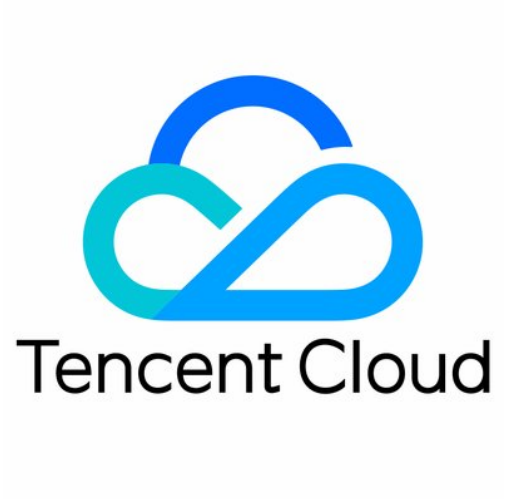
Tencent Effect SDK

機能の実践

製品ドキュメント



カタログ：

機能の実践
SDKパッケージの簡素化
iOS

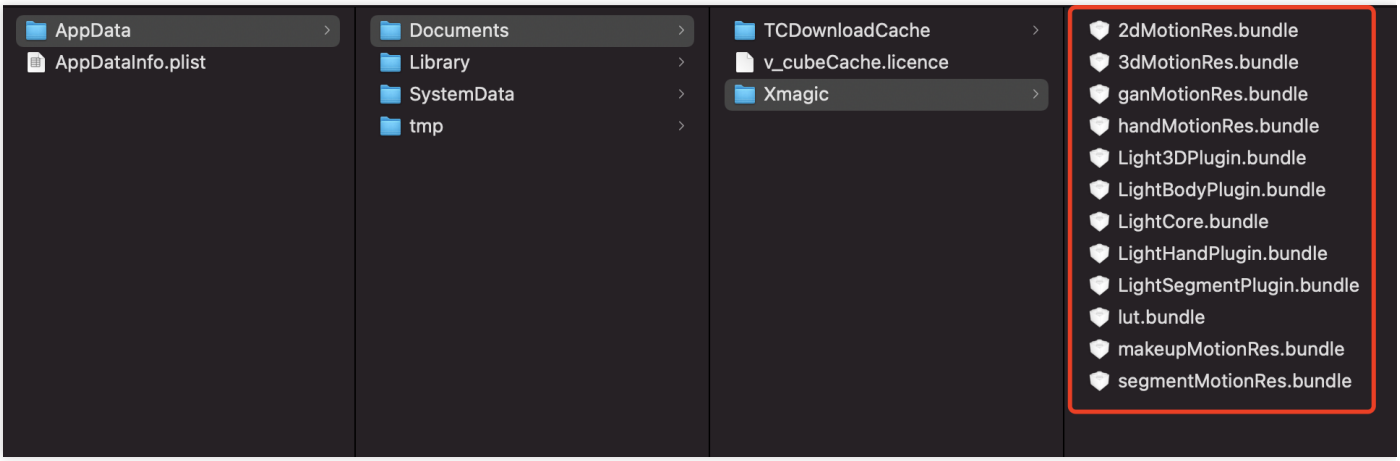
最終更新日：：2022-08-12 15:14:58

リソースのダイナミックダウンロード

パッケージのサイズを抑えるため、SDKが必要とするモデルリソースとモーションリソースMotionRes（一部のベーシック版SDKにはモーションリソースがありません）をオンラインダウンロードに変更してください。ダウンロード後に、上記のファイルのパスをSDKに設定します。

1. 美顔リソースのZIPパッケージをクラウドにアップロードして、ダウンロードURLを生成します。
   例：`https://服务器地址/LightCore.bundle.zip`。

2. プロジェクトの中で生成されたURLからファイルをダウンロードして、サンドボックスに解凍します(例：サンドボックスパス`Document/Xmagic`）。ここまで操作すると、Document/XmagicフォルダーにSDKが必要とするリソースが揃っています。



3. SDKを初期化するとき、root_pathフィールドにステップ2のサンドボックスパスを渡します。

```
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
@"root_path":_filePath ,//_filePathはローカルにダウンロードされた美顔リソースが所在する親ディレクトリ:Ducument/Xmagicです。
@"tnn_"
@"beauty_config":beautyConfigJson
```

```
};
// Init beauty kit @"root_path":Ducument/Xmagic,
self.beautyKit = [[XMagic alloc] initWithRenderSize:_inputSize assetsDict:asset
sDict];
```

4. 美顔パネルの各美顔エフェクトのアイコンを設定し、ダウンロードしたリソースファイルから該当するimageを取得します。

```
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] bu
ndlePath], model.key];
[arrayModels addObject:model];
}else{
if(_useNetResource && _filePath != nil){ //ネット上のリソースを使用する場合
NSString *DirPath = [_filePath stringByAppendingPathComponent:@"2dMotionRes.bun
dle/"]; //美顔リソースの絶対パスを取得します
model.icon = [NSString stringWithFormat:@"%@/%@/template.png", DirPath, model.k
ey];
}else{
model.icon = [NSString stringWithFormat:@"%@/%@/template.png", [[NSBundle mainB
undle] pathForResource:@"2dMotionRes" ofType:@"bundle"], model.key];
}
if ([fileManager fileExistsAtPath:model.icon]) {
[arrayModels addObject:model];
}
}
}
```

5. 美顔エフェクトのパラメータの受け渡しを設定します（詳しくはAPIドキュメントをご参照ください）。

```
/// @brief 美顔の各エフェクトを設定します
/// @param propertyType エフェクトタイプ 文字列：beauty, lut, motion
/// @param propertyName エフェクト名
/// @param propertyValue エフェクトの値
/// @param extraInfo リザーブド拡張、オプショナル設定項目dictあり
/// @return 成功した場合は0、失敗した場合はその他を返します
/// @note 詳細説明
/**
| エフェクトタイプ | エフェクト名 | エフェクトの値 | 説明 | 備考 |
```

```
| :---- | :---- |:---- | :---- | :---- |
| beauty | 美顔id名 | 美顔エフェクトの強度値 |美顔タイプ設定インターフェース | なし |
| lut | フィルターのパス+フィルター名 | フィルターの強度値 | フィルタータイプ設定インターフェース | なし |
| motion | モーションパス名 | モーションパス | モーションタイプ設定インターフェース| **注意**：リソースにzipがある場合、入力モーションパスが書き込み可能なパスであることを確認してください。そうでなければ、appパッケージにパッケージングした場合、手動でunzipしないと使用できません |
**/
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString *_Nonnull)propertyName withData:(NSString*_Nonnull)propertyValue withExtraInfo:(id _Nullable)extraInfo;
```

例

美顔の各エフェクトを設定します

「美顔」と「美ボディ」のエフェクトは処理する必要はありません。SDKの内部では自動的にダウンロードしたリソースファイルを使用します。美顔配下の美白エフェクトを使用することを例として、SDKに渡したパラメーターを以下に示します：

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

この場合、SDKに渡した各パラメーターの値は以下のとおりです：

フィールド	値
propertyType	beauty
propertyName	beauty.whiten
propertyValue	30
extraInfo	nil

フィルターのエフェクトを設定します

keyを処理する必要があります。固有のローカル美顔リソースまたはネットからローカルにダウンロードした美顔リソースを使用できます。

```
NSString *key = [_model.lutIDs[index] path];
if (key != nil) {
key = [@"lut.bundle/" stringByAppendingPathComponent:key];//フィルターエフェクト画像
```

```
の相対パス
}
if(_useNetResource && _filePath != nil){ //ダウンロードした美顔リソースを使用する場合
key = [_filePath stringByAppendingPathComponent:key];//エフェクト画像の絶対パスを生成
します
}
[self.beautyKitRef configPropertyWithType:@"lut" withName:key withData:[NSString
stringWithFormat:@"%f",value] withExtraInfo:nil];
```

フィルター配下のホワイトニングエフェクトを設定します

ローカルリソースとネットワークリソースを使用する時に渡すパラメータの例を以下に示します：

フィールド	ローカルリソースを使用する時に渡すパラメータ	ネットワークリソースを使用する時に渡
propertyType	lut	lut
propertyName	lut.bundle/n_baixi.png	/var/mobile/Containers/Data/A 73F6-4F1B-AEB6- 5EE03A221D18/Documents/Xmagic/
propertyValue	60.000000	60.000000
extraInfo	null	null

モーション、メイク、分割エフェクトを設定します

propertyValueフィールドを処理する必要があります。固有のローカル美顔リソースまたはネットからローカルにダウンロードした美顔リソースを使用できます。

```
NSString *key = [_model.motionIDs[index] key];
NSString *path = [_model.motionIDs[index] path];
NSString *motionRootPath = path==nil?[[NSBundle mainBundle] pathForResource:@"Mot
ionRes" ofType:@"bundle"]:path;
if(_useNetResource && _filePath != nil){ //ダウンロードした美顔リソースを使用する場合
motionRootPath = [_filePath stringByAppendingPathComponent:@"2dMotionRes.bundle"
];//2dMotionResの絶対パスを生成します
}
[self.beautyKitRef configPropertyWithType:@"motion" withName:key withData:motionR
ootPath withExtraInfo:nil];
```

2Dモーション配下の可愛い落書きエフェクトを設定します

ローカルリソースとネットワークリソースを使用する時に渡すパラメータの例を以下に示します：

フィールド	ローカルリソースを使用する時に渡すパラメータ	ネットワークリソ
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/C 73F6-4F1B-AEB 5EE03A221D18/I`
extraInfo	nil	nil

Android

最終更新日：：2022-08-12 15:16:28

パケットのサイズを小さくするために、SDKに必要なassetsリソース、soライブラリ、およびダイナミックエフェクトリソースMotionRes（一部のベーシック版SDKにはダイナミックエフェクトリソースはありません）をネットワークからのダウンロードに変更することができます。ダウンロード完了後、上記ファイルのパスをSDKに設定します。

Demoのダウンロードロジックを再利用することをお勧めします。当然のことながら、既存のダウンロードサービスを使用することもできます。

Demoのダウンロードロジックを再利用する場合は、次の点に注意してください：Demoでは、ブレークポイントからのダウンロード再開機能がデフォルトで有効になっています。これにより、ダウンロードが異常に中断された場合、次回の時に中断された場所から引き続きダウンロードできます。ブレークポイントからのダウンロード再開機能を有効にする場合は、ダウンロードサーバーがブレークポイントからのダウンロード再開機能をサポートしていることを確認してください。

検査方法

サーバーがブレークポイントからのダウンロード再開機能をサポートしているかどうかを判断するには、Webサーバーが`Range`リクエストをサポートしているかどうかを確認してください。テスト方法は、コマンドラインで以下の`curl`コマンドを実行することです：

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

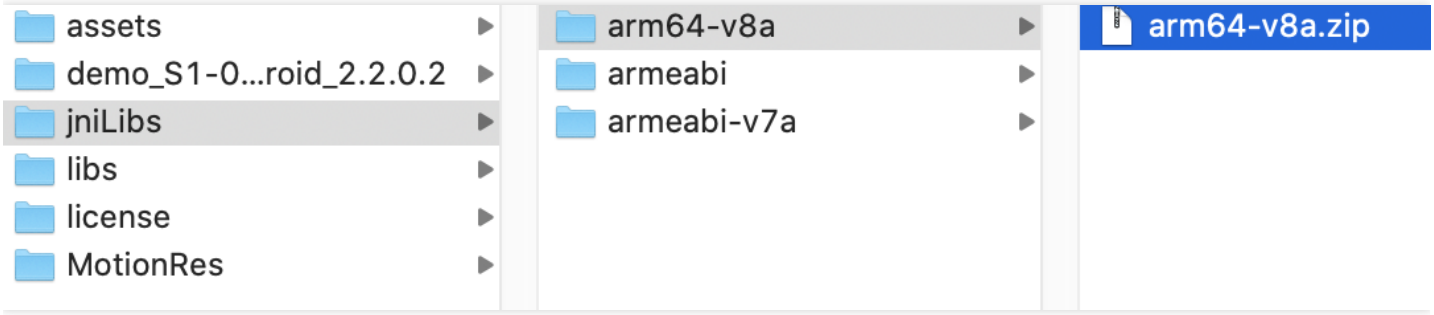
例：

```
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
```

返された内容に`Content-Range`フィールドがある場合は、サーバーがブレークポイントからのダウンロード再開機能をサポートしていることを意味します。

soのダイナミックダウンロード

下図に示すように、so圧縮パッケージは `jniLibs/arm64-v8a` と `jniLibs/armeabi-v7a` にあります：



- Demoでダウンロードサービスを再利用したい場合
- 自分でダウンロードサービスをしたい場合

1. 2つのZIPパッケージのMD5値を計算します。Macでは、コマンドラインで `md5ファイルパス/ファイル名` を使用して直接MD5を計算することができます。または他のツールソフトウェアで計算することもできます。
2. 圧縮パッケージをサーバーにアップロードして、ダウンロードURLを取得します。
3. DemoプロジェクトのResDownloadConfigにある次の定数値を更新します：

```
public class ResDownloadConfig {
    //Directory structure
    //
    //---mylibs.zip(You can give it a custom name)
    //------liblibpag.so
    //------liblight-sdk.so
    //------libv8jni.so
    //
    //   You, A minute ago • Uncommitted changes
    public static final String DOWNLOAD_URL_LIBS_V8A = "https:/
    public static final String DOWNLOAD_URL_LIBS_V7A = "https:/
    //MD5 checksum of the ZIP file
    public static final String DOWNLOAD_MD5_LIBS_V8A = "libs-v8
    public static final String DOWNLOAD_MD5_LIBS_V7A = "libs-v7
```

4. `ResDownloadUtil.checkOrDownloadFiles` を呼び出すことで、ダウンロードを開始します。

注意：

- SDKのバージョン更新により、対応するsoが変更される可能性があります。そのため、これらのsoを再度ダウンロードしてください。Demoのメソッドを参照し、検証にMD5を使用することをお勧めします。
- 自分でsoをダウンロードするか、Demoのダウンロードサービスを再利用するかにかかわらず、SDKのauthインターフェースを呼び出す前に、soがダウンロードされているかどうかを先に確認してください。ResDownloadUtilは、以下の確認方法を提供しています。すでにダウンロードされている場合は、SDKでパスを以下のように設定します：

```
String validLibsDirectory = ResDownloadUtil.getValidLibsDirectory(LaunchActivity.
this,
isCpuV8a() ? ResDownloadConfig.DOWNLOAD_MD5_LIBS_V8A : ResDownloadConfig.DOWNLOAD
_MD5_LIBS_V7A);
if (validLibsDirectory == null) {
Toast.makeText(LaunchActivity.this,"libsがダウンロードされていない場合は先にダウンロードしてください",Toast.LENGTH_LONG).show();
return;
}
XmagicApi.setLibPathAndLoad(validLibsDirectory);
auth();
```

assetsリソースのダイナミックダウンロード

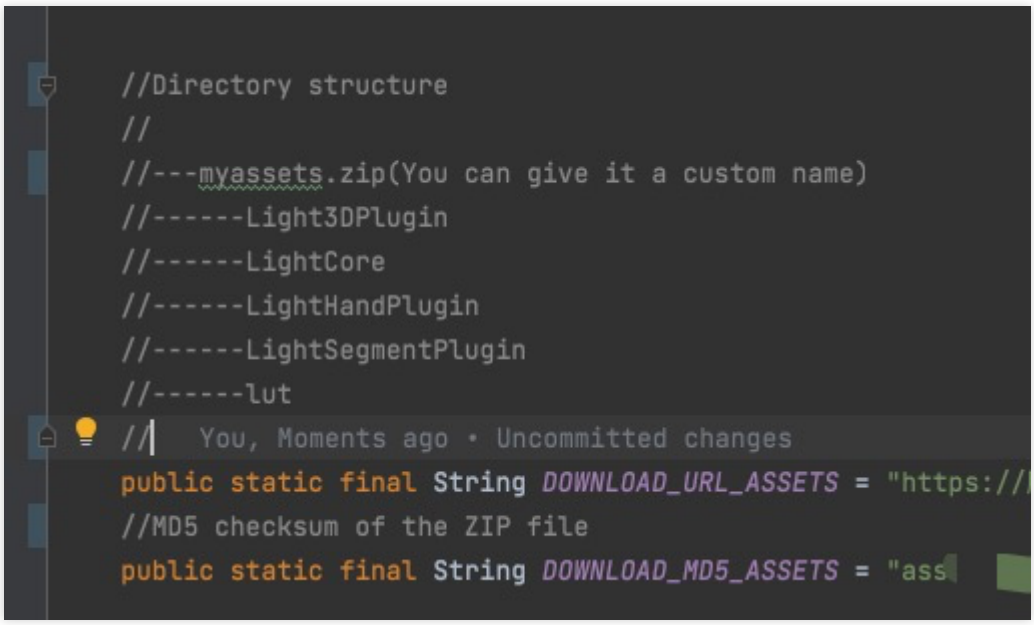
assetsリソースのダイナミックダウンロードについての具体的な操作は以下のとおりです：

1. ローカルプロジェクトのassetsで構成を行います。
   - **2.4.0以降のバージョン**：ローカルassetsのディレクトリにファイルを保存する必要はありません。
   - **2.4.0より前のバージョン**：Licenseファイル と `brand_name.json` 、 `device_config.json` 、 `phone_info.json` 、 `score_phone.json` の4つのJSON構成ファイルを保存してください。

2. SDKでパッケージ化された `download_assets.zip` を見つけます。

3. 上記のsoファイルの処理方法と同じように、このZIPパッケージのMD5を計算し、サーバーにアップロードしてダウンロードアドレスを取得します。
   - Demoでダウンロードサービスを再利用したい場合
   - 自分でダウンロードサービスをしたい場合

   i. 下図に示すダウンロードアドレスとMD5を更新します。



   ii. `ResDownloadUtil.checkOrDownloadFiles` を呼び出してダウンロードを開始し、ResDownloadUtil.getValidAssetsDirectory を呼び出してダウンロード後の`assets`のパスを取得します。詳しい方法については、 LaunchActivity.java`をご参照ください。

注意：

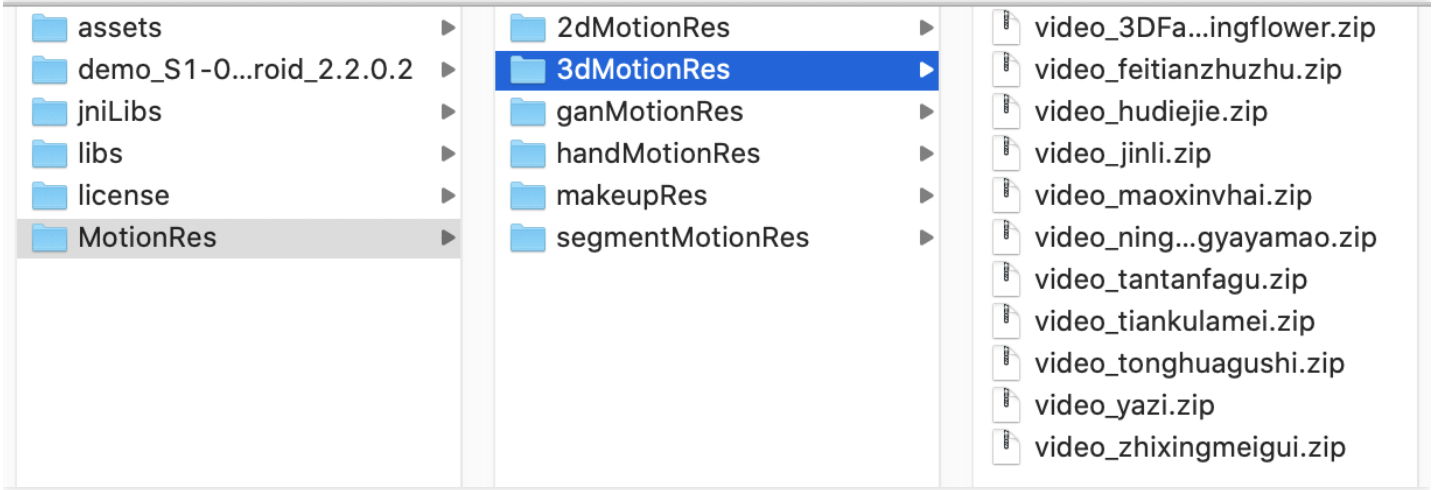
- SDKのバージョン更新により、対応するassetsが変更される可能性があります。互換性を確保するために、これらのassetsを再度ダウンロードしてください。Demoのメソッドを参照し、検証にMD5を使用することをお勧めします。
- 自分でassetsをダウンロードするか、Demoのダウンロードサービスを再利用するかにかかわらず、撮影する前に、assetsがダウンロードされているかどうかを先に確認してください。ResDownloadUtilは、以下の確認方法を提供しています。すでにダウンロードされている場合は、XmagicResParserでパスを設定します。詳しい方法については、 `LaunchActivity.java` をご参照ください。

```
String validAssetsDirectory = ResDownloadUtil.getValidAssetsDirectory(LaunchActiv
ity.this,ResDownloadConfig.DOWNLOAD_MD5_ASSETS);
if (validAssetsDirectory == null) {
Toast.makeText(LaunchActivity.this,"assetsがダウンロードされていない場合は先にダウンロードしてください",Toast.LENGTH_LONG).show();
return;
}
XmagicResParser.setResPath(validAssetsDirectory);
startActivity(intent);
```

ダイナミックエフェクトリソースMotionResのダウンロード

一部のベーシックパッケージにはダイナミックエフェクトリソースがありません。この場合は、このセクションをスキップしてもかまいません。

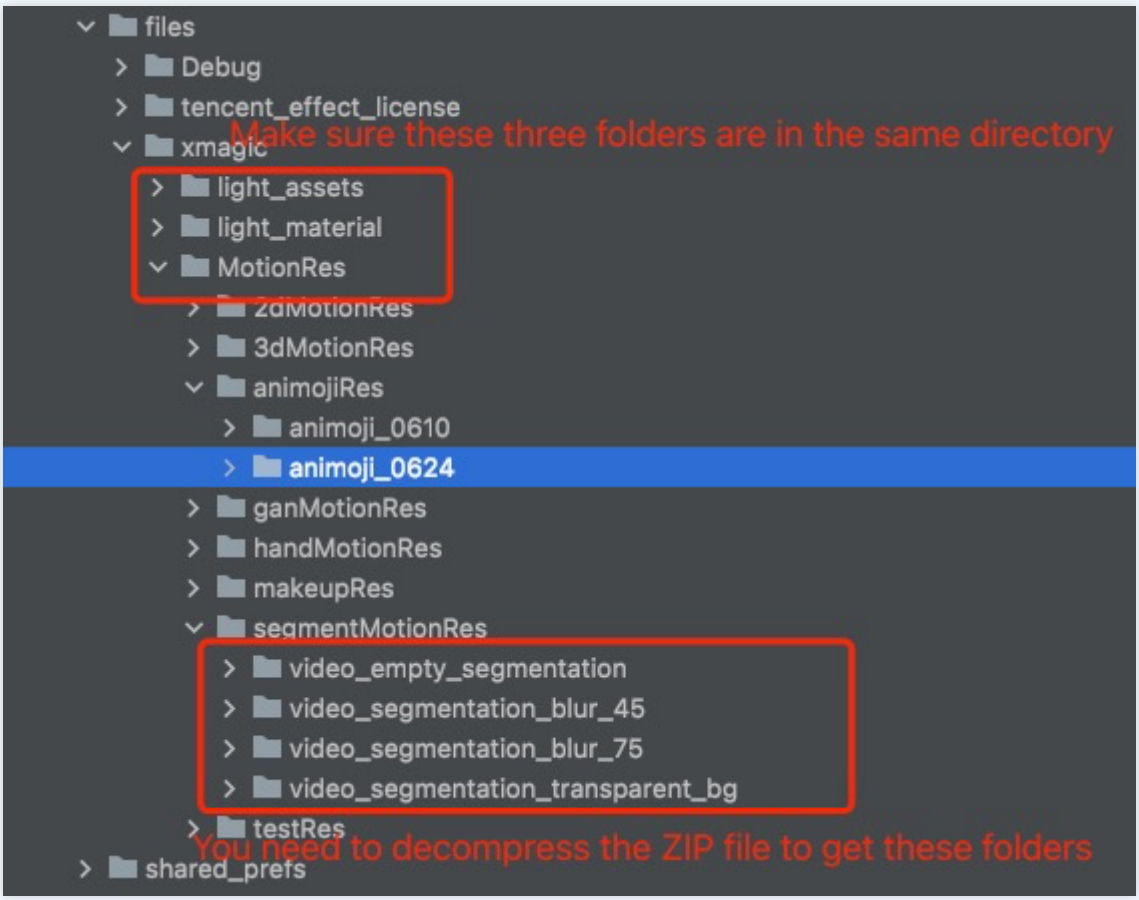
ダイナミックエフェクトは6つのカテゴリーに分類され、それぞれに複数のZIPパッケージがあり、各ZIPパッケージは1種類のダイナミックエフェクトです。購入したパッケージタイプによって、そのファイルの内容は異なります。



ダイナミックエフェクトリソースは、必要に応じて（たとえば、ユーザーが関連する機能ページに入った後、またはダイナミックエフェクトのアイコンをクリックした後など）ダウンロードできます。

これらのZIPパッケージをサーバーにアップロードし、各ZIPパッケージのダウンロードアドレスを取得する必要があります。

注意：
ダイナミックエフェクトリソースのダウンロード後のディレクトリMotionResは、前のセクションのlight_assetsおよびlight_materialと同じレイヤーにある必要があります。また、下図に示すように、ZIPパッケージを直接配置するのではなく、各エフェクトを解凍してください：



MotionResをダウンロードするには、ResDownloadUtil.checkOrDownloadMotionsメソッドをご参照ください。必要に応じて1つずつダウンロードすることをお勧めします。

Demoでダウンロードサービスを再利用したい場合は、ResDownloadConfigのMOTION_RES_DOWNLOAD_PREFIX定数値を自分のダウンロードURLのプレフィックスに置き換えてください。

美顔シーン推奨パラメータ

最終更新日：：2022-07-18 10:06:17

Tencent Effectのユースケースについて、参考までに、次のいくつかのシーンでの機能推奨パラメータを記載します。

方法1：自然な効果

不自然でない程度に美顔を使用したいシーンでは、次のパラメータを参照できます。

機能のタイプ	推奨パラメータ
美白	30
美肌	45
肌色補正	20
デカ目	20
顔やせ（自然オプション）	30

方法2：女神効果

女神のようなエフェクトで美顔を使用したいシーンでは、次のパラメータを参照できます。

機能のタイプ	推奨パラメータ
美白	60
美肌	70
肌色補正	35
デカ目	40
顔やせ（自然オプション）	40
立体（自然オプション）	50
キラキラ目	30

機能のタイプ	推奨パラメータ
小鼻	10

方法3：イケメンエフェクト

男性が自然な美顔を使用したいシーンでは、次のパラメータを参照できます。

機能のタイプ	推奨パラメータ
美白	25
美肌	40
肌色補正	20
顔やせ（イケメンオプション）	40

ショート動画（エンタープライズ版）の移行ガイド

最終更新日：：2022-07-18 10:06:18

現在、ショート動画のエンタープライズ版はサポートが終了しています。そのうち、美顔モジュールはデカップリングおよびアップグレードされ、Tencent Effect SDKになりました。Tencent Effect SDKの美顔効果はより自然であるほか、製品の機能はより強力であり、統合方法はより柔軟です。このドキュメントは、ショート動画のエンタープライズ版をTencent Effect SDK（美顔エフェクト）にアップグレードするための移行ガイドです。

注意事項

1. xmagicモジュールのglideライブラリのバージョン番号を変更して、実際に使用するものと一致させます。
2. xmagicモジュールの最小バージョン番号を変更して、実際に使用するものと一致させます。

統合の手順

手順1：Demoプロジェクトの解凍

1. Tencent Effect TEを統合したUGSV Demoプロジェクトをダウンロードします。このDemoは、Tencent Effect SDK S1-04パッケージに基づいて作成されています。
2. リソースを置き換えます。このDemoプロジェクトで使用されるSDKパッケージは実際のパッケージと同じではない可能性があるため、このDemoの関連SDKファイルを実際に使用するパッケージのSDKファイルに置き換えてください。具体的な操作は以下のとおりです：
   - xmagicモジュールのlibsディレクトリにある `.aar` ファイルを削除し、SDKのlibsディレクトリにある.aar`ファイルをxmagicモジュールのlibsディレクトリにコピーします。
   - xmagicモジュールのassetsディレクトリにあるすべてのファイルを削除し、SDKの `assets/` ディレクトリにあるすべてのリソースをxmagicモジュールの `../src/main/assets` ディレクトリにコピーします。SDKパッケージのMotionResフォルダにリソースがある場合は、このフォルダを `../src/main/assets` ディレクトリにコピーします。
   - xmagicモジュールのjniLibsディレクトリにあるすべての.soファイルを削除し、SDKパッケージのjniLibsで対応する.soファイルを見つけて（SDKのjinLibsフォルダにあるarm64-v8aおよびarmeabi-v7aの.soファイルが圧縮パッケージに存在しているため、先に解凍してください）、xmagicモジュールの `../src/main/jniLibs` ディレクトリにコピーします。
3. Demoプロジェクトのxmagicモジュールを実際のプロジェクトにインポートします。

手順2：SDKバージョンのアップグレード

SDKをEnterprise版からProfessional版にアップグレードします。

- **置換前**： `implementation 'com.tencent.liteav:LiteAVSDK_Enterprise:latest.release'`
- **置換後**： `implementation 'com.tencent.liteav:LiteAVSDK_Professional:latest.release'`

手順3：美顔Licenseの設定

1. プロジェクトのapplicationのoncreateメソッドで以下のメソッドを呼び出します：

```
XMagicImpl.init(this);
XMagicImpl.checkAuth(null);
```

2. XMagicImpl型で、申請したTencent Effectの**License URLとKey**に置き換えます。

手順4：コードの実装

ショート動画のレコーディングインターフェース（TCVideoRecordActivity.java）を例として説明します。

1. `TCVideoRecordActivity.java` 型で、以下の変数コードを追加します。

```
private XMagicImpl mXMagic;
private int isPause = 0;//0：一時停止ではない、1：一時停止、2：一時停止中、3：廃棄が必要
```

2. `TCVideoRecordActivity.java` 型のonCreateメソッドの後に以下のコードを追加します。

```
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()) { //メインスレッドではない
```

```
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", "認証に失敗しました。認証urlおよびkeyを確認してください" + errorCode +
" " + msg);
}
});
```

3. onStopメソッドで以下のコードを追加します：

```
isPause = 1;
if (mXMagic != null) {
mXMagic.onPause();
}
```

4. onDestroyメソッドで以下のコードを追加します：

```
isPause = 3;
XmagicPanelDataManager.getInstance().clearData();
```

5. onActivityResultメソッドの一番前に以下のコードを追加します：

```
if (mXMagic != null) {
mXMagic.onActivityResult(requestCode, resultCode, data);
}
```

6. このデータ型の最後に以下の2つのメソッドを追加します：

```
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();
}
/**
```

- 美顔SDKの初期化
- /

```
private void initXMagic() {
if (mXMagic == null) {
  mXMagic = <span class="hljs-keyword">new</span> XMagicImpl(<span class="hljs-ke
  yword">this</span>, mUGCKitVideoRecord.getBeautyPanel());
}else{
  mXMagic.onResume();
}
}
```

手順5：その他のデータ型の変更

1. AbsVideoRecordUI型のmBeautyPanel型をRelativeLayout型に変更し、getBeautyPanel()メソッドの戻り型をRelativeLayoutに変更します。同時に、対応するXMLの構成を変更し、エラーコードをコメントアウトします。
2. UGCKitVideoRecord型のエラーコードをコメントアウトします。
3. ScrollFilterView型のコードを変更し、mBeautyPanel変数を削除して、エラーコードをコメントアウトします。

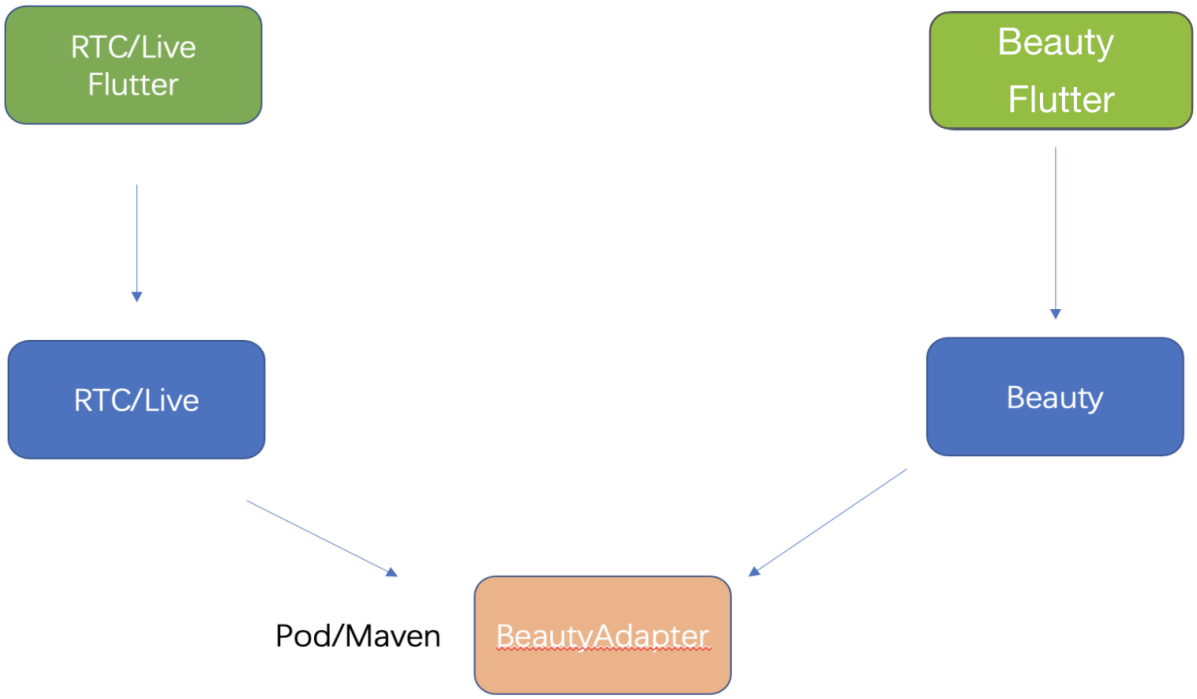
手順6：beautysettingkitモジュールへの依存関係の削除

ugckitモジュールのbuild.gradleファイルで、beautysettingkitモジュールへの依存関係を削除し、プロジェクトをコンパイルして、エラーコードをコメントアウトしてください。

サードパーティプッシュによる美顔の接続（Flutter）

最終更新日：：2022-12-15 11:30:53

Flutter端末のGL環境はネイティブ端末の環境から隔離されているため、Flutterに美顔を接続する際に直接バインド関係を確立することができません。下図のようにネイティブ端末で関係のバインドを行う必要があります。



実現方式の全体フロー

1. 美顔側で1層のインターフェースを抽象化し、美顔側にインターフェースを実装します。
2. アプリケーションの起動時にこのインターフェースをサードパーティプッシュ端末に登録することで、サードパーティプッシュ端末がこのインターフェースによって美顔インスタンスを作成、使用、破棄できるようになります。
3. サードパーティプッシュ端末は美顔の作成および破棄機能を自身のFlutter端末に公開して、お客様が使用できるようにします。
4. 美顔属性の設定は美顔のFlutter SDK機能によって処理することができます。

TRTCの例

美顔側が定義するインターフェース：

```
public interface ITXCustomBeautyProcesserFactory {
/**
* 美顔インスタンスの作成
* @return
*/
ITXCustomBeautyProcesser createCustomBeautyProcesser();
/**
* 美顔インスタンスの破棄（GLスレッドで呼び出す必要があります）
*/
void destroyCustomBeautyProcesser();
}
public interface ITXCustomBeautyProcesser {
//美顔がサポートするビデオフレームのピクセル形式を取得します。美顔がサポートしているのはOpenGL 2Dテクスチャです。
TXCustomBeautyPixelFormat getSupportedPixelFormat();
//美顔がサポートするビデオデータパッケージ形式を取得します。美顔がサポートしているのはV2TXLiveBufferTypeTextureです。テクスチャIDを直接操作でき、パフォーマンスが最良で、画質ロスが最少です。
TXCustomBeautyBufferType getSupportedBufferType();
//GLスレッドで呼び出します（srcFrameにはRGBAテクスチャ、およびwidth、heightが含まれる必要があります）。美顔処理後に処理後のテクスチャオブジェクトをdstFrame内のtexture.textureId内に配置します。
void onProcessVideoFrame(TXCustomBeautyVideoFrame srcFrame, TXCustomBeautyVideoFrame dstFrame);
}
```

1. TRTCで登録メソッドを提供しています。アプリケーションの起動時に、美顔側のITXCustomBeautyProcesserFactoryインターフェースの実装クラス `com.tencent.effect.tencent_effect_flutter.XmagicProcesserFactory` をTRTCに登録しま

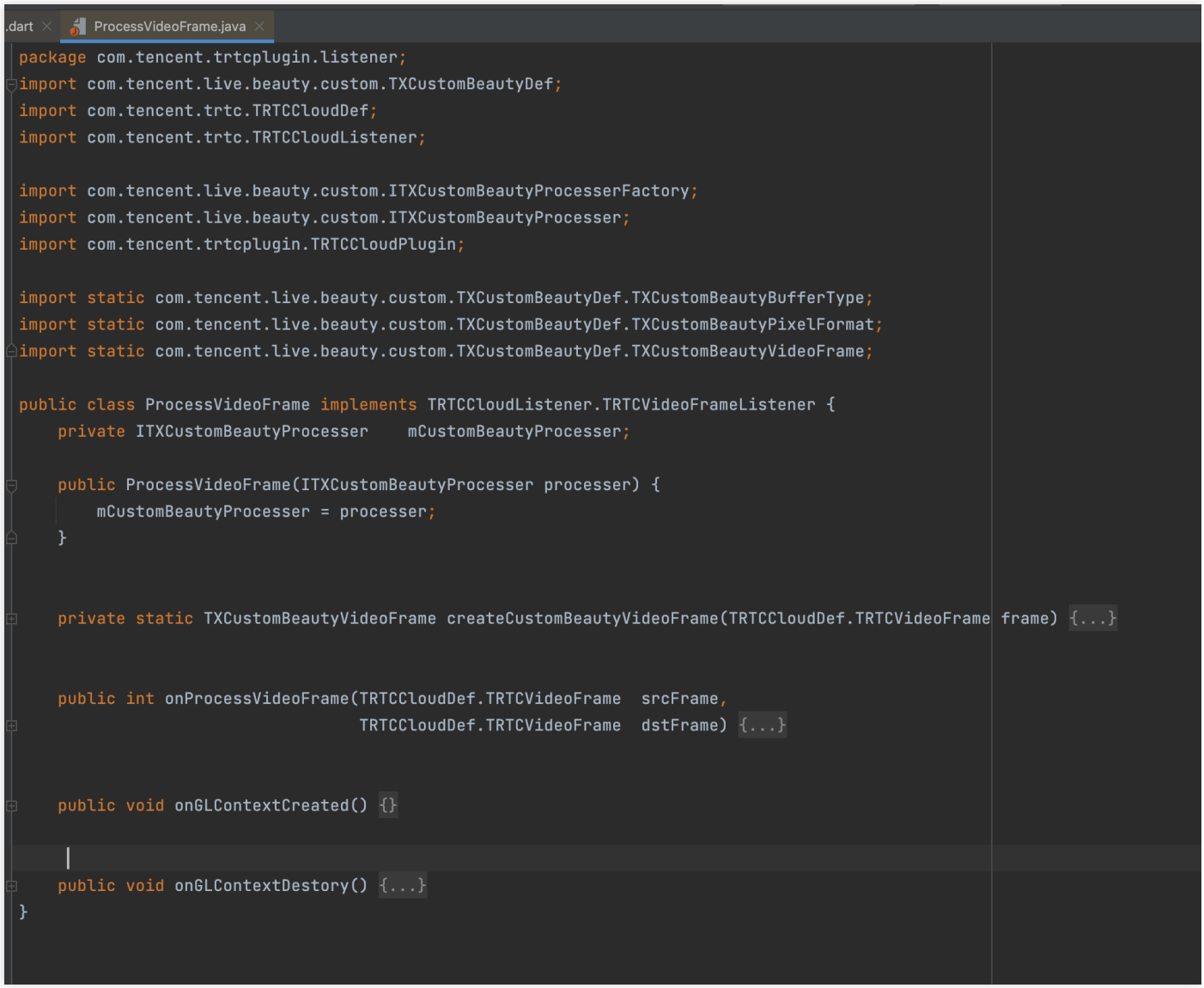
す（ネイティブ端末で行います）。

```
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

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. `Flutter` 層で、カスタム美顔インターフェースの有効化と無効化を行う `Future<v2txlivecode> enableCustomVideoProcess(bool enable)` インターフェースを提供します。
3. TRTCネイティブ端末に美顔オン/オフメソッドを実装します。

```
public void enableCustomVideoProcess(MethodCall call, MethodChannel.Result result) {
    boolean enable = CommonUtil.getParam(call, result, "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(convertTRTCPixelFormat(pixelFormat), convertTRTCBufferType(bufferType), processVideo);
        result.success(ret);
    } else {
        int ret = trtcCloud.setLocalVideoProcessListener(convertTRTCPixelFormat(pixelFormat), convertTRTCBufferType(bufferType), null);
        // processerFactory.destroyCustomBeautyProcesser();
        mCustomBeautyProcesser = null;
        result.success(ret);
    }
}
```

When set to null, the onGlContextDestroy method of the last set processVideo will be called back



```
.dart  ProcessVideoFrame.java
package com.tencent.trtcplugin.listener;
import com.tencent.live.beauty.custom.TXCustomBeautyDef;
import com.tencent.trtc.TRTCCloudDef;
import com.tencent.trtc.TRTCCloudListener;

import com.tencent.live.beauty.custom.ITXCustomBeautyProcesserFactory;
import com.tencent.live.beauty.custom.ITXCustomBeautyProcesser;
import com.tencent.trtcplugin.TRTCCloudPlugin;

import static com.tencent.live.beauty.custom.TXCustomBeautyDef.TXCustomBeautyBufferType;
import static com.tencent.live.beauty.custom.TXCustomBeautyDef.TXCustomBeautyPixelFormat;
import static com.tencent.live.beauty.custom.TXCustomBeautyDef.TXCustomBeautyVideoFrame;

public class ProcessVideoFrame implements TRTCCloudListener.TRTCVideoFrameListener {
    private ITXCustomBeautyProcesser    mCustomBeautyProcesser;

    public ProcessVideoFrame(ITXCustomBeautyProcesser processer) {
        mCustomBeautyProcesser = processer;
    }


    private static TXCustomBeautyVideoFrame createCustomBeautyVideoFrame(TRTCCloudDef.TRTCVideoFrame frame) {...}


    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame  srcFrame,
                                   TRTCCloudDef.TRTCVideoFrame  dstFrame) {...}


    public void onGLContextCreated() {}

    public void onGLContextDestory() {...}
}
```

付録

美顔が提供する抽象化層の依存

```
///
implementation 'com.tencent.liteav:custom-video-processor:latest.release'
```