短视频 SDK

不含 UI 集成方案

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

版权所有：腾讯云计算（北京）有限责任公司

不含 UI 集成方案
SDK 集成
SDK 集成(XCode)

最近更新时间：2022-11-14 18:18:58

支持平台

SDK 支持 iOS 8.0 以上系统。

开发环境

- Xcode 9 或更高版本。
- iOS12.0及以上系统。

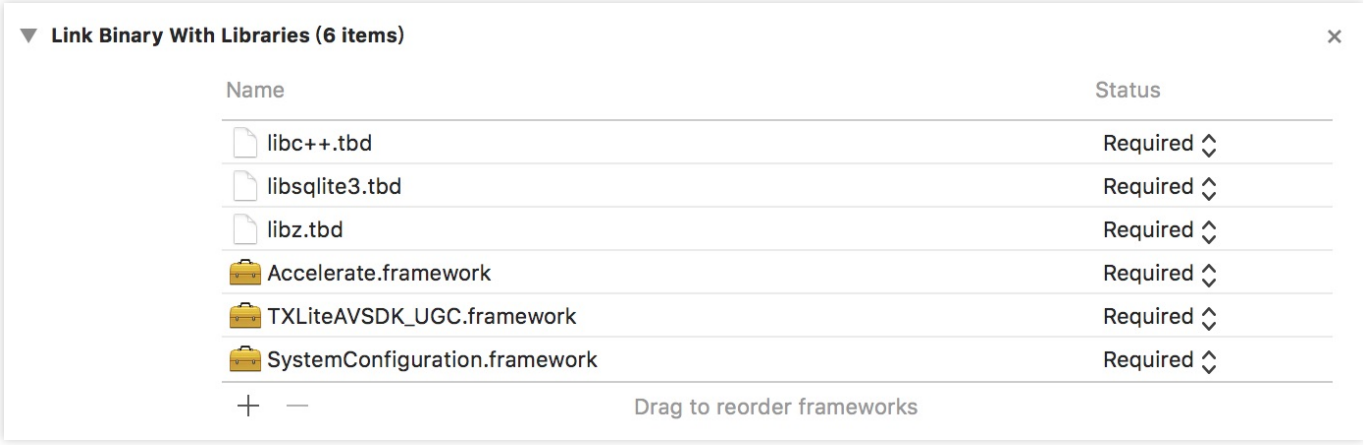
设置步骤

步骤1：链接 SDK 及系统库

将下载的 SDK 资源包解压，并将 SDK 文件夹中 TXLiteAVSDK_ 开头的 framework（如 TXLiteAVSDK_UGC.framework）复制到工程所在文件夹，并拖动到工程当中。

1. 按照 SDK 的版本，集成对应的库：
   - TXLiteAVSDK 9.5 以下添加集成库
   - TXLiteAVSDK 10.0以上添加系统库
   i. 选中工程的 Target，添加以下系统库：
      - Accelerate.framework
      - SystemConfiguration.framework
      - libc++.tbd
      - libsqlite3.tbd

ii. 添加完毕后，工程库依赖如下图所示：



2. 选中工程的 Target，在 **Build Settings** 中搜索 `bitcode`，将 **Enable Bitcode** 设置为 **NO**。

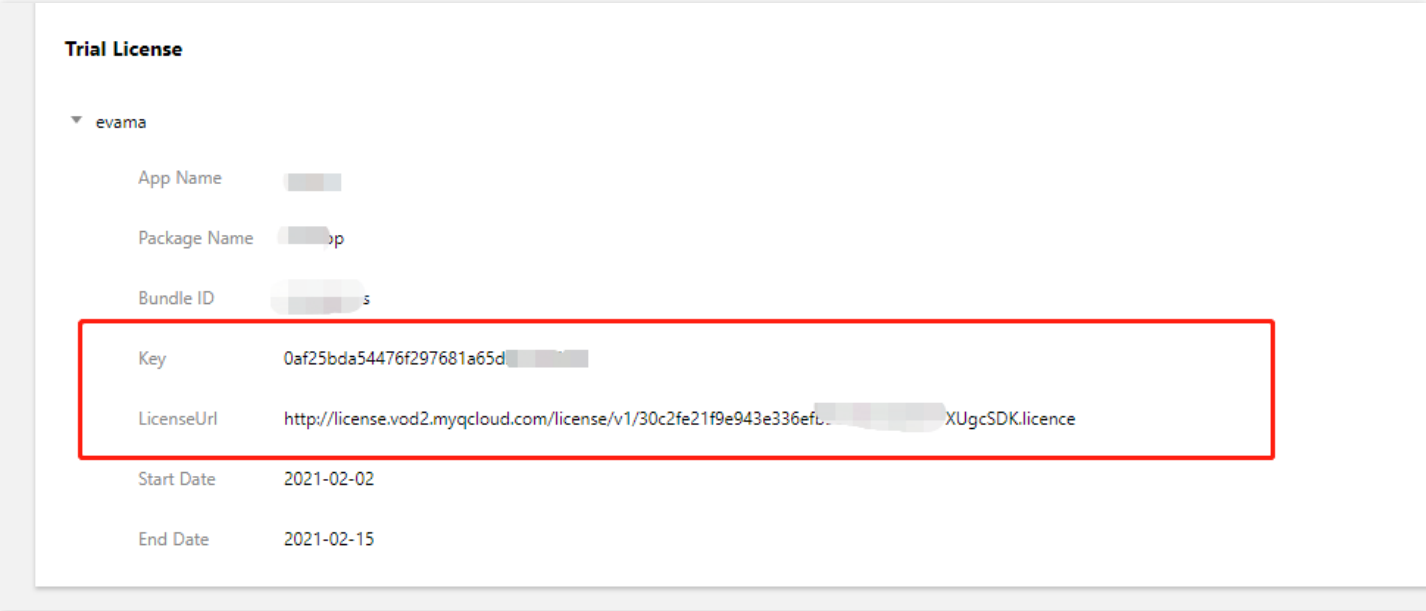
步骤2：配置 App 权限

应用会需要相册及相册的访问权限，需要在 Info.plist 中添加对应项，可以通过在 Info.plist 中右键选 Open as / Source Code 粘贴并修改以下内容进行配置。

```
<key>NSAppleMusicUsageDescription</key>
<string>视频云工具包需要访问您的媒体库权限以获取音乐，不允许则无法添加音乐</string>
<key>NSCameraUsageDescription</key>
<string>视频云工具包需要访问您的相机权限，开启后录制的视频才会有画面</string>
<key>NSMicrophoneUsageDescription</key>
<string>视频云工具包需要访问您的麦克风权限，开启后录制的视频才会有声音</string>
<key>NSPhotoLibraryAddUsageDescription</key>
<string>视频云工具包需要访问您的相册权限，开启后才能保存编辑的文件</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>视频云工具包需要访问您的相册权限，开启后才能编辑视频文件</string>
```

步骤3：SDK License 设置与基本信息获取

通过 License 申请 的指引申请 License 后，从 控制台 复制 key 和 url，见下图。



2. 在您的应用中使用短视频功能之前，建议在 `- [AppDelegate application:didFinishLaunchingWithOptions:]` 中进行如下设置：

```
@import TXLiteAVSDK_UGC;
@implementation AppDelegate
- (BOOL)application:(UIApplication*)applicationdidFinishLaunchingWithOptions:(NSD
ictinoary*)options {
NSString * const licenceURL = @"<获取到的licnseUrl>";
NSString * const licenceKey = @"<获取到的key>";
[TXUGCBase setLicenceURL:licenceURL key:licenceKey];
NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

说明：
对于使用**4.7版本 License 的用户**，如果您升级了 SDK 到4.9版本，您可以登录控制台，单击下图的 **切换到新版License** 生成对应的 key 和 url，切换后的 License 必须使用4.9及更高的版本，切换后按照上述操作集成即可。

步骤4：Log 配置

在 TXLiveBase 中可以设置 log 是否在控制台打印以及 log 的级别，相关接口如下：

- **setConsoleEnabled**
  设置是否在 xcode 的控制台打印 SDK 的相关输出。
- **setLogLevel**
  设置是否允许 SDK 打印本地 log，SDK 默认会将 log 写到当前 App 的 **Documents/logs** 文件夹下。
  如果您需要我们的技术支持，建议将此开关打开，在重现问题后提供 log 文件，非常感谢您的支持。
- **Log 文件的查看**
  小直播 SDK 为了减少 log 的存储体积，对本地存储的 log 文件做了加密，并且限制了 log 数量的大小，所以要查看 log 的文本内容，需要使用 log 解压缩工具。

```
[TXLiveBase setConsoleEnabled:YES];
[TXLiveBase setLogLevel:LOGLEVEL_DEBUG];
```

步骤5：编译运行

如果前面各步骤都操作正确的话，HelloSDK 工程就可以顺利编译通过。在 Debug 模式下运行 App，Xcode 的 Console 窗格会打印出 SDK 的版本信息：

```
2017-09-26 16:16:15.767 HelloSDK[17929:7488566] SDK Version = 5.2.5541
```

快速接入功能模块

为了方便您快速集成 SDK 各项功能，我们提供了 UGCKit。UGCKit 是在短视频 SDK 基础上构建的一套 UI 组件库。

您可以通过 GitHub 或 资源下载 中提供的 SDK 压缩包获取 UGCKit。UGCKit 位于压缩包 Demo/TXLiteAVDemo/UGC/UGCKit 目录下。

UGCKit 开发环境要求

- Xcode 10 及以上。
- iOS 9.0 及以上。

步骤1：集成 UGCKit

1. 项目中使用 cocoapods，根据实际情况选择其中一种操作：

- 在项目根目录，执行 `pod init && pod install`，可得到 Podfile 文件。
- 把 **UGCKit** 文件夹拷贝到项目根目录下（Podfile 同级目录）。

2. 打开 Podfile 文件，增加：

```
pod 'UGCKit', :path => 'UGCKit/UGCKit.podspec', :subspecs => ["UGC"] #subspecs
根据SDK来选择
```

3. 如果集成基础美颜，把**BeautySettingKit**文件夹拷贝到项目根目录下（Podfile 同级目录），并且在Podfile文件中，增加：

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

4. 如果集成腾讯特效，把**xmagickit**文件夹拷贝到项目根目录下（Podfile 同级目录），并且在Podfile文件中增加：

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

5. 执行 **pod install**，并打开 `项目名.xcworkspace` ，可以看到在 `Pods/Development Pods` 目录下已有 `UGCKit BeautySettingKit x magickit` 。

步骤2：使用 UGCKit

1. **录制**

`UGCKitRecordViewController` 提供了完整的录制功能，您只需实例化这个控制器后展现在界面中即可。

```
UGCKitRecordViewController *recordViewController = [[UGCKitRecordViewController
alloc] initWithConfig:nil theme:nil];
[self.navigationController pushViewController:recordViewController]
```

录制后的结果将通过 completion block 回调，示例如下：

```
recordViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
// 录制出错
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// 用户取消录制，退出录制界面
[self.navigationController popViewControllerAnimated:YES];
} else {
// 录制成功，用结果进行下一步处理
[self processRecordedVideo:result.media];
}
}
};
```

2. **编辑**

`UGCKitEditViewController` 提供了完整的图片转场和视频编辑功能，实例化时需要传入待编辑的媒体对象，以处理录制结果为例，示例如下：

```
- (void)processRecordedVideo:(UGCKitMedia *)media {
// 实例化编辑控制器
UGCKitEditViewController *editViewController = [[UKEditViewController alloc] in
itWithMedia:media conifg:nil theme:nil];
// 展示编辑控制器
[self.navigationController pushViewController:editViewController animated:YES];
```

编辑后的结果将通过 completion block 回调，示例如下：

```
editViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
// 出错
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// 用户取消录制，退出编辑界面
[self.navigationController popViewControllerAnimated:YES];
} else {
// 编辑保存成功，用结果进行下一步处理
[self processEditedVideo:result.path];
}
}
}
```

3. **从相册中选择视频或图片**

i. `UGCKitMediaPickerViewController` 用来处理媒体的选择与合并，当选择多个视频时，将会返回拼接后的视频，示例如下：

```
// 初始化配置
UGCKitMediaPickerConfig *config = [[UGCKitMediaPickerConfig alloc] init];
config.mediaType = UGCKitMediaTypeVideo;//选择视频
config.maxItemCount = 5; //最多选5个
// 实例化媒体选择器
UGCKitMediaPickerViewController *mediaPickerViewController = [[UGCKitMediaPic
kerViewController alloc] initWithConfig:config theme:nil];
// 展示媒体选择器
```

版权所有：腾讯云计算（北京）有限责任公司

```
[self presentViewController:mediaPickerViewController animated:YES completio
n:nil];
```

ii. 选择的结果将通过 completion block 回调，示例如下：

```
mediaPickerViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
// 出错
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// 用户取消录制，退出选择器界面
[self dismissViewControllerAnimated:YES completion:nil];
} else {
// 编辑保存成功，用结果进行下一步处理
[self processEditedVideo:result.media];
}
}
}
```

4. **裁剪**

`UGCKitCutViewController` 提供视频的裁剪功能，与编辑接口相同，在实例化时传入媒体对象，在 completion 中处理剪辑结果即可。示例如下：

```
UGCKitMedia *media = [UGCKitMedia mediaWithVideoPath:@"<#视频路径#>"];
UGCKitCutViewController *cutViewController = [[UGCKitCutViewController alloc] i
nitWithMedia:media theme:nil];
cutViewController.completion = ^(UGCKitResult *result) {
if (!result.cancelled && !result.error) {
[self editVideo:result.media];
} else {
[self.navigationController popViewControllerAnimated:YES];
}
}
[self.navigationController pushViewController: cutViewController]
```

详细介绍

以下为 SDK 各模块的详细说明：

- 视频录制

- 视频编辑
- 视频拼接
- 视频上传
- 视频播放

版权所有：腾讯云计算（北京）有限责任公司

SDK 集成(Android Studio)

最近更新时间：2022-11-15 10:14:52

Android 工程配置

系统要求

SDK建议使用Android 5.0（API Level 21）及以上系统

开发环境

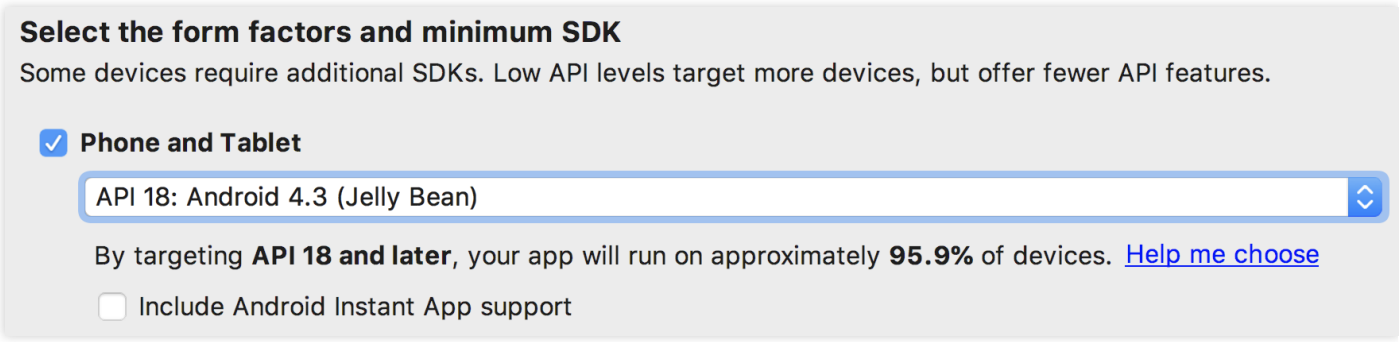
以下是 SDK 的开发环境，App 开发环境不需要与 SDK 一致，但要保证兼容：

- Android NDK：android-ndk-r12b
- Android SDK Tools：android-sdk_25.0.2
- minSdkVersion：21
- targetSdkVersion：26
- Android Studio（推荐您使用 Android Studio）

步骤1：集成 SDK

- aar 方式集成
- jar+so 方式集成
- gradle 集成方式

1. **新建工程**



2. **工程配置**
   - 在工程 App 目录下的 build.gradle 中，添加引用 aar 包的代码：

```
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
// 导入短视频 SDK aar，LiteAVSDK_UGC_x.y.zzzz 请自行修改为最新版本号
```

```
compile(name: 'LiteAVSDK_UGC_10.7.1136', ext: 'aar')
...
}
```

- 在工程目录下的 build.gradle 中，添加 flatDir，指定本地仓库：

```
allprojects {
repositories {
jcenter()
flatDir {
dirs 'libs'
}
}
}
```

- 在 App 工程目录下的 build.gradle 的 defaultConfig 里面，指定 ndk 兼容的架构：

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

- 最后单击 **Sync Now**，编译工程。

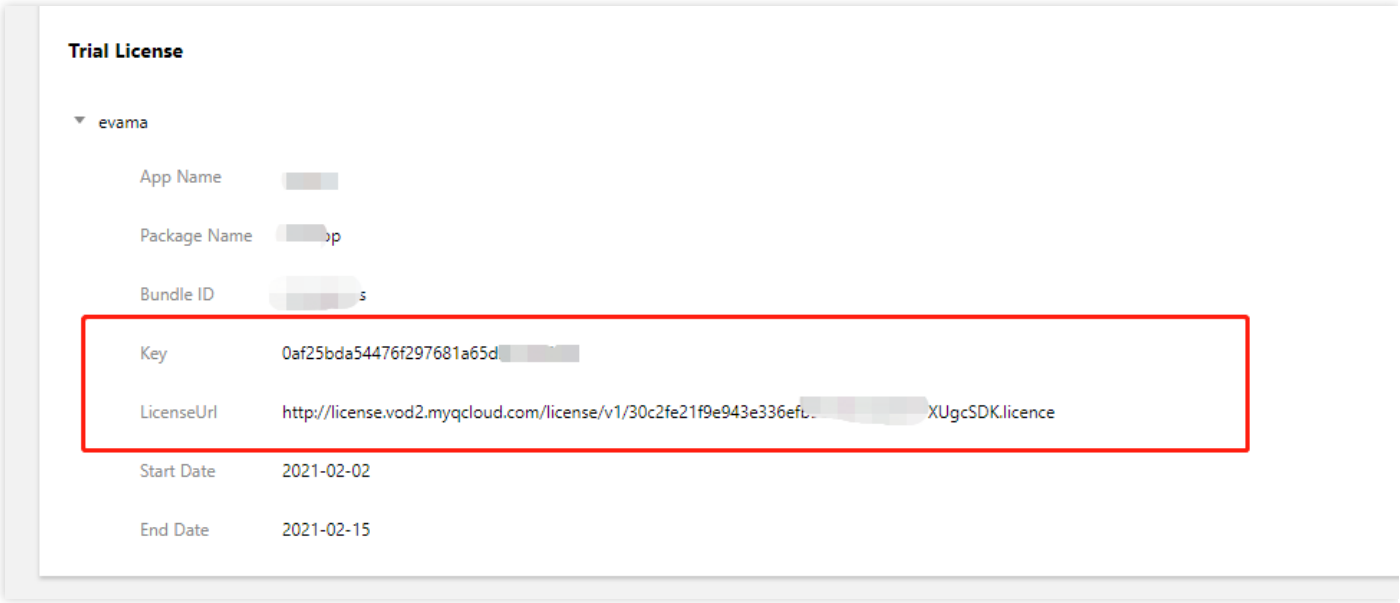
步骤2：配置 App 权限

在 AndroidManifest.xml 中配置 App 的权限，音视频类 App 一般需要以下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.Camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

步骤3：设置 License

1. 申请 License 后，从 云点播控制台 复制 License Key 和 License URL，如下图所示：

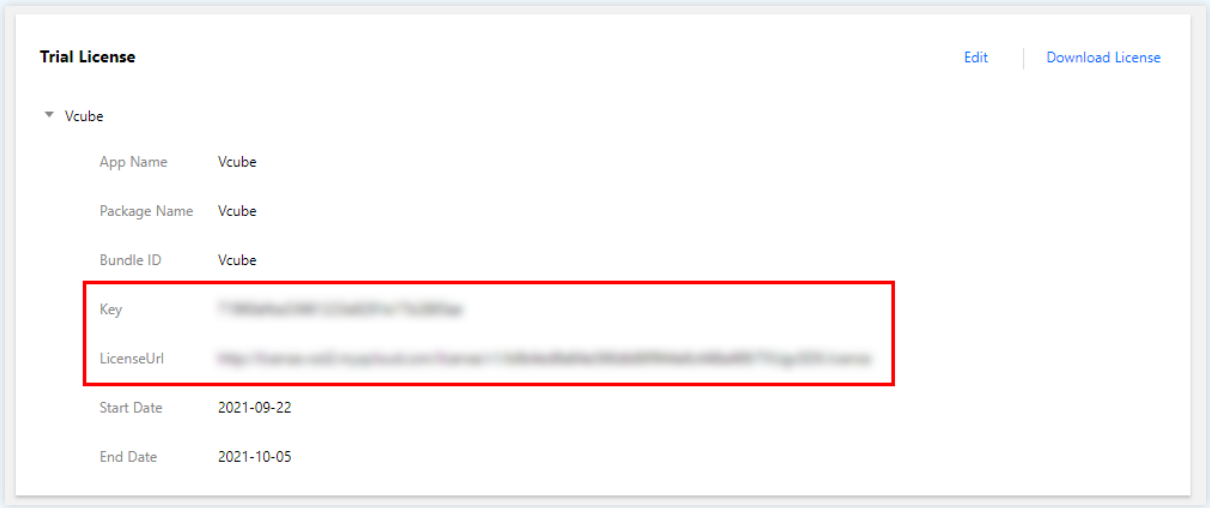


2. 在您的应用中使用短视频功能之前，建议在 `- Application onCreate()` 中进行如下设置：

```
public class DemoApplication extends Application {
String ugcLicenceUrl = ""; // 填入您从控制台申请的 licence url
String ugcKey = ""; // 填入您从控制台申请的 licence key
@Override
public void onCreate() {
super.onCreate();
TXUGCBase.getInstance().setLicence(instance, ugcLicenceUrl, ugcKey);
}
}
```

说明：
对于使用4.7版本 License 的用户，如果您升级了 SDK 到4.9版本，您可以登录控制台，单击下图的**切换到新版 License** 按钮生成对应的 License Key 和 License URL，切换后的 License 必须使用4.9及更高的版本，切

换后按照上述操作集成即可。



步骤4：打印 log

在 TXLiveBase 中可以设置 log 是否在控制台打印以及 log 的级别，具体代码如下：

- **setConsoleEnabled**
  设置是否在 Android Studio 的控制台打印 SDK 的相关输出。
- **setLogLevel**
  设置是否允许 SDK 打印本地 log，SDK 默认会将 log 写到 sdcard 上，**Android/data/应用包名/files/log/tencent/liteav** 文件夹下。如果您需要我们的技术支持，建议将此开关打开，在重现问题后提供 log 文件，非常感谢您的支持。

```
TXLiveBase.setConsoleEnabled(true);
TXLiveBase.setLogLevel(TXLiveConstants.LOG_LEVEL_DEBUG);
```

步骤5：编译运行

在工程中调用 SDK 接口，获取 SDK 版本信息，以验证工程配置是否正确。

1. **引用 SDK**：
   在 MainActivity.java 中引用 SDK 的 class：

```
import com.tencent.rtmp.TXLiveBase;
```

2. **调用接口**：
   在 onCreate 中调用 getSDKVersioin 接口获取版本号：

```
String sdkver = TXLiveBase.getSDKVersionStr();
Log.d("liteavsdk", "liteav sdk version is : " + sdkver);
```

3. **编译运行**：
   如果前面各步骤都操作正确，Demo工程将顺利编译通过，运行之后将在 logcat 中看到如下 log 信息：
   `09-26 19:30:36.547 19577-19577/ D/liteavsdk: liteav sdk version is : 7.4.9211`

问题排查

如果您将 SDK 导入到您的工程，编译运行出现类似以下错误：

```
Caused by: android.view.InflateException:
Binary XML file #14:Error inflating class com.tencent.rtmp.ui.TXCloudVideoView
```

可以按照以下流程来排查问题：

1. 确认是否已经将 SDK 中的 jar 包和 so 库放在 jniLibs目录下。

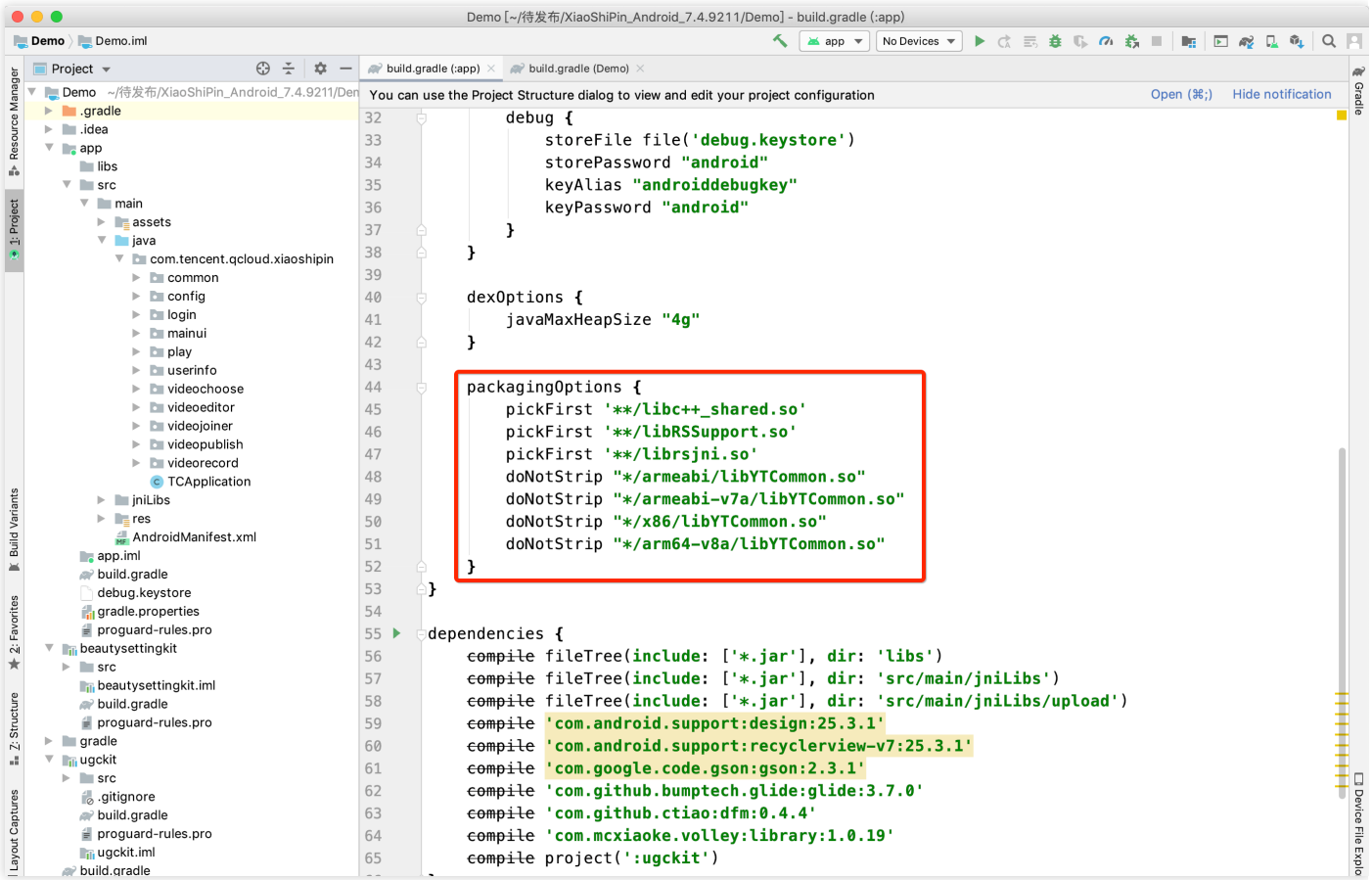
2. 如果您使用 aar 集成方式的完整版本，在工程目录下的 build.gradle 的 defaultConfig 里面确认下是否将 x64 架构的 so 库过滤掉。

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

3. 检查下混淆规则，确认已将 SDK 的相关包名加入了不混淆名单。

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

4. 配置 App 打包参数。



快速接入短视频功能模块

下述内容主要讲解如何在已有的项目中快速集成短视频 SDK，完成从录制，编辑，合成的完整过程。文中所需要的代码及资源文件均在 资源下载 中 SDK 的压缩包中以及 短视频 Demo 提供。

集成 UGCKit

1. 新建工程（Empty Activity）

i. 创建一个空的 Android Studio 工程，工程名可以为 `ugc`，包名可自定义，保证新建的空工程编译通过。

ii. 配置 Project 的 `build.gradle`。

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
repositories {
```

```
google()
jcenter()
}
dependencies {
# 拷贝开始
classpath 'com.android.tools.build:gradle:3.6.1'
# 拷贝结束
// NOTE: Do not place your application dependencies here; they belong
// in the individual module build.gradle files
}
}
allprojects {
repositories {
google()
jcenter()
# 拷贝开始
flatDir {
dirs 'src/main/jniLibs'
dirs project(':ugckit').file('libs')
}
# 拷贝结束
jcenter() // Warning: this repository is going to shut down soon
}
}
task clean(type: Delete) {
delete rootProject.buildDir
}
# 拷贝开始
ext {
compileSdkVersion = 29
buildToolsVersion = "29.0.2"
supportSdkVersion = "26.1.0"
minSdkVersion = 21
targetSdkVersion = 26
versionCode = 1
versionName = "v1.1"
proguard = true
rootPrj = "$projectDir/.."
ndkAbi = 'armeabi-v7a'
liteavSdk = "com.tencent.liteav:LiteAVSDK_UGC:latest.release"
}
# 拷贝结束
```

iii. 配置 app 的 build.gradle 。

```
plugins {
id 'com.android.application'
}
android {
# 拷贝开始
compileSdkVersion = rootProject.ext.compileSdkVersion
buildToolsVersion = rootProject.ext.buildToolsVersion
# 拷贝结束
defaultConfig {
applicationId "com.yunxiao.dev.liteavdemo"
# 拷贝开始
minSdkVersion rootProject.ext.minSdkVersion
targetSdkVersion rootProject.ext.targetSdkVersion
versionCode rootProject.ext.versionCode
versionName rootProject.ext.versionName
renderscriptTargetApi = 19
renderscriptSupportModeEnabled = true
multiDexEnabled = true
ndk {
abiFilters rootProject.ext.ndkAbi
}
# 拷贝结束
testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
}
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'progu
ard-rules.pro'
}
}
compileOptions {
sourceCompatibility JavaVersion.VERSION_1_8
targetCompatibility JavaVersion.VERSION_1_8
}
}
dependencies {
# 拷贝开始
implementation fileTree(include: ['*.jar'], dir: 'libs')
implementation 'com.google.android.material:material:1.0.0'
implementation 'androidx.recyclerview:recyclerview:1.0.0'
implementation 'com.google.code.gson:gson:2.3.1'
implementation 'com.tencent.rqd:crashreport:3.4.4'
implementation 'com.tencent.rqd:nativecrashreport:3.9.2'
implementation 'com.github.castorflex.verticalviewpager:library:19.0.1'
implementation 'com.squareup.okhttp3:okhttp:3.11.0'
```

版权所有：腾讯云计算（北京）有限责任公司

```
implementation 'de.hdodenhof:circleimageview:3.1.0'
implementation rootProject.ext.liteavSdk
implementation project(':ugckit')
implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
implementation('com.blankj:utilcode:1.25.9', {
exclude group: 'com.google.code.gson', module: 'gson'
})
# 拷贝结束
}
```

iv. 配置 Gradle 版本：

```
distributionUrl=https\://services.gradle.org/distributions/gradle-5.6.4-bin.zip
```

2. **导入相关 module**

i. 拷贝 `ugckit module` 到 您新建的工程 ugc 目录下。

ii. 如何集成基础美颜，拷贝 `beautysettingkit module` 到您新建的工程 ugc 目录下。

iii. 如何集成腾讯特效，拷贝 `xmagickit module` 到您新建的工程 ugc 目录下，参考文档。

iv. 在工程的 `settings.gradle` 中导入 ugckit。

v. 在新建的工程 `UGC/settings.gradle` 下指明引入这几个 module：

```
include ':ugckit'
include ':beautysettingkit'
include ':xmagickit'
```

vi. 在工程 app module 中依赖 UGCKit module：

```
implementation project(':ugckit')
```

3. **申请 Licence**

在使用 UGCKit 之前要先设置 License。

实现录制、导入、裁剪、特效编辑功能

1. 设置 Licence，初始化 UGCKit

在您使用短视频功能之前尽可能早的设置 Licence，初始化 UGCKit。

```
// 设置Licence
TXUGCBase.getInstance().setLicence(this, ugcLicenceUrl, ugcKey);
// 初始化UGCKit
UGCKit.init(this);
```

2. 视频录制

i. 新建录制 xml， 加入如下配置：

```
<com.tencent.qcloud.ugckit.UGCKitVideoRecord
android:id="@+id/video_record_layout"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

ii. 在 `res/values/styles.xml` 中新建空的录制主题，继承 UGCKit 默认录制主题。

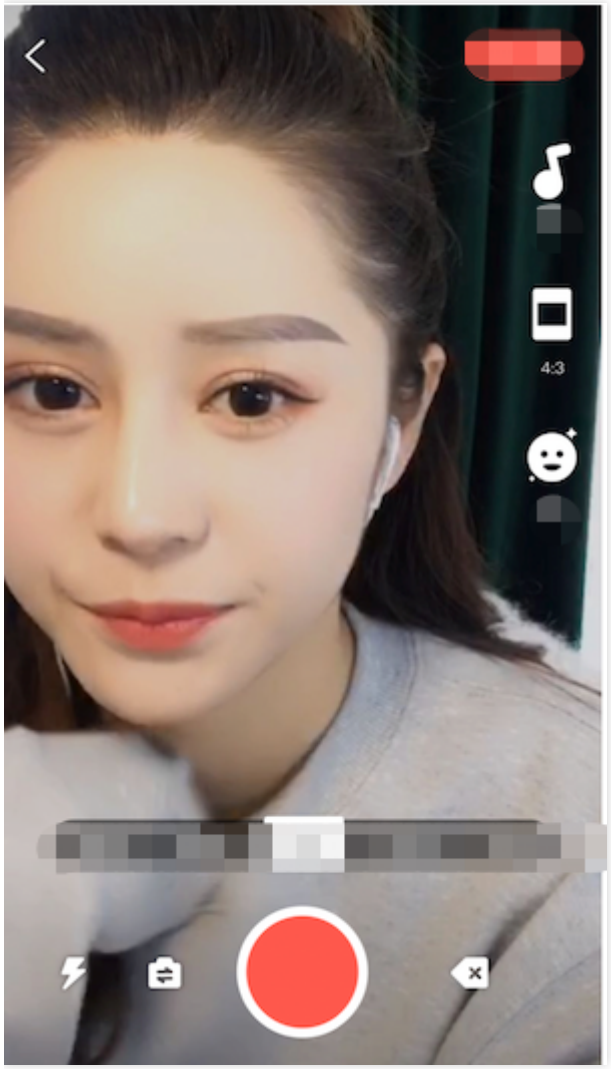
```
<style name="RecordActivityTheme" parent="UGCKitRecordStyle"/>
```

iii. 新建录制 Activity，继承 `FragmentActivity`，实现接口 `ActivityCompat.OnRequestPermissionsResultCallback`，获取 UGCKitVideoRecord 对象并设置回调方法。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// 必须在代码中设置主题(setTheme)或者在AndroidManifest中设置主题(android:theme)
setTheme(R.style.RecordActivityTheme);
setContentView(R.layout.activity_video_record);
// 获取UGCKitVideoRecord
mUGCKitVideoRecord = (UGCKitVideoRecord) findViewById(R.id.video_record_layou
t);
// 设置录制监听
mUGCKitVideoRecord.setOnRecordListener(new IVideoRecordKit.OnRecordListener()
{
@Override
```

```
public void onRecordCanceled() {
// 录制被取消
}
@Override
public void onRecordCompleted(UGCKitResult result) {
// 录制完成回调
}
});
}
@Override
protected void onStart() {
super.onStart();
// 判断是否开启了“相机”和“录音权限”(如何判断权限，参考Github/Demo示例)
if (hasPermission()) {
// UGCKit接管录制的生命周期（关于更多UGCKit接管录制生命周期的方法，参考Github/Demo示例)
mUGCKitVideoRecord.start();
}
}
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
if (grantResults != null && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
mUGCKitVideoRecord.start();
}
}
```

效果如下：



3. 视频导入

1. 新建 xml，加入如下配置：

```
<com.tencent.qcloud.ugckit.UGCKitVideoPicker
android:id="@+id/video_picker"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

2. 在 `res/values/styles.xml` 中新建空的主题，继承 UGCKit 默认视频导入主题。

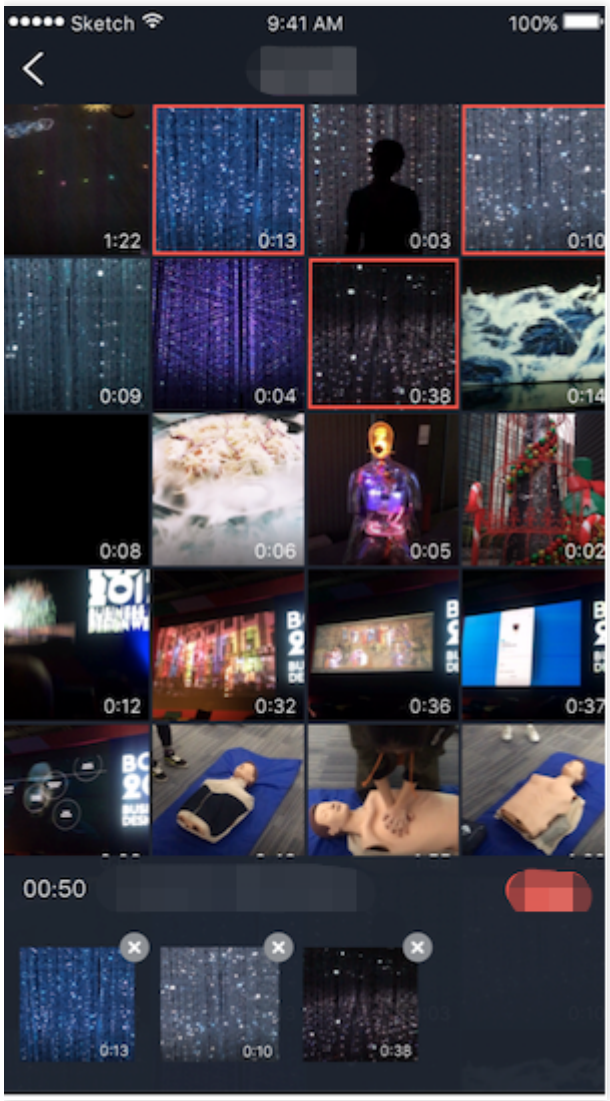
```
<style name="PickerActivityTheme" parent="UGCKitPickerStyle"/>
```

3. 新建 activity，继承 Activity，获取 UGCKitVideoPicker 对象，设置对象回调。

版权所有：腾讯云计算（北京）有限责任公司

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
// 必须在代码中设置主题(setTheme)或者在AndroidManifest中设置主题(android:theme)
setTheme(R.style.PickerActivityTheme);
setContentView(R.layout.activity_video_picker);
// 获取UGCKitVideoPicker
mUGCKitVideoPicker = (UGCKitVideoPicker) findViewById(R.id.video_picker);
// 设置视频选择监听
mUGCKitVideoPicker.setOnPickerListener(new IPickerLayout.OnPickerListener() {
@Override
public void onPickedList(ArrayList<TCVideoFileInfo> list) {
// UGCKit返回选择的视频路径集合
}
});
}
```

效果如下：



4. **视频裁剪**

1. 新建 xml，加入如下配置：

```
<com.tencent.qcloud.ugckit.UGCKitVideoCut
android:id="@+id/video_cutter"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

2. 在 `res/values/styles.xml` 中新建空的主题，继承 UGCKit 默认编辑主题。

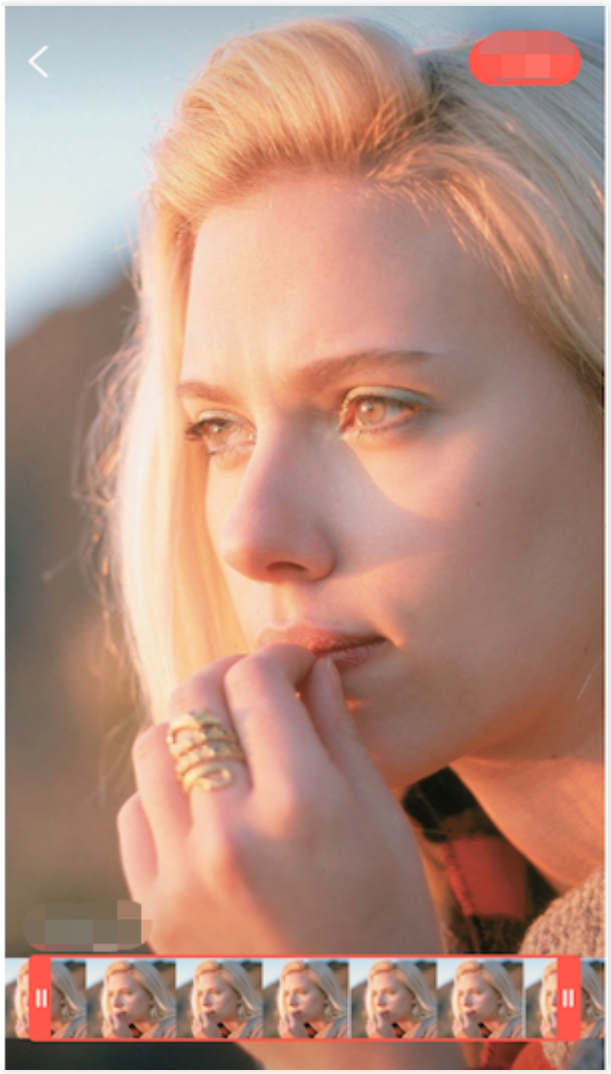
```
<style name="EditerActivityTheme" parent="UGCKitEditerStyle"/>
```

3. 新建 Activity，实现接口 `FragmentActivity`，获取 UGCKitVideoCut 对象，并设置回调方法。

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// 必须在代码中设置主题(setTheme)或者在AndroidManifest中设置主题(android:theme)
setTheme(R.style.EditerActivityTheme);
setContentView(R.layout.activity_video_cut);
mUGCKitVideoCut = (UGCKitVideoCut) findViewById(R.id.video_cutter);
// 获取上一个界面视频导入传过来的视频源路径
mVideoPath = getIntent().getStringExtra(UGCKitConstants.VIDEO_PATH);
// UGCKit设置视频源路径
mUGCKitVideoCut.setVideoPath(mVideoPath);
// 设置视频生成的监听
mUGCKitVideoCut.setOnCutListener(new IVideoCutKit.OnCutListener() {

@Override
public void onCutterCompleted(UGCKitResult ugcKitResult) {
// 视频裁剪进度条执行完成后调用
}
@Override
public void onCutterCanceled() {
// 取消裁剪时被调用
}
});
}
@Override
protected void onResume() {
super.onResume();
// UGCKit接管裁剪界面的生命周期（关于更多UGCKit接管裁剪生命周期的方法，参考Github/Demo示例）
mUGCKitVideoCut.startPlay();
}
```

效果如下：



5. 视频特效编辑

1. 在编辑 activity 的 xml 中加入如下配置：

```
<com.tencent.qcloud.ugckit.UGCKitVideoEdit
android:id="@+id/video_edit"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

2. 新建编辑 Activity，继承 `FragmentActivity`，获取 UGCKitVideoEdit 对象并设置回调方法。

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// 必须在代码中设置主题(setTheme)或者在AndroidManifest中设置主题(android:theme)
```

```
setTheme(R.style.EditerActivityTheme);
setContentView(R.layout.activity_video_editer);
// 设置视频源路径（非必须，如果上个界面是裁剪界面，且设置setVideoEditFlag(true)则可以不用设置视频源）
mVideoPath = getIntent().getStringExtra(UGCKitConstants.VIDEO_PATH);
mUGCKitVideoEdit = (UGCKitVideoEdit) findViewById(R.id.video_edit);
if (!TextUtils.isEmpty(mVideoPath)) {
mUGCKitVideoEdit.setVideoPath(mVideoPath);
}
// 初始化播放器
mUGCKitVideoEdit.initPlayer();
mUGCKitVideoEdit.setOnVideoEditListener(new IVideoEditKit.OnEditListener() {
@Override
public void onEditCompleted(UGCKitResult ugcKitResult) {
// 视频编辑完成
}
@Override
public void onEditCanceled() {

}
});
}
@Override
protected void onResume() {
super.onResume();
// UGCKit接管编辑界面的生命周期（关于更多UGCKit接管编辑生命周期的方法，参考Github/Demo示例）
mUGCKitVideoEdit.start();
}
```

效果如下：

详细介绍

以下为各模块的详细说明：

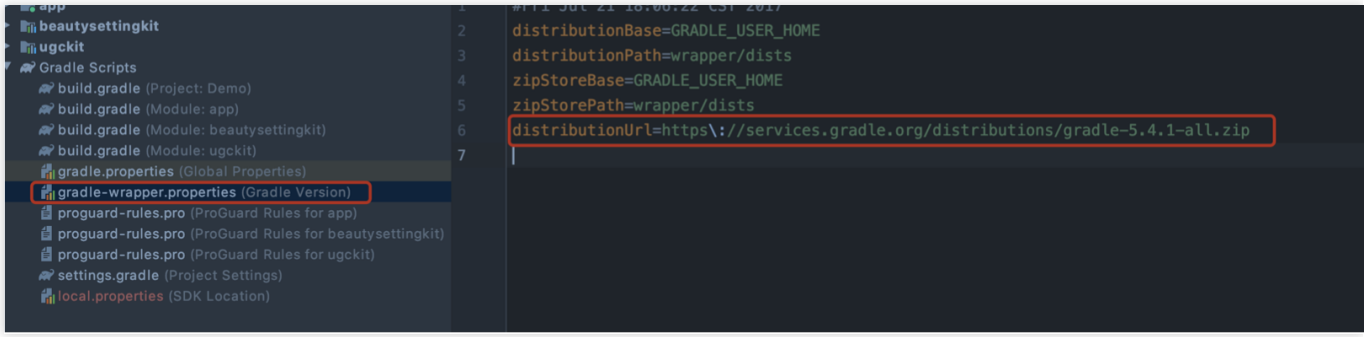
- 视频录制
- 视频编辑
- 视频拼接
- 视频上传
- 视频播放

常见问题

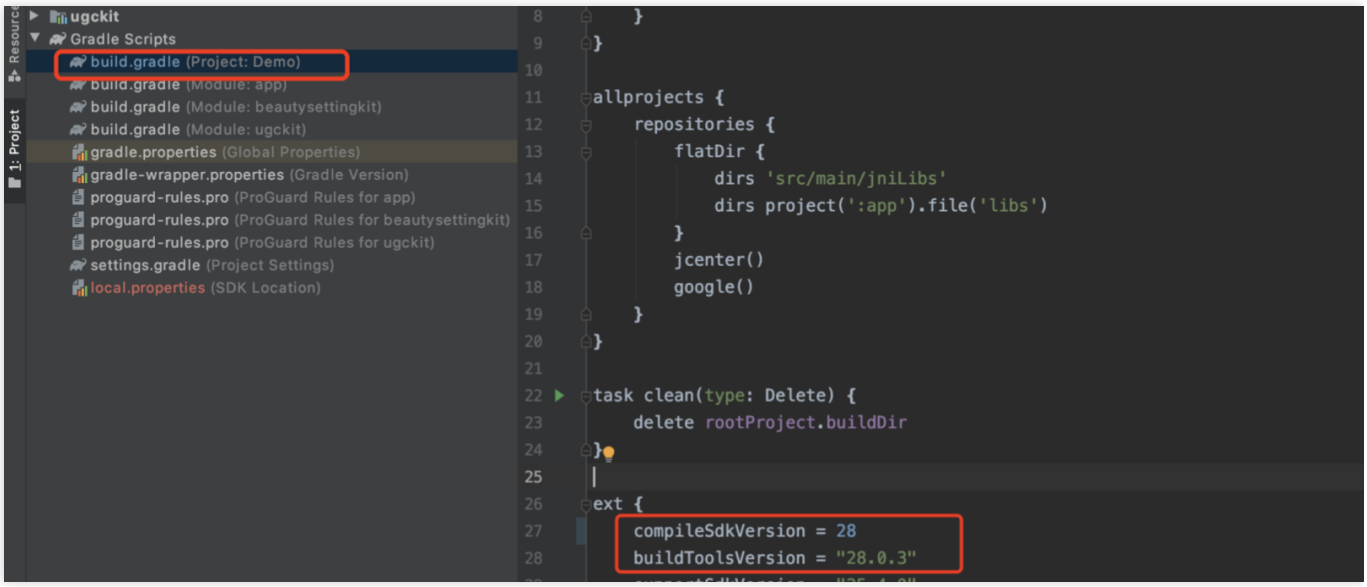
是否支持 AndroidX？

UGCKit 最新版本已经使用了AndroidX，如果您使用的还是基于Support包的UGCKit，可以更新到最新版本，也可以按照以下步骤切换到AndroidX：为了方便说明，以腾讯云 UGSVSDK 为例，此 Demo 中同样使用了 UGCKit 模块。

1. **前提准备：**
   - 将Android Studio更新至 Android Studio 3.2及以上。
   - Gradle 插件版本改为 4.6及以上。

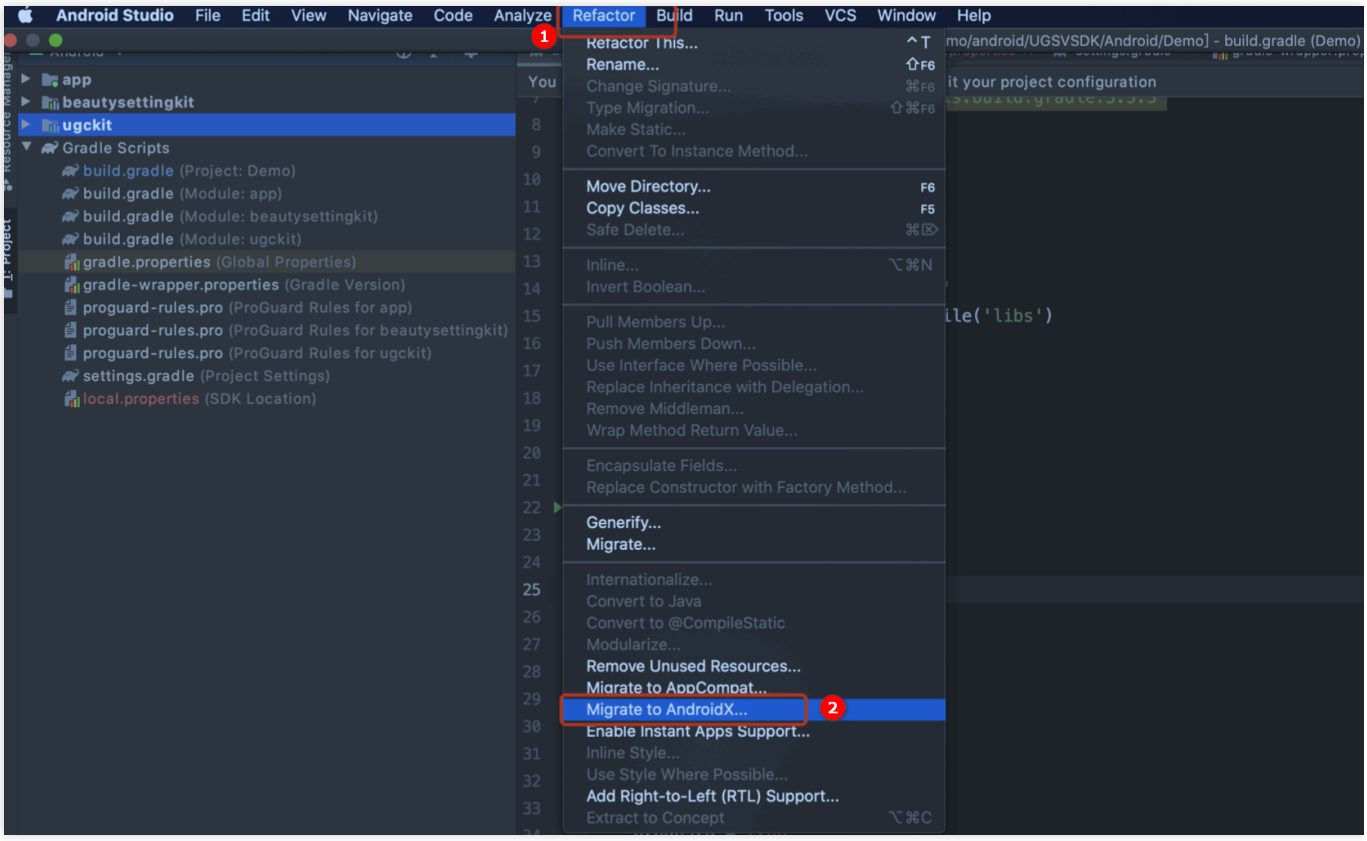


   - compileSdkVersion 版本升级到 28 及以上。
   - buildToolsVersion 版本改为 28.0.2 及以上。

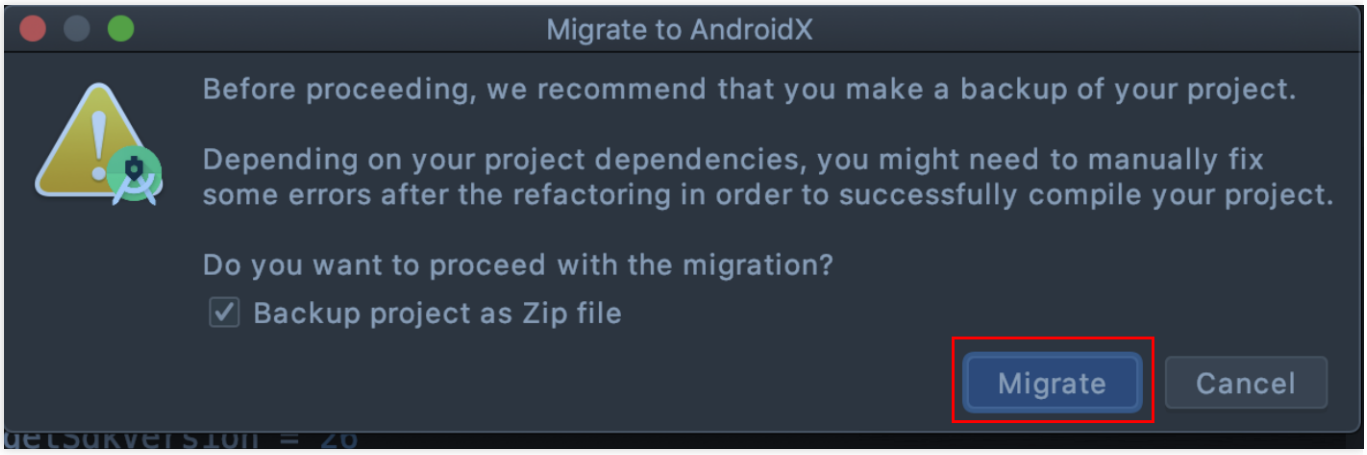


2. **开启迁移：**

i. 使用 Android Studio 导入项目后，从菜单栏中依次选择 **Refactor > Migrate to AndroidX**。



ii. 单击 **Migrate**，即可将现 有项目迁移到 AndroidX。



UGCKit 编译版本错误？

- **报错信息**：

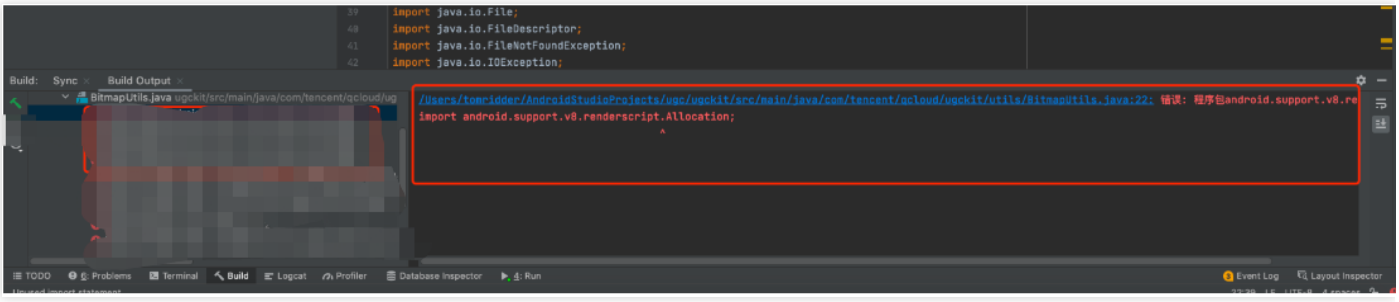
```
ERROR: Unable to find method 'org.gradle.api.tasks.compile.CompileOptions.setBo
otClasspath(Ljava/lang/String;)V'.
```

```
Possible causes for this unexpected error include:
```

- **问题原因**：UGCKit 使用的 Gradle 插件版本为 2.2.3 ， Gradle版本为 3.3。
- **解决方法**：请检查 `Android Studio Gradle` 插件版本和 Gradle 版本是否匹配，具体请参见 查看 Gradle 插件对应Gradle版本。

UGCKit 包编译时出现报错？

- **报错信息**：



- **问题原因**：主要是 `ugckit module` 缺少 `renderscript-v8.jar` 。这个库主要是对图形的处理，模糊，渲染。
- **解决方法**： `renderscript-v8.jar` 包的目录在 `\sdk\build-tools\` 里，您需在 `ugckit module` 下新建一个 libs 包，然后将 `renderscript-v8.jar` 加入 libs 包即可。

拍照和录制

拍照和录制

iOS

最近更新时间：2022-10-25 11:26:58

功能概览

视频录制包括视频变速录制、美颜、滤镜、声音特效、背景音乐设置等功能。

使用类介绍

腾讯云 UGC SDK 提供了相关接口用来实现短视频的录制，其详细定义如下：

接口文件	功能
`TXUGCRecord.h`	小视频录制功能
`TXUGCRecordListener.h`	小视频录制回调
`TXUGCRecordEventDef.h`	小视频录制事件回调
`TXUGCRecordTypeDef.h`	基本参数定义
`TXUGCPartsManager.h`	视频片段管理类，用于视频的多段录制，回删等

使用说明

视频录制的基本使用流程如下：

1. 配置录制参数。
2. 启动画面预览。
3. 设置录制效果。
4. 完成录制。

示例

```
@interface VideoRecordViewController <TXUGCRecordListener> {
UIView *_videoRecordView;
}
@implementation VideoRecordViewController
- (void)viewDidLoad {
[super viewDidLoad];
// 创建一个视图用于显示相机预览图片
_videoRecordView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view addSubview:_videoRecordView];
// 1. 配置录制参数
TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];
param.videoQuality = VIDEO_QUALITY_MEDIUM;
// 2. 启动预览，设置参数与在哪个View上进行预览
[[TXUGCRecord shareInstance] startCameraSimple:param preview:_videoRecordView];
// 3. 设置录制效果，这里以添加水印为例
UIImage *watermarke = [UIImage imageNamed:@"watermarke"];
[[TXUGCRecord shareInstance] setWaterMark:watermarke normalizationFrame:CGRectMak
e(0.01, 0.01, 0.1, 0)];
}
// 4. 开始录制
- (IBAction)onStartRecord:(id)sender {
[TXUGCRecord shareInstance].recordDelegate = self;
int result = [[TXUGCRecord shareInstance] startRecord];
if(0 != result) {
if(-3 == result) [self alert:@"启动录制失败" msg:@"请检查摄像头权限是否打开"];
else if(-4 == result) [self alert:@"启动录制失败" msg:@"请检查麦克风权限是否打开"];
else if(-5 == result) [self alert:@"启动录制失败" msg:@"licence 验证失败"];
} else {
// 启动成功
}
}
// 结束录制
- (IBAction)onStopRecord:(id)sender {
[[TXUGCRecord shareInstance] stopRecord];
}
// 录制完成回调
-(void) onRecordComplete:(TXUGCRecordResult*)result
{
if (result.retCode == UGC_RECORD_RESULT_OK) {
// 录制成功，  视频文件在result.videoPath中
} else {
// 错误处理，错误码定义请参见 TXUGCRecordTypeDef.h 中 TXUGCRecordResultCode 的定义
}
}
-(void)alert:(NSString *)title msg:(NSString *)msg
{
```

```
UIAlertView *alert = [[UIAlertView alloc] initWithTitle:title message:msg delegat
e:self cancelButtonTitle:@"确定" otherButtonTitles:nil, nil];
[alert show];
}
@end
```

画面预览

TXUGCRecord（位于 TXUGCRecord.h）负责小视频的录制功能，我们的第一个工作是先把预览功能实现。startCameraSimplePreview 函数用于启动预览。由于启动预览要打开摄像头和麦克风，所以这里可能会有权限申请的提示窗。

1. 启动预览

```
TXUGCRecord *record = [TXUGCRecord sharedInstance];
record.recordDelegate = self; //设置录制回调，回调方法见 TXUGCRecordListener
//配置相机及启动预览
TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_LOW; // 360p
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM; // 540p
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH; // 720p
param.frontCamera = YES; //使用前置摄像头
param.minDuration = 5; //视频录制的最小时长5s
param.maxDuration = 60; //视频录制的最大时长60s
param.enableBFrame = YES; // 开启B帧，相同码率下能获得更好的画面质量
//在self.previewView中显示照相机预览画面
[recorder startCameraSimple:param preview:self.previewView];
//结束画面预览
[[TXUGCRecord shareInstance] stopCameraPreview];
```

2. 调整预览参数

如果在相机启动后，可以通过以下方法修改：

```
// 切换视频录制分辨率到540p
[recorder setVideoResolution: VIDEO_RESOLUTION_540_960];
// 切换视频录制码率到6500Kbps
[recorder setVideoBitrate: 6500];
// 设置焦距为3，当为1的时候为最远视角（正常镜头），当为5的时候为最近视角（放大镜头）
[recorder setZoom: 3];
// 切换到后置摄像头 YES 切换到前置摄像头 NO 切换到后置摄像头
[recorder switchCamera: NO];
// 打开闪光灯 YES为打开， NO为关闭.
```

版权所有：腾讯云计算（北京）有限责任公司

```
[recorder toggleTorch: YES];
// 设置自定义图像处理回调
recorder.videoProcessDelegate = delegate;
```

录制过程控制

录制的开始、暂停与恢复

```
// 开始录制
[recorder startRecord];
// 开始录制，可以指定输出视频文件地址和封面地址
[recorder startRecord:videoFilePath coverPath:coverPath];
// 开始录制，可以指定输出视频文件地址、视频分片存储地址和封面地址
[recorder startRecord:videoFilePath videoPartsFolder:videoPartFolder coverPath:co
verPath];
// 暂停录制
[recorder pauseRecord];
// 继续录制
[recorder resumeRecord];
// 结束录制
[recorder stopRecord];
```

录制的过程和结果是通过 TXUGCRecordListener（位于 TXUGCRecordListener.h 中定义）协议进行回调：

- onRecordProgress 用于反馈录制的进度，参数 millisecond 表示录制时长，单位毫秒。

```
@optional
(void)onRecordProgress:(NSInteger)milliSecond;
```

- onRecordComplete 反馈录制的结果，TXRecordResult 的 retCode 和 descMsg 字段分别表示错误码和错误描述信息，videoPath 表示录制完成的小视频文件路径，coverImage 为自动截取的小视频第一帧画面，便于在视频发布阶段使用。

```
@optional
(void)onRecordComplete:(TXUGCRecordResult*)result;
```

- onRecordEvent 录制事件回调预留的接口，暂未使用。

```
@optional
(void)onRecordEvent:(NSDictionary*)evt;
```

版权所有：腾讯云计算（北京）有限责任公司

录制属性设置

1. 画面设置

```
// 设置横竖屏录制
[recorder setHomeOrientation:VIDOE_HOME_ORIENTATION_RIGHT];
// 设置视频预览方向
// rotation : 取值为 0 , 90, 180, 270（其他值无效） 表示视频预览向右旋转的角度
// 注意：需要在startRecord 之前设置，录制过程中设置无效
[recorder setRenderRotation:rotation];
// 设置录制的宽高比
// VIDEO_ASPECT_RATIO_9_16 宽高比为9:16
// VIDEO_ASPECT_RATIO_3_4 宽高比为3:4
// VIDEO_ASPECT_RATIO_1_1 宽高比为1:1
// 注意：需要在startRecord 之前设置，录制过程中设置无效
[recorder setAspectRatio:VIDEO_ASPECT_RATIO_9_16];
```

2. 速度设置

```
// 设置视频录制速率
// VIDEO_RECORD_SPEED_SLOWEST, 极慢速
// VIDEO_RECORD_SPEED_SLOW, 慢速
// VIDEO_RECORD_SPEED_NOMAL, 正常速
// VIDEO_RECORD_SPEED_FAST, 快速
// VIDEO_RECORD_SPEED_FASTEST, 极快速
[recorder setRecordSpeed:VIDEO_RECORD_SPEED_NOMAL];
```

3. 声音设置

```
// 设置麦克风的音量大小，播放背景音混音时使用，用来控制麦克风音量大小
// 音量大小,1为正常音量,建议值为0-2,如果需要调大音量可以设置更大的值.
[recorder setMicVolume:volume];
// 设置录制是否静音 参数 isMute 代表是否静音，默认不静音
[recorder setMute:isMute];
```

拍照

```
// 截图/拍照，startCameraSimplePreview 或者 startCameraCustomPreview 之后调用有效
[recorder snapshot:^(UIImage *image) {
// image 为截图结果
}];
```

设置效果

在视频录制的过程中，您可以给录制视频的画面设置各种特效。

1. 水印效果

```
// 设置全局水印
// normalizationFrame : 水印相对于视频图像的归一化值，sdk 内部会根据水印宽高比自动计算 height
// 例如视频图像大小为（540，960） frame 设置为（0.1，0.1，0.1，0）
// 水印的实际像素坐标为
// (540*0.1，960*0.1，540*0.1，540*0.1*waterMarkImage.size.height / waterMarkImage.size.width)
[recorder setWaterMark:waterMarkImage normalizationFrame:frame)
```

2. 滤镜效果

```
//设置风格滤镜
// 设置颜色滤镜：浪漫、清新、唯美、粉嫩、怀旧...
// filterImage : 指定滤镜用的颜色查找表。注意：一定要用 png 格式
// demo 用到的滤镜查找表图片位于 FilterResource.bundle 中
[recorder setFilter:filterImage];
// 用于设置滤镜的效果程度，从0到1，越大滤镜效果越明显，默认取值0.5
[recorder setSpecialRatio:ratio];
// 设置组合滤镜特效
// mLeftBitmap 左侧滤镜
// leftIntensity 左侧滤镜强度
// mRightBitmap 右侧滤镜
// rightIntensity 右侧滤镜强度
// leftRatio 左侧图片占的比例大小
// 可以此接口实现滑动切换滤镜的效果，详见 demo。
[recorder setFilter:leftFilterImgage leftIntensity:leftIntensity rightFilter:rightFilterImgage rightIntensity:rightIntensity leftRatio:leftRatio];
```

3. 美颜效果

```
// 设置美颜风格、级别、美白及红润的级别
// beautyStyle的定义如下:
// typedef NS_ENUM(NSInteger, TXVideoBeautyStyle) {
// VIDOE_BEAUTY_STYLE_SMOOTH = 0, // 光滑
// VIDOE_BEAUTY_STYLE_NATURE = 1, // 自然
// };
// 级别的范围为0-9 0为关闭,  1-9值越大, 效果越明显
[recorder setBeautyStyle:beautyStyle beautyLevel:beautyLevel whitenessLevel:white
nessLevel ruddinessLevel:ruddinessLevel];
```

高级功能

多段录制
录制草稿箱
添加背景音乐
变声和混响
定制视频数据

Android

最近更新时间：2022-05-24 15:45:11

视频录制包括视频变速录制、美颜、滤镜、声音特效、背景音乐设置等功能。

相关类介绍

类	功能
TXUGCRecord	实现视频的录制功能
TXUGCPartsManager	视频片段管理类，用于视频的多段录制，回删等
ITXVideoRecordListener	录制回调
TXRecordCommon	基本参数定义，包括了视频录制回调及发布回调接口

使用说明

视频录制的基本使用流程如下：

1. 配置录制参数。
2. 启动画面预览。
3. 设置录制效果。
4. 完成录制。

代码示例

```
// 创建一个用户相机预览的 TXCloudVideoView
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view);
// 1、配置录制参数，已推荐配置 TXUGCSimpleConfig 为例
TXRecordCommon.TXUGCSimpleConfig param = new TXRecordCommon.TXUGCSimpleConfig();
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM;
// 2、启动画面预览
mTXCameraRecord.startCameraSimplePreview(param, mVideoView);
// 3、设置录制效果，这里以添加水印为例
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
```

```
mTXCameraRecord.setWatermark(BitmapFactory.decodeResource(getResources(), R.drawa
ble.watermark), rect);
// 4、开始录制
int result = mTXCameraRecord.startRecord();
if (result != TXRecordCommon.START_RECORD_OK) {
if (result == -4) {画面还没出来}
else if (result == -3) {//版本太低}
else if (result == -5) {// licence 验证失败] }
}
else{// 启动成功}
// 结束录制
mTXCameraRecord.stopRecord();
// 录制完成回调
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
if (result.retCode >= 0 ) {
// 录制成功,  视频文件在 result.videoPath 中
}else{
// 错误处理, 错误码定义请参见 TXRecordCommon 中“录制结果回调错误码定义”
}
}
```

画面预览

TXUGCRecord（位于 TXUGCRecord.java）负责小视频的录制功能，我们的第一个工作是先把预览功能实现。startCameraSimplePreview 函数用于启动预览。由于启动预览要打开摄像头和麦克风，所以这里可能会有权限申请的提示窗。

1. 启动预览

```
TXUGCRecord mTXCameraRecord = TXUGCRecord.getInstance(this.getApplicationContext
());
mTXCameraRecord.setVideoRecordListener(this); // 设置录制回调
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view); // 准备一个预览摄像头
画面的
TXRecordCommon.TXUGCSimpleConfig param = new TXRecordCommon.TXUGCSimpleConfig();
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_LOW; // 360p
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM; // 540p
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH; // 720p
param.isFront = true; // 是否使用前置摄像头
param.minDuration = 5000; // 视频录制的最小时长 ms
param.maxDuration = 60000; // 视频录制的最大时长 ms
param.touchFocus = false; // false 为自动聚焦；true 为手动聚焦
mTXCameraRecord.startCameraSimplePreview(param,mVideoView);
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 结束画面预览
mTXCameraRecord.stopCameraPreview();
```

2. 调整预览参数

在相机启动后，可以通过以下方法调整预览参数：

```
// 切换视频录制分辨率到540p
mTXCameraRecord.setVideoResolution(TXRecordCommon.VIDEO_RESOLUTION_540_960);
// 切换视频录制码率到6500Kbps
mTXCameraRecord.setVideoBitrate(6500);
// 获取摄像头支持的最大焦距
mTXCameraRecord.getMaxZoom();
// 设置焦距为3，当为1的时候为最远视角（正常镜头），当为5的时候为最近视角（放大镜头）
mTXCameraRecord.setZoom(3);
// 切换到后置摄像头 true 切换到前置摄像头；false 切换到后置摄像头
mTXCameraRecord.switchCamera(false);
// 打开闪光灯 true 为打开， false 为关闭.
mTXCameraRecord.toggleTorch(false);
// param.touchFocus 为 true 时为手动聚焦，可以通过下面接口设置聚焦位置
mTXCameraRecord.setFocusPosition(eventX, eventY);
// 设置自定义图像处理回调
mTXCameraRecord.setVideoProcessListener(this);
```

拍照

在相机开启预览后，即可使用拍照的功能。

```
// 截图/拍照，startCameraSimplePreview 或者 startCameraCustomPreview 之后调用有效
mTXCameraRecord.snapshot(new TXRecordCommon.ITXSnapshotListener() {
@Override
public void onSnapshot(Bitmap bmp) {
// 保存或者显示截图
}
});
```

录制过程控制

录制的开始、暂停与恢复。

```
// 开始录制
mTXCameraRecord.startRecord();
// 开始录制，可以指定输出视频文件地址和封面地址
mTXCameraRecord.startRecord(videoFilePath, coverPath);
// 开始录制,可以指定输出视频文件地址、视频分片存储地址、封面地址
mTXCameraRecord.startRecord(videoFilePath, videoPartFolder, coverPath);
// 暂停录制
mTXCameraRecord.pauseRecord();
// 继续录制
mTXCameraRecord.resumeRecord();
// 结束录制
mTXCameraRecord.stopRecord();
```

录制的过程和结果是通过 TXRecordCommon.ITXVideoRecordListener（位于 TXRecordCommon.java 中定义）接口反馈：

- onRecordProgress 用于反馈录制的进度，参数 millisecond 表示录制时长，单位：毫秒。

```
@optional
void onRecordProgress(long milliSecond);
```

- onRecordComplete 反馈录制的结果，TXRecordResult 的 retCode 和 descMsg 字段分别表示错误码和错误描述信息，videoPath 表示录制完成的小视频文件路径，coverImage 为自动截取的小视频第一帧画面，便于在视频发布阶段使用。

```
@optional
void onRecordComplete(TXRecordResult result);
```

- onRecordEvent 录制事件回调，包含事件id和事件相关的参数(key,value)格式

```
@optional
void onRecordEvent(final int event, final Bundle param);
```

录制属性设置

画面设置

```
// 设置横竖屏录制
mTXCameraRecord.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
// 设置视频预览方向
// TXLiveConstants.RENDER_ROTATION_0(常规竖屏)
```

```
// TXLiveConstants.RENDER_ROTATION_90(左旋90度)
// TXLiveConstants.RENDER_ROTATION_180(左旋180度)
// TXLiveConstants.RENDER_ROTATION_270(左旋270度)
// 注意：需要在 startRecord 之前设置，录制过程中设置无效
mTXCameraRecord.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);
// 设置录制的宽高比
// TXRecordCommon.VIDEO_ASPECT_RATIO_9_16 宽高比为9:16
// TXRecordCommon.VIDEO_ASPECT_RATIO_3_4 宽高比为3:4
// TXRecordCommon.VIDEO_ASPECT_RATIO_1_1 宽高比为1:1
// 注意：需要在 startRecord 之前设置，录制过程中设置无效
mTXCameraRecord.setAspectRatio(TXRecordCommon.VIDEO_ASPECT_RATIO_9_16);
```

速度设置

```
// 设置视频录制速率
// TXRecordCommon.RECORD_SPEED_SLOWEST(极慢速)
// TXRecordCommon.RECORD_SPEED_SLOW(慢速)
// TXRecordCommon.RECORD_SPEED_NORMAL(标准)
// TXRecordCommon.RECORD_SPEED_FAST(快速)
// TXRecordCommon.RECORD_SPEED_FASTEST(极快速)
mTXCameraRecord.setRecordSpeed(TXRecordCommon.VIDEO_RECORD_SPEED_NORMAL);
```

声音设置

```
// 设置麦克风的音量大小，播放背景音混音时使用，用来控制麦克风音量大小
// 音量大小，1为正常音量，建议值为0-2，如果需要调大音量可以设置更大的值。
mTXCameraRecord.setMicVolume(volume);
// 设置录制是否静音 参数 isMute 代表是否静音，默认不静音
mTXCameraRecord.setMute(isMute);
```

设置效果

在视频录制的过程中，您可以给录制视频的画面设置各种特效。

水印

```
// 设置全局水印
// TXRect-水印相对于视频图像的归一化值，sdk 内部会根据水印宽高比自动计算 height
// 例如视频图像大小为（540，960） TXRect 三个参数设置为0.1，0.1，0.1
// 水印的实际像素坐标为（540 * 0.1，960 * 0.1，540 * 0.1 ，
// 540 * 0.1 * watermarkBitmap.height / watermarkBitmap.width）
mTXCameraRecord.setWatermark(watermarkBitmap, txRect)
```

滤镜

```
// 设置颜色滤镜：浪漫、清新、唯美、粉嫩、怀旧...
// filterBitmap ：指定滤镜用的颜色查找表。注意：一定要用 png 格式。
// demo 用到的滤镜查找表图片位于 RTMPAndroidDemo/app/src/main/res/drawable-xxhdpi/ 目录下。
mTXCameraRecord.setFilter(filterBitmap);
// 设置组合滤镜特效
// mLeftBitmap 左侧滤镜
// leftIntensity 左侧滤镜程度
// mRightBitmap 右侧滤镜
// rightIntensity 右侧滤镜程度
// leftRadio 左侧图片占的比例大小
// 可以此接口实现滑动切换滤镜的效果，详见 demo。
mTXCameraRecord.setFilter(mLeftBitmap, leftIntensity, mRightBitmap, rightIntensity, leftRatio);
// 用于设置滤镜的效果程度，从0到1，越大滤镜效果越明显，默认取值0.5
mTXCameraRecord.setSpecialRatio(0.5);
```

美颜

```
// 设置美颜类型
mTXCameraRecord.setBeautyStyle(style);
// 设置大眼效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setEyeScaleLevel(eyeScaleLevel);
// 设置瘦脸效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setFaceScaleLevel(faceScaleLevel);
// 设置v脸效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setFaceVLevel(level);
// 设置下巴拉伸或收缩效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setChinLevel(scale);
// 设置缩脸效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setFaceShortLevel(level);
// 设置小鼻效果 建议0-9，如果需要更明显可以设置更大值
mTXCameraRecord.setNoseSlimLevel(scale);
// 设置绿幕文件:目前图片支持 jpg/png，视频支持 mp4/3gp 等 Android 系统支持的格式并支持循环播放
mTXCameraRecord.setGreenScreenFile(path, isLoop);
// 设置动效贴纸 motionTmplPath 动效文件路径： 空 String "" 则取消动效
mTXCameraRecord.setMotionTmp(motionTmplPath);
// 设置动效贴纸是否静音: true: 动效贴纸静音；false：动效贴纸不静音
mTXCameraRecord.setMotionMute(true);
```

获取 License 信息

新版本的 SDK 增加了短视频 License 的校验，如果校验没通过，您可以通过该接口来查询 License 中具体信息：

```
TXUGCBase.getInstance().getLicenceInfo(Context context);
```

高级功能

多段录制
录制草稿箱
添加背景音乐
变声和混响
定制视频数据

多段录制

iOS

最近更新时间：2022-05-24 15:59:44

视频多段录制基本使用流程如下：

1. 启动画面预览。
2. 开始录制。
3. 开始播放 BGM。
4. 暂停录制。
5. 暂停播放 BGM。
6. 继续播放 BGM。
7. 继续录制。
8. 停止录制。
9. 停止播放 BGM。

```
//开启画面预览
recorder = [TXUGCRecord shareInstance];
[recorder startCameraCustom:param preview:preview];
// 开始录制
[recorder startRecord];
//设置BGM
[recorder setBGM:BGMPath];
//开始播放BGM
[recorder playBGMFromTime:beginTime toTime:_BGMDuration withBeginNotify:^(NSInteg
er errCode) {
//开始播放
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
//播放进度
} andCompleteNotify:^(NSInteger errCode) {
//播放结束
}];
// 调用 pauseRecord 后会生成一段视频，视频可以在 TXUGCPartsManager 里面获取管理
[recorder pauseRecord];
//暂停播放BGM
[recorder pauseBGM];
//继续播放BGM
[recorder resumeBGM];
// 继续录制视频
[recorder resumeRecord];
// 停止录制，将多段视频合成为一个视频输出
```

版权所有：腾讯云计算（北京）有限责任公司

```
[recorder stopRecord];
// 停止播放BGM
[recorder stopBGM];
//获取视频分片管理对象
TXUGCPartsManager *partsManager = recorder.partsManager;
//获取当前所有视频片段的总时长
[partsManager getDuration];
//获取所有视频片段路径
[partsManager getVideoPathList];
// 删除最后一段视频
[partsManager deleteLastPart];
// 删除指定片段视频
[partsManager deletePart:1];
// 删除所有片段视频
[partsManager deleteAllParts];
//您可以添加当前录制视频之外的视频
[partsManager insertPart:videoPath atIndex:0];
//合成所有片段视频
[partsManager joinAllParts: videoOutputPath complete:complete];
```

Android

最近更新时间：2022-05-24 16:00:25

视频多段录制基本使用流程如下：

1. 启动画面预览。
2. 开始录制。
3. 开始播放 BGM。
4. 暂停录制。
5. 暂停播放 BGM。
6. 继续录制。
7. 继续播放 BGM。
8. 停止录制。
9. 停止 BGM。

```
// 开始录制
mTXCameraRecord.startRecord();
// pauseRecord 后会生成一段视频，视频可以在 TXUGCPartsManager 里面获取
mTXCameraRecord.pauseRecord();
mTXCameraRecord.pauseBGM();
// 继续录制视频
mTXCameraRecord.resumeRecord();
mTXCameraRecord.resumeBGM();
// 停止录制，将多段视频合成为一个视频输出
mTXCameraRecord.stopBGM();
mTXCameraRecord.stopRecord();
// 获取片段管理对象
mTXCameraRecord.getPartsManager();
// 获取当前所有视频片段的总时长
mTXUGCPartsManager.getDuration();
// 获取所有视频片段路径
mTXUGCPartsManager.getPartsPathList();
// 删除最后一段视频
mTXUGCPartsManager.deleteLastPart();
// 删除指定片段视频
mTXUGCPartsManager.deletePart(index);
// 删除所有片段视频
mTXUGCPartsManager.deleteAllParts();
// 您可以添加当前录制视频之外的视频
mTXUGCPartsManager.insertPart(videoPath, index) ;
```

版权所有：腾讯云计算（北京）有限责任公司

录制草稿箱

iOS

最近更新时间：2022-05-24 16:02:20

草稿箱实现步骤：

第一次录制

1. 开始录制。
2. 暂停/结束第一次录制。
3. 缓存视频分片到本地（草稿箱）。

第二次录制

1. 预加载本地缓存视频分片。
2. 继续录制。
3. 结束录制。

```
//获取第一次视频录制对象
record = [TXUGCRecord shareInstance];
//开始录制
[record startRecord];
//暂停录制，缓存视频分片
[record pauseRecord:^{
NSArray *videoPathList = record.partsManager.getVideoPathList;
//videoPathList 写本地
}];
//获取第二次视频录制对象
record2 = [TXUGCRecord shareInstance];
//预加载本地缓存分片
[record2.partsManager insertPart:videoPath atIndex:0];
//开始录制
[record2 startRecord];
//结束录制,SDK会合成缓存视频片段和当前录制视频片段
[record2 stopRecord];
```

注意：

具体实现方法请参考 小视频源码 中的 UGCKitRecordViewController 类。

Android

最近更新时间：2022-05-24 16:02:44

草稿箱实现步骤：

第一次录制

1. 开始录制。
2. 暂停/结束第一次录制。
3. 缓存视频分片到本地（草稿箱）。

第二次录制

1. 预加载本地缓存视频分片。
2. 继续录制。
3. 结束录制。

```
// 获取第一次视频录制对象
mTXCameraRecord = TXUGCRecord.getInstance(this.getApplicationContext());

// 开始录制
mTXCameraRecord.startRecord();

// 暂停录制
mTXCameraRecord.pauseRecord();

// 获取缓存的录制分片，记录到本地
List<string> pathList = mTXCameraRecord.getPartsManager().getPartsPathList(); //
pathList 写本地

// 第二次打开 app，获取录制对象
mTXCameraRecord2 = TXUGCRecord.getInstance(this.getApplicationContext());

// 预加载本地缓存片段
mTXCameraRecord2.getPartsManager().insertPart(videoPath, 0);

// 开始录制
mTXCameraRecord2.startRecord();

// 结束录制，SDK 会把缓存视频片段和当前录制视频片段合成
mTXCameraRecord2.stopRecord();
```

版权所有：腾讯云计算（北京）有限责任公司

说明：

具体实现方法请参考 小视频源码 中录制中的 RecordDraftManager 类的使用。

添加背景音乐

iOS

最近更新时间：2022-05-24 16:04:44

录制添加 BGM

```
//获取 recorder 对象
TXUGCRecord *recorder = [TXUGCRecord shareInstance];
// 设置 BGM 文件路径
[recorder setBGMAsset:path];
// 设置 BGM，从系统媒体库 loading 出来的音乐，可以直接传入对应的 AVAsset
[recorder setBGMAsset:asset];
// 播放 BGM
[recorder playBGMFromTime:beginTime
toTime:endTime
withBeginNotify:^(NSInteger errCode) {
// 播放开始回调，errCode 0为成功其它为失败
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
// progressMS: 已经播放的时长，  durationMS: 总时长
} andCompleteNotify:^(NSInteger errCode) {
// 播放结束回调，errCode 0为成功其它为失败
}];
// 停止播放 BGM
[recorder stopBGM];
// 暂停播放 BGM
[recorder pauseBGM];
// 继续播放 BGM
[recorder resumeBGM];
// 设置麦克风的音量大小，播放背景音乐混音时使用，用来控制麦克风音量大小
// volume: 音量大小，1为正常音量，建议值为0-2，如果需要调大音量可以设置更大的值
[recorder setMicVolume:1.0];
// setBGMVolume 设置背景音乐的音量大小，播放背景音乐混音时使用，用来控制背景音音量大小
// volume: 音量大小，1为正常音量，建议值为0-2，如果需要调大背景音量可以设置更大的值
[recorder setBGMVolume:1.0];
```

编辑添加 BGM

```
//初始化编辑器
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = videoView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;
ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];
//设置 BGM 路径
[ugcEdit setBGMAsset:fileAsset result:^(int result) {
}];
//设置 BGM 开始和结束时间
[ugcEdit setBGMStartTime:0 endTime:5];
//设置 BGM 是否循环
[ugcEdit setBGMLoop:YES];
//设置 BGM 在视频添加的起始位置
[ugcEdit setBGMAtVideoTime:0];
//设置视频声音大小
[ugcEdit setVideoVolume:1.0];
//设置 BGM 声音大小
[ugcEdit setBGMVolume:1.0];
```

BGM 设置完之后，当启动编辑器预览，BGM 就会根据设置的参数播放，当启动编辑器生成，BGM 也会按照设置的参数合成到生成的视频中。

Android

最近更新时间：2022-05-24 16:04:15

录制添加 BGM

```
// 设置 BGM 路径
mTXCameraRecord.setBGM(path);

// 设置 BGM 播放回调 TXRecordCommon.ITXBGMNotify
mTXCameraRecord.setBGMNofify(notify);

// 播放 BGM
mTXCameraRecord.playBGMFromTime(startTime, endTime)

// 停止播放 BGM
mTXCameraRecord.stopBGM();

// 暂停播放 BGM
mTXCameraRecord.pauseBGM();

// 继续播放 BGM
mTXCameraRecord.resumeBGM();

// 设置背景音乐的音量大小，播放背景音乐混音时使用，用来控制背景音音量大小
// 音量大小，1为正常音量，建议值为0~2，如果需要调大背景音量可以设置更大的值
mTXCameraRecord.setBGMVolume(x);

// 设置背景音乐播放的开始位置和结束位置，在startPlay之前调用，若在暂停时调用则无效
mTXCameraRecord.seekBGM(startTime, endTime);
```

编辑添加 BGM

```
// 设置 BGM 路径，返回值为0表示设置成功； 其他表示失败，如：不支持的音频格式。
public int setBGM(String path);

// 设置 BGM 开始和结束时间，单位毫秒
public void setBGMStartTime(long startTime, long endTime);

// 设置背景音乐是否循环播放：true：循环播放，false：不循环播放
```

```
public void setBGMLoop(boolean looping);

// 设置 BGM 在视频添加的起始位置
public void setBGMAtVideoTime(long videoStartTime);

// 设置视频声音大小， volume 表示声音的大小， 取值范围0 - 1 ， 0 表示静音， 1 表示原声大小。
public void setVideoVolume(float volume);

// 设置BGM声音大小，volume 表示声音的大小， 取值范围0 - 1 ， 0 表示静音， 1 表示原声大小。
public void setBGMVolume(float volume);
```

说明：
BGM 设置完之后，当启动编辑器预览，BGM 就会根据设置的参数播放，当启动编辑器生成，BGM 也会按照设置的参数合成到生成的视频中。

变声和混响

iOS

最近更新时间：2022-05-24 16:06:14

录制变声混响：

```
//获取 recorder 对象
recorder = [TXUGCRecord shareInstance];
// 设置混响
// TXRecordCommon.VIDOE_REVERB_TYPE_0 关闭混响
// TXRecordCommon.VIDOE_REVERB_TYPE_1 KTV
// TXRecordCommon.VIDOE_REVERB_TYPE_2 小房间
// TXRecordCommon.VIDOE_REVERB_TYPE_3 大会堂
// TXRecordCommon.VIDOE_REVERB_TYPE_4 低沉
// TXRecordCommon.VIDOE_REVERB_TYPE_5 洪亮
// TXRecordCommon.VIDOE_REVERB_TYPE_6 金属声
// TXRecordCommon.VIDOE_REVERB_TYPE_7 磁性
[recorder setReverbType:VIDOE_REVERB_TYPE_1];
// 设置变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0 关闭变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1 熊孩子
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2 萝莉
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3 大叔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4 重金属
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6 外国人
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7 困兽
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8 死肥仔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9 强电流
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10 重机械
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11 空灵
[record setVoiceChangerType:VIDOE_VOICECHANGER_TYPE_1];
```

说明：

变声混响只针对录制人声有效，针对 BGM 无效。

Android

最近更新时间：2022-05-24 16:06:42

录制变声混响：

```
// 设置混响
// TXRecordCommon.VIDOE_REVERB_TYPE_0 关闭混响
// TXRecordCommon.VIDOE_REVERB_TYPE_1 KTV
// TXRecordCommon.VIDOE_REVERB_TYPE_2 小房间
// TXRecordCommon.VIDOE_REVERB_TYPE_3 大会堂
// TXRecordCommon.VIDOE_REVERB_TYPE_4 低沉
// TXRecordCommon.VIDOE_REVERB_TYPE_5 洪亮
// TXRecordCommon.VIDOE_REVERB_TYPE_6 金属声
// TXRecordCommon.VIDOE_REVERB_TYPE_7 磁性
mTXCameraRecord.setReverb(TXRecordCommon.VIDOE_REVERB_TYPE_1);
// 设置变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0 关闭变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1 熊孩子
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2 萝莉
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3 大叔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4 重金属
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6 外国人
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7 困兽
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8 死肥仔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9 强电流
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10 重机械
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11 空灵
mTXCameraRecord.setVoiceChangerType(TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1);
```

说明：
变声混响只针对录制人声有效，针对 BGM 无效。

预览裁剪和拼接
视频编辑
iOS

最近更新时间：2022-05-24 15:54:35

功能概览

视频编辑包括视频裁剪、时间特效（慢动作、倒放、重复）、滤镜特效（动感光波、暗黑幻影、灵魂出窍、画面分裂）、滤镜风格（唯美、粉嫩、蓝调等）、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能。

相关类介绍

类名	功能
TXVideoInfoReader.h	媒体信息获取
TXVideoEditer.h	视频编辑

使用说明

视频编辑的基本使用流程如下：

1. 设置视频路径。
2. 添加效果。
3. 生成视频到指定文件。
4. 监听生成事件。

示例

```
// 这以使用了 Demo 中的 Common/UGC/VideoPreview 来做预览的视图
#import "VideoPreview.h"
@implementation EditViewController
{
TXVideoEditer *editor;
VideoPreview *_videoPreview;
}
```

```
- (void)viewDidLoad {
[super viewDidLoad];
_videoPreview = [[VideoPreview alloc] initWithFrame:self.view.bounds];
[self.view addSubview:_videoPreview];
// 编辑预览参数
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = _videoPreview.renderView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;
// 1. 初始化编辑器，如无需预览，可以传 nil 或直接调用 init 方法
TXVideoEditer *editor = [[TXVideoEditer alloc] initWithPreview:param];
// 设置源视频路径
NSString *path = [[NSBundle mainBundle] pathForResource:@"demo" ofType:@"mp4"]
[editor setVideoPath: path];
// 配置代理
editor.generateDelegate = self; // 设置生成事件的回调委托对象，可以获取生成进度与结果
// 2. 对视频进行处理，这里以添加水印为例
[editor setWaterMark:[UIImage imageNamed:@"water_mark"]
normalizationFrame:CGRectMake(0,0,0.1,0)];
}
// 3. 生成视频，以响应用户点击为例
- (IBAction)onGenerate:(id)sender {
NSString *output = [NSTemporaryDirectory() stringByAppendingPathComponent:@"temp.
mp4"];
[editor generateVideo:VIDEO_COMPRESSED_720P videoOutputPath:output];
}
// 4. 获取生成进度
-(void) onGenerateProgress:(float)progress
{
}
// 获取生成结果
-(void) onGenerateComplete:(TXGenerateResult *)result
{
if (result.retCode == 0) {
// 生成成功
} else {
// 生成失败，原因可以查看 result.descMsg
}
}
@end
```

视频信息获取

TXVideoInfoReader 的 getVideoInfo 方法可以获取指定视频文件的一些基本信息，相关接口如下：

```
// 获取视频文件的信息
+ (TXVideoInfo *)getVideoInfo:(NSString *)videoPath;
/** 获取视频文件信息
* @param videoAsset 视频文件属性
* @return 视频信息
*/
+ (TXVideoInfo *)getVideoInfoWithAsset:(AVAsset *)videoAsset;
```

返回的 TXVideoInfo 定义如下：

```
/// 视频信息
@interface TXVideoInfo : NSObject
/// 视频首帧图片
@property (nonatomic, strong) UIImage* coverImage;
/// 视频时长(s)
@property (nonatomic, assign) CGFloat duration;
/// 视频大小(byte)
@property (nonatomic, assign) unsigned long long fileSize;
/// 视频fps
@property (nonatomic, assign) float fps;
/// 视频码率 (kbps)
@property (nonatomic, assign) int bitrate;
/// 音频采样率
@property (nonatomic, assign) int audioSampleRate;
/// 视频宽度
@property (nonatomic, assign) int width;
/// 视频高度
@property (nonatomic, assign) int height;
/// 视频旋转角度
@property (nonatomic, assign) int angle;
@end
```

缩略图获取

缩略图的接口主要用于生成视频编辑界面的预览缩略图，或获取视频封面等。

1. 按个数平分时间获取缩略图

TXVideoInfoReader 的 getSampleImages 可以获取按指定数量，时间间隔相同的预览图：

```
/** 获取视频的采样图列表
* @param count 获取的采样图数量（均匀采样）
* @param maxSize 缩略图的最大尺寸，生成的缩略图大小不会超出这个宽高
```

```
 * @param videoAsset 视频文件属性
 * @param sampleProcess 采样进度
 */
+ (void)getSampleImages:(int)count
maxSize:(CGSize)maxSize
videoAsset:(AVAsset *)videoAsset
progress:(sampleProcess)sampleProcess;
```

开发包中的 VideoRangeSlider 即使用了 getSampleImages 获取了10张缩略图来构建一个由视频预览图组成的进度条。

2. 根据时间列表获取缩略图

```
/**
 * 根据时间列表获取缩略图列表
 * @param asset 视频文件对象
 * @param times 获取的时间列表
 * @param maxSize 缩略图大小
 */
+ (UIImage *)getSampleImagesFromAsset:(AVAsset *)asset
times:(NSArray<NSNumber*> *)times
maxSize:(CGSize)maxSize
progress:(sampleProcess)sampleProcess;
```

编辑预览

视频编辑提供了**定点预览**（将视频画面定格在某一时间点）与**区间预览**（循环播放某一时间段 A<=>B 内的视频片段）两种效果预览方式，使用时需要给 SDK 绑定一个 UIView 用于显示视频画面。

1. 绑定 UIView

TXVideoEditer 的 initWithPreview 函数用于绑定一个 UIView 给 SDK 来渲染视频画面，通过控制 TXPreviewParam 的 renderMode 来设置**自适应**与**填充**两种模式。

```
PREVIEW_RENDER_MODE_FILL_SCREEN - 填充模式，尽可能充满屏幕不留黑边，所以可能会裁剪掉一部分画面。
PREVIEW_RENDER_MODE_FILL_EDGE - 适应模式，尽可能保持画面完整，但当宽高比不合适时会有黑边出现。
```

2. 定点预览

TXVideoEditer 的 previewAtTime 函数用于定格显示某一个时间点的视频画面。

```
/** 渲染某一时刻的视频画面
* @param time 预览帧时间(s)
*/
- (void)previewAtTime:(CGFloat)time;
```

3. 区间预览

TXVideoEditer 的 startPlayFromTime 函数用于循环播放某一时间段 A<=>B 内的视频片段。

```
/** 播放某一时间段的视频
* @param startTime 播放起始时间(s)
* @param endTime 播放结束时间(s)
*/
- (void)startPlayFromTime:(CGFloat)startTime
toTime:(CGFloat)endTime;
```

4. 预览的暂停与恢复

```
/// 暂停播放
- (void)pausePlay;
/// 继续播放
- (void)resumePlay;
/// 停止播放
- (void)stopPlay;
```

5. 美颜滤镜

您可以给视频添加滤镜效果，例如美白、浪漫、清新等滤镜，demo 提供了多种滤镜选择，对应的滤镜资源在 Common/Resource/Filter/FilterResource.bundle 中，同时也可以设置自定义的滤镜。
设置滤镜的方法为：

```
- (void) setFilter:(UIImage *)image;
```

其中 image 为滤镜映射图，image 设置为 nil，会清除滤镜效果。

Demo 示例：

```
TXVideoEditer *_ugcEdit;
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofTyp
e:@"bundle"];
path = [path stringByAppendingPathComponent:@"langman.png"];
UIImage* image = [UIImage imageWithContentsOfFile:path];
[_ugcEdit setFilter:image];
```

版权所有：腾讯云计算（北京）有限责任公司

6. 设置水印

1. 设置全局水印

您可以为视频设置水印图片，并且可以指定图片的位置。
设置水印的方法为：

```
- (void) setWaterMark:(UIImage *)waterMark normalizationFrame:(CGRect)normalizationFrame;
```

其中 waterMark 表示水印图片，normalizationFrame 是相对于视频图像的归一化 frame，frame 的 x、y、width、height 的取值范围都为0 - 1。

Demo 示例：

```
UIImage *image = [UIImage imageNamed:@"watermark"];
[_ugcEdit setWaterMark:image normalizationFrame:CGRectMake(0, 0, 0.3 , 0.3 * image.size.height / image.size.width)];//水印大小占视频宽度的30%，高度根据宽度自适应
```

2. 设置片尾水印

您可以为视频设置片尾水印，并且可以指定片尾水印的位置。
设置片尾水印的方法为：

```
- (void) setTailWaterMark:(UIImage *)tailWaterMark normalizationFrame:(CGRect)normalizationFrame
duration:(CGFloat)duration;
```

其中 tailWaterMark 表示片尾水印图片，normalizationFrame 是相对于视频图像的归一化 frame，frame 的 x、y、width、height 的取值范围都为0 - 1，duration 为水印的持续时长。
Demo 示例：设置水印在片尾中间，持续时间1s。

```
UIImage *tailWaterimage = [UIImage imageNamed:@"tcloud_logo"];
float w = 0.15;
float x = (1.0 - w) / 2.0;
float width = w * videoMsg.width;
float height = width * tailWaterimage.size.height / tailWaterimage.size.width;
float y = (videoMsg.height - height) / 2 / videoMsg.height;
[_ugcEdit setTailWaterMark:tailWaterimage normalizationFrame:CGRectMake(x,y,w,0)
duration:1];
```

版权所有：腾讯云计算（北京）有限责任公司

压缩裁剪

视频码率设置

```
/**
* 设置视频码率
* @param bitrate 视频码率 单位:kbps
* 如果设置了码率，SDK 生成视频会优先使用这个码率，注意码率不要太大或则太小，码率太小视频会模糊不清，码率太大，生成视频体积会很大
* 这里建议设置范围为：600-12000，如果没有调用这个接口，SDK内部会根据压缩质量自动计算码率
*/
- (void) setVideoBitrate:(int)bitrate;
```

视频裁剪

视频编辑类操作都符合同一个操作原则：即先设定操作指定，最后用 generateVideo 将所有指令顺序执行，这种方式可以避免多次重复压缩视频引入的不必要的质量损失。

```
TXVideoEditer* _ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];
// 设置裁剪的起始时间和结束时间
[_ugcEdit setCutFromTime:_videoRangeSlider.leftPos toTime:_videoRangeSlider.rightPos];
// ...
// 生成最终的视频文件
_ugcEdit.generateDelegate = self;
[_ugcEdit generateVideo:VIDEO_COMPRESSED_540P videoOutputPath:_videoOutputPath];
```

输出时指定文件压缩质量和输出路径，输出的进度和结果会通过 `generateDelegate` 以回调的形式通知用户。

高级功能

- 类抖音特效
- 设置背景音乐
- 贴纸字幕
- 图片编辑

Android

最近更新时间：2022-10-25 11:12:50

功能概览

视频编辑包括视频裁剪、时间特效（慢动作、倒放、重复）、滤镜特效（动感光波，暗黑幻影，灵魂出窍，画面分裂）、滤镜风格（唯美，粉嫩，蓝调等）、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能。

相关类介绍

类名	功能
`TXVideoInfoReader`	媒体信息获取
`TXVideoEditer`	视频编辑

使用说明

视频编辑的基本使用流程如下：

1. 设置视频路径。
2. 视频导入。
3. 添加效果。
4. 生成视频到指定文件。
5. 监听生成事件。
6. 资源释放。

视频信息获取

TXVideoInfoReader 的 **getVideoFileInfo** 方法可以获取指定视频文件的一些基本信息, 相关接口如下：

```
/**
* 获取视频信息
* @param videoPath 视频文件路径
* @return
```

```
*/
public TXVideoEditConstants.TXVideoInfo getVideoFileInfo(String videoPath);
```

返回的 TXVideoInfo 定义如下：

```
public final static class TXVideoInfo {
public Bitmap coverImage; // 视频首帧图片
public long duration; // 视频时长(ms)
public long fileSize; // 视频大小(byte)
public float fps; // 视频 fps
public int bitrate; // 视频码率 (kbps)
public int width; // 视频宽度
public int height; // 视频高度
public int audioSampleRate; // 音频码率
}
```

完整示例如下：

```
//sourcePath 为视频源路径
String sourcePath = Environment.getExternalStorageDirectory() + File.separator +
"temp.mp4";
TXVideoEditConstants.TXVideoInfo info = TXVideoInfoReader.getInstance().getVideoF
ileInfo(sourcePath);
```

缩略图获取

缩略图的接口主要用于生成视频编辑界面的预览缩略图，或获取视频封面等。

1. 按个数平分时间获取缩略图

快速导入生成精准缩略图

调用接口如下：

```
/**
* 获取缩略图列表
* @param count 缩略图张数
* @param width 缩略图宽度
* @param height 缩略图高度
* @param fast 缩略图是否关键帧的图片
* @param listener 缩略图的回调函数
*/
public void getThumbnail(int count, int width, int height, boolean fast, TXThumbn
ailListener listener)
```

参数 @param fast 可以使用两种模式：

- 快速出图：输出的缩略图速度比较快，但是与视频对应不精准，传入参数 true。
- 精准出图：输出的缩略图与视频时间点精准对应，但是在高分辨率上速度慢一些，传入参数 false。

完整示例如下：

```
mTXVideoEditer.getThumbnail(TCVideoEditerWrapper.mThumbnailCount, 100, 100, false
, mThumbnailListener);
private TXVideoEditer.TXThumbnailListener mThumbnailListener = new TXVideoEditer.
TXThumbnailListener() {
@Override
public void onThumbnail(int index, long timeMs, final Bitmap bitmap) {
Log.i(TAG, "onThumbnail: index = " + index + ",timeMs:" + timeMs);
//将缩略图放入图片控件上
}
};
```

全功能导入获取缩略图

参见下面 视频导入。

2. 根据时间列表获取缩略图

```
List<Long> list = new ArrayList<>();
list.add(10000L);
list.add(12000L);
list.add(13000L);
list.add(14000L);
list.add(15000L);
TXVideoEditer txVideoEditer = new TXVideoEditer(TCVideoPreviewActivity.this);
txVideoEditer.setVideoPath(mVideoPath);
txVideoEditer.setThumbnailListener(new TXVideoEditer.TXThumbnailListener() {
@Override
public void onThumbnail(int index, long timeMs, Bitmap bitmap) {
Log.i(TAG, "bitmap:" + bitmap + ",timeMs:" + timeMs);
saveBitmap(bitmap, timeMs);
}
});
txVideoEditer.getThumbnailList(list, 200, 200);
```

注意：

- List 中时间点不能超出视频总时长，对于超出总时长的返回最后一张图片。
- 设置的时间点单位是毫秒（ms）。

视频导入

1. 快速导入

快速导入视频，可以直接观看到视频编辑的预览效果，支持视频裁剪、时间特效（慢动作）、滤镜特效、滤镜风格、音乐混音、动态贴纸、静态贴纸、气泡字幕等功能，不支持的功能有时间特效（重复、倒放）。

2. 全功能导入

全功能导入，支持所有的功能，包括时间特效（重复、倒放）。需要为视频先预处理操作。
经过全功能导入后的视频可以精确的 seek 到每个时间点，看到对应的画面，预处理操作同时还可以精确的生成当前时间点视频缩略图。

全功能导入步骤及调用接口如下：

1. 设置精确输出缩略图。

```
/**
```

- 设置预处理输出的缩略图
- /

```
public void setThumbnail(TXVideoEditConstants.TXThumbnail thumbnail)
```

2. 设置输出缩略图的回调。

```
/**
```

- 设置预处理输出缩略图回调

- @param listener
- /
  public void setThumbnailListener(TXThumbnailListener listener)

注意：
缩略图的宽高最好不要设置视频宽高，SDK 内部缩放效率更高。
3. 设置视频预处理回调接口。

```
/**
* 设置视频预处理回调
* @param listener
*/
public void setVideoProcessListener(TXVideoProcessListener listener)
```

4. 进行视频预处理。

```
public void processVideo();
```

完整示例如下：

```
int thumbnailCount = 10; //可以根据视频时长生成缩略图个数
TXVideoEditConstants.TXThumbnail thumbnail = new TXVideoEditConstants.TXThumbnail
();
thumbnail.count = thumbnailCount;
thumbnail.width = 100; // 输出缩略图宽
thumbnail.height = 100; // 输出缩略图高
mTXVideoEditer.setThumbnail(thumbnail); // 设置预处理生成的缩略图
mTXVideoEditer.setThumbnailListener(mThumbnailListener); // 设置缩略图回调
mTXVideoEditer.setVideoProcessListener(this); // 视频预处理进度回调
mTXVideoEditer.processVideo(); // 进行预处理
```

编辑预览

视频编辑提供了 定点预览（将视频画面定格在某一时间点）与区间预览（播放某一时间段 A<=>B 内的视频片段）两种效果预览方式，使用时需要给 SDK 绑定一个 UIView 用于显示视频画面。

1. 设置预览播放的 Layout

```
public void initWithPreview(TXVideoEditConstants.TXPreviewParam param)
```

设置预览Layout时，可以设置两种视频画面渲染模式，在TXVideoEditConstants常量中定义了这两种渲染模式

```
public final static int PREVIEW_RENDER_MODE_FILL_SCREEN = 1; // 填充模式，尽可能充满屏幕不留黑边，所以可能会裁剪掉一部分画面
public final static int PREVIEW_RENDER_MODE_FILL_EDGE = 2; // 适应模式，尽可能保持画面完整，但当宽高比不合适时会有黑边出现
```

2. 定点预览

经过 全功能导入 的视频可以精确预览到某一个时间点的视频画面。

```
public void previewAtTime(long timeMs);
```

3. 区间预览

TXVideoEditer 的 startPlayFromTime 函数用于播放某一时间段 A<=>B 内的视频片段。

```
// 播放某一时间段的视频，从 startTime 到 endTime 的视频片段
public void startPlayFromTime(long startTime, long endTime);
```

4. 预览的暂停与恢复

```
// 暂停播放视频
public void pausePlay();

// 继续播放视频
public void resumePlay();

// 停止播放视频
public void stopPlay();
```

5. 美颜滤镜

您可以给视频添加滤镜效果，例如美白、浪漫、清新等滤镜，demo 提供了16种滤镜选择，同时也可以设置自定义的滤镜。

设置滤镜的方法为：

```
void setFilter(Bitmap bmp)
```

版权所有：腾讯云计算（北京）有限责任公司

其中 Bitmap 为滤镜映射图，bmp 设置为 null，会清除滤镜效果。

```
void setSpecialRatio(float specialRatio)
```

该接口可以调整滤镜程度值，一般为0.0 - 1.0。

```
void setFilter(Bitmap leftBitmap, float leftIntensity, Bitmap rightBitmap, float
rightIntensity, float leftRatio)
```

该接口能够实现组合滤镜，即左右可以添加不同的滤镜。leftBitmap 为左侧滤镜、leftIntensity 为左侧滤镜程度值；rightBitmap 为右侧滤镜、rightIntensity 为右侧滤镜程度值；leftRatio 为左侧滤镜所占的比例，一般为0.0 - 1.0。当 leftBitmap 或 rightBitmap 为 null，则该侧清除滤镜效果。

6. 水印

1. 设置全局水印

您可以为视频设置水印图片，并且可以指定图片的位置。

设置水印的方法为：

```
public void setWaterMark(Bitmap waterMark, TXVideoEditConstants.TXRect rect);
```

其中 waterMark 表示水印图片，rect 是相对于视频图像的归一化 frame，frame 的 x、y、width、height 的取值范围都为 0 - 1。

Demo 示例：

```
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
mTXVideoEditer.setWaterMark(mWaterMarkLogo, rect);
```

2. 设置片尾水印

您可以为视频设置片尾水印，并且可以指定片尾水印的位置。
设置片尾水印的方法为：

```
setTailWaterMark(Bitmap tailWaterMark, TXVideoEditConstants.TXRect txRect, int du
ration);
```

其中 tailWaterMark 表示片尾水印图片，txRect 是相对于视频图像的归一化 txRect，txRect 的 x、y、width 取值范围都为0 - 1，duration 为水印的持续时长，单位：秒。

版权所有：腾讯云计算（北京）有限责任公司

Demo 实例：设置水印在片尾中间，持续3秒

```
Bitmap tailWaterMarkBitmap = BitmapFactory.decodeResource(getResources(), R.drawa
ble.tcloud_logo);
TXVideoEditConstants.TXRect txRect = new TXVideoEditConstants.TXRect();
txRect.x = (mTXVideoInfo.width - tailWaterMarkBitmap.getWidth()) / (2f * mTXVideo
Info.width);
txRect.y = (mTXVideoInfo.height - tailWaterMarkBitmap.getHeight()) / (2f * mTXVid
eoInfo.height);
txRect.width = tailWaterMarkBitmap.getWidth() / (float) mTXVideoInfo.width;
mTXVideoEditer.setTailWaterMark(tailWaterMarkBitmap, txRect, 3);
```

压缩裁剪

视频码率设置

目前支持自定义视频的码率，这里建议设置的范围600 - 12000kbps，如果设置了这个码率，SDK 最终压缩视频时会优先选取这个码率，注意码率不要太大或太小，码率太大，视频的体积会很大，码率太小，视频会模糊不清。

```
public void setVideoBitrate(int videoBitrate);
```

视频裁剪

设置视频裁剪的开始时间和结束时间

```
/**
* 设置视频剪切范围
* @param startTime 视频剪切的开始时间(ms)
* @param endTime 视频剪切的结束时间(ms)
*/
public void setCutFromTime(long startTime, long endTime)

// ...
// 生成最终的视频文件
public void generateVideo(int videoCompressed, String videoOutputPath)
```

参数 videoCompressed 在 TXVideoEditConstants 中可选常量。

```
VIDEO_COMPRESSED_360P ——压缩至360P分辨率（360*640）
VIDEO_COMPRESSED_480P ——压缩至480P分辨率（640*480）
VIDEO_COMPRESSED_540P ——压缩至540P分辨率 （960*540）
VIDEO_COMPRESSED_720P ——压缩至720P分辨率 （1280*720）
VIDEO_COMPRESSED_1080P ——压缩至1080P分辨率 （1920*1080）
```

版权所有：腾讯云计算（北京）有限责任公司

如果源视频的分辨率小于设置的常量对应的分辨率，按照原视频的分辨率。
如果源视频的分辨率大于设置的常量对象的分辨率，进行视频压缩至相应分辨率。

资源释放

当您不再使用 mTXVideoEditer 对象时，一定要记得调用 **release()** 释放它。

高级功能

- 类抖音特效
- 设置背景音乐
- 贴纸字幕
- 图片编辑

视频拼接
iOS

最近更新时间：2022-05-24 15:47:54

复用现有 UI

视频拼接器具有比较复杂的交互逻辑，这也决定了其 UI 复杂度很高，所以我们比较推荐复用 SDK 开发包中的 UI 源码。VideoJoiner 目录包含短视频拼接器的 UI 源码。

- **VideoJoinerController**：用于实现上图中的视频拼接列表，支持上下拖拽调整顺序。
- **VideoJoinerCell**：用于实现拼接列表中的每一个视频片段。
- **VideoEditPrevController**：用于预览拼接后的视频观看效果。

自己实现 UI

如果您不考虑复用我们开发包中的 UI 代码，想自己实现 UI 部分，则可以参考如下的攻略进行对接。

1. 选择视频文件

Demo 中使用了 QBImagePicker 这样一个开源库实现了多个文件的选择功能，相关代码在 Demo 的 MainViewController 里有所体现。

2. 设置预览 View

视频合成需要创建 TXVideoJoiner 对象，同 TXUGCEditer 类似，预览功能也需要上层提供预览 UIView：

```
//准备预览 View
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = _videoPreview.renderView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;
// 创建 TXVideoJoiner 对象并设置预览 view
TXVideoJoiner* _videoJoin = [[TXVideoJoiner alloc] initWithPreview:param];
_videoJoin.previewDelegate = _videoPreview;
// 设置待拼接的视频文件组 _composeArray，也就是第一步中选择的若干个文件
[_videoJoin setVideoPathList:_composeArray];
```

设置好预览 view 同时传入待合成的视频文件数组后，可以开始播放预览，合成模块提供了一组接口来做视频的播放预览：

- `startPlay`：表示视频播放开始。

- `pausePlay` ：表示视频播放暂停。
- `resumePlay` ：表示视频播放恢复。

3. 生成最终文件

预览效果满意后调用生成接口即可生成合成后的文件：

```
_videoJoin.joinerDelegate = self;
[_videoJoin joinVideo:VIDEO_COMPRESSED_540P videoOutputPath:_outFilePath];
```

合成时指定文件压缩质量和输出路径，输出的进度和结果会通过 `joinerDelegate` 以回调的形式通知用户。

Android

最近更新时间：2022-05-24 15:51:21

复用现有 UI

视频拼接器具有比较复杂的交互逻辑，这也决定了其 UI 复杂度很高，所以我们比较推荐复用 SDK 开发包中的 UI 源码。 videojoiner 目录包含短视频拼接器的 UI 源码。

- TCVideoJoinerActivity 用于实现上图中的视频拼接列表，支持上下拖拽调整顺序。
- TCVideoJoinerPreviewActivity 用于预览拼接后的视频观看效果。

自己实现 UI

如果您不考虑复用我们开发包中的 UI 代码，决心自己实现 UI 部分，则可以参考如下的攻略进行对接：

1. 选择视频文件

自己实现多选文件功能。

2. 设置预览 View

视频合成需要创建 TXVideoJoiner 对象，同 TXVideoEditer 类似，预览功能也需要上层提供预览 FrameLayout：

```
//准备预览 View
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreviewPar
am();
param.videoView = mVideoView;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
// 创建 TXUGCJoiner 对象并设置预览 view
TXVideoJoiner mTXVideoJoiner = new TXVideoJoiner(this);
mTXVideoJoiner.setTXVideoPreviewListener(this);
mTXVideoJoiner.initWithPreview(param);
// 设置待拼接的视频文件组 mVideoSourceList，也就是第一步中选择的若干个文件
mTXVideoJoiner.setVideoPathList(mVideoSourceList);
```

设置好预览 view 同时传入待合成的视频文件数组后，可以开始播放预览，合成模块提供了一组接口来做视频的播放预览：

- startPlay：表示视频播放开始。
- pausePlay：表示视频播放暂停。
- resumePlay：表示视频播放恢复。

3. 生成最终文件

预览效果满意后调用生成接口即可生成合成后的文件：

```
mTXVideoJoiner.setVideoJoinerListener(this);
mTXVideoJoiner.joinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mVideoOutput
Path);
```

合成时指定文件压缩质量和输出路径，输出的进度和结果会通过 TXVideoJoiner.TXVideoJoinerListener 以回调的形式通知用户。

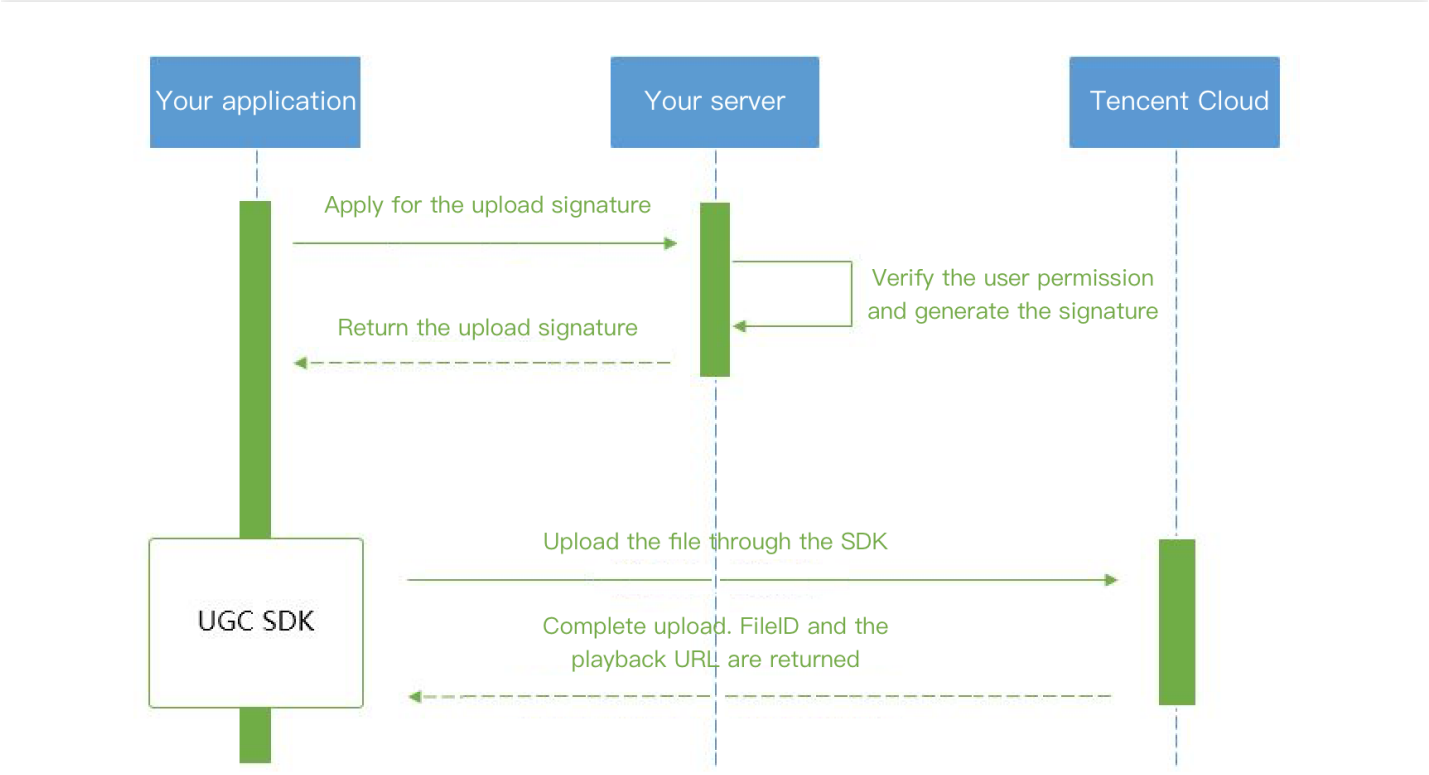
上传和播放
签名派发

最近更新时间：2020-08-28 16:53:58

客户端视频上传，是指 App 的最终用户将本地视频直接上传到腾讯云点播。客户端上传的详细介绍请参考点播 客户端上传指引，本文将以最简洁的方式介绍客户端上传的签名生成方法。

总体介绍

客户端上传的整体流程如下图所示：



为了支持客户端上传，开发者需要搭建两个后台服务：签名派发服务和事件通知接收服务。

- 客户端首先向签名派发服务请求上传签名。
- 签名派发服务校验该用户是否有上传权限，若校验通过，则生成签名并下发；否则返回错误码，上传流程结束。
- 客户端拿到签名后使用短视频 SDK 中集成的上传功能来上传视频。
- 上传完成后，点播后台会发送 上传完成事件通知 给开发者的事件通知接收服务。
- 如果签名派发服务在签名中指定了视频处理 任务流，点播服务会在视频上传完成后根据指定流程自动进行视频处理。短视频场景下的视频处理一般为 AI 鉴黄。

版权所有：腾讯云计算（北京）有限责任公司

- 视频处理完成之后，点播后台会发送 任务流状态变更事件通知 给开发者的事件通知接收服务。

至此整个视频上传 - 处理流程结束。

签名生成

有关客户端上传签名的详细介绍请参考点播 客户端上传签名。

签名派发服务实现示例

```
/**
* 计算签名
*/
function createFileUploadSignature({ timeStamp = 86400, procedure = '', classId =
0, oneTimeValid = 0, sourceContext = '' }) {
// 确定签名的当前时间和失效时间
let current = parseInt((new Date()).getTime() / 1000)
let expired = current + timeStamp; // 签名有效期：1天
// 向参数列表填入参数
let arg_list = {
//required
secretId: this.conf.SecretId,
currentTimeStamp: current,
expireTime: expired,
random: Math.round(Math.random() * Math.pow(2, 32)),
//opts
procedure,
classId,
oneTimeValid,
sourceContext
}
// 计算签名
let orignal = querystring.stringify(arg_list);
let orignal_buffer = new Buffer(orignal, "utf8");
let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
let hmac_buffer = hmac.update(orignal_buffer).digest();
let signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");
return signature;
}
/**
* 响应签名请求
*/
```

```
function getUploadSignature(req, res) {
res.json({
code: 0,
message: 'ok',
data: {
signature: gVodHelper.createFileUploadSignature({})
}
});
}
```

视频上传
iOS

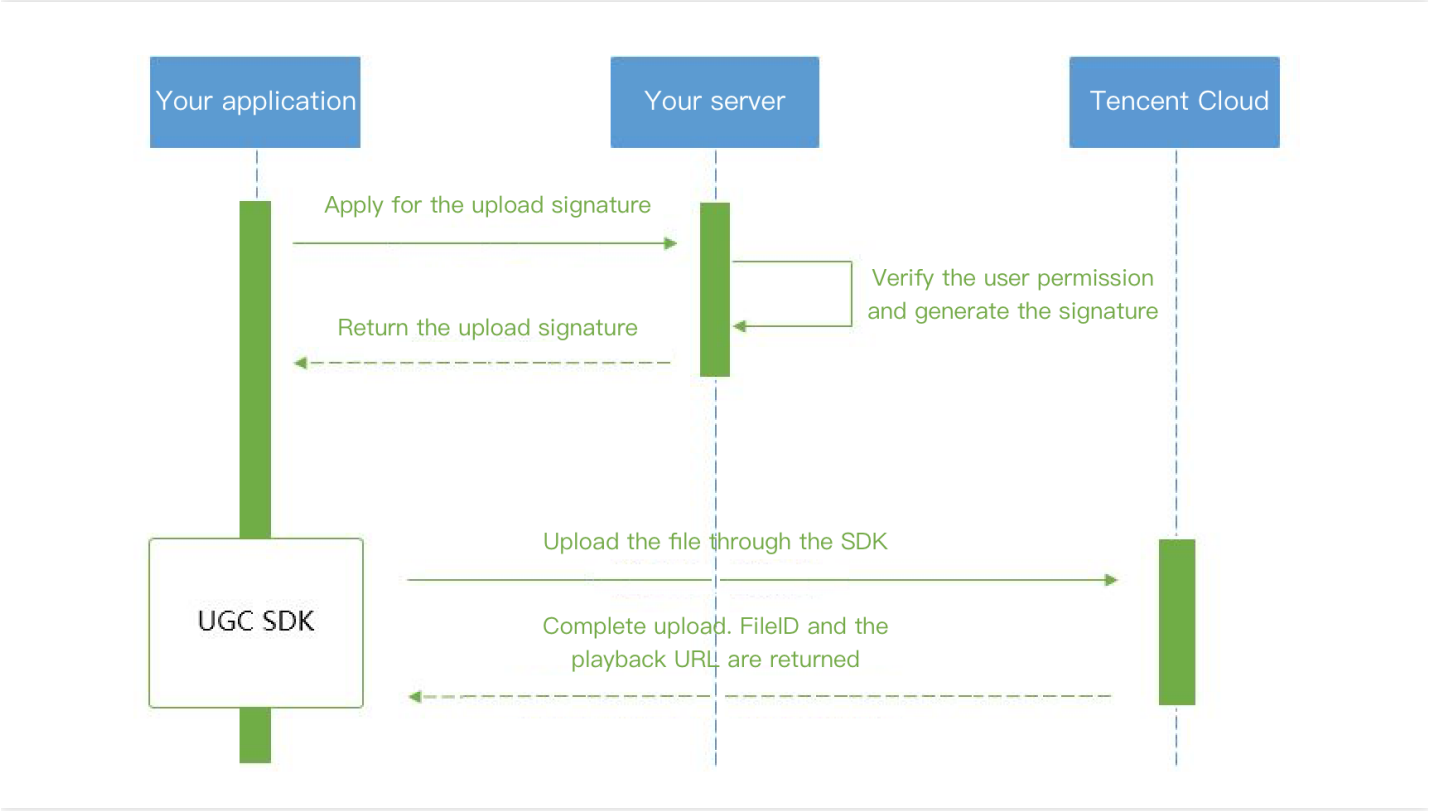
最近更新时间：2022-05-24 16:09:21

计算上传签名

客户端视频上传，是指 App 的最终用户将本地视频直接上传到腾讯云点播。客户端上传的详细介绍请参考点播 客户端上传指引，本文将以最简洁的方式介绍客户端上传的签名生成方法。

总体介绍

客户端上传的整体流程如下图所示：



为了支持客户端上传，开发者需要搭建两个后台服务：签名派发服务和事件通知接收服务。

- 客户端首先向签名派发服务请求上传签名。
- 签名派发服务校验该用户是否有上传权限，若校验通过，则生成签名并下发；否则返回错误码，上传流程结束。
- 客户端拿到签名后使用短视频 SDK 中集成的上传功能来上传视频。
- 上传完成后，点播后台会发送 上传完成事件通知 给开发者的事件通知接收服务。

- 如果签名派发服务在签名中指定了视频处理 任务流，点播服务会在视频上传完成后根据指定流程自动进行视频处理。短视频场景下的视频处理一般为 AI 鉴黄。
- 视频处理完成之后，点播后台会发送 任务流状态变更事件通知 给开发者的事件通知接收服务。

至此整个视频上传-处理流程结束。

签名生成

有关客户端上传签名的详细介绍请参见点播 客户端上传签名。

签名派发服务实现示例

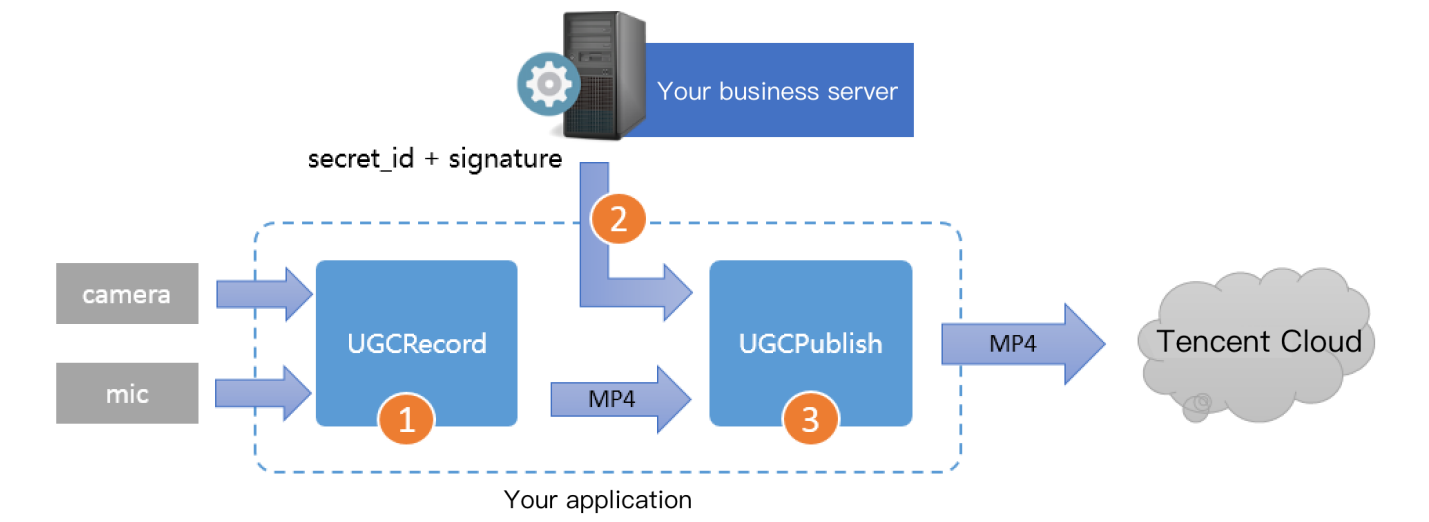
```
/**
* 计算签名
*/
function createFileUploadSignature({ timeStamp = 86400, procedure = '', classId =
0, oneTimeValid = 0, sourceContext = '' }) {
// 确定签名的当前时间和失效时间
let current = parseInt((new Date()).getTime() / 1000)
let expired = current + timeStamp; // 签名有效期：1天
// 向参数列表填入参数
let arg_list = {
//required
secretId: this.conf.SecretId,
currentTimeStamp: current,
expireTime: expired,
random: Math.round(Math.random() * Math.pow(2, 32)),
//opts
procedure,
classId,
oneTimeValid,
sourceContext
}
// 计算签名
let orignal = querystring.stringify(arg_list);
let orignal_buffer = new Buffer(orignal, "utf8");
let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
let hmac_buffer = hmac.update(orignal_buffer).digest();
let signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");
return signature;
}
/**
* 响应签名请求
*/
function getUploadSignature(req, res) {
res.json({
```

```
code: 0,
message: 'ok',
data: {
signature: gVodHelper.createFileUploadSignature({})
}
});
}
```

对接流程

短视频发布

将 MP4 文件上传到腾讯视频云，并获得在线观看 URL， 腾讯视频云支持视频观看的就近调度、秒开播放、动态加速 以及海外接入等要求，从而确保优质的观看体验。



- 第一步：使用 TXUGCRecord 接口录制一段小视频，录制结束后会生成一个小视频文件（MP4）回调给客户。
- 第二步：您的 App 向您的业务服务器申请上传签名。上传签名是 App 将 MP4 文件上传到腾讯云视频分发平台的“许可证”，为了确保安全性，这些上传签名都要求由您的业务 Server 进行签发，而不能由终端 App 生成。
- 第三步：使用 TXUGCPublish 接口发布视频，发布成功后 SDK 会将观看地址的 URL 回调给您。

特别注意

- App 千万不要把计算上传签名的 SecretID 和 SecretKey 写在客户端的代码里，这两个关键信息泄露将导致安全隐患，如恶意攻击者一旦破解 App 获取该信息，就可以免费使用您的流量和存储服务。
- 正确的做法是在您的服务器上用 SecretID 和 SecretKey 生成一次性的上传签名然后将签名交给 App。因为服务器一般很难被攻陷，所以安全性是可以保证的。
- 发布短视频时，请务必保证正确传递 Signature 字段，否则会发布失败。

版权所有：腾讯云计算（北京）有限责任公司

对接攻略

1. 选择视频

可以接着上篇文档中的录制或者编辑，把生成的视频进行上传，或者可以选择手机本地的视频进行上传。

2. 压缩视频

对选择的视频进行压缩，使用 TXVideoEditer.generateVideo(int videoCompressed, String videoOutputPath) 接口，支持4种分辨率的压缩，后续会增加自定义码率的压缩。

3. 发布视频

把刚才生成的 MP4 文件发布到腾讯云上，App 需要拿到上传文件用的短期有效上传签名，这部分有独立的文档介绍，详情请参考 签名派发。
TXUGCPublish（位于 TXUGCPublish.h）负责将 MP4 文件发布到腾讯云视频分发平台上，以确保视频观看的就近调度、秒开播放、动态加速以及海外接入等需求。

```
TXPublishParam * param = [[TXPublishParam alloc] init];
param.signature = _signature; // 需要填写第四步中计算的上传签名
// 录制生成的视频文件路径 TXVideoRecordListener 的 onRecordComplete 回调中可以获取
param.videoPath = _videoPath;
// 录制生成的视频首帧预览图路径。值为通过调用 startRecord 指定的封面路径，或者指定一个路径，然后将TXVideoRecordListener 的 onRecordComplete 回调中获取到的 UIImage 保存到指定路径下，可以置为 nil。
param.coverPath = _coverPath;
TXUGCPublish *_ugcPublish = [[TXUGCPublish alloc] init];
// 文件发布默认是采用断点续传
_ugcPublish.delegate = self; // 设置 TXVideoPublishListener 回调
[_ugcPublish publishVideo:param];
```

发布的过程和结果是通过 TXVideoPublishListener（位于 TXUGCPublishListener.h 头文件中定义）接口反馈出来的：

- onPublishProgress 用于反馈文件发布的进度，参数 uploadBytes 表示已经上传的字节数，参数 totalBytes 表示需要上传的总字节数。

```
@optional
-(void) onPublishProgress:(NSInteger)uploadBytes totalBytes: (NSInteger)totalBytes;
```

- onPublishComplete 用于反馈发布结果，TXPublishResult 的字段 errCode 和 descMsg 分别表示错误码和错误描述信息，videoURL 表示短视频的点播地址，coverURL 表示视频封面的云存储地址，videoId 表示视频文件云存储 Id，您可以通过这个 Id 调用点播 服务端API接口。

```
@optional
-(void) onPublishComplete:(TXPublishResult*)result;
```

- 发布结果
  通过 错误码表 来确认短视频发布的结果。

4. 播放视频

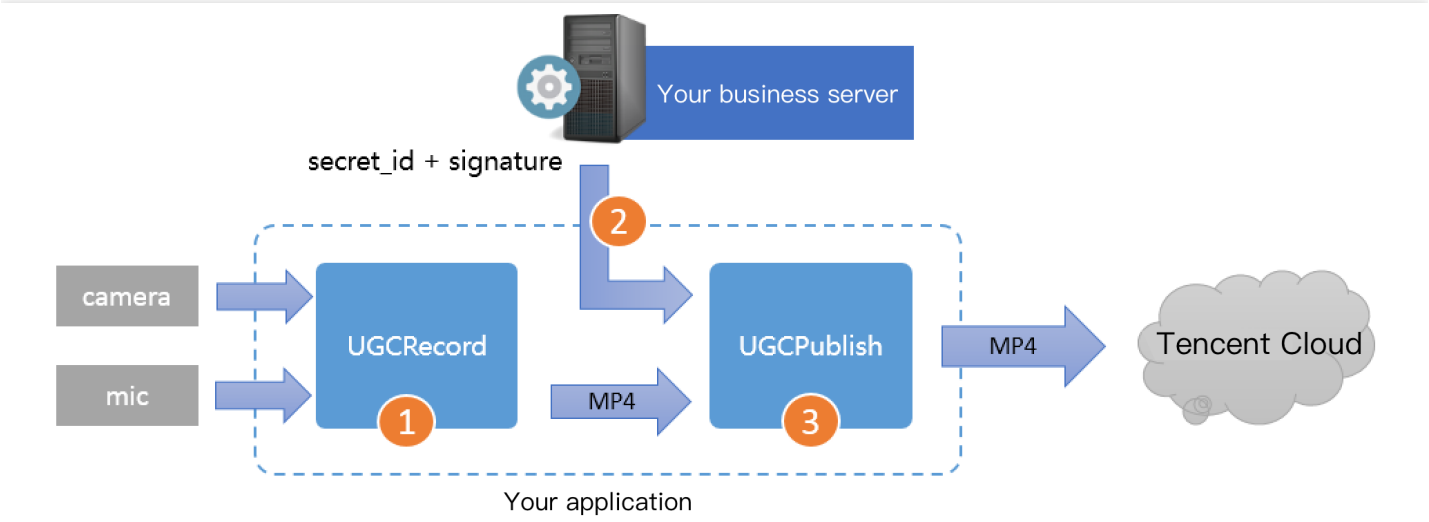
第3步上传成功后，会返回视频的 fileId，播放地址 URL，封面 URL。用 点播播放器 可以直接传入 fileId 播放，或者 URL 播放。

Android

最近更新时间：2022-05-24 16:09:10

对接流程

短视频发布：将 MP4 文件上传到腾讯视频云，并获得在线观看 URL， 腾讯视频云满足视频观看的就近调度、秒开播放、动态加速以及海外接入等要求，确保了优质的观看体验。



- Step1. 使用 TXUGCRecord 接口录制一段小视频，录制结束后会生成一个小视频文件（MP4）回调给客户。
- Step2. App 向您的业务服务器申请上传签名（App 将 MP4 文件上传到腾讯云视频分发平台的“许可证”）。为了确保安全性，上传签名由您的业务 Server 进行签发，而不能由终端 App 生成。
- Step3. 使用 TXUGCPublish 接口发布视频，发布成功后，SDK 会将观看地址的 URL 回调给您。

注意事项

- App 不能把计算上传签名的 SecretID 和 SecretKey 写在客户端代码里，这两个关键信息泄露将导致安全隐患，如果恶意攻击者通过破解 App 来获取该信息，则可以免费使用您的流量和存储服务。
- 正确的做法是在您的服务器上，用 SecretID 和 SecretKey 生成一次性的上传签名，然后将签名交给 App。
- 发布短视频时，请务必正确传递 Signature 字段，否则会发布失败。

对接攻略

请参见 Android 上传 SDK 来接入短视频上传功能。

版权所有：腾讯云计算（北京）有限责任公司

1. 选择视频

将录制、编辑、拼接后的视频进行上传，或者选择本地视频进行上传。

2. 压缩视频

- 压缩视频会减小视频文件的大小，同时也会降低视频的清晰度，您可以按需决定是否进行压缩。
- 对视频进行压缩，使用 TXVideoEditer.generateVideo(int videoCompressed, String videoOutputPath)接口，支持4种分辨率的压缩，后续会增加自定义码率的压缩。

3. 发布视频

将生成的 MP4 文件发布到腾讯云上，App 需要拿到上传文件的短期有效上传签名，详细请参见 签名派发。
TXUGCPublish（位于 TXUGCPublish.java）负责将 MP4 文件发布到腾讯云视频分发平台上，以满足视频观看的就近调度、秒开播放、动态加速以及海外接入等要求。

```
mVideoPublish = new TXUGCPublish(TCVideoPublisherActivity.this.getApplicationCont
ext());
// 文件发布默认是采用断点续传
TXUGCPublishTypeDef.TXPublishParam param = new TXUGCPublishTypeDef.TXPublishParam
();
param.signature = mCosSignature; // 需要填写第四步中计算的上传签名
// 录制生成的视频文件路径，ITXVideoRecordListener 的 onRecordComplete 回调中可以获取
param.videoPath = mVideoPath;
// 录制生成的视频首帧预览图，ITXVideoRecordListener 的 onRecordComplete 回调中可以获取
param.coverPath = mCoverPath;
mVideoPublish.publishVideo(param);
```

发布的过程和结果通过 TXRecordCommon.ITXVideoPublishListener（位于 TXRecordCommon.java 头文件中）接口反馈：

- onPublishProgress 用于反馈发布进度，参数 uploadBytes 表示已上传的字节数，参数 totalBytes 表示需要上传的总字节数。

```
void onPublishProgress(long uploadBytes, long totalBytes);
```

- onPublishComplete 用于反馈发布结果。

```
void onPublishComplete(TXPublishResult result);
```

参数 TXPublishResult 中的字段及含义如下表所示：

字段	含义
errCode	错误码。
descMsg	错误描述信息。
videoURL	短视频的点播地址。
coverURL	视频封面的云存储地址。
videoId	视频文件云存储 ID，您可以通过这个 ID 调用云点播 服务端 API 接口。

- 通过 错误码表 来确认短视频的发布结果。

4. 播放视频

第3步发布视频成功后，会返回视频的 fileId、播放地址 URL 及封面 URL，然后在 点播播放器 中传入 fileId 或 URL 进行视频播放。

视频播放
iOS

最近更新时间：2022-11-21 17:28:41

产品概述

腾讯云视立方 iOS 播放器组件是腾讯云开源的一款播放器组件，简单几行代码即可拥有类似腾讯视频强大的播放功能，包括横竖屏切换、清晰度选择、手势和小窗等基础功能，还支持视频缓存，软硬解切换和倍速播放等特殊功能，相比系统播放器，支持格式更多，兼容性更好，功能更强大，同时还具备首屏秒开、低延迟的优点，以及视频缩略图等高级能力。

若播放器组件满足不了您的业务的个性化需求，且您具有一定的开发经验，可以集成 视立方播放器 SDK，自定义开发播放器界面和播放功能。

准备工作

1. 开通 云点播 相关服务，未注册用户可注册账号 试用。
2. 下载 Xcode，如您已下载可略过该步骤，您可以进入 App Store 下载安装。
3. 下载 Cocoapods，如您已下载可略过该步骤，您可以进入 Cocoapods官网 按照指引进行安装。

通过本文您可以学会

- 如何集成腾讯云视立方 iOS 播放器组件
- 如何创建和使用播放器

集成准备

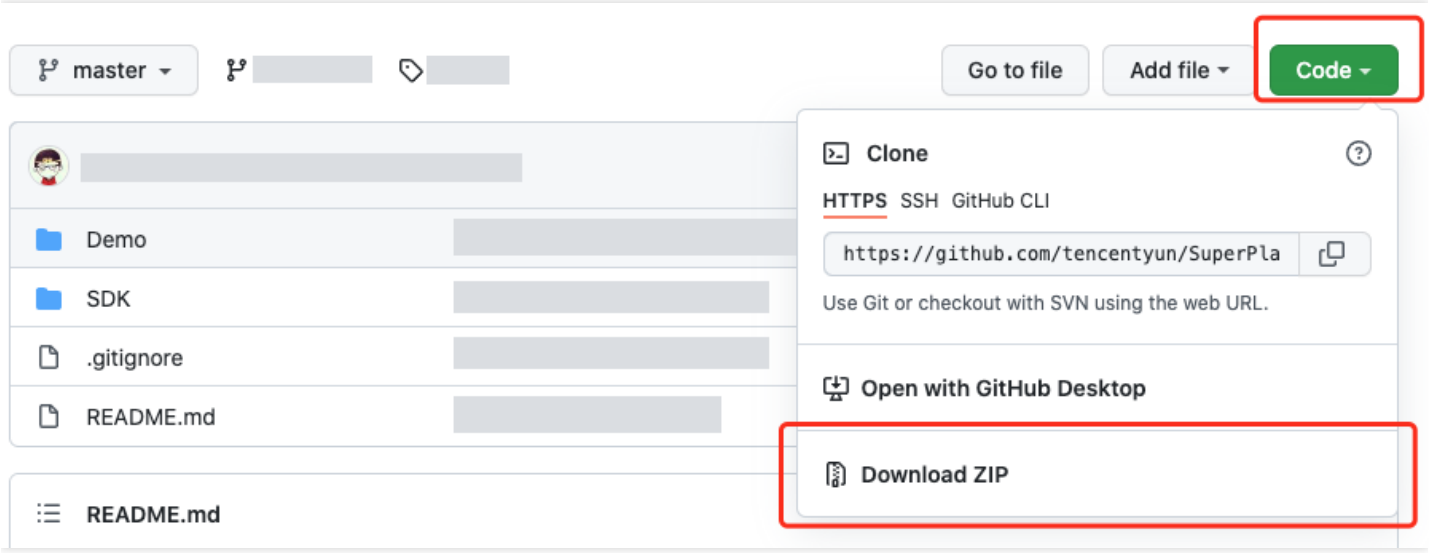
步骤1：项目下载

腾讯云视立方 iOS 播放器的项目地址是 LiteAVSDK/Player_iOS 。

您可通过 **下载播放器组件 ZIP 包** 或 **Git 命令下载** 的方式下载腾讯云视立方 iOS 播放器组件项目工程。

- 下载播放器组件 ZIP 包
- Git 命令下载

您可以直接下载播放器组件 ZIP 包，单击页面的 **Code** > **Download ZIP** 下载



步骤2：集成指引

本步骤，用于指导用户如何集成播放器，推荐用户选择使用 **Cocoapods 集成** 或者 **手动下载 SDK** 再将其导入到您当前的工程项目中。

- Cocoapods 集成
- 手动下载 SDK

1. 本项目支持 Cocoapods 安装，只需要将如下代码添加到 Podfile 中：
   （1）Pod 方式直接集成 SuperPlayer

```
pod 'SuperPlayer
```

   如果您需要依赖 Player 版，可以在 podfile 文件中添加如下依赖：

```
pod 'SuperPlayer/Player'
```

   如果您需要依赖专业版，可以在 podfile 文件中添加如下依赖：

```
pod 'SuperPlayer/Professional'
```

2. 执行 `pod install` 或 `pod update` 。

步骤3：使用播放器功能

版权所有：腾讯云计算（北京）有限责任公司

本步骤，用于指导用户创建和使用播放器，并使用播放器进行视频播放。

1. **创建播放器：**

   播放器主类为 `SuperPlayerView`，创建后即可播放视频。

```
// 引入头文件
#import <SuperPlayer/SuperPlayer.h>
// 创建播放器
_playerView = [[SuperPlayerView alloc] init];
// 设置代理，用于接受事件
_playerView.delegate = self;
// 设置父 View，_playerView 会被自动添加到 holderView 下面
_playerView.fatherView = self.holderView;
```

2. **配置 License 授权**

   若您已获得相关 License 授权，需在 腾讯云视立方控制台 获取 License URL 和 License Key。

   若您暂未获得 License 授权，需联系我们获取相关授权。

   获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 `-[AppDelegate application:didFinishLaunchingWithOptions:]` 中进行如下设置：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
NSString * const licenceURL = @"<获取到的licenseUrl>";
NSString * const licenceKey = @"<获取到的key>";

//TXLiveBase 位于 "TXLiveBase.h" 头文件中
[TXLiveBase setLicenceURL:licenceURL key:licenceKey];
NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
```

3. **播放视频：**

   本步骤，用于指导用户播放视频，腾讯云视立方 iOS 播放器组件支持 云点播 FileId 或者 使用 URL 进行播放，推荐您选择**集成 FileId** 使用更完善的能力。

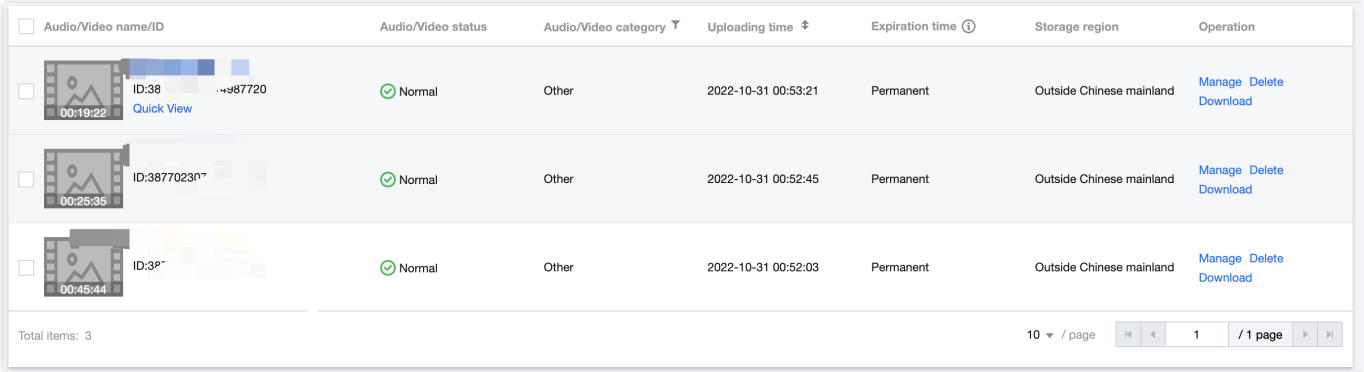
   - 云点播 FileId 播放
   - 使用 URL 播放

   视频 FileId 在一般是在视频上传后，由服务器返回：

   i. 客户端视频发布后，服务器会返回 FileId 到客户端。

   ii. 服务端视频上传时，在 确认上传的通知中包含对应的 FileId。

   如果文件已存在腾讯云，则可以进入 媒资管理，找到对应的文件，查看 FileId。如下图所示，ID 即表示 FileId：



>!

>- 通过 FileId 播放时，需要首先使用 Adaptive-HLS(10) 转码模板对视频进行转码，或者使用播放器组件签名 psign 指定播放的视频，否则可能导致视频播放失败。转码教程和说明可参见 用播放器组件播放视频，psign 生成教程可参见 psign 教程。

>- 若您在通过 FileId 播放时出现“no v4 play info”异常，则说明您可能存在上述问题，建议您根据上述教程调整。同时您也可以直接获取源视频播放链接，通过 URL 播放 的方式实现播放。

>- **未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放。**

```
//在未开启防盗链进行播放的过程中，如果出现了“no v4 play info”异常，建议您使用Adaptive-HLS(10)转码模板对视频进行转码，或直接获取源视频播放链接通过url方式进行播放。

SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// 配置 AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"5285890799710173650"; // 配置 FileId
//私有加密播放需填写 psign，  psign 即播放器组件签名，签名介绍和生成方式参见链接：https://www.tencentcloud.com/document/product/266/38099
//model.videoId.pSign = @"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZmlsZUlkIjoiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsImV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-LzU";
[_playerView playWithModelNeedLicence:model];
```

- **退出播放：**

  当不需要播放器时，调用 `resetPlayer` 清理播放器内部状态，释放内存。

```
[_playerView resetPlayer];
```

您已完成了腾讯云视立方 iOS 播放器组件的创建、播放视频和退出播放的能力集成。

功能使用

1、全屏播放

播放器组件支持全屏播放，在全屏播放场景内，同时支持锁屏、手势控制音量和亮度、弹幕、截屏、清晰度切换等功能设置。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** 中体验，单击界面右下角**全屏**即可进入全屏播放界面。

在窗口播放模式下，可通过调用下述接口进入全屏播放模式：

```
- (void)superPlayerFullScreenChanged:(SuperPlayerView *)player {
//用户可在此自定义切换全屏后的逻辑
}
```

全屏播放界面功能介绍

- 返回窗口模式
- 锁屏
- 弹幕
- 截屏
- 清晰度切换

通过**返回**即可返回窗口播放模式，单击后 SDK 处理完全屏切换的逻辑后会触发的代理方法为：

```
// 返回事件
- (void)superPlayerBackAction:(SuperPlayerView *)player;
单击左上角返回按钮触发
// 全屏改变通知
- (void)superPlayerFullScreenChanged:(SuperPlayerView *)player;
```

2、悬浮窗播放

播放器组件支持悬浮窗小窗口播放，可以在切换到应用内其它页面时，不打断视频播放功能。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** 中体验，单击界面左上角**返回**，即可体验悬浮窗播放功能。

```
// 如果在竖屏且正在播放的情况下单击返回按钮会触发接口
[SuperPlayerWindowShared setSuperPlayer:self.playerView];
[SuperPlayerWindowShared show];
// 单击浮窗返回窗口触发的代码接口
SuperPlayerWindowShared.backController = self;
```

3、视频封面

播放器组件支持用户自定义视频封面，用于在视频接收到首帧画面播放回调前展示。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** > **自定义封面演示** 视频中体验。

- 当播放器组件设置为自动播放模式 `PLAY_ACTION_AUTO_PLAY` 时，视频自动播放，此时将在视频首帧加载出来之前展示封面。
- 当播放器组件设置为手动播放模式 `PLAY_ACTION_MANUAL_PLAY` 时，需用户单击**播放**后视频才开始播放。在单击**播放**前将展示封面；在单击**播放**后到视频首帧加载出来前也将展示封面。

视频封面支持使用网络 URL 地址或本地 File 地址，使用方式可参见下述指引。若您通过 FileID 的方式播放视频，则可直接在云点播内配置视频封面。

```
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
SuperPlayerVideoId *videoId = [SuperPlayerVideoId new];
videoId.fileId = @"8602268011437356984";
model.appId = 1400329071;
model.videoId = videoId;
//播放模式，可设置自动播放模式：PLAY_ACTION_AUTO_PLAY，手动播放模式：PLAY_ACTION_MANUAL_PLAY
model.action = PLAY_ACTION_MANUAL_PLAY;
//设定封面的地址为网络url地址，如果coverPictureUrl不设定，那么就会自动使用云点播控制台设置的封面
model.customCoverImageUrl = @"http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500005830/cc1e282086022680110873365l8/MXUW1a5I9TsA.png";
[self.playerView playWithModelNeedLicence:model];
```

4、视频列表轮播

播放器组件支持视频列表轮播，即在给定一个视频列表后：

- 支持按顺序循环播放列表中的视频，播放过程中支持自动播放下一集也支持手动切换到下一个视频。
- 列表中最后一个视频播放完成后将自动开始播放列表中的第一个视频。

功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** > **视频列表轮播演示** 视频中体验。

```
//步骤1：构建轮播数据的 NSMutableArray
NSMutableArray *modelArray = [NSMutableArray array];
SuperPlayerModel *model = [SuperPlayerModel new];
SuperPlayerVideoId *videoId = [SuperPlayerVideoId new];
videoId.fileId = @"8602268011437356984";
model.appId = 1252463788;
model.videoId = videoId;
[modelArray addObject:model];
model = [SuperPlayerModel new];
videoId = [SuperPlayerVideoId new];
```

```
videoId.fileId = @"4564972819219071679";
model.appId = 1252463788;
model.videoId = videoId;
[modelArray addObject:model];
//步骤2：调用 SuperPlayerView 的轮播接口
[self.playerView playWithModelListNeedLicence:modelArray isLoopPlayList:YES startIndex:0];
```

```
(void)playWithModelListNeedLicence:(NSArray *)playModelList isLoopPlayList:(BOOL)isLoop startIndex:(NSInteger)index;
```

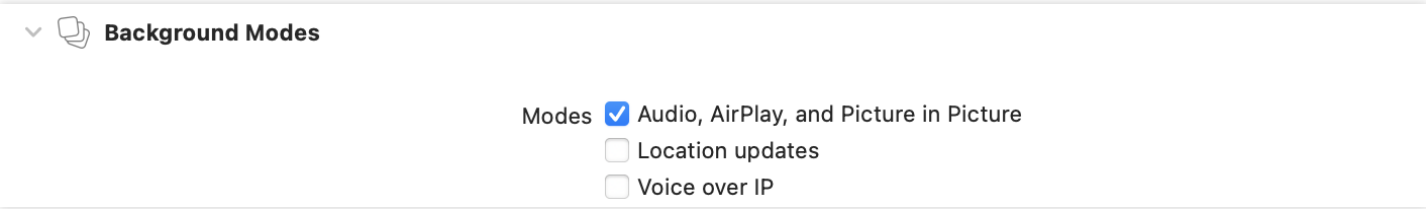
接口参数说明

参数名	类型	描述
playModelList	NSArray *	轮播数据列表
isLoop	Boolean	是否循环
index	NSInteger	开始播放的视频索引

5、画中画功能

画中画（PictureInPicture）在 iOS 9就已经推出了，不过之前都只能在 iPad 上使用，iPhone 要使用画中画需更新到 iOS 14才能使用。

目前腾讯云播放器可以支持应用内和应用外画中画能力，极大的满足用户的诉求。使用前需要开通后台模式，步骤为：XCode 选择对应的 Target -> Signing & Capabilities -> Background Modes，勾选“Audio, AirPlay, and Picture in Picture”。



使用画中画能力代码示例：

```
// 进入画中画
if (![TXVodPlayer isSupportPictureInPicture]) {
return;
}
[_vodPlayer enterPictureInPicture];
// 退出画中画
[_vodPlayer exitPictureInPicture];
```

6、视频试看

播放器组件支持视频试看功能，可以适用于非 VIP 试看等场景，开发者可以传入不同的参数来控制视频试看时长、提示信息、试看结束界面等。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件 > 试看功能演示** 视频中体验。

```
//步骤1：创建试看model
TXVipWatchModel *model = [[TXVipWatchModel alloc] init];
model.tipTtitle = @"可试看15秒，开通VIP观看完整视频";
model.canWatchTime = 15;
//步骤2：设置试看model
self.playerView.vipWatchModel = model;
//步骤3：调用方法展示试看功能
[self.playerView showVipTipView];
```

TXVipWatchModel 类参数说明：

参数名	类型	描述
tipTtitle	NSString	试看提示信息
canWatchTime	float	试看时长，单位为妙

7、动态水印

播放器组件支持在播放界面添加不规则跑动的文字水印，有效防盗录。全屏播放模式和窗口播放模式均可展示水印，开发者可修改水印文本、文字大小、颜色。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件 > 动态水印演示** 视频中体验。

```
//步骤1：创建视频源信息 model
SuperPlayerModel * playermodel = [SuperPlayerModel new];
//添加视频源其他信息
//步骤2：创建动态水印 model
DynamicWaterModel *model = [[DynamicWaterModel alloc] init];
//步骤3：设置动态水印的数据
model.dynamicWatermarkTip = @"shipinyun";
model.textFont = 30;
model.textColor = [UIColor colorWithRed:255.0/255.0 green:255.0/255.0 blue:255.0/
255.0 alpha:0.8];
playermodel.dynamicWaterModel = model;
//步骤4：调用方法展示动态水印
[self.playerView playWithModelNeedLicence:playermodel];
```

DynamicWaterModel 类参数说明：

参数名	类型	描述
dynamicWatermarkTip	NSString	水印文本信息
textFont	CGFloat	文字大小
textColor	UIColor	文字颜色

Demo 体验

更多完整功能可直接运行工程 Demo，或扫码下载移动端 Demo 腾讯云视立方 App体验。

运行工程 Demo

1. 在 Demo 目录，执行命令行 `pod update`，重新生成 `TXLiteAVDemo.xcworkspace` 文件。
2. 双击打开工程，修改证书选择真机运行。
3. 成功运行 Demo 后，进入 **播放器** > **播放器组件**，可体验播放器功能。

腾讯云视立方 App

在 **腾讯云视立方 App** > **播放器** 中可体验更多播放器组件功能。

Android

最近更新时间：2022-11-14 18:18:58

产品概述

腾讯云视立方 Android 播放器组件是腾讯云开源的一款播放器组件，集质量监控、视频加密、极速高清、清晰度切换、小窗播放等功能于一体，适用于所有点播、直播播放场景。封装了完整功能并提供上层 UI，可帮助您在短时间内，打造一个媲美市面上各种流行视频 App 的播放软件。

若播放器组件满足不了您的业务的个性化需求，且您具有一定的开发经验，可以集成 视立方播放器 SDK，自定义开发播放器界面和播放功能。

准备工作

1. 为了您体验到更完整全面的播放器功能，建议您开通 云点播 相关服务，未注册用户可注册账号 试用。若您不使用云点播服务，可略过此步骤，但集成后仅可使用播放器基础能力。
2. 下载 Android Studio，您可以进入 Android Studio 官网 下载安装，如已下载可略过该步骤。

通过本文您可以学会

1. 如何集成腾讯云视立方 Android 播放器组件
2. 如何创建和使用播放器

集成准备

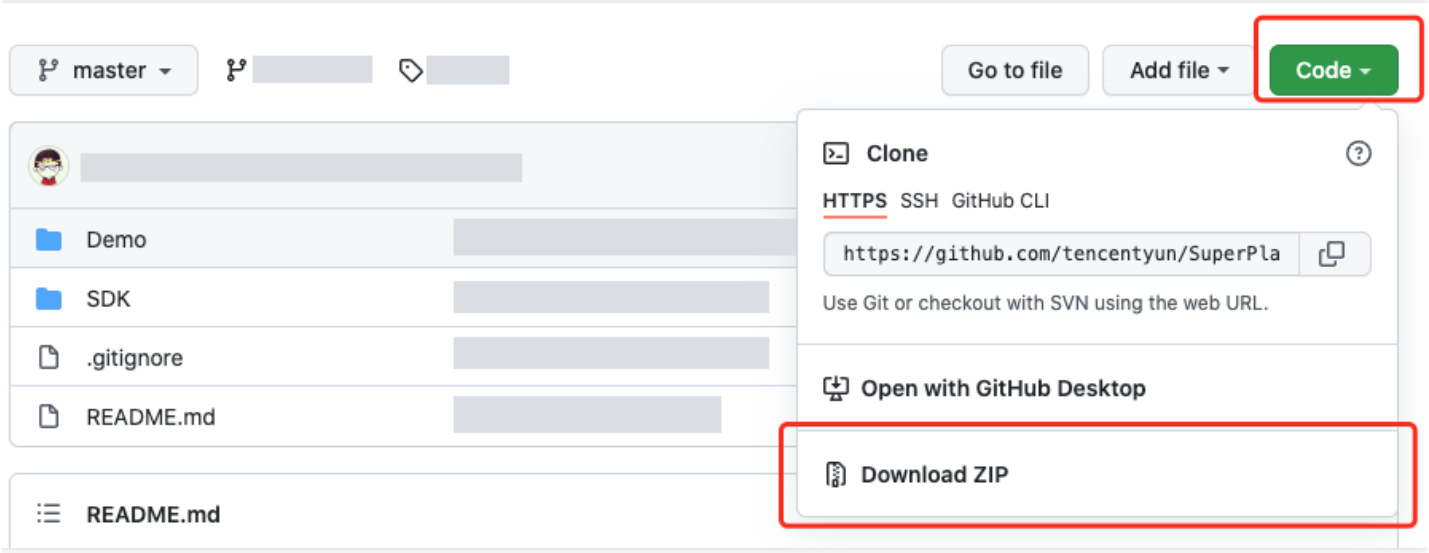
步骤1：项目下载

腾讯云视立方 Android 播放器组件的项目地址是 SuperPlayer_Android。

您可通过 **下载播放器组件 ZIP 包** 或 **Git 命令下载** 的方式下载腾讯云视立方 Android 播放器组件项目工程。

- 下载播放器组件 ZIP 包
- Git 命令下载

您可以直接下面播放器组件 ZIP包，单击页面的 **Code** > **Download ZIP** 下载。



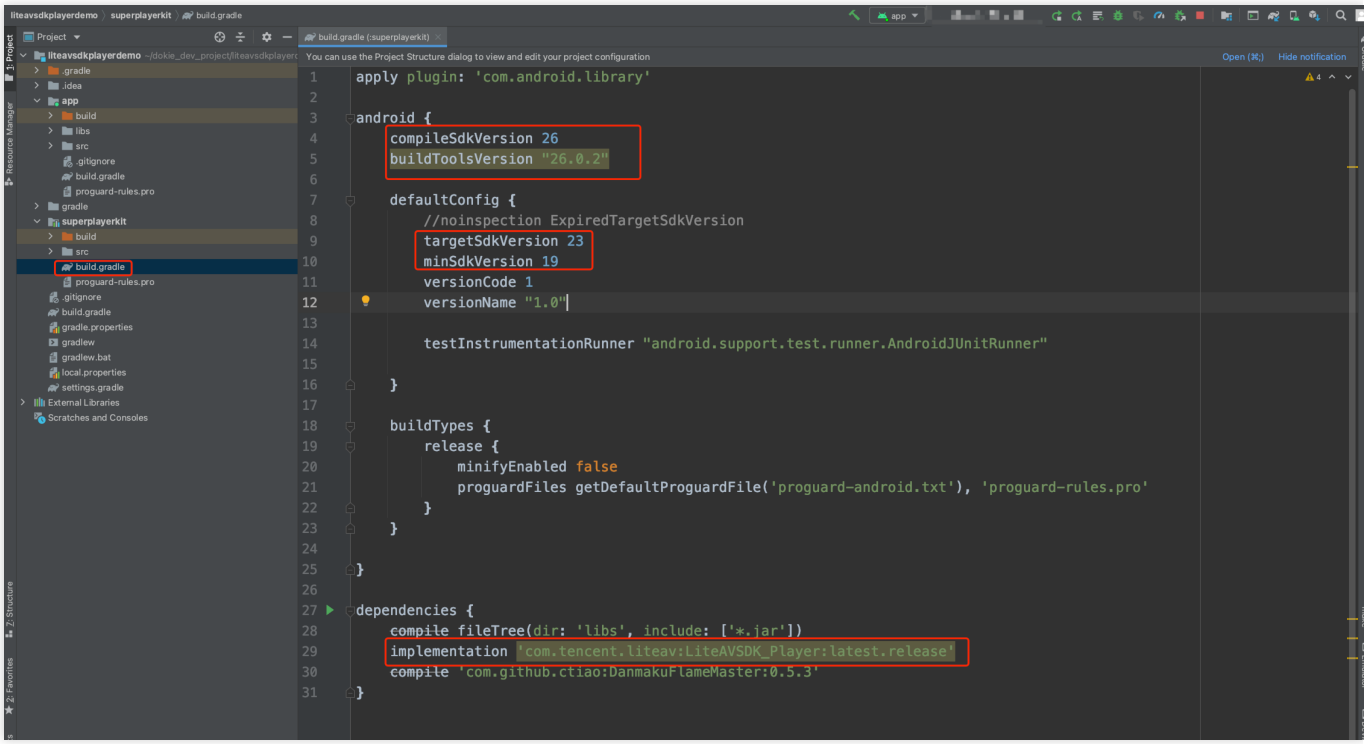
步骤2：集成指引

本步骤可指导您如何集成播放器，您可选择使用 Gradle 自动加载的方式，手动下载 aar 再将其导入到您当前的工程或导入 jar 和 so 库的方式集成项目。

- Gradle 自动加载（AAR）
- Gradle 手动下载（AAR）
- 集成 SDK（jar+so）

1. 下载 SDK + Demo 开发包，项目地址为 Android。

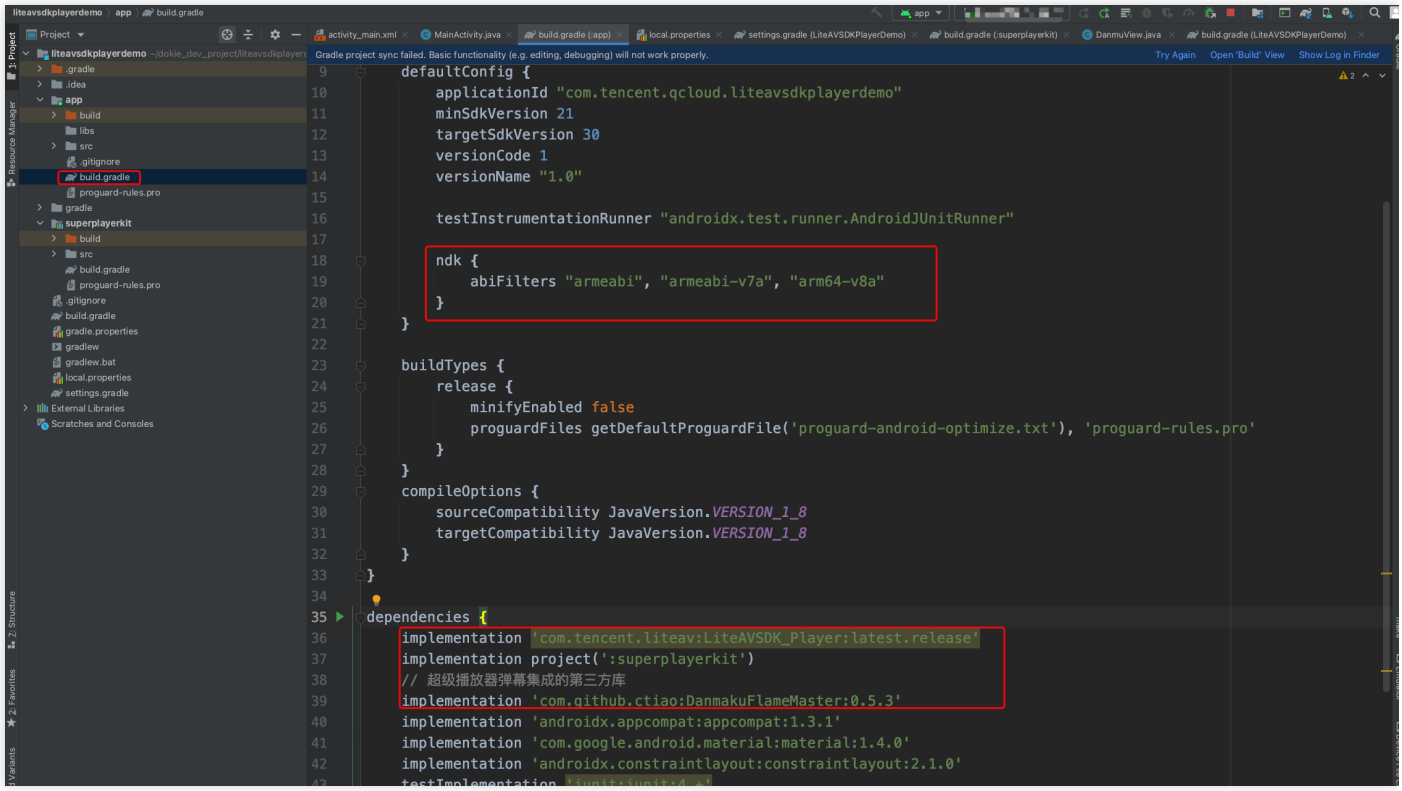
2. 把 `Demo/superplayerkit` 这个 module 复制到工程中，然后进行下面的配置：
   - 在工程目录下的setting.gradle 导入 `superplayerkit` 。
     ```
     include ':superplayerkit'
     ```
   - 打开 `superplayerkit` 工程的 build.gradle 文件修改 compileSdkVersion，buildToolsVersion，minSdkVersion，targetSdkVersion 和 rootProject.ext.liteavSdk 的常量值。




```
compileSdkVersion 26
buildToolsVersion "26.0.2"
defaultConfig {
targetSdkVersion 23
minSdkVersion 19
}
dependencies {
//如果要集成历史版本，可将 latest.release 修改为对应的版本，例如：8.5.290009
implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
}
```

请参见上面的步骤，把 `common` 模块导入到项目，并进行配置。

3. 通过在 gradle 配置 mavenCentral 库，自动下载更新 LiteAVSDK，打开 `app/build.gradle` ，进行下面的配置：



i. 在 dependencies 中添加 LiteAVSDK_Player 的依赖。

```
dependencies {
implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
implementation project(':superplayerkit')
// 播放器组件弹幕集成的第三方库
implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
}
```

如果您需要集成历史版本的 LiteAVSDK_Player SDK，可以在 MavenCentral 查看历史版本，然后通过下面的方式进行集成：

```
dependencies {
// 集成8.5.10033 版本LiteAVSDK_Player SDK
implementation 'com.tencent.liteav:LiteAVSDK_Player:8.5.10033'
}
```

ii. 在 `app/build.gradle` defaultConfig 中，指定 App 使用的 CPU 架构（目前 LiteAVSDK 支持 armeabi、armeabi-v7a 和 arm64-v8a，可根据项目需求配置）。

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

如果之前没有使用过9.4以及更早版本的 SDK 的 下载缓存功能（TXVodDownloadManager 中的相关接口），并且不需要在9.5及后续 SDK 版本播放9.4及之前缓存的下载文件，可以不需要该功能的 so 文件，达到减少安装包的体积，例如：在9.4及之前版本使用了 TXVodDownloadManager 类的 setDownloadPath 和 startDownloadUrl 函数下载了相应的缓存文件，并且应用内存储了 TXVodDownloadManager 回调的 getPlayPath 路径用于后续播放，这时候需要 libijkhlscache-master.so 播放该 getPlayPath 路径文件，否则不需要。可以在 app/build.gradle 中添加：

```
packagingOptions {
exclude "lib/armeabi/libijkhlscache-master.so"
exclude "lib/armeabi-v7a/libijkhlscache-master.so"
exclude "lib/arm64-v8a/libijkhlscache-master.so"
}
```

iii. 在工程目录的 build.gradle 添加 mavenCentral 库。

```
repositories {
mavenCentral()
}
```

4. 单击 Sync Now 按钮同步 SDK，如果您的网络连接 mavenCentral 没有问题，很快 SDK 就会自动下载集成到工程里。

您已经完成了腾讯云视立方 Android 播放器组件项目集成的步骤。

步骤3：配置 App 权限

在 AndroidManifest.xml 中配置 App 的权限，LiteAVSDK 需要以下权限：

```
<!--网络权限-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--点播播放器悬浮窗权限-->
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<!--存储-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

步骤4：设置混淆规则

在 proguard-rules.pro 文件，将 TRTC SDK 相关类加入不混淆名单：

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

您已经完成了腾讯云视立方 Android 播放器组件 app 权限配置的步骤。

步骤5：使用播放器功能

本步骤，用于指导用户创建和使用播放器，并使用播放器进行视频播放。

1. **创建播放器**

   播放器主类为 `SuperPlayerView` ，创建后即可播放视频， 支持集成 FileID 或者 URL 进行播放。在布局文件创建 SuperPlayerView：

```
<!-- 播放器组件-->
<com.tencent.liteav.demo.superplayer.SuperPlayerView
android:id="@+id/superVodPlayerView"
android:layout_width="match_parent"
android:layout_height="200dp" />
```

2. **配置 License 授权**

   若您已获得相关 License 授权，需在 腾讯云视立方控制台 获取 License URL 和 License Key。
   若您暂未获得 License 授权，需联系我们 获取相关授权。

   获取到 License 信息后，在调用 SDK 的相关接口前，通过下面的接口初始化 License，建议在 Application 类中进行如下设置：

```
public class MApplication extends Application {
@Override
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
TXLiveBase.getInstance().setLicence(this, licenceURL, licenceKey);
TXLiveBase.setListener(new TXLiveBaseListener() {
@Override
public void onLicenceLoaded(int result, String reason) {
Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
}
});
}
}
```

3. **播放视频**

本步骤用于指导用户播放视频。腾讯云视立方 Android 播放器组件可用于直播和点播两种播放场景，具体如下：

- 点播播放：播放器组件支持两种点播播放方式，可以 通过 FileID 播放 腾讯云点播媒体资源，也可以直接使用 URL 播放 地址进行播放。
- 直播播放：播放器组件可使用 URL 播放 的方式实现直播播放。通过传入 URL 地址，即可拉取直播音视频流进行直播播放。腾讯云直播URL生成方式可参见 自主拼装直播 URL。
  - 通过 URL 播放（直播、点播）
  - 通过 FileID 播放（点播）

  URL可以是点播文件播放地址，也可以是直播拉流地址，传入相应 URL 即可播放相应视频文件。

```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329073; // 配置 AppId
model.url = "http://your_video_url.mp4"; // 配置您的播放视频url
mSuperPlayerView.playWithModelNeedLicence(model);
```

4. **退出播放**

当不需要播放器时，调用 `resetPlayer` 清理播放器内部状态，释放内存。

```
mSuperPlayerView.resetPlayer();
```

至此，您已经完成了腾讯云视立方 Android 播放器组件创建、播放视频和退出播放的能力即成。

功能使用

本章将为您介绍几种常见的播放器功能使用方式，更为完整的功能使用方式可参见 Demo 体验，播放器组件支持的功能可参见 能力清单。

1、全屏播放

播放器组件支持全屏播放，在全屏播放场景内，同时支持锁屏、手势控制音量和亮度、弹幕、截屏、清晰度切换等功能设置。功能效果可在 **腾讯云视立方 App > 播放器 > 播放器组件** 中体验，单击界面右下角即可进入全屏播放界面。

在窗口播放模式下，可通过调用下述接口进入全屏播放模式：

```
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.FULLSCREEN);
```

全屏播放界面功能介绍

- 返回窗口

- 锁屏
- 弹幕
- 截屏
- 清晰度切换

单击**返回**，即可返回至窗口播放模式。

```
//单击后触发下面的接口
mControllerCallback.onBackPressed(SuperPlayerDef.PlayerMode.FULLSCREEN);
onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

2、悬浮窗播放

播放器组件支持悬浮窗小窗口播放，可在切换到其它应用时，不打断视频播放功能。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** 中体验，单击界面左上角**返回**，即可体验悬浮窗播放功能。

悬浮窗播放依赖于 AndroidManifest 中的以下权限：

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
```

```
// 切换悬浮窗触发的代码接口
mSuperPlayerView.switchPlayMode(SuperPlayerDef.PlayerMode.FLOAT);
//单击浮窗返回窗口触发的代码接口
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

3、视频封面

播放器组件支持用户自定义视频封面，用于在视频接收到首帧画面播放回调前展示。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** > **自定义封面演示** 视频中体验。

- 当播放器组件设置为自动播放模式 `PLAY_ACTION_AUTO_PLAY` 时，视频自动播放，此时将在视频首帧加载出来之前展示封面；
- 当播放器组件设置为手动播放模式 `PLAY_ACTION_MANUAL_PLAY` 时，需用户单击**播放**后视频才开始播放。在单击**播放**前将展示封面；在单击**播放**后到视频首帧加载出来前也将展示封面。

视频封面支持使用网络 URL 地址或本地 File 地址，使用方式可参见下述指引。若您通过 FileID 的方式播放视频，则可直接在云点播内配置视频封面。

```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = "您的appid";
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "您的fileId";
```

```
//播放模式，可设置自动播放模式：PLAY_ACTION_AUTO_PLAY，手动播放模式：PLAY_ACTION_MANUAL_PLAY
model.playAction = PLAY_ACTION_MANUAL_PLAY;
//设定封面的地址为网络url地址，如果coverPictureUrl不设定，那么就会自动使用云点播控制台设置的封面
model.coverPictureUrl = "http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500005830/cc1e282086022680110873365l8/MXUW1a5I9TsA.png"
mSuperPlayerView.playWithModelNeedLicence(model);
```

4、视频列表轮播

播放器组件支持视频列表轮播，即在给定一个视频列表后：

- 支持按顺序循环播放列表中的视频，播放过程中支持自动播放下一集也支持手动切换到下一个视频。
- 列表中最后一个视频播放完成后将自动开始播放列表中的第一个视频。

功能效果可在 **腾讯云视立方 App**> **播放器** > **播放器组件** > **视频列表轮播演示**视频中体验。

```
//步骤1:构建轮播的List<SuperPlayerModel>
ArrayList<SuperPlayerModel> list = new ArrayList<>();
SuperPlayerModel model = new VideoModel();
model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071568";
list.add(model);
model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071679";
list.add(model);
//步骤2：调用轮播接口
mSuperPlayerView.playWithModelListNeedLicence(list, true, 0);
```

```
public void playWithModelListNeedLicence(List<SuperPlayerModel> models, boolean isLoopPlayList, int index);
```

接口参数说明

参数名	类型	描述
models	List	轮播数据列表
isLoopPlayList	boolean	是否循环

参数名	类型	描述
index	int	开始播放的 SuperPlayerModel 索引

5、视频试看

播放器组件支持视频试看功能，可以适用于非 VIP 试看等场景，开发者可以传入不同的参数来控制视频试看时长、提示信息、试看结束界面等。功能效果可在 **腾讯云视立方 App**> **播放器** > **播放器组件** > **试看功能演示** 视频中体验。

```
方法一：
//步骤1：创建视频mode
SuperPlayerModel mode = new SuperPlayerModel();
//...添加视频源信息
//步骤2：创建试看信息 mode
VipWatchModel vipWatchModel = new VipWatchModel("可试看%ss，开通 VIP 观看完整视频",15
);
mode.vipWatchMode = vipWatchModel;
//步骤3：调用播放视频方法
mSuperPlayerView.playWithModelNeedLicence(mode);
方法二：
//步骤1：创建试看信息 mode
VipWatchModel vipWatchModel = new VipWatchModel("可试看%ss，开通 VIP 观看完整视频",15
);
//步骤2：调用设置试看功能方法
mSuperPlayerView.setVipWatchModel(vipWatchModel);
```

```
public VipWatchModel(String tipStr, long canWatchTime)
```

VipWatchModel 接口参数说明：

参数名	类型	描述
tipStr	String	试看提示信息
canWatchTime	Long	试看时长，单位为妙

6、动态水印

播放器组件支持在播放界面添加不规则跑动的文字水印，有效防盗录。全屏播放模式和窗口播放模式均可展示水印，开发者可修改水印文本、文字大小、颜色。功能效果可在 **腾讯云视立方 App** > **播放器** > **播放器组件** > **动态水印** 演示视频中体验。

```
方法一：
//步骤1：创建视频mode
SuperPlayerModel mode = new SuperPlayerModel();
//...添加视频源信息
//步骤2：创建水印信息mode
DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig("shipinyun", 30, C
olor.parseColor("#80FFFFFF"));
mode.dynamicWaterConfig = dynamicWaterConfig;
//步骤3：调用播放视频方法
mSuperPlayerView.playWithModelNeedLicence(mode);
方法二：
//步骤1：创建水印信息mode
DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig("shipinyun", 30, C
olor.parseColor("#80FFFFFF"));
//步骤2：调用设置动态水印功能方法
mSuperPlayerView.setDynamicWatermarkConfig(dynamicWaterConfig);
```

```
public DynamicWaterConfig(String dynamicWatermarkTip, int tipTextSize, int tipTex
tColor)
```

接口参数说明

参数名	类型	描述
dynamicWatermarkTip	String	水印文本信息
tipTextSize	int	文字大小
tipTextColor	int	文字颜色

Demo 体验

更多完整功能可直接运行工程 Demo，或扫码下载移动端 Demo 腾讯云视立方 App 体验。

运行工程 Demo

1. 在 Android Studio 的导航栏选择 **File** > **Open**，在弹框中选择 **Demo** 工程目录：`$SuperPlayer_Android/Demo`，待成功导入 Demo 工程后，单击 **Run app**，即可成功运行 Demo 。
2. 成功运行 Demo 后如下图，进入**播放器** > **播放器组件**，可体验播放器功能。

腾讯云视立方 App

在 **腾讯云视立方 App** > **播放器** 中可体验更多播放器组件功能。

美颜特效
SDK 功能说明

最近更新时间：2022-06-24 14:59:35

腾讯特效 SDK 共有 11 个套餐，11 个套餐分为 2 个系列：A 系列基础套餐 和 S 系列高级套餐。不同系列的不同套餐对应不同功能，各套餐支持的功能详情如下表。

A 系列基础套餐功能

A 系列基础套餐提供通用美型功能，适用于对脸部美颜调整要求较低的客户。

套餐功能		套餐编号					
		A1 - 01	A1 - 02	A1 - 03	A1 - 04	A1 - 05	A1 - 06
基础功能	**基础美颜** 美白、磨皮、红润	✓	✓	✓	✓	✓	✓
	画面调整 对比度、饱和度、清晰度	✓	✓	✓	✓	✓	✓
	基础美型 大眼、瘦脸（自然、女神、英俊）	✓	✓	✓	✓	✓	✓
	滤镜 （默认 10 款通用滤镜）	✓	✓	✓	✓	✓	✓
可拓展功能	**贴纸** （赠送 10 款可选 2D 通用贴纸）	-	✓	✓	✓	✓	✓
	通用美型 SDK （窄脸/下巴/发际线/瘦鼻）	-	-	✓	-	-	-
	手势识别 （赠送 1 款指定手势贴纸）	-	-	-	✓	-	-
	人像分割 / 虚拟背景 （赠送 3 款指定分割贴纸）	-	-	-	-	✓	-
	美妆 （赠送 3 款指定整妆）	-	-	-	-	-	✓
SDK 下载	iOS & Android	-					

S 系列高级套餐功能

S 系列高级套餐提供高级美型功能（包括特效贴纸和美妆），适用于对脸部美颜调整需求较高的客户。

套餐功能		套餐编号				
		S1 - 00	S1 - 01	S1 - 02	S1 - 03	S1 - 04
基础功能	**基础美颜** 美白、磨皮、红润	✓	✓	✓	✓	✓
	画面调整 对比度、饱和度、清晰度	✓	✓	✓	✓	✓
	高级美型 大眼、窄脸、瘦脸（自然、女神、英俊）、V 脸、下巴、短脸、脸型、发际线、亮眼、眼距、眼角、瘦鼻、鼻翼、瘦颧骨、鼻子位置、白牙、去皱、去法令纹、去眼袋、嘴型、嘴唇厚度、口红、腮红、立体	✓	✓	✓	✓	✓
	滤镜 （默认通用滤镜）	✓	✓	✓	✓	✓
	贴纸 （默认通用 2D 贴纸）	-	✓	✓	✓	✓
	高级贴纸 （默认通用 3D 贴纸）	-	✓	✓	✓	✓
	美妆 整妆	-	✓	✓	✓	✓
可拓展功能	**手势识别** （赠送 1 款指定手势贴纸）	-	-	✓	-	✓
	人像分割 / 虚拟背景 （赠送 3 款指定分割贴纸）	-	-	-	✓	✓
SDK 下载	iOS & Android	-				

SDK 集成指引
iOS

最近更新时间：2022-11-14 18:18:58

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。

```
[TXUGCBase setLicenceURL:LicenseURL key:LicenseKey];
[TELicenseCheck setTELicense:LicenseURLkey:LicenseKey completion:^(NSInteger auth
result, 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.currentPreviewResoluti
on];
NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirec
tory, NSUserDomainMask, YES) lastObject];
beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_conf
ig.json"];
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !i
sDir) {
```

```
NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfigPa
th 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:assets
Dict];
```

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:(i
d _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 withOrienta
tion:YtLightCameraRotation0]
```

步骤六：暂停/恢复 SDK

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

版权所有：腾讯云计算（北京）有限责任公司

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：2022-11-14 18:18:58

步骤一：解压 Demo 工程

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

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

替换为您购买的 套餐依赖包。

- 如果您的套餐包含动效和滤镜功能，那么需要在腾讯特效 SDK 下载页面 下载对应的资源，将动效和滤镜素材放置在 `xmagickit module` 下的如下目录：
  - 动效：`../assets/MotionRes`
  - 滤镜：`../assets/lut`

3. 将 Demo 工程中的 xmagickit 模块引入到实际项目工程中。

步骤二：打开 app 模块的 build.gradle

将 applicationId 修改成与申请的测试授权一致的包名。

步骤三：SDK 接口集成

可参考 Demo 工程的 UGCKitVideoRecord 类。

1. **授权：**

```
//鉴权注意事项及错误码详情，请参考 https://www.tencentcloud.com/document/product/1143/45385#.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 + " " + ms
g);
}
}
});
```

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()`），更多使用请参考视力中 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 工程的 UGCKitVideoRecord类。

短视频企业版迁移指引

最近更新时间：2022-08-02 15:04:47

目前，短视频企业版已经下线，其中美颜模块解耦升级成为腾讯特效SDK。腾讯特效 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腾讯特效（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. 下载集成了腾讯特效 TE 的 UGSV Demo 工程。本 Demo 基于腾讯特效 SDK S1-04 套餐构建。
2. 替换资源。由于本 Demo 工程使用的 SDK 套餐未必与您实际的套餐一致，因此要将本 Demo 中的相关 SDK 文件替换为您实际使用的套餐的 SDK 文件。具体操作如下：
   - 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 工程中的 xmagic 模块引入到实际项目工程中。

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腾讯特效 **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。

高级功能和特效
类抖音特效
iOS

最近更新时间：2022-05-24 16:22:28

滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持11种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置特效的方法为：

```
- (void) startEffect:(TXEffectType)type startTime:(float)startTime;
- (void) stopEffect:(TXEffectType)type endTime:(float)endTime;
//特效的类型（type 参数），在常量 TXEffectType 中有定义：
typedef NS_ENUM(NSInteger,TXEffectType)
{
TXEffectType_ROCK_LIGHT, //动感光波
TXEffectType_DARK_DRAEM, //暗黑幻境
TXEffectType_SOUL_OUT, //灵魂出窍
TXEffectType_SCREEN_SPLIT,//视频分裂
TXEffectType_WIN_SHADOW, //百叶窗
TXEffectType_GHOST_SHADOW,//鬼影
TXEffectType_PHANTOM, //幻影
TXEffectType_GHOST, //幽灵
TXEffectType_LIGHTNING, //闪电
TXEffectType_MIRROR, //镜像
TXEffectType_ILLUSION, //幻觉
};
- (void) deleteLastEffect;
- (void) deleteAllEffect;
```

调用 deleteLastEffect() 删除最后一次设置的滤镜特效。

调用 deleteAllEffect() 删除所有设置的滤镜特效。

Demo 示例：

在1 - 2s之间应用第一种滤镜特效；在3 - 4s之间应用第2种滤镜特效；删除3 - 4s设置的滤镜特效。

```
//在1-2s之间应用第一种滤镜特效
[_ugcEdit startEffect:TXEffectType_SOUL_OUT startTime:1.0];
```

```
[_ugcEdit stopEffect:TXEffectType_SOUL_OUT startTime:2.0)];
//在3-4s之间应用第2种滤镜特效
[_ugcEdit startEffect:TXEffectType_SPLIT_SCREEN startTime:3.0];
[_ugcEdit stopEffect:TXEffectType_SPLIT_SCREEN startTime:4.0];
//删除3-4s设置的滤镜特效
[_ugcEdit deleteLastEffect];
```

慢/快动作

您可以进行多段视频的慢速/快速播放，设置慢速/快速播放的方法为：

```
- (void) setSpeedList:(NSArray *)speedList;
//TXSpeed 的参数如下：
@interface TXSpeed: NSObject
@property (nonatomic, assign) CGFloat startTime; //加速播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //加速播放结束时间(s)
@property (nonatomic, assign) TXSpeedLevel speedLevel; //加速级别
@end
// 目前支持变速速度的几种级别，在常量 TXSpeedLevel 中有定义：
typedef NS_ENUM(NSInteger, TXSpeedLevel) {
SPEED_LEVEL_SLOWEST, //极慢速
SPEED_LEVEL_SLOW, //慢速
SPEED_LEVEL_NOMAL, //正常速
SPEED_LEVEL_FAST, //快速
SPEED_LEVEL_FASTEST, //极快速
};
```

Demo 示例：

```
// SDK 拥有支持多段变速的功能。 此 Demo 仅展示一段慢速播放
TXSpeed *speed =[[TXSpeed alloc] init];
speed.startTime = 1.0;
speed.endTime = 3.0;
speed.speedLevel = SPEED_LEVEL_SLOW;
[_ugcEdit setSpeedList:@[speed]];
```

倒放

您可以将视频画面倒序播放，设置倒放的方法：

```
- (void) setReverse:(BOOL)isReverse;
```

Demo 示例：

```
[_ugcEdit setReverse:YES];
```

重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。

设置重复片段方法：

```
- (void) setRepeatPlay:(NSArray *)repeatList;
//TXRepeat 的参数如下：
@interface TXRepeat: NSObject
@property (nonatomic, assign) CGFloat startTime; //重复播放起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //重复播放结束时间(s)
@property (nonatomic, assign) int repeatTimes; //重复播放次数
@end
```

Demo 示例：

```
TXRepeat *repeat = [[TXRepeat alloc] init];
repeat.startTime = 1.0;
repeat.endTime = 3.0;
repeat.repeatTimes = 3; //重复次数
[_ugcEdit setRepeatPlay:@[repeat]];
```

版权所有：腾讯云计算（北京）有限责任公司

Android

最近更新时间：2022-05-24 16:22:44

滤镜特效

您可以为视频添加多种滤镜特效，我们目前支持11种滤镜特效，每种滤镜您也可以设置视频作用的起始时间和结束时间。如果同一个时间点设置了多种滤镜特效，SDK 会应用最后一种滤镜特效作为当前的滤镜特效。

设置滤镜特效：

```
/**
* 设置滤镜特效开始时间
* @param type 滤镜特效类型
* @param startTime 滤镜特效开始时间（ms）
*/
public void startEffect(int type, long startTime);
/**
* 设置滤镜特效结束时间
* @param type 滤镜特效类型
* @param endTime 滤镜特效结束时间（ms）
*/
public void stopEffect(int type, long endTime);
```

参数说明：@param type：滤镜特效的类型，在常量 TXVideoEditConstants 中有定义：

```
public static final int TXEffectType_SOUL_OUT = 0; //滤镜效果1
public static final int TXEffectType_SPLIT_SCREEN = 1; //滤镜效果2
public static final int TXEffectType_DARK_DRAEM = 2; //滤镜效果3
public static final int TXEffectType_ROCK_LIGHT = 3; //滤镜效果4
public static final int TXEffectType_WIN_SHADDOW = 4; //滤镜效果5
public static final int TXEffectType_GHOST_SHADDOW = 5; //滤镜效果6
public static final int TXEffectType_PHANTOM_SHADDOW = 6; //滤镜效果7
public static final int TXEffectType_GHOST = 7; //滤镜效果8
public static final int TXEffectType_LIGHTNING = 8; //滤镜效果9
public static final int TXEffectType_MIRROR = 9; //滤镜效果10
public static final int TXEffectType_ILLUSION = 10; //滤镜效果11
```

删除最后一个设置的滤镜特效：

```
public void deleteLastEffect();
```

删除所有设置的滤镜特效：

```
public void deleteAllEffect();
```

完整示例如下：

在1 - 2s之间应用第一种滤镜特效；在3 - 4s之间应用第2种滤镜特效；删除3 - 4s设置的滤镜特效。

```
//在1-2s之间应用第一种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 1000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 2000);
//在3-4s之间应用第2种滤镜特效
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 3000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 4000);
//删除3-4s设置的滤镜特效
mTXVideoEditer.deleteLastEffect();
```

慢/快动作

您可以进行多段视频的慢速/快速播放，设置慢速/快速播放的方法为：

```
public void setSpeedList(List speedList)；
//TXSpeed 的参数如下：
public final static class TXSpeed {
public int speedLevel; // 变速级别
public long startTime; // 开始时间
public long endTime; // 结束时间
}
// 目前支持变速速度的几种级别，在常量 TXVideoEditConstants 中有定义：
SPEED_LEVEL_SLOWEST - 极慢
SPEED_LEVEL_SLOW - 慢
SPEED_LEVEL_NORMAL - 正常
SPEED_LEVEL_FAST - 快
SPEED_LEVEL_FASTEST - 极快
```

完整示例如下：

```
List<TXVideoEditConstants.TXSpeed> list = new ArrayList<>();
TXVideoEditConstants.TXSpeed speed1 = new TXVideoEditConstants.TXSpeed();
speed1.startTime = 0;
speed1.endTime = 1000;
speed1.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; // 慢速
list.add(speed1);
TXVideoEditConstants.TXSpeed speed2 = new TXVideoEditConstants.TXSpeed();
speed2.startTime = 1000;
```

版权所有：腾讯云计算（北京）有限责任公司

```
speed2.endTime = 2000;
speed2.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOWEST; // 极慢速
list.add(speed2);
TXVideoEditConstants.TXSpeed speed3 = new TXVideoEditConstants.TXSpeed();
speed3.startTime = 2000;
speed3.endTime = 3000;
speed3.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW; //慢速
list.add(speed3);
mTXVideoEditer.setSpeedList(list);
```

倒放

您可以将视频画面倒序播放。通过调用 **setReverse(true)** 开启倒序播放，调用 **setReverse(false)** 停止倒序播放。

> 注意：
> **setTXVideoReverseListener()** 老版本（SDK 4.5以前）首次监听需要手动调用，新版本不需要调用即可生效。

Demo 示例：

```
mTXVideoEditer.setTXVideoReverseListener(mTxVideoReverseListener);
mTXVideoEditer.setReverse(true);
```

重复视频片段

您可以设置重复播放一段视频画面，声音不会重复播放。目前 Android 只支持设置一段画面重复，重复三次。
如需取消之前设置的重复片段，调用 **setRepeatPlay(null)** 即可。

设置重复片段方法：

```
public void setRepeatPlay(List repeatList);
//TXRepeat 的参数如下：
public final static class TXRepeat {
public long startTime; //重复播放起始时间(ms)
public long endTime; //重复播放结束时间(ms)
public int repeatTimes; //重复播放次数
}
```

Demo 示例：

```
long currentPts = mVideoProgressController.getCurrentTimeMs();
List repeatList = new ArrayList<>();
TXVideoEditConstants.TXRepeat repeat = new TXVideoEditConstants.TXRepeat();
repeat.startTime = currentPts;
repeat.endTime = currentPts + DEAULT_DURATION_MS;
repeat.repeatTimes = 3; //目前只支持重复三次
repeatList.add(repeat); //目前只支持重复一段时间
mTXVideoEditer.setRepeatPlay(repeatList);
```

贴纸和字幕

iOS

最近更新时间：2022-05-24 16:23:29

静态贴纸

```
- (void) setPasterList:(NSArray *)pasterList;
// TXPaster 的参数如下：
@interface TXPaster: NSObject
@property (nonatomic, strong) UIImage* pasterImage; //贴纸图片
@property (nonatomic, assign) CGRect frame; //贴纸 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat startTime; //贴纸起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //贴纸结束时间(s)
@end
```

动态贴纸

```
- (void) setAnimatedPasterList:(NSArray *)animatedPasterList;
// TXAnimatedPaster 的参数如下：
@interface TXAnimatedPaster: NSObject
@property (nonatomic, strong) NSString* animatedPasterpath; //动图文件路径
@property (nonatomic, assign) CGRect frame; //动图的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat rotateAngle; //动图旋转角度 (0 ~ 360)
@property (nonatomic, assign) CGFloat startTime; //动图起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //动图结束时间(s)
@end
```

Demo 示例：

```
- (void)setVideoPasters:(NSArray*)videoPasterInfos
{
NSMutableArray* animatePasters = [NSMutableArray new];
NSMutableArray* staticPasters = [NSMutableArray new];
for (VideoPasterInfo* pasterInfo in videoPasterInfos) {
if (pasterInfo.pasterInfoType == PasterInfoType_Animate) {
TXAnimatedPaster* paster = [TXAnimatedPaster new];
```

```
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.rotateAngle = pasterInfo.pasterView.rotateAngle * 180 / M_PI;
paster.animatedPasterpath = pasterInfo.path;
[animatePasters addObject:paster];
}
else if (pasterInfo.pasterInfoType == PasterInfoType_static){
TXPaster *paster = [TXPaster new];
paster.startTime = pasterInfo.startTime;
paster.endTime = pasterInfo.endTime;
paster.frame = [pasterInfo.pasterView pasterFrameOnView:_videoPreview];
paster.pasterImage = pasterInfo.pasterView.staticImage;
[staticPasters addObject:paster];
}
}
[_ugcEditer setAnimatedPasterList:animatePasters];
[_ugcEditer setPasterList:staticPasters];
}
```

自定义动态贴纸

动态贴纸的本质是：将**一串图片**，按照**一定的顺序**以及**时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

如何自定义贴纸？

以工具包 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

版权所有：腾讯云计算（北京）有限责任公司

SDK 内部将获取到该动态贴纸对应的 config.json，并且按照 json 中定义的格式进行动态贴纸的展示。

> 说明：
> 该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸。

添加字幕

1. 气泡字幕

您可以为视频添加字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置字幕的方法为：

```
- (void) setSubtitleList:(NSArray *)subtitleList;
TXSubtitle 的参数如下：
@interface TXSubtitle: NSObject
@property (nonatomic, strong) UIImage* titleImage; //字幕图片 （这里需要客户把承载文字的控件转成 image 图片）
@property (nonatomic, assign) CGRect frame; //字幕的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
@property (nonatomic, assign) CGFloat startTime; //字幕起始时间(s)
@property (nonatomic, assign) CGFloat endTime; //字幕结束时间(s)
@end
```

- titleImage：表示字幕图片，如果上层使用的是 UILabel 之类的控件，请先把控件转成 UIImage，具体方法可以参照 demo 的示例代码。
- frame：表示字幕的 frame，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的 frame，具体可以参照 demo 的示例代码。
- startTime：字幕作用的起始时间。
- endTime：字幕作用的结束时间。

因为字幕这一块的 UI 逻辑比较复杂，我们已经在 demo 层有一整套的实现方法，推荐客户直接参考 demo 实现， 可以极大降低您的接入成本。

Demo 示例：

```
@interface VideoTextInfo : NSObject
@property (nonatomic, strong) VideoTextFiled* textField;
```

```
@property (nonatomic, assign) CGFloat startTime; //in seconds
@property (nonatomic, assign) CGFloat endTime;
@end
videoTextInfos = @[VideoTextInfo1, VideoTextInfo2 ...];
for (VideoTextInfo* textInfo in videoTextInfos) {
TXSubtitle* subtitle = [TXSubtitle new];
subtitle.titleImage = textInfo.textField.textImage; //UILabel (UIView) -> UIImag
e
subtitle.frame = [textInfo.textField textFrameOnView:_videoPreview]; //计算相对于渲
染 view 的坐标
subtitle.startTime = textInfo.startTime; //字幕起始时间
subtitle.endTime = textInfo.endTime; //字幕结束时间
[subtitles addObject:subtitle]; //添加字幕列表
}

[_ugcEditer setSubtitleList:subtitles]; //设置字幕列表
```

2. 如何自定义气泡字幕？

气泡字幕所需要的参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

> 说明：
> 以上单位均为 px。

封装格式

由于气泡字幕中携带参数较多，我们建议您可以在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 json 格式封装：

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
```

```
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

说明：
该封装格式用户可以自行决定，非 SDK 强制性要求。

字幕过长

字幕若输入过长时，如何进行排版才能够使字幕与气泡美观地合并？

我们在 Demo 中提供了一个自动排版的控件。若在当前字体大小下，字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。

您也可以修改相关控件源代码，来满足自身的业务要求。

Android

最近更新时间：2022-05-24 16:24:10

静态贴纸

设置静态贴纸的方法：

```
public void setPasterList(List pasterList);
// TXPaster 的参数如下：
public final static class TXPaster {
public Bitmap pasterImage; // 贴纸图片
public TXRect frame; // 贴纸的 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
public long startTime; // 贴纸起始时间(ms)
public long endTime; // 贴纸结束时间(ms)
}
```

动态贴纸

设置动态贴纸的方法：

```
public void setAnimatedPasterList(List animatedPasterList);
// TXAnimatedPaster 的参数如下：
public final static class TXAnimatedPaster {
public String animatedPasterPathFolder; // 动态贴纸图片地址
public TXRect frame; // 动态贴纸 frame（注意这里的 frame 坐标是相对于渲染 view 的坐标）
public long startTime; // 动态贴纸起始时间(ms)
public long endTime; // 动态贴纸结束时间(ms)
public float rotation;
}
```

Demo示例：

```
List animatedPasterList = new ArrayList<>();
List pasterList = new ArrayList<>();
for (int i = 0; i < mTCLayerViewGroup.getChildCount(); i++) {
PasterOperationView view = (PasterOperationView) mTCLayerViewGroup.getOperationVi
ew(i);
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX();
rect.y = view.getImageY();
```

版权所有：腾讯云计算（北京）有限责任公司

```
rect.width = view.getImageWidth();
TXCLog.i(TAG, "addPasterListVideo, adjustPasterRect, paster x y = " + rect.x +
"," + rect.y);
int childType = view.getChildType();
if (childType == PasterOperationView.TYPE_CHILD_VIEW_ANIMATED_PASTER) {
TXVideoEditConstants.TXAnimatedPaster txAnimatedPaster = new TXVideoEditConstant
s.TXAnimatedPaster();
txAnimatedPaster.animatedPasterPathFolder = mAnimatedPasterSDcardFolder + view.ge
tPasterName() + File.separator;
txAnimatedPaster.startTime = view.getStartTime();
txAnimatedPaster.endTime = view.getEndTime();
txAnimatedPaster.frame = rect;
txAnimatedPaster.rotation = view.getImageRotate();
animatedPasterList.add(txAnimatedPaster);
TXCLog.i(TAG, "addPasterListVideo, txAnimatedPaster startTimeMs, endTime is : " +
txAnimatedPaster.startTime + ", " + txAnimatedPaster.endTime);
} else if (childType == PasterOperationView.TYPE_CHILD_VIEW_PASTER) {
TXVideoEditConstants.TXPaster txPaster = new TXVideoEditConstants.TXPaster();
txPaster.pasterImage = view.getRotateBitmap();
txPaster.startTime = view.getStartTime();
txPaster.endTime = view.getEndTime();
txPaster.frame = rect;
pasterList.add(txPaster);
TXCLog.i(TAG, "addPasterListVideo, txPaster startTimeMs, endTime is : " + txPaste
r.startTime + ", " + txPaster.endTime);
}
}
mTXVideoEditer.setAnimatedPasterList(animatedPasterList); //设置动态贴纸
mTXVideoEditer.setPasterList(pasterList); //设置静态贴纸
```

自定义动态贴纸

动态贴纸的本质是：将**一串图片**，按照**一定的顺序**以及**时间间隔**，插入到视频画面中去，形成一个动态贴纸的效果。

封装格式

以 Demo 中一个动态贴纸为例：

```
{
"name":"glass", // 贴纸名称
"count":6, // 贴纸数量
"period":480, // 播放周期：播放一次动态贴纸的所需要的时间(ms)
"width":444, // 贴纸宽度
```

```
"height":256, // 贴纸高度
"keyframe":6, // 关键图片：能够代表该动态贴纸的一张图片
"frameArray": [ // 所有图片的集合
{"picture":"glass0"},
{"picture":"glass1"},
{"picture":"glass2"},
{"picture":"glass3"},
{"picture":"glass4"},
{"picture":"glass5"}
]
}
```

SDK 内部将获取到该动态贴纸对应的 config.json，并且按照 json 中定义的格式进行动态贴纸的展示。

> 说明：
> 该封装格式为 SDK 内部强制性要求，请严格按照该格式描述动态贴纸。

添加字幕

1. 气泡字幕

您可以为视频设置气泡字幕，我们支持对每一帧视频添加字幕，每个字幕您也可以设置视频作用的起始时间和结束时间。所有的字幕组成了一个字幕列表， 您可以把字幕列表传给 SDK 内部，SDK 会自动在合适的时间对视频和字幕做叠加。

设置气泡字幕的方法为：

```
public void setSubtitleList(List subtitleList);
//TXSubtitle 的参数如下：
public final static class TXSubtitle {
public Bitmap titleImage; // 字幕图片
public TXRect frame; // 字幕的 frame
public long startTime; // 字幕起始时间(ms)
public long endTime; // 字幕结束时间(ms)
}
public final static class TXRect {
public float x;
public float y;
public float width;
}
```

- titleImage：表示字幕图片，如果上层使用的是 TextView 之类的控件，请先把控件转成 Bitmap，具体方法可以参照 demo 的示例代码。
- frame：表示字幕的 frame，注意这个 frame 是相对于渲染 view（initWithPreview 时候传入的 view）的 frame，具体可以参照 demo 的示例代码。
- startTime：字幕作用的起始时间。
- endTime：字幕作用的结束时间。

因为字幕的 UI 逻辑比较复杂，我们已经在 demo 层有一整套的实现方法，推荐客户直接参考 demo 实现， 可以极大降低您的接入成本。

Demo 示例：

```
mSubtitleList.clear();
for (int i = 0; i < mWordInfoList.size(); i++) {
TCWordOperationView view = mOperationViewGroup.getOperationView(i);
TXVideoEditConstants.TXSubtitle subTitle = new TXVideoEditConstants.TXSubtitle();
subTitle.titleImage = view.getRotateBitmap(); //获取Bitmap
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = view.getImageX(); // 获取相对 parent view 的 x 坐标
rect.y = view.getImageY(); // 获取相对 parent view 的 y 坐标
rect.width = view.getImageWidth(); // 图片宽度
subTitle.frame = rect;
subTitle.startTime = mWordInfoList.get(i).getStartTime(); // 设置开始时间
subTitle.endTime = mWordInfoList.get(i).getEndTime(); // 设置结束时间
mSubtitleList.add(subTitle);
}
mTXVideoEditer.setSubtitleList(mSubtitleList); // 设置字幕列表
```

2.自定义气泡字幕？

气泡字幕所需要的参数

- 文字区域大小： top、left、right、bottom
- 默认的字体大小
- 宽、高

> 说明：
> 以上单位均为 px。

封装格式

由于气泡字幕中携带参数较多，我们建议您可以在 Demo 层封装相关的参数。如腾讯云 Demo 中使用的 json 格式封装。

```
{
"name":"boom", // 气泡字幕名称
"width": 376, // 宽度
"height": 335, // 高度
"textTop":121, // 文字区域上边距
"textLeft":66, // 文字区域左边距
"textRight":69, // 文字区域右边距
"textBottom":123, // 文字区域下边距
"textSize":40 // 字体大小
}
```

说明：
该封装格式用户可以自行决定，非 SDK 强制性要求。

字幕过长

字幕若输入过长时，如何进行排版才能够使字幕与气泡美观地合并？

我们在 Demo 中提供了一个自动排版的控件。若在当前字体大小下，字幕过长时，控件将自动缩小字号，直到能够恰好放下所有字幕文字为止。

您也可以修改相关控件源代码，来满足自身的业务要求。

视频合唱
iOS

最近更新时间：2022-05-24 16:25:50

本篇教程向大家介绍如何从零开始完成合唱的基础功能。

过程简介

1. 在界面上放两个 View， 一个用来播放，一个用来录制。
2. 再放一个按钮和进度条来开始录制和显示进度。
3. 录制与源视频相同的时长后停止。
4. 把录好的视频与源视频左右合成。
5. 预览合成好的视频。

界面搭建

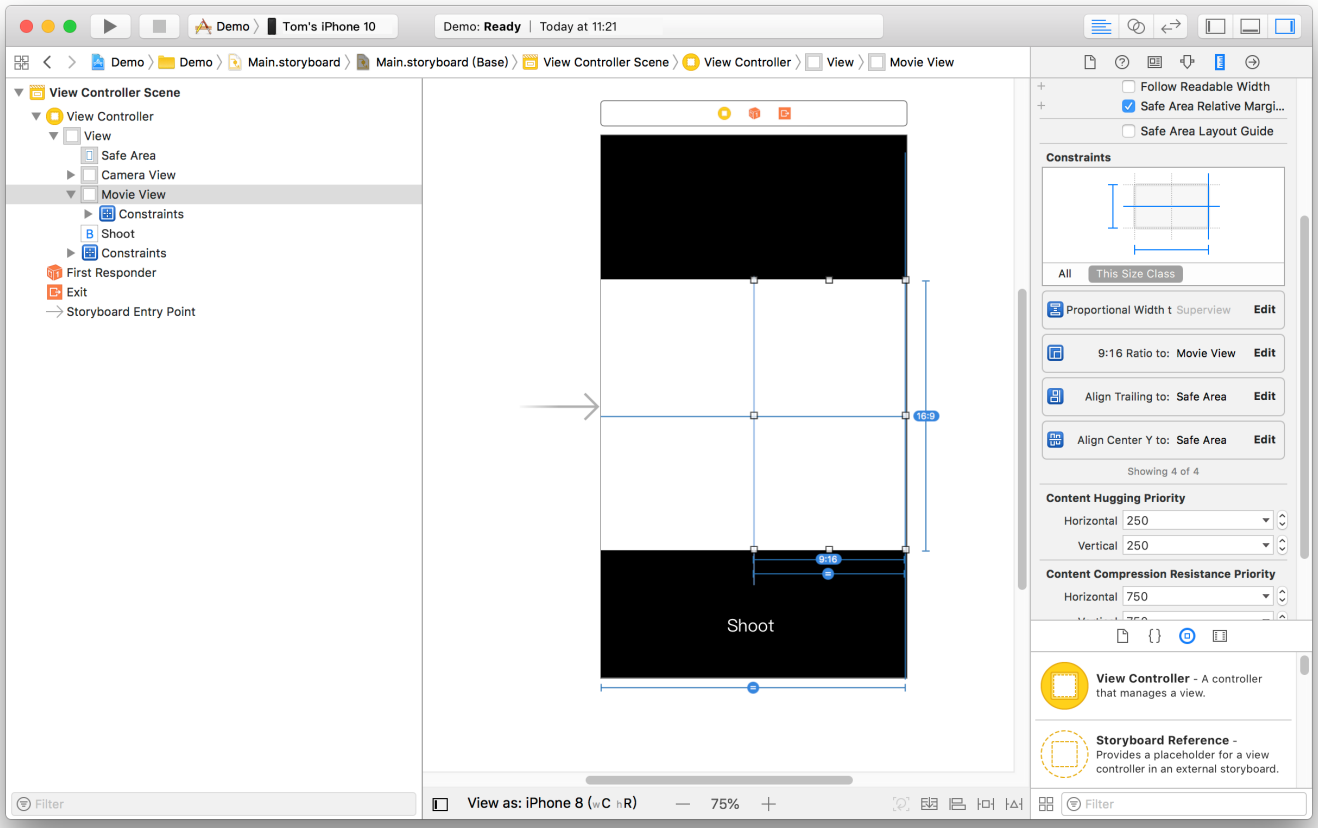
首先来开始工程的创建，打开 Xcode，File > New > Project，然后起好工程名创建工程，这里方便起见叫做 Demo，因为要录像，所以我们需要相机和麦克风的权限，在 Info 中配置一下增加以下两项：

```
Privacy - Microphone Usage Description
Privacy - Camera Usage Description
```

这两项的值可以随便填写，如"录制视频"。

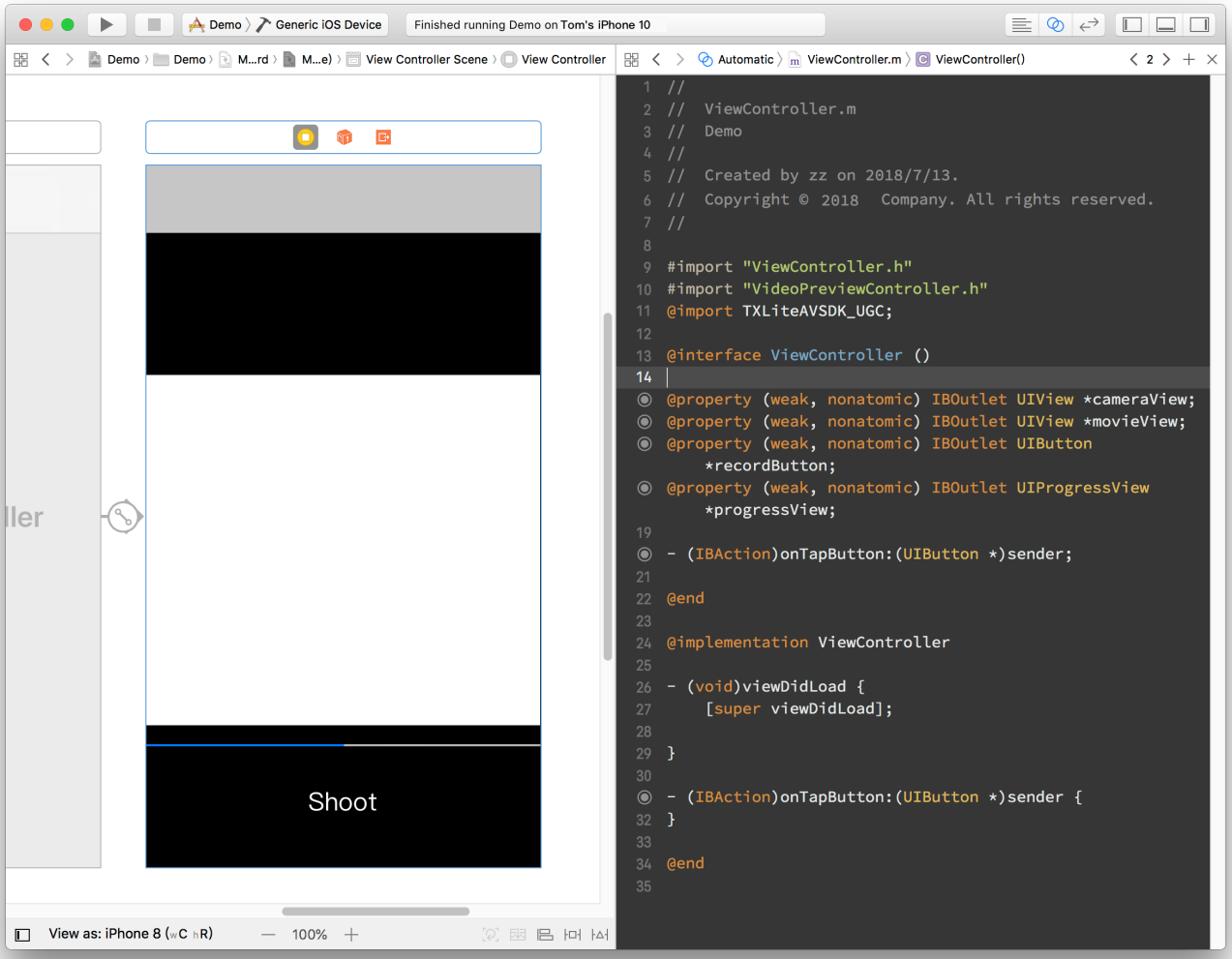
接下来我们配置一个简单的录制界面，打开 Main.storyboard，拖进去两个 UIView，配置宽度为 superview 的0.5倍，长宽比16:9。

版权所有：腾讯云计算（北京）有限责任公司



然后加上进度条，在 ViewController.m 中设置 IBOutlet 绑定界面，并设置好按钮的 IBAction。因为录制好后我们还要跳转到预览界面，还需要一个导航，单击黄色 VC 图标，在菜单栏依次进入 Editor > Embeded In，单击 Navigation

Controller 给 ViewController 套一层 Navigation Controller。完成基本 UI 的搭建。



代码部分

对于合唱功能主要使用三大块功能：播放、录制、以及录制后和原视频进行合成，这三个功能对应到 SDK 的类为：TXVideoEditer、TXUGCRecord、TXVideoJoiner。

在使用前要配置 SDK 的 Licence，打开 AppDelegate.m 在里面添加以下代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
[TXUGCBase setLicenceURL:@"<Licence的URL>" key:@"<Licence的Key>"];
return YES;
}
```

这里的 Licence 参数需要到 短视频控制台 去申请，提交申请后一般很快就会审批下来。然后页面上就会有相关的信息。

1. 首先是声明与初始化。

   打开 ViewContorller.m，引用 SDK 并声明上述三个类的实例。另外这里播放、录制和合成视频都是异步操作，需要监听他们的事件，所以要加上实现 TXVideoJoinerListener、TXUGCRecordListener、TXVideoPreviewListener 这三个协议的声明。加好后如下所示：

```
#import "ViewController.h"
@import TXLiteAVSDK_UGC;
@interface ViewController () <TXVideoJoinerListener, TXUGCRecordListener, TXVid
eoPreviewListener>
{
TXVideoEditer *_editor;
TXUGCRecord *_recorder;
TXVideoJoiner *_joiner;
TXVideoInfo *_videoInfo;

NSString *_recordPath;
NSString *_resultPath;
}
@property (weak, nonatomic) IBOutlet UIView *cameraView;
@property (weak, nonatomic) IBOutlet UIView *movieView;
@property (weak, nonatomic) IBOutlet UIButton *recordButton;
@property (weak, nonatomic) IBOutlet UIProgressView *progressView;
```

-

```
(IBAction)onTapButton:(UIButton *)sender;
@end
```

   准备好成员变量和接口实现声明后，我们在 viewDidLoad 中对上面的成员变量进行初始化。

```
- (void)viewDidLoad {
[super viewDidLoad];
// 这里找一段 mp4 视频放到了工程里，或者用手机录制的 mov 格式视频也可以
NSString *mp4Path = [[NSBundle mainBundle] pathForResource:@"demo" ofType:@"mp
```

```
4"];
_videoInfo = [TXVideoInfoReader getVideoInfo:mp4Path];
TXAudioSampleRate audioSampleRate = AUDIO_SAMPLERATE_48000;
if (_videoInfo.audioSampleRate == 8000) {
audioSampleRate = AUDIO_SAMPLERATE_8000;
}else if (_videoInfo.audioSampleRate == 16000){
audioSampleRate = AUDIO_SAMPLERATE_16000;
}else if (_videoInfo.audioSampleRate == 32000){
audioSampleRate = AUDIO_SAMPLERATE_32000;
}else if (_videoInfo.audioSampleRate == 44100){
audioSampleRate = AUDIO_SAMPLERATE_44100;
}else if (_videoInfo.audioSampleRate == 48000){
audioSampleRate = AUDIO_SAMPLERATE_48000;
}

// 设置录像的保存路径
_recordPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"record.m
p4"];
_resultPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"result.m
p4"];

// 播放器初始化
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = self.movieView;
param.renderMode = RENDER_MODE_FILL_EDGE;
_editor = [[TXVideoEditer alloc] initWithPreview:param];
[_editor setVideoPath:mp4Path];
_editor.previewDelegate = self;

// 录像参数初始化
_recorder = [TXUGCRecord shareInstance];
TXUGCCustomConfig *recordConfig = [[TXUGCCustomConfig alloc] init];
recordConfig.videoResolution = VIDEO_RESOLUTION_720_1280;
//这里保证录制视频的帧率和合唱视频的帧率一致，否则可能出现音画不同步的现象
//注意：这里获取的合唱视频的帧率是平均帧率，有可能为小数，做一下四舍五入操作
recordConfig.videoFPS = (int)(_videoInfo.fps + 0.5);
//这里保证录制视频的音频采样率和合唱视频的音频采样率一致，否则可能出现音画不同步的现象
recordConfig.audioSampleRate = audioSampleRate;
recordConfig.videoBitratePIN = 9600;
recordConfig.maxDuration = _videoInfo.duration;
_recorder.recordDelegate = self;

// 启动相机预览
[_recorder startCameraCustom:recordConfig preview:self.cameraView];

// 视频拼接
_joiner = [[TXVideoJoiner alloc] initWithPreview:nil];
```

```
_joiner.joinerDelegate = self;
[_joiner setVideoPathList:@[_recordPath, mp4Path]];
}
```

2. 接下来是录制部分，只要响应用户单击按钮调用 SDK 方法就可以了，为了方便起见，这里复用了这个按钮来显示当前状态。另外加上在进度条上显示进度的逻辑。

```
- (IBAction)onTapButton:(UIButton *)sender {
[_editor startPlayFromTime:0 toTime:_videoInfo.duration];
if ([_recorder startRecord:_recordPath coverPath:[_recordPath stringByAppending
String:@".png"]] != 0) {
NSLog(@"相机启动失败");
}
[sender setTitle:@"录像中" forState:UIControlStateNormal];
sender.enabled = NO;
}
#pragma mark TXVideoPreviewListener
```

```
-(void) onPreviewProgress:(CGFloat)time
{
self.progressView.progress = time / _videoInfo.duration;
}
```

3. 录制好后开始完成拼接部分, 这里需要指定两个视频在结果中的位置，这里设置一左一右。

```
-(void)onRecordComplete:(TXUGCRecordResult*)result;
{
NSLog(@"录制完成，开始合成");
[self.recordButton setTitle:@"合成中..." forState:UIControlStateNormal];

//获取录制视频的宽高
TXVideoInfo *videoInfo = [TXVideoInfoReader getVideoInfo:_recordPath];
CGFloat width = videoInfo.width;
CGFloat height = videoInfo.height;

//录制视频和原视频左右排列
CGRect recordScreen = CGRectMake(0, 0, width, height);
CGRect playScreen = CGRectMake(width, 0, width, height);
[_joiner setSplitScreenList:@[[NSValue valueWithCGRect:recordScreen],[NSValue v
```

```
alueWithCGRect:playScreen]] canvasWidth:width * 2 canvasHeight:height];
[_joiner splitJoinVideo:VIDEO_COMPRESSED_720P videoOutputPath:_resultPath];
}
```

4. 实现合成进度的委托方法, 在进度条中显示进度。

```
-(void) onJoinProgress:(float)progress
{
NSLog(@"视频合成中%d%%",(int)(progress * 100));
self.progressView.progress = progress;
}
```

5. 实现合成完成的委托方法，并切换到预览界面。

```
#pragma mark TXVideoJoinerListener
```

```
-(void) onJoinComplete:(TXJoinerResult *)result
{
NSLog(@"视频合成完毕");
VideoPreviewController *controller = [[VideoPreviewController alloc]
initWithVideoPath:_resultPath];
[self.navigationController pushViewController:controller animated:YES];
}
```

至此就制作完成了，上面提到了一个视频预览的 `VideoPreviewController` 代码如下：

- `VideoPreviewController.h`

```
#import <UIKit/UIKit.h>
@interface VideoPreviewController : UIViewController
```

-

```
(instancetype)initWithVideoPath:(NSString *)path;
@end
```

- `VideoPreviewController.m:`

```
@import TXLiteAVSDK_UGC;
@interface VideoPreviewController () <TXVideoPreviewListener>
{
TXVideoEditer *_editor;
}
@property (strong, nonatomic) NSString *videoPath;
@end
@implementation VideoPreviewController
```

- (instancetype)initWithVideoPath:(NSString *)path {
  if (self = [super initWithNibName:nil bundle:nil]) {

```
<span class="hljs-keyword">self</span>.videoPath = path;
```

  }
  return self;
  }

- (void)viewDidLoad {
  [super viewDidLoad];
  TXPreviewParam *param = [[TXPreviewParam alloc] init];
  param.videoView = self.view;
  param.renderMode = RENDER_MODE_FILL_EDGE;
  _editor = [[TXVideoEditer alloc] initWithPreview:param];
  _editor.previewDelegate = self;
  [_editor setVideoPath:self.videoPath];
  [_editor startPlayFromTime:0 toTime:[TXVideoInfoReader
  getVideoInfo:self.videoPath].duration];
  }

-

```
(void) onPreviewFinished
{
[_editor startPlayFromTime:0 toTime:[TXVideoInfoReader
getVideoInfo:self.videoPath].duration];
}
@end
```

至此就完成了全部合唱的基础功能，功能更加丰富的示例可以参考 小视频源码。

Android

最近更新时间：2022-05-24 16:26:13

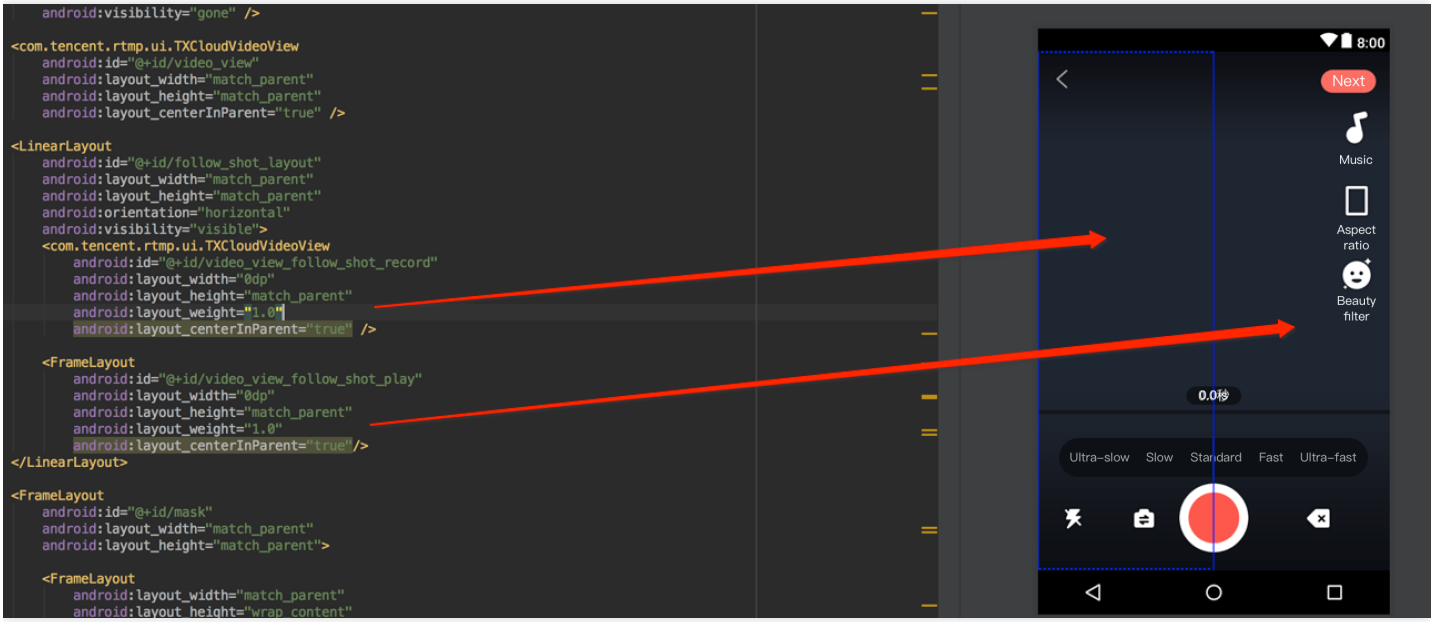
本篇教程向大家介绍如何完成合唱的基础功能。

过程简介

1. 在界面上放两个 View, 一个用来播放，一个用来录制。
2. 再放一个按钮和进度条来开始录制和显示进度。
3. 录制与源视频相同的时长后停止。
4. 把录好的视频与源视频左右合成。
5. 预览合成好的视频。

界面搭建

在录制界面 TCVideoRecordActivity 的 activity_video_record.xml 中创建两个 view，左半边是录制界面，右半边是播放界面。



代码部分

对于合唱功能主要使用三大块功能：播放、录制、以及录制后和原视频进行合成，这三个功能对应到 SDK 的类为：TXVideoEditer、TXUGCRecord、TXVideoJoiner，其中播放也可以换成 TXVodPlayer 去播放。

1. 从小视频主页的视频列表中，选择一个视频进入播放界面 TCVodPlayerActivity，单击右下角的“合拍”按钮。
   首先会下载该视频到本地 sdcard 中，并获取该视频的音频采样率以及 fps 等信息后进入录制界面。

2. 进入录制界面 TCVideoRecordActivity 进行合唱。需要注意以下几点：

- 录制进度条以跟拍视频的进度为最大长度。
- 保证录制视频的帧率和合唱视频的帧率一致，否则可能出现音画不同步的现象。
- 保证录制视频的音频采样率和合唱视频的音频采样率一致，否则可能出现音画不同步的现象。
- 录制设置渲染模式为自适应模式，在9:16的宽高比时能等比例缩放。
- Android 的录制需要设置静音，否则会造成与跟拍视频的“二重唱”。

```
// 录制的界面
mVideoView = mVideoViewFollowShotRecord;
// 播放的视频
mFollowShotVideoPath = intent.getStringExtra(TCConstants.VIDEO_EDITER_PATH);
mFollowShotVideoDuration = (int)(intent.getFloatExtra(TCConstants.VIDEO_RECORD_
DURATION, 0) * 1000);
initPlayer();
// 录制进度条以跟拍视频的进度为最大长度，fps 以跟拍视频的 fps 为准
mMaxDuration = (int)mFollowShotVideoDuration;
mFollowShotVideoFps = intent.getIntExtra(TCConstants.RECORD_CONFIG_FPS, 20);
mFollowShotAudioSampleRateType = intent.getIntExtra(TCConstants.VIDEO_RECORD_AU
DIO_SAMPLE_RATE_TYPE, TXRecordCommon.AUDIO_SAMPLERATE_48000);
// 初始化合拍的接口
mTXVideoJoiner = new TXVideoJoiner(this);
mTXVideoJoiner.setVideoJoinerListener(this);
```

```
// 播放器初始化，这里使用 TXVideoEditer，也可以使用 TXVodPlayer
mTXVideoEditer = new TXVideoEditer(this);
mTXVideoEditer.setVideoPath(mFollowShotVideoPath);
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreviewPar
am();
param.videoView = mVideoViewPlay;
```

```
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
mTXVideoEditer.initWithPreview(param);
```

```
customConfig.videoFps = mFollowShotVideoFps;
customConfig.audioSampleRate = mFollowShotAudioSampleRateType; // 录制的视频的音频采样率必须与跟拍的音频采样率相同
customConfig.needEdit = false;
mTXCameraRecord.setVideoRenderMode(TXRecordCommon.VIDEO_RENDER_MODE_ADJUST_RESOLUTION); // 设置渲染模式为自适应模式
mTXCameraRecord.setMute(true); // 跟拍不从喇叭录制声音，因为跟拍的视频声音也会从喇叭发出来被麦克风录制进去，造成跟原视频声音的"二重唱"。
```

3. 接下来就可以开始录制了，在录制到最大长度后，会回调 onRecordComplete，继续完成拼接部分，这里需要指定两个视频在结果中的位置。

```
private void prepareToJoiner(){
List<String> videoSourceList = new ArrayList<>();
videoSourceList.add(mRecordVideoPath);
videoSourceList.add(mFollowShotVideoPath);
mTXVideoJoiner.setVideoPathList(videoSourceList);
mFollowShotVideoOutputPath = getCustomVideoOutputPath("Follow_Shot_");
// 以左边录制的视频宽高为基准，右边视频等比例缩放
int followVideoWidth;
int followVideoHeight;
if ((float) followVideoInfo.width / followVideoInfo.height >= (float)recordVideoInfo.width / recordVideoInfo.height) {
followVideoWidth = recordVideoInfo.width;
followVideoHeight = (int) ((float)recordVideoInfo.width * followVideoInfo.height / followVideoInfo.width);
} else {
followVideoWidth = (int) ((float)recordVideoInfo.height * followVideoInfo.width / followVideoInfo.height);
followVideoHeight = recordVideoInfo.height;
}
TXVideoEditConstants.TXAbsoluteRect rect1 = new TXVideoEditConstants.TXAbsoluteRect();
rect1.x = 0; //第一个视频的左上角位置
rect1.y = 0;
rect1.width = recordVideoInfo.width; //第一个视频的宽高
rect1.height = recordVideoInfo.height;
TXVideoEditConstants.TXAbsoluteRect rect2 = new TXVideoEditConstants.TXAbsoluteRect();
rect2.x = rect1.x + rect1.width; //第2个视频的左上角位置
rect2.y = (recordVideoInfo.height - followVideoHeight) / 2;
rect2.width = followVideoWidth; //第2个视频的宽高
rect2.height = followVideoHeight;
```

```
List<TXVideoEditConstants.TXAbsoluteRect> list = new ArrayList<>();
list.add(rect1);
list.add(rect2);
mTXVideoJoiner.setSplitScreenList(list, recordVideoInfo.width + followVideoWidt
h, recordVideoInfo.height); //第2、3个 param：两个视频合成画布的宽高
mTXVideoJoiner.splitJoinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mFoll
owShotVideoOutputPath);
}
```

4. 监听合成的回调，在 onJoinComplete 后跳转到预览界面播放。

```
@Override
public void onJoinComplete(TXVideoEditConstants.TXJoinerResult result) {
mCompleteProgressDialog.dismiss();
if(result.retCode == TXVideoEditConstants.JOIN_RESULT_OK){
runOnUiThread(new Runnable() {
@Override
public void run() {
isReadyJoin = true;
startEditerPreview(mFollowShotVideoOutputPath);
if(mTXVideoEditer != null){
mTXVideoEditer.release();
mTXVideoEditer = null;
}
}
});
}else{
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(TCVideoRecordActivity.this, "合成失败", Toast.LENGTH_SHORT).show
();
}
});
}
}
```

至此就完成了全部合唱的基础功能，完整代码可以参考 小视频源码。

图片转场特效

iOS

最近更新时间：2022-05-24 16:29:21

SDK 在4.7版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画、BGM、贴纸等效果。
接口函数如下：

```
/*
*pitureList：转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考 demo 用法），否则内存占用可能过大，导致编辑过程异常）
*fps：转场图片生成视频后的 fps（15 - 30）
* 返回值：
* 0：设置成功；
* -1：设置失败，请检查图片列表是否存在，图片数量是否大于等于3张，fps 是否正常；
*/
- (int)setPictureList:(NSArray<UIImage *> *)pitureList fps:(int)fps;
/*
*transitionType：转场类型，详情见 TXTransitionType
* 返回值：
* duration：转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
- (void)setPictureTransition:(TXTransitionType)transitionType duration:(void(^)(CGFloat))duration;
```

- setPictureList 接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition 接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过 duration 获取转场的时长。
- 需要注意接口调用顺序，先调用 setPictureList，再调用 setPictureTransition。
- 图片编辑暂不支持的功能：重复、倒放、快速/慢速、片尾水印。其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

版权所有：腾讯云计算（北京）有限责任公司

Android

最近更新时间：2022-05-24 16:29:36

SDK 在4.9版本后增加了图片编辑功能，用户可以选择自己喜欢的图片，添加转场动画、BGM、贴纸等效果。
接口函数如下：

```
/*
* bitmapList：转场图片列表,至少设置三张图片（tips：图片最好压缩到720P以下（参考 demo 用法），否则内存占用可能过大，导致编辑过程异常）
* fps：转场图片生成视频后的 fps（15 - 30）
* 返回值：
* 0：设置成功；
* -1：设置失败，请检查图片列表是否存在
*/
public int setPictureList(List<Bitmap> bitmapList, int fps);
/*
* type：转场类型，详情见 TXVideoEditConstants
* 返回值：
* duration：转场视频时长（tips：同一个图片列表，每种转场动画的持续时间可能不一样，这里可以获取转场图片的持续时长）；
*/
public long setPictureTransition(int type)
```

- 其中，setPictureList 接口用于设置图片列表，最少设置三张，如果设置的图片过多，要注意图片的大小，防止内存占用过多而导致编辑异常。
- setPictureTransition 接口用于设置转场的效果，目前提供了6种转场效果供用户设置，每种转场效果持续的时长可能不一样，这里可以通过返回值获取转场的时长。
- 需要注意接口调用顺序，先调用 setPictureList，再调用 setPictureTransition。
- 图片编辑暂不支持的功能：重复、倒放、快速/慢速。其他视频相关的编辑功能，图片编辑均支持，调用方法和视频编辑完全一样。

版权所有：腾讯云计算（北京）有限责任公司

定制视频数据

iOS

最近更新时间：2022-06-24 14:56:48

录制预处理回调

```
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
* @param texture 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理
* 说明：SDK 回调出来的纹理类型是 GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GL_TEXTU
RE_2D; 该回调在 SDK 美颜之后. 纹理格式为 GL_RGBA
*/
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFlo
at)height;

/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
- (void)onTextureDestoryed;
```

编辑预处理回调

```
/**
在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
@param texture 纹理 ID
@param width 纹理的宽度
@param height 纹理的高度
@param timestamp 纹理 timestamp 单位 ms
@return 返回给 SDK 的纹理
说明：SDK 回调出来的纹理类型是 GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 GL_TEXTURE
_2D; 该回调在 SDK 美颜之后. 纹理格式为 GL_RGBA
timestamp 为当前视频帧的 pts ，单位是 ms ，客户可以根据自己的需求自定义滤镜特效
*/
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFlo
at)height timestamp:(UInt64)timestamp;
```

```
/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
- (void)onTextureDestoryed;
```

Android

最近更新时间：2022-05-24 16:29:51

录制预处理回调

```
public interface VideoCustomProcessListener {
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
* @param textureId 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可
* 说明：SDK 回调出来的纹理类型是 GLES20.GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 G
LES20.GL_TEXTURE_2D
*/
int onTextureCustomProcess(int textureId, int width, int height);
/**
* 增值版回调人脸坐标
* @param points 归一化人脸坐标，每两个值表示某点 P 的 X、Y 值。值域[0.f,1.f]
*/
void onDetectFacePoints(float[] points);
/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
void onTextureDestroyed();
}
```

编辑预处理回调

```
public interface TXVideoCustomProcessListener {
/**
* 在 OpenGL 线程中回调，在这里可以进行采集图像的二次处理
*
* @param textureId 纹理 ID
* @param width 纹理的宽度
* @param height 纹理的高度
* @return 返回给 SDK 的纹理 ID，如果不做任何处理，返回传入的纹理 ID 即可
* <p>
* 说明：SDK 回调出来的纹理类型是 GLES20.GL_TEXTURE_2D，接口返回给 SDK 的纹理类型也必须是 G
LES20.GL_TEXTURE_2D
```

```
*/
int onTextureCustomProcess(int textureId, int width, int height, long timestamp);
/**
* 在 OpenGL 线程中回调，可以在这里释放创建的 OpenGL 资源
*/
void onTextureDestroyed();
}
```

视频鉴黄

最近更新时间：2020-09-01 14:52:10

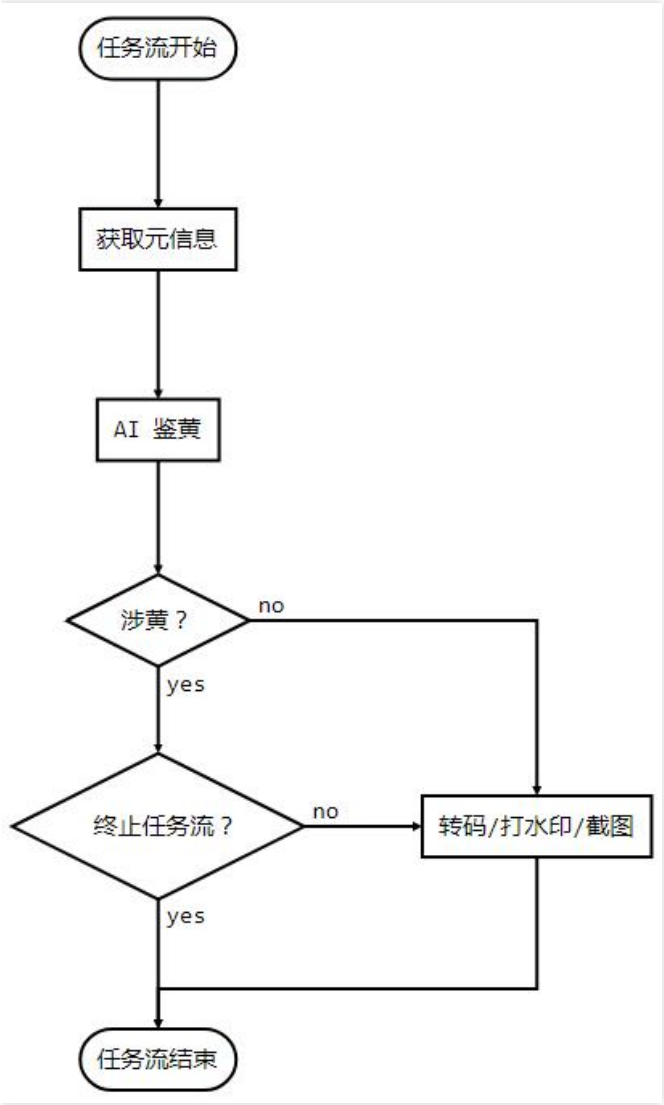
在个人直播录制、UGC 短视频等场景中，视频内容是不可预知的。为了避免一些违规内容出现在点播平台上，开发者会要求先对上传的视频做审核，确认合规后再进行转码和分发。腾讯云短视频解决方案支持对视频进行 AI 鉴黄，自动识别视频是否涉及色情内容。

使用 AI 鉴黄

AI 鉴黄功能需要集成在视频处理任务流中使用，它依赖于云点播后台的 视频 AI - 内容审核，审核结果通过事件通知的方式来通知 App 后台服务。云点播内置了一个任务流 QCVB_ProcessUGCFile 用于 UGC 短视频鉴黄场景，当用户使用该任务流并指定进行 AI 鉴黄时，鉴黄操作将优先执行，并根据鉴黄结果来决定是否进行后续处理（转码、打

版权所有：腾讯云计算（北京）有限责任公司

水印和截图等）。流程图如下：



AI 模版介绍

模版 ID	涉黄处理	采样频率
10	终止任务流（转码、打水印、截图等均不会执行）	时长小于500秒的视频，每1秒采样1次；时长大于或等于500秒的视频，每1%时长采样1次
20	继续执行任务流	无

AI 鉴定接入示例

步骤1：在生成上传签名时提交 AI 鉴黄任务

```
/**
* 响应签名请求并上传带有 AI 鉴黄任务
*/
function getUploadSignature(req, res) {
res.json({
code: 0,
message: 'ok',
data: {
//上传时指明模版参数与任务流
signature: gVodHelper.createFileUploadSignature({ procedure: 'QCVB_SimpleProcessF
ile({1,1,1,1})' })
}
});
}
```

步骤2：在获事件通知时获取 AI 鉴黄结果

```
/**
* 获取事件的 AI 鉴黄结果
*/
function getAiReviewResult(event){
let data = event.eventContent.data;
if(data.aiReview){
return data.aiReview;
}
return {};
}
```

UGCKit 主题定制
iOS

最近更新时间：2022-11-14 18:25:34

UGCKit 可以让您自由修改预置的文字、颜色和图标。

文字

UGCKit 默认提供中英文两套语言包，其中所有的本地化字符串均在 `UGCKit/UGCKitResources/Localizable.strings` 中，以默认的标准的本地化字符串格式存储，您可以通过替换其中的文本来修改默认的字符串，或者增加新的语言。

以动效名称为例，其中文内容位于 `UGCKit/UGCKitResources/zh-Hans.lproj/Localizable.strings`，内容示例如下:

```
"UGCKit.Edit.VideoEffect.DynamicLightWave" = "动感光波";
"UGCKit.Edit.VideoEffect.DarkFantasy" = "暗黑幻境";
"UGCKit.Edit.VideoEffect.SoulOut" = "灵魂出窍";
"UGCKit.Edit.VideoEffect.ScreenSplit" = "画面分裂";
"UGCKit.Edit.VideoEffect.Shutter" = "百叶窗";
"UGCKit.Edit.VideoEffect.GhostShadow" = "鬼影";
"UGCKit.Edit.VideoEffect.Phantom" = "幻影";
"UGCKit.Edit.VideoEffect.Ghost" = "幽灵";
"UGCKit.Edit.VideoEffect.Lightning" = "闪电";
"UGCKit.Edit.VideoEffect.Mirror" = "镜像";
"UGCKit.Edit.VideoEffect.Illusion" = "幻觉";
```

如果需要修改 "幻觉"效果为“幻像”，只需修改最后一行为

```
"UGCKit.Edit.VideoEffect.Illusion" = "幻像";
```

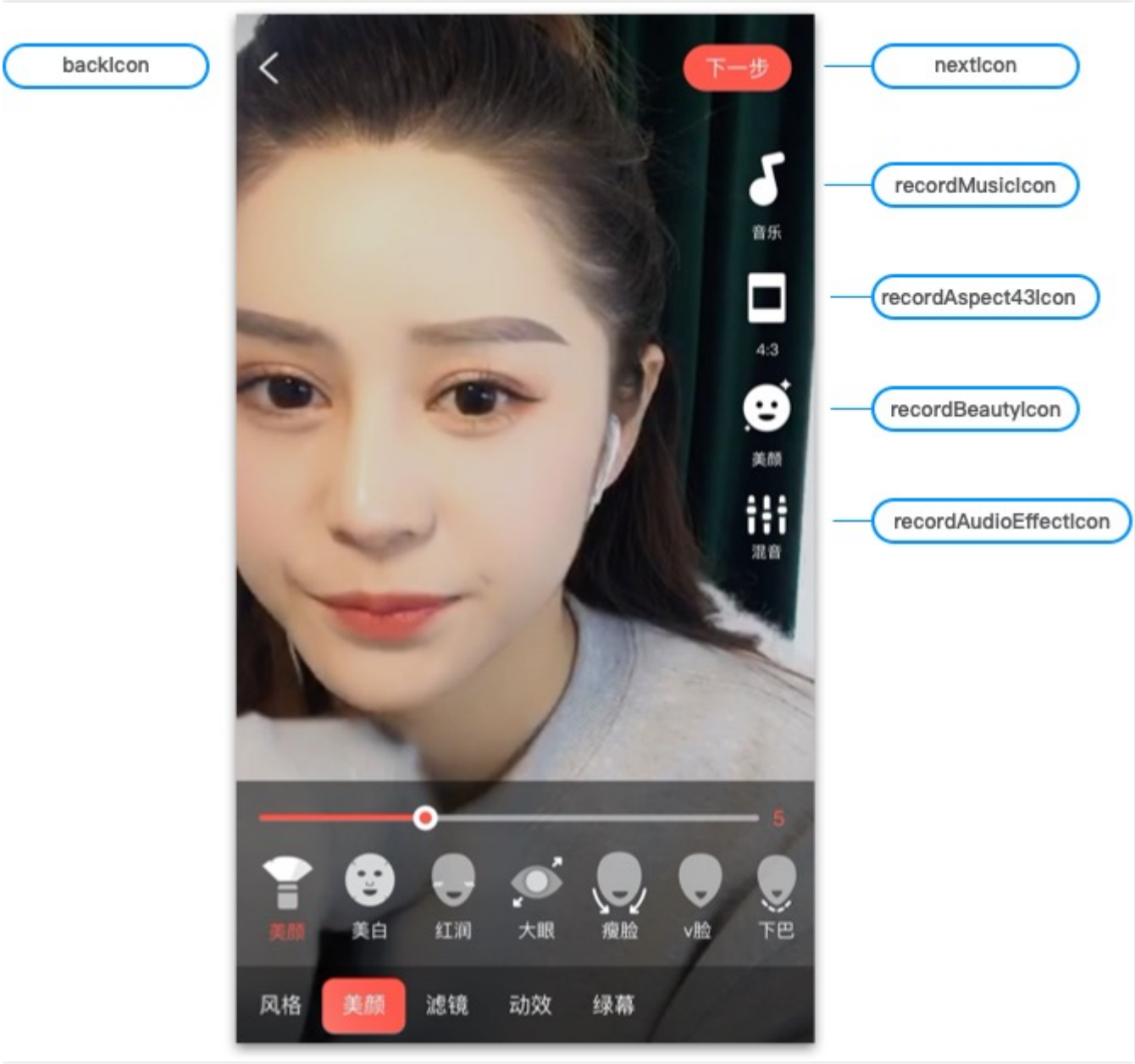
颜色

UGCKit 中所有界面的颜色获取方法均在 UGCKitTheme 类中定义，您可以通过修改对应属性的值来进行修改，具体资源的名请查看 `UGCKitTheme.h` 中的注释。
以 App 界面背景颜色为例:

```
UGCKitTheme *theme = [[UGCKitTheme alloc] init];
theme.backgroundColor = [UIColor whiteColor]; // 改为白色背景
UGCKitEditViewController *editViewController = [[UKEditViewController alloc] init
WithMedia:media config:nil theme:theme]; // 使用自定义主题创建控制器
```

图标

UGCKit 中所有界面的图标均在 `UGCKit.xcassets` 资源中，您可以自由替换。具体图标的文件名可以在 `UGCKitTheme.h` 中查看，图标资源的命名与其中的属性方法名相同，界面上的每个图标在其中均有定义，以录制界面为例，图中的标注为对应图标在 UGCKitTheme 中的属性名，也是在 `UGCKit.xcassets` 中的资源名称。



UGCKit 更换图标有以下两种方式：

- 直接替换换 UGCKitTheme.xcassets 中的图标。
- 通过代码向 UGCKitTheme 对象赋值来进行修改。

使用代码修改录制界面图标的示例如下：

```
UGCKitTheme *theme = [[UGCKitTheme alloc] init];
theme.nextIcon = [UIImage imageName:@"myConfirmIcon"]; // 修改完成按钮图标
theme.recordMusicIcon = [UIImage imageName:@"myMusicIcon"]; // 设置“音乐”功能图标
theme.beautyPanelWhitnessIcon = [UIImage imageNamed:@"beauty_whitness"]; // 设置”美白“效果图标
```

```
UGCKitRecordViewController *viewController = [[UGCKitRecordViewController alloc]
initWithConfig:nil theme:theme]; // 使用自定义主题创建控制器
```

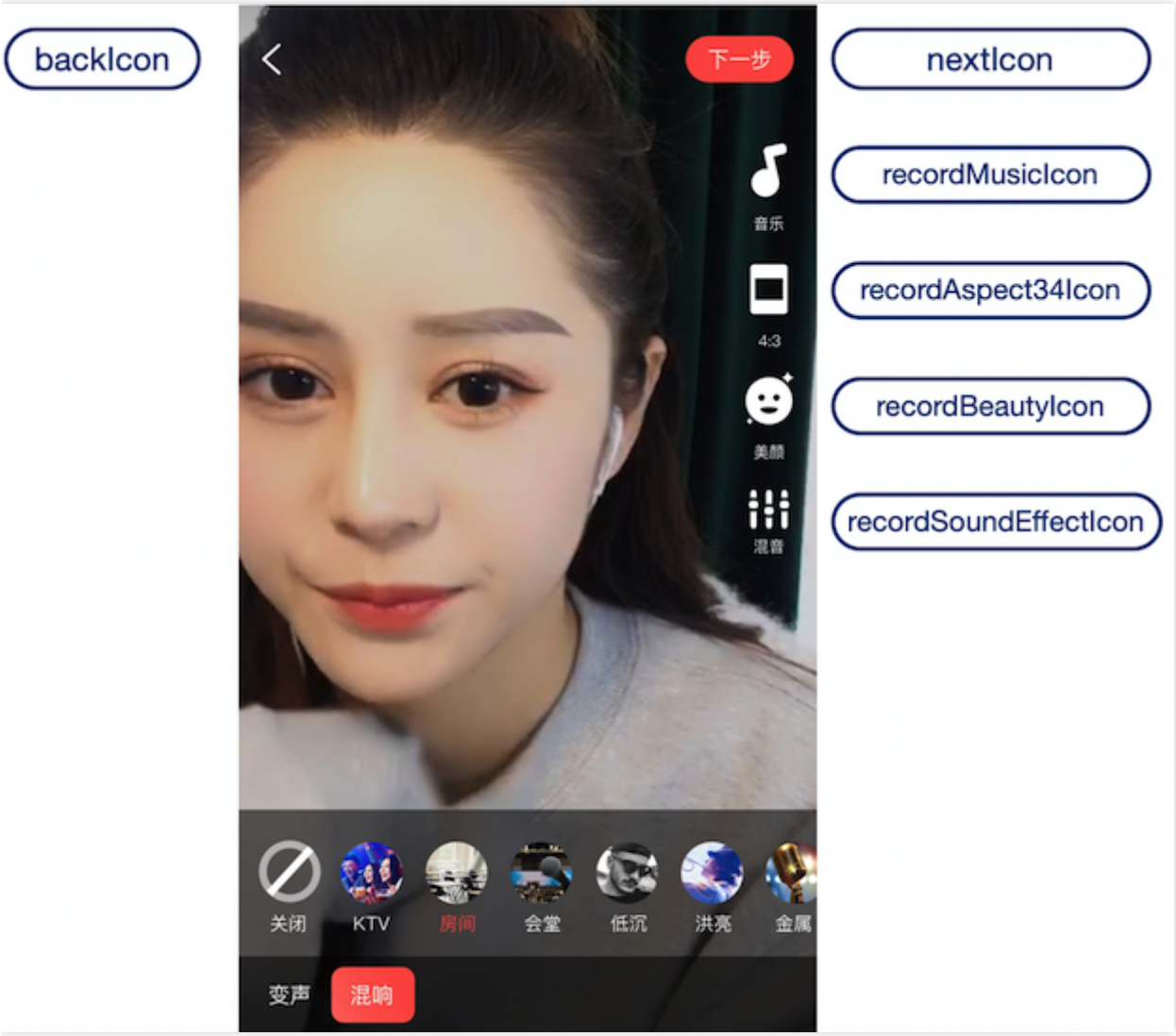
Android

最近更新时间：2022-11-14 18:25:34

UGCKit 是封装好的一整套 UI 交互界面，默认为小视频主题风格。如果您想简单定制自己的主题风格，只需要简单修改主题样式即可实现换图标、换文字、换颜色。

定制录制主题

录制界面包括右侧的图标工具栏、底部的工具栏、音乐面板、美颜面板、音效面板等。

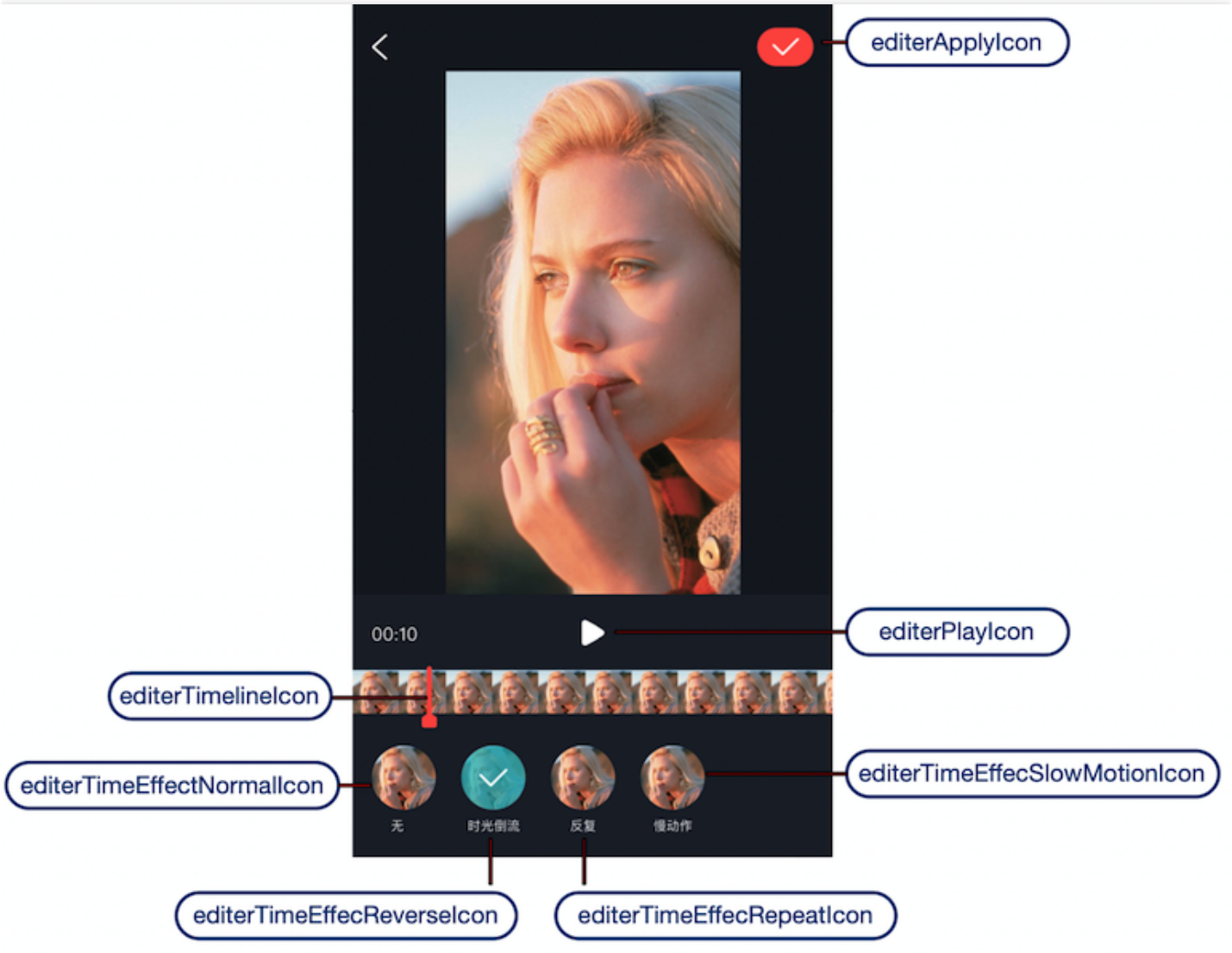


1. 在 `app/res/values/style.xml` 中声明一个 `<style>`，父主题指定为 `RecordStyle`，重写您需要替换的样式，这些可替换的样式在 `app/ugckit/res/values.theme_style.xml` 中可找到。
   如下所示，替换录制界面的音乐图标和美颜图标：

2. 在 `AndroidManifest.xml` 中声明您定制的主题：

定制编辑主题

编辑界面包括编辑裁剪界面、动态滤镜特效面板、速度面板、滤镜面板、贴纸面板、气泡字幕面板。



1. 在 `app/res/values/style.xml` 中声明一个 `<style>`，父主题指定为 `EditerStyle`，重写您需要替换的样式，这些可替换的样式在 `app/ugckit/res/values.theme_style.xml` 中可找到。
   如下所示，替换编辑界面的播放图标和暂停图标：
2. 在 `AndroidManifest.xml` 中声明您定制的主题：