短视频 SDK

单功能集成(Android)

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】

©2013-2022 腾讯云版权所有

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

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

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

【联系我们】

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

文档目录

版权所有：腾讯云计算（北京）有限责任公司

单功能集成(Android)
SDK 集成(Android Studio)

最近更新时间：2021-12-22 16:37:50

Android 工程配置

系统要求

SDK 支持 在 Android 4.0.3（API 15）及以上系统上运行，但只有（Android 4.3）API 18 以上的系统才能开启硬件编码。

开发环境

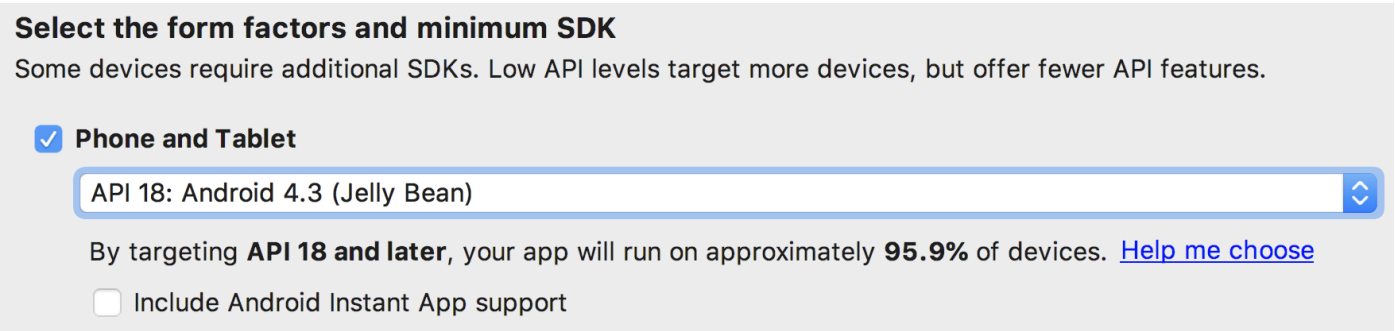
以下是 SDK 的开发环境，App 开发环境不需要与 SDK 一致，但要保证兼容：

- Android NDK：android-ndk-r12b
- Android SDK Tools：android-sdk_25.0.2
- minSdkVersion：15
- targetSdkVersion：26
- Android Studio（推荐您也使用 Android Studio，当然您也可以使用 Eclipse + ADT）

步骤1：集成 SDK

aar 方式集成

1. 新建工程



2. 工程配置

i. 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_7.4.9211', ext: 'aar')
```

```
...
}
```

ii. 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'
}
}
}
```

iii. 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", "armeabi-v7a"
}
}
```

iv. 最后单击 **Sync Now**，编译工程。

jar+so 方式集成

1. 库说明

解压 zip 压缩包后得到 libs 目录，里面主要包含 jar 文件和 so 文件，文件清单如下：

jar 文件	说明
liteavsdk.jar	短视频 SDK Android 核心库

so 文件	说明
libliteavsdk.so	短视频 SDK 核心组件
libtxffmpeg.so	ffmpeg 基础库（ijk 版本），用于点播播放功能，解决一些视频格式的兼容问题

libtxplayer.so	ijkplayer 开源库，用于点播播放功能，解决一些视频格式的兼容问题
libtxsdl.so	ijkplayer 开源库，用于点播播放功能，解决一些视频格式的兼容问题

2. **拷贝文件**

   如果您的工程之前没有指定过 jni 的加载路径，推荐您将刚才得到的 jar 包和 so 库拷贝到 **Demo\app\src\main\jniLibs** 目录下，这是 Android Studio 默认的 jni 加载目录。

   如果您使用的是企业版，那么解压 zip 包后，除了 jar 包和 so 库增加了以外，还多了 assets 目录下的文件，这些是动效所需要的，需要全部拷贝到工程的 assets 目录下，请参见 动效变脸 – 工程配置。

3. **工程配置**

   在工程 App 目录下的 build.gradle 中，添加引用 jar 包和 so 库的代码。

```
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
// 导入腾讯云短视频SDK jar
compile fileTree(dir: 'src/main/jniLibs', includes: ['*.jar'])
...
}
```

4. **减少 APK 体积**

   整个 SDK 的体积主要来自于 so 文件，这些 so 文件是 SDK 正常运行所依赖的音视频编解码库、图像处理库以及声学处理组件，如果短视频 SDK 的功能不是 App 的核心功能，您可以考虑采用在线加载的方式减少最终 apk 安装包的大小。

   i. **上传 so 文件**

   将 SDK 压缩包中的 so 文件上传到 COS，并记录下载地址，例如 http://xxx-appid.cossh.myqcloud.com/so_files.zip。

   ii. **启动准备**

   在用户启动 SDK 相关功能前，例如开始播放视频之前，先用 loading 动画提示用户“正在加载相关的功能模块”。

   iii. **下载 so 文件**

   在用户等待过程中，App 就可以到 http://xxx-appid.cossh.myqcloud.com/so_files.zip 下载 so 文件，并存入应用目录下（例如应用根目录下的 files 文件夹），为了确保这个过程不受运营商 DNS 拦截的影响，请在文件下载完成后校验 so 文件的完整性。

   iv. **加载 so 文件**

   等待所有 so 文件就位以后，调用 TXLiveBase 的 setLibraryPath 将下载的目标 path 设置给 SDK，然后再调用 SDK 的相关功能。之后，SDK 会到这些路径下加载需要的 so 文件并启动相关功能。

gradle 集成方式

1. 在 dependencies 中添加 LiteAVSDK_UGC 的依赖。
   - 若使用3.x版本的 com.android.tools.build:gradle 工具，请执行以下命令：

```
dependencies {
implementation 'com.tencent.liteav:LiteAVSDK_UGC:latest.release'
}
```

   - 若使用2.x版本的 com.android.tools.build:gradle 工具，请执行以下命令：

```
dependencies {
compile 'com.tencent.liteav:LiteAVSDK_UGC:latest.release'
}
```

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

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

> 说明：
> 目前 SDK 支持 armeabi、armeabi-v7a 和 arm64-v8a。

3. 单击 **Sync Now**，自动下载 SDK 并集成到工程里。

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：

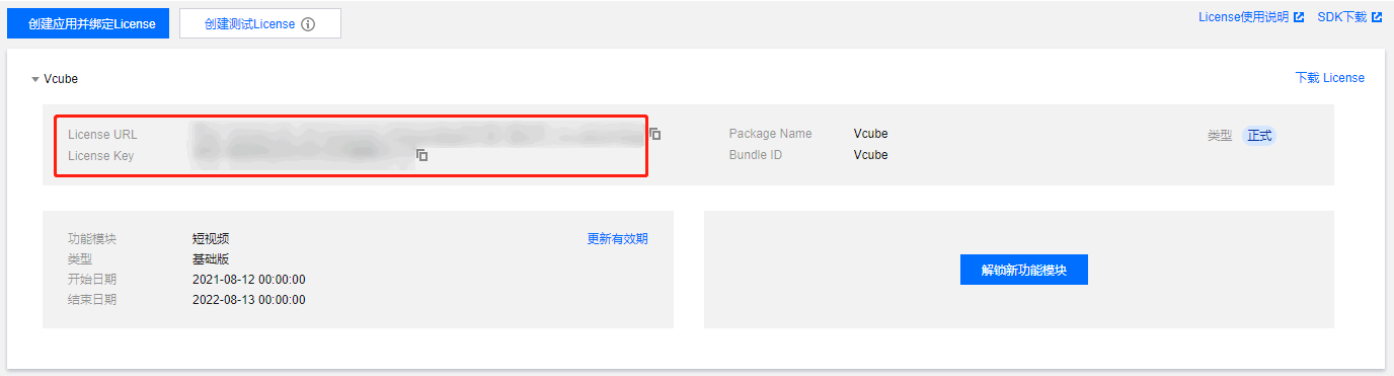
```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

版权所有：腾讯云计算（北京）有限责任公司

```
<uses-permission android:name="android.permission.READ_LOGS" />
<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. 参考 License 申请 的指引申请 License 后，从 云点播控制台 复制 License Key 和 License URL，如下图所示：



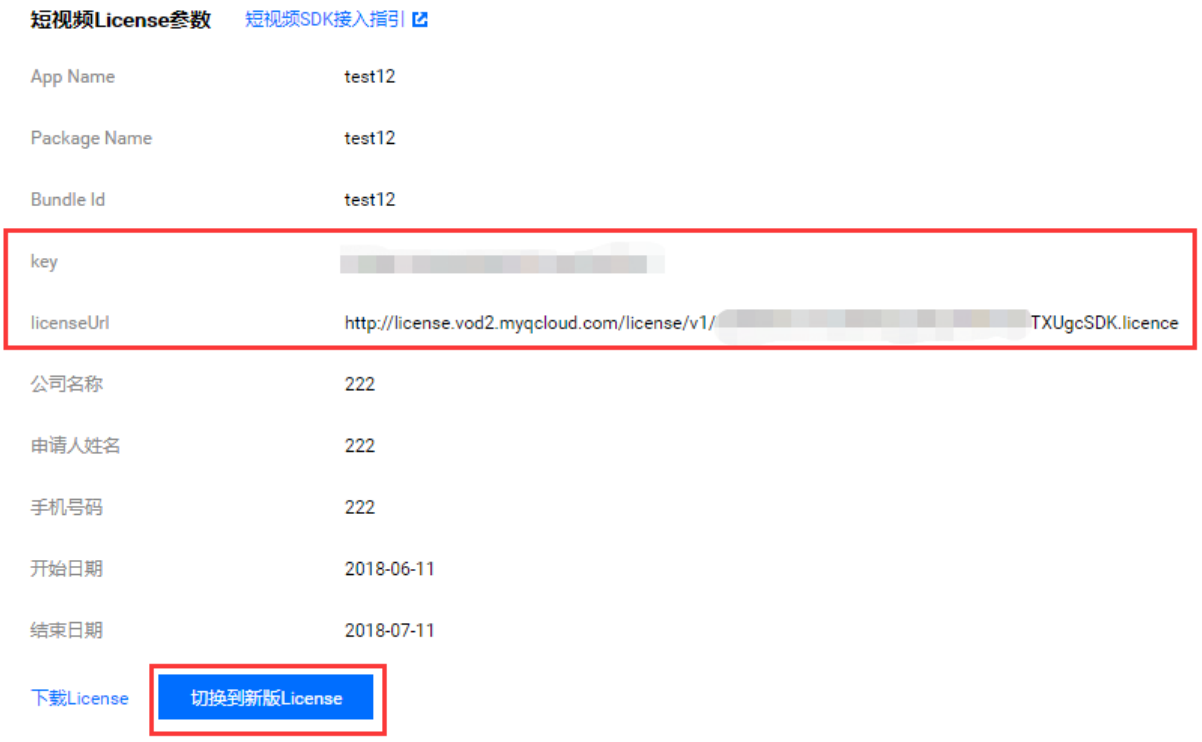
2. 在您的应用中使用短视频功能之前，建议在 - Application onCreate()中进行如下设置：

```
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/com.tencent.liteav.demo/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。
- **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(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：

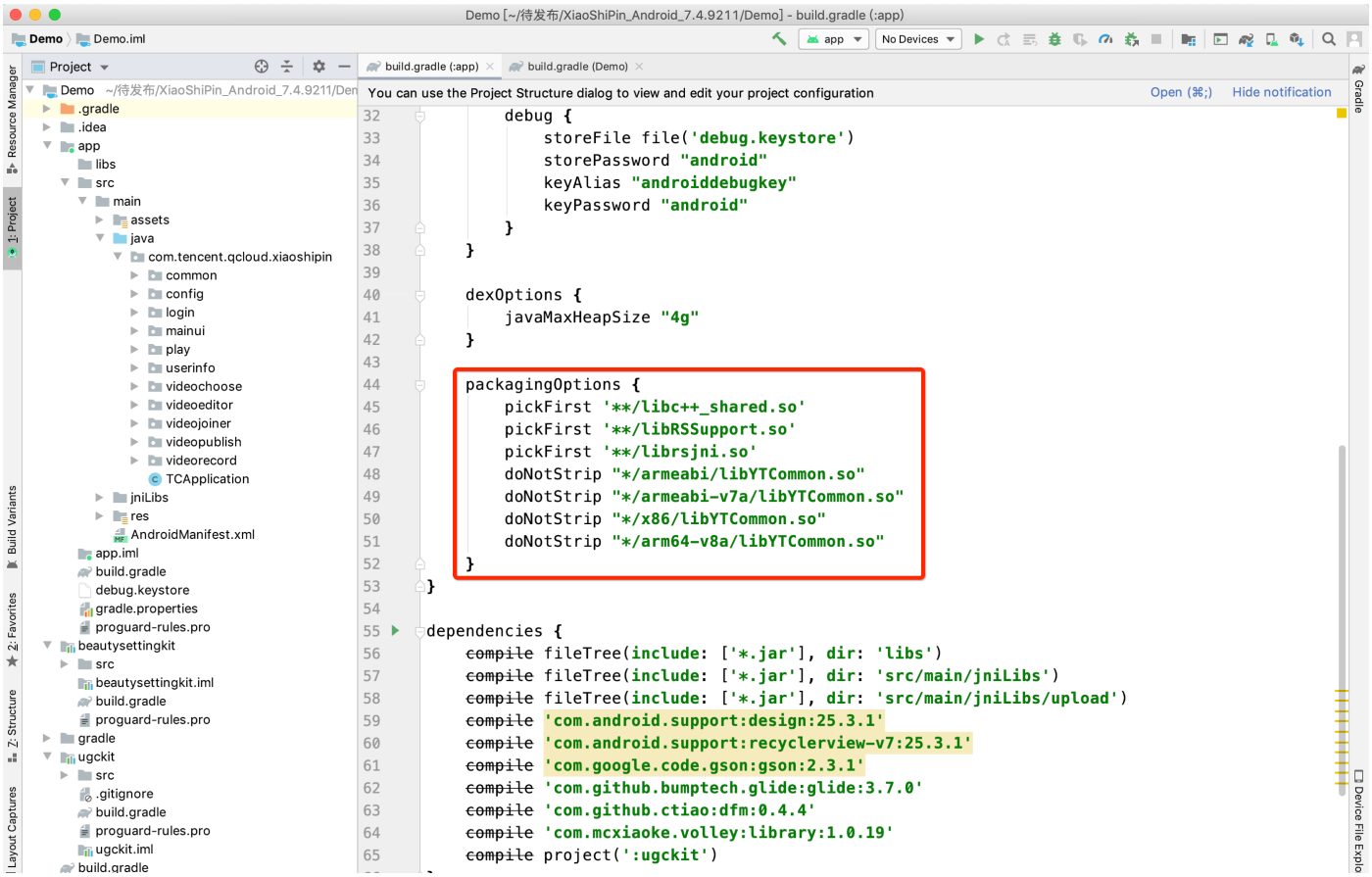
1. **确认是否已经将 SDK 中的 jar 包和 so 库放在 jniLibs目录下。**
2. **如果您使用 aar 集成方式的完整版本，在工程目录下的 build.gradle 的 defaultConfig 里面确认下是否将 x64 架构的 so 库过滤掉。因为完整版本中连麦功能所使用的声学组件库暂时不支持 x64 架构的手机。**

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

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

步骤1：新建工程（Empty Activity）

1. 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。
2. 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 = 25
buildToolsVersion = "25.0.2"
supportSdkVersion = "25.4.0"
minSdkVersion = 16
targetSdkVersion = 23
versionCode = 1
versionName = "v1.0"
proguard = true
rootPrj = "$projectDir/.."
```

```
ndkAbi = 'armeabi-v7a'
noffmpeg = false
noijkplay = false
aekit_version = '1.0.16-cloud'
liteavSdk="com.tencent.liteav:LiteAVSDK_Professional:latest.release"
}
# 拷贝结束
```

3. 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'), 'proguard-rules.pro'
}
}
compileOptions {
sourceCompatibility JavaVersion.VERSION_1_8
targetCompatibility JavaVersion.VERSION_1_8
}
}

# 如果您使用的是商业版或商业版Pro，请加如下这段，基础版/精简版不需要
packagingOptions {
pickFirst '**/libc++_shared.so'
doNotStrip "*/armeabi/libYTCommon.so"
doNotStrip "*/armeabi-v7a/libYTCommon.so"
doNotStrip "*/x86/libYTCommon.so"
doNotStrip "*/arm64-v8a/libYTCommon.so"
}
# 如果您使用的是商业版或商业版Pro，请加如上这段，基础版/精简版不需要

dependencies {
# 拷贝开始
compile fileTree(include: ['*.jar'], dir: 'libs')
compile fileTree(include: ['*.jar'], dir: 'src/main/jniLibs')
compile 'com.mcxiaoke.volley:library:1.0.19'
compile 'com.android.support:design:25.3.1'
compile 'com.android.support:recyclerview-v7:25.3.1'
compile 'com.google.code.gson:gson:2.3.1'
compile 'com.github.bumptech.glide:glide:3.7.0'
compile 'com.github.ctiao:dfm:0.4.4'
compile project(':ugckit')
compile 'com.android.support.constraint:constraint-layout:1.1.3'
# 拷贝结束
}
```

4. 配置 Gradle 版本：

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

步骤2：导入相关 module

1. 拷贝 ugckit module 到 您新建的工程 ugc 目录下。
2. 拷贝 beautysettingkit module 到 您新建的工程 ugc 目录下。
3. 在工程的 settings.gradle中导入 ugckit。
4. 在新建的工程 UGC/settings.gradle 下指明引入这几个 module：

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

5. 在工程 app module 中依赖 UGCKit module：

```
compile project(':ugckit')
```

步骤3：申请 Licence

在使用 UGCKit 之前要先设置 License，License 的获取方法请参见 License申请。

实现录制、导入、裁剪、特效编辑功能

1. 设置 Licence，初始化 UGCKit

在 Application.java 中设置 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

1. 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" />
```

2. 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"/>
```

3. 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。

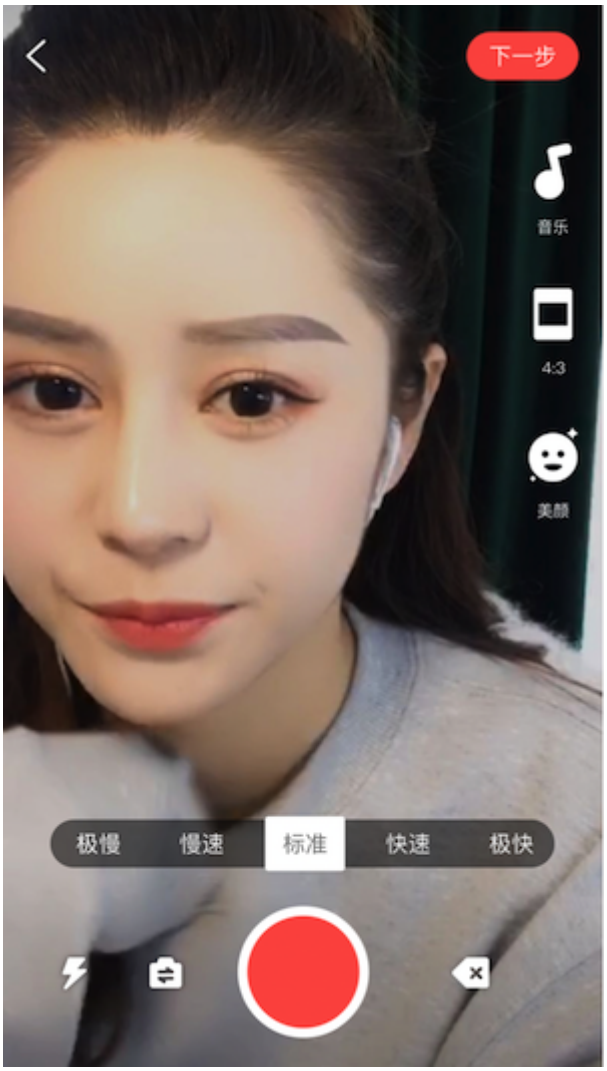
```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// 必须在代码中设置主题(setTheme)或者在AndroidManifest中设置主题(android:theme)
setTheme(R.style.RecordActivityTheme);
setContentView(R.layout.activity_video_record);
// 获取UGCKitVideoRecord
mUGCKitVideoRecord = (UGCKitVideoRecord) findViewById(R.id.video_record_layout);
// 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, @N
onNull 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" />
```

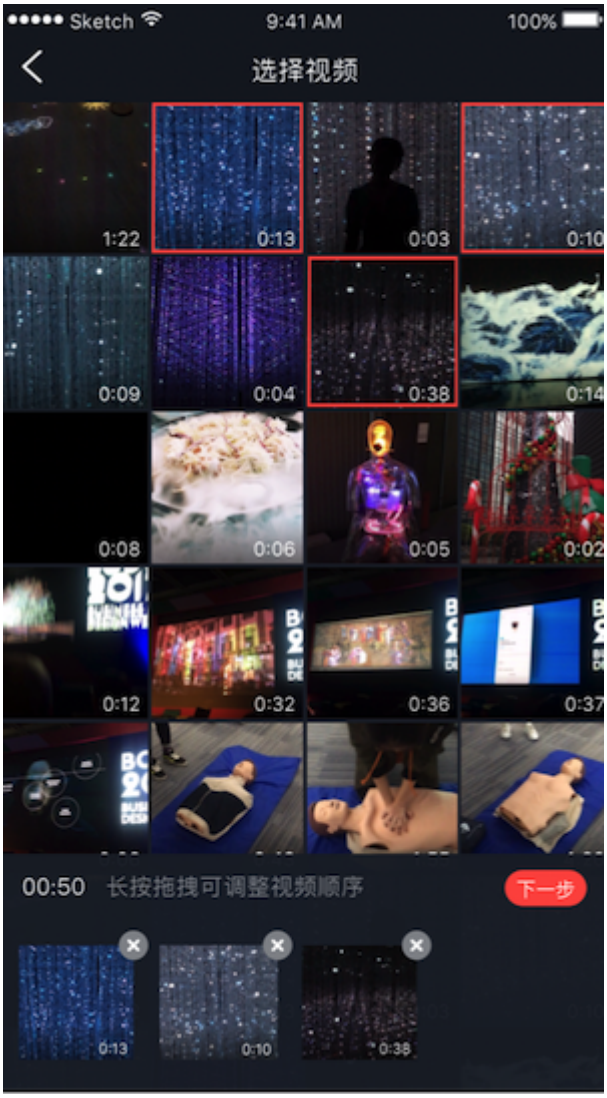
2. 在 res/values/styles.xml 中新建空的主题，继承 UGCKit 默认视频导入主题。

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

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

```
@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。

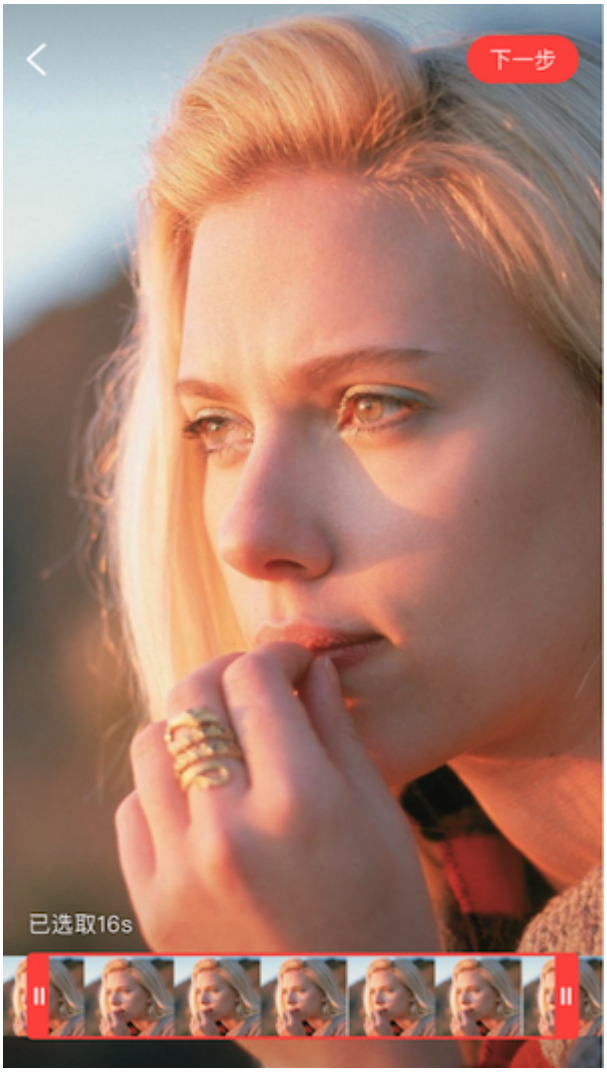
```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// 必须在代码中设置主题(setTheme)或者在AndroidManifest中设置主题(android:theme)
setTheme(R.style.EditerActivityTheme);
setContentView(R.layout.activity_video_cut);
mUGCKitVideoCut = (UGCKitVideoCut) findViewById(R.id.video_cutter);
// 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
- 动效变脸（企业版）

常见问题

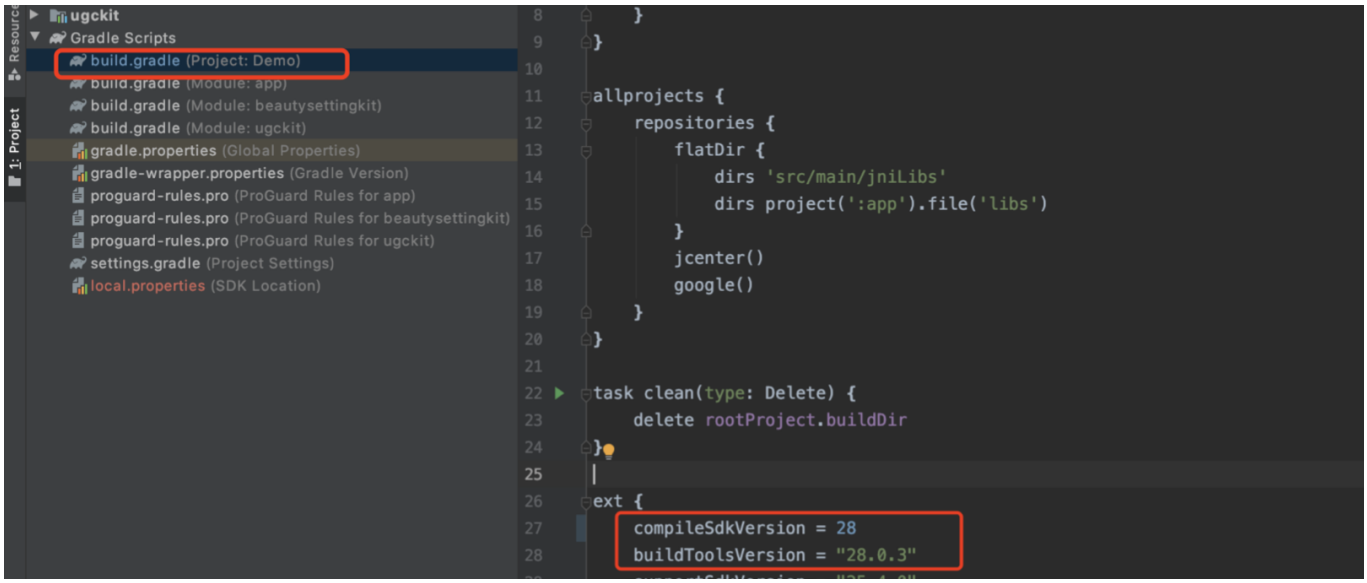
是否支持 AndroidX?

因为服务客户较多，且大部分客户的工程中目前仍在使用 support 包，基于此，目前 UGCKit 暂时还是基于 support 包。但是考虑到客户对 Androidx 的需求，现提供 UGCKit 迁移 Androidx 方案文档。

为了方便说明，以腾讯云 UGSVSDK 为例，此 Demo 中同样使用了 UGCKit 模块。

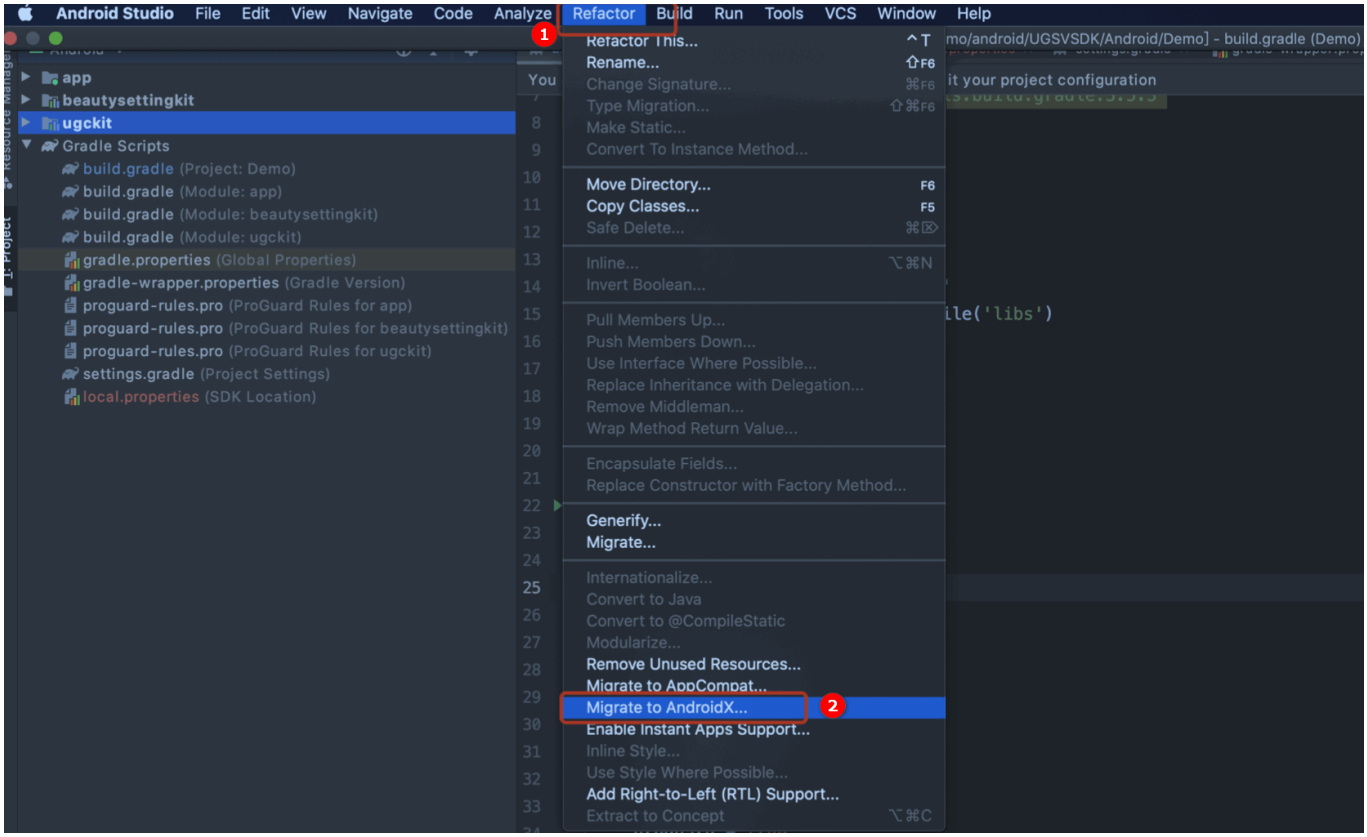
1. **前提准备：**
   - 将Android Studio更新至 Android Studio 3.2及以上。
   - Gradle 插件版本改为 4.6及以上。

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

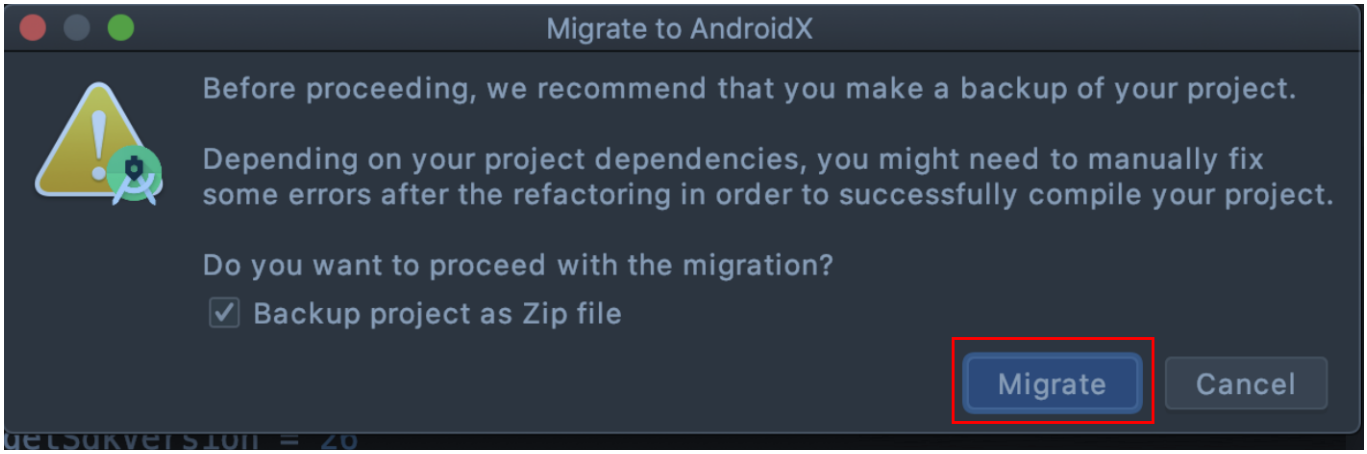


2. **开启迁移：**

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**:

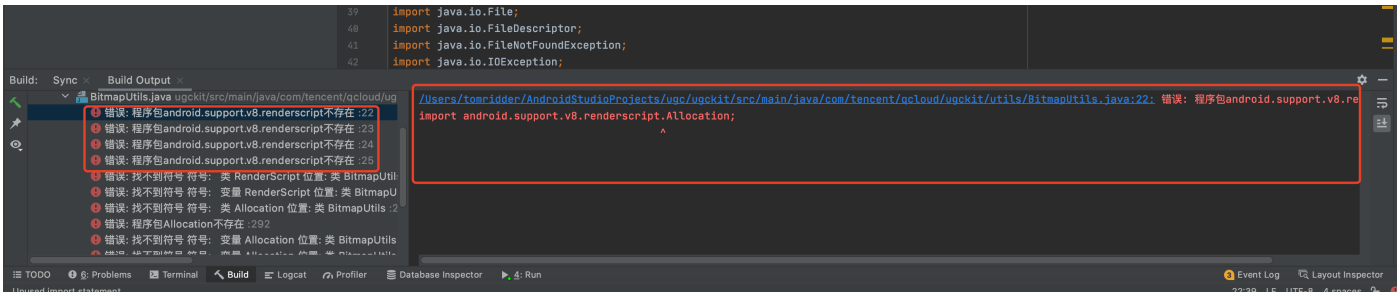
```
ERROR: Unable to find method 'org.gradle.api.tasks.compile.CompileOptions.setBootClasspath(Ljava/lang/String;)V'.
Possible causes for this unexpected error include:
```

- **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(Android)

最近更新时间：2021-08-23 18:06:52

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

使用流程

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.drawable.
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);

// 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);
```

版权所有：腾讯云计算（北京）有限责任公司

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(Android)

最近更新时间：2021-01-27 15:47:30

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();
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 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) ;
```

录制草稿箱(Android)

最近更新时间：2021-12-20 16:20:03

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();
```

版权所有：腾讯云计算（北京）有限责任公司

```
// 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();
```

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(Android)

最近更新时间：2022-01-06 16:22:33

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(Android)

最近更新时间：2021-01-27 15:36:27

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(Android)

最近更新时间：2021-10-08 10:21:19

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().getVideoFileInfo(sou
rcePath);
```

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, TXThumbnailListener list
ener)
```

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, mThu
mbnailListener);

private TXVideoEditer.TXThumbnailListener mThumbnailListener = new TXVideoEditer.TXThum
bnailListener() {
@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)
```

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, floa
t 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 duration);
```

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。
Demo 实例：设置水印在片尾中间，持续3秒

```
Bitmap tailWaterMarkBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.tclo
ud_logo);
TXVideoEditConstants.TXRect txRect = new TXVideoEditConstants.TXRect();
txRect.x = (mTXVideoInfo.width - tailWaterMarkBitmap.getWidth()) / (2f * mTXVideoInfo.width);
txRect.y = (mTXVideoInfo.height - tailWaterMarkBitmap.getHeight()) / (2f * mTXVideoInfo.heigh
t);
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)
```

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

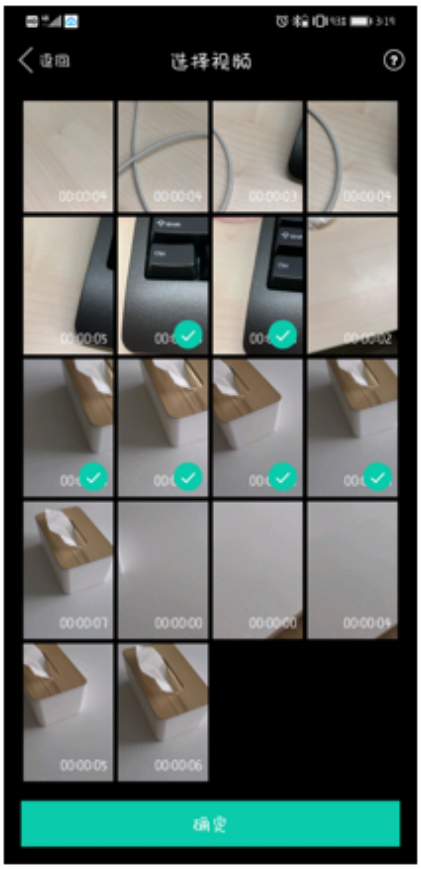
- 贴纸字幕
- 图片编辑

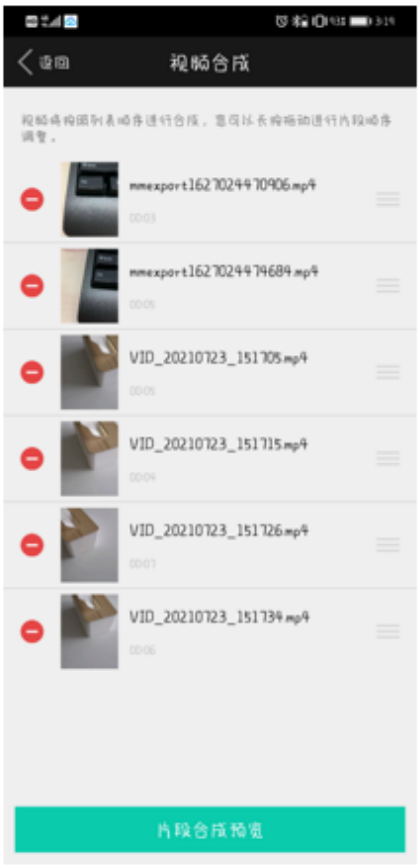
视频拼接(Android)

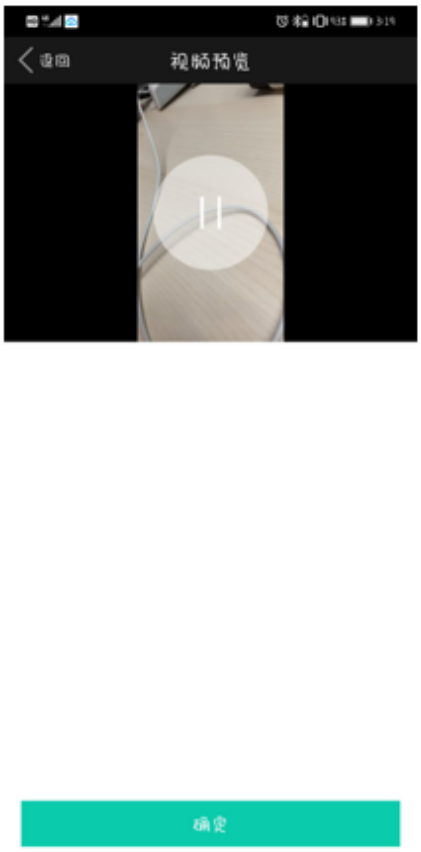
最近更新时间：2021-07-26 09:46:06

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.TXPreviewPara
m();
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, mVideoOutputPat
h);
```

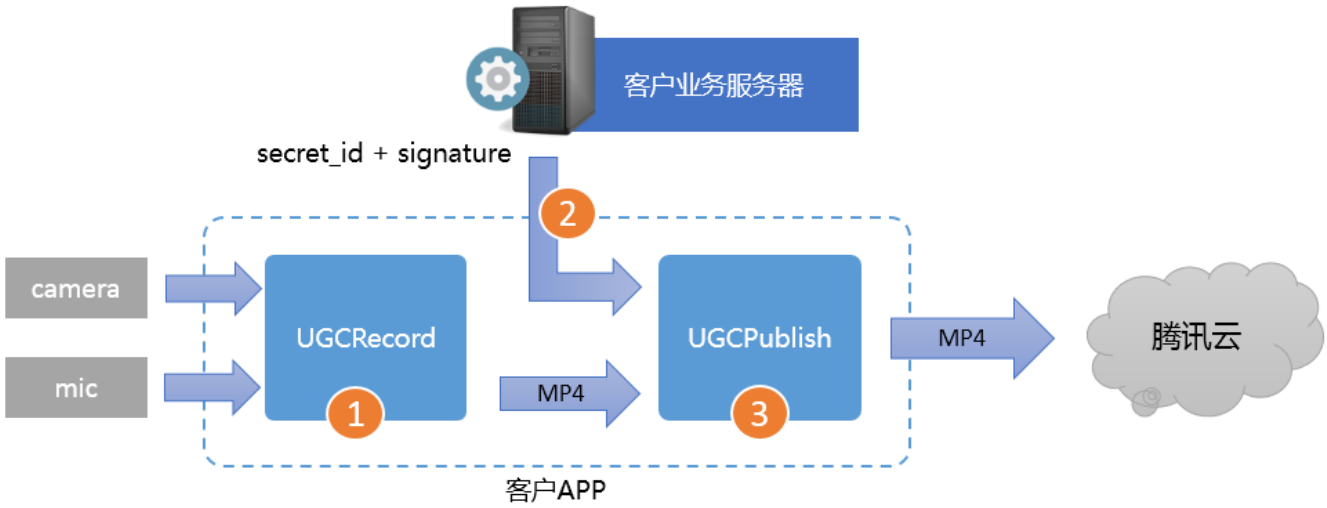
合成时指定文件压缩质量和输出路径，输出的进度和结果会通过 TXVideoJoiner.TXVideoJoinerListener 以回调的形式通知用户。

上传和播放
视频上传(Android)

最近更新时间：2021-07-26 09:49:58

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。

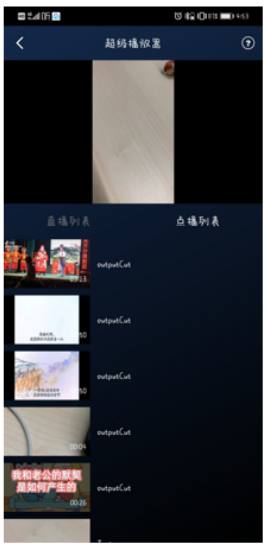
对接攻略

上传入口

选择视频

压缩视频

预览发布



视频点播

请参见 Android 上传 SDK 来接入短视频上传功能。

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.getApplicationContext());
// 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。

Android 播放器 SDK

最近更新时间：2022-03-08 14:22:20

Android 超级播放器 SDK 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。

SDK 下载

点播 Android 超级播放器的项目地址是 SuperPlayer_Android。

阅读对象

本文档部分内容为腾讯云专属能力，使用前请开通 腾讯云 相关服务，未注册用户可注册账号 免费试用。

快速集成

aar 集成

1. 下载 SDK + Demo 开发包，项目地址为 Android。
2. 导入SDK/LiteAVSDK_XXX.aar 以及 Demo/superplayerkit 这个 module 复制到工程中。
3. 在 app/build.gralde 中添加依赖：

```
compile(name: 'LiteAVSDK_Professional', ext: 'aar')
compile(name: 'libsuperplayer', ext: 'aar')
// 超级播放器弹幕集成的第三方库
compile 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
```

4. 在项目 build.gralde 中添加：

```
...
allprojects {
repositories {
flatDir {
dirs 'libs'
}
...
```

```
}
}
...
```

5. 权限声明：

```
<!--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" />
```

> ⚠ 注意：
>
> lib_tcsuperplayer.aar 以 moudle 方式开源，您可在 Demo/lib_tcsuperplayer 中找到所有源代码。

使用播放器

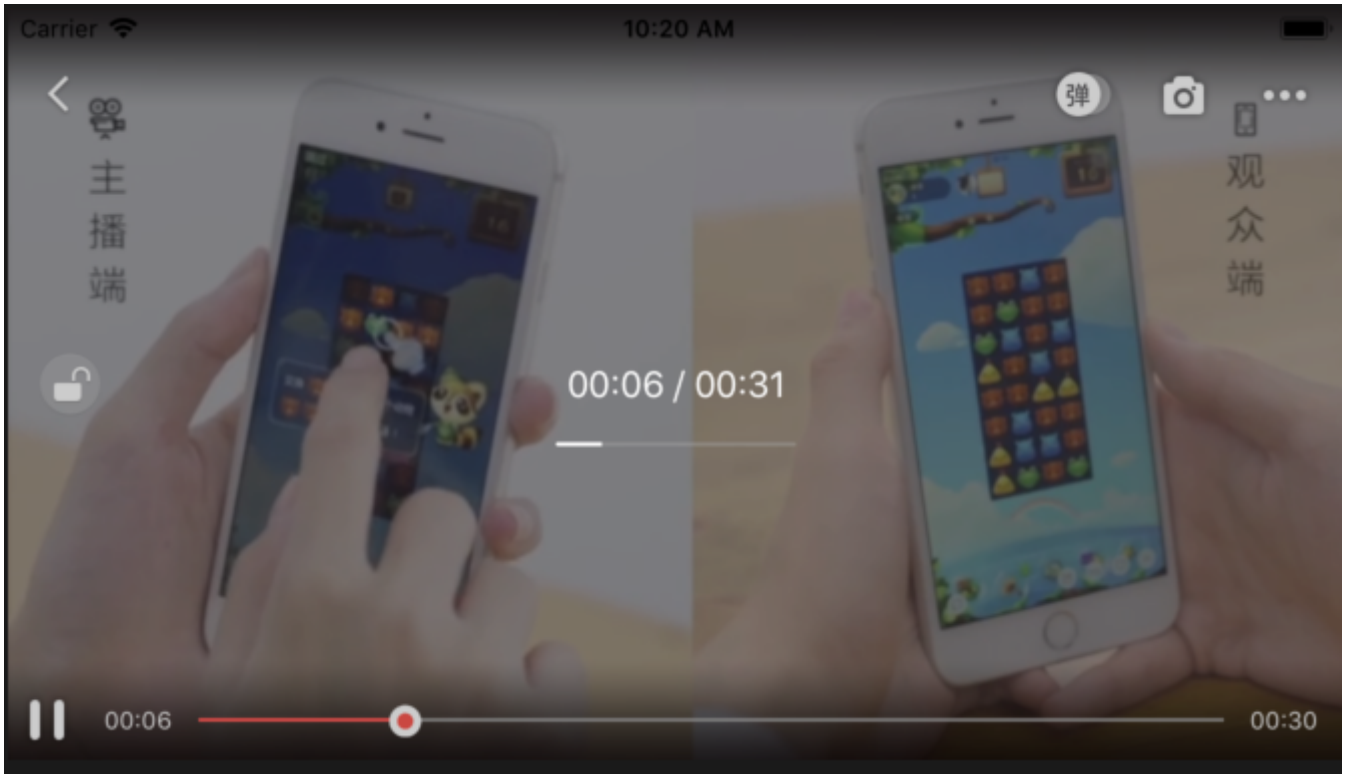
播放器主类为 SuperPlayerView ，创建后即可播放视频。

```
//不开防盗链
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329073;// 配置 AppId
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "5285890799710670616"; // 配置 FileId
mSuperPlayerView.playWithModel(model);

//开启防盗链需填写 psign， psign 即超级播放器签名，签名介绍和生成方式参见链接：https://cloud.tencent.com/document/product/266/42436
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329071;// 配置 AppId
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "5285890799710173650"; // 配置 FileId
mSuperPlayerView.playWithModel(model);
```

```
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZmlsZUlkIjoiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsImV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-LzU";
mSuperPlayerView.playWithModel(model);
```

运行代码，可以看到视频在手机上播放，并且界面上大部分功能都处于可用状态。



选择 FileId

视频 FileId 在一般是在视频上传后，由服务器返回：

1. 客户端视频发布后，服务器会返回 FileId 到客户端。
2. 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：

打点功能

在播放长视频时，打点信息有助于观众找到感兴趣的点。使用 修改媒体文件属性 API，通过 AddKeyFrameDescs.N 参数可以为视频设置打点信息。

调用后，播放器的界面会增加新的元素。

小窗播放

小窗播放可以悬浮在所有 Activity 之上播放。使用小窗播放非常简单，只需要在开始播放前调用下面代码即可：

```
// 播放器配置
SuperPlayerGlobalConfig prefs = SuperPlayerGlobalConfig.getInstance();
// 开启悬浮窗播放
prefs.enableFloatWindow = true;
```

```
//设置悬浮窗的初始位置和宽高
SuperPlayerGlobalConfig.TXRect rect = new SuperPlayerGlobalConfig.TXRect();
rect.x = 0;
rect.y = 0;
rect.width = 810;
rect.height = 540;
// ...其他配置
```

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();
```

更多功能

完整功能可扫码下载视频云工具包体验，或直接运行工程 Demo。