引入 SDK

Android 相关

Android 版本的SDK包括的资源有：MtSDK.aar 库，assets文件夹，jniLibs 文件夹，以及可选择使用的开源UI mt 文件夹。

拷贝 MtSDK.aar 库，放入对应模块(如 app 模块)的 libs 文件夹中，然后在此模块的 build.gradle 文件的 dependencies 中，添加依赖语句：

```
dependencies {
  implementation files('libs/MtSDK.aar')
 }
```

拷贝 assets 文件夹，放入对应模块的目录中；

jniLibs 文件夹提供了两种架构的 so 包，可以根据需要将对应的 so 包拷贝到项目目录中。

IOS 版本的 SDK 包 mtsdk_standard 包含：MtSDK.framework、MtSDK.bundle 、 MTUI 文件夹 (可选)。

mtsdk_standard 文件夹拷贝到 IOS 工程目录，然后引入到工程中;

如果您原工程已添加 Masonry, SSZipArchive, fmdb 可将 MTUI/Lib 目录下的 Masonry.framework、 ZipArchive.framework,fmdb 删除，避免冲突;

工程导入

（1）mtsdk_standard 文件夹拷贝到 IOS 工程目录，然后引入到工程中,其中 MtSDK.framework 与 UI 文件中的 Masonry, SSZipArchive 均为动态库需要做嵌入操作;

（2）如果您原工程已添加 Masonry, SSZipArchive, 可将 Lib 目录下的 Masonry.framework 、ZipArchive.framework 删除，避免冲突;

（3）添加 SDK 所使用的依赖库 libc++.tbd

使用开源 UI（可选）

Android 相关

可以选择使用我们提供的开源 UI 库 mt，方法是将 mt 文件夹拷贝到根目录下，然后在 settings.gradle 文件中添加 include 语句：

```
include ':mt'
```

然后在 app 模块的 build.gradle 文件中添加 dependencies 依赖：

```
implementation project(':mt')
```

IOS 相关

使用 MTUI 方法如下:

引入头文件:

```
#
import "MTUIManager.h"
```

MTUIManager 创建 UI:

```
// 加载 UI
- (void)viewDidLoad {
[super viewDidLoad];
...
...
[[MTUIManager shareManager]loadToWindowDelegate:self];
//[[MTUIManager shareManager] loadToView:self.view forDelegate:self];
//UIView *view = [[MTUIManager shareManager]returnLoadToViewDelegate:self];
}

// 加载 UI，添加 MTUIManagerDelegate 代理
// 测试拍照和切换摄像头可设置以下参数 注意 showsDefaultUI 默认为 NO 在 loadTo 方法之前使用
[MTUIManager shareManager].showsDefaultUI = YES;
//在 loadTo 方法之后添加默认按钮
[self.view addSubview:[MTUIManager shareManager].defaultButton
];

//弹出美颜 UI 展示美颜界面调用此方法
[[MTUIManager shareManager] showMainMenuView];
```

MTUIManager 销毁 UI:

```
- (void)dealloc {
[[MTUIManager shareManager] destroy];
}
```

# 调用

SDK 初始化

初始化方法需要使用 key 进行鉴权，鉴权成功方可使用。

Android 初始化方法调用如下：

```
MtSDK.get().initSDK(context, "key", new MtSDK.InitCallback() {
        @Override
         public void onSuccess() { }

         @Override
         public void onFailure() { }
        });
```

IOS 初始化方法调用如下：

初始化函数程序中调用一次即可生效，建议用户在 Application 创建的时候调用;如果渲染功能使用不频繁，也可以在使用的时候调用

```
 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
        {
        // 在线鉴权初始化方法

         [[MtSDK Get] initSDK:"鉴权的 key" withDelegate:self];

           return YES;
        }

        #pragma mark - MtSDKDelegate
        - (void)onSuccess {
              NSLog(@"MtSDK 加载成功");
        }
        - (void)onFailure {
              NSLog(@"MtSDK 加载失败");
        }
```

萌图 SDK 有初始化日志，在日志栏搜索 InitInfo，可查看版本信息和初始化状态。

渲染步骤

Android 平台

定义布尔变量 bool，用来标志渲染方法是否初始化完成，然后根据得到的视频帧格式的不同，使用对应的方法进行渲染；

GL_TEXTURE_EXTERNAL_OES 纹理格式，首先调用渲染初始化方法 initRenderTextureOES，当返回 true 时初始化完成；然后调用渲染方法 renderTextureOES，返回值为 GL_TEXTURE_2D 类型的纹理：

```
/**
        * 纹理渲染初始化
        *
        * @param width       纹理宽度
        * @param height      纹理高度
        * @param rotation  纹理是否需要旋转，不需旋转为 CLOCKWISE_0
        * @param isMirror  纹理是否存在镜像
        * @param number  人脸检测数目上限设置，推荐取值范围为 1~5
        */
        public boolean initRenderTextureOES(int width, int height, MtRotation rotation, boolean isMirror, int number);

        /**
        * 渲染
        *
        * @param textureOES      纹理 id
        */
        public int renderTextureOES(int textureOES);

        /**
        * 使用方法
        */
        if (!bool) {
            bool = MtSDK.get().initRenderTextureOES(width, height, rotation, isMirror, number);
        }
        int textureId = MtSDK.get().renderTextureOES(textureOES);
```

byte[] 视频帧，首先调用渲染初始化方法 initRenderPixels，当返回 true 时初始化完成；然后调用渲染方法 renderPixels：

```
/**
        * 视频帧渲染初始化
        *
        * @param width       视频帧宽度
        * @param height      视频帧高度
        * @param rotation  视频帧是否需要旋转，不需旋转为 CLOCKWISE_0
        * @param isMirror  视频帧是否存在镜像
        * @param number  人脸检测数目上限设置，推荐取值范围为 1~5
        */
        public boolean initRenderPixels(int width, int height, MtRotation rotation, boolean isMirror, int number);

        /**
        * 渲染
        *
        * @param data       视频帧数组
        */
        public void renderPixels(byte[] data);

        /**
        * 使用方法
        */
        if (!bool) {
            bool = MtSDK.get().initRenderPixels(width, height, rotation, isMirror, number);
        }
        MtSDK.get().renderPixels(data);
```

结束渲染时，根据视频帧格式的不同，调用对应的 destroy 方法释放掉资源，调用位置通常在 视频帧回调接口 的销毁处，或者是 Activity， Fragment 的销毁处，同时将定义的布尔变量 bool 置为 false:

```
/**
```

```
* 使用其中一个
*/
    MtSDK.get().destroyRenderPixels();
    MtSDK.get().destroyRenderTexture();
    MtSDK.get().destroyRenderTextureOES();

/*
* 将 bool 置为 false
*/
    bool = false;
```

IOS 平台

新建 [MtSDK Get]管理器对象，所有的渲染都必须通过 [MtSDK Get] 对象调用， [MtSDK Get] 的默认构造函数，新建默认渲染参数的渲染器对象;也可以通过具体功能函数来 预设 [MtSDK Get]的渲染参数，代码如下:

```
//自定义渲染参数
    [[MtSDK Get] setRenderEnable:YES];
    [[[MtSDK Get] setWhitenessValue: 100];
    ......
```

渲染 GL_TEXTURE_2D 的纹理，调用 [MtSDK Get] 的 renderTexture2D 方法，返回 GL_TEXTURE_2D 类型的纹理 ID，代码如下:

```
// 默认人脸相关特效可支持人脸数
    BOOL isRenderInit = false; //initRenderTextureWidth 初始化方法一般只执行一次该 isRenderInit 参数应设为全局变量在释放资源的时候从新设为 NO
        if(!isRenderInit){
        isRenderInit = [[MtSDK Get] initRenderTextureWidth:imageWidth
                            Height:imageHeight
                        Rotation:CLOCKWISE_90
                            Mirror:YES
                    FaceNumber:5];
        }
    int textureId = [[MtSDK Get] renderTexture:texture2D];//2D 纹理 Id
```

渲染 CVPixelBufferRef 视频帧，调用 [MtSDK Get] 的 renderPixels 方法，返回渲染后的视频 帧，代码如下:

```
// 默认人脸相关特效可支持人脸数
        BOOL isRenderInit = false;//initRenderTextureWidth 初始化方法一般只执行一次该 isRenderInit 参数应设为全局变量在释放资源的时候从新设为 NO
            if(!isRenderInit){
               isRenderInit = [[MtSDK Get] initRenderPixelsFormat:BGRA
                             Width:imageWidth
                            Height:imageHeight
                          Rotation:CLOCKWISE_90
                            Mirror:YES
                        FaceNumber:5];
            }
            [[MtSDK Get] renderPixels:baseAddress];
```

渲染结束，要释放渲染资源，否则会造成内存泄漏，一般随着渲染数据回调接口的销毁释放，或者随着生命周期结束释放。释放资源调用 [MtSDK Get] 的 destroy 方法，代码如下:

```
 //TEXTURE_2D 的纹理的释放方法
            [[MtSDK Get] destroyRenderTexture];
            _isRenderInit = false;
          //视频帧的释放方法
            [[MtSDK Get] destroyRenderPixels];
            _isRenderInit = false;
```

API

Android 平台

下载资源配置在我们的服务器，我们建议将这些资源放在自己的服务器上，方法如下:

```
MtSDK.get().setResUrl(String path);//设置资源路径
        MtSDK.get().getResUrl();//获取资源路径
```

下载资源必须解压到指定的路径下，获取资源下载地址和资源保存路径的方法如下:

```
MtSDK.get().getStickerUrl();//动态贴纸资源下载地址
        MtSDK.get().getExpressionUrl();//表情娱乐资源下载地址
        MtSDK.get().getMaskUrl();//面具资源下载地址
        MtSDK.get().getAtmosphereUrl();//气氛道具资源下载地址
```

MtSDK.get().getStickerPath();//动态贴纸资源保存路径

MtSDK.get().getExpressionPath();//表情娱乐资源保存路径

MtSDK.get().getMaskPath();//面具资源保存路径

MtSDK.get().getAtmospherePath();//气氛道具资源保存路径

设置美颜效果的方法，参考下表

IOS 平台

萌图 SDK 所有资源均支持放在用户自己的服务器，所有资源文件下载完成以后必须解压到指定的沙盒路径下使用以下方法进行设置：

- (void)setResUrl:(NSString *)resUrl;//将资源保存在自定义的网络存储中的情况下，设置网络资源路径

- (NSString *)getResUrl; //获取当前的网络资源路径

// 获取资源及其图片的下载地址

- (NSString *)getStickerUrl; //获取动态特效网络资源路径

- (NSString *)getAtmosphereUrl;//获取气氛特效网络资源路径

- (NSString *)getMaskUrl;//获取道具特效网络资源路径

- (NSString *)getExpressionUrl;//获取表情特效网络资源路径

- (NSString *)getPortraitUrl;//获取人像抠图特效网络资源路径

- (NSString *)getGreenScreenUrl;//获取绿幕抠图特效网络资源路径

// 获取保存资源的沙盒路径

- (NSString *)getStickerPath;//获取动态特效资源沙盒路径

- (NSString *)getAtmospherePath;//获取气氛特效资源沙盒路径

- (NSString *)getMaskPath;//获取道具特效资源沙盒路径

- (NSString *)getExpressionPath;//获取表情特效资源沙盒路径

- (NSString *)getPortraitPath;//获取人像抠图特效资源沙盒路径

- (NSString *)getGreenScreenPath;//获取绿幕抠图特效资源沙盒路径

具体使用方法如下：

[[MtSDK Get] setResUrl:(NSString *)url];

设置美颜效果的方法，参考下表

| 模块 | 功能 | 方法 | 参数 |
| --- | --- | --- | --- |
| 美颜 | 开关 | setFaceBeautyEnable:(BOOL)enable | true:美颜生效;false:美颜失效 |
| 美颜 | 美白 | setWhitenessValue:(int)value | 美白参数，范围[0,100]，0 为没有效果 |
| 美颜 | 磨皮 | setBlurrinessValue:(int)value | 磨皮参数，范围[0,100] |
| 美颜 | 红润 | setRosinessValue:(int)value | 红润参数，范围[0,100] |
| 美颜 | 鲜明 | setClearnessValue:(int)value | 鲜明参数，范围[-50,50] |
| 美颜 | 亮度 | setBrightnessValue:(int)value | 亮度参数，范围[-50,50] |
| 美颜 | 美肤 | setPrecisenessValue:(int)value | 美肤参数，范围[0,100] |
| 美型 | 开关 | setFaceShapeEnable:(BOOL)enable | true:美型生效;false:美型失效 |
| 美型 | 大眼 | setEyeEnlargingValue:(int)value | 大眼参数，范围[0，100] |
| 美型 | 瘦脸 | setCheekThinningValue:(int)value | 瘦脸参数，范围[0，100] |

| | | | |
|---|---|---|---|
| 美型 | 窄脸 | setCheekNarrowingValue:(int)value | 窄脸参数，范围[-50，50] |
| 美型 | 瘦鼻 | setNoseThinningValue:(int)value | 瘦鼻参数，范围[-50，50] |
| 美型 | 长鼻 | setNoseEnlargingValue:(int)value | 长鼻参数，范围[-50，50] |
| 美型 | 额头 | setForeheadTrimmingValue:(int)value | 额头参数，范围[-50，50] |
| 美型 | 下巴 | setChinTrimmingValue:(int)value | 下巴参数，范围[-50，50] |
| 美型 | 嘴型 | setMouthTrimmingValue:(int)value | 嘴型参数，范围[-50，50] |
| 美型 | 眼间距 | setEyeSpacingTrimmingValue:(int)value | 眼间距参数，范围[-50，50] |
| 美型 | 眼角 | setEyeCornerTrimmingValue:(int)value | 眼角参数，范围[-50，50] |
| 美型 | 颧骨 | setCheekboneThinningValue:(int)value | 颧骨参数，范围[0，100],0 为无效果 |
| 美型 | 下颌骨 | setJawboneThinningValue:(int)value | 下颌骨参数，范围[0，100],0 为无效果 |
| 美型 | 太阳穴 | setTempleEnlargingValue:(int)value | 太阳穴参数，范围[0，100],0 为无效果 |
| 美型 | 小头 | setHeadLesseningValue:(int)value | 小头参数，范围[0，100],0 为无效果 |
| 美 | 小脸 | setFaceLesseningValue:(int)value | 小脸参数，范围[0，100],0 为无效果 |

| | | | |
|---|---|---|---|
| 型 | | | |
| 美型 | 人中 | setPhiltrumTrimmingValue:(int)value | 人中参数，范围[-50，50],0 为无效果 |
| 美型 | 开眼角 | setEyeCornerEnlargingValue:(int)value | 开眼角参数，范围[0，100],0 为无效果 |
| 美型 | 鼻头 | setNoseApexLesseningValue:(int)value | 鼻头参数，范围[0，100],0 为无效果 |
| 美型 | 山根 | setNoseRootEnlargingValue:(int)value | 山根参数，范围[0，100],0 为无效果 |
| 美型 | 嘴角 | setMouthSmilingEnlargingValue:(int)value | 嘴角参数，范围[0，100],0 为无效果 |
| 美型 | 黑眼圈 | setDarkCircleRemovingValue:(int)value | 黑眼圈参数，范围[0，100],0 为无效果 |
| 萌图 | 动态 | setDynamicStickerName:(NSString *)name | 切换动态特效，json 文件中选择 |
| 萌图 | 表情 | setExpressionRecreationName:(NSString *)name | 切换表情特效，json 文件中选择 |
| 萌图 | 道具 | setMaskName:(NSString *)name | 切换道具特效，json 文件中选择 |
| 萌图 | 气氛 | setAtmosphereItemName:(NSString *)giftName | 切换气氛特效，json 文件中选择 |
| 萌图 | 水印 | setWatermarkName:(NSString *)name Left:(int)x Top:(int)y Ratio:(int)ratio; | 切换水印特效，json 文件中选择 |
| 萌图 | 人像抠图 | setPortraitName:(NSString *)name | 切换人像抠图特效，json 文件中选择 |

| | | | |
|---|---|---|---|
| 萌图 | 绿幕抠图 | setGreenScreenName:(NSString *)name | 切换绿幕抠图特效，json 文件中选择 |
| 滤镜 | 美颜滤镜 | setBeautyFilterName:(NSString *)name Value:(int)value; | 切换美颜滤镜特效，json 文件中选择 |
| 滤镜 | 特效滤镜 | setEffectFilterType:(MtEffectFilter)type Value:(int)value; | 切换特效滤镜 从 MtEffectFilter 中选择 |
| 滤镜 | 趣味滤镜 | setFunnyFilterType:(MtFunnyFilter)type | 切换趣味滤镜 从 MtEffectFilter 中选择 |
| 滤镜 | 魔法滤镜 | setMagicFilterType:(MtMagicFilter)type | 切换魔法滤镜 从 MtEffectFilter 中选择 |
| 滤镜 | 调色滤镜 | setToneFilterType:(MtToneFilter)type Value:(int)value; | 调色滤镜参数，范围[0，100],0 为无效果 |
| 滤镜 | 一键美颜 | setBeautyStyle:(MtBeautyStyleEnum)style Value:(int)value; | 一键美颜参数，范围[0，100],0 为无效果 |

IOS 平台

萌图 SDK 所有资源均支持放在用户自己的服务器，所有资源文件下载完成以后必须解压到指定的沙盒路径下

使用以下方法进行设置：

- (void)setResUrl:(NSString *)resUrl;//将资源保存在自定义的网络存储中的情况下，设置网络资源路径

- (NSString *)getResUrl; //获取当前的网络资源路径

// 获取资源及其图片的下载地址

- (NSString *)getStickerUrl; //获取动态特效网络资源路径

- (NSString *)getAtmosphereUrl;//获取气氛特效网络资源路径

- (NSString *)getMaskUrl;//获取道具特效网络资源路径

- (NSString *)getExpressionUrl;//获取表情特效网络资源路径

- (NSString *)getPortraitUrl;//获取人像抠图特效网络资源路径

- (NSString *)getGreenScreenUrl;//获取绿幕抠图特效网络资源路径

// 获取保存资源的沙盒路径

- (NSString *)getStickerPath;//获取动态特效资源沙盒路径

- (NSString *)getAtmospherePath;//获取气氛特效资源沙盒路径

- (NSString *)getMaskPath;//获取道具特效资源沙盒路径

- (NSString *)getExpressionPath;//获取表情特效资源沙盒路径

- (NSString *)getPortraitPath;//获取人像抠图特效资源沙盒路径

- (NSString *)getGreenScreenPath;//获取绿幕抠图特效资源沙盒路径

具体使用方法如下:

```
[[MtSDK Get] setResUrl:(NSString *)url];
```

设置美颜效果的方法，参考下表

| 模块 | 功能 | 方法 | 参数 |
| --- | --- | --- | --- |
| 美颜 | 开关 | setFaceBeautyEnable:(BOOL)enable | true:美颜生效;false:美颜失效 |
| 美颜 | 美白 | setWhitenessValue:(int)value | 美白参数，范围[0,100]，0 为没有效果 |
| 美颜 | 磨皮 | setBlurrinessValue:(int)value | 磨皮参数，范围[0,100] |
| 美颜 | 红润 | setRosinessValue:(int)value | 红润参数，范围[0,100] |
| 美颜 | 鲜明 | setClearnessValue:(int)value | 鲜明参数，范围[-50,50] |
| 美颜 | 亮度 | setBrightnessValue:(int)value | 亮度参数，范围[-50,50] |
| 美颜 | 美肤 | setPrecisenessValue:(int)value | 美肤参数，范围[0,100] |

| | | | |
|---|---|---|---|
| 美型 | 开关 | setFaceShapeEnable:(BOOL)enable | true:美型生效;false:美型失效 |
| 美型 | 大眼 | setEyeEnlargingValue:(int)value | 大眼参数，范围[0，100] |
| 美型 | 瘦脸 | setCheekThinningValue:(int)value | 瘦脸参数，范围[0，100] |
| 美型 | 窄脸 | setCheekNarrowingValue:(int)value | 窄脸参数，范围[-50，50] |
| 美型 | 瘦鼻 | setNoseThinningValue:(int)value | 瘦鼻参数，范围[-50，50] |
| 美型 | 长鼻 | setNoseEnlargingValue:(int)value | 长鼻参数，范围[-50，50] |
| 美型 | 额头 | setForeheadTrimmingValue:(int)value | 额头参数，范围[-50，50] |
| 美型 | 下巴 | setChinTrimmingValue:(int)value | 下巴参数，范围[-50，50] |
| 美型 | 嘴型 | setMouthTrimmingValue:(int)value | 嘴型参数，范围[-50，50] |
| 美型 | 眼间距 | setEyeSpacingTrimmingValue:(int)value | 眼间距参数，范围[-50，50] |
| 美型 | 眼角 | setEyeCornerTrimmingValue:(int)value | 眼角参数，范围[-50，50] |
| 美型 | 颧骨 | setCheekboneThinningValue:(int)value | 颧骨参数，范围[0，100],0 为无效果 |
| 美 | 下颌骨 | setJawboneThinningValue:(int)value | 下颌骨参数，范围[0，100],0 为无效果 |

| | | | |
|---|---|---|---|
| 型 | | | |
| 美型 | 太阳穴 | setTempleEnlargingValue:(int)value | 太阳穴参数，范围[0，100],0 为无效果 |
| 美型 | 小头 | setHeadLesseningValue:(int)value | 小头参数，范围[0，100],0 为无效果 |
| 美型 | 小脸 | setFaceLesseningValue:(int)value | 小脸参数，范围[0，100],0 为无效果 |
| 美型 | 人中 | setPhiltrumTrimmingValue:(int)value | 人中参数，范围[-50，50],0 为无效果 |
| 美型 | 开眼角 | setEyeCornerEnlargingValue:(int)value | 开眼角参数，范围[0，100],0 为无效果 |
| 美型 | 鼻头 | setNoseApexLesseningValue:(int)value | 鼻头参数，范围[0，100],0 为无效果 |
| 美型 | 山根 | setNoseRootEnlargingValue:(int)value | 山根参数，范围[0，100],0 为无效果 |
| 美型 | 嘴角 | setMouthSmilingEnlargingValue:(int)value | 嘴角参数，范围[0，100],0 为无效果 |
| 美型 | 黑眼圈 | setDarkCircleRemovingValue:(int)value | 黑眼圈参数，范围[0，100],0 为无效果 |
| 萌图 | 动态 | setDynamicStickerName:(NSString *)name | 切换动态特效，json 文件中选择 |
| 萌图 | 表情 | setExpressionRecreationName:(NSString *)name | 切换表情特效，json 文件中选择 |
| 萌图 | 道具 | setMaskName:(NSString *)name | 切换道具特效，json 文件中选择 |

| | | | |
|---|---|---|---|
| 萌图 | 气氛 | setAtmosphereItemName:(NSString *)giftName | 切换气氛特效，json 文件中选择 |
| 萌图 | 水印 | setWatermarkName:(NSString *)name Left:(int)x Top:(int)y Ratio:(int)ratio; | 切换水印特效，json 文件中选择 |
| 萌图 | 人像抠图 | setPortraitName:(NSString *)name | 切换人像抠图特效，json 文件中选择 |
| 萌图 | 绿幕抠图 | setGreenScreenName:(NSString *)name | 切换绿幕抠图特效，json 文件中选择 |
| 滤镜 | 美颜滤镜 | setBeautyFilterName:(NSString *)name Value:(int)value; | 切换美颜滤镜特效，json 文件中选择 |
| 滤镜 | 特效滤镜 | setEffectFilterType:(MtEffectFilter)type Value:(int)value; | 切换特效滤镜 从 MtEffectFilter 中选择 |
| 滤镜 | 趣味滤镜 | setFunnyFilterType:(MtFunnyFilter)type | 切换趣味滤镜 从 MtEffectFilter 中选择 |
| 滤镜 | 魔法滤镜 | setMagicFilterType:(MtMagicFilter)type | 切换魔法滤镜 从 MtEffectFilter 中选择 |
| 滤镜 | 调色滤镜 | setToneFilterType:(MtToneFilter)type Value:(int)value; | 调色滤镜参数，范围[0，100],0 为无效果 |
| 滤镜 | 一键美颜 | setBeautyStyle:(MtBeautyStyleEnum)style Value:(int)value; | 一键美颜参数，范围[0，100],0 为无效果 |