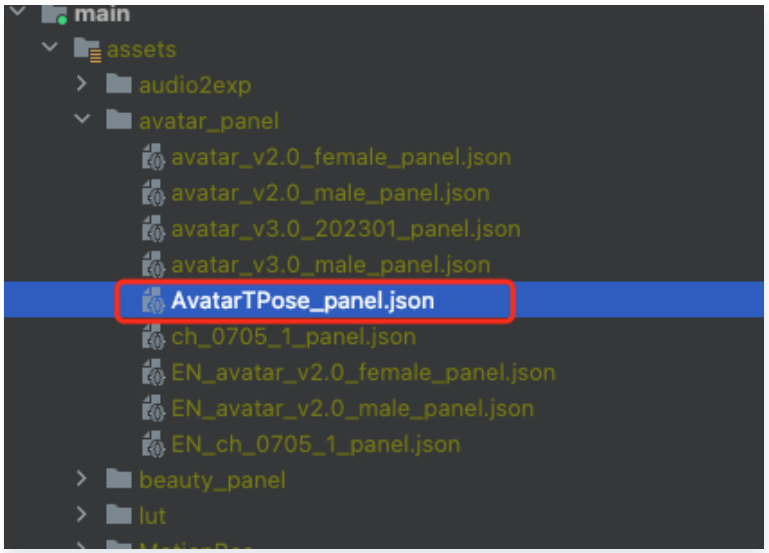
## Demo UI 说明

保存
用于展示 avatar 形象
切换男女形象
性别
背景面板入口
背景
切换 Avatar 形象视角
视角
用于切换换装和捏脸面板
换装
捏脸
一级菜单
上衣
裤子
连衣裙
鞋子
袜子
二级菜单
属性展示区域



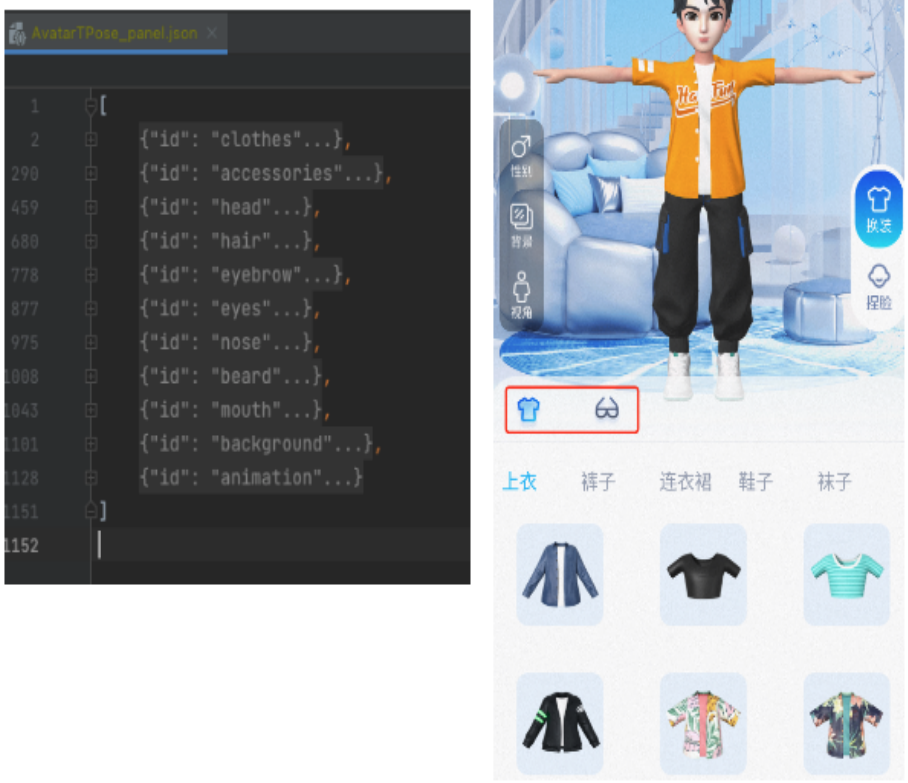
## 实现方式

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。

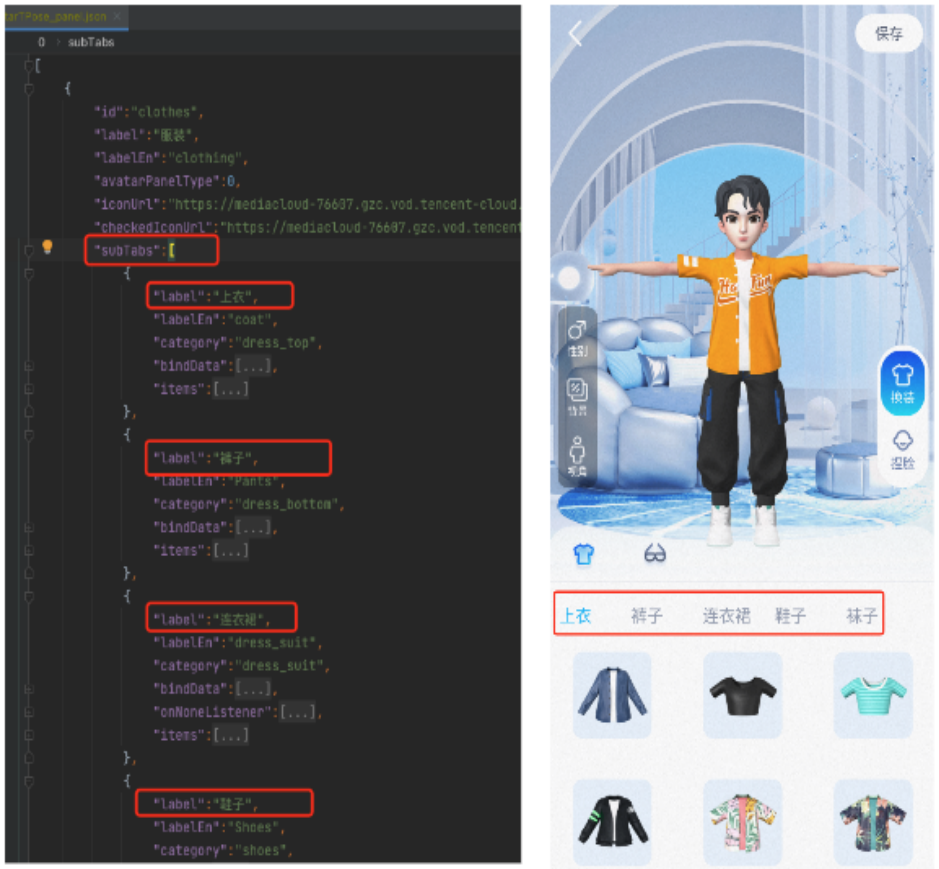



**Json 结构和 UI 面板对应关系：**

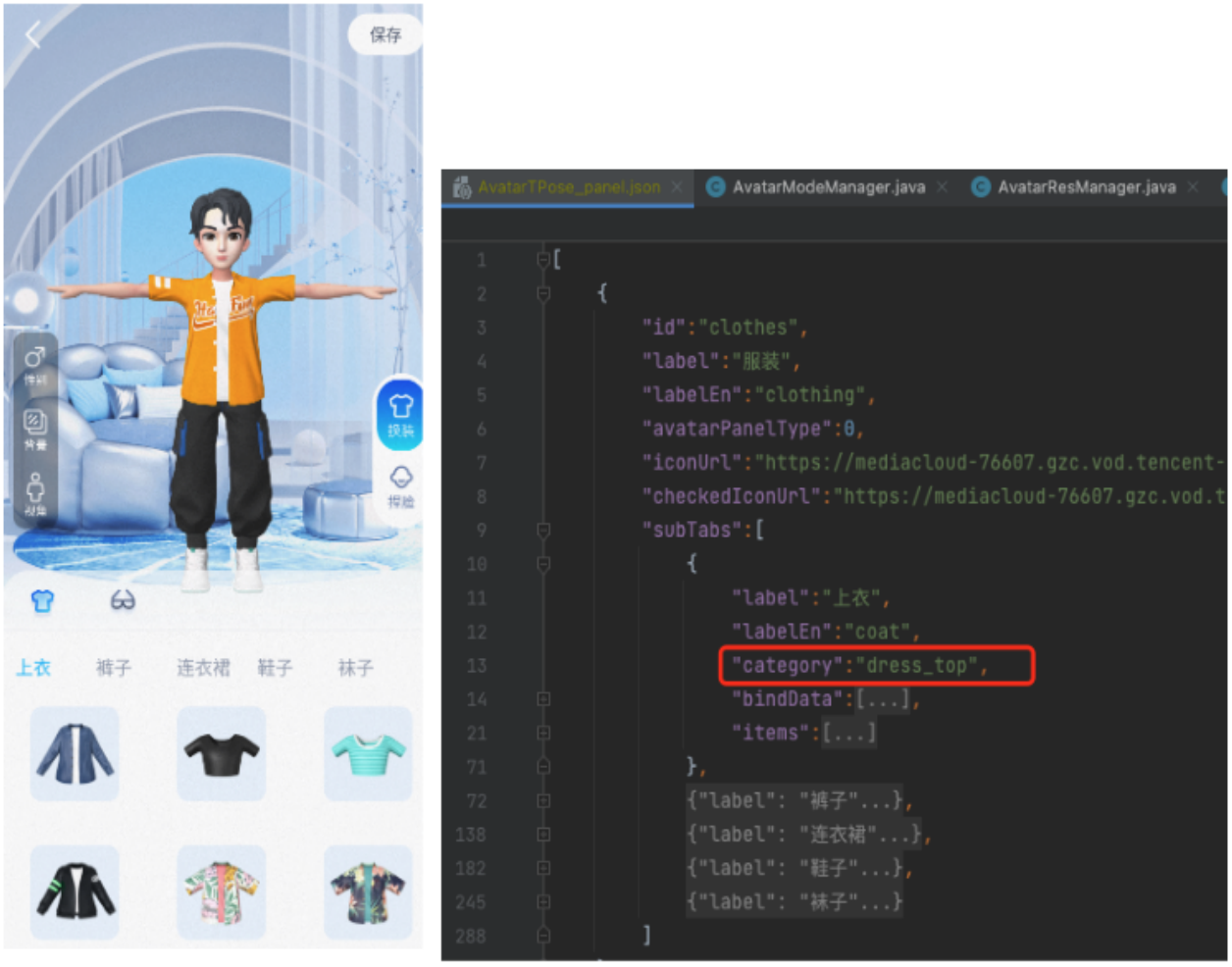
- 左侧 item 对应右侧页面一级菜单，clothes 为第一个 icon 选中的内容：




- 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：<br>0：换装面板<br>1：捏脸面板<br>2：背景面板<br>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：<br>0： 表示icon类型，默认值。<br>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。<br>举例：<br>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。<br>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。<br>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。<br>举例：<br>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。<br>举例：<br>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。<br>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。<br>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 |

# Avatar SDK 说明

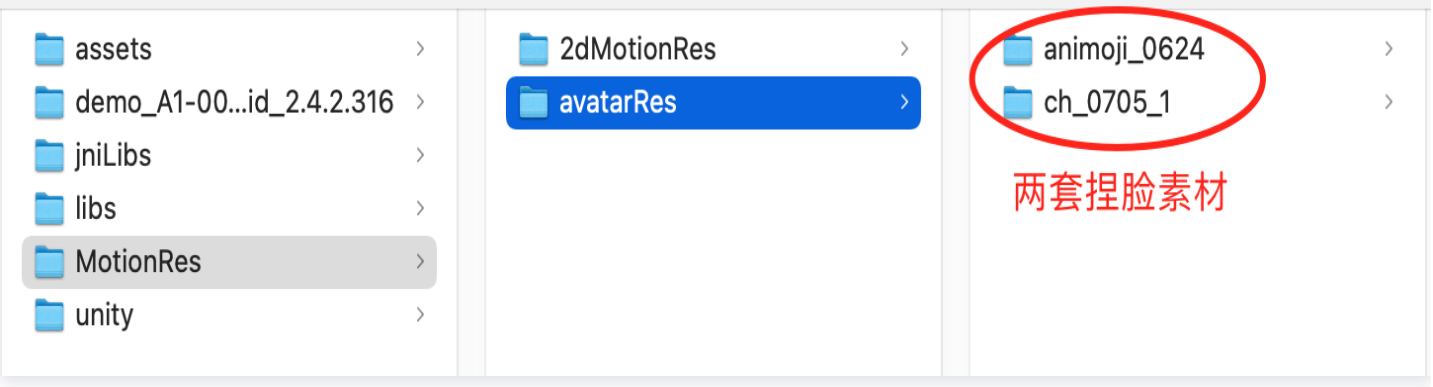
最近更新时间：2023-05-15 14:46:14

## 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 。

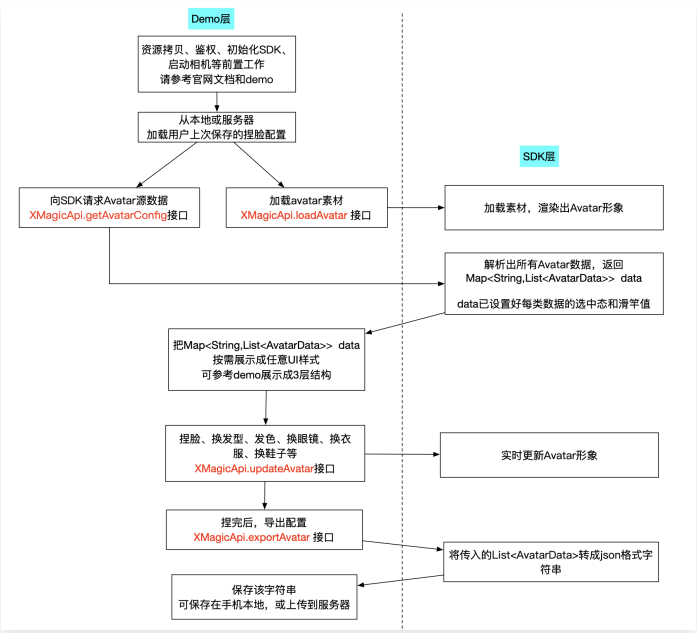
## 准备捏脸素材

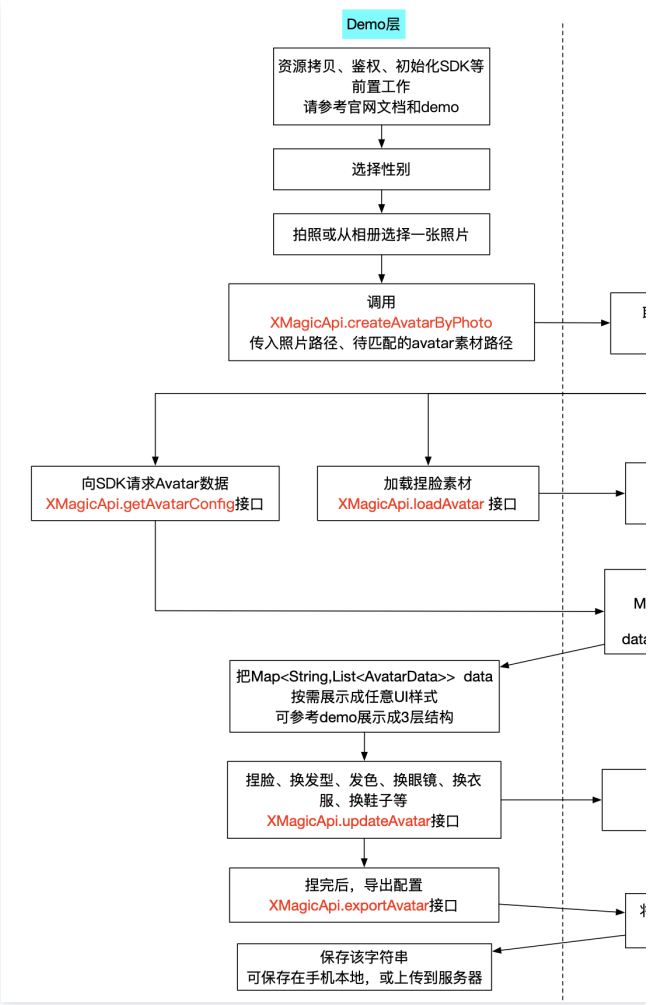
目前我们随 SDK 提供了若干套捏脸、换装素材，素材在 SDK 解压后的 `MotionRes/avatarRes` 目录中，与其他的动效素材一样，您需要把它 copy 到工程的 assets 目录：




## 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;
}
```

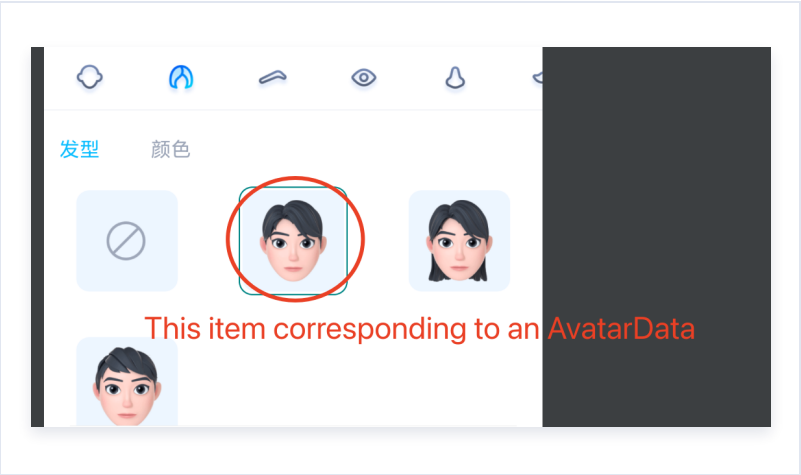
## 7. 调用 AvatarData

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。此类主要包含一下信息

```
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。

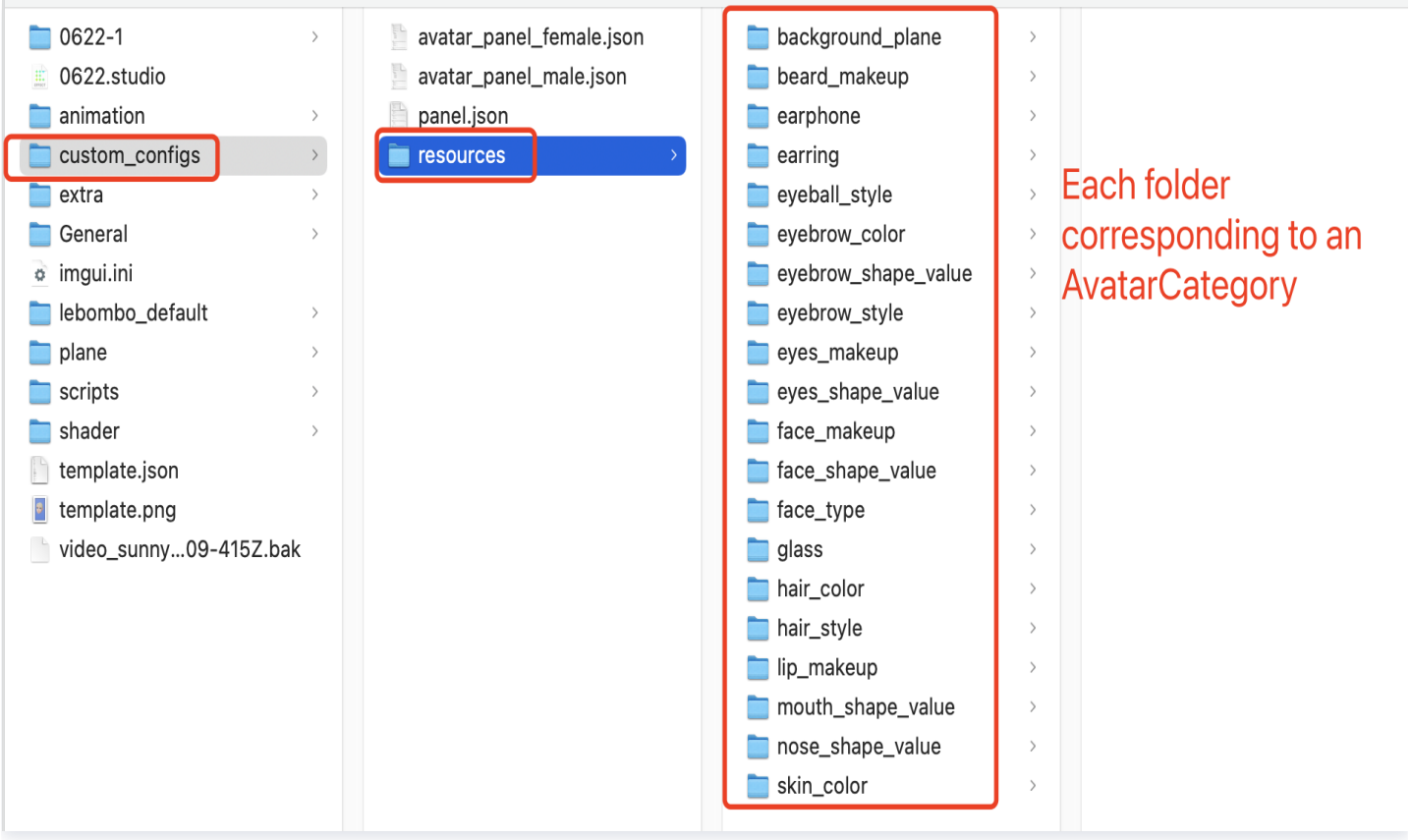
## action 和 value 字段

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 。