腾讯特效 SDK

集成指引

腾讯云

文档目录

集成指引
移动端/ PC 端美颜特效
SDK 集成指引（含 UI）
通用集成美颜特效
Android

最近更新时间：2025-02-28 12:24:13

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 集成美颜特效 。

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：**
> 此库只支持美颜特效 SDK V3.5.0及以上版本。

第一步：添加美颜特效 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..json	美体配置文件
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

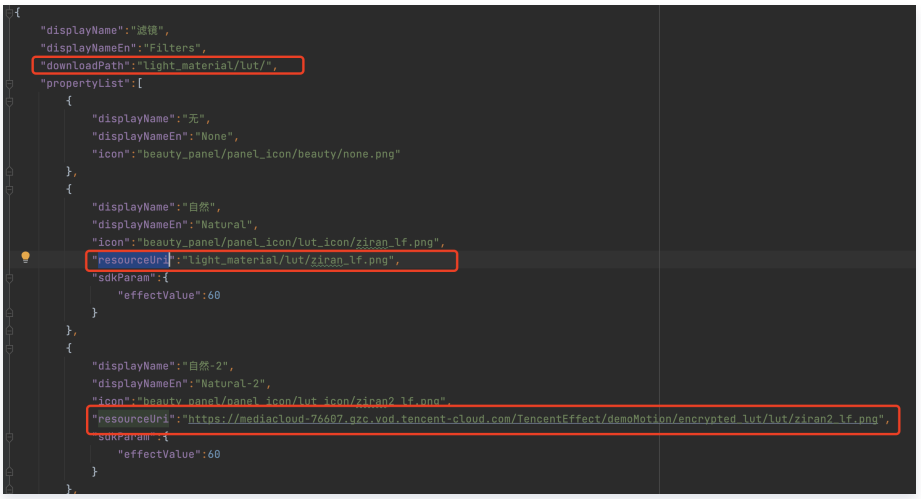
- **美颜、美体 。**

```
{
    "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 。**



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**

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：2025-02-28 12:24:13

功能说明

TEBeautyKit 是美颜特效美颜模块的 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_06,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;
}
```

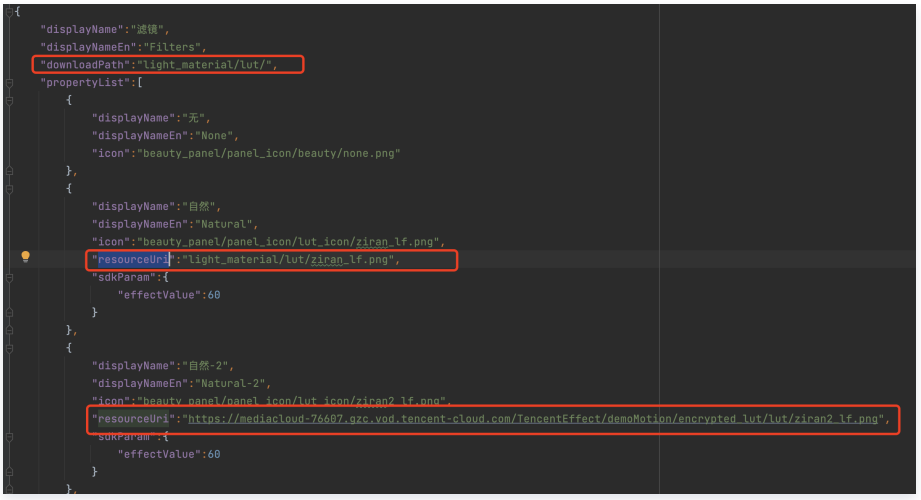
附录

面板 JSON 文件说明

- 美颜、美体 。

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 。**



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**

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_06,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 集成美颜特效
iOS

最近更新时间：2024-10-11 17:41:41

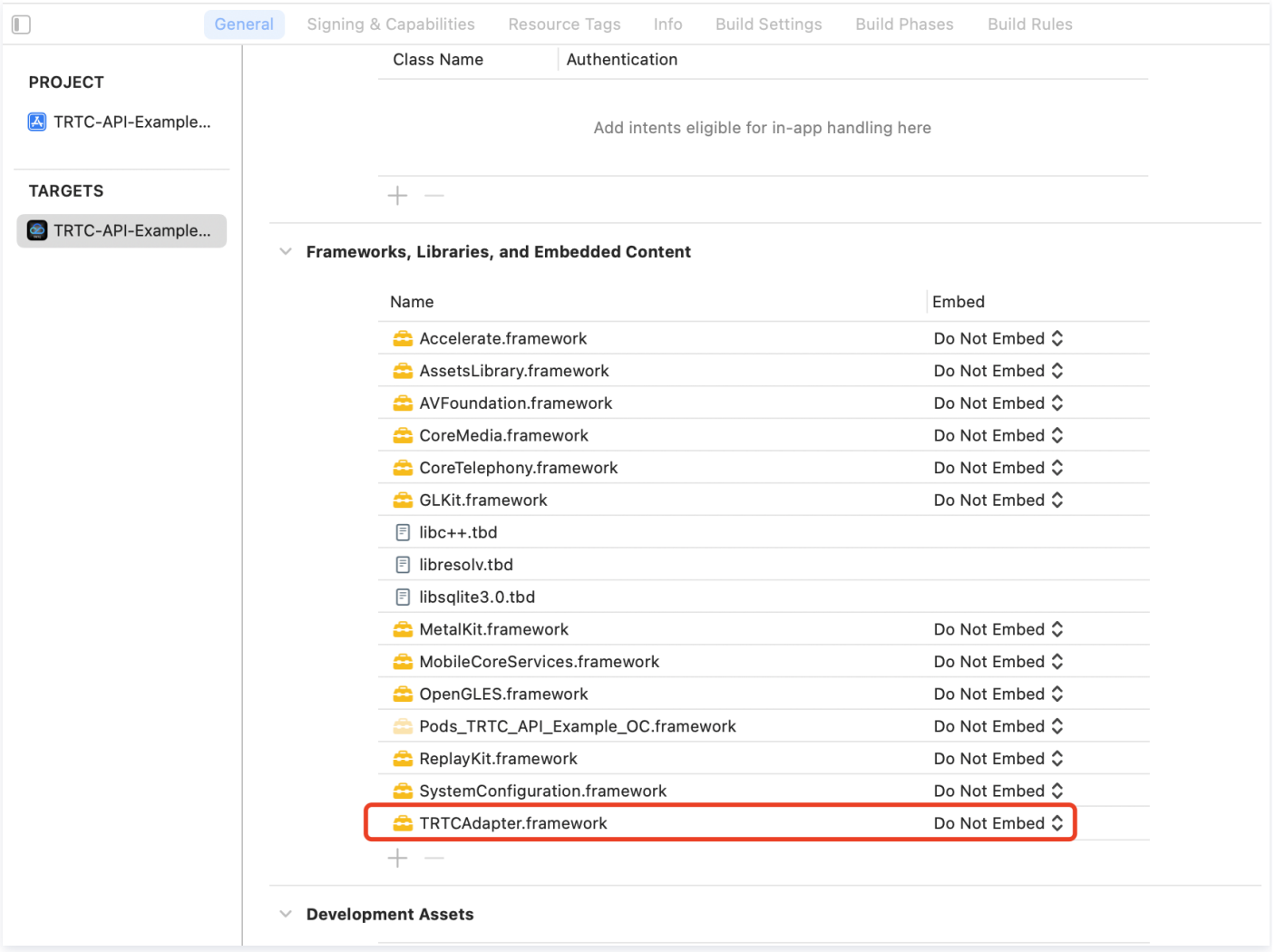
SDK 集成

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：2025-02-28 12:24:13

SDK 集成

1. 集成美颜特效
   - 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.mXMagicLicenceUrl,LicenseConstant.mXMagi
cKey, (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

V3.9.0之后

```
this.beautyAdapter.bind(this, this.mTRTCCloud, new ITEBeautyAdapter.CallBack() {
    @Override
    public void onCreatedTEBeautyApi(XmagicApi xmagicApi) {
        Log.e(TAG, "onCreatedTEBeautyApi  ");
```

```
        mBeautyKit = new TEBeautyKit(xmagicApi);
        mPanelView.setupWithTEBeautyKit(mBeautyKit);
        setTipListener(mBeautyKit);
        setLastParam(mBeautyKit);
    }

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

V3.9.0及之前

```
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：2025-03-03 17:11:42

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，美颜特效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：

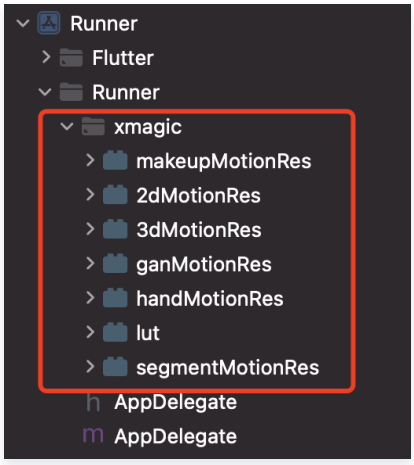
```
<application>
    <uses-native-library
        android:name="libOpenCL.so"
        android:required="false" />
</application>
```

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

iOS

将 demo 工程中 ios/Runne r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：

```objc
#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

```dart
void _initSettings(InitXmagicCallBack callBack) async {
  String resourceDir = await ResPathManager.getResManager().getResPath();
  TXLog.printlog('$TAG method is _initResource ,xmagic resource dir is $resourceDir');
  TencentEffectApi.getApi()?.setResourcePath(resourceDir);
  /// 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 _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/关闭美颜

```
///开启美颜操作
/// 设置ture 表示开启美颜，设置false 表示关闭美颜
 var enableCustomVideo = await trtcCloud.enableCustomVideoProcess(open);
```

注意：在页面中开启美颜，关闭相机的时候需要先关闭美颜，开启和关闭是成对使用。

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。

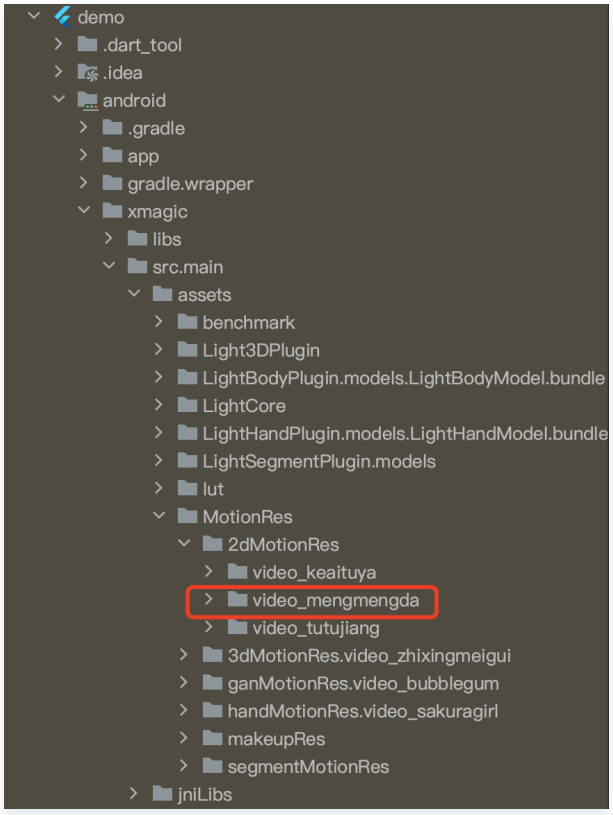
7. 添加和删除美颜面板上的美颜数据

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：

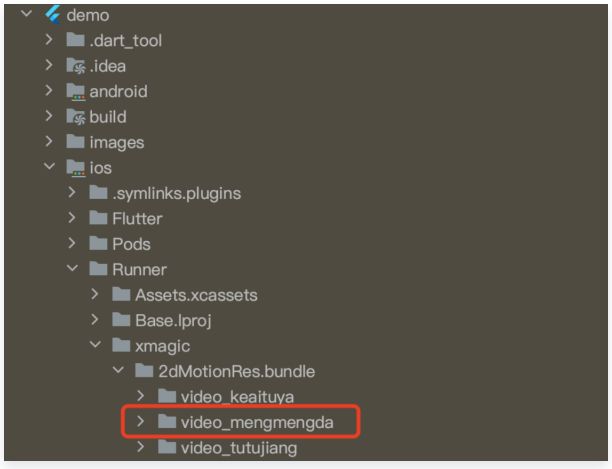
Android

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



iOS

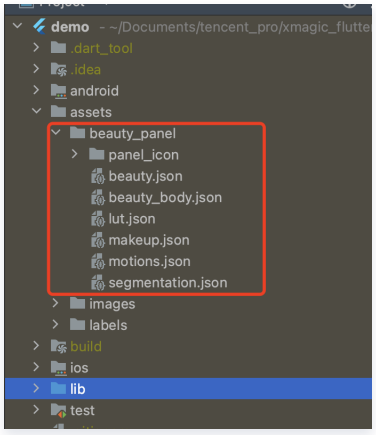
1. 并且把资源添加到工程的 `ios/Runner/xmagic/2dMotionRes.bundle` 目录下。



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。

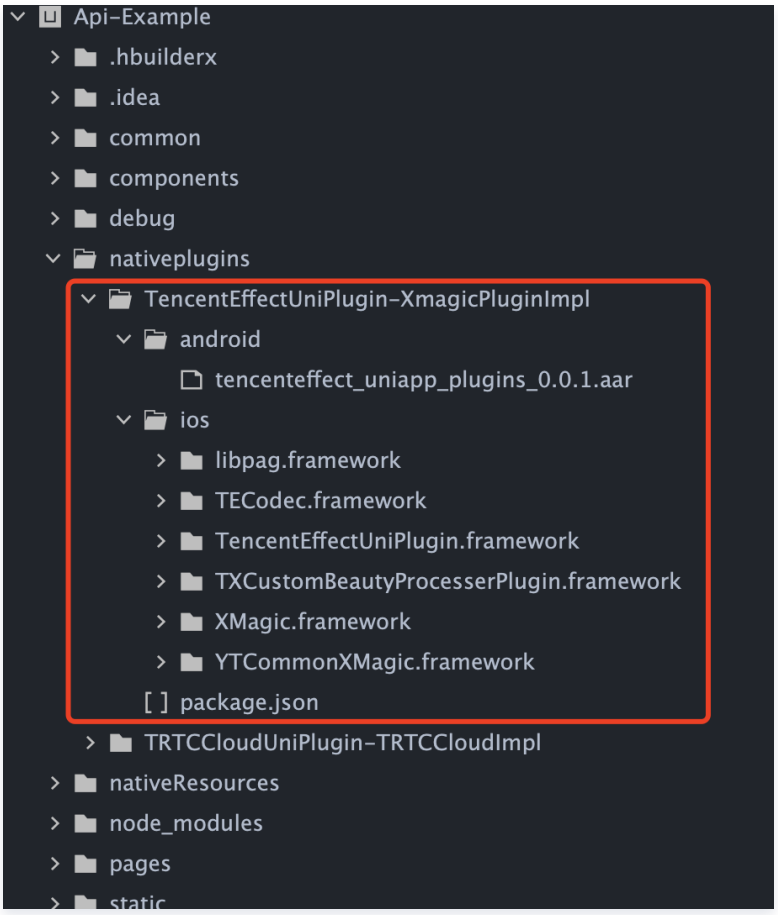
```
// Map<String, String> lipsResPathNames = {};
// lipsResPathNames["lips_fuguhong.png"] = "复古红";
// lipsResPathNames["lips_mitaose.png"] = "蜜桃色";
// lipsResPathNames["lips_shanhuju.png"] = "珊瑚橘";
// lipsResPathNames["lips_wenroufen.png"] = "温柔粉";
// lipsResPathNames["lips_huolicheng.png"] = "活力橙";
// 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：2025-02-28 12:24:13

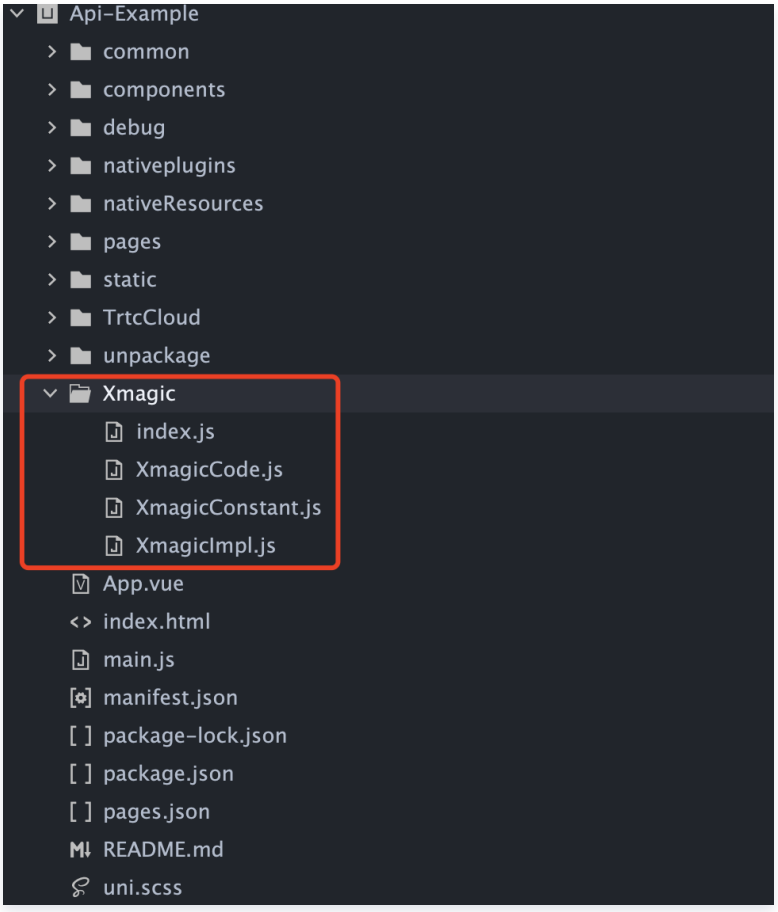
集成指引

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。

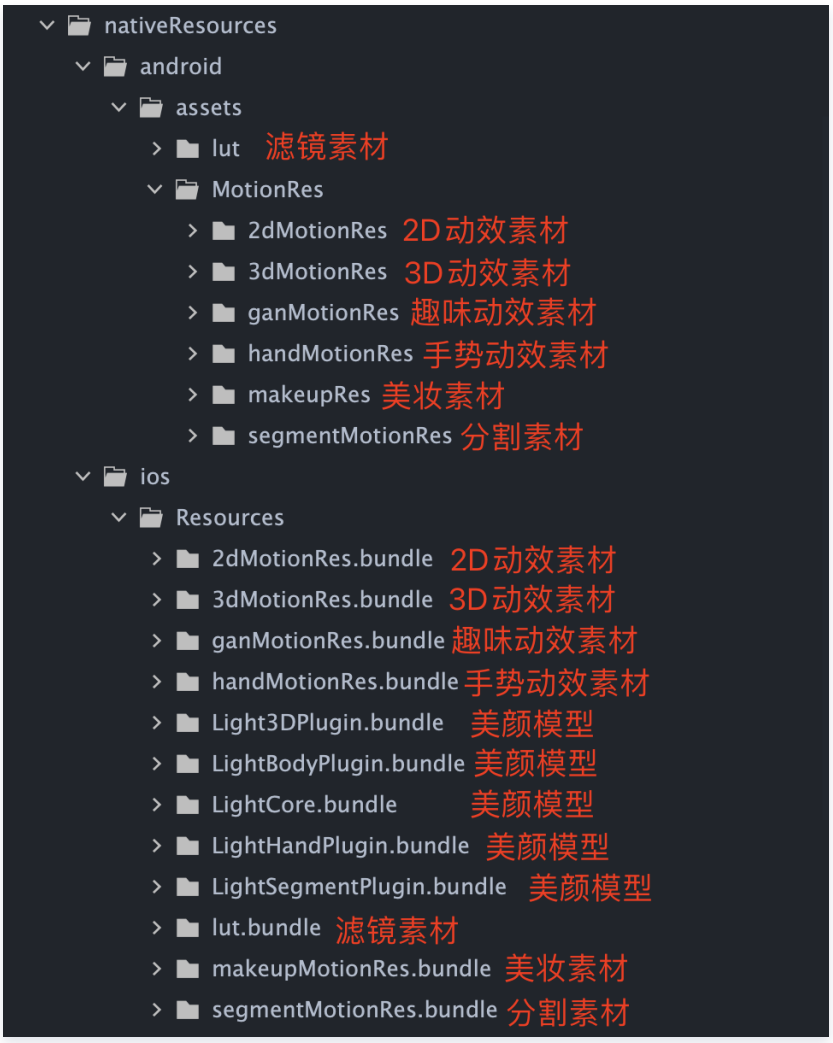
4. **把 SDK/js 目录下的 XMagic 文件夹拷贝到自己项目工程中。**



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 ）

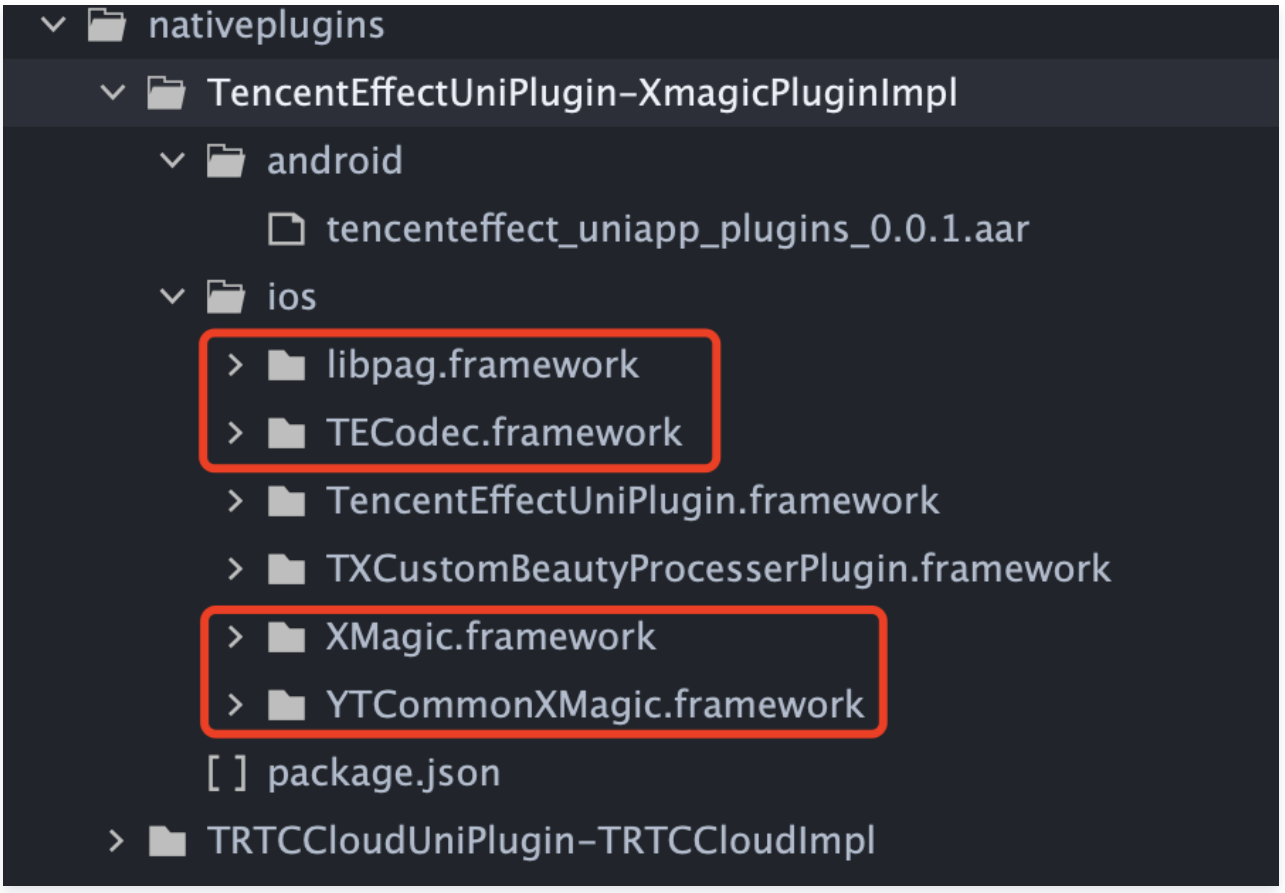


6. **添加美颜特效 SDK**

   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` **字段中美颜特效 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：下载美颜特效 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 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 集成美颜特效
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 。

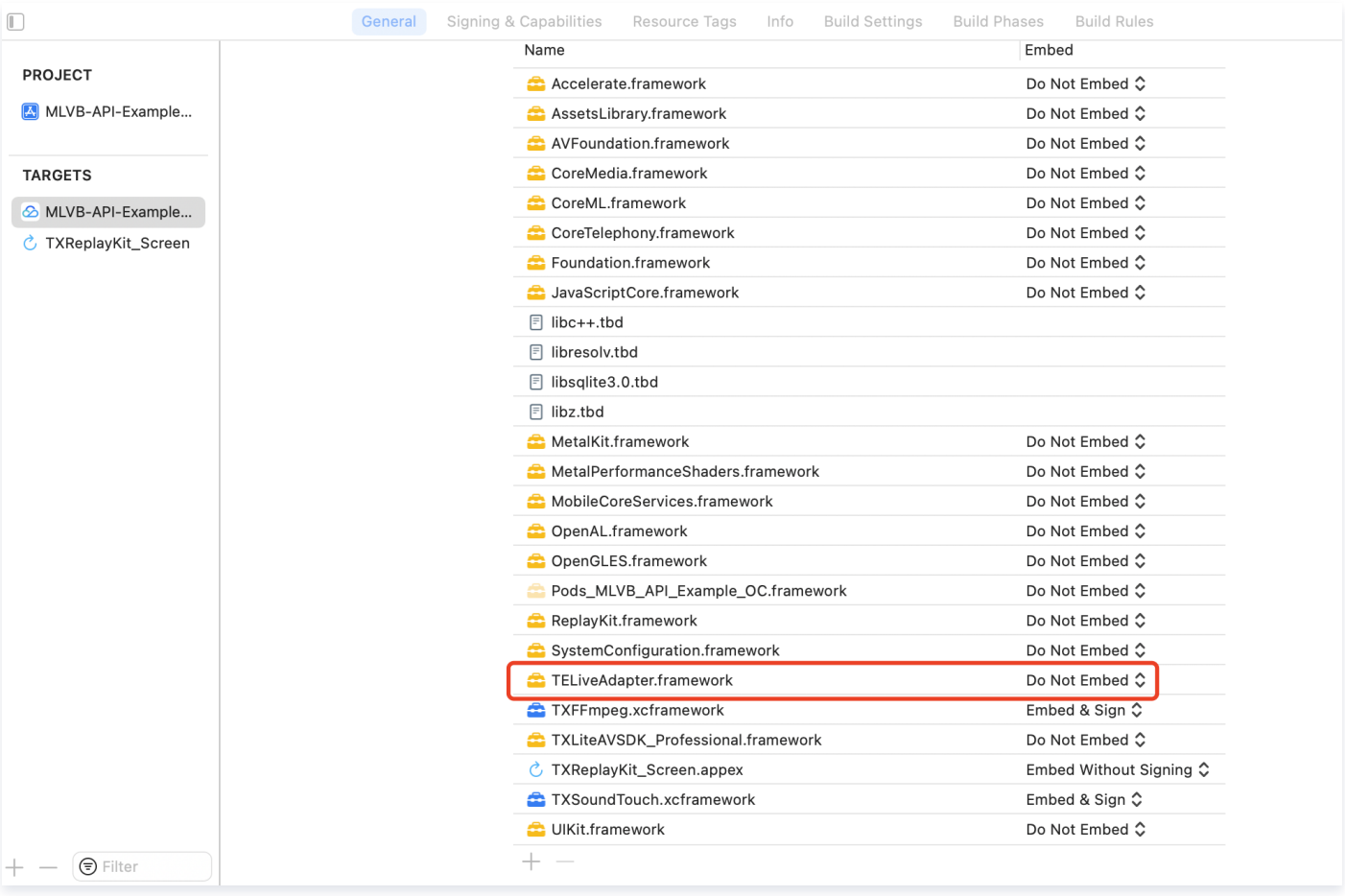
SDK 使用

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：2025-02-28 12:24:13

SDK 集成

1. 集成美颜特效
   - 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.mXMagicLicenceUrl,LicenseConstant.mXMagi
cKey, (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_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.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

V3.9.0之后

```
this.beautyLiveAdapter.bind(this, mLivePusher, new ITEBeautyAdapter.CallBack() {
    @Override
    public void onCreatedTEBeautyApi(XmagicApi xmagicApi) {
        mBeautyKit = new TEBeautyKit(xmagicApi);
        tePanelView.setupWithTEBeautyKit(mBeautyKit);
        setTipListener(mBeautyKit);
```

```
            setLastParam(mBeautyKit);
            Log.e("beautyLiveAdapter", "onCreatedTEBeautyKit");
        }

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

V3.9.0及之前

```
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：2025-03-03 17:11:42

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：

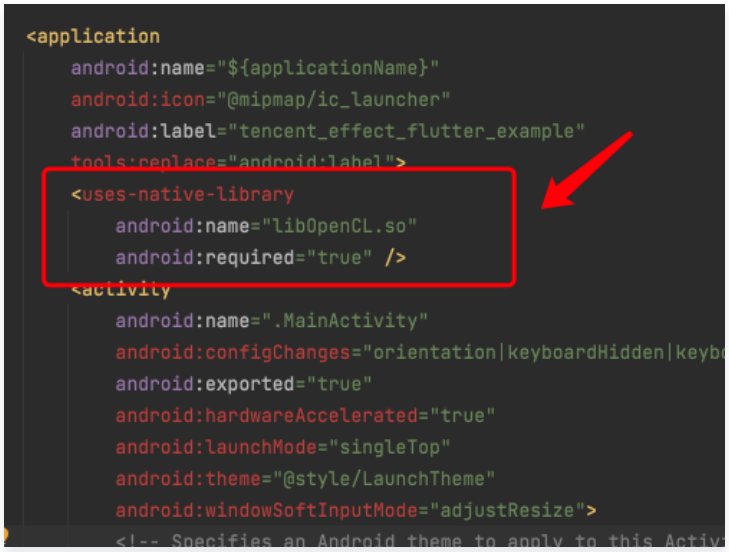
```
dependencies {
    implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

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，美颜特效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：

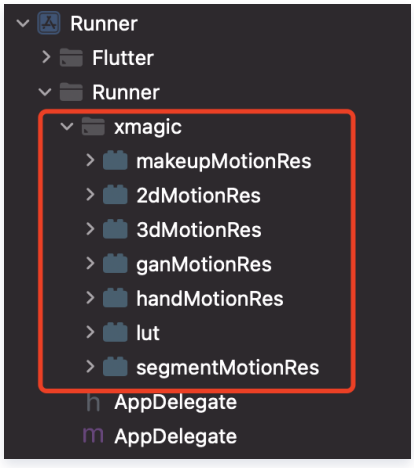


4. 混淆配置
   - 如果您在打 release 包时，启用了编译优化（把 minifyEnabled 设置为 true），会裁掉一些未在 java 层调用的代码，而这些代码有可能会被 native 层调用，从而引起 `no xxx method` 的异常。
   - 如果您启用了这样的编译优化，那就要添加这些 keep 规则，防止 xmagic 的代码被裁掉：

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

iOS

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 {
  String resourceDir = await ResPathManager.getResManager().getResPath();
  TXLog.printlog('$TAG method is _initResource ,xmagic resource dir is $resourceDir');
  TencentEffectApi.getApi()?.setResourcePath(resourceDir);
  /// 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 _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/关闭美颜

```
///开启美颜操作
/// 设置ture 表示开启美颜，设置false 表示关闭美颜
 var enableCustomVideo = await trtcCloud.enableCustomVideoProcess(open);
```

注意：在页面中开启美颜，关闭相机的时候需要先关闭美颜，开启和关闭是成对使用。

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。

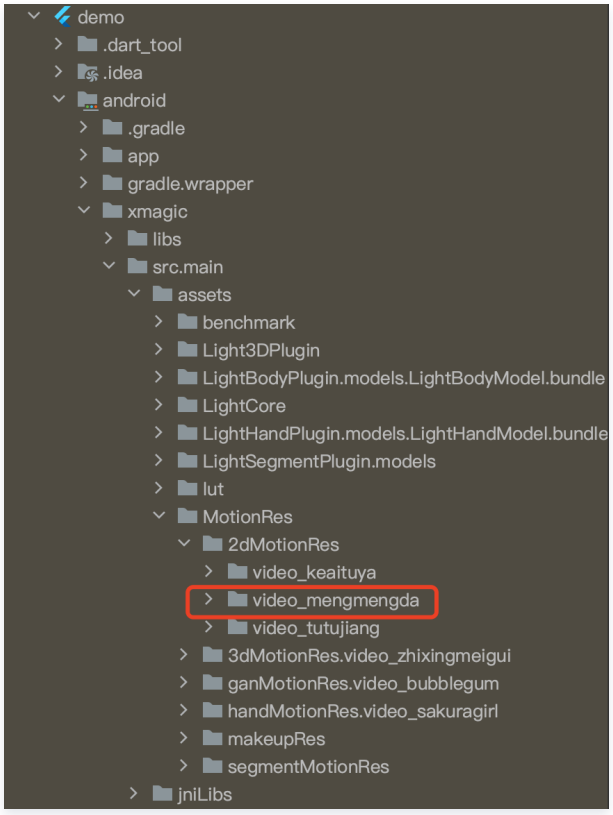
7. 添加和删除美颜面板上的美颜数据

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：

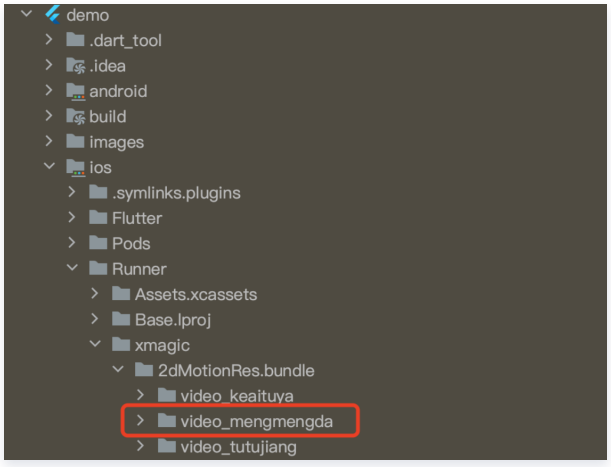
Android

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



iOS

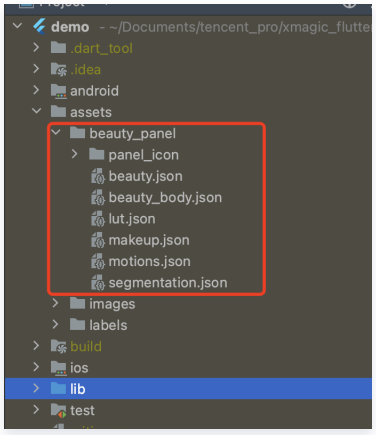
并且把资源添加到工程的 `ios/Runner/xmagic/2dMotionRes.bundle` **目录下。**



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 集成美颜特效
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：2025-02-28 12:24:13

步骤一：解压 Demo 工程

1. 下载集成了美颜特效 TE 的 UGSV Demo 工程。本 Demo 基于美颜特效 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，那么需要在美颜特效 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

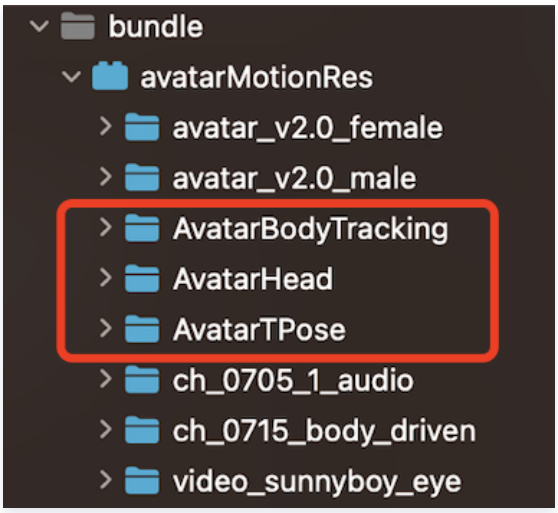
最近更新时间：2024-08-20 14:07:41

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：2025-02-28 12:24:13

由于 Avatar 是美颜特效的部分功能，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 独立集成美颜特效 进行了解与集成。

步骤1：准备 Avatar 素材

1. 集成美颜特效 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;
}
```

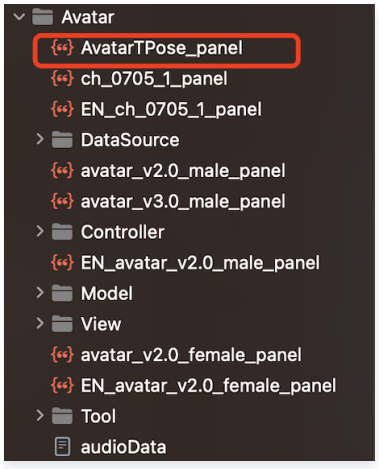
自定义 Avatar UI

最近更新时间：2024-08-20 14:07:41

Demo UI 说明

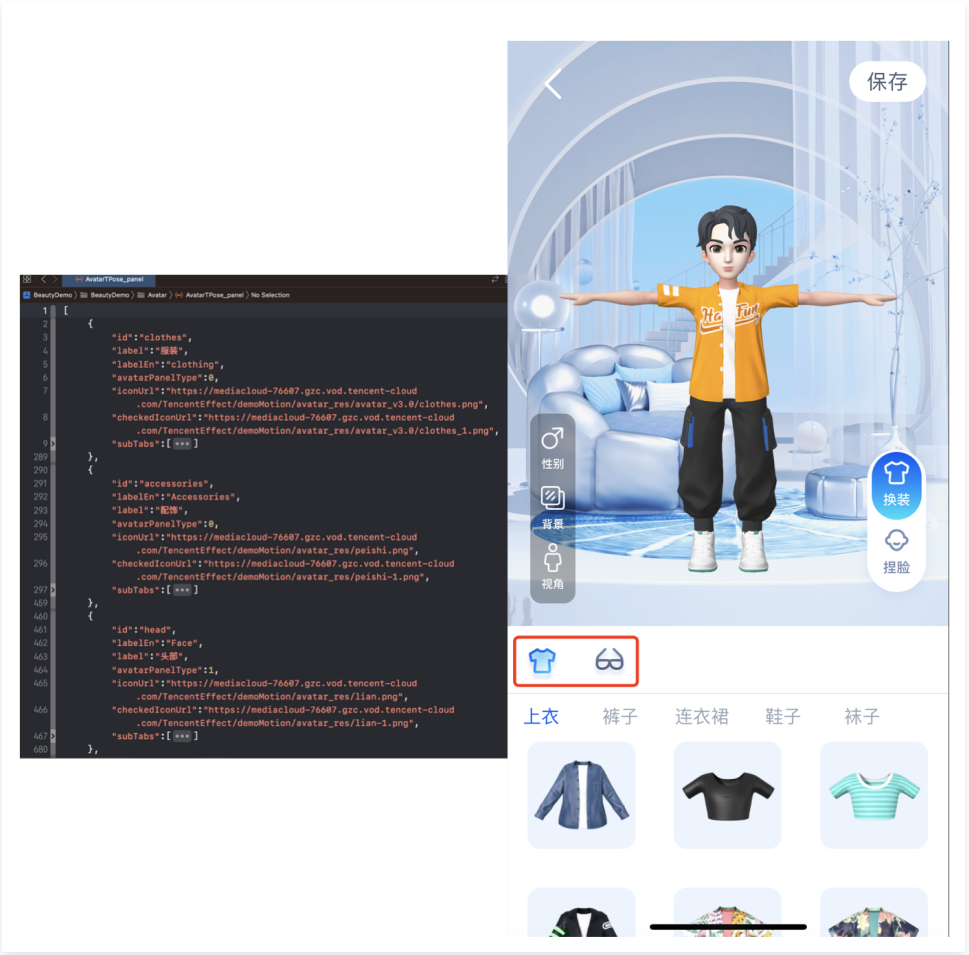
实现方式

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。

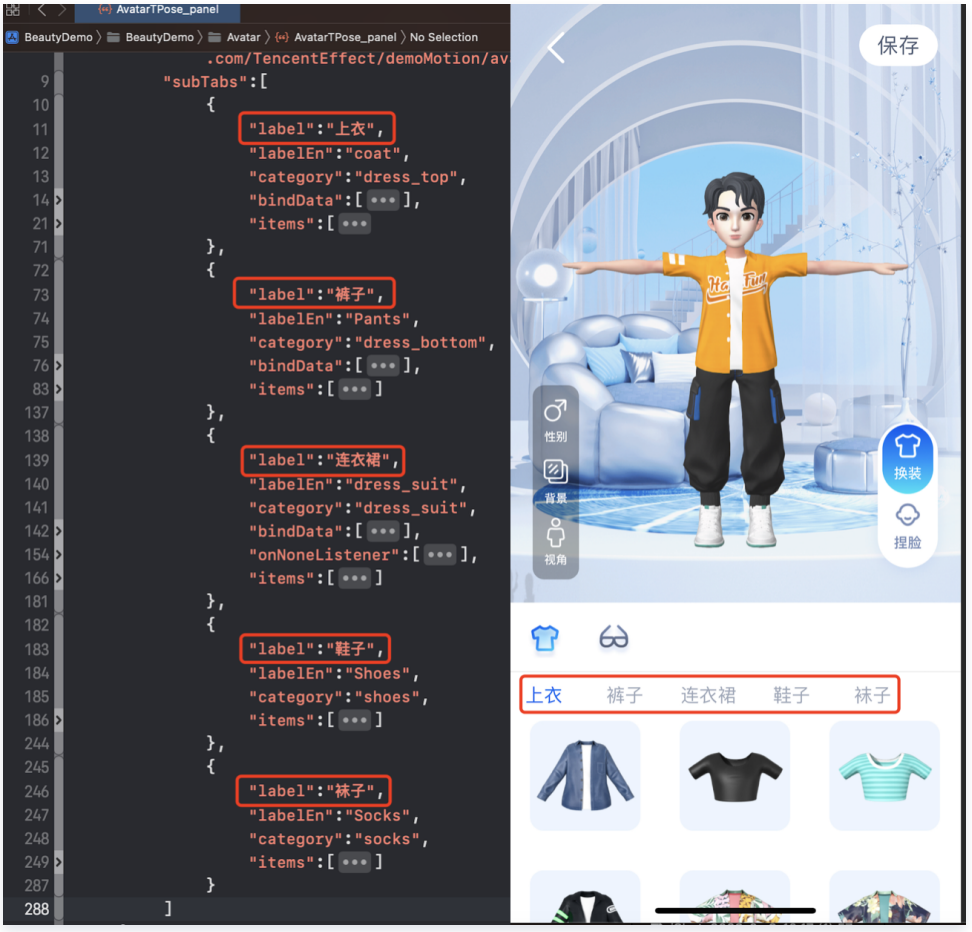


Json 结构和 UI 面板对应关系：

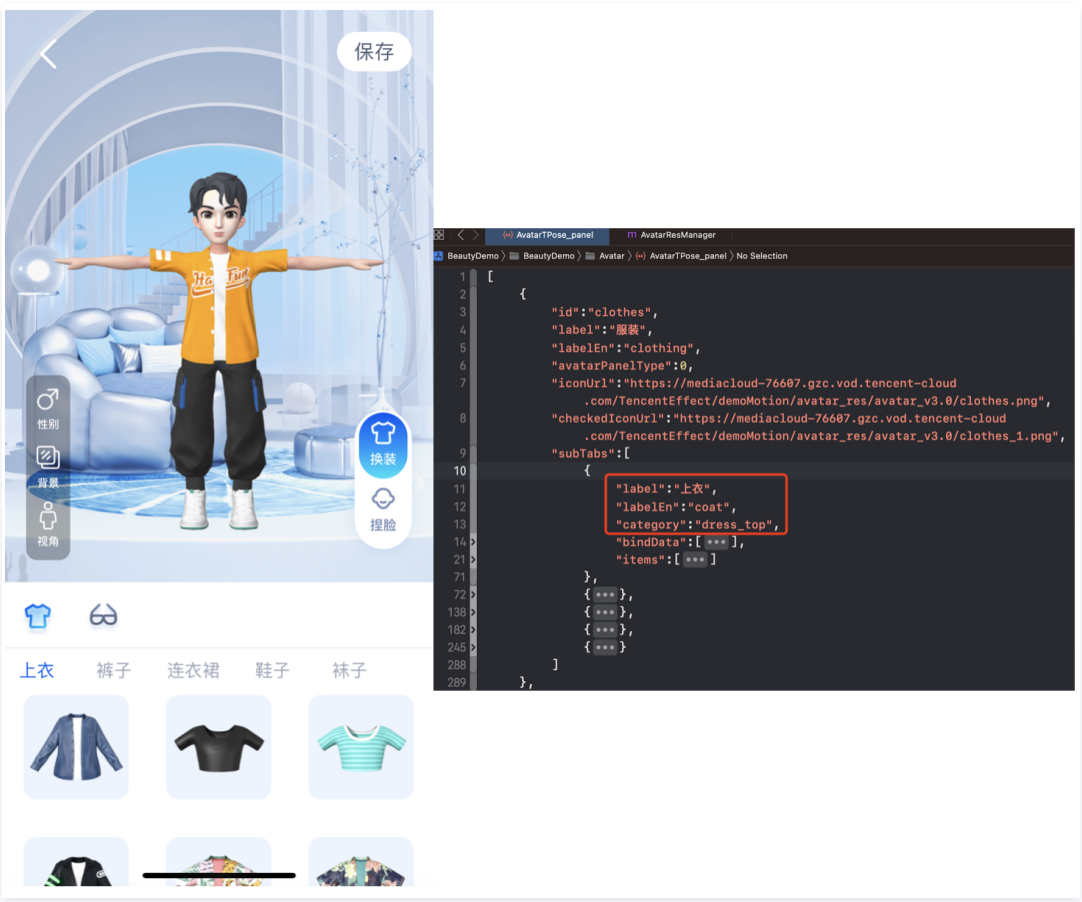
- 左侧 item 对应右侧页面一级菜单，clothes 为第一个 icon 选中的内容：



- 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

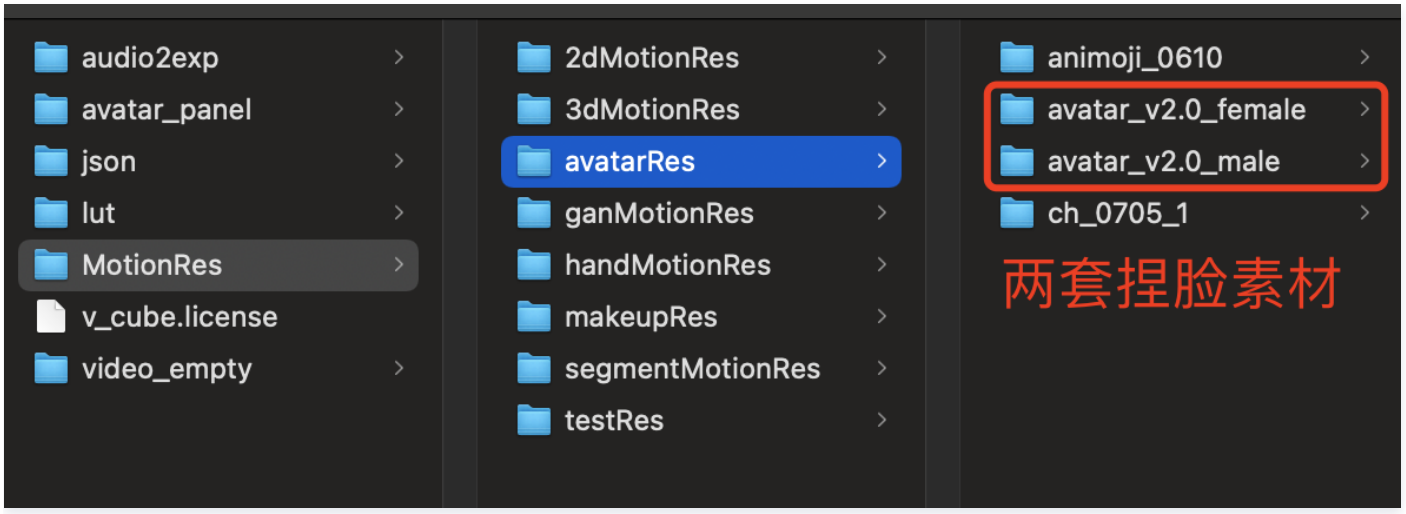
最近更新时间：2025-02-28 12:24:13

SDK 接入

SDK 的下载、接入、鉴权、跑通 Demo 请参见 独立集成美颜特效 。

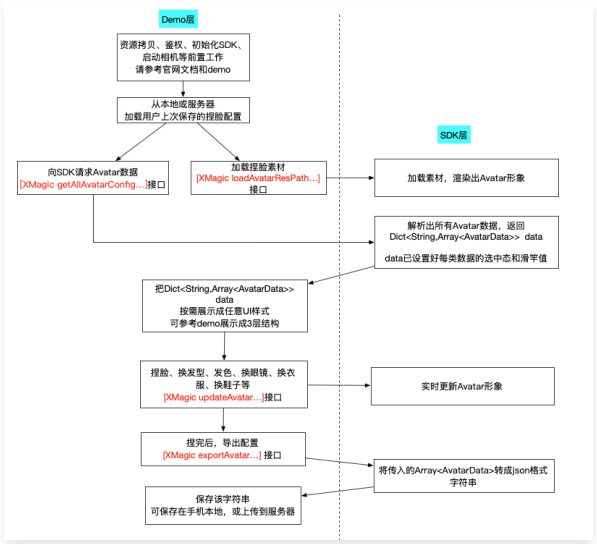
准备捏脸素材

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：

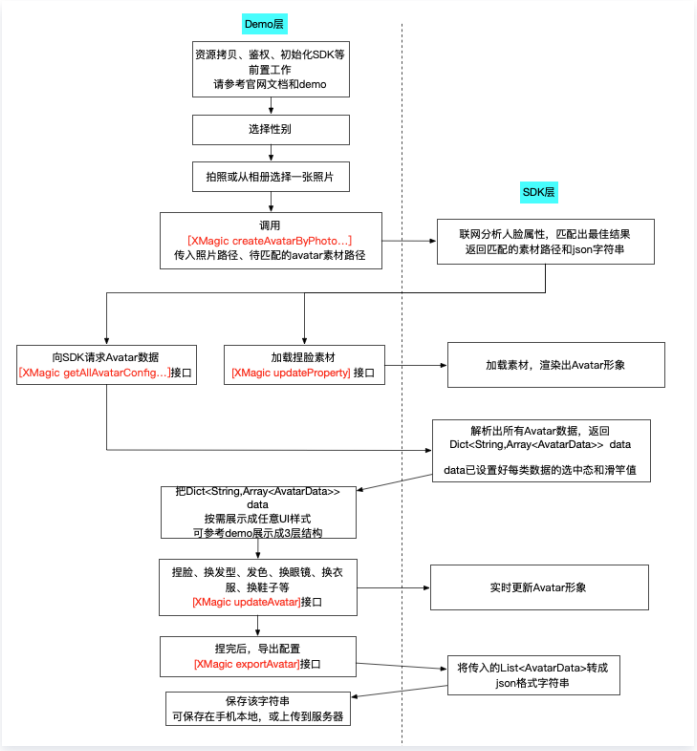


捏脸流程与 SDK 接口

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}"`。

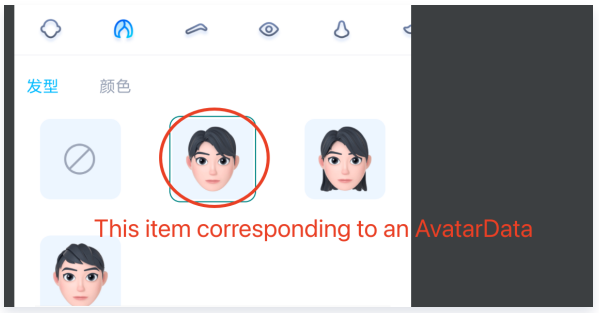
8. 调用 AvatarData

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。

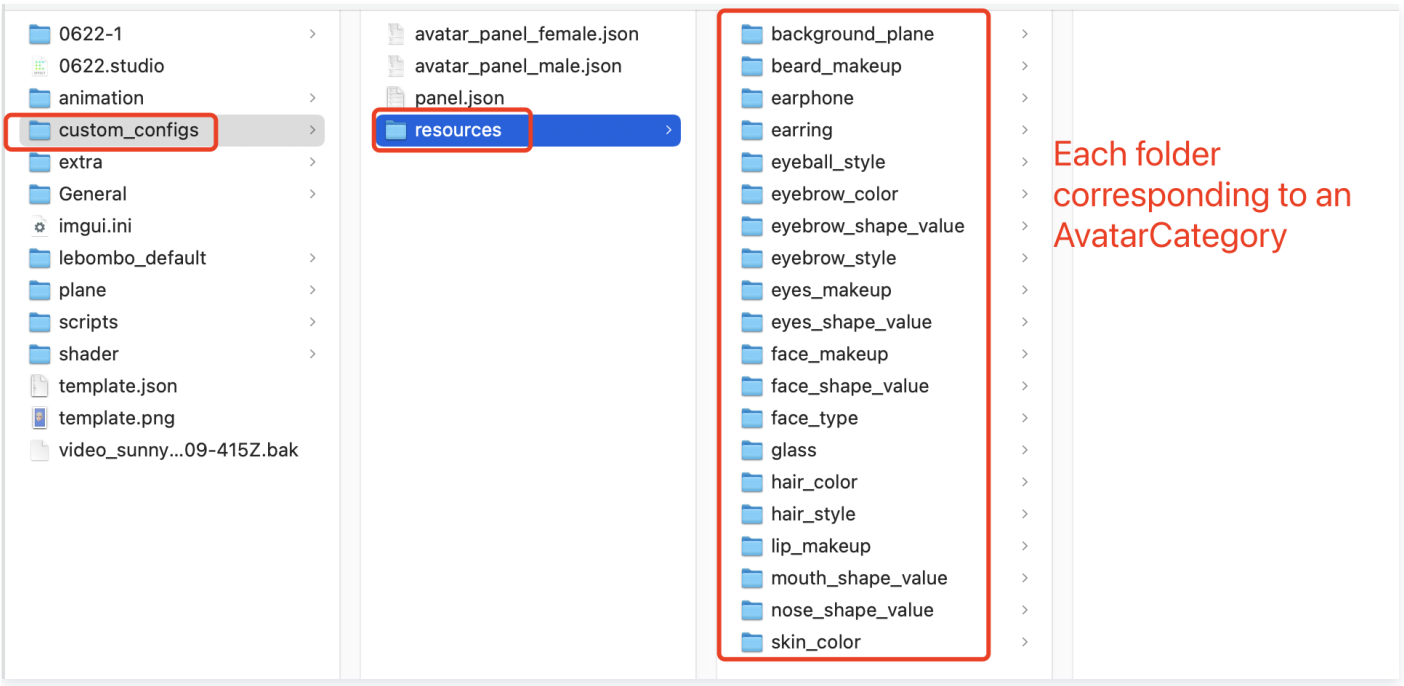
action 和 value字段

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`）：



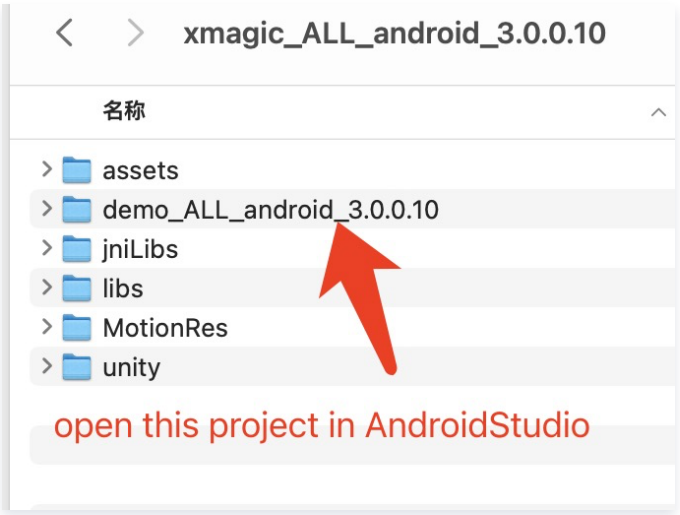
这些配置文件是自动生成的，通常不需要手动配置。自动生成的方式如下：

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 。

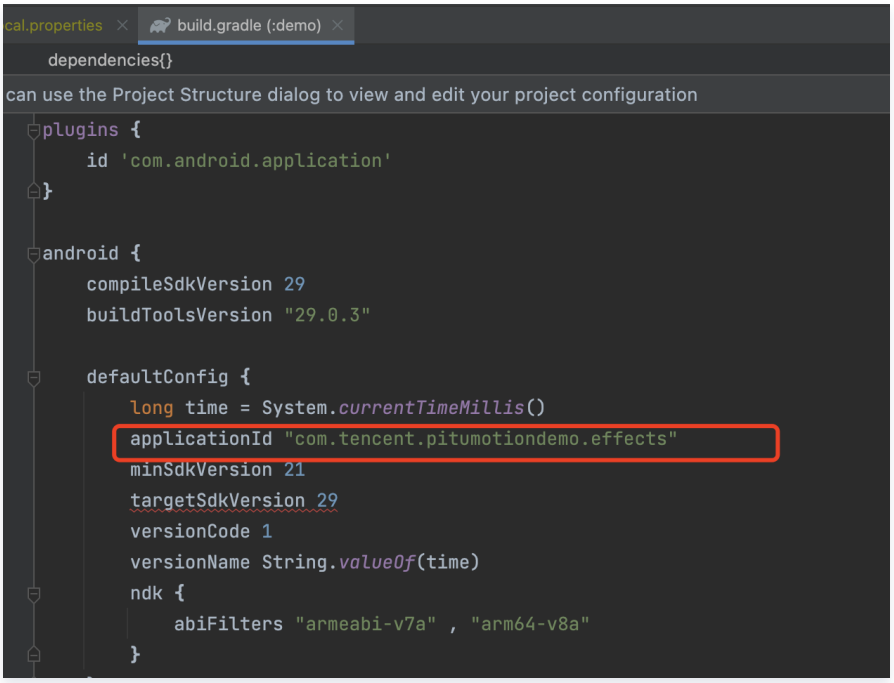
Android
快速跑通 Demo

最近更新时间：2024-02-19 11:42:11

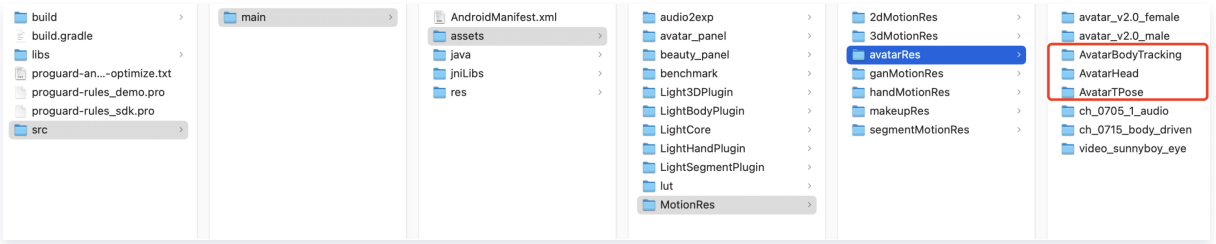
1. 使用 Android Studio 打开项目工程。



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。



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：2025-02-28 12:24:13

由于 Avatar 只是美颜特效的部分功能，所以在接入时需参考 美颜特效接入文档，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. 按照美颜特效文档进行接入，请参见 独立集成美颜特效 。
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

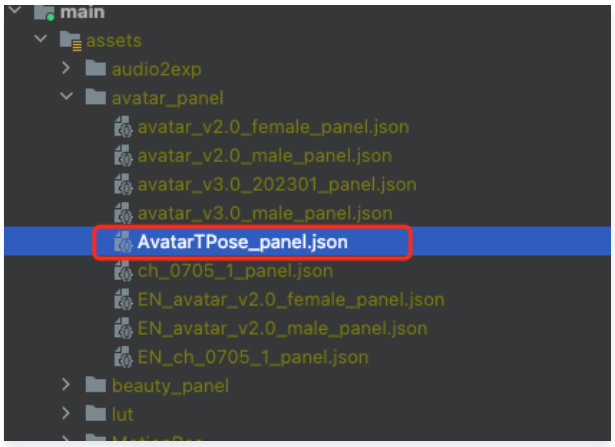
自定义 Avatar UI

最近更新时间：2023-05-17 17:08:14

Demo UI 说明

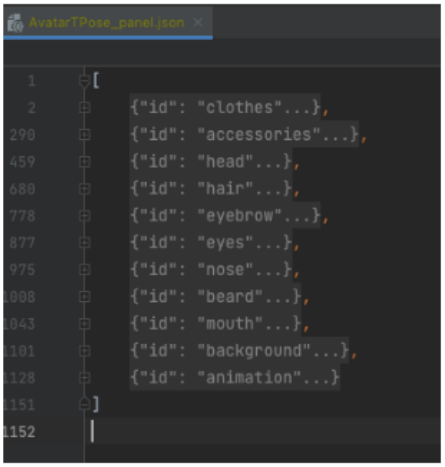
实现方式

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



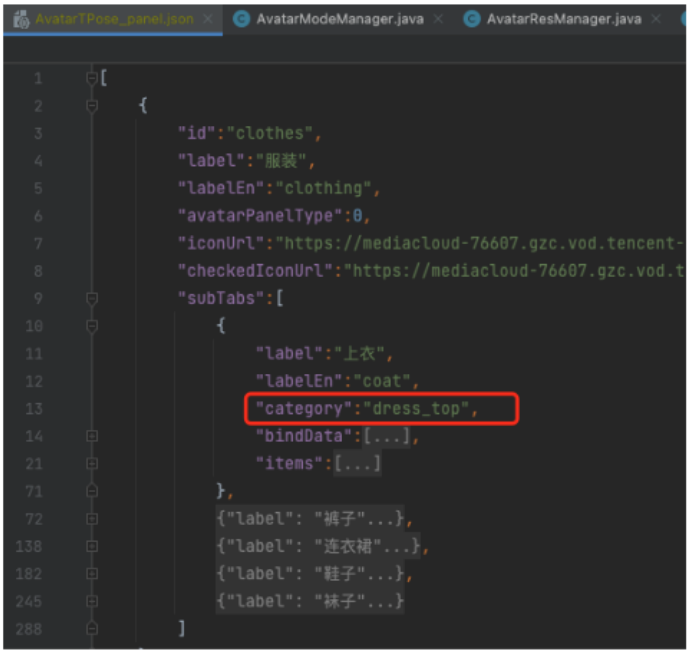
Json 结构和 UI 面板对应关系：

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



- 左侧红框 subTabs 对应右侧二级菜单：

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



Demo 重要类说明

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

1. 加载 Avatar 资源

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

2. 获取面板数据

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

3. 从面板数据中解析出用户设置的属性或默认属性

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

4. 从 bindData 中解析出对应的 avatarData

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

附录

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

MainTab

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

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

SubTab

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

AvatarItem

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

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

BindData

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

Avatar SDK 说明

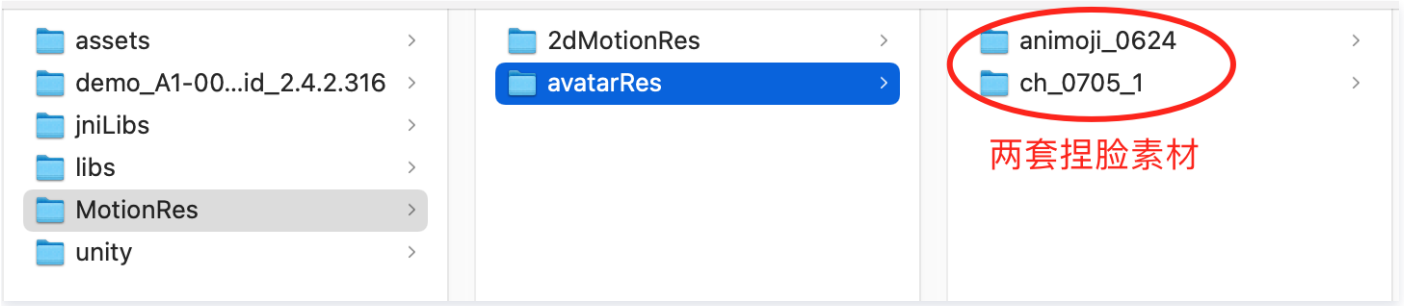
最近更新时间：2025-02-28 12:24:13

SDK 接入

SDK 的下载、接入、鉴权、跑通 Demo 请参见 独立集成美颜特效 。

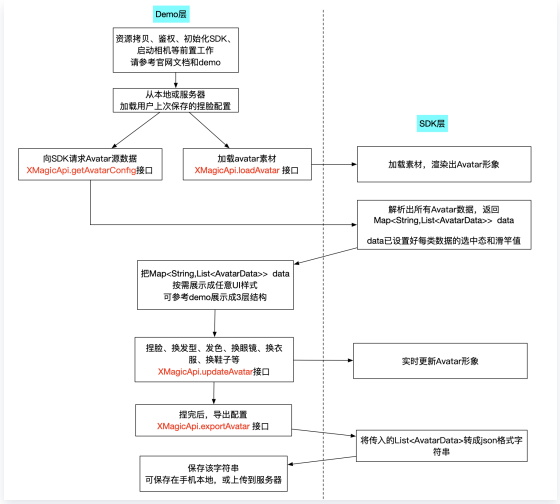
准备捏脸素材

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：

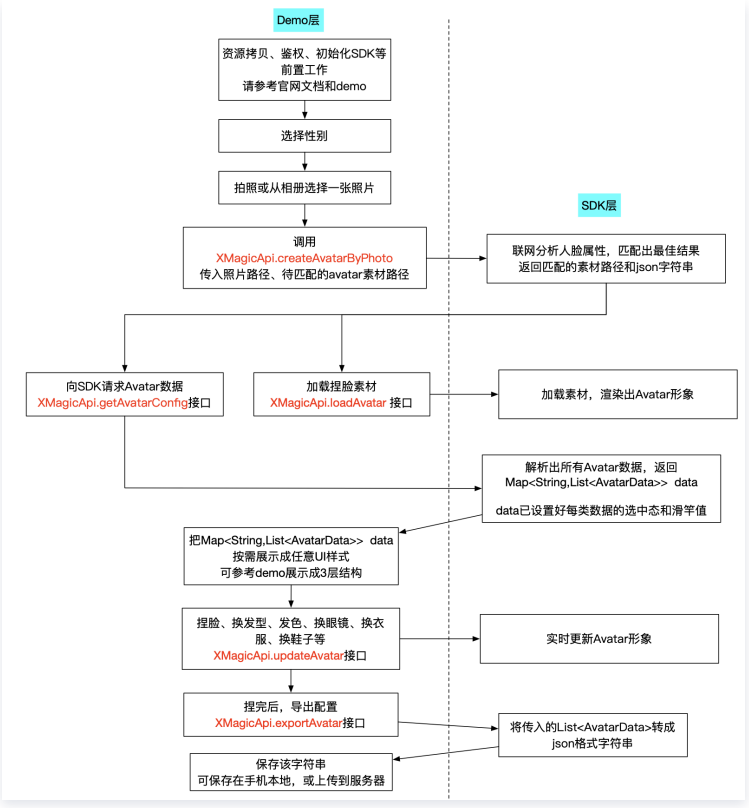


捏脸流程与 SDK 接口

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;
}
```

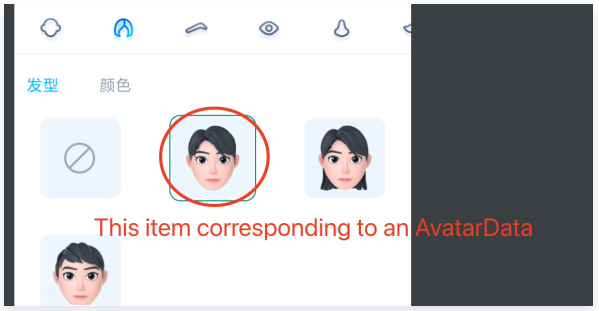
7. 调用 AvatarData

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

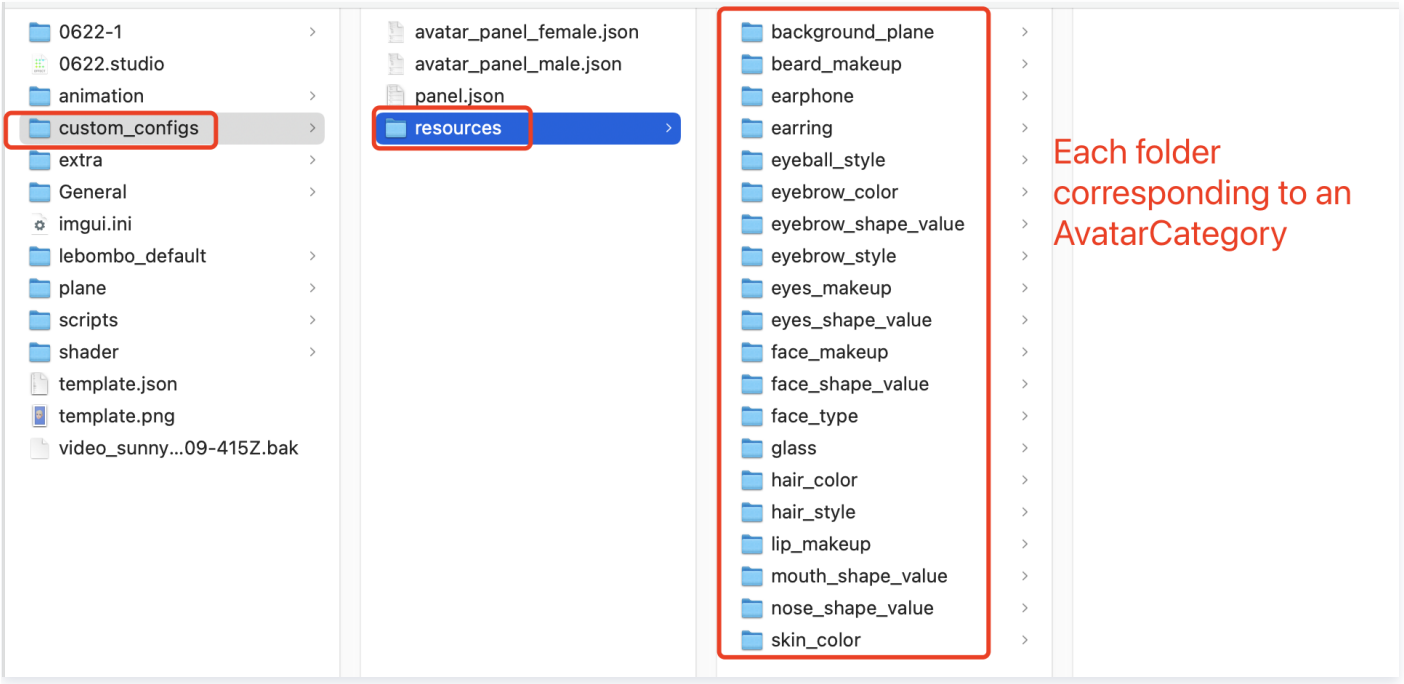
action 字段表示对 entityName 执行什么操作（action）。SDK 内定义了五种 action，均定义在 `AvatarAction.java` 中，每种 action 的含义及 value 要求如下：

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 。

SDK 集成指引（无 UI）
通用集成美颜特效
Android

最近更新时间：2025-02-28 12:24:13

开发者环境要求

- 最低兼容 Android 4.4（SDK API Level 19），建议使用 Android 7.0（SDK API Level 24）及以上版本。
- Android Studio 3.5 及以上版本。

Demo 工程： TEBeauty_API_Example

从 github clone 出 Demo 工程，其中的 TEBeautyDemo 是含 UI 的 demo 工程，TEBeauty_API_Example 是不含 UI 的 demo 工程。
按照 TEBeauty_API_Example/README 文档中的指引将 TEBeauty_API_Example 运行起来，然后结合本文了解无UI 集成 SDK 的详细步骤。

集成SDK

注意：
Github 上的 demo 工程采用 Maven 方式集成 SDK。

Maven 集成

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

```
dependencies {
 //例如：S1-04套餐如下：
 implementation 'com.tencent.mediacloud:TencentEffect_S1-04:版本号'
//“版本号”可以在官网的“SDK下载/版本历史”页面看到，例如 3.0.0.13。“版本号”也可以使用"latest.release" ，
//但请注意：这会让您使用的SDK始终保持最新版，在一些变化比较大的版本上可能不符合您的预期，请慎重使用"latest.release"
}
```

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

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

说明：
目前特效 SDK 支持 armeabi-v7a 和 arm64-v8a。

3. 单击 Sync Now，自动下载 SDK 并集成到工程里。

各套餐对应的 Maven 地址

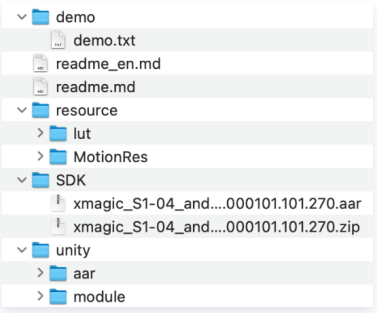
版本	Maven 地址
A1 – 01	implementation 'com.tencent.mediacloud:TencentEffect_A1-01:版本号'
A1 – 02	implementation 'com.tencent.mediacloud:TencentEffect_A1-02:版本号'
A1 – 03	implementation 'com.tencent.mediacloud:TencentEffect_A1-03:版本号'

A1 – 04	implementation 'com.tencent.mediacloud:TencentEffect_A1-04:版本号'
A1 – 05	implementation 'com.tencent.mediacloud:TencentEffect_A1-05:版本号'
A1 – 06	implementation 'com.tencent.mediacloud:TencentEffect_A1-06:版本号'
S1 – 00	implementation 'com.tencent.mediacloud:TencentEffect_S1-00:版本号'
S1 – 01	implementation 'com.tencent.mediacloud:TencentEffect_S1-01:版本号'
S1 – 02	implementation 'com.tencent.mediacloud:TencentEffect_S1-02:版本号'
S1 – 03	implementation 'com.tencent.mediacloud:TencentEffect_S1-03:版本号'
S1 – 04	implementation 'com.tencent.mediacloud:TencentEffect_S1-04:版本号'
S1 – 05	implementation 'com.tencent.mediacloud:TencentEffect_S1-05:版本号'
S1 – 06	implementation 'com.tencent.mediacloud:TencentEffect_S1-06:版本号'
S1 – 07	implementation 'com.tencent.mediacloud:TencentEffect_S1-07:版本号'

手动集成（资源内置）

下载 SDK

下载 SDK，并解压。目录结构如下：



集成

将 `SDK` 文件夹下的 `xmagic-xxxx.aar` 文件拷贝到您工程 `libs` 目录下。

导入方法

打开 app 模块的 `build.gradle` 添加依赖引用：

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

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

```
}
```

注意：
项目中还需添加如下依赖：

```
dependencies{
    implementation 'com.google.code.gson:gson:2.8.2'
    //版本在2.6.0到3.1.0.2 需要添加
    implementation 'androidx.exifinterface:exifinterface:1.3.3'
    //3.5.0 版本及之后需要添加
    implementation 'com.tencent.tav:libpag:4.3.33-noffavc'
}
```

如果您想使用其他版本的 pag，请点击 此处 查看。

手动集成（资源动态下载）

动态下载 assets、so、动效资源指引

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

集成素材

如果您的套餐包含动效和滤镜功能，那么需要在 SDK 下载页面 下载对应的套餐包，解压之后，将 `resource` 目录下的 `lut` 和 `MotionRes` 放置到您工程的 `assets` 目录下：

- 动效：`..src/main/assets/MotionRes`
- 滤镜：`..src/main/assets/lut`

更多素材配置可参考 素材使用指南。

注意：
SDK中包含的素材是测试素材，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。

SDK 使用流程

步骤一：鉴权

1. 申请授权，得到 License URL 和 License KEY，请参见 License 指引。
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。
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, null);
```

3. 然后在真正要使用美颜功能前，再去做鉴权：

```
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, new TELicenseCheckListener() {

        @Override
        public void onLicenseCheckFinish(int errorCode, String msg) {
            //注意：此回调不一定在调用线程
```

```
            if (errorCode == TELicenseCheck.ERROR_OK) {
                //鉴权成功
            } else {
                //鉴权失败
            }
        }
    });
```

鉴权 errorCode 说明：

错误码	说明
0	成功。Success
-1	输入参数无效，例如 URL 或 KEY 为空
-3	下载环节失败，请检查网络设置
-4	从本地读取的 TE 授权信息为空，可能是 IO 失败引起
-5	读取 VCUBE TEMP License文件内容为空，可能是 IO 失败引起
-6	v_cube.license 文件 JSON 字段不对。请联系腾讯云团队处理
-7	签名校验失败。请联系腾讯云团队处理
-8	解密失败。请联系腾讯云团队处理
-9	TELicense 字段里的 JSON 字段不对。请联系腾讯云团队处理
-10	从网络解析的 TE 授权信息为空。请联系腾讯云团队处理
-11	把TE授权信息写到本地文件时失败，可能是 IO 失败引起
-12	下载失败，解析本地 asset 也失败
-13	鉴权失败，请检查 so 是否在包里，或者已正确设置 so 路径
3004/3005	无效授权。请联系腾讯云团队处理
3015	Bundle Id / Package Name 不匹配。检查您的 App 使用的 Bundle Id / Package Name 和申请的是否一致，检查是否使用了正确的授权文件
3018	授权文件已过期，需要向腾讯云申请续期
其他	请联系腾讯云团队处理

步骤二：资源拷贝

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 app 的私有目录。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`

```
String resPath = new File(getFilesDir(),
AppConfig.getInstance().getBeautyFileDirName()).getAbsolutePath();
if (!resPath.endsWith(File.separator)) {
    resPath = resPath + File.separator;
}
AppConfig.resPathForSDK = resPath;
AppConfig.lutFilterPath = resPath + "light_material/lut";
AppConfig.motionResPath = resPath + "MotionRes";
```

```
new Thread(() -> {
    Context context = getApplicationContext();
    int addResult = XmagicApi.addAiModeFilesFromAssets(context, AppConfig.resPathForSDK);
    Log.d(TAG, "copyRes, add ai model files result = " + addResult);

    String lutDirNameInAsset = "lut";
    boolean result = FileUtil.copyAssets(context, lutDirNameInAsset, AppConfig.lutFilterPath);
    Log.d(TAG, "copyRes, copy lut, result = " + result);

    String motionResDirNameInAsset = "MotionRes";
    boolean result2 = FileUtil.copyAssets(context, motionResDirNameInAsset, AppConfig.motionResPath);
    Log.d(TAG, "copyRes, copy motion res, result = " + result2);
}).start();
```

步骤三：SDK 初始化及使用方法

1. 快速实现相机（可选）

   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：

```
@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，可以参见 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" />
```

2. 初始化美颜 SDK，可以在任意线程初始化，建议在子线程中进行。**创建时机需要在 License 鉴权成功之后。**

```
//AppConfig.resPathForSDK是资源拷贝环节确定的资源路径
mXmagicApi = new XmagicApi(this, AppConfig.resPathForSDK);
```

   参数

参数	含义
Context context	上下文
String resDir	资源文件目录，详见请参见 步骤二
OnXmagicPropertyErrorListener errorListener	可选，回调函数实现类，用于回调SDK初始化和使用过程中的一些错误码，错误码含义请参见 API 文档

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

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

```
        mToast.show(MainActivity.this, tips, duration);
    }

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

4. 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。

```
@Override
public int onCustomProcessTexture(int textureId, int textureWidth, int textureHeight) {
    return mXmagicApi.process(textureId, textureWidth, textureHeight);
}
```

5. 设置美颜或特效。

- 3.5.0版本及以后使用 `setEffect` 方法 方法详细说明见 API 文档。
- 3.3.0版本及之前使用 `updateProperty` 方法详细说明见 API 文档。

```
//3.5.0版本及之后使用此方法
mXmagicApi.setEffect(String effectName, int effectValue, String resourcePath, Map<String, String>
extraInfo)
//例如设置美白,强度50
//mXmagicApi.setEffect(XmagicConstant.EffectName.BEAUTY_WHITEN, 50, null,null);


// 可用的入参属性可以从 XmagicResParser.parseRes() 获得
// 3.3.0版本及之前使用此方法
@Deprecated
mXmagicApi.updateProperty(XmagicProperty<?> p);
```

6. 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。

```
mXmagicApi.onResume();
```

7. 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。

```
mXmagicApi.onPause();
```

8. 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() {
    mXmagicApi.onDestroy();
}
```

步骤四：混淆配置

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

```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
```

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

附件（SDK文件结构）：

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

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

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

Windows

最近更新时间：2024-03-20 17:48:51

集成准备

开发者环境要求

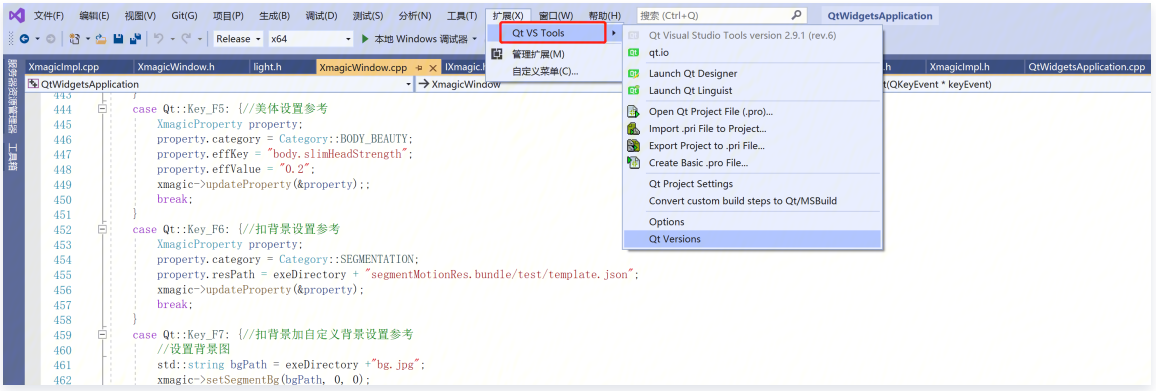
- 开发工具：VS2019（建议）
- 运行环境依赖：QT5.15.2 + QT VS 插件

硬件环境要求

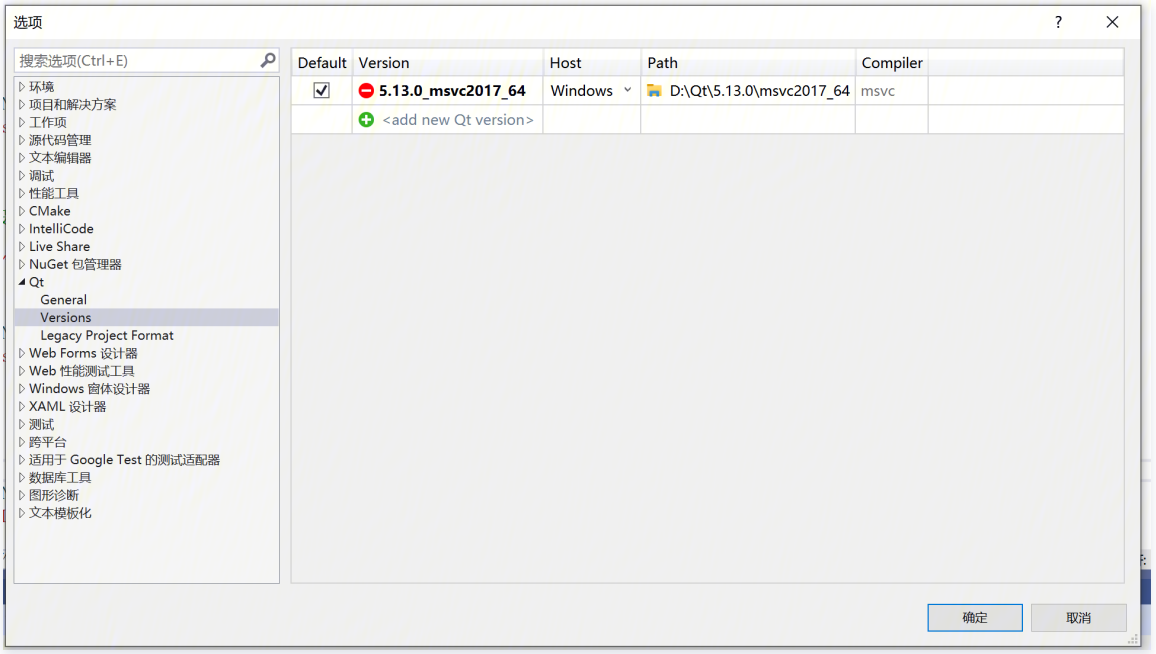
- 显卡支持 OpenGL 4.1 或者以上
- 系统要求 windows8 或者以上
- CPU 支持 AVX 和 AVX2 指令

配置开发者环境

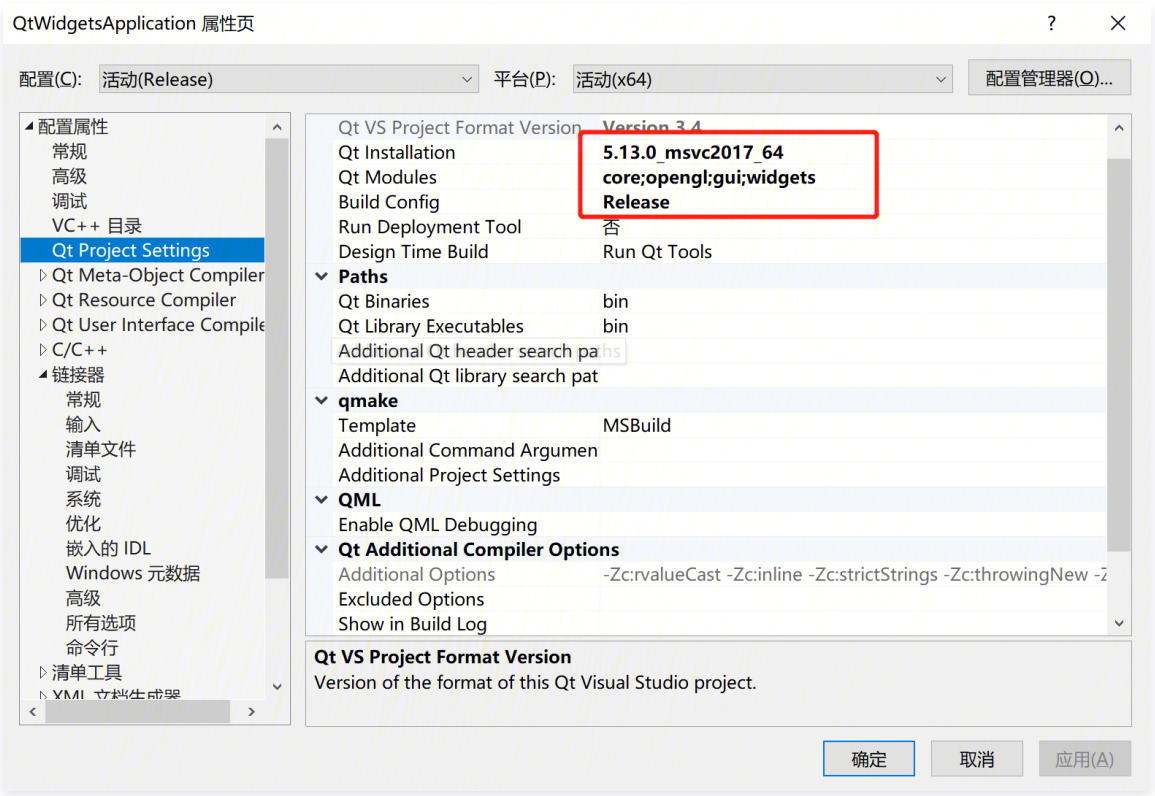
1. 安装 QT 插件。QT 插件安装成功后界面如下：



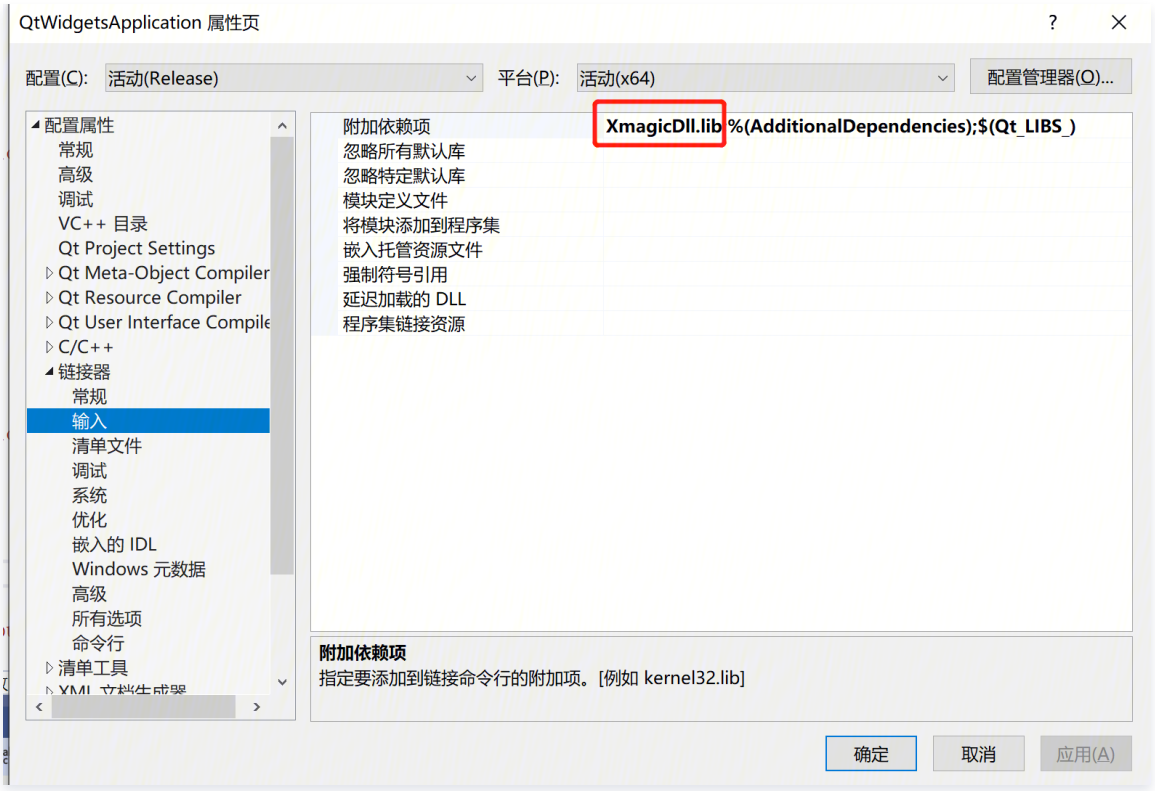
2. 配置 QT 插件版本，选择对应的 QT 版本。



3. 配置工程依赖的 QT 组件。



4. 添加头文件和 Xmagic 的 dll 库。



接口概览

Windows 端 SDK 的集成涉及到的接口如下：

接口名	描述功能	代码示例

setTELicense	鉴权接口	示例
createXmagic	Xmagic 创建接口	示例
destroyXmagic	Xmagic 销毁接口	示例
updateProperty	设置美颜参数（如美颜、动效、化妆等）	示例
process	处理美颜接口，返回对应的美颜处理数据（如像素、纹理）	
setSegmentBg	设置自定义人像分割	
setRenderSize	重置输入数据大小	-
onPasue	暂停美颜	-
onResume	开始美颜	-

接口使用说明

鉴权接口

```
XMAGIC_API void setTELicense(const char* url, const char* key, int (*TELicenseCallback)(int, const char*));
```

参数含义：

参数	含义
url	对应的鉴权证书链接（在后台获取）
key	对应的鉴权证书 key（在后台获取）
TELicenseCallback	鉴权成功或者失败的回调

鉴权回调 code 的含义：

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 页失败
-13	鉴权失败

其他	请联系腾讯云团队处理

创建 Xmagic 接口

```
XMAGIC_API IXmagic* createXmagic(std::string& resDir, int width, int height);
```

参数含义：

参数	含义
resDir	对应的资源文件（SDK 依赖的资源路径）
width	处理数据宽
height	处理数据高

销毁 Xmagic 接口

```
XMAGIC_API void destroyXmagic(IXmagic** xmagic);
```

设置美颜参数

```
virtual void updateProperty(XmagicProperty* property) = 0;
```

参数含义：

参数	说明
category	美颜类型，包括以下几种： • 美颜：BEAUTY = 0 • 美体：BODY_BEAUTY • 滤镜：LUT • 动效：MOTION • 人像分割：SEGMENTATION • 化妆：MAKEUP
resPath	对应设置资源的地址（如动效、化妆、人像分割等）
effKey	美颜的 key
effValue	美颜的值
isAuth	默认 true

处理美颜（像素）

```
virtual void process(YTImagePixelData* srcImage, YTImagePixelData* dstImage) = 0;
```

参数说明：

参数	说明
srcImage	输入的美颜数据
dstImage	输出的美颜数据

YTImagePixelData 结构说明：

```
struct YTImagePixelData {
        /// 字段含义：像素格式
        PixelFormat pixelFormat;

        /// 字段含义
        uint8_t* data;

        /// 字段含义：数据的长度，单位是字节
        int32_t length;

        /// 字段含义：宽度
        int32_t width;

        /// 字段含义：高度
        int32_t height;

        YTImagePixelData() : pixelFormat(PixelFormat::PixelFormatRGBA32), data(nullptr), length(0),
width(0), height(0) {}
    };
```

处理美颜纹理，返回对应的处理纹理

```
virtual int process(int textureId, int width, int height) = 0;
```

参数说明：

参数	说明
textureId	输入的纹理 ID
width	纹理宽度
height	纹理高度

设置自定义人像分割

```
virtual void setSegmentBg(std::string &segmentBgName ,int segmentBgType ,int timeOffset) = 0;
```

参数说明

参数	说明
segmentBgName	自定义人像分割路径
segmentBgType	设置背影类型（0为图片，1为视频）
timeOffset	如果为视频，设置视频播放时长

重置输入大小

```
virtual void setRenderSize(int width, int height) = 0;
```

参数说明：

参数	说明
width	重置后的宽度
height	重置后的高度

暂停美颜

```
virtual void onPasue() = 0;
```

开始美颜

```
virtual void onResume() = 0;
```

代码示例

> **说明**
> 以下仅为部分示例代码。

鉴权接口

```
//鉴权接口
    auto respCallback = [](
        int ret, const char* data) -> int {
            int retCode = ret;
            const char* msg = data;
            return 0;
    };

    setTELicense("url", "key", respCallback);
```

创建 Xmaigc

```
//创建Xmaigc
    std::string exeFilePath = "资源位置";
    IXmagic* xmagic = createXmagic(exeDirectory, 720, 1280);
```

销毁 Xmaigc

```
//销毁
if (xmagic) {
    destroyXmagic(&xmagic);
}
```

设置属性和输出

```
//设置属性和输出
    YTImagePixelData src,dst;
    src.width = 720;
    src.height = 1280;
    src.length = 4 * 720 * 1280;
    src.pixelFormat = PixelFormat::PixelFormatRGBA32;
    //uint8_t* rgbaBuffer = (uint8_t*)malloc(src.length);
    //int w = 0, h = 0, comp = 0;
    uint8_t* imageData;
    int w = 0, h = 0, comp = 0;
    stbi_set_flip_vertically_on_load(false);  // SDK要求输入的图是倒置的，否则有问题
    imageData = stbi_load(path.c_str(), &w, &h, &comp, STBI_rgb_alpha);
    src.data = imageData;
```

```
    dst.width = 720;
    dst.height = 1280;
    dst.length = 4 * 720 * 1280;
    dst.pixelFormat = PixelFormat::PixelFormatRGBA32;
    uint8_t* rgbaBuffer = (uint8_t*)malloc(dst.length);//这个要释放
    dst.data = rgbaBuffer;

    std::string bgPath = exeDirectory + "bg.jpg";
    xmagic->setSegmentBg(bgPath, 0, 0);

    XmagicProperty property;
    property.category = Category::SEGMENTATION;
    property.resPath = exeDirectory + "segmentMotionRes.bundle/video_empty_segmentation/template.json";
    xmagic->updateProperty(&property);


    for(int i =0; i < 2; i++){
        xmagic->process(&src, &dst);
    }
```

Mac

最近更新时间：2025-02-28 12:24:13

集成准备

开发者环境要求

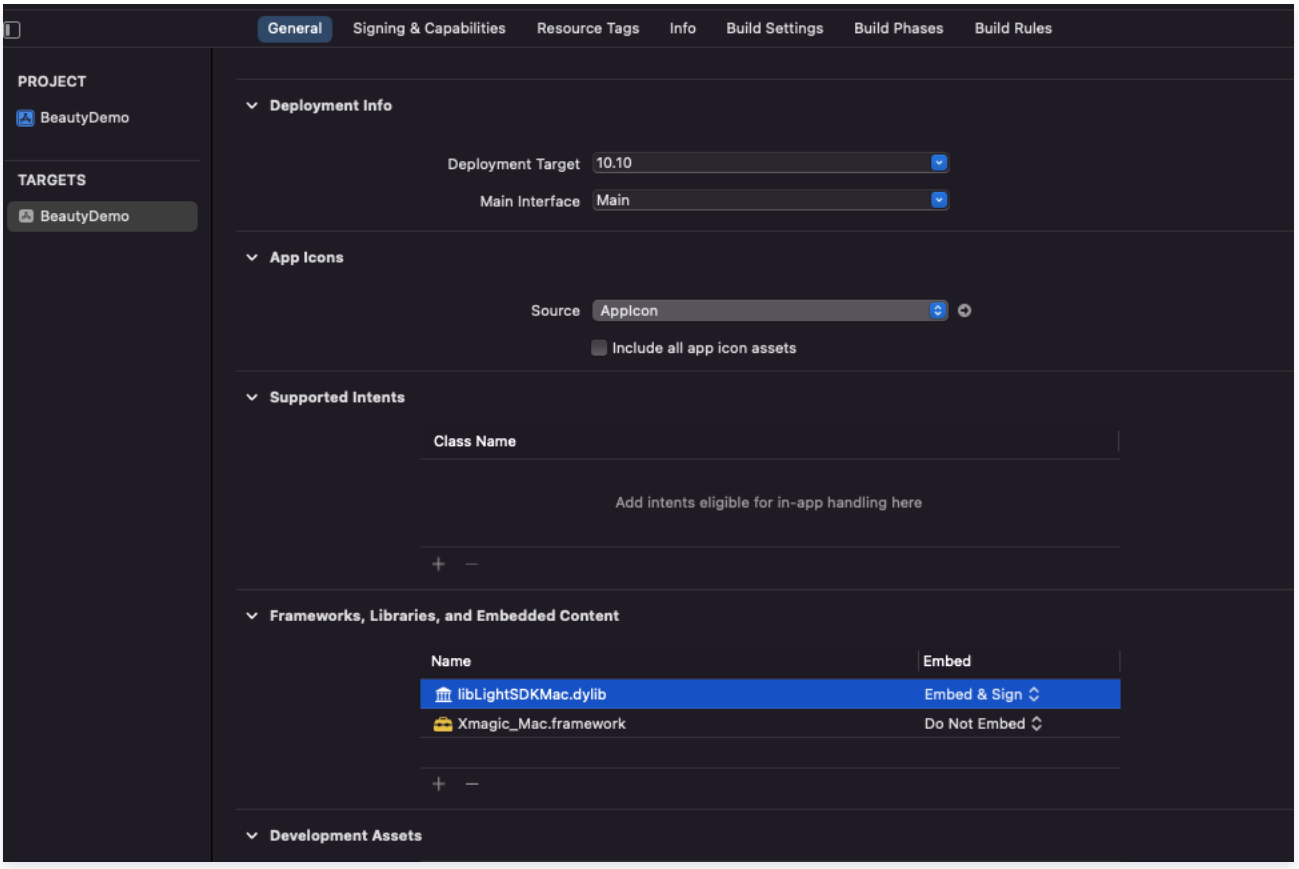
- 开发工具 XCode 11 及以上：App Store 或单击 下载 Demo 。
- 建议运行环境：
  - 设备要求：macOS 系统设备。
  - 系统要求：macOS 10.15 及以上。

导入SDK

您可以先将 SDK 下载到本地，再将其手动导入到您当前的项目中。

下载 SDK 并手动导入

1. 下载并解压 SDK 和美颜资源，Libs 文件夹里面是 `sdk` 、 `resources` 文件夹里面是美颜的 bundle 资源。
2. 打开您的 Xcode 工程项目，把Libs文件夹里面的 Xmagic_Mac.framework、libLightSDKMac、libpag.framework 添加到实际工程中，选择要运行的 target , 选中 **General** 项，单击 **Frameworks,Libraries,and Embedded Content** 项展开，将 libLightSDKMac和libpag.framework 的 Embed 改成 `Embed&Sign` 。



3. 选择对应的 target，选中 Build Settings，找到 Other Linker Flags 添加“-lstdc++”。
4. 在项目里找到 `xxx(项目名).entitlements` 文件，增加如下 key-value：

```
<key>com.apple.security.cs.allow-dyld-environment-variables</key>
 <true/>
```

5. 把 resources 夹里面的美颜资源添加到实际工程中。
6. 将 Bundle ID 修改成与申请的测试授权一致。

动态下载集成

为了减少包大小，您可以将 SDK 所需的模型资源和动效资源 MotionRes（部分基础版 SDK 无动效资源）改为联网下载。在下载成功后，将上述文件的路径设置给 SDK。
动态下载的详细指引，请参见 SDK 包体瘦身（iOS）。

配置权限

在 Info.plist 文件中添加相应权限的说明，否则程序在 iOS 10 系统上会出现崩溃。请在 Privacy - Camera Usage Description 中开启相机权限，允许 App 使用相机。Privacy - Microphone Usage Description 开启麦克风权限，允许使用麦克风。在 target-->signing & Capabilities--> Hardened Runtime 勾选 **Audio Input**、**Camera**。

集成步骤

步骤一：鉴权

1. 申请授权，得到 LicenseURL 和 LicenseKEY，请参见 License 指引。

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

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

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

鉴权 errorCode 说明：

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

-13	鉴权失败
其他	请联系腾讯云团队处理

步骤二：加载 SDK（XMagic.framework）

使用美颜特效 SDK 生命周期大致如下：

1. 加载美颜相关资源。

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

2. 初始化美颜特效 SDK。

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

3. 美颜特效 SDK 处理每帧数据并返回相应处理结果。

```
process: (XMagic)
```

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

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

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

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

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

4. 给 SDK 设置特效。

```
/// 四端统一新接口
/// @brief 配置美颜各种效果（使用方法可参考文档和demo）      Configure various effects of beauty (refer to the
documentation and demo for usage instructions.)
/// @param effectName 效果类型 字符串:  lut, motion.....     property type String:  lut, motion.....
/// @param effectValue 效果数值      property value
/// @param resourcePath 素材路径 resource path
/// @param extraInfo 预留扩展，附加额外配置dict      reserved extension, additional configuration dict
- (void)setEffect:(NSString * _Nullable)effectName
      effectValue:(int)effectValue
     resourcePath:(NSString * _Nullable)resourcePath
        extraInfo:(NSDictionary * _Nullable)extraInfo;
```

```
/// @brief 配置美颜各种效果，旧接口，建议使用上方新接口
/// @param propertyType 效果类型 字符串：beauty, lut, motion
/// @param propertyName 效果名称
/// @param propertyValue 效果数值
/// @param extraInfo 预留扩展，附加额外配置dict
/// @return 成功返回0，失败返回其他
/// @note 具体说明
/**
| 效果类型 | 效果名称 | 效果值 | 说明 | 备注 |
| :---- | :---- |:---- | :---- | :---- |
| beauty | 美颜id名称 | 美颜效果强度数值 |美颜类型配置接口 | 无 |
| lut | 滤镜路径+滤镜名称 | 滤镜强度数值 | 滤镜类型配置接口 | 无 |
| motion | 动效路径名称 | 动效路径 | 动效类型配置接口| **注意**：如果资源中有zip，请确保传入动效路径为可写路径，否则跟app包走需要手动unzip才可以使用 |
**/
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString *_Nonnull)propertyName withData:(NSString*_Nonnull)propertyValue withExtraInfo:(id _Nullable)extraInfo;
```

参数的设置和说明，请参见 参数设置说明 。

5. 释放美颜特效 SDK。

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

iOS

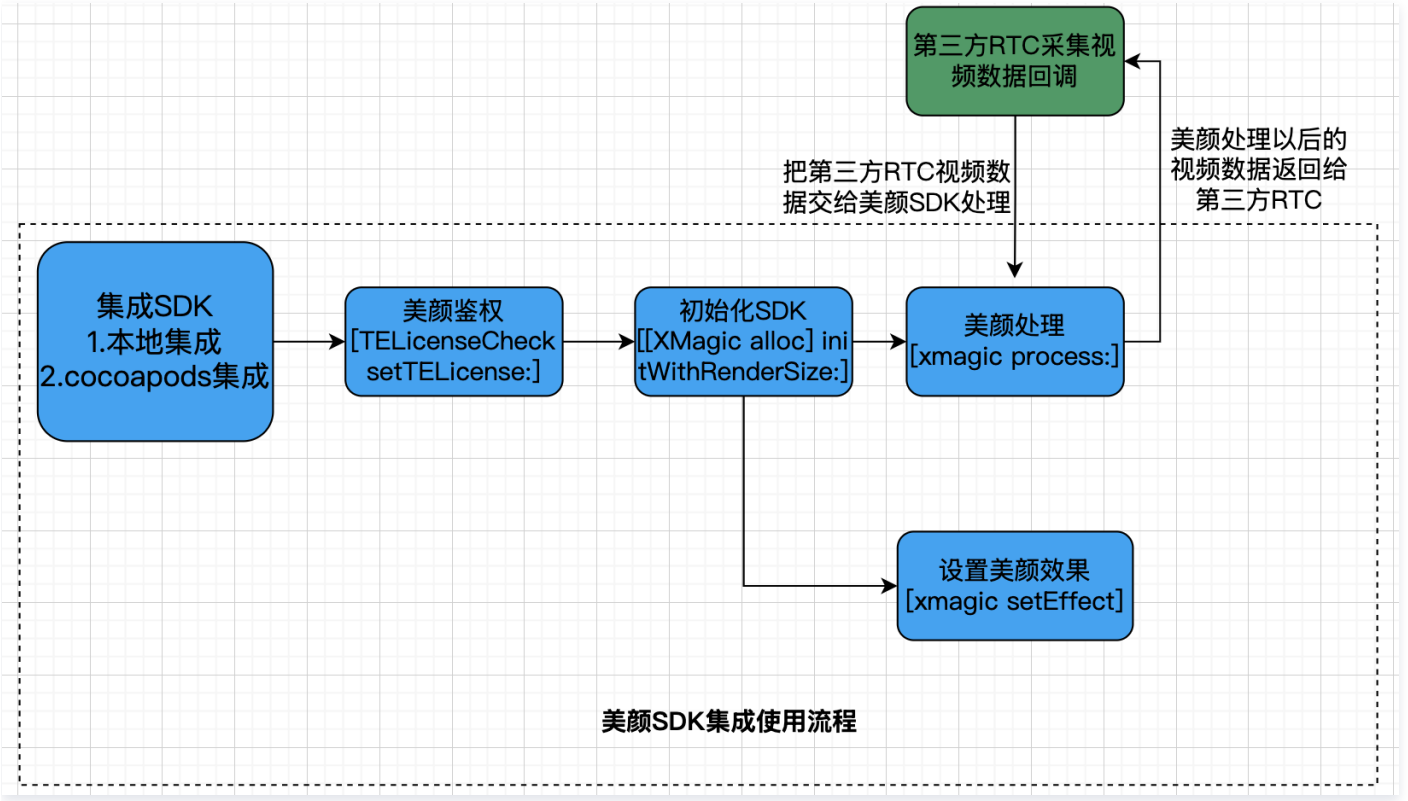
最近更新时间：2025-02-28 12:24:13

集成准备

开发者环境要求

- 开发工具 XCode 11 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 12.2 及以上。

美颜特效 SDK 集成使用流程



导入 SDK

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

使用 CocoaPods

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

```
sudo gem install cocoapods
```

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

```
pod init
```

3. **编辑 Podfile 文件**

- XMagic 版本在3.0.1以前：
  根据您的项目需要选择合适的版本，并编辑 Podfile 文件：
  - **XMagic 普通版**
    请按如下方式编辑 Podfile文件：

```
platform :ios, '8.0'

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

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

```
platform :ios, '8.0'

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

- XMagic 版本在3.0.1及以后：
  根据您的项目套餐选择合适的版本，并编辑 Podfile 文件：

```
#请根据您的套餐pod install对应的库
#例如：如果您的套餐是all类型，那么只需要pod 'TencentEffect_All'
#例如：如果您的套餐是S1-04类型，那么只需要pod 'TencentEffect_S1-04'
pod 'TencentEffect_All'
#pod 'TencentEffect_A1-00'
#pod 'TencentEffect_A1-01'
#pod 'TencentEffect_A1-02'
#pod 'TencentEffect_A1-03'
#pod 'TencentEffect_A1-04'
#pod 'TencentEffect_A1-05'
#pod 'TencentEffect_A1-06'
#pod 'TencentEffect_S1-00'
#pod 'TencentEffect_S1-01'
#pod 'TencentEffect_S1-02'
#pod 'TencentEffect_S1-03'
#pod 'TencentEffect_S1-04'
#pod 'TencentEffect_S1-05'
#pod 'TencentEffect_S1-06'
#pod 'TencentEffect_S1-07'
#pod 'TencentEffect_X1-01'
#pod 'TencentEffect_X1-02'
```

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

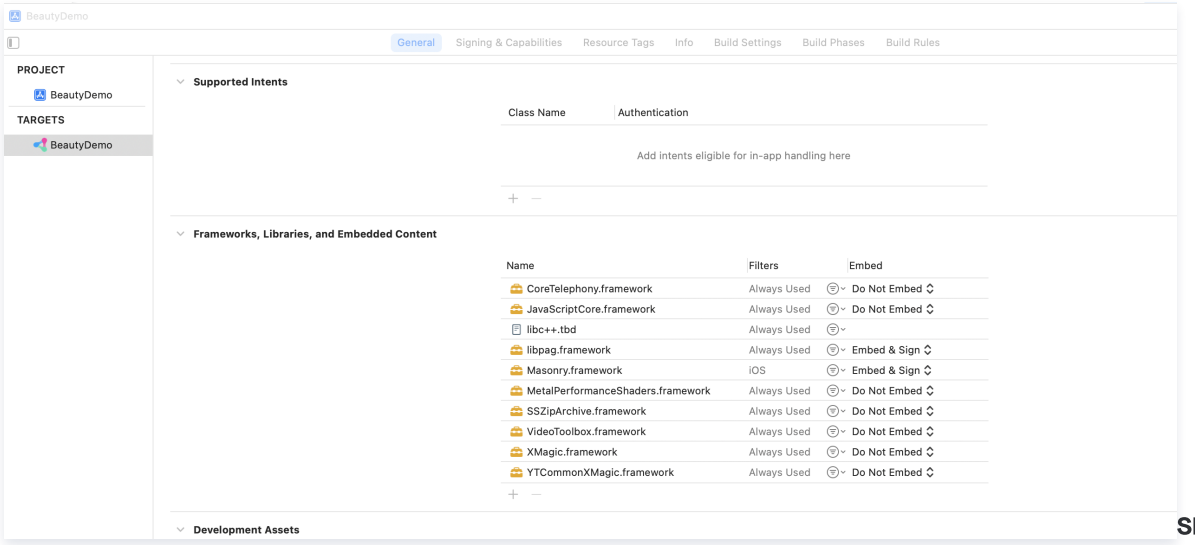
```
pod install
```

   pod 命令执行完后，会生成集成了 SDK 的 `.xcworkspace` 后缀的工程文件，双击打开即可。
5. **添加美颜资源到实际项目工程中**
   下载并解压对应套餐的 SDK 和美颜资源，将 **resources/**motionRes 文件夹下**bundle 资源**添加到实际工程中。
   在 Build Settings 中的Other Linker Flags添加 `-ObjC` 。
6. 将 Bundle Identifier 修改成与申请的测试授权一致。

下载 SDK 并手动导入

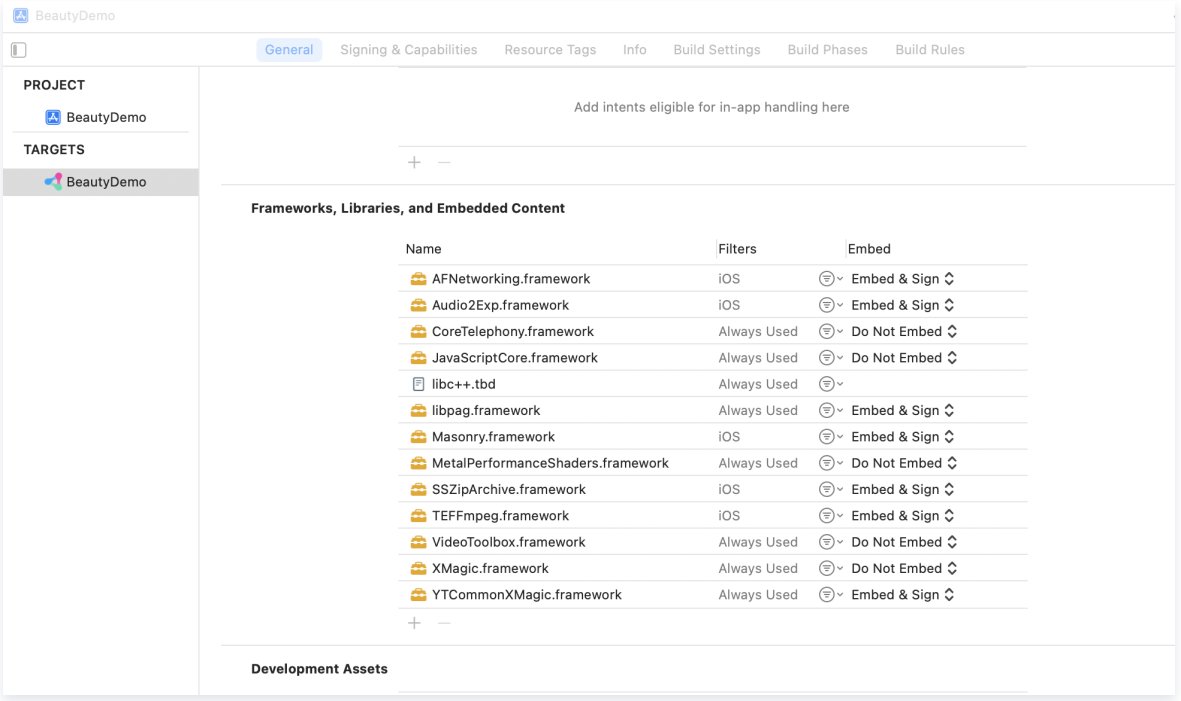
1. 下载并解压 SDK 和美颜资源，frameworks 文件夹里面是sdk、resources 文件夹里面是美颜的bundle资源。
2. **SDK版本在2.5.1以前：**

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



SDK版本在2.5.1及以后：

打开您的 Xcode 工程项目，把 frameworks 文件夹里面的 framework 添加到实际工程中，选择要运行的 target，选中 **General** 项，单击 **Frameworks,Libraries,and Embedded Content** 项展开，单击底下的“+”号图标去添加依赖库。依次添加下载的 `XMagic.framework`、`YTCommonXMagic.framework`、`libpag.framework`、`Audio2Exp.framework`、`TEFFmpeg.framework(version3.0.0以后，改名为：TECodec.framework)` 及其所需依赖库 `MetalPerformanceShaders.framework`、`CoreTelephony.framework`、`JavaScriptCore.framework`、`VideoToolbox.framework`、`libc++.tbd`，根据需要添加其它工具库 `Masonry.framework`（控件布局库）、`SSZipArchive`（文件解压库）。



3. 把 resources 夹里面的美颜资源添加到实际工程中。

4. 在 Build Settings 中的Other Linker Flags添加 `-ObjC` 。
5. 将 Bundle Identifier 修改成与申请的测试授权一致。

动态下载集成

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

配置权限

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

集成步骤

步骤一：鉴权

1. 申请授权，得到 LicenseURL 和 LicenseKEY，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。也可以在 AppDelegate 的 didFinishLaunchingWithOptions 方法里触发下载。其中，LicenseURL 和 LicenseKey 是控制台绑定 License 时生成的授权信息。
   SDK 版本在2.5.1以前，`TELicenseCheck.h` 在 `XMagic.framework` 里面；SDK版本在2.5.1及以后，`TELicenseCheck.h` 在 `YTCommonXMagic.framework` 里面。

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

鉴权 errorCode 说明：

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

-13	鉴权失败
其他	请联系腾讯云团队处理

步骤二：加载 SDK（XMagic.framework）

使用美颜特效 SDK 生命周期大致如下：

1. 加载美颜相关资源。

```
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
 @"root_path":[[NSBundle mainBundle] bundlePath] // LightCore.bundle所在的目录。
};
```

2. 初始化美颜特效 SDK。

```
/**
previewSize:视图的宽高
assetsDict：上一步配置的LightCore.bundle及其路径
*/
self.xMagicApi = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDict];
```

3. 美颜特效 SDK 处理每帧数据并返回相应处理结果。

```
/**
以设备摄像头数据输出为例
*/

//sampleBuffer：设备摄像头输出的数据
-(CMSampleBufferRef)didProcessCPUData:(CMSampleBufferRef)sampleBuffer{
    CVPixelBufferRef pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
    YTProcessInput *input = [[YTProcessInput alloc] init];
    input.pixelData = [[YTImagePixelData alloc] init];
    input.pixelData.data = pixelBuffer;
    input.dataType = kYTImagePixelData;
    YTProcessOutput *output = [self.xMagicKit process:input];
    if (output.pixelData.data != nil) { //output.pixelData.data：美颜SDK处理以后的数据
        CMSampleBufferRef outSampleBuffer = [self sampleBufferFromPixelBuffer:output.pixelData.data];
        return outSampleBuffer;
    }
    return nil;
}

//PixelBuffer转sampleBuffer
- (CMSampleBufferRef)sampleBufferFromPixelBuffer:(CVPixelBufferRef)pixelBuffer
{
    CFRetain(pixelBuffer);
    CMSampleBufferRef outputSampleBuffer = NULL;
    CMSampleTimingInfo timing = {kCMTimeInvalid, kCMTimeInvalid, kCMTimeInvalid};
    CMVideoFormatDescriptionRef videoInfo = NULL;
    OSStatus result = CMVideoFormatDescriptionCreateForImageBuffer(NULL, pixelBuffer, &videoInfo);
    result = CMSampleBufferCreateForImageBuffer(kCFAllocatorDefault, pixelBuffer, true, NULL, NULL,
videoInfo, &timing, &outputSampleBuffer);
    CFArrayRef attachments = CMSampleBufferGetSampleAttachmentsArray(outputSampleBuffer, YES);
    CFMutableDictionaryRef dict = (CFMutableDictionaryRef)CFArrayGetValueAtIndex(attachments, 0);
    CFDictionarySetValue(dict, kCMSampleAttachmentKey_DisplayImmediately, kCFBooleanTrue);
    CFRelease(videoInfo);
    CFRelease(pixelBuffer);
    return outputSampleBuffer;
```

```
}
```

美颜 process 方法及更多 API 详情见 API 文档。

4. 释放美颜特效 SDK。

```
// 在需要释放SDK资源的地方调用
[self.xMagicApi deinit]
```

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

常见问题

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

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

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

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

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

问题3：滤镜设置没反应?

检查下设置的值是否正确，范围为 0 – 100，可能值太小了效果不明显。

问题4：iOS Demo 编译，生成 dSYM 时报错?

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

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

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

  1.1 打开 `demo/copy_framework.sh`

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

  ```
  which cmake
  ```

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

原子能力集成指引
手势识别

最近更新时间：2025-02-28 12:24:13

功能说明

输入相机的 openGL 纹理，实时输出手势检测数据。您可以利用这些数据做进一步的开发。

Android 接口说明

Android 集成指引

Android 集成美颜特效 SDK，具体请参见 独立集成美颜特效。

Android 接口调用

1. 打开手势检测功能开关（XmagicApi.java）。

```
public void setFeatureEnableDisable(String featureName, boolean enable);
```

   featureName 填 `XmagicConstant.FeatureName.HAND_DETECT`，enable 填 `true`。
2. 设置数据回调（XmagicApi.java）

```
void setAIDataListener(XmagicApi.OnAIDataListener aiDataListener)

public interface OnAIDataListener {
    void onFaceDataUpdated(List<TEFaceData> faceDataList);
    void onHandDataUpdated(List<TEHandData> handDataList);
    void onBodyDataUpdated(List<TEBodyData> bodyDataList);
    void onAIDataUpdated(String data);
}
```

   `onAIDataUpdated` 返回 JSON 结构的 string 数据。

iOS 接口说明

iOS 集成指引

iOS 集成美颜特效 SDK，具体请参见 独立集成美颜特效。

iOS 接口调用

1. 打开手势检测功能开关（Xmagic.h）。

```
- (void)setFeatureEnableDisable:(NSString *_Nonnull)featureName enable:(BOOL)enable;
```

   featureName 填 `HAND_DETECT`（可在 TEDefine.h 中引入），enable 填 `true`。
2. 设置数据回调（Xmagic.h）

```
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

- (void)onAIEvent:(id)event
{
    NSDictionary *eventDict = (NSDictionary *)event;
    if (eventDict[@"ai_info"] != nil) {
      NSLog(@"ai_info %@",eventDict[@"ai_info"]);
    }
}
```

`eventDict[@"ai_info"]` 即为返回的 JSON 结构的 string 数据。

回调 JSON 数据说明

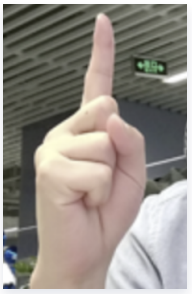
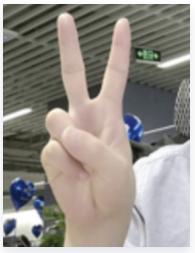
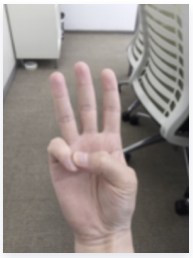
在回调的 JSON 数据中，`"hand_info"` 里是手势相关的数据，格式如下所示。

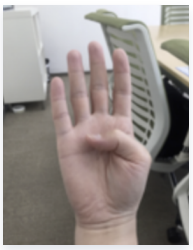
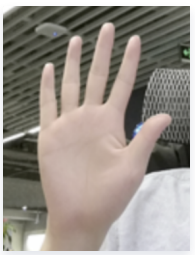
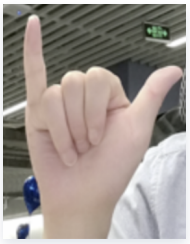
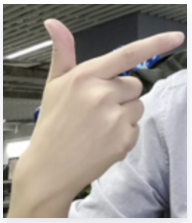
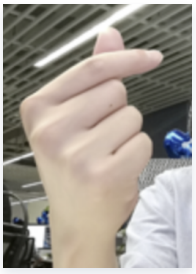
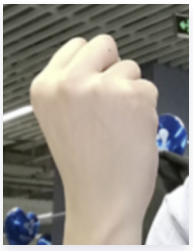
```
"hand_info": {
    "gesture": "PAPER",
    "hand_point_2d": [180.71888732910156, 569.2958984375, ... , 353.8714294433594, 836.246826171875]
}
```

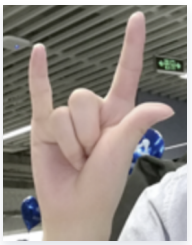
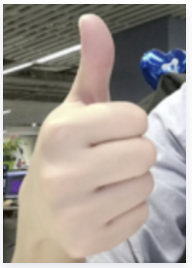
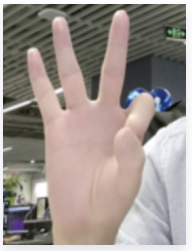
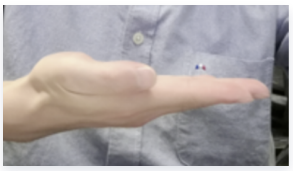
`hand_info` 中各字段说明如下：

字段	含义
gesture	手势类型名称
hand_point_2d	捕捉到手势的数据信息

目前支持以下手势：

序号	手势	类型名称	示例图
1	手势1	ONE	
2	手势2	SCISSOR	
3	手势3	THREE	

4	手势4	FOUR	
5	手势5（open）	PAPER	
6	手势6	SIX	
7	手势8	EIGHT	
8	比心	HEART	
9	拳头	FIST	

10	我爱你	LOVE	
11	点赞	LIKE	
12	OK	OK	
13	摇滚手势	ROCK	
14	托	LIFT	

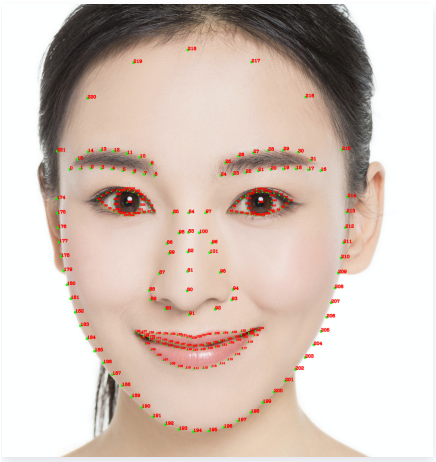
如果为不可识别的手势，则类型名称为 `OTHER` 。

人脸点位

最近更新时间：2025-02-28 12:24:13

人脸检测（识别人脸出框、多人脸、面部遮挡），256个面部关键点位识别与输出。

人脸256点对应索引图



iOS 接口说明

iOS 集成指引

iOS 集成 SDK 指引，请参见 独立集成美颜特效 。

Xmagic 接口回调注册

```
/// @brief SDK事件监听接口
/// @param listener 事件监听器回调，主要分为AI事件，Tips提示事件，Asset事件
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;
```

YTSDKEventListener 回调说明

```
#pragma mark - 事件回调接口
/// @brief SDK内部事件回调接口
@protocol YTSDKEventListener <NSObject>
/// @brief YTDataUpdate事件回调
/// @param event NSString*格式的回调
- (void)onYTDataEvent:(id _Nonnull)event;
/// @brief AI事件回调
/// @param event dict格式的回调
- (void)onAIEvent:(id _Nonnull)event;
/// @brief 提示事件回调
/// @param event dict格式的回调
- (void)onTipsEvent:(id _Nonnull)event;
/// @brief 资源包事件回调
/// @param event string格式的回调
- (void)onAssetEvent:(id _Nonnull)event;
@end
```

2.6.0及之前版本 设置回调成功后，每一帧人脸事件会回调：

```
- (void)onYTDataEvent:(id _Nonnull)event;
```

3.0.0版本 设置回调成功后，每一帧人脸事件会回调：

```
- (void)onAIEvent:(id _Nonnull)event;
   //在onAIEvent方法中可通过下边方法可以获取到数据
   NSDictionary *eventDict = (NSDictionary *)event;
    if (eventDict[@"ai_info"] != nil) {
      NSLog(@"ai_info %@",eventDict[@"ai_info"]);
    }
```

回调 data 是一个 JSON 格式数据，具体含义如下（256点对应上图的位置）：

```
/// @note 字段含义列表
/**
| 字段 | 类型 | 值域 | 说明 |
| :---- | :---- |:---- | :---- |
| trace_id | int | [1,INF) | 人脸id，连续取流过程中，id相同的可以认为是同一张人脸 |
| face_256_point | float | [0,screenWidth或screenHeight] | 共512个数，人脸256个关键点，屏幕左上角为(0,0) |
| face_256_visible | float | [0,1] | 人脸256关键点可见度 |
| out_of_screen | bool | true/false | 人脸是否出框 |
| left_eye_high_vis_ratio | float | [0,1] | 左眼高可见度点位占比 |
| right_eye_high_vis_ratio | float | [0,1] | 右眼高可见度点位占比 |
| left_eyebrow_high_vis_ratio | float | [0,1] | 左眉高可见度点位占比 |
| right_eyebrow_high_vis_ratio | float | [0,1] | 右眉高可见度点位占比 |
| mouth_high_vis_ratio | float | [0,1] | 嘴高可见度点位占比 |
**/
- (void)onYTDataEvent:(id _Nonnull)event;
```

Android 接口说明

Android 集成指引

Android 集成 SDK 指引，具体请参见 独立集成美颜特效 。

Xmagic 接口回调注册

设置人脸点位信息等数据回调。

2.6.0及之前版本使用如下方法

```
void setYTDataListener(XmagicApi.XmagicYTDataListener ytDataListener)
设置人脸信息等数据回调

public interface XmagicYTDataListener {
    void onYTDataUpdate(String data)
}
```

3.0.0版本使用如下方法

```
void setAIDataListener(XmagicApi.OnAIDataListener aiDataListener)

public interface OnAIDataListener {
    void onFaceDataUpdated(List<TEFaceData> faceDataList);
    void onHandDataUpdated(List<TEHandData> handDataList);
    void onBodyDataUpdated(List<TEBodyData> bodyDataList);
    void onAIDataUpdated(String data); //此方法是3.0.0版本新增方法，数据结构和之前XmagicYTDataListener接口的数据保
持一致
}
```

onYTDataUpdate 和 onAIDataUpdated 返回 JSON string 结构，最多返回5个人脸信息：

```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
    180.0,
    112.2,
    ...
  ],
  "face_256_visible":[
    0.85,
    ...
  ],
  "out_of_screen":true,
  "left_eye_high_vis_ratio":1.0,
  "right_eye_high_vis_ratio":1.0,
  "left_eyebrow_high_vis_ratio":1.0,
  "right_eyebrow_high_vis_ratio":1.0,
  "mouth_high_vis_ratio":1.0
 },
 ...
 ]
}
```

字段含义

字段	类型	值域	说明
trace_id	int	[1,INF)	人脸 ID，连续取流过程中，ID 相同的可以认为是同一张人脸。
face_256_point	float	[0,screenWidth] 或 [0,screenHeight]	共512个数，人脸256个关键点，屏幕左上角为(0,0)。
face_256_visible	float	[0,1]	人脸256关键点可见度。
out_of_screen	bool	true/false	人脸是否出框。
left_eye_high_vis_ratio	float	[0,1]	左眼高可见度点位占比。
right_eye_high_vis_ratio	float	[0,1]	右眼高可见度点位占比。
left_eyebrow_high_vis_ratio	float	[0,1]	左眉高可见度点位占比。
right_eyebrow_high_vis_ratio	float	[0,1]	右眉高可见度点位占比。
mouth_high_vis_ratio	float	[0,1]	嘴高可见度点位占比。

参数

参数	含义
XmagicApi.XmagicYTDataListener ytDataListener	回调函数实现类。

人像分割

最近更新时间：2025-02-28 12:24:13

在直播、会议等场景实现虚拟背景，实时精准分割，支持自定义背景：

iOS 接口使用

iOS 集成指引

iOS 集成 SDK 指引，请参见 独立集成美颜特效 。

虚拟背景设置

```
NSString *motionSegResPath = [[NSBundle mainBundle] pathForResource:@"segmentMotionRes"
ofType:@"bundle"];//这里是segmentMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以分割为例
NSString *propertyName = @"video_segmentation_blur_75"; //配置美颜的名称，这里以背景模糊-强为例
NSString *propertyValue = motionSegResPath;  //配置动效的路径
NSDictionary *dic = @{@"bgName":@"BgSegmentation.bg.png", @"bgType":@0, @"timeOffset":
@0},@"icon":@"segmentation.linjian.png"};//配置预留字段
[self.beautyKit configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:dic];
```

自定义背景设置

```
NSString *motionSegResPath = [[NSBundle mainBundle] pathForResource:@"segmentMotionRes"
ofType:@"bundle"];//这里是segmentMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以分割为例
NSString *propertyName = @"video_empty_segmentation"; //配置美颜的名称，这里以自定义背景为例
NSString *propertyValue = motionSegResPath;  //配置动效的路径
NSString *imagePath = @"/var/mobile/Containers/Data/Application/06B00BBC-9060-450F-8D3A-
F6028D185682/Documents/MediaFile/image.png"; //自定义背景图片的绝对路径。如果自定义背景选择的是视频，需要对视频进行压
缩转码处理，使用压缩转码处理后的绝对路径
int bgType = 0;//自定义背景的类型。 0表示图片，1表示视频
int timeOffset = 0; //时长。图片背景时，为0；视频背景时为视频的时长
```

```
NSDictionary *dic = @{@"bgName":imagePath, @"bgType":@(bgType), @"timeOffset":
@(timeOffset)},@"icon":@"segmentation.linjian.png"};//配置预留字段
[self.beautyKit configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:dic];
```

Android 接口使用

Android 集成指引

Android 集成 SDK 指引，请参见 独立集成美颜特效 。

设置属性接口

```
void updateProperty(XmagicProperty<?> p)
```

分割参数：

属性字段	说明
category	Category.SEGMENTATION
ID	资源文件夹名称，必填示例：`video_segmentation_blur_45` • “无” ID 为 `XmagicProperty.ID_NONE` • 自定义分割 ID 值必须使用：`XmagicConstant.SegmentationId.CUSTOM_SEG_ID`
resPath	必填，自定义分割素材的全路径
effkey	null（自定义背景除外），自定义背景的值为选择的资源路径
effValue	null

虚拟背景设置

```
//初始化XmagicProperty
XmagicProperty xmagicProperty = new
XmagicProperty(Category.SEGMENTATION,"video_segmentation_blur_45",resPath,null,null);
//设置属性
xmagicApi.updateProperty(xmagicProperty)
```

自定义背景设置

```
XmagicProperty xmagicProperty = new
XmagicProperty(Category.SEGMENTATION,XmagicConstant.SegmentationId.CUSTOM_SEG_ID,resPath,null,null);

xmagicApi.updateProperty(xmagicProperty)
```

人脸属性
iOS

最近更新时间：2025-02-28 12:24:13

功能说明

该接口可通过上传包含人脸的照片获取人物面部特征信息，包括眼睛、眉毛、发型、肤色、性别、年龄等。该接口需要联网环境下才可通过 SDK 把照片上传到 Server 端进行解析。

集成指引

首先需要集成美颜特效 SDK，具体请参见 独立集成美颜特效 。

接口说明

接口所在类：XMagic.h

```
+ (void)getFaceFeatureFromPhoto:(UIImage * _Nullable)image success:(nullable void (^)
(FaceDetailAttributesinfo *_Nullable faceInfo))success failure:(nullable void (^)(NSInteger errorCode,
NSString *_Nullable msg))failure;
```

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

```
// 成功回调，返回识别结果
(nullable void (^)(FaceDetailAttributesinfo *_Nullable faceInfo))success;
// 失败回调
(nullable void (^)(NSInteger errorCode, NSString *_Nullable msg))failure;
```

- failure 回调：解析失败时会回调此接口，错误码如下。

```
typedef NS_ENUM(NSInteger, TEAvatarErrorCode) {
    TEAvatarErrorCodeNoAuth = 1,                // 没有权限
    TEAvatarErrorCodeResInValid = 5,            // 传入的Avatar素材路径无效
    TEAvatarErrorCodePhotoInValid = 10,         // 读取照片失败
    TEAvatarErrorCodeNetReqFailed = 20,         // 网络请求失败
    TEAvatarErrorCodeNetResParseFailed = 30,    // 网络返回数据解析失败
    TEAvatarErrorCodeAnalyzeFailed = 40,        // 人脸分析失败
    TEAvatarErrorCodeAvatarSourceEmpty = 50     // 加载Avatar源数据失败
};
```

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

```
/// @brief 拍照捏脸配置数据      Snapping face configuration data

// 眼睛     eye
@interface Eye : NSObject
/**
 识别是否双眼皮。-1:没识别，0:无，1:有
 Identify whether double eyelid. -1: not recognized, 0: no, 1: yes
```

```
 */
@property (nonatomic, assign) NSInteger eyelidType;

/**
 眼睛大小。-1:没识别，0:小眼睛，1:普通眼睛，2:大眼睛
 eye size. -1: No recognition, 0: Small eyes, 1: Normal eyes, 2: Big eyes
 */
@property (nonatomic, assign) NSInteger eyeSize;

/**
 识别是否佩戴眼镜。-1:没识别，0:无眼镜，1:普通眼镜，2:墨镜
 Identify whether glasses are worn. -1: No recognition, 0: No glasses, 1: Ordinary glasses, 2:
Sunglasses
 */
@property (nonatomic, assign) NSInteger glass;

/**
 识别眼睛是否睁开。-1:没识别，0:睁开，1:闭眼
 Identify if eyes are open. -1: No recognition, 0: Open, 1: Closed eyes
 */
@property (nonatomic, assign) NSInteger eyeOpen;
@end


// 眉毛     Eyebrow
@interface Eyebrow : NSObject
/**
 眉毛长短。0:短眉毛，1:长眉毛
 The length of the eyebrows. 0: short eyebrows, 1: long eyebrows
 */
@property (nonatomic, assign) NSInteger eyebrowLength;

/**
 眉毛浓密。0:淡眉，1:浓眉
 thick eyebrows. 0: light eyebrow, 1: thick eyebrow
 */
@property (nonatomic, assign) NSInteger eyebrowDensith;

/**
 眉毛弯曲。0:不弯，1:弯眉
 The eyebrows are curved. 0: not curved, 1: curved eyebrow
 */
@property (nonatomic, assign) NSInteger eyebrowCurve;
@end


// 头发     Hair
@interface Hair : NSObject
/**
 头发长度信息。0:光头，1:短发，2:中发，3:长发，4:绑发
 Hair length information. 0: bald, 1: short hair, 2: medium hair, 3: long hair, 4: tied hair
 */
@property (nonatomic, assign) NSInteger length;

/**
 刘海信息。0:无刘海，1:有刘海
 Bangs information. 0: no bangs, 1: with bangs
 */
@property (nonatomic, assign) NSInteger bang;

/**
```

```
 头发颜色信息。0:黑色，1:金色，2:棕色，3:灰白色
 Hair color information. 0: black, 1: gold, 2: brown, 3: off-white
 */
@property (nonatomic, assign) NSInteger color;
@end


// 帽子      Hat
@interface Hat : NSObject
/**
 帽子佩戴状态。0:不戴帽子，1:普通帽子，2:头盔，3:保安帽子
 The state of wearing the hat. 0: no hat, 1: ordinary hat, 2: helmet, 3: security hat
 */
@property (nonatomic, assign) NSInteger style;

/**
 帽子颜色。0:不戴帽子，1:红色系，2:黄色系，3:蓝色系，4:黑色系，5:灰白色系，6:混色系
 hat color. 0: no hat, 1: red, 2: yellow, 3: blue, 4: black, 5: off-white, 6: mixed color
 */
@property (nonatomic, assign) NSInteger color;
@end


@interface FaceDetailAttributesinfo : NSObject

@property (nonatomic, assign) NSInteger age; // [0,100]

/**
 0:自然，1:高兴，2:惊讶，3:生气，4:悲伤，5:厌恶，6:害怕
 0: Natural, 1: Happy, 2: Surprised, 3: Angry, 4: Sad, 5: Disgusted, 6: Scared
 */
@property (nonatomic, assign) NSInteger emotion;

/**
 性别。-1：没识别，0:男性，1:女性
 gender. -1: not identified, 0: male, 1: female
 */
@property (nonatomic, assign) NSInteger gender;

/**
 眼睛信息
 eye information
 */
@property (nonatomic, strong) Eye * _Nonnull eye;

/**
 眉毛信息
 eyebrow information
 */
@property (nonatomic, strong) Eyebrow * _Nonnull eyebrow;

/**
 头发信息
 hair information
 */
@property (nonatomic, strong) Hair * _Nonnull hair;

/**
 帽子信息
 hat information
 */
```

```
@property (nonatomic, strong) Hat * _Nonnull hat;

/**
 是否有口罩，-1:没识别，0:无，1:有
 Whether there is a mask, -1: no recognition, 0: no, 1: yes
 */
@property (nonatomic, assign) NSInteger mask;

/**
 胡子信息。-1:没识别，0:无胡子，1:有胡子
 Beard information. -1: no recognition, 0: no beard, 1: beard
 */
@property (nonatomic, assign) NSInteger moustache;

/**
 鼻子信息。-1:没识别，0:朝天鼻，1:鹰钩鼻，2:普通，3:圆鼻头
 nose information. -1: No recognition, 0: Upturned nose, 1: Hooked nose, 2: Normal, 3: Round nose
 */
@property (nonatomic, assign) NSInteger nose;

/**
 脸型信息。-1:没识别，0:方脸，1:三角脸，2:鹅蛋脸，3:心形脸，4:圆脸
 face information. -1: No recognition, 0: Square face, 1: Triangular face, 2: Oval face, 3: Heart-
shaped face, 4: Round face
 */
@property (nonatomic, assign) NSInteger shape;

/**
 肤色信息。-1:没识别，0:黄色皮肤，1:棕色皮肤，2:黑色皮肤，3:白色皮肤
 Skin color information. -1: not identified, 0: yellow skin, 1: brown skin, 2: black skin, 3: white
skin
 */
@property (nonatomic, assign) NSInteger skin;

/**
 微笑程度。[0,100]
 Smile level [0,100]
 */
@property (nonatomic, assign) NSInteger smile;
@end
```

Android

最近更新时间：2025-02-28 12:24:13

功能说明

该接口可通过上传包含人脸的照片获取人物面部特征信息，包括眼睛、眉毛、发型、肤色、性别、年龄等。该接口需要联网环境下才可通过 SDK 把照片上传到 Server 端进行解析。

集成指引

首先需要集成美颜特效 SDK，具体请参见 独立集成美颜特效 。

接口说明

接口所在类：XMagicApi.java

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

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

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

- onError 回调：解析失败时会回调此接口，错误码如下。

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

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

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

    public static class Eye {
        public int eyelidType;   // 识别是否双眼皮。-1:没识别，0：无，1：有。
        public int eyeSize;      // 眼睛大小。-1:没识别，0：小眼睛，1：普通眼睛，2：大眼睛。
```

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

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

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

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

人脸表情
iOS

最近更新时间：2025-02-28 12:24:13

功能说明

输入相机的 openGL 纹理，实时输出人脸52表情 BlendShape 数据，遵循苹果 ARKit 规范，详情请参见 ARFaceAnchor 。您可以利用这些表情数据做一进步的开发，例如传到 Unity 中驱动您的模型。

集成指引

iOS 集成 SDK 指引，具体请参见 独立集成美颜特效 。

接口调用

1. 打开功能开关：

```
[self.beautyKit setFeatureEnableDisable:ANIMOJI_52_EXPRESSION enable:YES];
```

2. 设置人脸点位信息数据回调。

2.6.0及之前版本使用如下方法

```
//XMagic.h
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

@implementation listener
- (void)onYTDataEvent:(id)event
{
    NSLog(@"YTData %@", event);
}
```

3.0.0版本使用如下方法

```
//XMagic.h
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

- (void)onAIEvent:(id)event
{
    NSDictionary *eventDict = (NSDictionary *)event;
    if (eventDict[@"ai_info"] != nil) {
      NSLog(@"ai_info %@",eventDict[@"ai_info"]);
    }
}
```

onYTDataUpdate 返回 JSON string 结构，最多返回5个人脸信息：
onAIEvent 中通过 `eventDict[@"ai_info"]` 获取到的 JSON string 结构数据，最多返回5个人脸信息：

```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
 180.0,
 112.2,
 ...
  ],
  "face_256_visible":[
 0.85,
```

```
  ...
  ],
  "out_of_screen":true,
  "left_eye_high_vis_ratio":1.0,
  "right_eye_high_vis_ratio":1.0,
  "left_eyebrow_high_vis_ratio":1.0,
  "right_eyebrow_high_vis_ratio":1.0,
  "mouth_high_vis_ratio":1.0,
  "expression_weights":[
     0.12,
     -0.32
     ...
 ]
 },
 ...
 ]
}
```

字段含义

- **trace_id**：人脸 ID，连续取流过程中，ID 相同的可以认为是同一张人脸。
- **expression_weights**：实时表情 blendshape 数据，数组长度为52，每个数值取值范围为 0到1.0。{
  "eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft","eyeSquintLeft","eyeWideLeft","eyeBlinkRight","eyeLookDownRight","eyeLookInRight","eyeLookOutRight","eyeLookUpRight","eyeSquintRight","eyeWideRight","jawForward","jawLeft","jawRight","jawOpen","mouthClose","mouthFunnel","mouthPucker","mouthRight","mouthLeft","mouthSmileLeft","mouthSmileRight","mouthFrownRight","mouthFrownLeft","mouthDimpleLeft","mouthDimpleRight","mouthStretchLeft","mouthStretchRight","mouthRollLower","mouthRollUpper","mouthShrugLower","mouthShrugUpper","mouthPressLeft","mouthPressRight","mouthLowerDownLeft","mouthLowerDownRight","mouthUpperUpLeft","mouthUpperUpRight","browDownLeft","browDownRight","browInnerUp","browOuterUpLeft","browOuterUpRight","cheekPuff","cheekSquintLeft","cheekSquintRight","noseSneerLeft","noseSneerRight","tongueOut"}
- 其他字段是 人脸信息，只有当您购买了相关 License 才有那些字段。如果您只想获取表情数据，请忽略那些字段。

Android

最近更新时间：2025-02-28 12:24:13

功能说明

输入相机的 openGL 纹理，实时输出人脸52表情 BlendShape 数据，遵循苹果 ARKit 规范，详情请参见 ARFaceAnchor 。您可以利用这些表情数据做一进步的开发，例如传到 Unity 中驱动您的模型。

集成指引

Android 集成 SDK 指引，具体请参见 独立集成美颜特效 。

接口调用

1. 打开功能开关：

```
//XmagicApi.java
//featureName = XmagicConstant.FeatureName.ANIMOJI_52_EXPRESSION
public void setFeatureEnableDisable(String featureName, boolean enable);
```

2. 设置人脸点位信息数据回调。

2.6.0及之前版本使用如下方法

```
//XmagicApi.java
void setYTDataListener(XmagicApi.XmagicYTDataListener ytDataListener)

public interface XmagicYTDataListener {
 void onYTDataUpdate(String data)
}
```

3.0.0版本使用如下方法

```
void setAIDataListener(XmagicApi.OnAIDataListener aiDataListener)

public interface OnAIDataListener {
    void onFaceDataUpdated(List<TEFaceData> faceDataList);
    void onHandDataUpdated(List<TEHandData> handDataList);
    void onBodyDataUpdated(List<TEBodyData> bodyDataList);
    void onAIDataUpdated(String data); //此方法是3.0.0版本新增方法，数据结构和之前XmagicYTDataListener接口的数
据保持一致
}
```

onYTDataUpdate 和 onAIDataUpdated 返回 JSON string 结构，最多返回5个人脸信息：

```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
 180.0,
 112.2,
 ...
  ],
  "face_256_visible":[
 0.85,
 ...
  ],
  "out_of_screen":true,
```

```
    "left_eye_high_vis_ratio":1.0,
    "right_eye_high_vis_ratio":1.0,
    "left_eyebrow_high_vis_ratio":1.0,
    "right_eyebrow_high_vis_ratio":1.0,
    "mouth_high_vis_ratio":1.0,
    "expression_weights":[
        0.12,
        -0.32
        ...
   ]
   },
   ...
   ]
}
```

字段含义

- **trace_id**：人脸 ID，连续取流过程中，ID 相同的可以认为是同一张人脸。
- **expression_weights**：实时表情blendshape数据，数组长度为52，每个数值取值范围为 0到1.0。{

```
"eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft","eyeSquintLeft","eyeWideLeft","eyeBlinkRight","eyeLookDownRight","eyeLookInRight","eyeLookOutRight","eyeLookUpRight","eyeSquintRight","eyeWideRight","jawForward","jawLeft","jawRight","jawOpen","mouthClose","mouthFunnel","mouthPucker","mouthRight","mouthLeft","mouthSmileLeft","mouthSmileRight","mouthFrownRight","mouthFrownLeft","mouthDimpleLeft","mouthDimpleRight","mouthStretchLeft","mouthStretchRight","mouthRollLower","mouthRollUpper","mouthShrugLower","mouthShrugUpper","mouthPressLeft","mouthPressRight","mouthLowerDownLeft","mouthLowerDownRight","mouthUpperUpLeft","mouthUpperUpRight","browDownLeft","browDownRight","browInnerUp","browOuterUpLeft","browOuterUpRight","cheekPuff","cheekSquintLeft","cheekSquintRight","noseSneerLeft","noseSneerRight","tongueOut"}
```

- 其他字段是 人脸信息，只有当您购买了相关 License 才有那些字段。如果您只想获取表情数据，请忽略那些字段。

身体点位
iOS

最近更新时间：2025-02-28 12:24:13

功能说明

输入相机的 openGL 纹理，实时输出身体3D数据。您可以利用这些3D数据做一进步的开发，例如传到 Unity 中驱动您的模型。

集成指引

首先需要集成美颜特效 SDK，具体请参见 独立集成美颜特效 。

接口调用

1. 打开功能开关（XMagic.h）。

```
- (void)setFeatureEnableDisable:(NSString *_Nonnull)featureName enable:(BOOL)enable;
```

featureName 填 `BODY_3D_POINT`（可以从 `TEDefine.h` 引入）。

2. 设置数据回调（XMagic.h）。

2.6.0及之前版本使用如下方法

```
//XMagic.h
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

@implementation listener
- (void)onYTDataEvent:(id)event
{
    NSLog(@"YTData %@", event);
}
@end
```

onYTDataEvent 返回 JSON 结构的 string 数据，其示例如下：

- "face_info" 里是人脸相关的数据，与身体 3D 数据无关，可以忽略。
- "body_3d_info" 里各字段说明见 身体点位及数据说明 。

3.0.0版本使用如下方法

```
//XMagic.h
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

- (void)onAIEvent:(id)event
{
    NSDictionary *eventDict = (NSDictionary *)event;
    if (eventDict[@"ai_info"] != nil) {
      NSLog(@"ai_info %@",eventDict[@"ai_info"]);
    }
}
```

eventDict[@"ai_info"] 返回 JSON 结构的 string 数据，其示例如下：

- "face_info" 里是人脸相关的数据，与身体 3D 数据无关，可以忽略。
- "body_3d_info" 里各字段说明见 身体点位及数据说明 。

身体点位及点位数据说明

相关说明请参见 身体点位及数据说明 。

Android

最近更新时间：2025-02-28 12:24:13

功能说明

输入相机的 openGL 纹理，实时输出身体3D数据。您可以利用这些3D数据做一进步的开发，例如传到 Unity 中驱动您的模型。

集成指引

首先需要集成美颜特效 SDK，具体请参见 独立集成美颜特效 。

接口调用

1. 打开功能开关（XmagicApi.java）。

```
public void setFeatureEnableDisable(String featureName, boolean enable);
```

featureName 填 `XmagicConstant.FeatureName.BODY_3D_POINT` 。

2. 设置数据回调（XmagicApi.java）。

2.6.0及之前版本使用如下方法

```
void setYTDataListener(XmagicApi.XmagicYTDataListener ytDataListener)

public interface XmagicYTDataListener {
 void onYTDataUpdate(String data)
}
```

onYTDataUpdate 返回 JSON 结构的 string 数据，其示例如下：

- "face_info" 里是人脸相关的数据，与身体 3D 数据无关，可以忽略。
- "body_3d_info" 里各字段说明见下文。

3.0.0版本使用如下方法

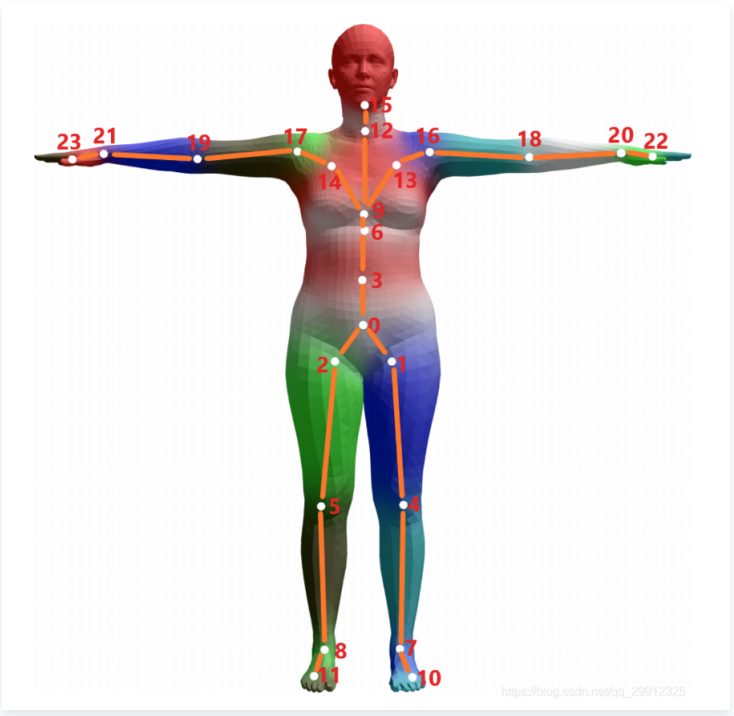
```
void setAIDataListener(XmagicApi.OnAIDataListener aiDataListener)

public interface OnAIDataListener {
    void onFaceDataUpdated(List<TEFaceData> faceDataList);
    void onHandDataUpdated(List<TEHandData> handDataList);
    void onBodyDataUpdated(List<TEBodyData> bodyDataList);
    void onAIDataUpdated(String data); //此方法是3.0.0版本新增方法，数据结构和之前XmagicYTDataListener接口的数
据保持一致
}
```

onAIDataUpdated 返回 JSON 结构的 string 数据，其示例如下：

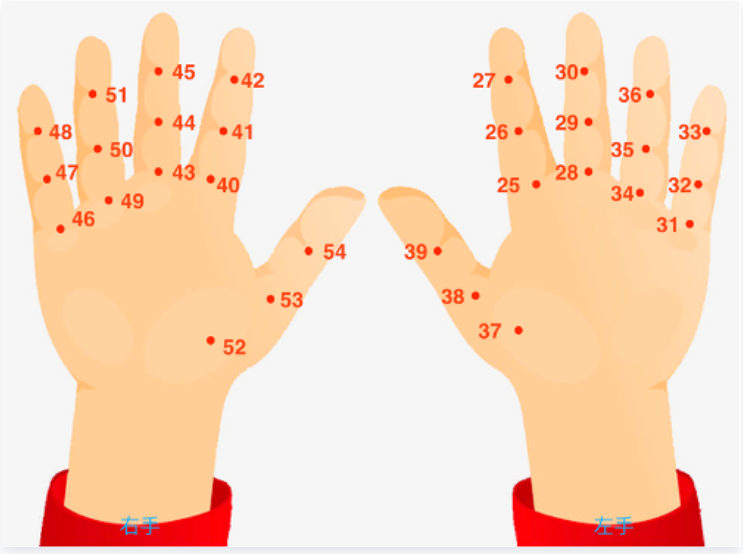
- "face_info" 里是人脸相关的数据，与身体 3D 数据无关，可以忽略。
- "body_3d_info" 里各字段说明见下文。

身体点位及点位数据说明

- 标准 SMPL 点位定义



- 标准 SMPLX 手部骨骼点位定义



SDK 输出的 JSON 数据示例如下：

```
{
    "face_info": [{ ... }],
    "body_3d_info": {
        "imageHeight": 652,
        "imageWidth": 320,
        "items": [{
            "index": 1,
            "pose": [-0.014862474985420704, 0.7201729
            "position_x": [150.62857055664062, 190.15
            "position_y": [605.454833984375, 655.2125
            "position_z": [0, 0, 0, 0, 0, 0, 0, 0, 0,
            "rotation": [{
                "data": [0.9990578889846802, 0.025805
            },
            ......
            {
                "data": [0.9546145796775818, 0.263716
            }]
        }]
    }
}
```

body_3d_info 里各个字段说明如下：

- imageWidth，imageHeight：输入给 SDK 的图像的宽高。
- items：数组，目前只有一个元素。
- index：保留位，目前可以忽略它。
- pose：
  - [0,2]位置，人体位置，以相机为中心，人体根骨骼的3D位置 x、y、z。
  - [3,12]位置，人体形态，10个 float 数，以标准 SMPL 的10套不同mesh为基底，组合得到人体型的估计。
  - [13]位置，Focal_length，固定值为5000。
  - [14,29]位置，OpenGL 投影矩阵，基于 focal_length 得到的在3D空间中渲染物体的投影矩阵。4X4投影矩阵在算法内部计算方式：

```
matrix={
    2 * focal_length / img_wid, 0, 0, 0,
    0, 2 * focal_length / img_hei, 0,0,
    0,0, (zf + zn) / (zn - zf), -1,
    0, 0, (2.0f * zf * zn) / (zn - zf), 0};
}
```

  - [30,33]位置，接地数据，脚是否踩地，左脚跟、左脚尖、右脚跟、右脚尖。
- position_x，position_y，position_z：
  - [0,23]位置，人体2D点位，见上文的图1, 2D点的 position_z 都是0。
  - [24,47]位置，人体3D点位，见上文的图1。
- rotation
  - [0,23]位置，人体骨骼旋转四元数，每个四元数的属性顺序是 w、x、y、z。
  - [25,54]位置，手部骨骼旋转四元数，左手15个，右手15个，每个四元数的属性顺序是 w、x、y、z。

骨骼的不同命名方式及对应关系

序号	Bone Names	Bone Names 2
0	"pelvis",	"Hips"
1	"left_hip",	"LeftUpLeg"
2	"right_hip",	"RightUpLeg"
3	"spine1",	"Spine"
4	"left_knee",	"LeftLeg"

5	"right_knee",	"RightLeg"
6	"spine2",	"Spine1"
7	"left_ankle",	"LeftFoot"
8	"right_ankle",	"RightFoot"
9	"spine3",	"Spine2"
10	"left_foot",	""
11	"right_foot",	""
12	"neck",	"Neck"
13	"left_collar",	"LeftShoulder"
14	"right_collar",	"RightShoulder"
15	"head",	"Head"
16	"left_shoulder",	"LeftArm"
17	"right_shoulder",	"RightArm"
18	"left_elbow",	"LeftForeArm"
19	"right_elbow",	"RightForeArm"
20	"left_wrist",	"LeftHand"
21	"right_wrist",	"RightHand"
22	"left_hand"	""
23	"right_hand"	""
25	"left_index1"	IndexFinger1_L
26	"left_index2"	IndexFinger2_L
27	"left_index3"	IndexFinger3_L
28	"left_middle1"	MiddleFinger1_L
29	"left_middle2"	MiddleFinger2_L
30	"left_middle3"	MiddleFinger3_L
31	"left_pinky1"	PinkyFinger1_L
32	"left_pinky2"	PinkyFinger2_L
33	"left_pinky3"	PinkyFinger3_L
34	"left_ring1"	RingFinger1_L
35	"left_ring2"	RingFinger2_L
36	"left_ring3"	RingFinger3_L
37	"left_thumb1"	ThumbFinger1_L
38	"left_thumb2"	ThumbFinger2_L
39	"left_thumb3"	ThumbFinger3_L
40	"right_index1"	IndexFinger1_R
41	"right_index2"	IndexFinger2_R
42	"right_index3"	IndexFinger3_R
43	"right_middle1"	MiddleFinger1_R
44	"right_middle2"	MiddleFinger2_R
45	"right_middle3"	MiddleFinger3_R
46	"right_pinky1"	PinkyFinger1_R
47	"right_pinky2"	PinkyFinger2_R
48	"right_pinky3"	PinkyFinger3_R
49	"right_ring1"	RingFinger1_R
50	"right_ring2"	RingFinger2_R
51	"right_ring3"	RingFinger3_R
52	"right_thumb1"	ThumbFinger1_R
53	"right_thumb2"	ThumbFinger2_R
54	"right_thumb3"	ThumbFinger3_R

语音转表情
iOS

最近更新时间：2025-02-28 12:24:13

功能说明

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

接入方式

方式1：与美颜特效 SDK 一起使用

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

方式2：通过独立的语音转表情 SDK 接入

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

接入步骤

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

接口说明

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

集成代码示例

```
// 初始化语音转表情sdk
NSString *path = [[NSBundle mainBundle] pathForResource:@"audio2exp" ofType:@"bundle"];
```

```
int ret = [Audio2ExpApi initWithModelPath:path];
// 语音数据转52表情数据
NSArray *emotionArray = [Audio2ExpApi parseAudio:floatArr];
// 释放sdk
[Audio2ExpApi releaseSdk];

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

说明：
完整的示例代码请参考 美颜特效 SDK demo 工程 。

- 录音可以参考 `TXCAudioRecorder` 。
- 接口使用可以参考： `VoiceViewController` 及其相关类。

Android

最近更新时间：2025-02-28 12:24:13

功能说明

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

接入方式

方式1：通过美颜特效 SDK 接入

语音转表情集成在美颜特效 SDK 中，因此第一步需要按照美颜特效文档进行接入。

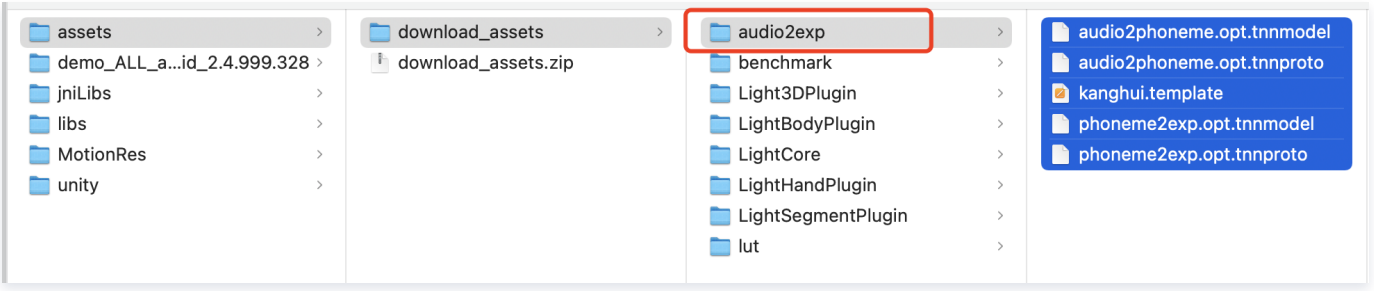
1. 下载 美颜特效 SDK 完整版 。
2. 参考 独立集成美颜特效 文档完成集成。

方式2：通过独立的语音转表情 SDK 接入

如果您只需要语音转表情，不需要用到美颜特效 SDK 的任何能力，则可以考虑使用独立的语音转表情 SDK，aar 包约6MB左右。请联系我们的架构师或销售获取此 SDK。

接入步骤

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



接口说明

接口	说明
public int Audio2ExpApi.init(String modelPath);	初始化，传入模型路径，见上文说明。返回值为0表示成功
public float[] Audio2ExpApi.parseAudio(float[] inputData);	输入的是音频数据，要求单通道，16K采样率，数组长度为267（即267个采样点），输出的数据是长度为52的float数组，表示52表情基，取值为0到1之间，顺序为 苹果标准顺序

	`{"eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft","eyeSquintLeft","eyeWideLeft","eyeBlinkRight","eyeLookDownRight","eyeLookInRight","eyeLookOutRight","eyeLookUpRight","eyeSquintRight","eyeWideRight","jawForward","jawLeft","jawRight","jawOpen","mouthClose","mouthFunnel","mouthPucker","mouthRight","mouthLeft","mouthSmileLeft","mouthSmileRight","mouthFrownRight","mouthFrownLeft","mouthDimpleLeft","mouthDimpleRight","mouthStretchLeft","mouthStretchRight","mouthRollLower","mouthRollUpper","mouthShrugLower","mouthShrugUpper","mouthPressLeft","mouthPressRight","mouthLowerDownLeft","mouthLowerDownRight","mouthUpperUpLeft","mouthUpperUpRight","browDownLeft","browDownRight","browInnerUp","browOuterUpLeft","browOuterUpRight","cheekPuff","cheekSquintLeft","cheekSquintRight","noseSneerLeft","noseSneerRight","tongueOut"}`
public int Audio2ExpApi.release();	使用完毕后调用，释放资源

集成代码示例

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

说明
完整的示例代码请参考 美颜特效 SDK demo 工程 。

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

API 文档
iOS

最近更新时间：2025-02-28 12:24:14

美颜特效 SDK 核心接口类 `XMagic.h`，用于初始化 SDK、更新美颜数值、调用动效等功能。

XMagic静态方法列表

API	描述
version	SDK当前版本名称
getDeviceLevel	获取设备等级，V3.7.0新增。您可以根据设备等级 开启或关闭 SDK 的相关功能，或在低等级手机上设置 高性能模式。

实例方法列表

API	描述
initWithRenderSize	初始化接口
initWithGlTexture	初始化接口
setEffect	配置美颜各种效果(3.5.0.2新增)
emitBlurStrengthEvent	设置后处理模糊强度（作用于所有模糊组件）
setRenderSize	设置 renderSize
deinit	资源释放接口
process:	图像数据处理接口，输入美颜前的图像，返回美颜后的图像
process:withOrigin:withOrientation:	图像数据处理接口，输入美颜前的图像，返回美颜后的图像，比上接口多两个参数
exportCurrentTexture	导出当前形象的图片。通过接口 `process` 和 `process:withOrigin:withOrientation:` 处理美颜以后，可以使用该接口获取当前形象的图片
processUIImage	处理图片
getConfigPropertyWithName	获取美颜参数配置信息
registerLoggerListener	日志注册接口
registerSDKEventListener	SDK 事件监听接口
clearListeners	注册回调清理接口
getCurrentGlContext	获取当前 GL 上下文接口
onPause	SDK 暂停接口
onResume	SDK 恢复接口
setAudioMute	动效素材使用时是否开启静音（V2.5.0.0新增） 参数：YES 表示静音，NO 表示非静音
setFeatureEnableDisable	设置某个特性的开启或关闭
素材叠加	如果想要某个动效/美妆/分割素材叠加在当前素材上，则设置该素材时，在 withExtraInfo 的字典中设置 mergeWithCurrentMotion 为 true
EffectMode(高性能模式)	使用参考 EffectMode（高性能模式）使用指引。

废弃方法列表

API	描述
configPropertyWithType	配置美颜各种效果。废弃，建议使用 setEffect 。
高性能模式	在 SDK 初始化时，字典中添加 `@"setDowngradePerformance":@(YES)` 。废弃，建议使用 EffectMode（高性能模式）使用指引 。
enableEnhancedMode	开启美颜增强模式（V2.5.1新增）。废弃，使用参考 增强模式使用指引 。
isBeautyAuthorized	获取美颜参数授权信息

getDeviceLevel

获取设备等级，V3.7.0新增。您可以根据设备等级 开启或关闭SDK的相关功能 ，或在低等级手机上设置 高性能模式 。

```
typedef NS_ENUM(NSInteger, DeviceLevel) {
    DEVICE_LEVEL_VERY_LOW = 1,
    DEVICE_LEVEL_LOW = 2,
    DEVICE_LEVEL_MIDDLE = 3,
    DEVICE_LEVEL_MIDDLE_HIGH = 4,
    DEVICE_LEVEL_HIGH = 5
};
+ (DeviceLevel)getDeviceLevel;
```

initWithRenderSize

初始化接口

```
- (instancetype _Nonnull)initWithRenderSize:(CGSize)renderSize
                        assetsDict:(NSDictionary* _Nullable)assetsDict;
```

参数

参数	含义
renderSize	渲染尺寸
assetsDict	资源 Dict

initWithGlTexture

初始化接口

```
- (instancetype _Nonnull)initWithGlTexture:(unsigned)textureID
                        width:(int)width
                        height:(int)height
                        flipY:(bool)flipY
                        assetsDict:(NSDictionary* _Nullable)assetsDict;
```

参数

参数	含义
textureID	纹理 ID
width	渲染尺寸
height	渲染尺寸

flipY	是否翻转图片
assetsDict	资源 Dict

setEffect（3.5.0.2新增）

配置美颜各种效果，具体使用示例请参见 美颜参数说明 。

```
- (void)setEffect:(NSString * _Nullable)effectName
      effectValue:(int)effectValue
     resourcePath:(NSString * _Nullable)resourcePath
        extraInfo:(NSDictionary * _Nullable)extraInfo;
```

参数

参数	含义
effectName	效果类型
effectValue	效果数值
resourcePath	素材路径
extraInfo	预留扩展，附加额外配置

emitBlurStrengthEvent

设置后处理模糊强度（作用于所有模糊组件）。

```
- (void)emitBlurStrengthEvent:(int)strength;
```

参数

参数	含义
strength	效果数值

setRenderSize

设置 renderSize。

```
- (void)setRenderSize:(CGSize)size;
```

参数

参数	含义
size	渲染尺寸

deinit

资源释放接口。

```
- (void)deinit;
```

process

处理数据接口，输入的数据格式有 `YTImagePixelData`、`YTTextureData`、`YTImageRawData`、`YTUIImageData` ，输出对应的数据格式。其中 `YTImagePixelData` 中的像素格式为 `RGBA`，`YTTextureData` 中的纹理格式为 `OpenGL 2D` 。

```
/// @brief 处理输入4选1     Process input 4 choose 1
@interface YTProcessInput : NSObject
/// 相机数据对象      camera data object
@property (nonatomic, strong) YTImagePixelData * _Nullable pixelData;
/// 纹理对象      texture object
@property (nonatomic, strong) YTTextureData * _Nullable textureData;
/// 原始数据对象      raw data object
@property (nonatomic, strong) YTImageRawData * _Nullable rawData;
/// UIImage对象      UIImage object
@property (nonatomic, strong) YTUIImageData * _Nullable UIImageData;
/// 输入数据类型      input data type
@property (nonatomic) enum YTProcessDataType dataType;
@end
/// @brief 处理输出      process output
@interface YTProcessOutput : NSObject
/// 纹理输出对象（一定会有）     Texture output object (always output)
@property (nonatomic, strong) YTTextureData * _Nullable textureData;
/// 相机输出对象（如果输入是相机采集数据）     Camera output object (if the input is camera acquisition data)
@property (nonatomic, strong) YTImagePixelData * _Nullable pixelData;
/// 原始输出对象（如果输入是原始数据）     raw output object (if input is raw data)
@property (nonatomic, strong) YTImageRawData * _Nullable rawData;
/// UIImage输出对象（如果输入是UIImage对象）      UIImage output object (if the input is a UIImage object)
@property (nonatomic, strong) YTUIImageData * _Nullable UIImageData;
/// 输出数据类型      output data type
@property (nonatomic) enum YTProcessDataType dataType;
@end

- (YTProcessOutput* _Nonnull)process:(YTProcessInput * _Nonnull)input;
```

参数

参数	含义
input	输入处理数据信息，输入的格式四选一（YTImagePixelData、YTTextureData、YTImageRawData、YTUIImageData）

YTProcessInput输入数据类型和说明

调用 process 接口时，输出数据类型跟输入数据类型一致，YTTextureData类型总是会输出。

类型	含义
YTImagePixelData	相机数据对象，像素格式为 RGBA
YTTextureData	纹理对象，纹理格式为 OpenGL 2D
YTImageRawData	原始数据对象
YTUIImageData	UIImage 对象

process:withOrigin:withOrientation:

处理数据接口，输入和输出的数据格式跟 process 一致。withOrigin：设置图像是否上下镜像翻转，withOrientation：设置图像旋转方向。

```
- (YTProcessOutput* _Nonnull)process:(YTProcessInput* _Nonnull)input withOrigin:(YtLightImageOrigin)origin
withOrientation:(YtLightDeviceCameraOrientation)orientation;
```

参数

参数	含义
input	输入处理数据信息
withOrigin	枚举值（YtLightImageOriginTopLeft、YtLightImageOriginBottomLeft），设置成 YtLightImageOriginBottomLeft 时，图像上下镜像翻转
withOrientation	枚举值：图像旋转角度，设置角度会修改输出的图像角度

exportCurrentTexture

导出当前形象的图片。通过接口 `process` 和 `process:withOrigin:withOrientation:` 处理美颜以后，可以使用该接口获取当前形象的图片。

```
/// @brief 导出当前形象的图片      Export a picture of the current image
- (void)exportCurrentTexture:(nullable void (^)(UIImage *_Nullable image))callback;
```

TEImageTransform 工具类

图像处理工具类，输入/输出的数据格式有 CVPixelBufferref、texture id。支持 CVPixelBufferref 数据的 bgra<-->yuv 格式的相互转换、旋转和上下/左右镜像。支持 texture id 格式输入的旋转和上下/左右镜像。

```
/// @param context 如果使用本类OpenGL接口,建议使用本方法初始化,可传[xMgiac getCurrentGlContext]
- (instancetype)initWithEAGLContext:(EAGLContext *)context;
```

参数	含义
context	使用OpenGLES的上下文环境，可传[xMagic getCurrentGlContext]

```
/// @brief CVPixelBufferRef yuv/rgb相互转换接口，目前只支持TEPixelFormatType内的三种类型转换    CVPixelBufferRef yuv/rgb transfor interface
/// @param pixelBuffer 输入pixelBuffer      input pixelBuffer
/// @param outputFormat 指定输出pixelBuffer的类型     out pixelBuffer format
- (CVPixelBufferRef)transformCVPixelBufferToBuffer:(CVPixelBufferRef)pixelBuffer outputFormat:(TEPixelFormatType)outputFormat;
```

参数	含义
pixelBuffer	输入的 pixelBuffer 数据
outputFormat	输出的 pixelBuffer 格式，支持 BGRA、NV12F(kCVPixelFormatType_420YpCbCr8BiPlanarFullRange)、NV12V(kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange)

```
/// 将yuv/rgb pixelBuffer转换为bgra格式的纹理id
/// @param pixelBuffer 输入pixelBuffer
- (GLuint)transformPixelBufferToBGRATexture:(CVPixelBufferRef)pixelBuffer;
```

参数	含义
pixelBuffer	输入的 pixelBuffer 数据，支持 BGRA、NV12F(kCVPixelFormatType_420YpCbCr8BiPlanarFullRange)、NV12V(kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange)

```
/// 对CVPixelBufferRef进行旋转和镜像翻转， 如果同时传旋转和镜像，处理逻辑为先镜像再旋转
```

```
- (CVPixelBufferRef)convertCVPixelBuffer:(CVPixelBufferRef)pixelBuffer rotaion:
(YtLightDeviceCameraOrientation)rotation flip:(TEFlipType)flipType;
```

参数	含义
pixelBuffer	输入的 pixelBuffer 数据
rotation	逆时针旋转角度，支持0度、90度、180度、270度。
flipType	镜像类型，水平镜像或者垂直镜像。如果同时传旋转和镜像，处理逻辑为先镜像再旋转

```
/// 对纹理Id进行旋转/镜像翻转，如果同时传旋转和镜像，处理逻辑为先镜像再旋转

- (GLuint)convert:(GLuint)srcId width:(int)width height:(int)height rotaion:
(YtLightDeviceCameraOrientation)rotation flip:(TEFlipType)flipType;
```

参数	含义
srcId	输入的纹理 ID
width	纹理的宽度
height	纹理的高度
rotation	逆时针旋转角度，支持0度、90度、180度、270度。
flipType	镜像类型，水平镜像或者垂直镜像。如果同时传旋转和镜像，处理逻辑为先镜像再旋转

processUIImage

处理图片。

```
- (UIImage* _Nullable)processUIImage:(UIImage* _Nonnull)inputImage needReset:(bool)needReset;
```

参数

参数	含义
inputImage	输入图片建议最大尺寸 2160×4096。超过这个尺寸的图片人脸识别效果不佳或无法识别到人脸，同时容易引起 OOM 问题，建议把大图缩小后再传入
needReset	以下场景中 needReset 需设置 为 true： • 切换图片 • 首次使用分割 • 首次使用动效 • 首次使用美妆

getConfigPropertyWithName

获取美颜参数配置信息。

```
- (YTBeautyPropertyInfo * _Nullable)getConfigPropertyWithName:(NSString *_Nonnull)propertyName;
```

参数

参数	含义
propertyName	配置名称

registerLoggerListener

日志注册接口。

```
- (void)registerLoggerListener:(id<YTSDKLogListener> _Nullable)listener withDefaultLevel:
(YtSDKLoggerLevel)level;
```

参数

参数	含义
listener	日志回调接口
level	日志输出 level，默认 ERROR

registerSDKEventListener

SDK 事件监听接口。

```
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;
```

参数

参数	含义
listener	事件监听器回调，主要分为 AI 事件，Tips 提示事件，Asset 事件

clearListeners

注册回调清理接口。

```
- (void)clearListeners;
```

getCurrentGlContext

获取当前 GL 上下文接口。

```
- (nullable EAGLContext*)getCurrentGlContext;
```

onPause

SDK 暂停接口，暂停特效里的声音播放。

```
/// @brief APP暂停时候需要调用SDK暂停接口，暂停特效里的声音播放
- (void)onPause;
```

onResume

SDK 恢复接口，恢复特效里的声音播放。

```
/// @brief APP恢复时候需要调用SDK恢复接口，恢复特效里的声音播放
- (void)onResume;
```

setAudioMute

动效素材使用时是否开启静音（V2.5.0新增）。

```
/// @brief 设置静音
```

```
- (void)setAudioMute:(BOOL)isMute;
```

setFeatureEnableDisable

设置某个特性开启或关闭或关 setFeatureEnableDisable。

```
/// @brief 设置某个特性的开或关      Set a feature on or off
/// @param featureName 取值见TEDefine FeatureName
/// @param enable 开或关      on or off
- (void)setFeatureEnableDisable:(NSString *_Nonnull)featureName enable:(BOOL)enable;
```

参数

参数	含义
featureName	原子能力名称 取值如下： • ANIMOJI_52_EXPRESSION 人脸表情能力 • BODY_3D_POINT 身体点位能力 • HAND_DETECT 手势检测能力 • BEAUTY_ONLY_WHITEN_SKIN 美白仅对皮肤生效 • SEGMENTATION_SKIN 皮肤分割能力 • SMART_BEAUTY 智能美颜(为男性、宝宝减淡美颜美妆效果)
enable	YES 表示开启此能力，NO 表示关闭此能力 **注：如果是降级模式，则不允许开启皮肤分割**

回调

API	描述
YTSDKEventListener	SDK 内部事件回调接口
YTSDKLogListener	日志监听回调

YTSDKEventListener

SDK 内部事件回调接口

```
@protocol YTSDKEventListener <NSObject>
```

成员函数

返回类型	名称
void	onAIEvent
void	onTipsEvent
void	onAssetEvent

函数说明

onAIEvent

AI 事件回调

```
/// @param event dict格式的回调
```

```
- (void)onAIEvent:(id _Nonnull)event;
```

最多返回5个人脸信息：

```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
    180.0,
    112.2,
    ...
  ],
  "face_256_visible":[
    0.85,
    ...
  ],
  "out_of_screen":true,
  "left_eye_high_vis_ratio":1.0,
  "right_eye_high_vis_ratio":1.0,
  "left_eyebrow_high_vis_ratio":1.0,
  "right_eyebrow_high_vis_ratio":1.0,
  "mouth_high_vis_ratio":1.0
 },
 ...
 ]
}
```

字段含义

字段	类型	值域	说明
trace_id	int	[1,INF)	人脸 ID，连续取流过程中，ID 相同的可以认为是同一张人脸
face_256_point	float	[0,screenWidth] 或 [0,screenHeight]	共512个数，人脸256个关键点，屏幕左上角为(0,0)
face_256_visible	float	[0,1]	人脸256关键点可见度
out_of_screen	bool	true/false	人脸是否出框
left_eye_high_vis_ratio	float	[0,1]	左眼高可见度点位占比
right_eye_high_vis_ratio	float	[0,1]	右眼高可见度点位占比
left_eyebrow_high_vis_ratio	float	[0,1]	左眉高可见度点位占比
right_eyebrow_high_vis_ratio	float	[0,1]	右眉高可见度点位占比
mouth_high_vis_ratio	float	[0,1]	嘴高可见度点位占比

onTipsEvent

提示事件回调

```
/// @param event dict格式的回调
```

```
- (void)onTipsEvent:(id _Nonnull)event;
```

onAssetEvent

资源包事件回调

```
/// @param event string格式的回调
- (void)onAssetEvent:(id _Nonnull)event;
```

YTSDKLogListener

日志监听回调

```
@protocol YTSDKLogListener <NSObject>
```

成员函数

返回类型	函数名称
void	onLog

函数说明

onLog

日志监听回调

```
/// @param loggerLevel 返回当前日志等级
/// @param logInfo 返回当前日志信息
- (void)onLog:(YtSDKLoggerLevel) loggerLevel withInfo:(NSString * _Nonnull) logInfo;
```

素材叠加(3.0.1.5新增)

如果想要某个动效/美妆/分割素材叠加在当前素材上，则设置该素材时，在 withExtraInfo 的字典中设置 mergeWithCurrentMotion 为 true，示例如下：

```
NSString *key = _xmagicUIProperty.property.Id;
NSString *value = [[NSBundle mainBundle] pathForResource:@"makeupMotionRes" ofType:@"bundle"];
NSDictionary* extraInfo = @{@"mergeWithCurrentMotion":@(true)};
 [self.beautyKitRef configPropertyWithType:@"motion" withName:key withData:[NSString
stringWithFormat:@"%@",value] withExtraInfo:extraInfo];
```

素材叠加注意事项：

1. 客户需要自行管理素材之间是否适合叠加。举两个例子：
   例1：特效 A 是变成贵妃脸，特效 B 是变成童话脸，这两个特效叠加后可能会导致画面非常别扭。
   例2：特效 A 是个兔耳朵，特效 B 是猪耳朵，两个叠加后，就有两种耳朵。
   例1和例2这两种情况不适合叠加。如果特效 A 是兔耳朵，特效 B 是送一个飞吻，这两个特效不会冲突，就适合叠加。
2. 只支持简单素材之间的叠加。简单素材是指只有单动效能力、或者单美妆效果、或者单抠背等，复杂素材是指包含了多种效果。简单素材和复杂素材没有明确的界定，建议客户充分测试后，自行管理哪些素材之间可以叠加，哪些不能叠加。
3. 叠加时，有动作触发的特效（例如伸出手触发某个特效、微笑触发某个特效等）属于复杂特效，需要放在前面，简单特效放在后面叠加在它之上。
4. 使用示例：主播使用了特效 A，然后观众送礼物特效 B，B 要叠加在 A 之上，一段时间后 B 消失，恢复成特效 A。那么设置步骤如下：
   4.1 设置特效 A，mergeWithCurrentMotion 设置为 false。
   4.2 设置特效 B，mergeWithCurrentMotion 设置为 true。
   4.3 一小段时间后，再设置 A，mergeWithCurrentMotion 设置为 false。

XMagic 废弃方法：

configPropertyWithType

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;
```

参数

参数	含义
propertyType	效果类型
propertyName	效果名称
propertyValue	效果数值
extraInfo	预留扩展, 附加额外配置 Dict

配置美颜效果示例

- **美颜：**配置美白效果

```
NSString *propertyType = @"beauty";         //配置美颜的效果类型，这里以美颜为例
NSString *propertyName = @"beauty.whiten"; //配置美颜的名称，这里以美白为例
NSString *propertyValue = @"60";            //配置美白的效果数值
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue withExtraInfo:nil];
```

- **滤镜：**配置心动效果

```
NSString *propertyType = @"lut";         //配置美颜的效果类型，这里以滤镜为例
NSString *propertyName = [@"lut.bundle/" stringByAppendingPathComponent:@"xindong_lf.png"]; //配置美颜的名称，这里以心动为例
NSString *propertyValue = @"60";            //配置滤镜的效果数值
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue withExtraInfo:nil];
```

- **美体：**配置长腿效果

```
NSString *propertyType = @"body";         //配置美颜的效果类型，这里以美体为例
NSString *propertyName = @"body.legStretch"; //配置美颜的名称，这里以长腿为例
NSString *propertyValue = @"60";            //配置长腿的效果数值
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue withExtraInfo:nil];
```

- **动效：**配置2D动效的可爱涂鸦效果

```
 NSString *motion2dResPath = [[NSBundle mainBundle] pathForResource:@"2dMotionRes" ofType:@"bundle"];//这里是2dMotionRes文件夹的绝对路径
 NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以动效为例
 NSString *propertyName = @"video_keaituya"; //配置美颜的名称，这里以2D动效的可爱涂鸦为例
 NSString *propertyValue = motion2dResPath;  //配置动效的路径
 [self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue withExtraInfo:nil];
```

- **美妆：**配置女团妆效果

```
NSString *motionMakeupResPath = [[NSBundle mainBundle] pathForResource:@"makeupMotionRes"
ofType:@"bundle"];//这里是makeupMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以美妆为例
NSString *propertyName = @"video_nvtuanzhuang"; //配置美颜的名称，这里以女团妆为例
NSString *propertyValue = motionMakeupResPath;  //配置动效的路径
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:nil];
//下面是要配置美妆的数值（上面的动效只需要调用一次，下面的配置美妆数值可以多次调用）
 NSString *propertyTypeMakeup = @"custom";          //配置美颜的效果类型，这里以美妆为例
 NSString *propertyNameMakeup = @"makeup.strength"; //配置美颜的名称，这里以女团妆为例
 NSString *propertyValueMakeup = @"60";              //配置美妆的效果数值
 [self.xmagicApi configPropertyWithType:propertyTypeMakeup withName:propertyNameMakeup
withData:propertyValueMakeup withExtraInfo:nil];
```

- **分割：配置背景模糊（强效果）**

```
NSString *motionSegResPath = [[NSBundle mainBundle] pathForResource:@"segmentMotionRes"
ofType:@"bundle"];//这里是segmentMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以分割为例
NSString *propertyName = @"video_segmentation_blur_75"; //配置美颜的名称，这里以背景模糊-强为例
NSString *propertyValue = motionSegResPath;  //配置动效的路径
NSDictionary *dic = @{@"bgName":@"BgSegmentation.bg.png", @"bgType":@0, @"timeOffset":
@0},@"icon":@"segmentation.linjian.png"};//配置预留字段
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:dic];
```

- **自定义背景：**

```
NSString *motionSegResPath = [[NSBundle mainBundle] pathForResource:@"segmentMotionRes"
ofType:@"bundle"];//这里是segmentMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以分割为例
NSString *propertyName = @"video_empty_segmentation"; //配置美颜的名称，这里以自定义背景为例
NSString *propertyValue = motionSegResPath;  //配置动效的路径
NSString *imagePath = @"/var/mobile/Containers/Data/Application/06B00BBC-9060-450F-8D3A-
F6028D185682/Documents/MediaFile/image.png"; //自定义背景图片的绝对路径。如果自定义背景选择的是视频，需要对视频进行
压缩转码处理，使用压缩转码处理后的绝对路径
int bgType = 0;//自定义背景的类型。 0表示图片，1表示视频
int timeOffset = 0; //时长。图片背景时，为0；视频背景时为视频的时长
NSDictionary *dic = @{@"bgName":imagePath, @"bgType":@(bgType), @"timeOffset":
@(timeOffset)},@"icon":@"segmentation.linjian.png"};//配置预留字段
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:dic];
```

高性能模式（V3.1.0新增）

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。

```
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle] bundlePath],
                                @"setDowngradePerformance":@(YES) //开启高性能模式

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

enableEnhancedMode

```
/// @brief 开启美颜增强模式
- (void)enableEnhancedMode;
```

开启美颜增强模式（V2.5.1新增）。默认未开启。

- 未开启时，应用层可以设置的各美颜项的强度范围为0到1或-1到1，如果超出此范围，SDK 会取边界值。例如应用层设置瘦脸为1.2，SDK 判断其超出了最大值1.0，则在内部把瘦脸值修正为1.0。
- 开启增强模式后，应用层可以设置更大范围的数值。例如想要瘦脸程度更大，则可以把瘦脸值设置为1.2，SDK 会接受并使用1.2这个数值，不会将其修正为1.0。

开启增强模式后，需要应用层自己管理每个美颜项可以设置的最大值，让用户在此范围内调整数值。我们提供了一份参考值，您可以根据产品需求自由调整，但不建议超出我们的推荐值，否则美颜效果可能变差。参考值见下：

美颜项名称	增强模式下，建议的最大值（放大倍数）
美白，短脸，V脸，眼距，鼻子位置，祛法令纹，口红，立体	1.3
亮眼	1.5
腮红	1.8
其他	1.2

isBeautyAuthorized

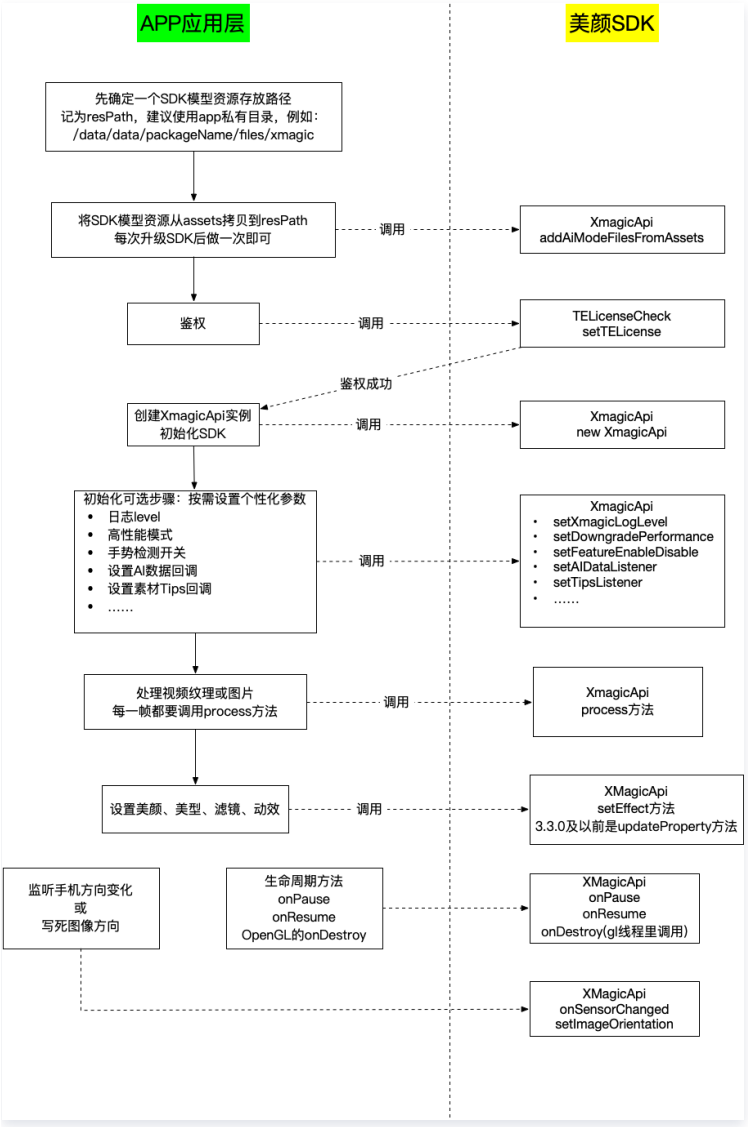
获取该美颜参数的授权信息（仅支持美颜和美体）

```
/// @param featureId 配置美颜参数
/// @return 返回对应美颜参数的授权结果
+ (BOOL)isBeautyAuthorized:(NSString * _Nullable)featureId;
```

Android

最近更新时间：2025-02-28 12:24:14

美颜特效 SDK 核心接口类 `XmagicApi.java`，用于初始化 SDK、更新美颜数值、调用动效等功能。
各 API 整体调用流程如下：



XmagicApi 的静态属性和方法列表

API	描述
VERSION	通过XmagicApi.VERSION可获取SDK的版本号（V3.5.0新增）。
setLibPathAndLoad	设置 so库的路径，如果so库是内置在apk包里的，则无需使用该接口。
addAiModeFilesFromAssets	将应用程序 assets 下 Light3DPlugin、LightCore、LightHandPlugin、LightBodyPlugin、LightSegmentPlugin 文件夹中的内容复制到指定目录中。
addAiModeFiles	将客户下载好的 AI 模型文件复制到对应的文件夹下。
getDeviceLevel	获取设备等级。

XmagicApi 的 Public 方法列表

API	描述

XmagicApi	构造函数。
process	SDK 渲染数据的方法，用于处理图片或视频流。
setEffect	设置美颜、美型、滤镜、美妆、贴纸、分割等效果，可在任意线程调用。
setXmagicLogLevel	设置 SDK 的 log 等级，默认为 `Log.WARN` 。开发调试阶段如有需要，可以将其设为 `Log.DEBUG`。正式发布时务必设置为 `Log.WARN` 或 `Log.ERROR` ，否则大量的日志会影响性能。 **在 new XmagicApi() 之后调用。**
enableHighPerformance	调用此方法开启高性能模式。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。 **在 new XmagicApi() 之后调用。**
setFeatureEnableDisable	开启或关闭某个能力。
setXmagicStreamType	设置输入数据类型，有相机数据和图片数据两类，默认是相机数据流。
setAIDataListener	设置人脸、手势、身体检测状态回调。
setAudioMute	动效素材使用时是否开启静音。参数：true 表示静音，false 表示非静音。
onPause	暂停特效里的声音播放，可与 Activity onPause 生命周期绑定。
onResume	恢复特效里的声音播放，可与 Activity onResume 生命周期绑定。
onDestroy	销毁 xmagic，需要在 GL 线程中调用。
setImageOrientation	设置图像方向，使 AI 对不同朝向的人脸都能识别到，如果设置，则忽略 `sensorChanged` 传入的方向。
sensorChanged	利用系统传感器判断当前手机旋转的角度，使 AI 对不同朝向的人脸都能识别到。
isDeviceSupport	将一个动效素材的路径传给SDK，检测当前设备是否完全支持这个动效。
isSupportBeauty	判断当前机型是否支持美颜（OpenGL3.0）。
exportCurrentTexture	获取当前纹理上的画面
setTipsListener	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。
• updateProperty • updateProperties • setYTDataListener • onPauseAudio • getDeviceAbilities • getPropertyRequiredAbilities • isBeautyAuthorized • enableEnhancedMode • setDowngradePerformance	已废弃接口，不建议使用。详情请点击 这里 。

静态方法 setLibPathAndLoad

设置 so 库的路径，并触发加载。
如果 so 库是内置在 apk 包里的，则无需使用该接口。如果 so 是动态下载的，则需要在鉴权和 `new XmagicApi()` 之前调用。

- 传入 null：表示从默认路径加载 so，请确保 so 是内置在 APK 包里的。
- 传入非 null：如 `data/data/包名/files/xmagic_libs` ，将从这个目录去加载 so。

```
static boolean setLibPathAndLoad(String path)
```

静态方法 addAiModeFilesFromAssets

SDK 的模型资源文件，可以内置在 apk 包的 assets 目录，也可以动态下载。如果是内置在 assets 里的，那么首次使用 SDK 之前，需要将 assets 下的模型资源文件复制到指定目录中。这个操作只需要做一次。在您升级 SDK 版本之后，才需要再执行一次。

- context 应用上下文。
- resDir 用于存放SDK模型资源的根目录，此目录和创建 new XmagicApi() 对象时传入的路径参数一致。
- 返回值：
  - 0：复制成功
  - -1：表示 context 为 null
  - -2：表示 IO 错误

```
static int addAiModeFilesFromAssets(Context context, String resDir)
```

静态方法 addAiModeFiles

如果您未将 SDK 的模型资源文件内置在APK包里，而是动态下载，那么下载成功并解压后，可以调用此方法将下载好的 SDK 模型资源文件复制到对应的文件夹下。

- inputResDir 下载成功的模型文件的文件夹。
- resDir 用于存放 SDK 模型资源的根目录，此目录和创建 new XmagicApi() 对象时传入的路径参数一致。
- 返回值：
  - 0：表示成功
  - -1：inputResDir is not exists
  - -2：表示 IO 错误

```
static int addAiModeFiles(String inputResDir, String resDir)
```

静态方法 getDeviceLevel

获取设备等级，V3.7.0新增。您可以根据设备等级 开启或关闭 SDK 的相关功能，或在低等级手机上设置 高性能模式 。

> ⚠ 注意：
> - V3.7.0 – V3.9.0期间，如果您需要在 `new XmagicApi` 之前就调用 getDeviceLevel 方法，则需要将 SDK 的 assets 目录下的 benchmark 文件夹内置在您的 APK 的 assets 里。如果在 `new XmagicApi` 之后才调用 getDeviceLevel，则无需内置。
> - 我们将在V3.9.1版本优化此问题。

```
static DeviceLevel getDevicLevel(Context context)

public enum DeviceLevel {
    DEVICE_LEVEL_VERY_LOW(1),
    DEVICE_LEVEL_LOW(2),
    DEVICE_LEVEL_MIDDLE(3),
    DEVICE_LEVEL_MIDDLE_HIGH(4),
    DEVICE_LEVEL_HIGH(5);

    private final int value;

    DeviceLevel(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

XmagicApi 构造方法

```
public XmagicApi(Context context, EffectMode effectMode, String resDir)

public XmagicApi(Context context, EffectMode effectMode, String resDir,OnXmagicPropertyErrorListener
xmagicPropertyErrorListener)

@Deprecated
public XmagicApi(Context context, String resDir)
@Deprecated
public XmagicApi(Context context, String resDir,OnXmagicPropertyErrorListener xmagicPropertyErrorListener)

public interface OnXmagicPropertyErrorListener {
    void onXmagicPropertyError(String errorMsg, int code);
}

public enum EffectMode{
    NORMAL(0),
    PRO(1);

    private final int value;

    EffectMode(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

参数	类型	含义
context	Context	上下文。
effectMode	EffectMode	V3.9.0新增此参数。 用于指定性能和效果模式，可取值 NORMAL 和 PRO。NORMAL 能满足大部分场景，性能更好，但功能会少一些，请按需选用。 二者具体区别见：EffectMode（高性能模式）使用指引。
resDir	String	SDK模型资源文件目录。 在使用SDK之前，需要先确定一个目录用于存放SDK的模型资源文件，建议使用app私有目录： `resDir = context.getFilesDir().getAbsolutePath() + "/xmagic"` 并且在 new XmagicApi 之前，请先调用 XmagicApi 的静态方法 addAiModeFilesFromAssets 或 addAiModeFiles 将模型资源文件 copy 到 resDir。
xmagicPropertyErrorListener	OnXmagicPropertyErrorListener	错误回调接口。在使用 SDK 过程中，此接口会回调内部的一些错误信息。一般在开发调试时使用。

OnXmagicPropertyErrorListener 返回错误码含义对照表：

错误码	含义
-1	未知错误。
-100	3D 引擎资源初始化失败。
-200	不支持 GAN 素材。

-300	设备不支持此素材组件。
-400	模板 JSON 内容为空。
-500	SDK 版本过低。
-600	不支持分割。
-700	不支持 OpenGL。
-800	不支持脚本。
5000	分割背景图片分辨率超过 2160×3840。
5001	分割背景图片所需内存不足。
5002	分割背景视频解析失败。
5003	分割背景视频超过200秒。
5004	分割背景视频格式不支持。
5005	分割背景图片存在旋转角度

process

SDK 渲染数据的方法，用于处理图片或视频流。处理视频流时，可在相机数据回调函数内使用。

```
//渲染纹理
int process(int srcTextureId, int srcTextureWidth, int srcTextureHeight)
//渲染bitmap
Bitmap process(Bitmap bitmap, boolean needReset)
```

参数	含义
int srcTextureId	需要被渲染的纹理。类型为：OpenGL 2D 纹理格式,像素格式为 RGBA。
int srcTextureWidth	需要被渲染的纹理宽。
int srcTextureHeight	需要被渲染的纹理高。
Bitmap bitmap	建议最大尺寸 2160×4096。超过这个尺寸的图片人脸识别效果不佳或无法识别到人脸，同时容易引起 OOM 问题，建议把大图缩小后再传入。
boolean needReset	以下几种场景请将 needReset 设置为 true • 首次 process 一张图片，或切换图片。 • 首次使用分割。 • 首次使用动效。 • 首次使用美妆。

setEffect

设置美颜、美型、滤镜、美妆、贴纸、分割等效果，可在任意线程调用。具体参数请参见 美颜参数说明 。

```
void setEffect(String effectName, int effectValue, String resourcePath, Map<String, String> extraInfo)
```

setXmagicLogLevel

在 new XmagicApi() 之后调用。设置 SDK 的 log 等级，默认为 `Log.WARN` 。开发调试阶段如有需要，可以将其设为 `Log.DEBUG`。 正式发布时务必设置为 `Log.WARN` 或 `Log.ERROR` ，否则大量的日志会影响性能。
如果设置了 `ITELogger` ，则会把SDK内部的日志回调给使用者。

```
public void setXmagicLogLevel(int level);
```

```
public void setXmagicLogLevel(int level, final ITELogger logger);

public interface ITELogger{
    void log(int severity, String tag, String msg);

    void log(int severity, String tag, String msg, Throwable throwable);

}
```

enableHighPerformance

调用此方法开启高性能模式，V3.7.0新增。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。但与常规模式相比，功能会少一些。
需要在 new XmagicApi() 之后立即调用。详细说明见 高性能模式使用指引 。

```
void enableHighPerformance()
```

setFeatureEnableDisable

开启或关闭某个能力，请按需使用。

```
void setFeatureEnableDisable(String featureName, boolean enable)
```

参数	含义
String featureName	原子能力名称 取值如下： • `XmagicConstant.FeatureName.SEGMENTATION_SKIN` 皮肤分割能力，开启后可使磨皮和美白区域更精准。 • `XmagicConstant.FeatureName.SEGMENTATION_FACE_BLOCK` 人脸遮挡检测能力，开启后可避免妆容画到遮挡物上。 • `XmagicConstant.FeatureName.WHITEN_ONLY_SKIN_AREA` 美白仅对皮肤生效 • `XmagicConstant.FeatureName.SMART_BEAUTY` 智能美颜(为男性、宝宝减淡美颜美妆效果) • `XmagicConstant.FeatureName.ANIMOJI_52_EXPRESSION` 人脸表情能力 • `XmagicConstant.FeatureName.BODY_3D_POINT` 身体点位能力 • `XmagicConstant.FeatureName.HAND_DETECT` 手势检测能力
boolean enable	true 表示开启此能力，false 表示关闭此能力

上述能力中，比较常用的是 `SEGMENTATION_SKIN` 和 `SEGMENTATION_FACE_BLOCK` ，SDK 会根据 getDeviceLevel 的值默认开启这两项能力。如果您需要手动开启，建议在 level ⩾ 4 时，开启 `SEGMENTATION_SKIN` ，在 level ⩾ 5时，开启 `SEGMENTATION_FACE_BLOCK` 。

setXmagicStreamType

设置输入数据类型，有相机数据和图片数据两类，默认是相机数据流。

```
void setXmagicStreamType(int type)
```

参数	含义
int type	数据源类型，有以下两种选择： • `XmagicApi.PROCESS_TYPE_CAMERA_STREAM` ：相机数据源。 • `XmagicApi.PROCESS_TYPE_PICTURE_DATA` ：图片数据源。

setAIDataListener

```
public void setAIDataListener(final XmagicAIDataListener aiDataListener);
```

```
public interface OnAIDataListener {
    void onFaceDataUpdated(List<TEFaceData> faceDataList);
    void onBodyDataUpdated(List<TEBodyData> bodyDataList);
    void onAIDataUpdated(String jsonString);
    @Deprecated
    void onHandDataUpdated(List<TEHandData> handDataList);
}
```

onFaceDataUpdated

SDK 开始 process 图像后就会有回调，识别到人脸时，回调 List<FaceData>，list 的 size 就是人脸个数。没有人脸时，回调一个空的 List。
TEFaceData 的定义如下，points 是人脸83点的坐标，每个点有x、y两个坐标值，因此 points 长度固定为166。每个点的坐标值的取值范围不是基于原始输入图，而是将原图的短边缩放到256之后得到的人脸坐标。
因此，这里的回调的数据，建议仅用于判断当面画面中是否有人脸、有几个人脸。如果需要更高精度的人脸点位，请使用 `onAIDataUpdated` 回调的数据。

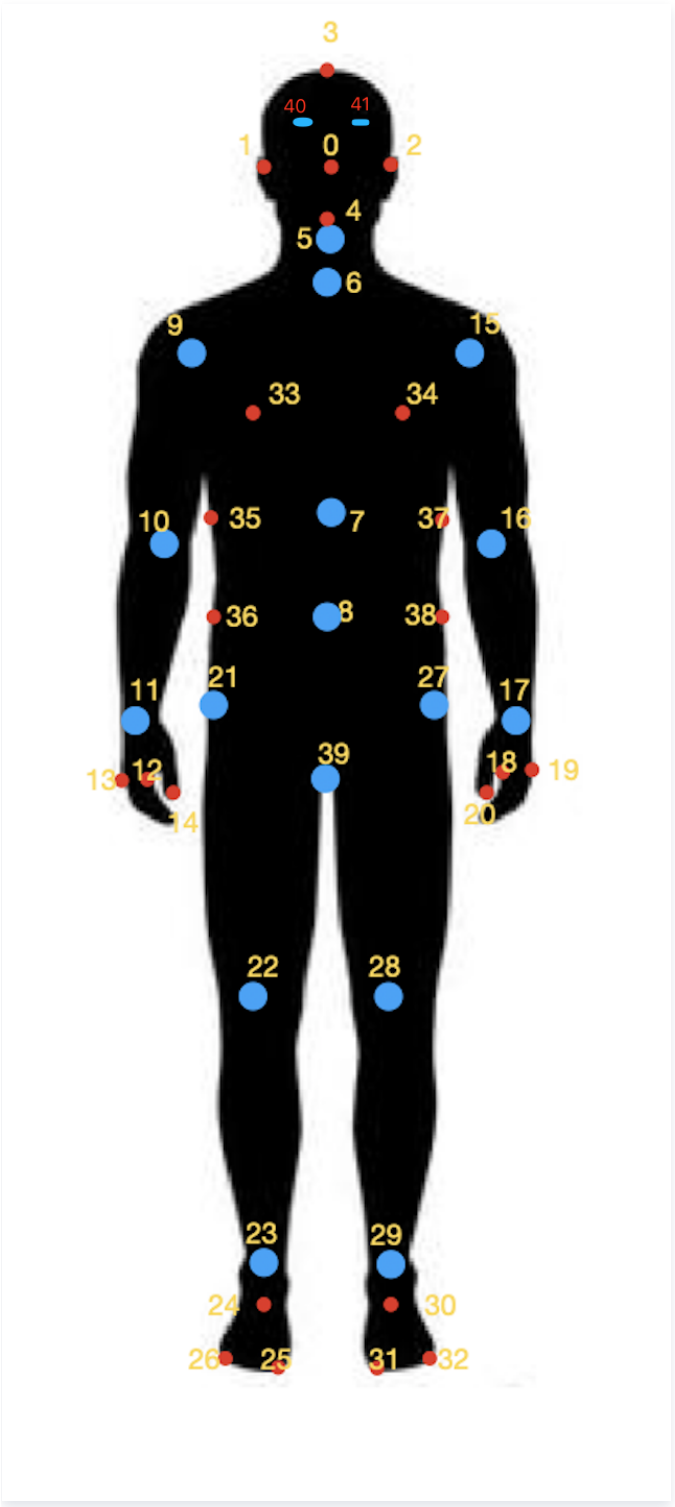
```
public class TEFaceData {
    public float[] points;

    public TEFaceData() {
    }

    public TEFaceData(float[] points) {
        this.points = points;
    }
}
```

onHandDataUpdated

已废弃接口，不会有数据回调。老版本 SDK 在设置了手势动效，且识别到手势时回调。
如果需要手势数据，请使用 `onAIDataUpdated` 回调的数据。

onBodyDataUpdated

设置了美体的属性并且识别到身体的时候回调，其他情况不回调。回调的是身体42点的坐标，坐标值基于输入图的宽高。如果某个点未检测到，则坐标值为0。



onAIDataUpdated

回调人脸、手势、身体3D的详细数据，通常需要使用特定的Licence才有数据。一份示例数据如下：

- 关于 `face_info` 的详细说明，请查看 人脸点位 。
- 关于 `hand_info` 的详细说明，请查看 手势识别 。
- 关于 `body_3d_info` 的详细说明，请查看 身体点位 Android 和 身体点位 iOS 。

```
{
    "face_info": [{
        "expression_weights": [0.001172, 0, 0.029249, ... , 0.060041, 0],
        "face_256_point": [211.844238, 673.247192, ... , 339.247925, 654.792603],
```

```
        "face_256_visible": [0.163925, 0.14921, ... , 0.99887, 0.99887],
        "face_3d_info": {
            "pitch": -3.860844850540161,
            "pitch_fixed": 2.1123428344726562,
            "roll": -12.797032356262207,
            "roll_fixed": 1.3187808990478516,
            "transform": [
                [0.8919625878334045, 0.2843534052371979, 0.3514907658100128, 0],
                [-0.17628398537635803, 0.9346542954444885, -0.3087802827358246, 0],
                [-0.41632509231567383, 0.21345829963684082, 0.88380366563797, 0],
                [-0.020958196371793747, -0.04502145200967789, -0.6078543663024902, 1]
            ],
            "yaw": 24.824481964111328,
            "yaw_fixed": 25.02082061767578
        },
        "left_eye_high_vis_ratio": 0,
        "left_eyebrow_high_vis_ratio": 0,
        "mouth_high_vis_ratio": 1,
        "out_of_screen": false,
        "right_eye_high_vis_ratio": 1,
        "right_eyebrow_high_vis_ratio": 0.821429,
        "trace_id": 21
    }],
    "hand_info": {
        "gesture": "PAPER",
        "hand_point_2d": [180.71888732910156, 569.2958984375, ... , 353.8714294433594, 836.246826171875]
    },
    "body_3d_info": {
        "imageHeight": 652,
        "imageWidth": 320,
        "items": [{
            "index": 1,
            "pose": [0.049122653901576996, ... , 0],
            "position_x": [190.47494506835938, 235.23098754882812, ... , 4.948424339294434,
173.59298706054688],
            "position_y": [777.2109375, 836.488037109375, ... , 161.19752502441406, 405.83905029296875],
            "position_z": [0, 0, ... , 0, 0],
            "rotation": [{
                "data": [0.9944382905960083, -0.09695644676685333, -0.0411277711391449,
0.000708006089553237]
            },
            ......
            {
                "data": [0.9907779693603516, 0.13549542427062988, 0, 0]
            }, {
                "data": [1, 0, 0, 0]
            }]
        }]
    }
}
```


setAudioMute

动效素材使用时是否开启静音（V2.5.0新增）：
参数：true 表示静音，false 表示非静音。

onPause

暂停特效里的声音播放，可与 Activity onPause 生命周期绑定。

```
void onPause()
```

onResume

恢复特效里的声音播放，可与 Activity onResume 生命周期绑定。

```
void onResume()
```

onDestroy

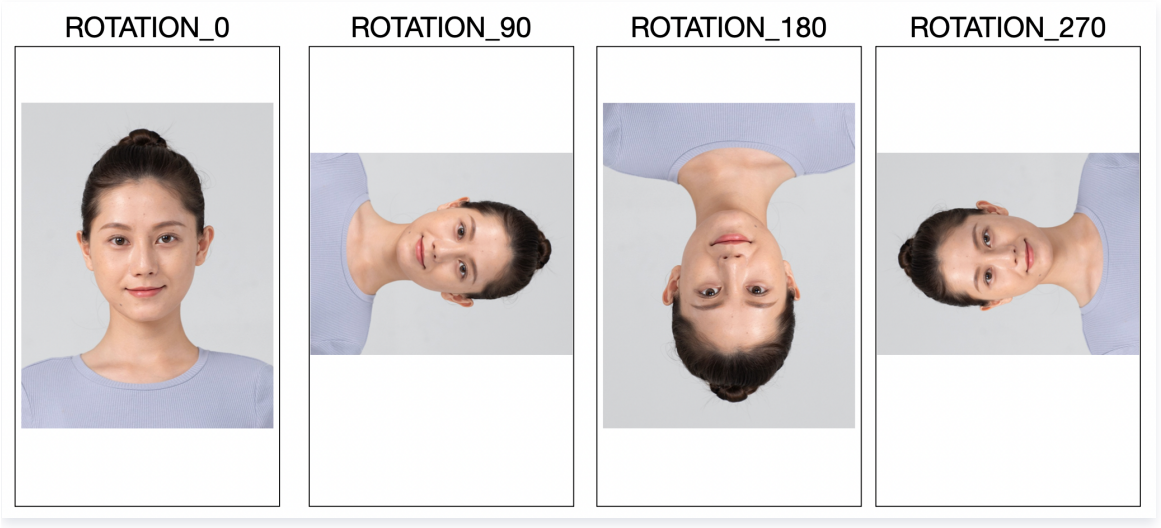
清理GL线程资源，需要在 GL 线程内调用。示例代码：

```
//示例代码见 TECameraBaseActivity.java
public void onGLContextDestroy() {
    if (this.mXMagicApi != null) {
        this.mXMagicApi.onDestroy();
        this.mXMagicApi = null;
    }
}
```

setImageOrientation

```
void setImageOrientation(TEImageOrientation orientation)

public enum TEImageOrientation {
    ROTATION_0,
    ROTATION_90,
    ROTATION_180,
    ROTATION_270
}
```

设置图像方向，使 AI 对不同朝向的人脸都能识别到，如果设置，则忽略 `sensorChanged` 传入的方向。方向示例如下：



如果您传给 SDK 的图像人物始终是正的（Rotation = 0），则无需调用此接口。

sensorChanged

```
void sensorChanged(android.hardware.SensorEvent event, android.hardware.Sensor accelerometer)
```

利用系统传感器判断当前手机旋转的角度，使 AI 对不同朝向的人脸都能识别到。

> ⚠ **注意：**
> 如果通过 `setImageOrientation` 设置了固定的方向，则会忽略 `sensorChagned` 传入的方向。

用法示例：

```
public class MyActivity implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mAccelerometer;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (mXMagicApi != null) {
            mXMagicApi.sensorChanged(event, mAccelerometer);
        }
    }
}
```

isDeviceSupport

```
boolean isDeviceSupport(String motionResPath)
```

将一个动效素材的路径传给 SDK，检测当前设备是否完全支持这个动效。
例如一个动效里有口红、面部贴纸、背景分割，如果当前设备不支持背景分割，则此接口会返回 false。false 并不意味着该动效完全无法使用，而是部分效果无法呈现出来。

isSupportBeauty

判断当前机型是否支持美颜（OpenGL3.0），目前市场上绝大多数手机都是支持 OpenGL3.0的，所以通常无需使用此接口。

```
boolean isSupportBeauty()
```

exportCurrentTexture

获取当前纹理上的画面

```
void exportCurrentTexture(ExportTextureCallback callback)

public interface ExportTextureCallback {
    void onCallback(Bitmap bitmap);
```

```
}
```

setTipsListener

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。例如某些素材会提示用户点点头、伸出手掌、比心等。
注意：回调方法不一定在主线程

```
void setTipsListener(XmagicApi.XmagicTipsListener effectTipsListener)
```

XmagicTipsListener 包含如下方法：

```
public interface XmagicTipsListener {

    /**
     * 显示tips。
     * @param tips 返回的提示语文本信息。
     * @param tipsIcon tips的icon的文件路径，可根据文件路径解析出对应图片，pag文件的话需要使用pagView来展示。
     *                 注意：tipsIcon有可能为空，素材中没有配置就返回空值
     * @param type tips类别，0表示tips和tipsIcon 都有值，1表示是pag素材只有tipsIcon有值。
     * @param duration tips显示时长，毫秒
     */
    void tipsNeedShow(String tips, String tipsIcon, int type, int duration);

    /***
     * 隐藏tips。
     * @param tips 返回的提示语文本信息。
     * @param tipsIcon tips的icon的文件路径，可根据文件路径解析出对应图片，pag文件的话需要使用pagView来展示。
     *                注意：tipsIcon有可能为空，素材中没有配置就返回空值
     * @param type tips类别，0表示tips和tipsIcon 都有值，1表示是pag素材只有tipsIcon有值。
     */
    void tipsNeedHide(String tips, String tipsIcon, int type);
}
```

示例代码如下：

```
public void tipsNeedShow(final String tips, String tipsIcon, int type, int duration) {
    final int tipsType = type;
    final String tipsIconPath = tipsIcon;
    mMainThreadHandler.post(new Runnable() {
        @Override
        public void run() {
            if (tipsType == 0) {
                if (!TextUtils.isEmpty(tipsIconPath) && !mImageTipsIsShow) {
                    mTipsImageView.setVisibility(View.VISIBLE);
                    Bitmap bitmap = BitmapUtils.decodeSampleBitmap(AEModule.getContext(), tipsIconPath,
Integer.MAX_VALUE, Integer.MAX_VALUE);
                    mTipsImageView.setImageBitmap(bitmap);
                    mImageTipsIsShow = true;
                } else {
                    mTipsImageView.setVisibility(View.GONE);
                }
                mTipsTextView.setText(tips);
            } else {
                pagFile = PAGFile.Load(tipsIconPath);
                tipsPAGView.post(new Runnable() {@
                    Override
                    public void run() {
                        if (pagFile != null) {
                            tipsPAGView.setRepeatCount(-1);
```

```
                            tipsPAGView.setComposition(pagFile);
                            tipsPAGView.setProgress(0);
                            tipsPAGView.play();
                            tipsPAGView.setVisibility(View.VISIBLE);
                        }
                    }
                });
            }
        }
    });
}
public void tipsNeedHide(String tips, String tipsIcon, int type) {
    final int tipsType = type;
    final String tipsIconPath = tipsIcon;
    mMainThreadHandler.post(new Runnable() {
        @Override
        public void run() {
            if (tipsType == 0) {
                mTipsContainer.setVisibility(View.GONE);
                mImageTipsIsShow = false;
            } else {
                tipsPAGView.post(new Runnable() {
                    @Override
                    public void run() {
                        tipsPAGView.stop();
                        tipsPAGView.setVisibility(View.GONE);
                    }
                });
            }
        }
    });
}
```

已废弃接口汇总

updateProperty，updateProperties

已废弃，请使用 setEffect 接口。
更改某一项或批量更改多项美颜、动效、滤镜，可在任意线程调用。

```
void updateProperty(XmagicProperty<?> p)
void updateProperties(List<XmagicProperty<?>> properties)
```

参数	含义

XmagicProperty<?> p	美颜特效数据实体类。
	• 以"磨皮"为例，可以按如下方式 new 一个实例：
	`new XmagicProperty<>(Category.BEAUTY, null, null, BeautyConstant.BEAUTY_SMOOTH, new XmagicPropertyValues(0, 100, 50, 0, 1));`
	• 以“2D 动效兔兔酱”为例，可以按如下方式 new 一个实例：
	`new XmagicProperty<>(Category.MOTION, "video_tutujiang" , "动效的文件路径", null, null);`
	如果想要某个动效/美妆/分割素材叠加在当前素材上，则将该素材 XmagicProperty 对象的 mergeWithCurrentMotion 设置为 true。关于素材叠加的详细说明见 素材叠加 。
	更多的例子，请参考 Demo 工程 assets/beauty_panel 文件夹下的配置信息。

XmagicProperty

已废弃。请使用 setEffect 接口设置美颜特效。

- 美颜

属性字段	说明
category	Category.BEAUTY
ID	null 特殊情况： • 瘦脸中的（自然、女神、英俊）ID 值分别为： `BeautyConstant.BEAUTY_FACE_NATURE_ID`、 `BeautyConstant.BEAUTY_FACE_FEMALE_GOD_ID`、 `BeautyConstant.BEAUTY_FACE_MALE_GOD_ID` • 口红中的 ID 值为：`XmagicConstant.BeautyConstant.BEAUTY_LIPS_LIPS_MASK` • 腮红中的 ID 值为：`XmagicConstant.BeautyConstant.BEAUTY_MAKEUP_MULTIPLY_MULTIPLY_MASK` • 立体中的 ID 值为：`XmagicConstant.BeautyConstant.BEAUTY_SOFTLIGHT_SOFTLIGHT_MASK`
resPath	null 特殊情况：口红、腮红、立体的 resPath 是资源图片的路径，具体请参考 Demo 中 assets/beauty_panel/advanced_beauty.json 文件。
effkey	必填，参见 Demo 示例：美白 BeautyConstant.BEAUTY_WHITEN
effValue	必填，参见 Demo 中 assets/beauty_panel/advanced_beauty.json 文件，demo 工程中解析该文件构造一个 XmagicPropertyValues 对象，XmagicPropertyValues 各个属性取值见 美颜参数说明

- 美体

属性字段	说明
category	Category.BODY_BEAUTY
ID	null
resPath	null
effkey	必填，参见 Demo 示例：长腿 BeautyConstant.BODY_LEG_STRETCH
effValue	必填，参见 Demo 中 assets/beauty_panel/beauty_body.json 文件。demo 工程中解析该文件构造一个 XmagicPropertyValues 对象，XmagicPropertyValues 各个属性取值见 美颜参数说明

- 滤镜

属性字段	说明
category	Category.LUT
ID	图片名称，必填 示例：dongjing_lf.png “无” ID 为 XmagicProperty.ID_NONE
resPath	滤镜图片路径，必填，“无”设置为 null
effkey	null
effValue	必填，“无”设置为 null。是一个 XmagicPropertyValues 对象，XmagicPropertyValues 各个属性取值见 美颜参数说明

- 动效

属性字段	说明
category	Category.MOTION

ID	资源文件夹名称，必填 示例：video_lianliancaomei “无” ID 为 XmagicProperty.ID_NONE
resPath	必填，参见 Demo
effkey	null
effValue	null

- 美妆

属性字段	说明
category	Category.MAKEUP
ID	资源文件夹名称，必填 示例：video_xuejiezhuang “无” ID 为 XmagicProperty.ID_NONE
resPath	必填，参见 Demo
effkey	必填，取值为：makeup.strength“无”设置为 null
effValue	必填，“无”设置为 null。是一个 XmagicPropertyValues 对象，XmagicPropertyValues 各个属性取值见 美颜参数说明

- 分割

属性字段	说明
category	Category.SEGMENTATION
ID	资源文件夹名称，必填 示例：video_segmentation_blur_45 “无” ID 为 XmagicProperty.ID_NONE 自定义分割 ID 值必须使用：`XmagicConstant.SegmentationId.CUSTOM_SEG_ID`
resPath	必填，参见 Demo
effkey	null（自定义背景除外），自定义背景的值为选择的资源路径
effValue	null

setYTDataListener

已废弃接口，请使用 setAIDataListener 的 onAIDataUpdated 回调。

onPauseAudio

当仅需要停止音频，但不需要释放 GL 线程时调用此函数。

getPropertyRequiredAbilities

传入一个动效资源列表，返回每一个资源所使用到的 SDK 原子能力列表。
方法的使用场景为：
您购买或制作了若干款动效素材，调用这个方法，会返回每一个素材需要使用的原子能力列表。例如素材1需要使用能力 A、B、C，素材2需要使用能力 B、C、D，然后您把这样的能力列表保持在服务器上。之后，当用户要从服务器下载动效素材时，用户先通过 getDeviceAbilities 方法获取他手机具备的原子能力列表（例如这台手机具备能力 A、B、C，但不具备能力 D），把他的能力列表传给服务器，服务器判断该设备不具备能力 D，因此不给该用户下发素材2。

参数	含义
List<XmagicProperty<?>> assets	需要检测原子能力的动效资源列表。

返回值 `Map<XmagicProperty<?>,ArrayList<String>>` ：

- key：动效资源素材实体类。

- value：所使用到的原子能力列表。

getDeviceAbilities

返回当前设备支持的原子能力表。与 getPropertyRequiredAbilities 方法搭配使用，详见 getPropertyRequiredAbilities 的说明。

```
Map<String,Boolean> getDeviceAbilities()
```

返回值 `Map<String,Boolean>`：

- key：原子能力名（与素材能力名字对应）。
- value：当前设备是否支持。

isBeautyAuthorized

判断当前的 License 授权支持哪些美颜或美体项。仅支持 BEAUTY 和 BODY_BEAUTY 类型的美颜项检测。检测后的结果会赋值到各个美颜对象 `XmagicProperty.isAuth` 字段中。如果 isAuth 字段为 false，可以在 UI 上屏蔽这些项的入口。

```
void isBeautyAuthorized(List<XmagicProperty<?>> properties)
```

enableEnhancedMode

开启美颜增强模式。默认未开启。具体请参见 增强模式使用指南 。

setDowngradePerformance

请使用 enableHightPerformance 接口开启高性能模式。

Flutter

最近更新时间：2025-02-28 12:24:14

美颜特效 SDK Flutter 版本核心接口类 `TencentEffectApi`，更新美颜数值、调用动效等功能。

Public 成员函数

API	描述
setResourcePath	设置美颜资源的本地存储路径（V0.3.5.0版本新增）
initXmagic	初始化美颜数据，使用美颜前必须先调用此方法(在V0.3.1.1版本及之前)
setLicense	进行美颜授权
setXmagicLogLevel	设置 SDK 的 log 等级，建议开发调试时设为 Log.DEBUG，正式发布时设置为 Log.WARN，如果正式发布设置为 Log.DEBUG，大量的日志会影响性能
onResume	恢复渲染，页面可见时调用
onPause	暂停渲染，页面不可见时调用
enableEnhancedMode	开启美颜增强模式，具体请参考 增强模式使用指南
setDowngradePerformance	调用此方法开启高性能模式。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。**注意：需要在调用其他方法之前调用**
setAudioMute	设置静音（因为有些贴纸中有声音）
setFeatureEnableDisable	开启或关闭某个特性
@Deprecated updateProperty	更新美颜属性， 可在任意线程调用（**此接口已废弃**）
setEffect	更新美颜属性（V0.3.5.0版本新增）
setOnCreateXmagicApiErrorListener	设置创建美颜对象时的回调接口（如果出错会回调此接口）
setTipsListener	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
setYTDataListener	设置人脸点位信息等数据回调（需要获得人脸点位的 Licence 授权（例如原子能力X102）才会有回调）
setAIDataListener	设置人脸、手势、身体检测状态回调
@Deprecated isBeautyAuthorized	判断当前的 lic 授权支持哪些美颜。 仅支持 BEAUTY 和 BODY_BEAUTY 类型的美颜项检测。检测后的结果会赋值到各个美颜对象 XmagicProperty.isAuth 字段中（**此接口已废弃**）
isSupportBeauty	判断当前机型是否支持美颜（OpenGL3.0）
getDeviceAbilities	返回当前设备支持的原子能力表
@Deprecated isDeviceSupport	将动效资源列表传入 SDK 中做检测，执行后 XmagicProperty.isSupport 字段标识该原子能力是否可用。 根据 XmagicProperty.isSupport 可 UI 层控制单击限制，或者直接从资源列表删除（**此接口已废弃**）
isDeviceSupportMotion	检测当前设备是否支持此素材
@Deprecated getPropertyRequiredAbilities	传入一个动效资源列表，返回每一个资源所使用到的 SDK 原子能力列表（**此接口已废弃**）

成员函数说明

setResourcePath（V0.3.5.0版本新增）

设置美颜资源存放的本地路径

```
///设置美颜资源存放的本地路径，使用美颜前必须先调用此方法。
///v0.3.5.0新增
void setResourcePath(String xmagicResDir);
```

参数

参数	含义
String xmagicResDir	资源文件放置的目录

initXmagic

初始化美颜数据。在V0.3.1.1版本及之前，使用美颜前必须先调用此方法。在V0.3.5.0版本，此方法每个版本只需要调用一次，并且在调用了此方法之前必须先调用 `setResourcePath` 方法设置了资源路径，在V0.3.5.0版本中删除了之前的 xmagicResDir 参数，可参考最新 demo。

V0.3.5.0版本

```
void initXmagic(InitXmagicCallBack callBack);

typedef InitXmagicCallBack = void Function(bool reslut);
```

V0.3.1.1版本及之前

```
void initXmagic(String xmagicResDir,InitXmagicCallBack callBack);

typedef InitXmagicCallBack = void Function(bool reslut);
```

参数

参数	含义
String xmagicResDir	资源文件放置的目录
InitXmagicCallBack callBack	初始化回调接口

setLicense

设置鉴权数据，进行美颜授权。

```
  ///美颜进行鉴权处理
void setLicense(String licenseKey, String licenseUrl, LicenseCheckListener checkListener);
///授权校验的结果回调方法
typedef LicenseCheckListener = void Function(int errorCode, String msg);
```

参数

参数	含义
String licenseKey	鉴权的 LicenseKey
String licenseUrl	鉴权的 LicenseUrl
LicenseCheckListener checkListener	授权结果回调接口

setXmagicLogLevel

设置 SDK 的 log 等级

```
void setXmagicLogLevel(int logLevel);
```

参数

参数	含义
int logLevel	可使用 LogLevel 定义好的类型进行设置

onResume

恢复美颜处理

```
void onResume();
```

onPause

暂停美颜处理

```
void onPause();
```

enableEnhancedMode

开启增强模式，具体请参考 增强模式使用指南 。

```
void enableEnhancedMode();
```

setDowngradePerformance（V0.3.1.1新增）

开启性能模式

```
void setDowngradePerformance();
```

setAudioMute（V0.3.1.1新增）

设置是否静音。参数：true 表示静音，false 表示非静音。

```
///背景音乐是否静音
void setAudioMute(bool isMute);
```

setFeatureEnableDisable（V0.3.1.1新增）

开启或关闭某个能力

```
/// 开启或关闭某个能力
void setFeatureEnableDisable(String featureName, bool enable);
```

参数

参数	含义
String featureName	原子能力名称 取值如下： • "ai.3dmmV2.enable" 人脸表情能力 • "ai.body3dpoint.enable" 身体点位能力 • "ai.hand.enable" 手势检测能力 • "beauty.onlyWhitenSkin" 美白仅对皮肤生效 • "ai.segmentation.skin.enable" 皮肤分割能力 • "auto_beauty_switch" 智能美颜(为男性、宝宝减淡美颜美妆效果)

boolean enable	true 表示开启此能力，false 表示关闭此能力 **注：如果是降级模式，则不允许开启皮肤分割**

~~updateProperty~~

设置某一项美颜数值或者动效、滤镜，可在任意线程调用。

```
void updateProperty(XmagicProperty xmagicProperty);
```

参数

参数	含义
XmagicProperty xmagicProperty	美颜属性封装对象

setEffect（V0.3.5.0新增）

设置美颜、美型、滤镜、美妆、贴纸、分割等效果，可在任意线程调用。具体参数请参考 美颜参数说明 。

```
///更新美颜属性。
void setEffect(String effectName,int effectValue,String? resourcePath,Map<String,String>? extraInfo);
```

setOnCreateXmagicApiErrorListener

设置美颜对象创建时的错误回调接口

```
 void setOnCreateXmagicApiErrorListener(OnCreateXmagicApiErrorListener? errorListener);
///创建美颜实例时的错误回调方法
typedef OnCreateXmagicApiErrorListener = void Function(String errorMsg, int code);
```

参数

参数	含义
OnCreateXmagicApiErrorListener? errorListener	创建美颜对象时错误信息回调接口

返回错误码含义对照表：

错误码	含义
-1	未知错误
-100	3D 引擎资源初始化失败
-200	不支持 GAN 素材
-300	设备不支持此素材组件
-400	模板 JSON 内容为空
-500	SDK 版本过低
-600	不支持分割
-700	不支持 OpenGL
-800	不支持脚本
5000	分割背景图片分辨率超过 2160×3840
5001	分割背景图片所需内存不足

5002	分割背景视频解析失败。
5003	分割背景视频超过200秒
5004	分割背景视频格式不支持

setTipsListener

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。比如某些素材会提示用户点点头、伸出手掌、比心等。

```
void setTipsListener(XmagicTipsListener? xmagicTipsListener);

abstract class XmagicTipsListener {
  /// 显示tips。Show the tip.
  /// @param tips tips字符串。Tip's content
  /// @param tipsIcon tips的icon。Tip's icon
  /// @param type tips类别，0表示字符串和icon都展示，1表示是pag素材只展示icon。tips category, 0 means that both
strings and icons are displayed, 1 means that only the icon is displayed for the pag material
  /// @param duration tips显示时长，毫秒。Tips display duration, milliseconds
  void tipsNeedShow(String tips, String tipsIcon, int type, int duration);

  /// *
  /// 隐藏tips。Hide the tip.
  /// @param tips tips字符串。Tip's content
  /// @param tipsIcon tips的icon。Tip's icon
  /// @param type tips类别，0表示字符串和icon都展示，1表示是pag素材只展示icon。tips category, 0 means that both
strings and icons are displayed, 1 means that only the icon is displayed for the pag material
  void tipsNeedHide(String tips, String tipsIcon, int type);
}
```

参数

参数	含义
XmagicTipsListener xmagicTipsListener	回调函数实现类

setYTDataListener

设置人脸点位信息等数据回调。

```
  ///设置人脸点位信息等数据回调,需要获得人脸点位的Licence授权（例如原子能力X102）才会有回调.
void setYTDataListener(XmagicYTDataListener? xmagicYTDataListener);
设置人脸信息等数据回调

abstract class XmagicYTDataListener {
  //优图AI数据回调。
  void onYTDataUpdate(String data);
}
```

onYTDataUpdate 返回 JSON string 结构，最多返回5个人脸信息：

```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
    180.0,
    112.2,
    ...
  ],
```

```
  "face_256_visible":[
    0.85,
    ...
  ],
  "out_of_screen":true,
  "left_eye_high_vis_ratio:1.0,
  "right_eye_high_vis_ratio":1.0,
  "left_eyebrow_high_vis_ratio":1.0,
  "right_eyebrow_high_vis_ratio":1.0,
  "mouth_high_vis_ratio":1.0
 },
 ...
 ]
}
```

字段含义

字段	类型	值域	说明
trace_id	int	[1,INF)	人脸 ID，连续取流过程中，ID 相同的可以认为是同一张人脸
face_256_point	float	[0,screenWidth] 或 [0,screenHeight]	共512个数，人脸256个关键点，屏幕左上角为(0,0)
face_256_visible	float	[0,1]	人脸256关键点可见度
out_of_screen	bool	true/false	人脸是否出框
left_eye_high_vis_ratio	float	[0,1]	左眼高可见度点位占比
right_eye_high_vis_ratio	float	[0,1]	右眼高可见度点位占比
left_eyebrow_high_vis_ratio	float	[0,1]	左眉高可见度点位占比
right_eyebrow_high_vis_ratio	float	[0,1]	右眉高可见度点位占比
mouth_high_vis_ratio	float	[0,1]	嘴高可见度点位占比

参数

参数	含义
XmagicYTDataListener xmagicYTDataListener	回调函数实现类

setAIDataListener

检测到人脸、身体、手势时，会回调这些部位的点位信息。

```
void setAIDataListener(XmagicAIDataListener? aiDataListener);

abstract class XmagicAIDataListener {
  void onFaceDataUpdated(String faceDataList);

  void onHandDataUpdated(String handDataList);

  void onBodyDataUpdated(String bodyDataList);
}
```

~~isBeautyAuthorized~~

判断当前的 License 授权支持哪些美颜或美体项。仅支持 BEAUTY 和 BODY_BEAUTY 类型的美颜项检测。检测后的结果会赋值到各个美颜对象 `XmagicProperty.isAuth` 字段中。如果 isAuth 字段为 false，可以在 UI 上屏蔽这些项的入口。

```
Future<List<XmagicProperty>> isBeautyAuthorized(
      List<XmagicProperty> properties);
```

参数

参数	含义
List<XmagicProperty> properties	需要检测的美颜项

isSupportBeauty

判断当前机型是否支持美颜（OpenGL3.0）。

```
  Future<bool> isSupportBeauty();
```

返回

返回值 bool：是否支持支持美颜。

getDeviceAbilities

返回当前设备支持的原子能力表。与 getPropertyRequiredAbilities 方法搭配使用.

```
  Future<Map<String, bool>> getDeviceAbilities();
```

返回

返回值 `Map<String, bool>`：

- key：原子能力名（与素材能力名字对应）。
- value：当前设备是否支持。

~~isDeviceSupport~~

将动效资源列表传入 SDK 中做检测，执行后 `XmagicProperty.isSupport` 字段标识该素材是否可用。根据 `XmagicProperty.isSupport` 可 UI 层控制单击限制，或者直接从资源列表删除。

```
 Future<List<XmagicProperty>> isDeviceSupport(List<XmagicProperty> assetsList);
```

参数

参数	含义
List<XmagicProperty> assetsList	需要检测的动效素材列表

isDeviceSupportMotion (V0.3.5.0版本新增)

检测当前设备是否支持此素材

```
Future<bool> isDeviceSupportMotion(String motionResPath);
```

参数

参数	含义

motionResPath	需要检测的动效素材的本地文件地址

getPropertyRequiredAbilities

传入一个动效资源列表，返回每一个资源所使用到的 SDK 原子能力列表。
方法的使用场景为：
您购买或制作了若干款动效素材，调用这个方法，会返回每一个素材需要使用的原子能力列表。例如素材1需要使用能力 A、B、C，素材2需要使用能力 B、C、D，然后您把这样的能力列表保持在服务器上。之后，当用户要从服务器下载动效素材时，用户先通过 getDeviceAbilities 方法获取他手机具备的原子能力列表（比如这台手机具备能力 A、B、C，但不具备能力 D），把他的能力列表传给服务器，服务器判断该设备不具备能力 D，因此不给该用户下发素材2。

```
Future<Map<XmagicProperty, List<String>?>> getPropertyRequiredAbilities(
    List<XmagicProperty> assetsList);
```

参数

参数	含义
List<XmagicProperty> assetsList	需要检测原子能力的动效资源列表

返回

返回值 Map<XmagicProperty, List<String>?>：

- key：动效资源素材实体类。
- value：所使用到的原子能力列表。

Web

最近更新时间：2024-07-18 17:32:01

本文档将介绍 Web 美颜特效 SDK 核心参数及方法。

> **说明：**
> Web美颜特效 SDK 需要浏览器支持并开启硬件加速才能够流畅渲染（小程序端无需判断），因此 SDK 提供了检测方法供业务提前判断，如不支持则进行屏蔽处理。

```
import {ArSdk, isWebGLSupported} from 'tencentcloud-webar'

if(isWebGLSupported()) {
    const sdk = new ArSdk({
    ...
})
} else {
    // 屏蔽逻辑
}
```

初始化参数

```
import { ArSdk } from 'tencentcloud-webar'
// 初始化SDK
const sdk = new ArSdk({
...
})
```

初始化 SDK 的 Config 支持以下参数：

参数	说明	类型	是否必传
module	模块配置	`type ModuleConfig = { beautify: boolean // 默认为true segmentation: boolean // 默认为false }`	否，默认为 `{beautify: true, segmentation: false}`
auth	鉴权参数	`type AuthConfig = { licenseKey: string // 控制台 Web License 管理 获取 appId: string // 控制台 账号信息 > 基本信息 查看 APPID authFunc:() => Promise<{ signature:string, timestamp:string }> // 请参见 License 配置使用 }`	是
input	输入源	MediaStream|HTMLImageElement|String	否
camera	内置相机	`type CameraConfig = { width: number, // 拍摄画面宽度`	否

		height: number, // 拍摄画面高度 mirror: boolean, // 是否开启左右镜像 frameRate: number // 画面采集帧率 }	

beautify	美颜参数	type BeautifyOptions = {
		whiten?: number, // 美白 0-1
		dermabrasion?: number // 磨皮0-1
		lift?: number // 瘦脸0-1
		shave?: number // 削脸0-1
		eye?: number // 大眼0-1
		chin?: number // 下巴0-1
		}
			否

background	背景参数	type BackgroundOptions = {
		type: 'image' | 'blur' | 'transparent',
		src?: string
		}
			否

loading	内置 loading icon 配置	type loadingConfig = {
		enable: boolean,
		size?: number
		lineWidth?: number
		strokeColor?: number
		}	

回调事件

```
let effectList = [];
let filterList = [];
// sdk 的回调用法
sdk.on('created', () => {
    // 在 created 回调中拉取特效和滤镜列表供页面展示
    sdk.getEffectList({
        Type: 'Preset',
        Label: '美妆',
    }).then(res => {
        effectList = res
    });
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})
sdk.on('cameraReady', async () => {
    // 在 cameraReady 回调中可以更早地获取输出画面，此时初始化传入的美颜参数还未生效
    // 适用于需要尽早地展示画面，但不要求画面一展示就有美颜的场景
    // 后续美颜生效后无需更新stream，SDK内部已处理
    const arStream = await ar.getOutput();
    // 本地播放
    // localVideo.srcObject = arStream

})
```

```
sdk.on('ready', () => {
    // 在 ready 回调中获取输出画面，此时初始化传入的美颜参数已生效
    // 区别上述cameraReady中获取output，适用于画面一展示就要有美颜的场景，但不要求尽早地展示画面
    // 根据自身业务需求选择一种处理方式即可
    const arStream = await ar.getOutput();
    // 本地播放
    // localVideo.srcObject = arStream

    // 在 ready 回调中调用 setBeautify/setEffect/setFilter 等渲染方法
    sdk.setBeautify({
        whiten: 0.3
    });
    sdk.setEffect({
        id: effectList[0].EffectId,
        intensity: 0.7
    });
    sdk.setEffect({
        id: effectList[0].EffectId,
        intensity: 0.7,
        filterIntensity: 0.5 // 0.1.18及以上版本支持单独设置effect中滤镜的强度，不传则默认与特效的intensity保持一致
    });
    sdk.setFilter(filterList[0].EffectId, 0.5)
})
```

事件	说明	回调参数
created	SDK 鉴权完成并成功创建实例时触发	-
cameraReady	SDK 的画面生成时触发，此时 output 已有画面但美颜仍无法生效	-
ready	SDK 内部检测初始化完成时触发，此时 output 画面已有美颜，也可以设置新的特效	-
error	SDK 发生错误时触发	error 对象

对象方法

接口	参数	返回	说明
async getOutput(fps)	fps：设置输出的媒体流帧率，默认无须设置	MediaStream|String	仅 Web 端提供，小程序暂不支持
setBeautify(options)	`type BeautifyOptions = { whiten?: number, // 美白 0-1 dermabrasion?: number // 磨皮0-1 lift?: number // 瘦脸 0-1 shave?: number // 削脸 0-1 eye?: number // 大眼0-1 chin?: number // 下巴 0-1 }`	-	设置美颜参数，需开启美颜模块
setEffect(effects,	• effects：特效 ID | effect 对象 | 特效 ID / effect 数组	-	设置特效，需开启美颜模块

callback)	effect:{
	id: string,
	intensity: number, // 特效强度，默认1，范围0-1
	filterIntensity: number // 单独控制特效中的滤镜强度，默认取intensity，范围0-1 (0.1.18及以上版本支持)
	}
	• callback：设置成功的回调
			3D类特效仅高级版License支持

setAvatar(params)	{
	mode: 'AR' | 'VR',
	effectId?: string, // 透传effectId使用内置模型
	url?: string, // 透传url使用自定义模型
	backgroundUrl?: string, // 背景图片链接，仅在VR模式下生效
	}
		–	设置 Animoji 表情或虚拟形象
			仅高级版License支持

setBackground(options)	{
	type: 'image|blur|transparent',
	src: string // 仅image类型需要
	}
		–	设置背景，需开启人像分割模块

setFilter(id, intensity, callback)	• id：滤镜 ID • intensity：滤镜强度，范围0-1 • callback：设置成功回调	–	设置滤镜

getEffectList(params)	{
	PageNumber: number,
	PageSize: number,
	Name: '',
	Label: Array,
	Type: 'Custom|Preset'
	}
		特效列表	拉取特效列表

getAvatarList(type)	type = 'AR' | 'VR'	虚拟形象列表	拉取虚拟形象列表
getEffect(effectId)	effectId：特效 ID	单个特效信息	拉取指定特效的信息

getCommonFilter()	-	内置滤镜列表	获取内置滤镜列表
async updateInputStream(src:MediaStream)（0.1.19版本后支持）	src：新的输入流MediaStream	-	更新输入流
disable()	-	-	停用面部检测，返回未处理的原始画面，可以降低 CPU 使用率
enable()	-	-	恢复面部检测，返回美颜等特效生效的画面
async startRecord()	-	-	开始录像（仅小程序端支持）
async stopRecord()	`{ useOriginAudio: boolean, // 是否录制视频原声 musicPath: string, // 背景音乐地址，useOriginAudio为false时生效 }`	录像	结束录像并返回录像结果（仅小程序端支持）
async takePhoto()	-	`{ data: Uint8Array, width: number, height: number }`	拍照，返回一个包含 buffer 数据的对象（仅小程序端支持）
destroy()	-	-	销毁当前 SDK 实例以及相关的纹理资源

错误处理

在 error 回调返回的对象中包含错误码与错误信息以方便进行错误处理。

```
sdk.on('error', (error) => {
    // 在 error 回调中处理错误
    const {code, message} = error
    ...
})
```

错误码	含义	备注
10000001	当前浏览器环境不兼容	建议用户使用 Chrome、Firefox、Safari、微信浏览器访问
10000002	当前渲染上下文丢失	-
10000003	渲染耗时长	考虑降低视频分辨率或屏蔽功能

10000005	输入源解析错误	-
10000006	浏览器特性支持不足，可能会出现卡顿情况	建议用户使用 Chrome、Firefox、Safari、微信浏览器访问
10001101	设置特效出错	-
10001102	设置滤镜出错	-
10001103	特效强度参数不正确	-
10001201	调起用户摄像头失败	-
10001202	摄像头中断	-
10001203	没有获取到摄像头权限	需要开启摄像头权限，设置-隐私-相机开启
20002001	缺少鉴权参数	-
20001001	鉴权失败	请确认是否创建 License，请确认签名是否正确
20001002	接口请求失败	回调会回传接口返回的数据，具体信息请参见 接口错误码
20001003	设置特效接口鉴权失败	无权访问的接口，基础版 License 无法使用高级版 License 功能
30000001	小程序 startRecord 失败	-
30000002	小程序 stopRecord 失败	-
40000000	未捕获的异常	-
40000001	当前使用 SDK 版本过低，部分特效无法正确展示，请升级 SDK 版本	-
50000002	分辨率改变导致特效丢失	需要重新设置特效

处理当前渲染上下文丢失

部分 PC 在长期切后台的场景可能触发处理 contextlost 错误，可以调用 `ArSdk.prototype.resetCore(input: MediaStream)` 恢复渲染上下文。

```
sdk.on('error', async (error) => {
    if (error.code === 10000002) {
        const newInput = await navigator.mediaDevices.getUserMedia({...})
        await sdk.resetCore(newInput)
    }
})
```

macOS

最近更新时间：2025-02-28 12:24:14

美颜特效 SDK 核心接口类 `XMagic.h`，用于初始化 SDK、更新美颜数值、调用动效等功能。

Public 成员函数

API	描述
initWithRenderSize	初始化接口
configPropertyWithType	配置美颜各种效果
setRenderSize	设置 renderSize
deinit	资源释放接口
process	处理数据接口
getConfigPropertyWithName	获取美颜参数配置信息
registerLoggerListener	日志注册接口
registerSDKEventListener	SDK 事件监听接口
clearListeners	注册回调清理接口
getCurrentGlContext	获取当前 GL 上下文接口
onPause	SDK 暂停接口
onResume	SDK 恢复接口
setAudioMute	动效素材使用时是否开启静音（V2.5.0新增） 参数：YES表示静音，NO表示非静音

initWithRenderSize

初始化接口

```
- (instancetype _Nonnull)initWithRenderSize:(CGSize)renderSize
                        assetsDict:(NSDictionary* _Nullable)assetsDict;
```

参数

参数	含义
renderSize	渲染尺寸
assetsDict	资源 Dict

configPropertyWithType

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;
```

参数

参数	含义
propertyType	效果类型
propertyName	效果名称
propertyValue	效果数值
extraInfo	预留扩展，附加额外配置 Dict

配置美颜效果示例

- **美颜：**配置美白效果

```
NSString *propertyType = @"beauty";         //配置美颜的效果类型，这里以美颜为例
NSString *propertyName = @"beauty.whiten"; //配置美颜的名称，这里以美白为例
NSString *propertyValue = @"60";            //配置美白的效果数值
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:nil];
```

- **滤镜：**配置心动效果

```
NSString *propertyType = @"lut";         //配置美颜的效果类型，这里以滤镜为例
NSString *propertyName = [@"lut.bundle/" stringByAppendingPathComponent:@"xindong_lf.png"]; //配置美颜的
名称，这里以心动为例
NSString *propertyValue = @"60";            //配置滤镜的效果数值
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:nil];
```

- **美体：**配置长腿效果

```
NSString *propertyType = @"body";         //配置美颜的效果类型，这里以美体为例
NSString *propertyName = @"body.legStretch"; //配置美颜的名称，这里以长腿为例
NSString *propertyValue = @"60";            //配置长腿的效果数值
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:nil];
```

- **动效：**配置2D动效的可爱涂鸦效果

```
 NSString *motion2dResPath = [[NSBundle mainBundle] pathForResource:@"2dMotionRes" ofType:@"bundle"];//
这里是2dMotionRes文件夹的绝对路径
 NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以动效为例
 NSString *propertyName = @"video_keaituya"; //配置美颜的名称，这里以2D动效的可爱涂鸦为例
 NSString *propertyValue = motion2dResPath;  //配置动效的路径
 [self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:nil];
```

- **美妆：**配置女团妆效果

```
NSString *motionMakeupResPath = [[NSBundle mainBundle] pathForResource:@"makeupMotionRes"
ofType:@"bundle"];//这里是makeupMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以美妆为例
NSString *propertyName = @"video_nvtuanzhuang"; //配置美颜的名称，这里以女团妆为例
NSString *propertyValue = motionMakeupResPath;  //配置动效的路径
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:nil];
//下面是要配置美妆的数值（上面的动效只需要调用一次，下面的配置美妆数值可以多次调用）
 NSString *propertyTypeMakeup = @"custom";          //配置美颜的效果类型，这里以美妆为例
 NSString *propertyNameMakeup = @"makeup.strength"; //配置美颜的名称，这里以女团妆为例
```

```
 NSString *propertyValueMakeup = @"60";              //配置美妆的效果数值
 [self.xmagicApi configPropertyWithType:propertyTypeMakeup withName:propertyNameMakeup
withData:propertyValueMakeup withExtraInfo:nil];
```

- **分割：配置背景模糊（强效果）**

```
NSString *motionSegResPath = [[NSBundle mainBundle] pathForResource:@"segmentMotionRes"
ofType:@"bundle"];//这里是segmentMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以分割为例
NSString *propertyName = @"video_segmentation_blur_75"; //配置美颜的名称，这里以背景模糊-强为例
NSString *propertyValue = motionSegResPath;  //配置动效的路径
NSDictionary *dic = @{@"bgName":@"BgSegmentation.bg.png", @"bgType":@0, @"timeOffset":
@0},@"icon":@"segmentation.linjian.png"};//配置预留字段
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:dic];
```

- **自定义背景:**

```
NSString *motionSegResPath = [[NSBundle mainBundle] pathForResource:@"segmentMotionRes"
ofType:@"bundle"];//这里是segmentMotionRes文件夹的绝对路径
NSString *propertyType = @"motion";          //配置美颜的效果类型，这里以分割为例
NSString *propertyName = @"video_empty_segmentation"; //配置美颜的名称，这里以自定义背景为例
NSString *propertyValue = motionSegResPath;  //配置动效的路径
NSString *imagePath = @"/var/mobile/Containers/Data/Application/06B00BBC-9060-450F-8D3A-
F6028D185682/Documents/MediaFile/image.png"; //自定义背景图片的绝对路径。如果自定义背景选择的是视频，需要对视频进行
压缩转码处理，使用压缩转码处理后的绝对路径
int bgType = 0;//自定义背景的类型。 0表示图片，1表示视频
int timeOffset = 0; //时长。图片背景时，为0；视频背景时为视频的时长
NSDictionary *dic = @{@"bgName":imagePath, @"bgType":@(bgType), @"timeOffset":
@(timeOffset)},@"icon":@"segmentation.linjian.png"};//配置预留字段
[self.xmagicApi configPropertyWithType:propertyType withName:propertyName withData:propertyValue
withExtraInfo:dic];
```

setRenderSize

设置 renderSize

```
- (void)setRenderSize:(CGSize)size;
```

参数

参数	含义
size	渲染尺寸

deinit

资源释放接口

```
- (void)deinit;
```

process

处理数据接口

```
- (YTProcessOutput* _Nonnull)process:(YTProcessInput * _Nonnull)input;
```

参数

参数	含义
input	输入处理数据信息

getConfigPropertyWithName

获取美颜参数配置信息

```
- (YTBeautyPropertyInfo * _Nullable)getConfigPropertyWithName:(NSString *_Nonnull)propertyName;
```

参数

参数	含义
propertyName	配置名称

registerLoggerListener

日志注册接口

```
- (void)registerLoggerListener:(id<YTSDKLogListener> _Nullable)listener withDefaultLevel:
(YtSDKLoggerLevel)level;
```

参数

参数	含义
listener	日志回调接口
level	日志输出 level，默认 ERROR

registerSDKEventListener

SDK 事件监听接口

```
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;
```

参数

参数	含义
listener	事件监听器回调，主要分为 AI 事件，Tips 提示事件，Asset 事件

clearListeners

注册回调清理接口

```
- (void)clearListeners;
```

getCurrentGlContext

获取当前 GL 上下文接口

```

- (nullable NSOpenGLContext*)getCurrentGlContext;
```

onPause

SDK 暂停接口

```
/// @brief APP暂停时候需要调用SDK暂停接口
- (void)onPause;
```

onResume

SDK 恢复接口

```
/// @brief APP恢复时候需要调用SDK恢复接口
- (void)onResume;
```

setAudioMute

动效素材使用时是否开启静音（V2.5.0新增）

```
/// @brief 设置静音
- (void)setAudioMute:(BOOL)isMute;
```

静态函数

API	描述
isBeautyAuthorized	获取该美颜参数的授权信息

isBeautyAuthorized

获取该美颜参数的授权信息（仅支持美颜和美体）

```
/// @param featureId 配置美颜参数
/// @return 返回对应美颜参数的授权结果
+ (BOOL)isBeautyAuthorized:(NSString * _Nullable)featureId;
```

回调

API	描述
YTSDKEventListener	SDK 内部事件回调接口
YTSDKLogListener	日志监听回调

YTSDKEventListener

SDK 内部事件回调接口

```
@protocol YTSDKEventListener <NSObject>
```

成员函数

返回类型	名称
void	onYTDataEvent
void	onAIEvent
void	onTipsEvent
void	onAssetEvent

函数说明

onYTDataEvent

YTDataUpdate 事件回调

```
/// @param event NSString*格式的回调
- (void)onYTDataEvent:(id _Nonnull)event;
```

返回 JSON string 结构，最多返回5个人脸信息：

```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
    180.0,
    112.2,
    ...
  ],
  "face_256_visible":[
    0.85,
    ...
  ],
  "out_of_screen":true,
  "left_eye_high_vis_ratio:1.0,
  "right_eye_high_vis_ratio":1.0,
  "left_eyebrow_high_vis_ratio":1.0,
  "right_eyebrow_high_vis_ratio":1.0,
  "mouth_high_vis_ratio":1.0
 },
 ...
 ]
}
```

字段含义

字段	类型	值域	说明
trace_id	int	[1,INF)	人脸 ID，连续取流过程中，ID 相同的可以认为是同一张人脸
face_256_point	float	[0,screenWidth] 或 [0,screenHeight]	共512个数，人脸256个关键点，屏幕左上角为(0,0)
face_256_visible	float	[0,1]	人脸256关键点可见度
out_of_screen	bool	true/false	人脸是否出框
left_eye_high_vis_ratio	float	[0,1]	左眼高可见度点位占比
right_eye_high_vis_ratio	float	[0,1]	右眼高可见度点位占比
left_eyebrow_high_vis_ratio	float	[0,1]	左眉高可见度点位占比
right_eyebrow_high_vis_ratio	float	[0,1]	右眉高可见度点位占比
mouth_high_vis_ratio	float	[0,1]	嘴高可见度点位占比

onAIEvent

AI 事件回调

```
/// @param event dict格式的回调
```

```
- (void)onAIEvent:(id _Nonnull)event;
```

onTipsEvent

提示事件回调

```
/// @param event dict格式的回调
- (void)onTipsEvent:(id _Nonnull)event;
```

onAssetEvent

资源包事件回调

```
/// @param event string格式的回调
- (void)onAssetEvent:(id _Nonnull)event;
```

YTSDKLogListener

日志监听回调

```
@protocol YTSDKLogListener <NSObject>
```

成员函数

返回类型	函数名称
void	onLog

函数说明

onLog

日志监听回调

```
/// @param loggerLevel 返回当前日志等级
/// @param logInfo 返回当前日志信息
- (void)onLog:(YtSDKLoggerLevel) loggerLevel withInfo:(NSString * _Nonnull) logInfo;
```

uniapp

最近更新时间：2025-02-28 12:24:14

美颜特效 SDK uni-app 版本核心接口类 XmagicApi，用于更新美颜数值、调用动效等功能。

接口说明：

API	描述
setResPath	设置资源存放的路径，用于把美颜资源复制到此路径下，如果不设置此路径，SDK 内部会有一个默认路径
copyXmaigcRes	初始化美颜，用于复制美颜资源
setLicense	美颜鉴权
setLogLevel	设置日志级别
enableCustomVideoProcess	打开或关闭美颜
onResume	恢复美颜
onPause	暂停美颜
setEffect	设置美颜属性（**V0.3.5.0版本新增**）
@Deprecated updateProperty	设置美颜属性效果（**此接口已废弃**）
enableEnhancedMode	开启美颜增强模式（V2.5.1新增）。默认未开启，具体请参考 增强模式使用指南
setDowngradePerformance	调用此方法开启高性能模式。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。此方法需要在 enableCustomVideoProcess 方法之前调用
setAudioMute	设置静音
setFeatureEnableDisable	开启某些特性
addAiMode	添加 AI 模型文件，将下载好的AI模型文件复制到对应的文件夹下。如果 AI 模型放在工程中，则不需要调用此接口
setLibPathAndLoad	动态加载网络下载的 so 文件，这里只需要出入存放 so 的文件夹即可，仅支持 Android
on	订阅 SDK 的回调事件
off	取消订阅的事件

成员函数说明

setResPath

设置资源存放的路径，用于把美颜资源复制到此路径下，如果不设置此路径，SDK 内部会有一个默认路径。

```
/**
    * 设置存放美颜资源的路径，
    * @param {String} resourceDir 用于存放美颜资源的路径文件夹(绝对路径)，可参考demo
    */
   static setResPath(resourceDir)
```

参数

参数	含义
resourceDir	资源文件放置的目录

copyXmaigcRes

用于复制美颜资源。

```
/**
     * 用于复制美颜资源
     * @param {Function(result)} callback  result===true表示成功，false 表示失败
     */
    static copyXmaigcRes(callback)
```

参数

参数	含义
callback	由于资源复制是耗时任务，处理完成之后通过此接口回调结果

setLicense

进行美颜鉴权，app 启动之后，需要成功鉴权一次才能使用美颜功能。

```
 /**
     * 鉴权
     * @param {String} licenseUrl
     * @param {String} licenseKey
     * @param {Function(code,message)} callback code===0表示成功，其他错误码请参考鉴权 errorCode 说明表
https://cloud.tencent.com/document/product/616/65891#errorCode
     */
    static setLicense(licenseUrl, licenseKey, callback)
```

参数

参数	含义
licenseUrl	官网申请的用于美颜鉴权的 licenseUrl
licenseKey	官网申请的用于美颜鉴权的 licenseKey
callback	鉴权结果回调方法，code==0表示鉴权成功

setLogLevel

设置 native SDK 的日志级别。

```
 /**
     * 设置日志级别
     * @param {number} logLevel 参见{@link LogLevel}
     */
    static setLogLevel(logLevel)
```

参数	含义
logLevel	日志级别，具体参考 SDK 中的 LogLevel

enableCustomVideoProcess

开启或关闭美颜。

```
  /**
     * 打开或关闭美颜
     * @param {boolean} enable
```

```
     * @returns
     */
    static enableCustomVideoProcess(enable)
```

参数	含义
enable	开启或关闭美颜，true 表示开启美颜，false 表示关闭美颜

onResume

恢复美颜处理。

```
    static onResume()
```

onPause

暂停美颜处理。

```
    static onPause()
```

setEffect（V0.3.5.0新增）

设置美颜、美型、滤镜、美妆、贴纸、分割等效果，可在任意线程调用。具体参数请参考 美颜参数说明 。

```
/**
 * 更新美颜对象
 * @param effect 对象结构如下
 * {
 *      effectName:"",不为空的字符串，参考美颜参数表
 *      effectValue: 数值，一般为-100---100的值，可参考官网的美颜参数表
 *      resourcePath: 资源文件的路径，请参考美颜参数表 https://cloud.tencent.com/document/product/616/103616
 *      extraInfo:一个map集合，具体数值请参考美颜参数表
 * }
 */
static setEffect(effect)
```

updateProperty（此接口已废弃）

设置美颜属性。

```
    /**
     * 设置美颜属性
     * @param {null} property
     */
    static updateProperty(property)
```

参数	含义
property	美颜参数信息，具体参数请参见 美颜参数说明 Android 和 美颜参数说明 iOS

enableEnhancedMode

开启美颜增强模式。默认未开启，具体请参考 增强模式使用指南 。

```
/**
     * 开启增强模式
     */
    static enableEnhancedMode()
```

setDowngradePerformance

调用此方法开启高性能模式。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。

```
    /**
     * 开启性能模式，如果要使用此方法，此方法需要在 enableCustomVideoProcess 方法之前调用
     */
    static setDowngradePerformance()
```

setAudioMute

设置静音。

```
    /**
     * 设置静音
     */
    static setAudioMute(isMute)
```

参数	含义
isMute	true：静音

setFeatureEnableDisable

调用此方法开启高性能模式。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。

```
    /**
     * 开启性能模式，如果要使用此方法，此方法需要在 enableCustomVideoProcess 方法之前调用
     */
    static setDowngradePerformance(feature)
```

参数	含义
feature	feature 为 object 对象，key 的值参考 SDK 中的 FeatureName 对象，value 设置为 true 开启，false 关闭

addAiMode

设置模型文件，入股模型文件是动态下载的，则需要在下载成功之后通过此接口将模型文件设置为 SDK，下次启动时不需要再设置。

```
/**
   *设置模型文件路径、
   *
   * @param {String} aiModePath 模型文件路径(绝对路径)
   * @param callback
   */
  static addAiMode(aiModePath, callback)
```

参数	含义
aiModePath	模型文件路径(绝对路径)
callback	结果回调方法

setLibPathAndLoad

设置网络下载的 so 文件，只支持 Android 平台。

```
  /**
```

```
    * 动态加载网络下载的so文件，这里只需要出入存放so的文件夹即可
    * @param {String} soDir 下载好的so文件的存放目录路径
    */
  static setLibPathAndLoad(soDir)
```

参数	含义
soDir	soDir 下载好的 so 文件的存放目录路径

on

订阅 SDK 返回的事件。

```
/**
   * 订阅回调事件
   *
   * @param {String} event 事件名称
   * @param {Function} callback 事件名称以及回到方法可参考 {@link EventName}
   */
  static on(event, callback)
```

参数	含义
event	事件名称，可以参考 EventName 类，订阅对应的事件
callback	SDK 通过此方法将订阅的事件返回给 JS 侧

off

取消对应的事件。

```
/**
   * 取消回调事件
   *
   * @param {String} event 事件名称 可参考 {@link EventName}
   */
  static off(event)
```

参数	含义
event	事件名称，可以参考 EventName 类

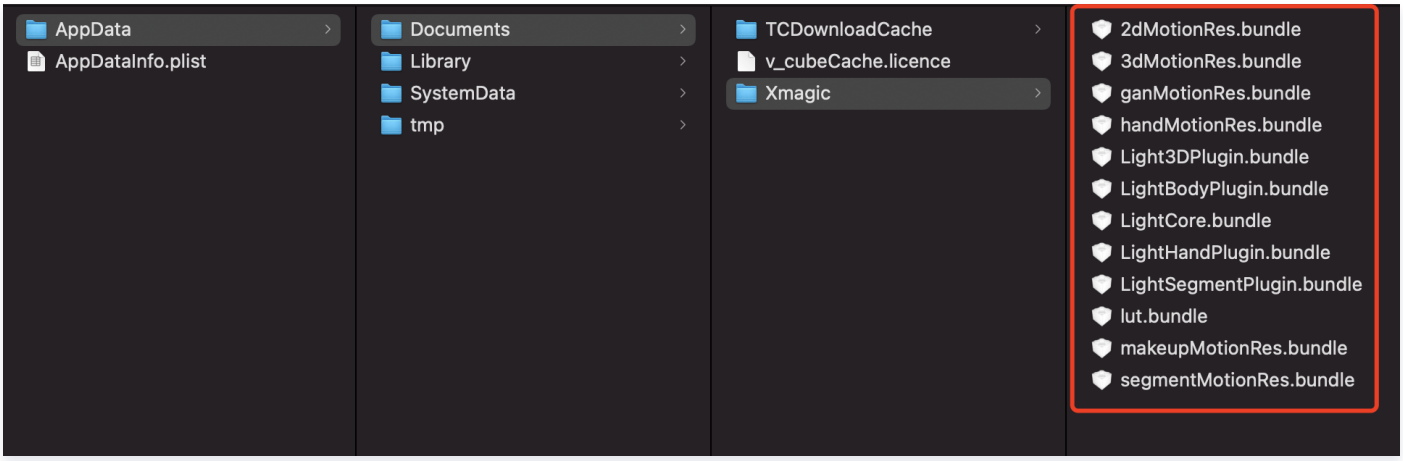
功能实践
SDK 包瘦身
iOS

最近更新时间：2024-05-31 14:41:22

资源动态下载

为了减少包大小，您可以将 SDK 所需的模型资源和动效资源 MotionRes（部分基础版 SDK 无动效资源）改为联网下载。在下载成功后，将上述文件的路径设置给 SDK。

1. 把美颜资源的 ZIP 包上传至云端，生成下载 URL。例如：`https://服务器地址/LightCore.bundle.zip`。
2. 在工程里面使用生成的 URL 下载文件并解压到沙盒（例如：沙盒路径 `Document/Xmagic`）。此时 Document/Xmagic 文件夹里面有 SDK 需要的资源。



3. SDK 初始化时，在 root_path 字段传入上一步的沙盒路径。

```
 NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                              @"root_path":_filePath ,//_filePath为美颜资源下载到本地后的父目
录:Ducument/Xmagic,
                              @"tnn_"
                              @"beauty_config":beautyConfigJson
 };
 // Init beauty kit                                  @"root_path":Ducument/Xmagic,
 self.beautyKit = [[XMagic alloc] initWithRenderSize:_inputSize assetsDict:assetsDict];
```

4. 设置美颜面板各个美颜效果的 icon，在下载的资源文件里面获取对应的 image。

```
NSMutableArray *arrayModels = [NSMutableArray array];
for (NSDictionary* dict in motionArray) {
 BeautyCellModel* model = [BeautyCellModel beautyWithDict:dict];
 // Load default mainbundle path of motionres
 if ([model.title isEqualToString:NSLocalizedString(@"item_none_label",nil)]) {
     model.icon = [NSString stringWithFormat:@"%@/%@.png", [[NSBundle mainBundle] bundlePath],
model.key];
     [arrayModels addObject:model];
 } else {
     if(_useNetResource && _filePath != nil){ //使用网络资源时
         NSString *DirPath = [_filePath stringByAppendingPathComponent:@"2dMotionRes.bundle/"]; //获取美
颜资源的绝对路径
         model.icon = [NSString stringWithFormat:@"%@/%@/template.png", DirPath, model.key];
     }else{
```

```
        model.icon = [NSString stringWithFormat:@"%@/%@/template.png", [[NSBundle mainBundle]
pathForResource:@"2dMotionRes" ofType:@"bundle"], model.key];
    }
    if ([fileManager fileExistsAtPath:model.icon]) {
        [arrayModels addObject:model];
    }
 }
}
```

5. 设置美颜效果的参数传递（参数的具体设置请参见 API 文档）：

```
/// @brief 配置美颜各种效果
/// @param propertyType 效果类型 字符串: beauty, lut, motion
/// @param propertyName 效果名称
/// @param propertyValue 效果数值
/// @param extraInfo 预留扩展，附加额外配置dict
/// @return 成功返回0，失败返回其他
/// @note 具体说明
/**
| 效果类型 | 效果名称 | 效果值 | 说明 | 备注 |
| :---- | :---- |:---- | :---- | :---- |
| beauty | 美颜id名称 | 美颜效果强度数值 |美颜类型配置接口 | 无 |
| lut | 滤镜路径+滤镜名称 | 滤镜强度数值 | 滤镜类型配置接口 | 无 |
| motion | 动效路径名称 | 动效路径 | 动效类型配置接口| **注意**: 如果资源中有zip，请确保传入动效路径为可写路径，否则跟app包走需要手动unzip才可以使用 |
**/
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString
*_Nonnull)propertyName withData:(NSString*_Nonnull)propertyValue withExtraInfo:(id _Nullable)extraInfo;
```

示例

设置美颜效果

“美颜”和“美体”的特效，不需要做处理，在 SDK 内部会自动使用下载的资源文件。以使用美颜中的美白效果为例，SDK 传参示例：

```
[self.beautyKitRef configPropertyWithType:@"beauty" withName:@"beauty.whiten" withData:@"30"
withExtraInfo:nil];
```

此时，传入到 SDK 的各参数的值分别是：

字段	值
propertyType	beauty
propertyName	beauty.whiten
propertyValue	30
extraInfo	nil

设置滤镜效果

需要对 key 做处理，可以使用内置的本地美颜资源或者网络下载到本地以后的美颜资源：

```
NSString *key = [_model.lutIDs[index] path];
if (key != nil) {
    key = [@"lut.bundle/" stringByAppendingPathComponent:key];//滤镜效果图片的相对路径
}
if(_useNetResource && _filePath != nil){ //如果使用下载的美颜资源
    key = [_filePath stringByAppendingPathComponent:key];//生成效果图片的绝对路径
```

```
}
[self.beautyKitRef configPropertyWithType:@"lut" withName:key withData:[NSString
stringWithFormat:@"%f",value] withExtraInfo:nil];
```

设置滤镜中的白皙效果

使用本地资源和网络资源的传参示例：

字段	使用本地资源时传入的参数	使用网络资源时传入的参数	备注
propertyType	lut	lut	-
propertyName	lut.bundle/n_baixi.png	/var/mobile/Containers/Data/Application/25C7D01A-73F6-4F1B-AEB6-5EE03A221D18/Documents/Xmagic/lut.bundle/n_baixi.png	文件路径
propertyValue	60.000000	60.000000	-
extraInfo	null	null	-

设置动效、美妆、分割效果

需要对 propertyValue 字段做处理，可以使用内置的本地美颜资源或者网络下载到本地以后的美颜资源。

```
NSString *key = [_model.motionIDs[index] key];
        NSString *path = [_model.motionIDs[index] path];
        NSString *motionRootPath = path==nil?[[NSBundle mainBundle] pathForResource:@"MotionRes"
ofType:@"bundle"]:path;
        if(_useNetResource && _filePath != nil){//如果使用下载的美颜资源
            motionRootPath = [_filePath stringByAppendingPathComponent:@"2dMotionRes.bundle"];//生成
2dMotionRes的绝对路径
        }
        [self.beautyKitRef configPropertyWithType:@"motion" withName:key withData:motionRootPath
withExtraInfo:nil];
```

设置 2D 动效—可爱涂鸦的效果

使用本地资源和网络资源的传参示例：

字段	使用本地资源时传入的参数	使用网络资源时传入的参数	备注
propertyType	motion	motion	-
propertyName	video_keaituya	video_keaituya	-
propertyValue	/private/var/containers/Bundle/Application/FD2D7912-E58E-4584-B7E4-8715B8D2338F/BeautyDemo.app/2dMotionRes.bundle	/var/mobile/Containers/Data/Application/25C7D01A-73F6-4F1B-AEB6-5EE03A221D18/Documents/Xmagic/2dMotionRes.bundle	文件路径
extraInfo	nil	nil	-

Android

最近更新时间：2024-05-31 14:41:22

为了减少包体大小，您可将 SDK 所需的 so 库、模型资源改为联网下载，只需要在 SDK 初始化之前下载好这些文件即可。对于滤镜和动效资源，建议在用户点击使用时，点击一项下载一项。

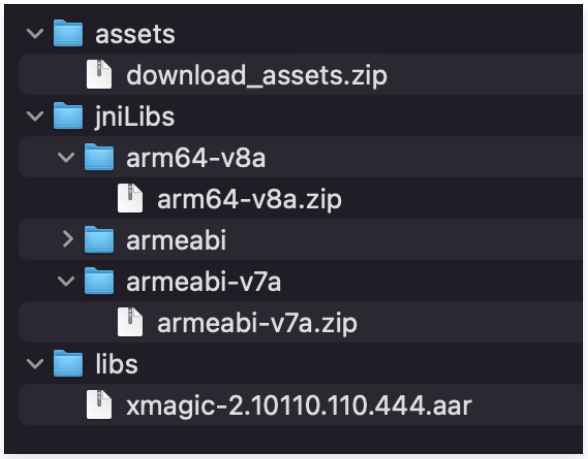
Demo工程： TEBeauty_Download_Example

从 github clone 出 demo工程，根据 TEBeauty_Download_Example/readme 文档配置和运行TEBeauty_Download_Example，以了解动态下载的整体流程。

动态下载 so 库和模型资源

如果您复用 Demo 中的下载代码

1. 将 demo 工程 `com.tencent.demo.download` 目录下的代码拷贝到您的工程。
2. 下载SDK，解压，然后从"SDK"目录找到 `.zip` 格式的压缩包，再次解压，您将看到如下文件：



3. 将 download_assets.zip, arm64-v8a.zip, armeabi-v7a.zip 上传到您的服务器，得到下载地址。计算出这3个 zip 文件的 MD5。将这3个下载地址和 MD5 填在 `ResDownloadConfig.java` 里对应的常量上。
4. 参见 `TEMenuActivity.java` 里的代码，通过 `ResDownloadUtil.getValidLibsDirectory` 检查 so 库是否已经下载好，如果没下载好，则调用 `ResDownloadUtil.checkOrDownloadFiles` 启动下载，下载成功后得到so库的路径sdkLibraryDirectory，然后调用 `XmagicApi.setLibPathAndLoad(sdkLibraryDirectory)` 加载so库。

```
String validLibsDirectory = ResDownloadUtil.getValidLibsDirectory(this, libraryMD5);
if (validLibsDirectory == null) {
    ResDownloadUtil.checkOrDownloadFiles(this, ResDownloadUtil.FILE_TYPE_LIBS, libraryURL,
libraryMD5,
        new TEDownloadListener() {
            @Override
            public void onDownloadSuccess(String directory) {
              sdkLibraryDirectory = directory;
            }

            @Override
            public void onDownloading(int progress) {
            }

            @Override
            public void onDownloadFailed(int errorCode) {
            }
        });
} else {
```

```
    sdkLibraryDirectory = validLibsDirectory;
}
```

5. 参见 `TEMenuActivity.java` 里的代码，通过 `ResDownloadUtil.getValidAssetsDirectory` 检查模型资源是否已经下载好，如果没下载好，则调用 `ResDownloadUtil.checkOrDownloadFiles` 启动下载，模块内部会把这些资源下载、整理、拷贝到 `AppConfig.resPathForSDK` 目录，在 new XmagicApi 时传给 SDK。

```
String validAssetsDirectory = ResDownloadUtil.getValidAssetsDirectory(this,
ResDownloadConfig.DOWNLOAD_MD5_ASSETS);
if (TextUtils.isEmpty(validAssetsDirectory)) {
    ResDownloadUtil.checkOrDownloadFiles(this, ResDownloadUtil.FILE_TYPE_ASSETS,
            ResDownloadConfig.DOWNLOAD_URL_ASSETS,
            ResDownloadConfig.DOWNLOAD_MD5_ASSETS, new TEDownloadListener() {
                @Override
                public void onDownloadSuccess(String directory) {
                }

                @Override
                public void onDownloading(int progress) {
                }

                @Override
                public void onDownloadFailed(int errorCode) {
                }
            });
} else {

}
```

6. Demo 中默认是开启断点续传功能的（`ResDownloadUtil.java` 的 `ENABLE_RESUME_FROM_BREAKPOINT` 属性为 `true`），可以确保在下载异常中断后，下次继续从中断点接着下载。如果您也想开启断点续传，请确保您的下载服务器支持断点续传能力。检测方法：

判断服务器是否支持断点续传，看Web服务器是否支持Range请求即可。测试方法是在命令行中执行curl命令：

```
curl -i --range 0-9 https://您的服务器地址/待下载的文件名
```

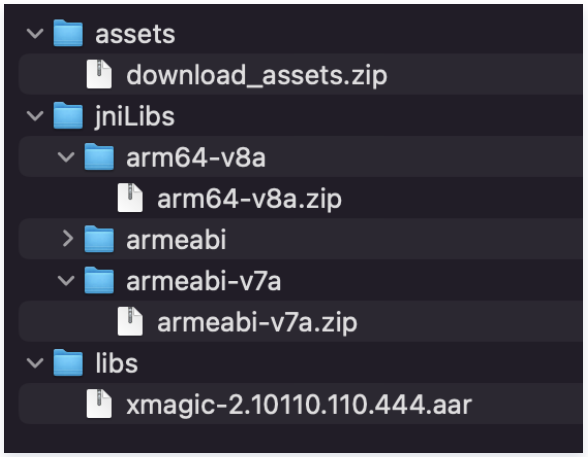
例如：

```
curl -i --range 0-9 https://mediacloud-76607.gzc.vod.tencent-
cloud.com/TencentEffect/Android/2.4.1.119/xmagic_S1-04_android_2.4.1.119.zip
```

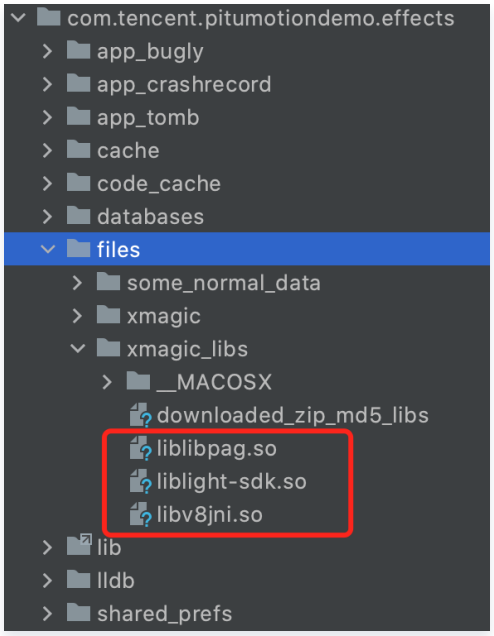
如果返回的内容有Content-Range 字段，则表示服务器支持断点续传。

如果您自己做下载

1. 下载SDK，解压，然后从"SDK"目录找到 .zip 格式的压缩包，再次解压，您将看到如下文件。assets 里的模型文件和 jniLibs 里的 so 文件可以动态下载。libs 里的 aar 则需要内置到包里。



2. 下载完 so 文件并解压后，调用 `XmagicApi.setLibPathAndLoad(/path/to/so/files)` 加载 so 库。



> **注意：**
> 强烈建议您将 so 下载到 App 的私有目录，而不是外部存储，以防 so 被清理软件误删。同时，建议您根据用户手机的 CPU 类型按需下载 v8a 或 v7a 的 so，以加快下载速度，这里可以参考 Demo 工程的 TEMenuActivity 的做法。

3. 对于 download_assets.zip 包里的文件，下载完成后，解压，然后调用下面的代码让 SDK 把文件拷贝到正确的目录（`AppConfig.resPathForSDK` 所指向的目录），代码中的 `downloadedDirectory` 是您解压后的文件所在目录。
`addAiModeFiles` 返回的错误码-2表示文件拷贝过程中失败了，可能是手机空间不足或 IO 异常，可尝试重新拷贝或重新下载。

```
  private static boolean organizeAssetsDirectory(String downloadedDirectory) {
    for (String path : XmagicResourceUtil.AI_MODE_DIR_NAMES) {
        if (XmagicApi.addAiModeFiles(downloadedDirectory + File.separator + path,
AppConfig.resPathForSDK) == -2) {
            return false;
        }
    }
    return true;
}
```

> **注意：**

> 当 SDK 版本更新时，对应的 so 和 assets 可能会发生变化，为确保兼容性，您需要重新下载这些文件。建议参见 Demo 中的方式，利用 MD5 进行校验 。

滤镜和动效资源下载

- 每个滤镜都是一张 png 格式的图片，每个动效都是一个文件夹，对于滤镜和动效资源，建议在用户点击使用时，点击一项下载一项。下载成功后，调用 SDK 的 XmagicApi.setEffect 接口，将滤镜路径或动效文件夹的路径设置给 SDK 即可。
- 滤镜和动效资源可以保存在手机任意目录，我们建议您保存在 app 私有目录，防止被误清理。

Flutter

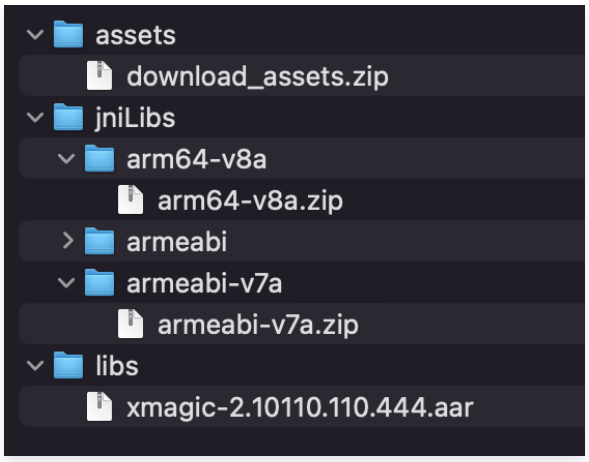
最近更新时间：2025-03-03 17:11:42

为了减少包体大小，您可以将 SDK 所需的 so 库（Android 有，iOS 没有）、模型资源和素材资源（滤镜和贴纸）改为联网下载。

Android

SDK 集成

下载 SDK，解压，然后从"SDK"目录找到 .zip 格式的压缩包，再次解压，您将看到如下文件。将 `assets` 里的模型文件和 `jniLibs` 里的 `so` 文件压缩包在您的服务器，然后联网下载使用。将 `libs` 下的 `xmagic-xxxx.aar` 复制到 `android/app/libs` 文件夹下，并在 `app/build.gradle` 中的 `dependencies` 添加 `api fileTree(dir: "libs", include: ['*.aar'])` 。



动态加载 so

将 so 压缩包下载到应用安装目录下，并进行解压，然后调用 `TencentEffectApiAndroid` 对象的 `setLibPathAndLoad` 方法加载so。

```
/**
* @param libPath 用于存放so的文件夹路径，比如: xxx/xxx/arm64-v8a 或者xxx/xxx/armeabi-v7a
**/
Future<bool> setLibPathAndLoad(String libPath);
```

动态加载模型

将模型文件下载到应用安装目录下，并进行解压，调用 `TencentEffectApiAndroid` 对象的 `addAiMode` 将模型文件复制到指定目录下。

```
 /**
 * @param inputDir 要复制的文件夹的路径，这个指的是"Light3DPlugin", "LightCore", "LightHandPlugin",
"LightBodyPlugin", "LightSegmentPlugin" 文件夹的路径
 * @param resDir 这个路径要和 setResourcePath 方法设置的路径，要和这个保持一致。
 * @param callBack 复制结果回调，0 表示复制成功
 */
  void addAiMode(String inputDir, String resDir, AddAiModeCallBack callBack) ;
```

动态加载素材

素材文件自行下载，下载到 SD 卡或者安装目录下，并解压，使用素材的时候调用 setEffect 方法， `resPath` 参数填写素材路径即可。

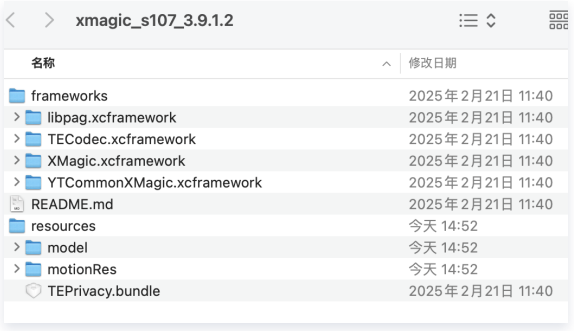
> ⚠ 注意：
>
> 1. 如果您是使用的demo中的面板，demo中的面板默认是使用的 `setResourcePath` 的路径+ `json` 中配置的路径，所以在使用面板并动态下载素材的时候，需要将素材下载到 `setResourcePath +json` 配置文件的路径下。

2. 这里的素材指的是 滤镜资源和贴纸资源。

iOS

SDK 集成

下载 SDK，解压，如下图，`frameworks` 是 SDK，`resources` 下 `model` 为模型文件，`motionRes` 是测试使用的素材。



打开您的 Xcode 工程项目，把 `frameworks` 文件夹里面的 `xcframework` 添加到实际工程中，选择要运行的 target，选中 General 项，单击 `Frameworks,Libraries,and Embedded Content` 项展开，单击底下的“+”号图标去添加依赖库。

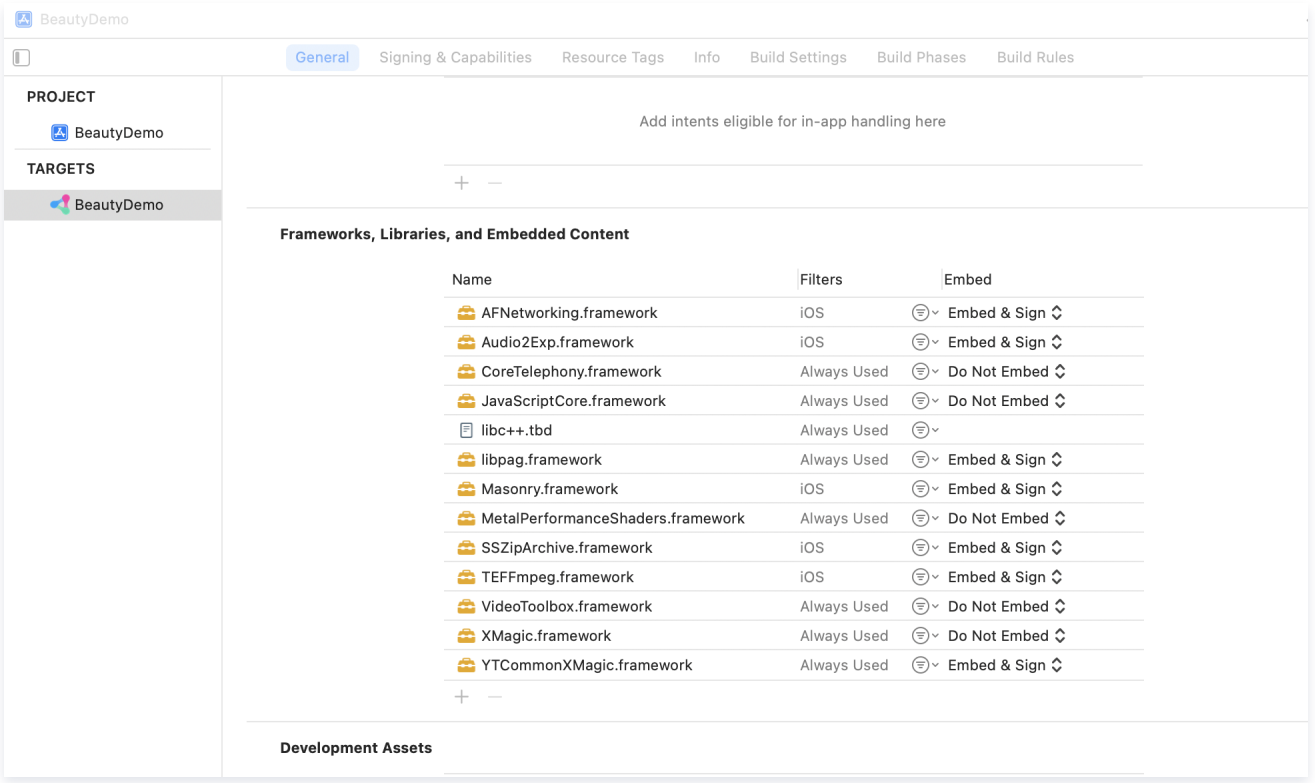
添加 SDK：

`XMagic.xcframework`、`YTCommonXMagic.xcframework`、`libpag.xcframework`、`Audio2Exp.xcframework`、`TECodec.xcframework`

添加依赖库：

`MetalPerformanceShaders.framework`、`CoreTelephony.framework`、`JavaScriptCore.framework`、`VideoToolbox.framework`、`libc++.tbd`

根据需要添加其它工具库 `Masonry.framework`（控件布局库）、SSZipArchive（文件解压库）。



动态加载模型

将模型文件下载到沙盒下，并解压（例如：沙盒路径 `Document/Xmagic`）。在使用 `setResourcePath` 的时候设置这个路径即可，这样美颜在初始化的时候就可以使用到下载的模型文件。

动态加载素材

素材文件自行下载，下载文件并解压到沙盒，并解压，使用素材的时候调用 **setEffect** 方法， `resPath` 参数填写素材路径即可。

> ⚠ **注意：**
>
> 1. 如您是使用的 demo 中的面板，demo 中的面板默认是使用的 `setResourcePath` 的路径+ `json` 中配置的路径，所以在使用面板并动态下载素材的时候，需要将素材下载到 `setResourcePath +json` 配置文件的路径下。
> 2. 这里的素材指的是 滤镜资源和贴纸资源。

> ⚠ **注意：**
>
> 模型文件需要在开启美颜之前处理，so 文件需要在鉴权之前处理，所以可以放在最早进行处理。

SDK 集成问题排查
Android

最近更新时间：2025-02-28 12:24:14

1. Android release 包报错找不到某些方法，如何解决?

- 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.** { *;}
```

2. Android SDK 集成到宿主工程报 gson 库冲突，如何解决?

在宿主工程 `build.gradle` 文件中添加如下代码：

```
Android{
  configurations {
    all*.exclude group: 'com.google.code.gson'
  }
}
```

3. Android targetSdkVersion 为31或更高时，so 库没有加载成功? 或者无法使用 GAN 类型特效（例如童话脸、童年泡泡糖等）?

Android targetSdkVersion 为31或更高版本时需要在 app 模块下找到 `AndroidManifest.xml` 文件，在 application 标签内加入如下标签：

```
  <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。不建议设置为true，否则可能导致用户无法安装app。
        //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，美颜特效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
```

具体请参见 开发指引 。

4. 使用美颜时传递的纹理是横向纹理，如何解决?

可以使用 demo 中工具类 `TextureConverter.java` 的`convert` 方法对纹理进行旋转，转换为竖屏，然后再传递给美颜 SDK。

```
    /**
     * 此方法用于对rgba纹理进行旋转和镜像处理。处理过程为：先顺时针旋转rotation度（可取值0,90,180，270），再进行左右翻转(flipHorizontal）和 上下翻转（flipVertical）。
     * 使用场景：某些推流SDK返回的纹理是横屏纹理或者画面中人物朝向不对，而美颜特效SDK要求纹理中的人物是正向的，所以可以通过此方法对纹理进行转换。
```

```
     *
     * @param srcID    rgba纹理
     * @param width    纹理宽度
     * @param height   纹理高度
     * @param rotation 需要进行旋转的角度。
     * @return 旋转后的纹理，注意：如果旋转90或者270度，那么宽度需要进行交换。
     */
    public int convert(int srcID, int width, int height, @RotationDegreesValue int rotation, boolean
flipVertical, boolean flipHorizontal)
```

5. 使用美颜时传递的纹理是 oes 纹理，如何解决?

可以使用 demo 中工具类 `TextureConverter.java` 的`oes2Rgba` 方法对纹理进行转换，转换为 RGBA 纹理，然后再传递给美颜 SDK。

```
   /**
     * 此方法用于将oes纹理转换为rgba纹理
     *
     * @param srcID  oes 纹理
     * @param width  纹理宽度
     * @param height 纹理高度
     * @return rgba纹理ID
     */
    public int oes2Rgba(int srcID, int width, int height)
```

6. 如果想使用别的版本的 pag 如何解决? V3.5.0及以上支持

客户集成美颜 SDK 时:
如果是通过 Maven 集成，通过 implementation TencentEffect 就能引入 pag。如果您不想用 TencentEffect 依赖的 pag，可以通过 exclude 排除，然后在自己 app 的 build.gradle 中引入您需要的 pag 版本:

```
implementation ('com.tencent.mediacloud:TencentEffect_S1-04:版本号'){
       exclude group:       "com.tencent.tav", module: "libpag"
}
```

如果是下载美颜 SDK 的 aar 手动集成，在项目中依赖 TencentEffect.aar，这个 aar 是不带 pag 的，您还需要在 app 的 build.gradle 加一句 implementation pag 引入 pag 才能用:

```
implementation 'com.tencent.tav:libpag:4.3.33-noffavc'
```

如果您想动态下载 pag 的 so，请从 pag 官网 找到您需要的版本，下载 aar，将 .aar 重命名为 .zip，解压，剔除其中的 so，再把剩余文件压缩为 .zip，然后重命名为 .aar，最后引入这个不包含 so 的 pag aar，pag 的 so 则联网动态下载。

iOS

最近更新时间：2024-05-31 14:41:22

1. iOS 导入资源运行后报错?

Xcode 12.X 版本编译提示：Building for iOS Simulator, but the linked and embedded framework '.framework'...
在 Build Settings > Build Options > Validate Workspace 改为 Yes，再单击运行。

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

2. 滤镜设置没反应?

检查下设置的值是否正确，范围为 0~100，可能值太小了效果不明显。

3. iOS Demo 编译，生成 dSYM 时报错?

- 报错信息：

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

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

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

1.1 打开 demo/copy_framework.sh 。
1.2 `$(which cmake)` 改为本地 cmake 绝对路径。
1.3 签名 `Apple Development: ......` 改为自己的账号。

4. iOS Demo，进入主页显示授权错误?

查看日志中打印的授权失败错误码。如果使用的是本地 License 文件，检查文件是否添加进工程。

5. iOS Demo 编译报错?

- 报错信息：

```
unexpected service error: build aborted due to an internal error: unable to write manifest to-xxxx-
manifest.xcbuild': mkdir(/data, S_IRWXU | S_IRWXG | S_IRWXO): Read-only file system (30):
```

- 解决方法：

1.1 在 File > Project settings > Build System 选择 Legacy Build System。
1.2 Xcode 13.0++ 需要在 File > Workspace Settings 勾选 Do not show a diagnostic issue about build system deprecation。

性能调优
低端机性能优化实践教程

最近更新时间：2025-02-28 12:24:14

美颜特效涉及 AI 检测、图像处理、2D 和 3D 图形渲染、动画特效等操作，会占用一定的 CPU 和 GPU 资源。如果在直播、拍摄场景时，系统本身负载已经很高，再叠加上美颜特效，在性能较差的设备上可能出现卡顿、掉帧现象。因此我们整理了低端机性能优化实践教程，尽可能减少美颜特效 SDK 在低端机上的性能开销，确保用户有良好的使用体验。

低端机的定义

SDK 提供了 `getDeviceLevel` 接口获取设备等级（接口说明见：Android ，iOS ），等级取值为1 ~ 5，1为最低端机，5为最高端机。我们建议将等级小于等于3的设备视为低端机。
您也可以根据自身产品数据以及自身App的性能消耗情况，自行判断当前设备的等级。
通过判断不同的设备等级，结合以下措施减少低端机的性能消耗：

措施一：使用 SDK 的 Normal 模式

从 SDK V3.9.0 开始，创建 SDK 时必须指定 EffectMode，它有两个取值： EffectMode_Normal 和 EffectMode_Pro。

- EffectMode_Normal 等价于旧版本 SDK 的“高性能模式”。
- EffectMode_Pro 等价于旧版本 SDK 的默认模式。

建议在低端机上使用 EffectMode_Normal。更多详细说明见：EffectMode（高性能模式）使用指引 。

措施二：关闭 SDK 的某些高级能力

通过 setFeatureEnableDisable 接口关闭某些高级能力：

- `FeatureName.WHITEN_ONLY_SKIN_AREA`
  美白仅对皮肤生效。默认未开启。开启此功能会触发开启“皮肤分割能力”。低端机上不建议开启。
- `FeatureName.SEGMENTATION_SKIN`
  皮肤分割能力，开启后可使磨皮和美白区域更精准，减少对周围环境的影响。SDK在设备等级大于等于4时会默认开启。低端机上不建议开启。
- `FeatureName.SEGMENTATION_FACE_BLOCK`
  人脸遮挡检测能力，开启后可避免妆容画到遮挡物上。SDK在设备等级大于等于5时会默认开启。低端机上不建议开启。
- `FeatureName.SMART_BEAUTY`
  智能美颜(为男性、宝宝减淡美颜美妆效果)。默认未开启。低端机上不建议开启。

此外，“美黑”能力也会触发开启 FeatureName.SEGMENTATION_SKIN 能力。低端机上不建议使用美黑能力。

措施三：使用轻美妆代替风格整妆

轻美妆是美颜特效 SDK 在V3.9.0版本推出的新功能，与之前的“风格整妆特效”相比，轻美妆性能更好，且能跟其他特效很好地叠加。
更多说明见：轻美妆使用说明 。

措施四：使用性能更好的特效素材

我们提供丰富的特效供客户选择。有些特效比较简单，在低端机上能流畅展示。但有些特效需要消耗较多的 CPU 和 GPU 资源，在低端机上不建议使用，例如 3D 特效，GAN 特效（例如变娃娃脸、变漫画脸），背景分割特效等等。
我们提供了低端机专区，供客户自行选择，详情见美颜特效 Demo。

其他优化措施

除了以上美颜特效相关的优化措施，也可以关注外部影响性能/流畅度的因素：

1、选择合适的分辨率

分辨率越高，SDK 需要处理的像素就越多。在低端机上直播或拍摄时，建议不要超过540P分辨率。

2、设置合适的日志开关

SDK 提供了 setXmagicLogLevel 接口（ Android ， iOS ）用于设置日志等级，默认等级为 WARN 或 INFO。您可以将它进一步提升到 ERROR 级别，以减少日志输出。切记不能设置为 DEBUG 级别，否则大量的日志会影响性能。

3、检查推流帧率

检查是否设置的比较低，建议调整到24fps以上。当您的应用在没有设置美颜的时候画面也不太流畅时，需要检查一下采集模块的相机帧率，可以适当提高相机帧率从而达到画面流畅的效果。如果您使用的是 TRTC，那么可以参见 此文档 调整帧率。

4、检查美颜特效之外的模块的性能

如果在使用美颜特效之前，您的应用就已经很卡了，或者 CPU 占用率已经很高，说明 APP 性能已经出现了问题，这种情况下再使用美颜特效，只能让情况变得更差。所以建议先优化美颜特效之外的模块的性能。

EffectMode（高性能模式）使用指引

最近更新时间：2024-11-20 12:00:32

EffectMode

从 SDK V3.9.0 开始，创建 SDK 时必须指定 EffectMode，它有两个取值： EffectMode_Normal 和 EffectMode_Pro。

- EffectMode_Normal 等价于旧版本 SDK 的“高性能模式”。
- EffectMode_Pro 等价于旧版本 SDK 的默认模式。

二者区别如下：

模式 / 差异	EffectMode_Normal (旧版的高性能模式)	EffectMode_Pro（旧版的默认模式）
性能差异	占用的系统 CPU/GPU 资源更少，可减少手机的发热和卡顿现象。适合低端机/中端机/高端机。	性能良好，在中、高端机上能流畅使用。
功能差异	以下项目不可用： 1. 眼部：眼宽、眼高、眼睛位置、祛眼袋、亮眼。 2. 眉毛：角度、距离、高度、长度、粗细、眉峰 3. 嘴部：微笑唇。 4. 面部：收下颌，祛皱、祛法令纹。 5. 鼻子：鼻梁、山根。 6. 其他：美黑、染发、弱光降噪。 另外，磨皮效果与 Pro 模式有差异，具体可在 Demo 中体验。	SDK 全功能可用。

高性能模式

- “高性能模式”是 SDK V3.9.0之前的概念，当时 SDK 有两种模式：高性能模式和默认模式。
- 从V3.9.0开始，高性能模式变成了 EffectMode_Normal，默认模式变成了 EffectMode_Pro。
- 高性能模式与默认模式的区别请参考上文中 EffectMode_Normal 和 EffectMode_Pro 的区别。

V3.9.0及之后如何设置 EffectMode

Android

方式一

如果您是直接使用的 `XmagicApi` 对象，那么请在创建 `XmagicApi` 对象时，在构造方法中指定 EffectMode：

```
public XmagicApi(Context context, EffectMode effectMode, String resDir)

public XmagicApi(Context context, EffectMode effectMode, String resDir, OnXmagicPropertyErrorListener
xmagicPropertyErrorListener)
```

方式二

如果您是使用的 TEBeautyKit 对象，可以调用如下方法开启高性能模式。

```
public TEBeautyKit(Context context, EffectMode effectMode)

public static void create(@NonNull Context context, EffectMode effectMode, @NonNull OnInitListener
initListener)
```

EffectMode 定义如下：

```
public enum EffectMode{
    NORMAL(0),
```

```
    PRO(1);

    private final int value;

    EffectMode(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

iOS

方式一

如果您是直接使用的 `XMagic` 对象，那么需要在初始化 `XMagic` 的时候指定 EffectMode，如下代码所示：

```
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle] bundlePath],
                             @"effect_mode":@(effectMode)
};
self.xmagic = [[XMagic alloc] initWithRenderSize:CGSizeMake(720, 1280) assetsDict:assetsDict];
```

方式二

如果您是使用的 TEBeautyKit 对象，请在调用 createXMagic 方法时传入 EffectMode 参数。

```
+ (void)createXMagic:(EffectMode)effectMode onInitListener:(OnInitListener _Nullable )onInitListener;
```

EffectMode 的定义如下：

```
typedef NS_ENUM(NSInteger, EffectMode) {
    EFFECT_MODE_NORMAL = 0,
    EFFECT_MODE_PRO = 1,
};
```

Flutter

可以通过调用 `TencentEffectApi` 的 `setDowngradePerformance` 方法开启。

> ⚠ **注意：**
> **此方法需要在开启美颜之前调用，也就是 `TRTC或者Live` 中的 `enableCustomVideoProcess` 方法之前调用。**

uniapp

可以通过调用 `XmagicApi` 的 `setDowngradePerformance` 方法开启。

> ⚠ **注意：**
> **此方法需要在开启美颜之前调用，也就是在 `enableCustomVideoProcess` 方法之前调用。**

V3.9.0之前如何开启高性能模式

Android

方式一

如果您是直接使用的 `XmagicApi` 对象，那么请在创建 `XmagicApi` 对象之后立即调用以下接口开启高性能模式：

- **SDK 3.7.0及以后：调用 `enableHighPerformance` 方法。**
- **SDK 3.7.0以前：调用 `setDowngradePerformance` 方法。**

方式二

如果您是使用的 TEBeautyKit 对象，可以调用如下方法开启高性能模式。

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

  /**
 *
 * 异步创建TEBeautyKit对象
 * @param context Android应用上下文
 * @param isEnableHighPerformance 是否开启增强模式
 * @param initListener 初始化回调接口
 */
public static void create(@NonNull Context context, boolean isEnableHighPerformance, @NonNull
OnInitListener initListener)
```

iOS

方式一

如果您是直接使用的 `XMagic` 对象，那么可以在初始化 `XMagic` 的时候开启：

- **SDK 3.7.0及以后：请在assetsDict字典中将 `enableHighPerformance` 设置为YES。**
- **SDK 3.7.0以前：请在assetsDict字典中将 `setDowngradePerformance` 设置为YES。**

```
NSDictionary *assetsDict = @{
@"core_name":@"LightCore.bundle",
@"root_path":[NSBundle mainBundle] bundlePath],
@"setDowngradePerformance":@(YES)//YES:开启高性能模式，NO：不开启高性能模式。默认不开启高性能模式。
};
self.xmagic = [[XMagic alloc] initWithRenderSize:CGSizeMake(720, 1280) assetsDict:assetsDict];
```

方式二

如果您是使用的 TEBeautyKit 对象，可以调用如下方法开启高性能模式。

```
 /**
 *
 * 创建TEBeautyKit对象
 * @param isEnableHighPerformance 是否开启高性能模式。YES：开启高性能模式；NO：不开启高性能模式
 * @param initListener 初始化回调接口
 */
+ (void)create:(BOOL)isEnableHighPerformance onInitListener:(OnInitListener _Nullable )onInitListener;
```

Flutter

可以通过调用 `TencentEffectApi` 的 `setDowngradePerformance` 方法开启。

> **注意：**
> 此方法需要在开启美颜之前调用，也就是 `TRTC`或者`Live` 中的 `enableCustomVideoProcess` 方法之前调用。

uniapp

可以通过调用 `XmagicApi` 的 `setDowngradePerformance` 方法开启。

> **注意：**
> 此方法需要在开启美颜之前调用，也就是在 `enableCustomVideoProcess` 方法之前调用。

性能问题排查

最近更新时间：2024-05-31 14:41:22

如果您的应用使用美颜时发现美颜处理过程耗时较长，可通过如下方法进行排查。

第一步：检查传入美颜画面的分辨率

- 原因：分辨率是指图像或视频的像素数量，通常以宽度和高度来表示。美颜处理涉及对图像进行复杂的算法计算和处理，例如磨皮、美白、去瑕疵等。因此，分辨率的大小会直接影响美颜处理的时长。
- 较高的分辨率意味着图像中有更多的像素，需要更多的计算和处理。这会导致美颜处理所需的时间更长。相比之下，较低的分辨率意味着图像中的像素较少，处理所需的计算量也较小，因此美颜处理的时长会相对较短。
- 此外，美颜处理通常涉及对图像的多个区域进行处理，例如人脸检测和人脸特征点定位。在较高分辨率的图像中，需要处理更多的像素和更复杂的图像细节，这可能需要更多的时间来完成。
- 因此，需要权衡分辨率和美颜效果之间的关系，以获得满意的处理速度和图像质量。

第二步：检查日志开关

当日志设置为 `Log.DEBUG` 时，美颜在处理过程中会打印大量的日志信息，从而影响性能，所以应用 release 包时设置为 `LOG.WARN` 。

第三步：3D/Gan 贴纸比较耗性能，在低端机上表现可能存在卡顿问题，可以根据实际情况是否开启使用。

第四步：画面卡顿问题，检查推流帧率是否设置的比较低，建议调整到24fps以上。

- 当您的应用在没有设置美颜的时候画面也不太流畅时，需要检查一下RTC模块的相机帧率，可以适当提高相机帧率从而达到画面流畅的效果。
- 如果您使用的是 TRTC，那么可以参见 此文档 调整帧率。

效果调优
增强模式使用指引

最近更新时间：2024-06-03 14:56:21

增强模式是什么?

SDK 建议设置的各项美颜参数范围是0 ~ 100或-100 ~ 100（ 见 美颜参数说明 ），在此范围内调整数值，通常都能达到令人满意的美颜效果。如果将强度调整到最大值或最小值之后仍然无法满足需求，则可以考虑使用增强模式，增强模式可以让美颜效果更明显，例如磨皮更明显、瘦脸瘦得更多等。

如何使用增强模式

在 SDK 3.5.0版本之后，我们优化了增强模式的使用方式，您**只需要设置更大的数值给 SDK 即可**，例如建议的数值范围是-100 ~ 100，那您可以设置-120 ~ 120给 SDK。

Android

1. **如果您使用了我们的 UI 组件 TEBeautyKit:**
   请调用 `TEBeautyKit` 的 `enableEnhancedMode` 方法，调用后，TEBeautyKit就会将面板上显示的数值乘以合适的倍数再设置给SDK。例如在UI面板上设置的瘦脸数值是80，TEBeautyKit会将它乘以 1.2 变成 96 再设置给 SDK。
2. **如果您没有使用 TEBeautyKit 而是直接使用 XmagicApi:**
   调用 XmagicApi 的 setEffect 方法时，将 value 数值乘以合适的倍数即可。

iOS

1. **如果您使用了我们的 UI 组件 TEBeautyKit:**
   使用 `TEPanelView` ，调用 `setEnhancedMode` 方法，调用后，TEBeautyKit 就会将面板上显示的数值乘以合适的倍数再设置给 SDK。例如在 UI 面板上设置的瘦脸数值是80，TEBeautyKit 会将它乘以1.2变成96再设置给 SDK。

```
/**
*
* 开启增强模式
* @param enhancedMode 是否开启增强模式。YES：开启增强模式；NO：不开启增强模式。默认不开启增强模式。
*/
[self.tePanelView setEnhancedMode:YES];
```

2. **如果您没有使用 TEBeautyKit而是直接使用 `XMagic` 对象:**
   调用 setEffect 方法时，将 value 数值乘以合适的倍数即可。

Flutter

1. 调用 `TencentEffectApi` 的 `enableEnhancedMode` 方法开启增强模式。
2. 使用 `setEffect` 方法设置美颜参数时， `effectValue` 的最大值可以为下表推荐的最大值。

```
void setEffect(String effectName,int effectValue,String? resourcePath,Map<String,String>?
extraInfo);
```

uniapp

1. 调用 `XmagicApi` 的 `enableEnhancedMode` 方法开启增强模式。
2. 使用 `setEffect` 方法设置美颜参数时， `effectValue` 的最大值可以为下表推荐的最大值。

```
/**
 * 更新美颜对象
 * @param effect 对象结构如下
 * {
 *      effectName:"",不为空的字符串，参考美颜参数表
 *      effectValue: 数值，一般为-100---100的值，可参考官网的美颜参数表
 *      resourcePath: 资源文件的路径，请参考美颜参数表
https://cloud.tencent.com/document/product/616/103616
 *      extraInfo:一个map集合，具体数值请参考美颜参数表
 * }
 */
static setEffect(effect)
```

增强模式推荐的增强倍数

我们提供了一份增强倍数的参考值，不建议超出我们的推荐值，否则美颜效果可能变差。参考值见下：

美颜项名称	建议最大增强倍数
美白，短脸，V脸，眼距，鼻子位置，祛法令纹，口红，立体	1.3倍
亮眼	1.5倍
腮红	1.8倍
其他	1.2倍

TEBeautyKit 在 `DefaultEnhancingStrategy.java` 中设置了上述增强倍数，您可以按需修改。如果是直接使用 Android 的 XmagicApi 或 iOS 的 XMagic ，那么在 setEffect 时，将 value 数值乘以合适的倍数即可。

效果问题排查

最近更新时间：2024-09-11 21:11:52

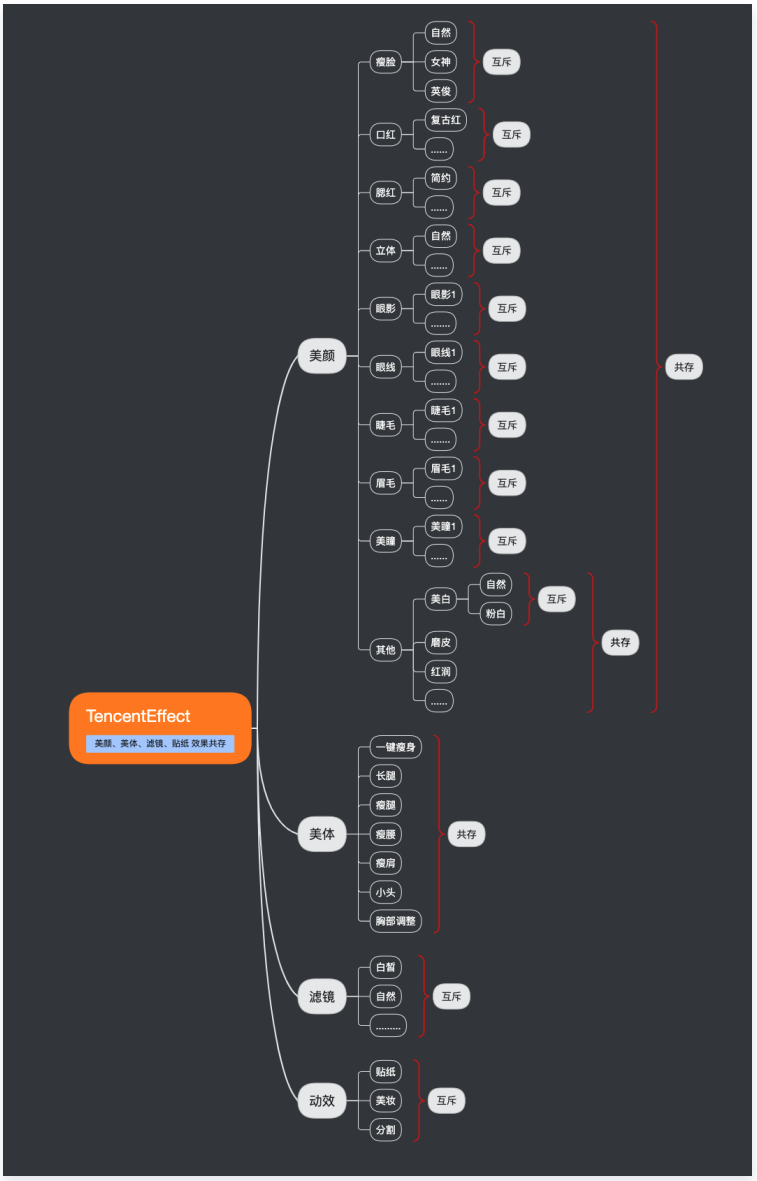
1. 画面出现噪点怎么办？

如果您在灯光弱的环境下，画面中出现了噪点，可以开启降噪属性。

2. 分割效果不太好怎么办？

在使用背景分割效果时，建议背景不要太复杂，背景颜色和衣服颜色不能太相似，否则分割效果会降低。

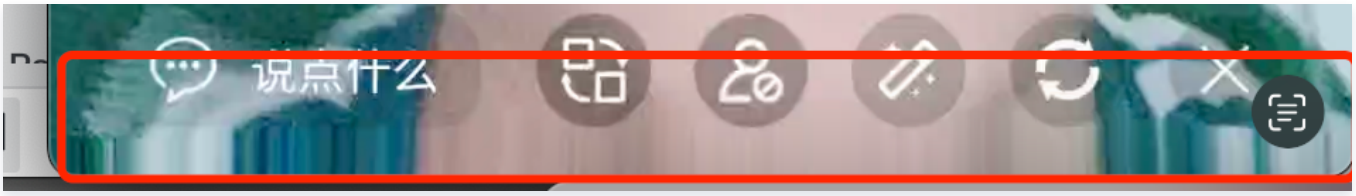
3. 美妆素材中的美颜跟美颜关系是什么？



4. 使用某一项美颜没有效果怎么办？

这里可能是license权限问题，可能是参数问题（例如滤镜和动效的路径问题），建议检查属性参数。

5. 美颜之后画面边缘模糊问题怎么办？



这种情况是因为开了瘦脸特效（瘦脸特效会导致拉伸脸部周围像素），如果脸比较靠近屏幕边缘，边缘的拉伸就更多。可通过裁剪画面边缘的方式进行处理，裁剪方法可以参考 demo。

6. 横屏时人脸没有效果怎么办?

检查画面中人脸方向，设置对应的偏移角度。

Android

1. Android 中可以使用 `readTexture` 方法获取当前画面，查看画面中人脸的方向，根据下图设置对应的角度。

```
  public static Bitmap readTexture(int texture, int width, int height) {
    int[] frame = new int[1];
    GLES20.glGenFramebuffers(1, frame, 0);
    GLES20.glBindFramebuffer(GLES20.GL_FRAMEBUFFER, frame[0]);
    GLES20.glFramebufferTexture2D(GLES20.GL_FRAMEBUFFER, GLES20.GL_COLOR_ATTACHMENT0,
GLES20.GL_TEXTURE_2D, texture, 0);
    byte[] data = new byte[width * height * 4];
    ByteBuffer buffer = ByteBuffer.wrap(data);
    GLES20.glPixelStorei(GLES20.GL_PACK_ALIGNMENT, GLES20.GL_TRUE);
    GLES20.glReadPixels(0, 0, width, height, GLES20.GL_RGBA, GLES20.GL_UNSIGNED_BYTE, buffer);
    Bitmap bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
    bitmap.copyPixelsFromBuffer(buffer);
    GLES20.glBindFramebuffer(GLES20.GL_FRAMEBUFFER, 0);
    GLES20.glDeleteFramebuffers(1, frame, 0);
    return bitmap;
}
```

2. Android 中调用 `setImageOrientation` 方法。

iOS

1. iOS中，可以使用 `readTexture` 方法获取当前画面，查看画面中人脸的方向，根据下图设置对应的角度。

```
  #import <OpenGLES/ES2/gl.h>
-(void)readTexture:(int)textureId width:(int)width height:(int)height{
    glBindTexture(GL_TEXTURE_2D, textureId);
    GLuint framebuffer;
    glGenFramebuffers(1, &framebuffer);
    glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, textureId, 0);
```

```
    GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
    if (status != GL_FRAMEBUFFER_COMPLETE) {
        NSLog(@"Framebuffer is not complete.");
    }
    GLubyte *pixels = (GLubyte *)malloc(width * height * 4 * sizeof(GLubyte));
    glReadPixels(0, 0, width, height, GL_RGBA, GL_UNSIGNED_BYTE, pixels);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glDeleteFramebuffers(1, &framebuffer);
    CVPixelBufferRef pixelBuffer = NULL;
    CVPixelBufferCreateWithBytes(NULL, width, height, kCVPixelFormatType_32BGRA, pixels, width * 4,
NULL, NULL, NULL, &pixelBuffer);
    free(pixels);
    CVPixelBufferRelease(pixelBuffer);
}
```

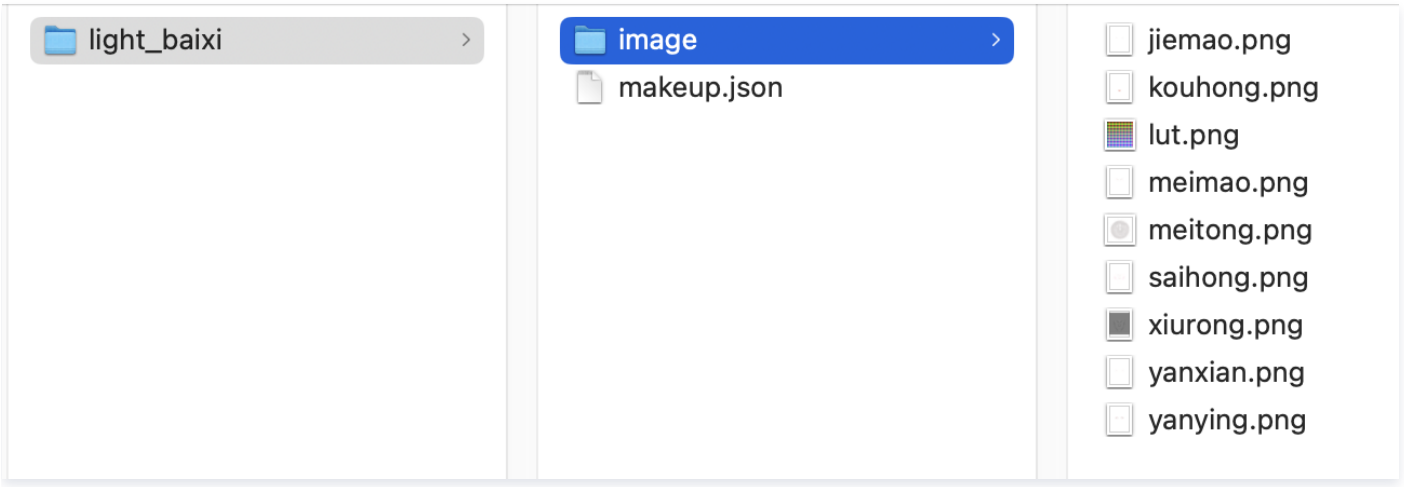
2. iOS中调用 `setImageOrientation` 方法。

轻美妆使用说明

最近更新时间：2025-02-28 12:24:14

什么是轻美妆

轻美妆是美颜特效 SDK 在V3.9.0版本推出的新功能。一套轻美妆里最多可包含这些美妆项目：滤镜、口红、腮红、立体、眼影、眼线、睫毛、眉毛、美瞳、双眼皮、卧蚕。轻美妆本质上跟SDK已有的“单点美妆”是同一个能力，可以理解为把多个单点美妆搭配组合在了一起。
与之前的“整妆特效”相比，**轻美妆性能更好，且能跟其他特效很好地叠加**。
一套轻美妆素材包含若干张美妆图片和一个 json 配置文件，例如“light_baixi”这套轻美妆的配置如下：



如何使用轻美妆

请调用 SDK 的 setEffect 接口使用轻美妆：

- effectName 为 `EFFECT_LIGHT_MAKEUP`。
- effectValue 为妆容强度，取值0 ~ 100。
- resourcePath 为轻美妆素材路径，即： path/to/your_light_makeup。
- extraInfo 是可选的，如果您想只修改轻美妆里的滤镜强度而不修改妆容强度，则在 extraInfo 里添加一对 key-value，key 为"makeupLutStrength"，value 为滤镜强度，取值为0 ~ 100，注意 value 也是字符串格式的。

注意事项

1、轻美妆与单点美妆的关系

轻美妆本质上是单点美妆的集合，因此后设置的会覆盖先设置的，具体如下：

- 场景1：如果先设置了若干单点美妆，再设置了一套轻美妆，则轻美妆效果会覆盖单点美妆的效果。
- 场景2：如果先设置了一套轻美妆（假设里面配置了口红、眼影、眉毛等），再设置单点美妆（例如口红），则最终效果是：新设置的这个口红 + 轻美妆里的眼影 + 轻美妆里的眉毛。

  对于场景2，我们 Demo 中的处理方式是：设置单点美妆时，清空轻美妆。您可以根据您的产品实际情况选择是否清空轻美妆。

2、轻美妆与其他特效的叠加关系

轻美妆可以和其他任意特效叠加，包括：美颜、美型、美体、贴纸、虚拟背景、运镜特效等。

3、轻美妆与贴纸特效里的“风格整妆”的关系

在V3.9.0之前，我们的整妆都是“风格整妆”，它本质上是一种特效，无法跟其他特效叠加，或叠加后不符合预期。而轻美妆可以跟其他任意特效叠加，也可以与风格整妆叠加（但不建议这么做）。在V3.9.0及之后的 Demo 中，我们把“风格整妆”的体验入口与 2D 贴纸和 3D 贴纸的入口放在了一起。

素材使用
素材集成指引
Android

最近更新时间：2025-02-28 12:24:14

滤镜

每个滤镜都是一张 png 格式的图片，使用时，您需要将图片路径传给 SDK。具体做法如下：

场景一：如果您使用了 TEBeautyKit

TEBeautyKit 是美颜特效的 UI 面板库，用于客户快速方便的使用和管理美颜功能。
操作步骤见下：

1. 参见文档 接入 TEBeautyKit。
2. 添加滤镜资源

   将新增加的滤镜图片放到您工程的 `assets/lut` 目录，然后，修改面板配置文件 `assets/beauty_panel/lut.json`，参见json中已有的内容新增加一项。APP 运行时，调用 `TEBeautyKit` 的 `copyRes` 方法，会把滤镜图片从 assets 目录 copy 到 `lut.json` 里配置的 downloadPath 目录。
3. 配置滤镜图标

   `lut.json` 的 icon 字段是该滤镜的图标，请把图标放在 `assets/beauty_panel/panel_icon/lut_icon` 目录。icon 字段的值也可以是图标的 URL，以 `http` 或 `https` 开头，TEBeautyKit 会从网络拉取这个图标。
4. 配置滤镜资源

   `lut.json` 的 resourceUri 字段是滤镜图片在 app 私有目录的保存路径，请参见json中已有的项目进行配置并把 resourceUri 的后缀"xxx.png"改为新增加的这个滤镜文件名，确保不会跟 `lut.json` 里的已有的滤镜冲突。resourceUri 字段也可以是滤镜图片的 URL，以 `http` 或 `https` 开头，点击后会联网下载，并保存在 `lut.json` 里配置的 downloadPath 目录。

场景二：如果您未使用 TEBeautyKit，而是直接集成美颜特效 SDK

1. 请将新增加的滤镜图片放到您工程的 assets 的任意目录，然后在 APP 初始化时，将它 copy 到 app 私有目录或 SD 卡，得到图片的路径，记为 `/path/to/your/lut_xxx.png`。为简化操作，建议您把图片放到 assets/lut 目录，然后从 demo 工程中把 TEBeautyKit 的 copyRes 代码 copy 过来使用。
2. 使用滤镜时，调用 SDK 的 setEffect 方法，将滤镜图片路径传给 SDK。

动效贴纸

每个动效都是一个文件夹，使用时，您需要将该文件夹的路径传给 SDK。具体做法如下：

场景一：如果您使用了 TEBeautyKit

TEBeautyKit 是美颜特效的 UI 面板库，用于客户快速方便的使用和管理美颜功能。
操作步骤见下：

1. 参见文档 接入 TEBeautyKit。
2. 添加动效素材

   将新增加的动效文件夹放到您工程的 `assets/MotionRes` 目录，然后，修改面板配置文件 `assets/beauty_panel/motions.json`，参见已有的内容新增加一项。APP 运行时，调用 `TEBeautyKit` 的 `copyRes` 方法，会把动效文件夹从 assets 目录 copy 到 `motions.json` 里配置的 downloadPath 目录。
3. 配置动效图标

   `motions.json` 的 icon 字段是该动效的图标，请把图标放在 `assets/beauty_panel/panel_icon/motions_icon` 目录。icon 字段的值也可以是图标的 URL，以 `http` 或 `https` 开头，TEBeautyKit 会从网络拉取这个图标。
4. 配置动效素材

   `motions.json` 的 resourceUri 字段是动效在 app 私有目录的保存路径，请参见已有的项目进行配置，并确保不会跟 `motions.json` 里的已有的动效冲突。resourceUri 字段也可以是动效压缩包的 URL，以 `http` 或 `https` 开头，点击后会联网下载，并保存在 `motions.json` 里配置的 downloadPath 目录。

场景二：如果您未使用 TEBeautyKit，而是直接集成美颜特效 SDK

请将新增加的动效文件夹放到您工程的 assets 的任意目录，然后在 APP 初始化时，将它 copy 到 app 私有目录或 SD 卡，得到动效的路径，记为 `/path/to/your/motion` 。使用动效时，调用 SDK 的 setEffect 方法，将该路径传给 SDK。

美妆、背景分割动效

与上文中的动效贴纸用法是一样的，二者对应的 json 文件分别是 `makeup.json` 和 `segmentation.json` 。

iOS

最近更新时间：2025-02-28 12:24:14

滤镜

每个滤镜都是一张 png 格式的图片，使用时，您需要将图片路径传给 SDK。

场景一：如果您使用了 TEBeautyKit

`TEBeautyKit` 是美颜特效的 UI 面板库，用于客户快速方便的使用和管理美颜功能。接入步骤见下：

1. 参见文档 接入 TEBeautyKit 。
2. 添加滤镜素材

   将新增加的滤镜图片放到您工程的 `lut.bundle` 目录，然后，修改面板配置文件 `TEBeautyKit/Assets/json/lut.json` ，参见json中已有的内容新增加一项。
3. 配置滤镜图标

   `lut.json` 的 icon 字段是该滤镜的图标，请把图标放在 `TEBeautyKit/Assets/BeautyRes` 目录。icon 字段的值也可以是图标的 URL，以 http 或 https 开头， `TEBeautyKit` 会从网络拉取这个图标。
4. 配置滤镜资源

   `lut.json` 的 `resourceUri` 字段是滤镜图片在 app 私有目录的保存路径，请参见json中已有的项目进行配置并把 `resourceUri` 的后缀 "xxx.png" 改为新增加的这个滤镜文件名，确保不会跟 lut.json 里的已有的滤镜冲突。 `resourceUri` 字段也可以是滤镜图片的 URL，以 http 或 https 开头，点击后会联网下载，并保存在 `lut.json` 里配置的 `downloadPath` 目录。

场景二：直接集成美颜特效 SDK

1. 请将新增加的滤镜图片放到您工程的 `lut.bundle` 目录。如果采用动态下载的方案，把滤镜图片下载到沙盒中，记录滤镜图片的路径。
2. 使用滤镜时，调用 SDK 的 `setEffect` 方法，将滤镜图片路径传给 SDK。操作方法参见 美颜参数说明 。

动效贴纸

每个动效都是一个文件夹，使用时，您需要将该文件夹的路径传给SDK。具体做法如下：

场景一：如果您使用了 TEBeautyKit

TEBeautyKit 是美颜特效的 UI 面板库，用于客户快速方便的使用和管理美颜功能。

1. 参见文档 接入 TEBeautyKit 。
2. 添加动效资源

   请将新增加的动效文件夹放到您工程对应的 `resource bundle` 目录： `2dMotionRes.bundle`中 是2D动效， `3dMotionRes.bundle` 中是3D动效， `ganMotionRes.bundle` 中是趣味动效， `handMotionRes.bundle` 中是手势动效， 然后，修改面板配置文件 `TEBeautyKit/Assets/json/motions.json` ，参考已有的内容新增加一项。
3. 配置动效icon

   `motions.json` 的 `icon` 字段是该动效的图标，请把图标放在 `TEBeautyKit/Assets/BeautyRes` 目录。icon字段的值也可以是图标的URL，以 `http`或`https` 开头， `TEBeautyKit` 会从网络拉取这个图标。
4. 配置动效资源

   `motions.json` 的 `resourceUri` 字段是动效在app私有目录的保存路径，请参考已有的项目进行配置，并确保不会跟 `motions.json` 里的已有的动效冲突。 `resourceUri` 字段也可以是动效压缩包的URL，以 `http`或`https` 开头，点击后会联网下载，并保存在 `motions.json` 里配置的 `downloadPath` 目录，动效压缩包需要解压以后才能使用。

场景二：如果您未使用 TEBeautyKit，而是直接集成美颜特效 SDK

请将新增加的动效文件夹放到您工程对应的 `resource bundle` 目录： `2dMotionRes.bundle` 中是2D动效， `3dMotionRes.bundle` 中是3D动效， `ganMotionRes.bundle` 中是趣味动效， `handMotionRes.bundle` 中是手势动效，在美颜特效 SDK version 3.6.0及以前的版本，如果是加密的动效文件，需要把动效文件拷贝到沙箱，记录这个动效文件的路径。如果采用动态下载的方案，把动效文件下载到沙盒中并解压，记录解压后的动效文件夹的路径。使用动效时，调用SDK的setEffect方法，将该路径传给SDK，详见： 美颜参数表 。

美妆、背景分割动效

与上文中的动效贴纸用法是一样的，二者对应的 json 文件分别是 `makeup.json` 和 `segmentation.json` 。

素材叠加指引

最近更新时间：2024-05-31 14:41:22

动效素材叠加是指多个动效素材可以同时生效。

素材叠加注意事项：

1. 客户需要自行管理素材之间是否适合叠加。举两个例子：
   例1：特效 A 是变成贵妃脸，特效 B 是变成童话脸，这两个特效叠加后可能会导致画面非常别扭。
   例2：特效 A 是个兔耳朵，特效 B 是猪耳朵，两个叠加后，就有两种耳朵。
   例1和例2这两种情况不适合叠加。如果特效 A 是兔耳朵，特效 B 是送一个飞吻，这两个特效不会冲突，就适合叠加。
2. 只支持简单素材之间的叠加。简单素材是指只有单动效能力、或者单美妆效果、或者单抠背等，复杂素材是指包含了多种效果。简单素材和复杂素材没有明确的界定，建议客户充分测试后，自行管理哪些素材之间可以叠加，哪些不能叠加。
3. 叠加时，有动作触发的特效（例如伸出手触发某个特效、微笑触发某个特效等）属于复杂特效，需要放在前面，简单特效放在后面叠加在它之上。
4. 使用示例：主播使用了特效 A，然后观众送礼物特效 B，B 要叠加在 A 之上，一段时间后 B 消失，恢复成特效 A。那么设置步骤如下：
   4.1 设置特效 A，mergeWithCurrentMotion 设置为 false。
   4.2 设置特效 B，mergeWithCurrentMotion 设置为 true。
   4.3 一小段时间后，再设置 A，mergeWithCurrentMotion 设置为 false。

如何配置同时生效?

V3.5.0及以上

1. 如果您使用 `setEffect` 方法来更新美颜属性，要实现素材叠加功能，可以在 `extrainfo` 中添加 `mergeWithCurrentMotion` 字段设置为 `"true"`
2. 如果使用的是 `updateProperty` 方法，那么可参见 V3.0.1 中列举的方法。

V3.0.1及以上

Android:

如果想要某个动效/美妆/分割素材叠加在当前素材上，则将该素材 XmagicProperty 对象的 mergeWithCurrentMotion 设置为 true。
XMagicProperty 对象的其他属性设置见 美颜参数设置 。

```
XmagicProperty xmagicProperty = new
XmagicProperty(XmagicProperty.Category.MOTION,"video_keaituya",mResPath+"xmagic/MotionRes/2dMotionRes/
video_keaituya",null,null);
xmagicProperty.mergeWithCurrentMotion = true;
mXMagicApi.updateProperty(xmagicProperty);
```

iOS:

如果想要某个动效/美妆/分割素材叠加在当前素材上，则设置该素材时，在 withExtraInfo 的字典中设置 mergeWithCurrentMotion 为 true，示例如下:

```
NSString *key = _xmagicUIProperty.property.Id;
NSString *value = [[NSBundle mainBundle] pathForResource:@"makeupMotionRes" ofType:@"bundle"];
NSDictionary* extraInfo = @{@"mergeWithCurrentMotion":@(true)};
 [self.beautyKitRef configPropertyWithType:@"motion" withName:key withData:[NSString
stringWithFormat:@"%@",value] withExtraInfo:extraInfo];
```

美颜参数说明
Android & iOS

最近更新时间：2025-02-28 12:24:14

当您使用 `setEffect` 函数更新美颜效果时，可参考如下参数表。参数表中的 `effectName` 常量定义在Android中位于 `XmagicConstant.java` 文件，iOS 位于 `XmagicConstant.h` 文件。

注意： 如果您使用的 SDK 版本是 `V3.3.0` 及之前版本，请参见 Android 旧版美颜参数表 ， iOS 旧版美颜参数表 。

美颜、美体

		effectName		effectValue	resourcePath
		常量名	常量值	效果强度	资源路径
美颜	美白-靓白 (V3.9.0)	BEAUTY_WHITEN0	beauty.lutFoundationAlpha0	0 ~ 100	V3.9.0以前：无 V3.9.0及之后：【可选】自定义美白滤镜路径
	美白-自然	BEAUTY_WHITEN	beauty.lutFoundationAlpha	0 ~ 100	V3.9.0以前：无 V3.9.0及之后：【可选】自定义美白滤镜路径
	美白-粉白	BEAUTY_WHITEN2	beauty.lutFoundationAlpha2	0 ~ 100	V3.9.0以前：无 V3.9.0及之后：【可选】自定义美白滤镜路径
	美白-冷白	BEAUTY_WHITEN3	beauty.lutFoundationAlpha3	0 ~ 100	V3.9.0以前：无 V3.9.0及之后：【可选】自定义美白滤镜路径
	美黑 (V3.7.0)	BEAUTY_BLACK_1	beauty.lutBlackAlpha1	0 ~ 100	无
	小麦色 (V3.7.0)	BEAUTY_BLACK_2	beauty.lutBlackAlpha2	0 ~ 100	无
	磨皮	BEAUTY_SMOOTH	smooth.smooth	0 ~ 100	无
	红润	BEAUTY_ROSY	smooth.rosy	0 ~ 100	无
画面调整	对比度	BEAUTY_CONTRAST	beauty.imageContrastAlpha	-100 ~ 100	无
	饱和度	BEAUTY_SATURATION	smooth.saturation	-100 ~ 100	无
	清晰度	BEAUTY_CLEAR	beauty.lutClearAlpha	0 ~ 100	无
	锐化	BEAUTY_SHARPEN	smooth.sharpen	0 ~ 100	无
	亮度 (V3.8.0)	BEAUTY_IMAGE_BRIGHTNESS	beauty.imageBrightness	-100 ~ 100	无
	弱光降噪 (V3.6.0)	BEAUTY_IMAGE_DENOISE	postEffect.denoise	0 ~ 100	无
	色温	BEAUTY_IMAGE_WARMTH	beauty.imageWarmth	-100 ~ 100	无
	色调	BEAUTY_IMAGE_TINT	beauty.imageTint	-100 ~ 100	无
高级美型	大眼	BEAUTY_ENLARGE_EYE	basicV7.enlargeEye	0 ~ 100	无
	亮眼	BEAUTY_EYE_LIGHTEN	beauty.eyeLighten	0 ~ 100	无
	眼距	BEAUTY_EYE_DISTANCE	basicV7.eyeDistance	-100 ~ 100	无
	眼角	BEAUTY_EYE_ANGLE	basicV7.eyeAngle	-100 ~ 100	无

眼宽	BEAUTY_EYE_WIDTH	basicV7.eyeWidth	-100 ~ 100	无
眼高	BEAUTY_EYE_HEIGHT	basicV7.eyeHeight	-100 ~ 100	无
眼睛位置 (V3.8.0)	BEAUTY_EYE_POSITION	basicV7.eyePosition	-100 ~ 100	无
外眼角 (V3.9.0)	BEAUTY_EYE_OUT_CORNER	basicV7.eyeOutCorner	-100 ~ 100	无
祛眼袋	BEAUTY_FACE_REMOVE_EYE_BAGS	beauty.removeEyeBags	0 ~ 100	无
眉毛角度	BEAUTY_EYEBROW_ANGLE	basicV7.eyebrowAngle	-100 ~ 100	无
眉毛距离	BEAUTY_EYEBROW_DISTANCE	basicV7.eyebrowDistance	-100 ~ 100	无
眉毛高度	BEAUTY_EYEBROW_HEIGHT	basicV7.eyebrowHeight	-100 ~ 100	无
眉毛长度	BEAUTY_EYEBROW_LENGTH	basicV7.eyebrowLength	-100 ~ 100	无
眉毛粗细	BEAUTY_EYEBROW_THICKNESS	basicV7.eyebrowThickness	-100 ~ 100	无
眉峰	BEAUTY_EYEBROW_RIDGE	basicV7.eyebrowRidge	-100 ~ 100	无
瘦鼻	BEAUTY_NOSE_THIN	basicV7.thinNose	0 ~ 100	无
鼻翼	BEAUTY_NOSE_WING	basicV7.noseWing	-100 ~ 100	无
鼻子位置	BEAUTY_NOSE_HEIGHT	basicV7.noseHeight	-100 ~ 100	无
鼻梁	BEAUTY_NOSE_BRIDGE_WIDTH	basicV7.noseBridgeWidth	-100 ~ 100	无
山根	BEAUTY_NASION	basicV7.nasion	-100 ~ 100	无
白牙	BEAUTY_TOOTH_WHITEN	beauty.toothWhiten	0 ~ 100	无
嘴型	BEAUTY_MOUTH_SIZE	basicV7.mouthSize	-100 ~ 100	无
嘴唇厚度	BEAUTY_MOUTH_HEIGHT	basicV7.mouthHeight	-100 ~ 100	无
嘴唇宽度	BEAUTY_MOUTH_WIDTH	basicV7.mouthWidth	-100 ~ 100	无
嘴唇位置	BEAUTY_MOUTH_POSITION	basicV7.mouthPosition	-100 ~ 100	无
微笑唇	BEAUTY_SMILE_FACE	basicV7.smileFace	-100 ~ 100	无
窄脸	BEAUTY_FACE_THIN	basicV7.thinFace	0 ~ 100	无
瘦脸-自然	BEAUTY_FACE_NATURE	basicV7.natureFace	0 ~ 100	无
瘦脸-女神	BEAUTY_FACE_GODNESS	basicV7.godnessFace	0 ~ 100	无
瘦脸-英俊	BEAUTY_FACE_MALE_GOD	basicV7.maleGodFace	0 ~ 100	无
V脸	BEAUTY_FACE_V	basicV7.vFace	0 ~ 100	无
收下颌	BEAUTY_FACE_JAW	basicV7.faceJaw	0 ~ 100	无
短脸	BEAUTY_FACE_SHORT	basicV7.shortFace	0 ~ 100	无
脸型	BEAUTY_FACE_BASIC	liquefaction.basic3	0 ~ 100	无
下巴	BEAUTY_FACE_THIN_CHIN	basicV7.chin	-100 ~ 100	无

	额头	BEAUTY_FACE_FOREHEAD	basicV7.forehead	-100 ~ 100	无
	额头2 (V3.9.1)	BEAUTY_FACE_FOREHEAD2	basicV7.forehead2	-100 ~ 100	无
	祛皱	BEAUTY_FACE_REMOVE_WRINKLE	beauty.removeWrinkle	0 ~ 100	无
	祛法令纹	BEAUTY_FACE_REMOVE_LAW_LINE	beauty.removeLawLine	0 ~ 100	无
	瘦颧骨	BEAUTY_FACE_THIN_CHEEKBONE	basicV7.cheekboneThin	0 ~ 100	无
单点美妆	口红	BEAUTY_MOUTH_LIPSTICK	beauty.faceFeatureLipsLut	0 ~ 100	口红图片在手机上的绝对路径 或者 相对于美颜模型文件目录的相对路径 示例： `/images/beauty/lips_fuguhong.png`
	腮红	BEAUTY_FACE_RED_CHEEK	beauty.faceFeatureRedCheek	0 ~ 100	示例： `/images/beauty/saihong_jianyue.png`
	立体	BEAUTY_FACE_SOFTLIGHT	beauty.faceFeatureSoftlight	0 ~ 100	示例： `/images/beauty/liti_ziran.png`
	染发 (V3.7.0)	BEAUTY_HAIR_COLOR_LUT	beauty.hairColorLut	0 ~ 100	示例： `/images/hair_color/red.png`
	眼影	BEAUTY_FACE_EYE_SHADOW	beauty.faceFeatureEyesMakeup.eyeShadow	0 ~ 100	示例： `/images/beauty/eyes_makeup_eye_shadow_0-albatross.png`
	眼线	BEAUTY_FACE_EYE_LINER	beauty.faceFeatureEyesMakeup.eyeLiner	0 ~ 100	示例： `/images/beauty/eyes_makeup_eye_liner_0.png`
	睫毛	BEAUTY_FACE_EYELASH	beauty.faceFeatureEyesMakeup.eyelash	0 ~ 100	示例： `/images/beauty/eyes_makeup_eyelash_0.png`
	眉毛	BEAUTY_FACE_EYEBROW	beauty.faceFeatureEyesMakeup.eyebrow	0 ~ 100	示例： `/images/beauty/eyes_makeup_eyebrow_0.png`
	美瞳	BEAUTY_FACE_EYEBALL	beauty.faceFeatureEyesMakeup.eyeball	0 ~ 100	示例： `/images/beauty/eyes_makeup_eyeball_0.png`
	双眼皮 (V3.8.0)	BEAUTY_FACE_MAKEUP_EYELIDS	beauty.faceFeatureEyesMakeup.eyelids	0 ~ 100	示例： `/images/beauty/eyes_makeup_eyelids_kaishan.png`

	卧蚕(V3.8.0)	BEAUTY_FACE_MAKEUP_EYEWOCAN	beauty.faceFeatureEyesMakeup.eyewocan	0 ~ 100	示例：`/images/beauty/eyes_makeup_eye_wocan_keai.png`
美体	一键瘦身	BODY_AUTOTHIN_BODY_STRENGTH	body.autothinBodyStrength	0 ~ 100	无
	长腿	BODY_LEG_STRETCH	body.legStretch	0 ~ 100	无
	瘦腿	BODY_SLIM_LEG_STRENGTH	body.slimLegStrength	0 ~ 100	无
	瘦腰	BODY_WAIST_STRENGTH	body.waistStrength	0 ~ 100	无
	瘦肩	BODY_THIN_SHOULDER_STRENGTH	body.thinShoulderStrength	0 ~ 100	无
	胸部调整	BODY_ENLARGE_CHEST_STRENGTH	body.enlargeChestStrength	-100 ~ 100	无
	小头	BODY_SLIM_HEAD_STRENGTH	body.slimHeadStrength	0 ~ 100	无
	瘦胳膊	BODY_SLIM_ARM_STRENGTH	body.slimArmStrength	-100 ~ 100	无

滤镜、美妆、动效、分割

	effectName		effectValue	resourcePath	extraInfo
	常量名	常量值	效果强度	资源路径	附加参数（键值对类型）
滤镜	EFFECT_LUT	lut	0~100	滤镜图片在手机上的绝对路径，示例：`/data/user/0/xxxxxx/files/xmagic/light_material/lut/aiqing_lf.png` 如果要取消滤镜，这里填null	无
轻美妆（V3.9.0）	EFFECT_LIGHT_MAKEUP	light.makeup	0~100	轻美妆素材在手机上的绝对路径。 如果要取消轻美妆，这里填null	[可选] makeupLutStrength：轻美妆素材中的滤镜强度，取值"0"到"100"
美妆	EFFECT_MAKEUP	makeup	0~100	美妆素材在手机上的绝对路径。 如果要取消美妆，这里填null	• [可选] `makeupLutStrength`：美妆素材中的滤镜强度，取值"0"到"100" • [可选] `mergeWithCurrentMotion`："true"或"false"，表示是否叠加在当前动效上。如果不填写此字段，则认为是 false
动效	EFFECT_MOTION	motion	无	动效素材在手机上的绝对路径，示例：`/data/user/0/xxxxxx/files/xmagic/light_material/motion/video_keaituya` 如果要取消动效，这里填null	• [可选] `mergeWithCurrentMotion`："true"或"false"，表示是否叠加在当前动效上。如果不填写此字段，则认为是 false
背景分割（普通）	EFFECT_SEGMENTATI	segmentation	无	背景分割素材在手机上的绝对路径 如果要取消分割，这里填null	• [可选] `mergeWithCurrentMotion`："true"或"false"，表示是否叠加在当前动效上。如果不填写此字段，则认为是 false

	ON				
背景分割（绿幕）	EFFECT_SEGMENTATION	segmentation	无	背景分割素材在手机上的绝对路径 如果要取消分割，这里填null	• **[必要]** `segType`：**"green_background"** • **[必要]** `bgType`：**自定义背景类型，"0"表示图片或pag，"1"表示视频** • **[可选]** `bgPath`：**自定义背景图片或视频路径** • **[可选]** `keyColor`：**绿幕颜色RGB，格式如"#00ff00"** • **[可选]** `mergeWithCurrentMotion`：**"true"或"false"，表示是否叠加在当前动效上。如果不填写此字段，则认为是 false** **注：** `bgPath` **和** `keyColor` **必须设置一项。**
背景分割（自定义）	EFFECT_SEGMENTATION	segmentation	无	背景分割素材在手机上的绝对路径 如果要取消分割，这里填null	• **[必要]** `segType`：**"custom_background"** • **[必要]** `bgType`：**自定义背景类型，"0"表示图片或者pag，"1"表示视频** • **[必要]** `bgPath`：**自定义背景图片或视频路径** • **[可选]** `mergeWithCurrentMotion`：**"true"或"false"，表示是否叠加在当前动效上。如果不填写此字段，则认为是 false**

美颜场景推荐参数

最近更新时间：2024-06-03 14:56:21

如下是项目 demo 中的一键美颜效果参数表，如果您想在应用中实现一键美颜效果，可以根据如下参数配置对应的美颜效果。

demo 默认效果：

功能类型	参数推荐
美白/自然	40
磨皮	40
清晰度	80
锐化	30
窄脸	5
瘦脸/自然	30
V脸	20
祛法令纹	30
大眼	20
亮眼	40
祛眼袋	50
瘦鼻	20
白牙	40

自然-1

功能类型	参数推荐
美白/自然	40
磨皮	40
清晰度	80
锐化	30
大眼	20
亮眼	40
祛眼袋	50
瘦鼻	20
白牙	40
窄脸	5
瘦脸/自然	30
祛法令纹	30
V脸	20

自然-2

功能类型	参数推荐
美白/自然	30
磨皮	30
对比度	-30
饱和度	-70
清晰度	15
锐化	25
大眼	20
亮眼	50
祛眼袋	5
瘦鼻	10
白牙	10
窄脸	10
瘦脸/自然	20
祛法令纹	30
V脸	10
口红/复古红	30
立体/自然	50

自然-3

功能类型	参数推荐
美白/粉白	40
磨皮	40
清晰度	70
锐化	30
大眼	10
亮眼	40
祛眼袋	50
瘦鼻	20
白牙	40
窄脸	20
祛法令纹	80
瘦颧骨	10
瘦脸/自然	30
V脸	20

口红/温柔粉	20
立体/自然	40

自然-4

功能类型	参数推荐
美白/自然	50
磨皮	20
饱和度	-60
大眼	20
瘦脸/自然	20
口红/蜜桃色	30
立体/自然	30

自然-5

功能类型	参数推荐
美白/自然	30
磨皮	40
红润	20
大眼	20
瘦脸/自然	30

女神-1

功能类型	参数推荐
美白/粉白	50
磨皮	50
清晰度	70
锐化	30
大眼	20
亮眼	40
祛眼袋	80
瘦鼻	20
白牙	40
窄脸	30
祛法令纹	80
瘦颧骨	10
瘦脸/自然	40
V脸	30

收下颌	10
口红/温柔粉	20
立体/自然	50

女神-2

功能类型	参数推荐
美白/粉白	60
磨皮	80
清晰度	70
锐化	30
大眼	35
亮眼	40
祛眼袋	100
瘦鼻	40
白牙	40
窄脸	40
祛法令纹	100
瘦颧骨	10
瘦脸/自然	60
V脸	40
收下颌	10
口红/温柔粉	20
立体/自然	50

女神-3

功能类型	参数推荐
美白/自然	60
磨皮	70
红润	35
大眼	40
瘦脸/自然	40
立体/自然	50
亮眼	30
瘦鼻	10

英俊

功能类型	参数推荐

美白/自然	20
磨皮	30
红润	5
瘦脸/英俊	60
脸型	20
额头	-20
立体/俊朗	60
瘦鼻	60

元气-1

功能类型	参数推荐
美白/自然	30
磨皮	80
红润	5
对比度	-40
饱和度	-30
清晰度	80
锐化	30
大眼	20
亮眼	40
祛眼袋	50
瘦鼻	20
白牙	40
窄脸	5
瘦脸/自然	30
祛法令纹	30
V脸	10

元气-2

功能类型	参数推荐
美白/自然	60
磨皮	65
红润	30
对比度	28
大眼	50
亮眼	60

瘦鼻	35
瘦脸/女神	50
窄脸	25
V脸	40
口红/复古红	30

短视频企业版迁移指引

最近更新时间：2025-02-28 12:24:14

目前，短视频企业版已经下线，其中美颜模块解耦升级成为美颜特效 SDK。美颜特效 SDK 美颜效果更加自然，产品功能更加强大，集成方式更加灵活。本文是短视频企业版升级为美颜特效（美颜特效）的迁移指引。

注意事项

1. 修改 xmagic 模块中的 glide 库的版本号，与实际使用保持一致。
2. 修改 xmagic 模块中的最低版本号，与实际使用保持一致。

集成步骤

步骤一：解压 Demo 工程

1. 下载集成了美颜特效 TE 的 UGSV Demo 工程。本 Demo 基于美颜特效 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：
   - 删除 xmagic 模块中 libs 目录下的 `.aar` 文件，将 SDK 中 libs 目录下的 `.aar` 文件拷贝进 xmagic 模块中 libs 目录下。
   - 删除 xmagic 模块中 assets 目录下的所有文件，将 SDK 中的 `assets/` 目录下的全部资源拷贝到 xmagic 模块 `../src/main/assets` 目录下，如果SDK 包中的 MotionRes 文件夹内有资源，将此文件夹也拷贝到 `../src/main/assets` 目录下 。
   - 删除 xmagic 模块中jniLibs目录下的所有 .so 文件，在 SDK 包内的 jniLibs 中找到对应的 .so 文件（由于 SDK 中 jinLibs 文件夹下的 arm64-v8a 和 armeabi-v7a 的 .so 文件在压缩包中，所以需要先解压），拷贝到 xmagic 模块中的 `../src/main/jniLibs` 目录下。
3. 将 Demo 程中的 xmagic 模块引 到实际项 程中。

步骤二：SDK 版本升级

将 SDK 从 Enterprise 版本升级为 Professional 版本。

- **替换前：** `implementation 'com.tencent.liteav:LiteAVSDK_Enterprise:latest.release'`
- **替换后：** `implementation 'com.tencent.liteav:LiteAVSDK_Professional:latest.release'`

步骤三：设置美颜 License

1. 在项目中的 application 的 oncreate 方法中调用如下方法：

```
XMagicImpl.init(this);
XMagicImpl.checkAuth(null);
```

2. 在 XMagicImpl 类中替换成您申请的美颜特效 License URL 和 Key。

步骤四：代码实现

以小视频录制界面（TCVideoRecordActivity.java）为例。

1. 在 `TCVideoRecordActivity.java` 类中添加如下变量代码：

```
private XMagicImpl mXMagic;
private int isPause = 0;//0 非暂停，1暂停，2暂停中 3.表示要销毁
```

2. 在 `TCVideoRecordActivity.java` 类 onCreate 方法后边添加如下代码：

```
TXUGCRecord instance = TXUGCRecord.getInstance(this);
instance.setVideoProcessListener(new TXUGCRecord.VideoCustomProcessListener() {
  @Override
  public int onTextureCustomProcess(int textureId, int width, int height) {
          if (isPause == 0 && mXMagic != null) {
                  return mXMagic.process(textureId, width, height);
          }
          return 0;
```

```
    }

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

    @Override
    public void onTextureDestroyed() {
            if (Looper.getMainLooper() != Looper.myLooper()) {  //非主线程
                    if (isPause == 1) {
                            isPause = 2;
                            if (mXMagic != null) {
                                    mXMagic.onDestroy();
                            }
                            initXMagic();
                            isPause = 0;
                    } else if (isPause == 3) {
                            if (mXMagic != null) {
                                    mXMagic.onDestroy();
                            }
                    }
            }
    }
});
XMagicImpl.checkAuth((errorCode, msg) -> {
    if (errorCode == TELicenseCheck.ERROR_OK) {
            loadXmagicRes();
    } else {
            TXCLog.e("TAG", "鉴权失败，请检查鉴权url和key" + errorCode + " " + msg);
    }
});
```

3. 在 onStop 方法中添加如下代码：

```
isPause = 1;
if (mXMagic != null) {
    mXMagic.onPause();
}
```

4. 在 onDestroy 方法中添加如下代码：

```
isPause = 3;
XmagicPanelDataManager.getInstance().clearData();
```

5. 在 onActivityResult 方法最前边添加如下代码：

```
if (mXMagic != null) {
    mXMagic.onActivityResult(requestCode, resultCode, data);
}
```

6. 在此类的最后添加如下两个方法：

```
private void loadXmagicRes() {
    if (XMagicImpl.isLoadedRes) {
            XmagicResParser.parseRes(getApplicationContext());
            initXMagic();
            return;
    }
```

```
        new Thread(() -> {
                XmagicResParser.setResPath(new File(getFilesDir(), "xmagic").getAbsolutePath());
                XmagicResParser.copyRes(getApplicationContext());
                XmagicResParser.parseRes(getApplicationContext());
                XMagicImpl.isLoadedRes = true;
                new Handler(Looper.getMainLooper()).post(() -> {
                        initXMagic();
                });
        }).start();

}
/**
* 初始化美颜SDK
*/
private void initXMagic() {
        if (mXMagic == null) {
                mXMagic = new XMagicImpl(this, mUGCKitVideoRecord.getBeautyPanel());
        }else {
                mXMagic.onResume();
        }
}
```

步骤五：对其他类的修改

1. 将 AbsVideoRecordUI 类的 mBeautyPanel 类型修改为 RelativeLayout 类型，getBeautyPanel() 方法返回类型也修改为 RelativeLayout，同时修改对应 XML 中的配置，注掉报错的代码。
2. 注释掉 UGCKitVideoRecord 类中报错的代码。
3. 修改 ScrollFilterView 类中的代码，删除 mBeautyPanel 变量，注释掉报错的代码。

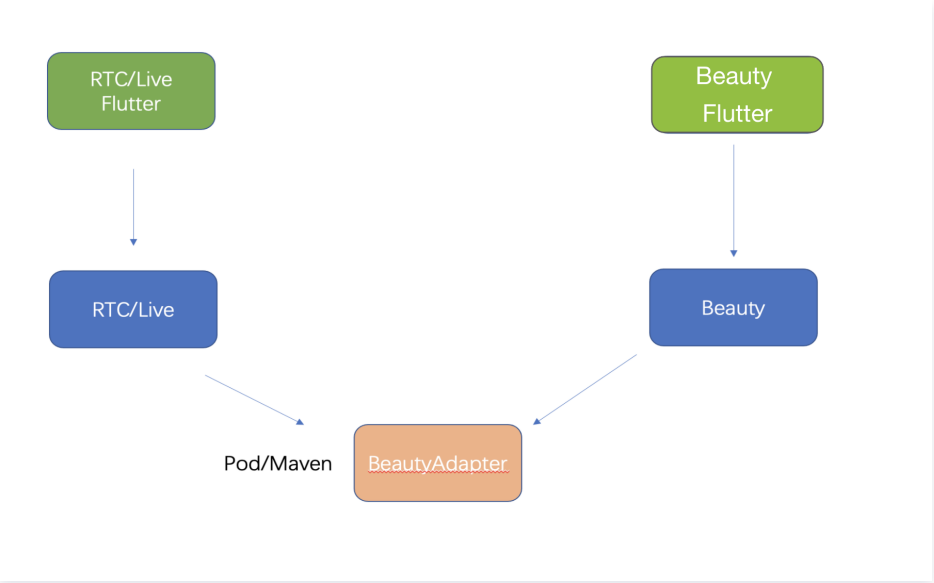
步骤六：删除对 beautysettingkit 模块的依赖

在 ugckit 模块的 build.gradle 文件中删除对 beautysettingkit 模块的依赖，编译项目将报错的代码注释掉即可。

第三方推流接入美颜（Flutter）

最近更新时间：2024-08-06 17:53:41

由于 Flutter 端的 GL 环境与原生端环境进行了隔离，所以 Flutter 中接入美颜时无法直接建立绑定关系，需要在原生端进行关系的绑定，如下图所示：



实现方式总体流程

1. 美颜侧抽象一层接口，并在美颜侧实现了接口。
2. 在应用启动时将此接口注册到三方推流端，这样三方推流端就可以通过此接口进行创建、使用、销毁美颜实例。
3. 第三方推流端再将创建和销毁美颜的能力暴露给自己的 Flutter 端供客户使用。
4. 美颜属性设置可通过美颜提供的 Flutter SDK 能力进行处理。

以 TRTC 为例

美颜侧定义的接口：

```
public interface ITXCustomBeautyProcesserFactory {

    /**
     * 创建美颜实例
     * @return
     */
    ITXCustomBeautyProcesser createCustomBeautyProcesser();

    /**
     * 销毁美颜实例（需要在GL线程调用）
     */
    void destroyCustomBeautyProcesser();
}
public interface ITXCustomBeautyProcesser {

    //获取美颜支持的视频帧的像素格式。美颜支持的是：OpenGL 2D 纹理。
    TXCustomBeautyPixelFormat getSupportedPixelFormat();
    //获取美颜支持的视频数据包装格式。美颜支持的是：V2TXLiveBufferTypeTexture  直接操作纹理 ID，性能最好，画质损失最少。
    TXCustomBeautyBufferType getSupportedBufferType();
    //在GL线程调用（srcFrame中需要包含RGBA纹理，以及width，height），美颜处理之后会将处理后的纹理对象放置在dstFrame中的texture.textureId中。
    void onProcessVideoFrame(TXCustomBeautyVideoFrame srcFrame, TXCustomBeautyVideoFrame dstFrame);
```

```
}
```

1. TRTC提供一个注册的方法，在应用启动时，需将美颜侧 ITXCustomBeautyProcesserFactory 接口的实现类 `com.tencent.effect.tencent_effect_flutter.XmagicProcesserFactory` 注册进 TRTC 中（在原生端进行）。

```java
package com.tencent.effect.tencent_effect_flutter_example;

import android.os.Bundle;

import androidx.annotation.Nullable;

import com.tencent.effect.tencent_effect_flutter.XmagicProcesserFactory;
import com.tencent.live.TXLivePluginManager;   kevinxlhua, 2022/5/9, 3:38 下午 • 添加对 直播flutter版本的依赖，由于现在直

import io.flutter.embedding.android.FlutterActivity;
import com.tencent.trtcplugin.TRTCCloudPlugin;

public class MainActivity extends FlutterActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TXLivePluginManager.register(new XmagicProcesserFactory());
        TRTCCloudPlugin.register(new XmagicProcesserFactory());
    }
}
```

2. 在 `Flutter` 层，提供 `Future<V2TXLiveCode> enableCustomVideoProcess(bool enable)` 接口进行开启或关闭自定义美颜接口。
3. TRTC原生端实现开关美颜方法。

```java
 */
public void enableCustomVideoProcess(MethodCall call, MethodChannel.Result result) {
    boolean enable = CommonUtil.getParam(call, result,  param: "enable");
    ITXCustomBeautyProcesserFactory processerFactory = TRTCCloudPlugin.getBeautyProcesserFactory();
    mCustomBeautyProcesser = processerFactory.createCustomBeautyProcesser();
    TXCustomBeautyBufferType bufferType = mCustomBeautyProcesser.getSupportedBufferType();
    TXCustomBeautyPixelFormat pixelFormat = mCustomBeautyProcesser.getSupportedPixelFormat();
    if(enable) {
        ProcessVideoFrame processVideo =  new ProcessVideoFrame(mCustomBeautyProcesser);
        int ret = trtcCloud.setLocalVideoProcessListener(pixelFormat.convertTRTCPixelFormat(), bufferType.convertTRTCBufferType(), processVideo;
        result.success(ret);
    } else {
        int ret = trtcCloud.setLocalVideoProcessListener(pixelFormat.convertTRTCPixelFormat(), bufferType.convertTRTCBufferType(),  trtcVideoFrameListener: null);
        mCustomBeautyProcesser = null;   clavie, 2022/5/23, 7:49 下午 • 美颜预处理
        result.success(ret);
    }
}
```

当设置为 null 是则会回调上次设置的 processVideo 的 onGlContextDestroy 方法

```java
package com.tencent.trtcplugin.listener;
import ...

public class ProcessVideoFrame implements TRTCCloudListener.TRTCVideoFrameListener {
    private ITXCustomBeautyProcesser    mCustomBeautyProcesser;

    public ProcessVideoFrame(ITXCustomBeautyProcesser processer) {   clavie, 2022/5/23, 7:49 下午 • 美颜预处理
        mCustomBeautyProcesser = processer;
    }
    /** 自定义视频处理回调 ...*/
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame  srcFrame,
                                   TRTCCloudDef.TRTCVideoFrame  dstFrame) {
        TXCustomBeautyVideoFrame srcThirdFrame = new TXCustomBeautyVideoFrame(srcFrame);
        TXCustomBeautyVideoFrame dstThirdFrame = new TXCustomBeautyVideoFrame(dstFrame);
        mCustomBeautyProcesser.onProcessVideoFrame(srcThirdFrame, dstThirdFrame);
        if (dstThirdFrame.texture != null) {
            dstFrame.texture.textureId = dstThirdFrame.texture.textureId;
        }
        dstFrame.data = dstThirdFrame.data;
        dstFrame.buffer = dstThirdFrame.buffer;
        dstFrame.width = dstThirdFrame.width;
        dstFrame.height = dstThirdFrame.height;
        dstFrame.rotation = dstThirdFrame.rotation;
        return 0;
    }

    /** SDK 内部的 OpenGL 环境的创建通知 */
    public void onGLContextCreated() {}

    /**
     * SDK 内部的 OpenGL 环境的销毁通知
     */
    public void onGLContextDestory() {
        ITXCustomBeautyProcesserFactory processerFactory = TRTCCloudPlugin.getBeautyProcesserFactory();
        if (processerFactory != null) {
            processerFactory.destroyCustomBeautyProcesser();
        }
        mCustomBeautyProcesser = null;
    }
}
```

附录

美颜提供的抽象层依赖

```
///
implementation 'com.tencent.liteav:custom-video-processor:latest.release'
```

小程序美颜特效实践

最近更新时间：2024-08-20 14:07:41

准备工作

- 小程序开发入门请参见 微信小程序文档 。
- 请阅读 Web 美颜特效 SDK 接入指南，熟悉 SDK 基本用法。

开始使用

步骤1：小程序后台配置域名白名单

SDK 内部会请求后台进行鉴权和资源加载，因此小程序创建完后，需要在小程序后台配置域名白名单。

1. 打开 小程序后台，进入**开发 > 开发管理 > 开发设置 > 服务器域名**。
2. 单击**修改**，配置以下域名并保存。
   - 请求域名：

```
https://webar.qcloud.com;
https://webar-static.tencent-cloud.com;
https://aegis.qq.com;
以及鉴权签名接口（get-ar-sign）的地址
```

   - downloadFile 域名：

```
https://webar-static.tencent-cloud.com
```

步骤2：安装并构建 npm

小程序 npm 相关请参见 小程序使用 npm 。

1. 安装：

```
npm install tencentcloud-webar
```

2. 构建：
   打开小程序开发者工具，顶部菜单选择**工具 > 构建 npm**。
3. 在 app.json 中配置 workers 路径：

```
"workers": "miniprogram_npm/tencentcloud-webar/worker"
```

步骤3：引入文件

```
// 0.3.0之前版本引用方式（1个文件）
// import "../../miniprogram_npm/tencentcloud-webar/lib.js";
// 0.3.0及之后版本引用方式（2个文件 + 按需初始化3d模块）
import '../../miniprogram_npm/tencentcloud-webar/lib.js';
import '../../miniprogram_npm/tencentcloud-webar/core.js';
// 按需初始化3d插件，如果不需要3d则可以不引用
import '../../miniprogram_npm/tencentcloud-webar/lib-3d.js';
import { plugin3d } from '../../miniprogram_npm/tencentcloud-webar/plugin-3d'
// 导入 ArSdk
import { ArSdk } from "../../miniprogram_npm/tencentcloud-webar/index.js";
```

> ⚠ **注意：**
> - 小程序有单文件不超过 500kb 的限制，因此 SDK 分为两个 js 文件提供。

- 0.3.0版本之后，对 SDK 进行了进一步的拆分，新增3D支持，针对3D模块提供按需加载方式，导入前请确认当前使用的 SDK 版本信息，选择对应的导入方式。

步骤4：初始化 SDK

⚠ 注意：
- 小程序初始化 SDK 前须在控制台配置小程序 APPID，请参见 快速上手 。
- 需在页面中插入 camera 标签来打开相机，然后设置 camera 参数，参数配置详情请参见 接入指南 。
- 小程序不支持 getOutput，需要自行传入一个在屏的 webgl canvas，SDK 直接输出画面到此 canvas 上。

示例代码如下：

```
// wxml
//打开相机，通过position使相机不展示
<camera
  device-position="{{'front'}}"
  frame-size="large" flash="off" resolution="medium"
  style="width: 750rpx; height: 134rpx;position:absolute;top:-9999px;"
/>
//sdk 将处理完的画面实时输出到此 canvas 上
<canvas
  type="webgl"
  canvas-id="main-canvas"
  id="main-canvas"
  style="width: 750rpx; height: 1334rpx;">
</canvas>
//拍照将 ImageData 对象绘制到此 canvas 上
<canvas
  type="2d"
  canvas-id="photo-canvas"
  id="photo-canvas"
  style="position:absolute;width:720px;height:1280px;top:-9999px;left:-9999px;">
</canvas>
// js
/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APPID = ''; // 此处请填写您自己的参数

/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube/web)，创建项目即可获得
LicenseKey
 */
const LICENSE_KEY = ''; // 此处请填写您自己的参数

/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过接口提供，前端调用拉取签名，
参考
 * [签名方法](https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8D.E6.96.B9.E6.B3.95)
 */
```

```
const token = ''; // 此处请填写您自己的参数

Component({
    data: {
        makeupList: [],
        stickerList: [],
        filterList: [],
        recording: false
    },
    methods: {
        async getCanvasNode(id) {
            return new Promise((resolve) => {
                this.createSelectorQuery()
                    .select(`#${id}`)
                    .node()
                    .exec((res) => {
                        const canvasNode = res[0].node;
                        resolve(canvasNode);
                    });
            });
        },
        getSignature() {
            const timestamp = Math.round(new Date().getTime() / 1000);
            const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
            return { signature, timestamp };
        },
        // 初始化相机类型
        async initSdkCamera() {
            // 获取在屏的 canvas，sdk 会将处理完的画面实时输出到 canvas 上
            const outputCanvas = await this.getCanvasNode("main-canvas");
            // 获取鉴权参数
            const auth = {
                licenseKey: LICENSE_KEY,
                appId: APP_ID,
                authFunc: this.getSignature
            };
            // 构造SDK初始化参数
            const config = {
                auth,
                camera: {
                    width:720,
                    height:1280,
                },
                output: outputCanvas,
                // 初始美颜效果（可选）
                beautify: {
                    whiten: 0.1, // 美白 0-1
                    dermabrasion: 0.3, // 磨皮 0-1
                    lift: 0, // 瘦脸 0-1
                    shave: 0, // 削脸 0-1
                    eye: 0.2, // 大眼 0-1
                    chin: 0, // 下巴 0-1
                }
            };
            const ar = new ArSdk(config);
            // created回调里可以开始获取内置特效与滤镜列表
            ar.on('created', () => {
                // 获取内置美妆、贴纸列表
                ar.getEffectList({
                    Type: 'Preset'
                }).then((res) => {
```

```
                const list = res.map(item => ({
                    name: item.Name,
                    id: item.EffectId,
                    cover: item.CoverUrl,
                    url: item.Url,
                    label: item.Label,
                    type: item.PresetType,
                }));
                const makeupList = list.filter(item=>item.label.indexOf('美妆')>=0)
                const stickerList = list.filter(item=>item.label.indexOf('贴纸')>=0)
                // 渲染特效列表
                this.setData({
                    makeupList,
                    stickerList
                });
            }).catch((e) => {
                console.log(e);
            });
            // 内置滤镜
            ar.getCommonFilter().then((res) => {
                const list = res.map(item => ({
                    name: item.Name,
                    id: item.EffectId,
                    cover: item.CoverUrl,
                    url: item.Url,
                    label: item.Label,
                    type: item.PresetType,
                }));
                // 渲染滤镜列表
                this.setData({
                    filterList: list
                });
            }).catch((e) => {
                console.log(e);
            });
        });
        // ready 回调中可以设置美颜、滤镜、特效效果
        ar.on('ready', (e) => {
            this._sdkReady = true
        });

        ar.on('error', (e) => {
            console.log(e);
        });

        this.ar = ar
    },
    // 改变美颜参数，需要确保sdk ready
    onChangeBeauty(val){
        if(!this._sdkReady) return
        // 可以通过setBeautify设置美颜效果，支持6种属性，详见SDK接入指南
        this.ar.setBeautify({
            dermabrasion: val.dermabrasion, // 磨皮 0-1
        });
    },
    // 改变美妆，需要确保sdk ready
    onChangeMakeup(id, intensity){
        if(!this._sdkReady) return
        // 使用setEffect设置特效，setEffect的输入参数支持三种格式，详见SDK接入指南
        this.ar.setEffect([{id, intensity}]);
    },
```

```
        // 改变贴纸，需要确保sdk ready
        onChangeSticker(id, intensity){
            if(!this._sdkReady) return
            // 使用setEffect设置特效，setEffect的输入参数支持三种格式，详见SDK接入指南
            this.ar.setEffect([{id, intensity}]);
        },
        // 改变滤镜，需要确保sdk ready
        onChangeFilter(id, intensity){
            if(!this._sdkReady) return
            // 使用setFilter设置滤镜，第二个参数表示滤镜强度（范围0-1）
            this.ar.setFilter(id, 1);
        }
    }
})
```

步骤5：拍照和录像功能实现

示例代码如下：

拍照

SDK 会返回包含宽高和 buffer 数据的对象，用户可以通过自己页面内预设的 2d canvas（上述代码中id为photo-canvas）绘制此数据并导出为图片文件。

```
async takePhoto() {
    const {uint8ArrayData, width, height} = this.ar.takePhoto(); // takePhoto 方法返回当前画面的 buffer
数据
    const photoCanvasNode = await this.getCanvasNode('photo-canvas');
    photoCanvasNode.width = parseInt(width);
    photoCanvasNode.height = parseInt(height);
    const ctx = photoCanvasNode.getContext('2d');
    // 用 sdk 返回的数据创建 ImageData 对象
    const imageData = photoCanvasNode.createImageData(uint8ArrayData, width, height);
    // 将 ImageData 对象绘制到 canvas 上
    ctx.putImageData(imageData,0,0,0,0,width,height);
    // 将 canvas 保存为本地图片
    wx.canvasToTempFilePath({
        canvas: photoCanvasNode,
        x: 0,
        y: 0,
        width: width,
        height: height,
        destWidth: width,
        destHeight: height,
        success: (res) => {
            // 保存照片到本地
            wx.saveImageToPhotosAlbum({
                filePath: res.tempFilePath
            });
        }
    })
}
```

录像

```
Component({
```

```
    methods: {
        // 开始录像
        startRecord() {
            this.setData({
                recording: true
            });
            this.ar.startRecord()
        }
        // 结束录像
        async stopRecord() {
            const res = await this.ar.stopRecord();
            // 保存录像到本地
            wx.saveVideoToPhotosAlbum({
                filePath: res.tempFilePath
            });
            this.setData({
                recording: false
            });
        }
    }
})
```

当小程序切换后台或检测到手机锁屏时，需要调用 stopRecord 停止录像，再次回到页面时重新开启SDK即可。

```
onShow() {
    this.ar && this.ar.start();
},
onHide() {
    this.ar && this.ar.stop();
},
async onUnload() {
    try {
        this.ar && this.ar.stop();
        if (this.data.recording) {
            await this.ar.stopRecord({
                destroy: true,
            });
        }
    } catch (e) {
    }
    this.ar && this.ar.destroy();
}
```

代码示例

您可以下载 示例代码 解压后查看 `ar-miniprogram` 代码目录。

礼物动画特效 Android 集成指引

最近更新时间：2025-02-28 12:24:15

本文主要介绍如何快速地将腾讯礼物动画特效 SDK 集成到您的项目中，按照如下步骤进行配置，就可以完成 SDK 的集成工作。

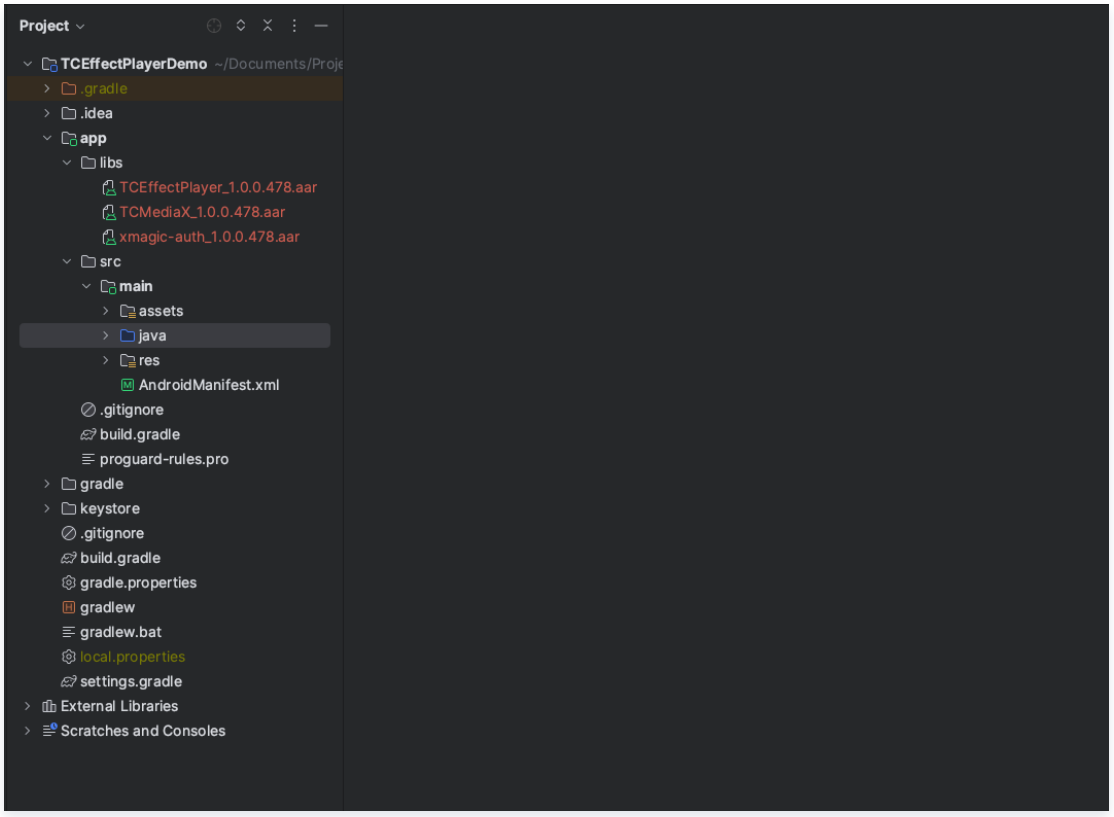
开发环境要求

- Android Studio 2.0+。
- Android 4.4（SDK API 19）及以上系统。

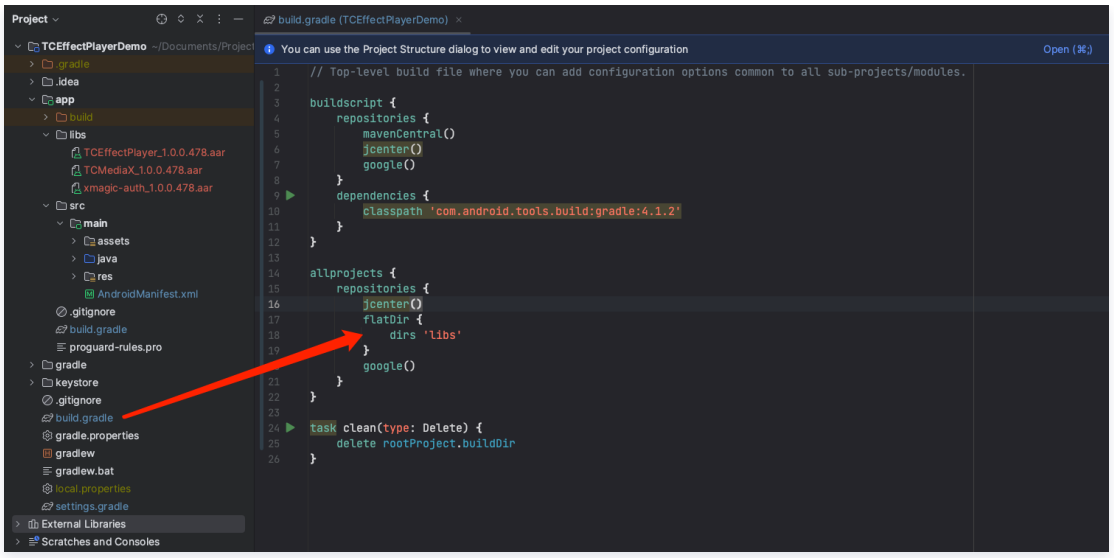
集成 SDK

手动集成

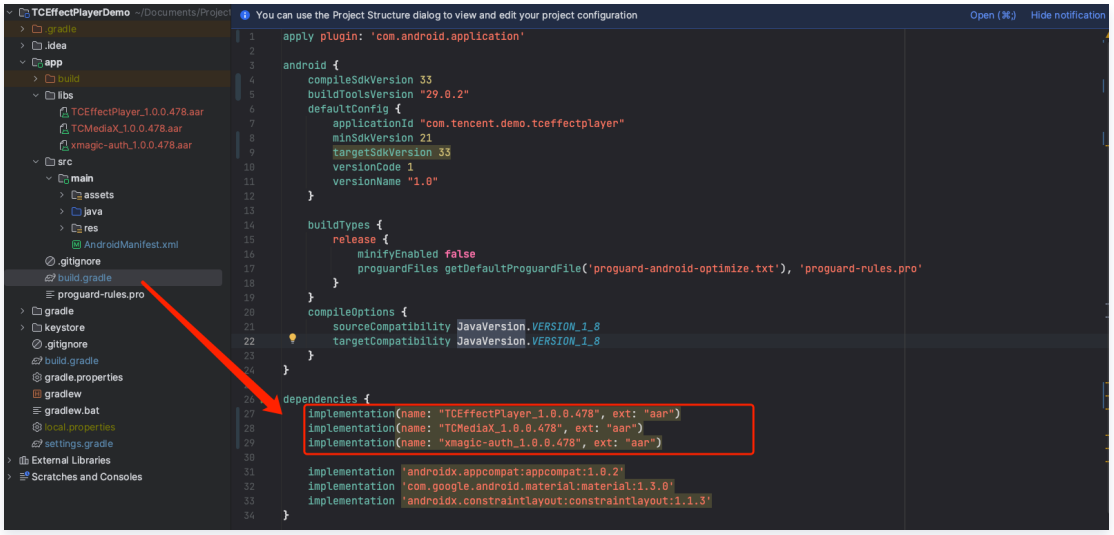
1. 下载 MediaX_Android_SDK_Latest.zip ，下载完成后进行解压获取到 SDK aar 文件。
2. 将下载文件解压之后 SDK 目录下的 aar 文件拷贝到工程的 app/libs 目录下：



3. 在工程根目录下的 build.gradle 中，添加 flatDir，指定本地仓库路径。



4. 添加 腾讯礼物动画特效SDK 依赖，在 app/build.gradle 中，添加引用 aar 包的代码。




```
implementation(name: "TCEffectPlayer_x.x.x", ext: "aar")
implementation(name: "TCMediaX_x.x.x", ext: "aar")
implementation(name: "xmagic-auth_x.x.x", ext: "aar")
```

> ⚠ **注意：**
> 如上示例中的 x.x.x 为版本号，具体以实际下载的SDK最新版本号为准；另外需要注意3个 aar 的版本号必须一致。

5. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构（目前支持 armeabi 、armeabi-v7a 和 arm64-v8a ）。

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

Gradle 方式集成

1. 打开对应的 build.gradle 文件，在 dependencies 标签内加上 腾讯礼物动画特效 SDK 依赖项：

```
dependencies {
    // 如果要集成历史版本，可将 latest.release 修改为对应的版本，例如：3.0.0.248
```

```
    implementation "com.tencent.mediacloud:TCEffectPlayer:latest.release"
}
```

2. 在 app/build.gradle 的 defaultConfig 中，指定 App 使用的 CPU 架构（目前支持 armeabi 、armeabi-v7a 和 arm64-v8a）。

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

最后，单击 Sync Now 同步 SDK，完成腾讯礼物动画特效 SDK 的集成工作。

设置混淆规则

在 proguard-rules.pro 文件中，将相关类加入不混淆名单：

```
-keep class com.tcmediax.** { *; }
-keep class com.tencent.** { *; }
-keep class com.tencent.xmagic.** { *; }
# 腾讯礼物动画特效SDK会通过 exifinterface 解析动效资源元数据，如果项目中引入了exifinterface， 必须加入下面的混淆配置
-keep class androidx.exifinterface.** {*;}
```

软解能力集成指引

优势

由于 Android 系统碎片化严重，在极个别手机上会出现无法避免的硬解失败问题，因此需要借助软解能力来提升播放成功率（成功率可提升至99.99%），因此强烈建议您同时接入软解库，以达到更好的特效播放体验。

集成步骤

已集成 LiteAVSDK_TRTC SDK

如果您之前的项目中已集成视立方相关 SDK，如：LiteAVSDK_TRTC、LiteAVSDK_Live 等。则需要集成腾讯云播放器精简版 SDK（LiteAVSDK_Player_Mini），把对应 LiteAVSDK_Player_Mini.aar 放到工程 libs 目录，然后在 dependencies 中添加对应依赖：

```
dependencies {
   implementation (name:'LiteAVSDK_Player_Mini', ext:'aar')
}
```

已集成 LiteAVSDK_Professional 相关 SDK

如果您之前的项目中已集成视立方相关SDK，如：LiteAVSDK_Player、LiteAVSDK_Player_Premium、LiteAVSDK_Professional、LiteAVSDK_UGC 等。则软解能力会自动支持，无需额外操作。

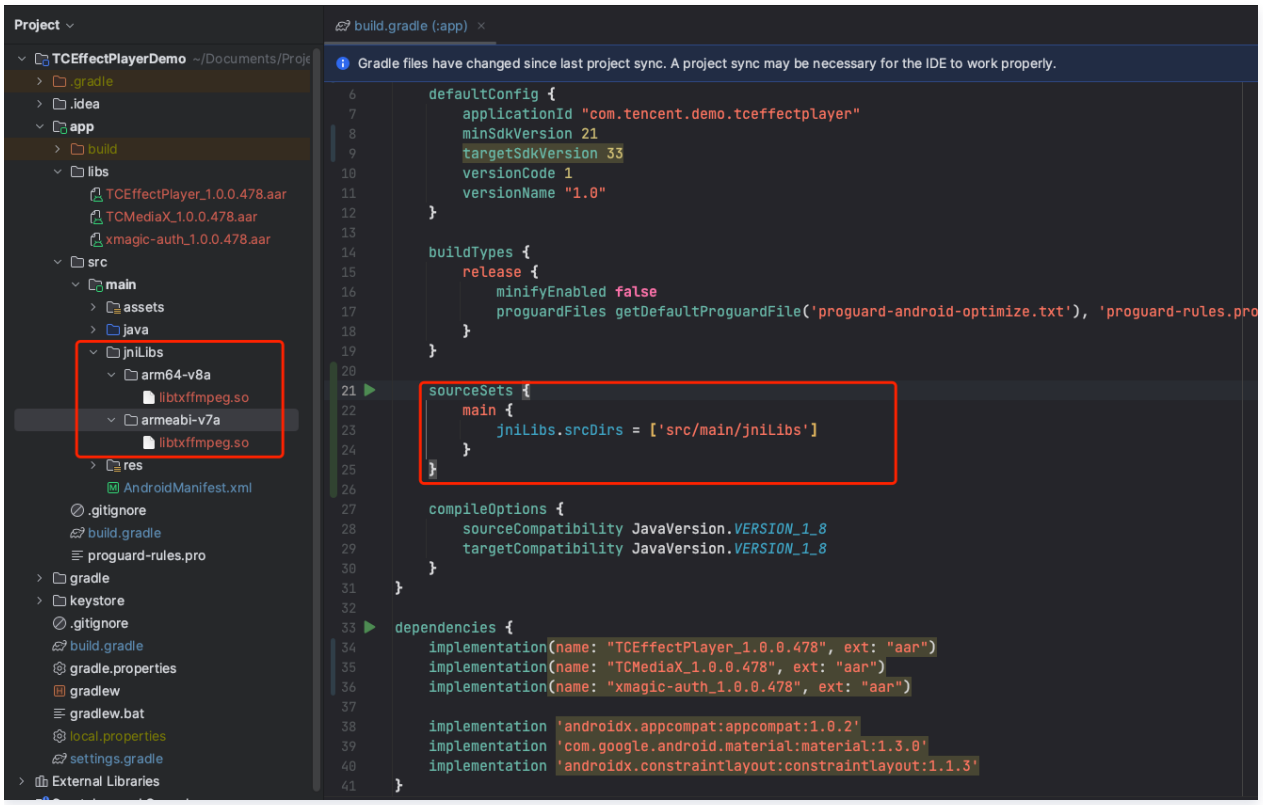
> ⚠ **注意：**
> 如果您的项目中已经集成了腾讯云视立方 SDK， 请检查是否包含播放器 SDK ，同时检查是否已经申请和配置播放器 License。

未集成视立方相关 SDK

如果您之前的项目中未集成视立方相关 SDK，则建议集成腾讯云播放器精简版 SDK（LiteAVSDK_Player_Mini）。将对应的 LiteAVSDK_Player_Mini.aar 文件放到工程的 libs 目录下，然后在 dependencies 中添加相应依赖：

```
dependencies {
   implementation (name:'LiteAVSDK_Player_Mini', ext:'aar')
}
```

最后一步，需要配置 so 文件：将 jniLibs_txffmpeg_so.zip 链接中 so 文件下载到本地之后解压，把解压后的 jniLibs 目录中的 so 文件复制到 app 的 src/main 目录中，并且在 app/build.gradle 中配置对应路径：




```
sourceSets {
    main {
        jniLibs.srcDirs = ['src/main/jniLibs']
    }
}
```

申请 License

使用腾讯礼物动画特效 SDK 需要申请对应的 License，详细申请流程请参见 控制台 License 申请指南。

常见问题

xmagic 冲突如何解决?

如果您在项目中已经接入了腾讯美颜 SDK，那么在接入礼物动画特效 SDK 时，会出现类冲突、so 文件冲突的问题，此时您需要移除掉：xmagic-auth-x.x.x.aar 依赖的引入。

如何处理加密视频配置?

如果您在使用 TepTools 工具转换动画时，勾选了：**Anim Encrypt** 开关，那么转换出来的动画则是加密的，此时在使用特效播放器播放时，如果您的应用 targetSDKVersion 大于或等于28，那么需要单独进行如下配置：

1. 在项目新建 res/xml/network_security_config.xml 文件，设置网络安全性配置：

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">localhost</domain>
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
</network-security-config>
```

2. 在 AndroidManifest.xml 文件下的 application 标签增加以下属性：

```
 <?xml version="1.0" encoding="utf-8"?>
<manifest ... >
   <application android:networkSecurityConfig="@xml/network_security_config"
                ... >
    ...
   </application>
</manifest>
```

使用文档

最近更新时间：2025-02-28 12:24:14

本文将详细介绍如何在项目中快速地使用腾讯礼物动画特效 SDK。只需遵循以下步骤，即可完成 SDK 的配置和使用。

初始化注册 License

在正式使用腾讯礼物动画特效 SDK 时，需要先设置 License，设置 License 成功之后，才可以进行后续的 SDK 使用。
设置 License 方式如下：

```
private static final String sLicenseUrl = "${licenseUrl}";
private static final String sLicenseKey = "${licenseKey}";
private int mRetryCount = 0;
private static final int MAX_RETRY_COUNT = 3;
private final ILicenseCallback mLicenseCallback = new ILicenseCallback() {
    @Override
    public void onResult(int errCode, String msg) {
        Log.d(TAG, "TCMediaX license result: errCode: " + errCode + ", msg: " + msg);
        if (errCode == TXLicenceErrorCode.LicenseCheckDownloadAndAssetBothFailed) {
            if(++mRetryCount <= MAX_RETRY_COUNT) {
                new Thread(() -> TCMediaXBase.getInstance().setLicense(DemoApplication.getAppContext(),
sLicenseUrl, sLicenseKey, mLicenseCallback)).start();
            } else {
                // 此时网络不可用，发送消息提示用户检测网络
            }
        }
    }
};

// 调用当前方法进行license设置
public void init() {
    TCMediaXBase.getInstance().setLicense(DemoApplication.getAppContext(), sLicenseUrl, sLicenseKey,
mLicenseCallback);
}
```

注意：

- License 是强线上检验逻辑，应用首次启动后调用 TCMediaXBase#setLicense 时，需确保网络可用。 在 App 首次启动时，可能还没有授权联网权限，则需要等授予联网权限后，再次调用 TCMediaXBase#setLicense。
- 监听 TCMediaXBase#setLicence 加载结果：ILicenseCallback#onResult 接口，如果失败要根据实际情况做对应重试及引导，如果多次失败后，可以限频，并业务辅以产品弹窗等引导，让用户检查网络情况。
- 对于多进程的 App， 确保每个使用特效播放器的进程启动时，都调用了 TCMediaXBase#setLicense。例如： Android 端使用独立进程播放特效的 App，后台播放时进程被系统 kill 掉重启时，也要调用 TCMediaXBase#setLicense。

日志管理

腾讯礼物动画特效 SDK 默认支持保存运行日志，如果测试过程中出现问题，可以提取日志反馈给腾讯云客服，您可以根据业务需要把此目录的日志上传到业务后台，用于定位线上用户问题。
腾讯礼物动画特效 SDK Android 端的日志保存在目录：`/sdcard/Android/data/${your_packagename}/files/TCMediaLog`，日志文件按照日期命名，把 TCMediaLog 文件夹导出即可：

> **注意：**
> **没有日志文件?**
> 请检查是否通过 TCMediaXBase#setLogEnable 传入 false 关闭了日志，默认文件日志是开启的。

播放器使用

把 TCEffectAnimView 加入布局里

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="0dp">
    <com.tencent.tcmediax.tceffectplayer.api.TCEffectAnimView
        android:id="@+id/video_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_centerInParent="true"
        android:visibility="visible" />
</FrameLayout>
```

初始化 TCEffectAnimView

```
private TCEffectAnimView mPlayerView;
mPlayerView = (TCEffectAnimView) findViewById(R.id.video_view);

// 可选：设置动画对齐方式（默认FIT_CENTER，支持自定义）
mPlayerView.setScaleType(TCEffectPlayerConstant.ScaleType.FIT_CENTER);
```

播放监听

可以在开始播放之前通过调用 `setPlayListener` 方法来设置动画播放状态监听：

```
// 设置播放状态回调
playerView.setPlayListener(new TCEffectAnimView.IAnimPlayListener() {
    @Override
    public void onPlayStart() {
        // 动画开始播放
    }

    @Override
    public void onPlayEnd() {
        // 动画结束播放
    }

    @Override
    public void onPlayError(int errorCode) {
```

```
        // 动画播放失败
    }

    @Override
    public void onPlayEvent(int event, Bundle param) {
        // 动画播放事件
    }
});
```

1. 如果您需要获取当前正在播放的动画信息，可以在 `onPlayStart()` 方法中(或者该方法执行之后)调用 `getTCAnimInfo()` 方法来获取到 `TCEffectAnimInfo` 对象实例，进而获取到当前播放动画的类型、时长、宽高、融合动画等信息。
2. `onPlayEvent()` 方法会回调动画播放中的事件，回调线程不保证都在主线程中。

播放配置

```
// 设置播放配置，非必需的步骤
TCEffectConfig config = new
TCEffectConfig.Builder().setCodecType(TCEffectConfig.CodecType.TC_MPLAYER).build();
mPlayerView.setConfig(config);
```

TCEffectConfig 中目前支持：

1. setCodecType(CodecType type)：它有三个取值，分别是：
   - TCEffectConfig.CodecType.TC_MPLAYER (MPLAYER播放引擎)
   - TCEffectConfig.CodecType.TC_MCODEC(MCODEC播放引擎)
   - TCEffectConfig.CodecType.TX_LITEAV_SDK (腾讯云播放器SDK)

> ⚠ **注意：**
> 1. 目前仅支持在播放器开始前调用 setConfig() 方法来设置播放配置，开始播放后不支持修改配置。
> 2. 目前支持的三种 CodecType 仅对 TEP 动画生效
> 3. 如果设置 CodecType 为 TCEffectConfig.CodecType.TX_LITEAV_SDK ，则还需要单独引入腾讯云播放器 SDK，以及申请、注册好其对应的license。
> 4. 如果不设置 config，则会默认使用 CodecType = TCEffectConfig.CodecType.TC_MPLAYER

2. setFreezeFrame(int frame)：用于设置播放动画冻结帧，详细解释见 API 文档 。
3. setAnimType(AnimType type)：用于指定后续要播放的动画格式，适用于某些情况下，要播放的动画文件后缀被修改的场景。可取值如下：
   - TCEffectConfig.AnimType.AUTO（sdk 默认策略，即以动画文件的后缀来判断动画格式，比如 tcmp4/mp4/tep/tepg 等格式）
   - TCEffectConfig.AnimType.MP4（MP4 动画格式，后续都将动画文件当做 MP4 类型来播放，无视文件后缀名）
   - TCEffectConfig.AnimType.TCMP4（TCMP4 动画格式，后续都将动画文件当做 TCMP4 类型来播放，无视文件后缀名）

融合动画配置

实现 mp4 或者 tcmp4 融合动画的播放，需要实现 IFetchResource 接口。

1. 如果是图片类型的融合动画，需要返回对应的 Bitmap 来替换对应的元素；
2. 如果是文字类型的融合动画，需要返回对应的 TCEffectText 对象实例，来指定文字的显示配置信息。

```
mPlayerView.setFetchResource(new IFetchResource() {
    @Override
    public void fetchImage(Resource resource, IFetchResourceImgResult result) {
        // 如果替换的是 tepg 文件资源的话，isTEPG() = true
        boolean isTEPG = resource.isTEPG();
        // 如果原始文件是pag的话，这里的 id 是原始pag文件中图片图层对应的index,如 0、1、2.....
        String index = resource.id;
        // 后续可以根据图层 index 的值，来选择您要替换返回的图片 Bitmap 资源
        // 如果想跳过指定图层不替换的话，可以返回 null
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inScaled = false;
```

```
        Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.test, options);
        // 回调结果出去
        result.fetch(bitmap);
    }
    @Override
    public void fetchText(Resource resource, IFetchResourceTxtResult result) {
        // 如果替换的是 tepg 文件资源的话，isTEPG() = true
        boolean isTEPG = resource.isTEPG();
        // 如果原始文件是pag的话，这里的 id 是原始pag文件中文字图层对应的index,如 0、1、2.....
        String index = resource.id;
        // 根据图层 index 的值，来选择您要替换返回的文字 String 即可
        TCEffectText tcEffectText = new TCEffectText();
        tcEffectText.text = "test";
//      tcEffectText.color = Color.YELLOW;
//      tcEffectText.alignment = TCEffectText.TEXT_ALIGNMENT_RIGHT;
        tcEffectText.fontStyle = "bold";
        result.loadTextForPlayer(tcEffectText);
    }
    @Override
    public void releaseResource(List<Resource> resources) {
        // 资源释放通知
        for (Resource resource : resources) {
            if (resource.bitmap != null) resource.bitmap.recycle();
        }
    }
});
```

如果需要监听融合动画自定义资源的点击事件，需要注册 OnResourceClickListener：

```
mPlayerView.setOnResourceClickListener(new OnResourceClickListener() {
    @Override
    public void onClick(Resource resource) {
        // 根据点击的资源类型来做展示处理
        if (resource.srcType.equals(Resource.SrcType.TXT)) {
            // 如果是 TXT 类型，则 id 字段是其图层id，text是其当时被替换的文本
            tvTest.setText(resource.text);
        } else {
            // 如果是 IMG 类型，则 id 字段是其图层id，bitmap是其当时被替换的图片
            ivTest.setImageBitmap(resource.bitmap);
        }
    }
});
```

播放控制

开始播放

TCEffectPlayer 支持播放 .mp4、.tcmp4 文件格式资源。

> **注意：**
> 只支持播放 sdcard 本地视频资源， 如果您使用的网络视频资源， 先下载到本地再播放。

```
String localPath = /sdcard/Android/${packageName}/files/tep_cool_ss.tep
mPlayerView.startPlay(localPath)
```

暂停播放

```
mPlayerView.pause()
```

继续播放

```
mPlayerView.resume()
```

停止播放

当不需要继续使用播放器时，需要停止播放，释放占用的资源。

```
mPlayerView.stopPlay(true)
```

常见问题

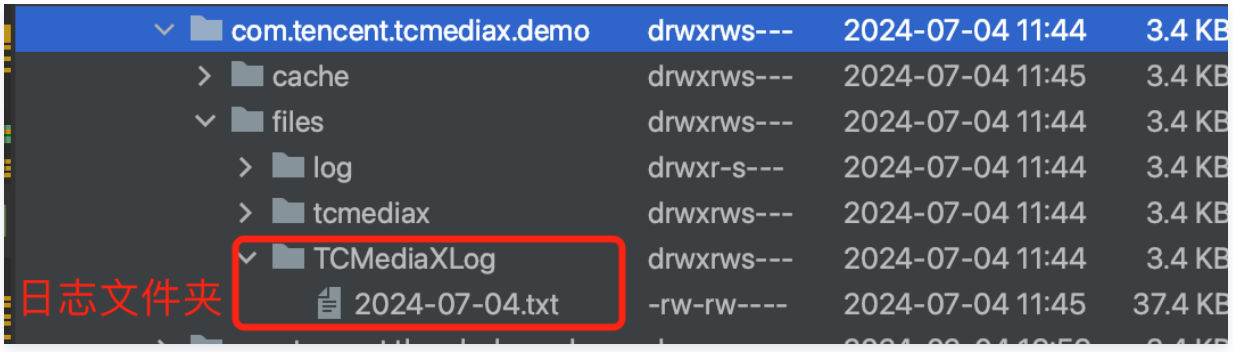
在播放过程中出现下面的日志信息，如何处理?

```
License checked failed! tceffect player license required!
```

请检查是否申请特效播放器 License，并进行了初始化注册。

日志文件如何提取?

1. 方法一：特效播放器的日志保存在目录： /sdcard/Android/data/${your_packagename}/files/TCMediaLog ， 日志文件按照日期命名，把 TCMediaLog 文件夹导出，压缩成 zip。



2. 方法二：如果有开发环境，可以通过 adb 运行下面的命令抓取全量日志：

```
adb logcat -v time > log.txt
```

常见错误码

错误码	含义
0	成功
-1	输入参数无效
-4	从本地读取的 TE 授权信息为空
-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	证书解密失败，可能使用了错误的证书、证书版本太旧或证书文件被破坏
3015	Bundle ID 不匹配，请检查 Bundle ID 是否和证书授权的一致。（iOS、Mac为Bundle ID、安卓为package name、windows为进程名）
3018	证书已过期, 请续期
3024	secret_key 和证书不匹配，请传入正确的 secret_key

API 文档

最近更新时间：2025-02-28 12:24:15

本文主要介绍腾讯礼物动画特效 SDK 的 Android 端 API 接口文档，以便于查阅和使用。

TCMediaXBase

getInstance

说明

获取 TCMediaXBase 的单例。

接口

```
public static TCMediaXBase getInstance()
```

参数说明

无

setLicense

说明

设置 License

接口

```
public void setLicense(Context context, String url, String key, ILicenseCallback callback)
```

参数说明

参数名	类型	描述
context	Context	应用的 Context
url	String	licence 的 url
key	String	licence 的 key
callback	ILicenseCallback	回调

回调声明：

```
public interface ILicenseCallback {
    void onResult(final int errCode, final String msg);
}
```

License 校验常见错误码：

错误码	含义
0	成功
-1	输入参数无效
-4	从本地读取的TE授权信息为空
-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	证书解密失败，可能使用了错误的证书、证书版本太旧或证书文件被破坏

3015	Bundle ID不匹配，请检查Bundle ID是否和证书授权的一致。（iOS、Mac为Bundle ID、安卓为package name、windows为进程名）
3018	证书已过期, 请续期
3024	secret_key和证书不匹配，请传入正确的secret_key

setLogEnable

说明

是否开启 Log 输出，默认开始。

> ⚠ **注意：**
> Android 端保存在 /sdcard/Android/data/packagename/files/TCMediaX 目录，您可以根据业务需要把此目录的日志上传到业务后台，用于定位线上用户问题。

接口

```
public void setLogEnable(Context context, boolean enable)
```

TCEffectAnimView

startPlay

说明

启动播放器。

接口

```
public int startPlay(String playUrl)
```

参数说明

playUrl 为视频资源地址。

> ⚠ **注意：**
> 只支持播放 sdcard 本地视频资源， 如果您使用的网络视频资源， 先下载到本地再播放。

setVideoMode

说明

设置 tep 动画的 alpha 和 rgb 区域的对齐方式。

接口

```
public void setVideoMode(TCEffectPlayerConstant.VideoMode mode)
```

参数说明

mode 支持以下格式：

枚举值	含义
TCEffectPlayerConstant.VideoMode#VIDEO_MODE_NONE	普通mp4文件
TCEffectPlayerConstant.VideoMode#VIDEO_MODE_SPLIT_HORIZONTAL	左右对齐（alpha左\rgb右）
TCEffectPlayerConstant.VideoMode#VIDEO_MODE_SPLIT_VERTICAL	上下对齐（alpha上\rgb下）
TCEffectPlayerConstant.VideoMode#VIDEO_MODE_SPLIT_HORIZONTAL_REVERSE	左右对齐（rgb左\alpha右）

TCEffectPlayerConstant.VideoMode#VIDEO_MODE_SPLIT_VERTICAL_REVERSE	上下对齐（rgb上\alpha下）

setConfig

说明

设置特效播放器参数，需要在启动播放前调用。

接口

```
public void setConfig(TCEffectConfig config)
```

参数说明

参考下面 TCEffectConfig 类。

setScaleType

说明

设置对齐方式

接口

```
public void setScaleType(TCEffectPlayerConstant.ScaleType type)
```

参数说明

type 支持以下格式：

枚举值	含义
TCEffectPlayerConstant.ScaleType.FIT_XY	完整填充整个布局，默认值
TCEffectPlayerConstant.ScaleType.FIT_CENTER	按视频比例在布局中间完整显示
TCEffectPlayerConstant.ScaleType.CENTER_CROP	按视频比例完整填充布局（多余部分不显示）

setFetchResource

说明

设置融合动画的资源

接口

```
public void setFetchResource(IFetchResource fetchResource)
```

参数说明

IFetchResource 接口：

```
public interface IFetchResource {
    // 获取图片
    void fetchImage(Resource resource, IFetchResourceImgResult result);

    // 获取文字
    void fetchText(Resource resource, IFetchResourceTxtResult result);

    // 资源释放通知
    void releaseResource(List<Resource> resources);
}
```

resource.tag 为融合动画的 tag（下标），可以判断 tag 传入不同的文字或图片资源。

setOnResourceClickListener

说明
设置融合动画的资源点击事件。
接口

```
public void setOnResourceClickListener(OnResourceClickListener listener)
```

参数说明
OnResourceClickListener 接口：

```
public interface OnResourceClickListener {

    void onClick(Resource resource);
}
```

根据 resource.tag 区分融合动画的资源。

setRenderRotation

说明
设置融合动画选择旋转角度，支持 0， 90，180, 270， 360 度。

接口

```
public void setRenderRotation(int rotation)
```

isPlaying

说明
返回特效播放器是否在播放中
接口

```
public boolean isPlaying()
```

resume

说明
恢复特效动画播放
接口

```
public void resume()
```

pause

说明
暂停特效动画播放
接口

```
public void pause()
```

seekTo

说明
跳转到指定位置开始播放

> ⚠ **注意：**
> 1. startPlay() 之后才可以调用该方法，否则不生效。

2. 该接口对于 tcmp4 动画、或者设置了 TCEffectConfig.CodecType.TX_LITEAV_SDK 时播放的mp4动画生效。

milliSec：要跳转到指定时长处开始播放，单位毫秒。

接口

```
public void seekTo(long milliSec)
```

seekProgress

说明

跳转到指定位置开始播放

progress：跳转到动画时长的指定百分比处开始播放，单位百分比，取值范围为：[0.0-1.0]

注意：

1. startPlay() 之后才可以调用该方法，否则不生效。
2. 该接口对于 tcmp4 动画、或者设置了 TCEffectConfig.CodecType.TX_LITEAV_SDK 时播放的mp4动画生效。
3. 入参 progress 取值范围为 [0.0-1.0]，超出范围的不生效。

接口

```
public void seekProgress(float progress)
```

setLoop

说明

设置循环播放。

true： 表示循环播放。

false： 表示关闭循环播放。

接口

```
public void setLoop(boolean isLoop)
```

setLoopCount

说明

设置循环播放次数。

loopCount：表示循环播放次数。当 loopCount <= 0 时，表示无限循环播放；当 loopCount=n(n>=1)时表示从开始播放到播放结束，共播放n次

注意：

1. loopCount 默认值是1，即当外部不主动调用该方法时，动画只播放一次就结束。
2. setLoop(boolean isLoop) 方法内部会调用当前方法，即当 isLoop = true 时，等价于调用 setLoopCount(-1)；即当 isLoop = false 时，等价于调用 setLoopCount(1)；因此这两个方法是互相影响的，后调用的会覆盖之前的调用。

接口

```
public void setLoopCount(int loopCount)
```

setDuration

说明

设置动画需要多长时间播放完成。设置之后，后续动画播放时自动调整动画播放速度，以保证动画在设置的规定时长时播放结束。

即：设置的时长超过动画原时长，则动画慢放；小于动画原时长，则快进播放。

durationInMilliSec： 要设置的时长，单位毫秒。

注意：

1. 目前仅对 tcmp4 格式的动画生效。
2. 当前方法和 setRate 设置倍速的方法是互斥的，后调用的会覆盖掉先调用的。

接口

```
public void setDuration(long durationInMilliSec)
```

stopPlay

说明
停止播放。
接口

```
public void stopPlay(boolean clearLastFrame)
```

setMute

说明
设置是否静音播放。
true： 静音播放。
false： 非静音播放。
接口

```
public void setMute(boolean mute)
```

setPlayListener

说明
设置特效播放器播放回调。
接口

```
public void setPlayListener(IAnimPlayListener listener)
```

参数说明
IAnimPlayListener 接口：

```
public interface IAnimPlayListener {
    void onPlayStart();

    void onPlayEnd();

    void onPlayError(int errorCode);

    void onPlayEvent(int event, Bundle param);
}
```

对于 onPlayEvent 方法中的 event 值详见 TCEffectPlayerConstant 类中的常量值：

```
public static final int REPORT_INFO_ON_PLAY_EVT_PLAY_END = 2006;
public static final int REPORT_INFO_ON_PLAY_EVT_RCV_FIRST_I_FRAME = 2003;
public static final int REPORT_INFO_ON_PLAY_EVT_CHANGE_RESOLUTION = 2009;
public static final int REPORT_INFO_ON_PLAY_EVT_LOOP_ONCE_COMPLETE = 6001;
public static final int REPORT_INFO_ON_VIDEO_CONFIG_READY = 200001;
public static final int REPORT_INFO_ON_NEED_SURFACE = 200002;
public static final int REPORT_INFO_ON_VIDEO_SIZE_CHANGE = 200003;
```

```
public static final int REPORT_ANIM_INFO = 200004;
```

对于 onPlayError 方法中的 errorCode 值详见 TCEffectPlayerConstant 类中的常量值：

```
public static final int REPORT_ERROR_TYPE_HEVC_NOT_SUPPORT = -10007; // 不支持h265
public static final int REPORT_ERROR_TYPE_INVALID_PARAM = -10008; // 参数非法
public static final int REPORT_ERROR_TYPE_INVALID_LICENSE = -10009; // License 不合法
public static final int REPORT_ERROR_TYPE_ADVANCE_MEDIA_PLAYER = -10010; // MediaPlayer播放失败
public static final int REPORT_ERROR_TYPE_MC_DECODER = -10011; // MediaCodec 中 Decoder 失败
public static final int REPORT_ERROR_TYPE_UNKNOWN_ERROR = -20000; // 未知错误
```

getTCAnimInfo

说明

获取当前播放动画对应的信息，返回 TCEffectAnimInfo 实例，详见 TCEffectAnimInfo 。

> **注意：**
> 该方法必须在 IAnimPlayListener#onPlayStart() 方法中，或者该方法执行之后调用才可以获取到当前动画的信息，否则返回 null。

接口

```
public TCEffectAnimInfo getTCAnimInfo()
```

getTCEffectPlayer

说明

获取 TCEffectPlayer 实例。

接口

```
public TCEffectPlayer getTCEffectPlayer()
```

onDestroy

说明

停止播放并释放资源。

接口

```
public void onDestroy()
```

TCEffectConfig

说明

构造特效播放器配置，需要通过 Builder 构造。

Builder接口

```
// 设置 CodecType
public Builder setCodecType(CodecType type)
```

参数说明

setCodecType(CodecType type)：它有三个取值，分别是：

- TCEffectConfig.CodecType.TC_MPLAYER (MPLAYER播放引擎)
- TCEffectConfig.CodecType.TC_MCODEC(MCODEC播放引擎)
- TCEffectConfig.CodecType.TX_LITEAV_SDK (腾讯云播放器SDK)

```
// 设置播放配置，非必需得步骤
```

```
TCEffectConfig config = new
TCEffectConfig.Builder().setCodecType(TCEffectConfig.CodecType.TC_MPLAYER).build();
mPlayerView.setConfig(config);
```

注意：

1. 目前仅支持在播放器开始前调用 setConfig() 方法来设置播放配置，开始播放后不支持修改配置。
2. 目前支持的三种 CodecType 仅对 TEP 动画生效
3. 如果设置 CodecType 为 TCEffectConfig.CodecType.TX_LITEAV_SDK ，则还需要单独引入腾讯云播放器 SDK，以及申请、注册好其对应的license。
4. 如果不设置 config，则会默认使用 CodecType = TCEffectConfig.CodecType.TC_MPLAYER

```
// 设置冻结停留帧索引
public Builder setFreezeFrame(int frame)
```

参数说明

入参 frame 表示当动画播放中需要停留在哪一帧。目前可选值：

- TCEffectConfig.FREEZE_FRAME_NONE：关闭 freezeFrame 能力，播放器正常播放暂停消失。
- TCEffectConfig.FREEZE_FRAME_LAST：当第一次播放完毕之后，画面停留在最后一帧。

```
// 设置后续要播放的动画格式
public Builder setAnimType(AnimType type)
```

参数说明

用于指定后续要播放的动画格式，适用于某些情况下，要播放的动画文件后缀被修改的场景。入参 type 表示要指定的动画格式。目前可选值：

- TCEffectConfig.AnimType.AUTO（sdk默认策略，即以动画文件的后缀来判断动画格式，比如 tcmp4/mp4/tep/tepg 等格式）
- TCEffectConfig.AnimType.MP4（MP4动画格式，后续都将动画文件当做MP4类型来播放，无视文件后缀名）
- TCEffectConfig.AnimType.TCMP4 (TCMP4动画格式，后续都将动画文件当做TCMP4类型来播放，无视文件后缀名)

TCEffectAnimInfo

说明

存储当前播放的动画信息

属性说明

属性名	类型	描述
type	int	当前动画类型，取值：TCEffectAnimInfo.TYPE_MP4 和 TCEffectAnimInfo.TYPE_TCMP4
duration	long	动画时长，单位毫秒
width	int	动画宽度
height	int	动画高度
encryptLevel	int	当前动画的高级加密类型，取值如果是 TCEffectAnimInfo#ENCRYPT_LEVEL_NONE 表示没有高级加密，否则表示已是高级加密
mixInfo	MixInfo	融合动画信息，为 null 时则表示该动画无融合信息

TCEffectAnimInfo.MixInfo

说明

存储当前播放动画中的融合信息

属性说明

属性名	类型	描述

textMixItemList	List<MixItem>	文字融合信息，为 null 时则表示无文字融合信息
imageMixItemList	List<MixItem>	图片融合信息，为 null 时则表示无图片融合信息

TCEffectAnimInfo.MixInfo.MixItem

说明

存储当前播放动画中的融合信息

属性说明

属性名	类型	描述
id	String	当前融合动画 id。如果是 tcmp4 时，则是对应要替换的图层 index；如果是 mp4 时，则是在工具中显示的 id 值
tag	String	当前融合动画 tag。如果是 tcmp4 时，则是固定值：TCEffectPlayerConstant.TAG_TPEG；如果是 mp4 时，则是在工具中填写的 tag 值
text	String	当前文字融合动画文字内容(图片融合动画时该值为空)。如果是 tcmp4 时，则是其内部原始的文字内容；如果是 mp4 时，则是在工具中填写的 tag 值

TCEffectText

说明

融合动画替换文本样式数据类

属性说明

属性名	类型	描述
text	String	最终要替换显示的文本内容
color	int	文字颜色，格式要求：ARGB，如 0xFFFFFFFF
fontStyle	String	文字显示样式，可取值："bold"表示粗体，不传则默认大小
alignment	int	文字对齐方式，可取值：TEXT_ALIGNMENT_NONE（默认值，即保持sdk默认对齐方式）、TCEffectText.TEXT_ALIGNMENT_LEFT(居左)、TCEffectText.TEXT_ALIGNMENT_CENTER(居中)、TCEffectText.TEXT_ALIGNMENT_RIGHT(居右)
fontSize	float	文字大小，单位是px；如果设置了文字大小(值大于0)，则内部自动缩放策略失效，强制以设置的文字大小为准，则可能出现文字过大显示不全的问题

iOS
集成文档

最近更新时间：2025-02-28 12:24:15

本文主要介绍如何快速将腾讯礼物动画特效 SDK 集成到您的项目中。按照以下步骤进行配置，即可完成 SDK 的集成工作。

开发环境要求

- Xcode 9.0+。
- iOS 9.0 以上的 iPhone 或者 iPad 真机。
- 项目已配置有效的开发者签名。

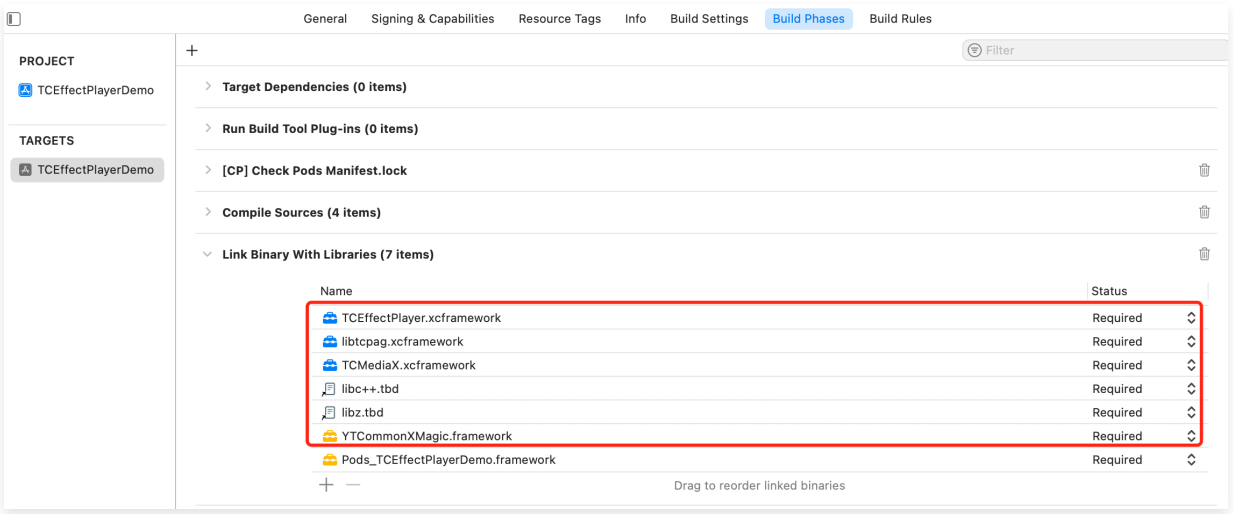
集成指引

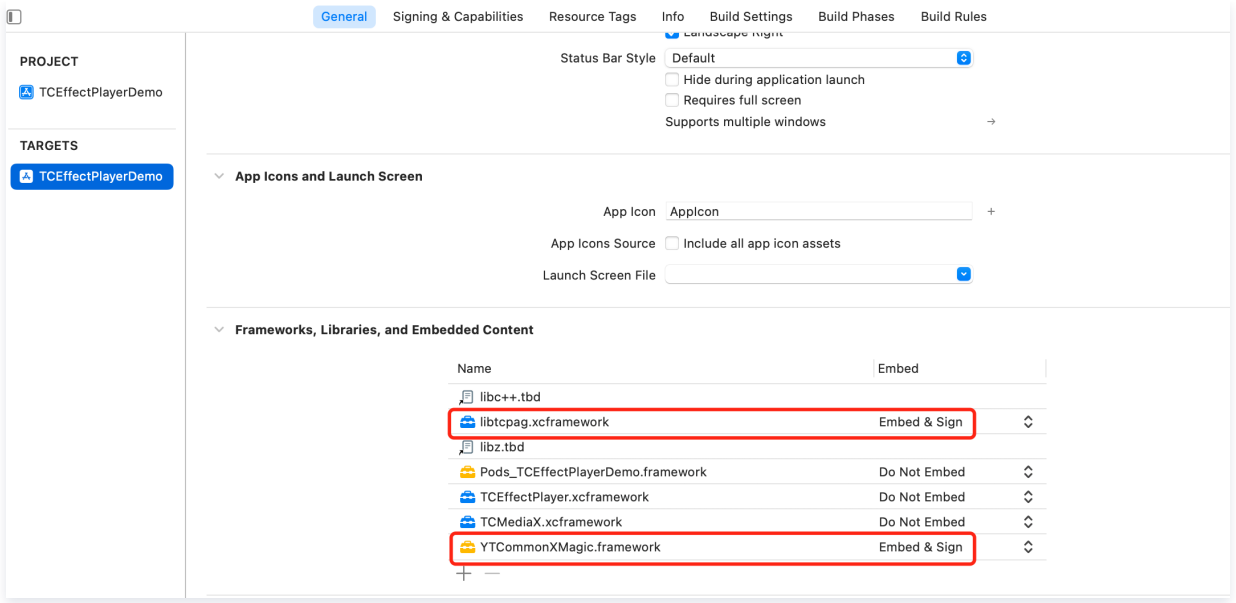
手动集成

1. 下载 MediaX_Android_iOS_Latest.zip ，下载完成后进行解压获取到 SDK 库文件。
2. 使用 TCEffectPlayer 需要集成下面的 framework：

   SDK库
   - TCMediaX.xcframework
   - TCEffectPlayer.xcframework
   - libtcpag.xcframework
   - YTCommonXMagic.framework

   系统库
   - libz.tab
   - libc++.tbd
   - AVFoundation.framework





3. 设置 Other linker Flags

   在“Build Settings”>“Other linker Flags”>设置“-ObjC”。

Pods 方式集成

目前特效播放器支持 Pods 渠道方式进行集成，集成 Pods 方法如下：

```
# 引入TCMediaX
pod 'TCMediaX', :podspec => 'https://mediacloud-76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/3.0.246/TCMediaX.podspec'

# 引入TCEffectPlayer
pod 'TCEffectPlayer', :podspec => 'https://mediacloud-76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/3.0.246/TCEffectPlayer.podspec'

# 引入YTCommonXMagic.framework
pod 'YTCommonXMagic', :podspec => 'https://mediacloud-76607.gzc.vod.tencent-
cloud.com/MediaX/iOS/podspec/release/YTCommonXMagic_1.1/YTCommonXMagic.podspec'
```

TCEffectPlayer 支持单独集成或依赖腾讯云播放器 SDK 联动集成的方式，两者之间的区别在于是否利用腾讯云播放器强大的解码能力。

- 若您使用独立集成的方式，则视频的解码器类型（ `TCEffectConfig#vapEngineType` ）需设置为 TCEPCodecTypeAVPlayer 联动腾讯云播放器 SDK。
- 如果您集成腾讯云播放器精简版 SDK（LiteAVSDK_Player_Mini），视频的解码器类型（TCEffectConfig – vapEngineType）需设置为 TCEPCodecTypeVODPlayer。播放器精简版 SDK不需要额外申请播放器 License。
- 如果您的项目中已经集成了腾讯云视立方 SDK，请检查是否包含播放器 SDK，同时检查是否已经申请和配置播放器 License，播放器 License 申请指引。
- 腾讯云播放器接入文档可参见 腾讯云播放器集成文档 。

申请动画特效播放 License

使用 TCEffectPlayer 需要申请动画特效播放 License，详细申请流程请参见 控制台 License 申请指南。

使用文档

最近更新时间：2025-02-28 12:24:14

本文将详细介绍如何在项目中快速地使用腾讯礼物动画特效 SDK。只需遵循以下步骤，即可完成 SDK 的配置和使用。

初始化注册 License

在正式使用 腾讯礼物动画特效 SDK 时，需要先设置 License，设置 License 成功之后，才可以进行后续的 SDK 使用。
设置 License 方式如下：

```
#import <TCMediaX/TCMediaX.h>
@interface AppDelegate ()<TCMediaXBaseDelegate>
@end

@implementation AppDelegate

// 使用时替换 LICENCE_URL 为自己的 LICENCE_URL
static NSString *LICENCE_URL = @"";
static NSString *LICENCE_KEY = @"";

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    // Override point for customization after application launch.
   [[TCMediaXBase getInstance] setLicenceURL:LICENCE_URL key:LICENCE_KEY];
   [[TCMediaXBase getInstance] setDelegate:self];
    return YES;
}


#pragma mark - TCMediaXBaseDelegate
- (void)onLicenseCheckCallback:(int)errcode withParam:(NSDictionary *)param {
    NSLog(@"onLicenseCheckCallback:%d",errcode);
}

@end
```

注意：

1. License 是强线上检验逻辑，应用首次启动后调用 TCMediaXBase#setLicense 时，需确保网络可用。 在App首次启动时，可能还没有授权联网权限，则需要等授予联网权限后，再次调用 TCMediaXBase#setLicense。
2. 监听 TCMediaXBase#setLicence 加载结果：ILicenseCallback#onResult 接口，如果失败要根据实际情况做对应重试及引导，如果多次失败后，可以限频，并业务辅以产品弹窗等引导，让用户检查网络情况。
3. 对于多进程的 App， 确保每个使用特效播放器的进程启动时，都调用了 TCMediaXBase#setLicense。例如： Android 端使用独立进程播放特效的 App， 后台播放时进程被系统 kill 掉重启时，也要调用 TCMediaXBase#setLicense。

日志管理

腾讯礼物动画特效 SDK 默认支持保存运行日志，如果测试过程中出现问题，可以提取日志反馈给腾讯云客服，您可以根据业务需要把此目录的日志上传到业务后台，用于定位线上用户问题。
腾讯礼物动画特效 SDK iOS 端的日志保存在沙箱 Documents/TCMedialog 文件夹，日志文件按照日期命名。详细日志导出教程 参考这里 。
播放日志默认是开启保存的， 如需要关闭，可通过 `TCMediaXBase#setLogEnable` 传入 NO 关闭。

播放器使用

初始化 TCEffectAnimView

```
- (TCEffectAnimView *)alphaAnimView {
    if (!_alphaAnimView) {
```

```
        _alphaAnimView = [[TCEffectAnimView alloc] init];
        _alphaAnimView.backgroundColor = [UIColor clearColor];
        _alphaAnimView.effectPlayerDelegate = self;
        _alphaAnimView.loop = YES;
        [_alphaAnimView setRenderMode:TCEPVPViewContentModeScaleToFill];
    }
    return _alphaAnimView;
}
```

添加 alphaAnimView 并布局

```
// 添加alphaAnimView
[self.view addSubview:self.alphaAnimView];

//布局
self.alphaAnimView.frame = self.view.bounds;
```

获取视频资源并进行播放

```
 [self.alphaAnimView startPlay:mp4Url];
```

融合动画配置

实现融合动画，需要实现 TCEffectAnimViewDelegate 代理方法。

融合动画文字替换

替换文本，同时指定文本的对齐方式、颜色、是否粗体，使用 loadTextForPlayer 接口。

```
// 替换文本，同时指定文本的对齐方式、颜色、是否粗体，使用 loadTextForPlayer 接口，推荐使用此接口
- (TCEffectText *)loadTextForPlayer:(ITCEffectPlayer *)player
                            withTag:(NSString *)tag {
    if (tag != nil && [tag isEqualToString:@"name"]) {
        TCEffectText *effectText = [[TCEffectText alloc] init];
        effectText.text = @"reText";
        // TCEPTextAlignmentLeft = 0,      ///< 表示文字左对齐
        // TCEPTextAlignmentCenter = 1,    ///< 表示文字居中对齐
        // TCEPTextAlignmentRight = 2      ///< 表示文字右对齐
        effectText.alignment = TCEPTextAlignmentLeft;
        //effectText.fontStyle = @"bold";
        //effectText.color = [UIColor blackColor];
        return effectText;;
    }
    return nil;
}
```

融合动画图片替换

通过获取 tag 值后返回对应的 UIImage ，融合动画播放器内部不处理转换过程，需要在此方法里自行处理。

```
- (void)loadImageForPlayer:(ITCEffectPlayer *)player
                   context:(NSDictionary *)context
                completion:(void(^)(UIImage *image,
                                    NSError *error))completionBlock {
    dispatch_async(dispatch_get_main_queue(), ^{
        // 获取到当前需要替换的图片对应的 tag，然后根据 tag 返回图片
        NSString *tag = context[TCEPContextSourceTypeImageIndex];
        ...
        // 下载并缓存图片
```

```
        if (image) {
            completionBlock(image, nil);
        } else {
            completionBlock(nil, error);
        }
        }];
    });
}
```

监听动画自定义资源被点击的事件，以 tag 来区分不同的融合动画区域。

```
- (void)tcePlayerTagTouchBegan:(ITCEffectPlayer *)player tag:(NSString *)tag {
    NSString *content = [NSString stringWithFormat:@"点击了:%@",tag];
    [self.view tuiad_alert:content];
}
```

播放配置

```
// 设置播放配置，非必需的步骤
TCEffectConfig *config = [[TCEffectConfig alloc] init];
[self.alphaAnimView setEffectPlayerConfig:config];
```

TCEffectConfig 中目前支持：

1. vapEngineType(CodecType type)：它有三个取值，分别是：
   - TCEPCodecTypeAVPlayer，礼物动画特效默认播放引擎。
   - TCEPCodecTypeVODPlayer ，腾讯云播放器SDK播放引擎。

> ⚠ 注意：
> 1. 目前仅支持在播放器开始前调用 setEffectPlayerConfig 方法来设置播放配置，开始播放后不支持修改配置。
> 2. 目前支持的 2 种 vapEngineType 仅对 MP4 动画生效。
> 3. 如果设置 vapEngineType为 TCEPCodecTypeVODPlayer，则还需要单独引入腾讯云播放器SDK，以及申请、注册好其对应的 license。

2. freezeFrame 用于设置播放动画冻结帧，目前可选值：
   - FRAME_NONE : 默认取值，关闭 freezeFrame 能力，播放器正常播放暂停消失。
   - FRAME_FIRST: 播放完毕，重新开始播放到下一次首帧出现时暂停。
   - FRAME_LAST : 当第一次播放完毕之后，画面停留在最后一帧。

播放控制

开始播放

TCEffectPlayer 支持播放 .mp4、.tcmp4 文件格式资源。

> ⚠ 注意：
> 只支持播放本地视频资源， 如果您使用的网络视频资源， 先下载到本地再播放。

```
NSString *mp4Url = @"xxx/xxx/tuiad_vapx_cool_sss.mp4"];
[self.alphaAnimView startPlay:mp4Url];
```

暂停播放

```
[self.alphaAnimView pause];
```

继续播放

```
[self.alphaAnimView resume];
```

停止播放

当不需要继续使用播放器时，需要停止播放，释放占用的资源。

```
[self.alphaAnimView stopPlay];
```

静音播放

```
[self.alphaAnimView setMute:YES];
```

循环播放

```
[self.alphaAnimView setLoop:YES];
```

常见问题

在播放过程中出现下面的日志信息，如何处理?

```
License checked failed! tceffect player license required!
```

请检查是否申请特效播放器 License，并进行了初始化注册。

如何提取运行日志?

提取礼物动画特效运行日志，请参见 日志管理 。

常见错误码

错误码	描述
-10003	创建线程失败
-10004	render 创建失败
-10005	配置解析失败
-10006	文件无法读取
-10007	动画视频编码格式是h265，在当前设备上不支持
-10008	参数非法
-10009	license 不合法
-10010	模拟器不支持
-100200	签名校验失败
-100201	解密数据失败
-100202	BundleID 或者 PackageName 不一致
-100203	资源过期
-100204	JSON 字段不正确
-100205	boxType 类型不支持
-100206	融合动画资源获取失败

API 文档

最近更新时间：2025-02-28 12:24:15

TCMediaXBase类

getInstance

说明
获取 TCMediaXBase 的单例。
接口

```
+ (instancetype)getInstance;
```

参数说明
无

setLicense

说明
设置 License
接口

```
- (void)setLicenceURL:(NSString *)url key:(NSString *)key;
```

参数说明

参数名	类型	描述
url	NSString	licence 的 url
key	NSString	licence 的 key

setDelegate

说明
设置 License 注册结果回调
接口

```
- (void)setDelegate:(id<TCMediaXBaseDelegate>)delegate;
```

回调声明：

```
- (void)onLicenseCheckCallback:(int)errcode withParam:(NSDictionary *)param;
```

License 校验错误码：

```
public interface TXLicenceErrorCode {
    TMXLicenseCheckOk = 0,                  // 成功
    TMXLicenseCheckInvalidInput = -1,       // 输入参数无效
    TMXLicenseCheckDownloadError = -3,      // 下载环节失败，请检查网络设置
    TMXLicenseCheckLocalLicenseEmpty = -4,  // 从本地读取的TE授权信息为空
    TMXLicenseCheckFileContentEmpty = -5,   // 读取VCUBE TEMP License文件内容为空
    TMXLicenseCheckJsonErrorVcube = -6,     // v_cube.license文件json字段错误
    TMXLicenseCheckSignatureError = -7,     // 签名校验失败
    TMXLicenseCheckDecodeError = -8,        // 解密失败
    TMXLicenseCheckTEJsonError = -9,        // TELicense字段里的json字段错误
    TMXLicenseCheckTEContentEmpty = -10,    // 从网络解析的TE授权信息为空
```

```
    TMXLicenseCheckWriteLocalFailed = -11,    // 把TE授权信息写到本地文件时失败
    TMXLicenseCheckDownloadAndAssetBothFailed = -12, // 下载失败，解析本地asset也失败
    TMXLicenseCheckAuthError = -13,           // 鉴权失败
}
```

setLogEnable

说明

是否开启 Log 输出，默认开始。

Android 端保存在 /sdcard/Android/data/packagename/files/TCMediaX 目录，iOS 端保存在 sandbox 的 Documents/TCMediaX 目录，您可以根据业务需要把此目录的日志上传到业务后台，用于定位线上用户问题。

接口

```
- (void)setLogEnable:(BOOL)enabled;
```

getSdkVersion

说明

获取当前 sdk 的版本。

接口

```
- (NSString *)getSdkVersion;
```

TCEffectAnimView类

initWithFrame

说明

创建 TCEffectAnimView。

接口

```
- (instancetype)initWithFrame:(CGRect)frame;
```

参数说明

frame： View 对象的 frame。

startPlay

说明

启动播放器。

接口

```
- (void)startPlay:(NSString *)url;
```

参数说明

url 为视频资源地址。

> ⚠ **注意：**
> 只支持播放本地视频资源， 如果您使用的网络视频资源， 先下载到本地再播放。

setVideoMode

说明

设置 mp4 动画的 alpha 和 rgb 区域的对齐方式。

接口

```
- (void)setVideoMode:(TCEPVAPVideoFrameTextureBlendMode)mode;
```

参数说明
mode 支持以下格式：

枚举值	含义
TCEPVAPVFTextureBlendMode_None	普通mp4文件
TCEPVAPVFTextureBlendMode_AlphaLeft	左右对齐（alpha左\rgb右）
TCEPVAPVFTextureBlendMode_AlphaTop	上下对齐（alpha上\rgb下）
TCEPVAPVFTextureBlendMode_AlphaRight	左右对齐（rgb左\alpha右）
TCEPVAPVFTextureBlendMode_AlphaBottom	上下对齐（rgb上\alpha下）

setEffectPlayerConfig

说明
设置特效播放器参数，需要在启动播放前调用。
接口

```
- (void)setEffectPlayerConfig:(TCEffectConfig *)config;
```

参数说明
参考 TCEffectConfig 类。

setRenderMode

说明
设置对齐方式
接口

```
- (void)setRenderMode:(TCEPVPViewContentMode)renderMode;
```

参数说明
renderMode 支持以下格式：

枚举值	含义
TCEPVPViewContentModeScaleToFill	内容被拉伸到填满整个视图，可能失真，默认值。
TCEPVPViewContentModeAspectFit	内容保持原始宽高比，缩放以完全显示，可能有空白区域。
TCEPVPViewContentModeAspectFill	内容保持原始宽高比，缩放以填满视图，可能会裁剪部分内容。

effectPlayerDelegate

说明
TCEPAnimViewDelegate 支持设置礼物动画特效播放事件回调，设置融合动画信息等。
播放回调接口

```
// 动画开始播放回调
- (void)tcePlayerStart:(ITCEffectPlayer *)player;

// 动画结束播放回调
- (void)tcePlayerEnd:(ITCEffectPlayer *)player;

// 动画播放错误回调
```

```
- (void)tcePlayerError:(ITCEffectPlayer *)player error:(NSError *)error;

// 播放器事件通知
- (void)onPlayEvent:(ITCEffectPlayer *)player
              event:(int)EvtID
         withParam:(NSDictionary *)param;
```

融合动画信息替换接口

```
// 替换融合动画资源配置中的文本占位符， 支持配置文件的对齐方式、颜色等属性。
- (TCEffectText *)loadTextForPlayer:(ITCEffectPlayer *)player
                            withTag:(NSString *)tag;


// 替换融合动画资源中的图片信息
// 获取当前图片对应 tag： NSString *tag = context[TCEPContextSourceTypeImageIndex];
- (void)loadImageForPlayer:(ITCEffectPlayer *)player
                   context:(NSDictionary *)context
                completion:(void(^)(UIImage *image,
                                    NSError *error))completionBlock;
```

融合动画点击接口

```
// 融合动画的资源点击事件回调
- (void)tcePlayerTagTouchBegan:(ITCEffectPlayer *)player tag:(NSString *)tag;
```

setRenderRotation

说明

设置动画旋转方向。

接口

```
- (void)setRenderRotation:(TCEP_Enum_Type_HomeOrientation)rotation;
```

参数说明

rotation 支持以下格式：

枚举值	含义
TCEP_HOME_ORIENTATION_RIGHT	HOME 键在右边，横屏模式。
TCEP_HOME_ORIENTATION_DOWN	HOME 键在下面，手机直播中最常见的竖屏直播模式
TCEP_HOME_ORIENTATION_LEFT	HOME 键在左边，横屏模式。
TCEPVAPVFTextureBlendMode_AlphaRight	HOME 键在上边，竖屏直播。

isPlaying

说明

返回特效播放器是否在播放中

接口

```
- (BOOL)isPlaying;
```

resume

说明

恢复特效动画播放

接口

```
- (void)resume;
```

pause

说明

暂停特效动画播放

接口

```
- (void)pause;
```

seekTo

说明

跳转到指定位置开始播放

> ⚠ **注意：**
> 1. startPlay() 之后才可以调用该方法，否则不生效。
> 2. 该接口对于 tcmp4 动画、或者设置了 vapEngineType 为 TCEPCodecTypeVODPlayer 时播放的 mp4 动画生效。
>
> time：要跳转到指定时长处开始播放，单位毫秒。

接口

```
- (void)seek:(float)time;
```

seekProgress

说明

跳转到指定位置开始播放

> ⚠ **注意：**
> 1. startPlay() 之后才可以调用该方法，否则不生效。
> 2. 该接口对于 tcmp4 动画、或者设置了 vapEngineType 为 TCEPCodecTypeVODPlayer 时播放的 mp4 动画生效。
> 3. 入参 progress 取值范围为 [0.0 – 1.0]，超出范围的不生效。
>
> progress：跳转到动画时长的指定百分比处开始播放，单位百分比，取值范围为：[0.0 – 1.0]。

接口

```
- (void)seekProgress:(float)progress;
```

setLoop

说明

设置循环播放。

YES： 表示循环播放。

NO： 表示关闭循环播放。

接口

```
- (void)setLoop:(BOOL)loop;
```

setLoopCount

说明

设置循环播放次数。

loopCount： 表示循环播放次数。当loopCount <=0 时，表示无限循环播放；当 loopCount=n(n>=1)时表示从开始播放到播放结束，共播放n次

> ⚠ 注意：
> 1. loopCount 默认值是1，即当外部不主动调用该方法时，动画只播放一次就结束。
> 2. setLoop 方法内部会调用当前方法，即当 isLoop = true 时，等价于调用 setLoopCount(-1)；即当 isLoop = false 时，等价于调用 setLoopCount(1)；因此这两个方法是互相影响的，后调用的会覆盖之前的调用。

接口

```
- (void)setLoopCount:(int)loopCount;
```

setDuration

说明

设置动画需要多长时间播放完成。设置之后，后续动画播放时自动调整动画播放速度，以保证动画在设置的规定时长时播放结束。
即：设置的时长超过动画原时长，则动画慢放；小于动画原时长，则快进播放。

> ⚠ 注意：
> 1. 目前仅对 tcmp4 格式的动画生效。
> 2. 当前方法和 setRate 设置倍速的方法是互斥的，后调用的会覆盖掉先调用的。
>
> durationInMilliSec： 要设置的时长，单位毫秒。

接口

```
- (void)setDuration:(long)durationInMilliSec;
```

stopPlay

说明

停止播放。

接口

```
- (void)stopPlay;
```

setMute

说明

设置是否静音播放。
YES： 静音播放。
NO： 非静音播放。

接口

```
- (void)setMute:(BOOL)bEnable;
```

getTCAnimInfo

说明

获取当前播放动画对应的信息，返回 TCEffectAnimInfo 实例，详见 TCEffectAnimInfo 。

> ⚠ 注意：
> 该方法必须在 IAnimPlayListener#onPlayStart() 方法中，或者该方法执行之后调用才可以获取到当前动画的信息，否则返回 null。

接口

```
- (nullable TCEffectAnimInfo *)getAnimInfo;
```

TCEffectConfig类

说明

构造特效播放器配置，支持属性有：

1. vapEngineType(CodecType type)：它有三个取值，分别是：
   - TCEPCodecTypeAVPlayer，礼物动画特效默认播放引擎。
   - TCEPCodecTypeVODPlayer ，腾讯云播放器SDK播放引擎。

```
// 设置播放配置，非必需的步骤
TCEffectConfig *config = [[TCEffectConfig alloc] init];
[self.alphaAnimView setEffectPlayerConfig:config];
```

> **注意：**
> 1. 目前仅支持在播放器开始前调用 setEffectPlayerConfig 方法来设置播放配置，开始播放后不支持修改配置。
> 2. 目前支持的 2 种 vapEngineType 仅对 MP4 动画生效。
> 3. 如果设置 vapEngineType为 TCEPCodecTypeVODPlayer，则还需要单独引入腾讯云播放器 SDK，以及申请、注册好其对应的 license。

2. freezeFrame 用于设置播放动画冻结帧，目前可选值：
   - FRAME_NONE：默认取值，关闭 freezeFrame 能力，播放器正常播放暂停消失。
   - FRAME_FIRST：播放完毕，重新开始播放到下一次首帧出现时暂停。
   - FRAME_LAST：当第一次播放完毕之后，画面停留在最后一帧。

TCEffectAnimInfo

说明

存储当前播放的动画信息

属性说明

属性名	类型	描述
type	TCEPAnimResourceType	当前动画类型，取值：TCEPAnimResourceTypeMP4 MP4类型的资源和 TCEPAnimResourceTypeTCMP4 TCMP4类型的资源。
duration	long	动画时长，单位毫秒
width	int	动画宽度
height	int	动画高度
encryptLevel	TCEPEncryptLevelType	当前动画的高级加密类型，取值如果是 TCEffectAnimInfo#ENCRYPT_LEVEL_NONE 表示没有高级加密，否则表示已是高级加密

TCEffectText

说明

融合动画替换文本样式数据类

属性说明

属性名	类型	描述
text	NSString	最终要替换显示的文本内容
color	UIColor	文字颜色，格式要求：ARGB，如 0xFFFFFFFF
fontStyle	NSString	文字显示样式，可取值："bold"表示粗体，不传则默认大小

alignment	int	文字对齐方式，可取值：TCEPTextAlignmentNone（默认值，即保持sdk默认对齐方式）、TCEPTextAlignmentLeft（居左）、TCEPTextAlignmentCenter（居中）、TCEPTextAlignmentRight（居右）。
fontSize	CGFloat	文字大小，单位是px；如果设置了文字大小(值大于0)，则内部自动缩放策略失效，强制以设置的文字大小为准，则可能出现文字过大显示不全的问题