腾讯云智能数智人

端渲染 SDK 接入（2D形象）

腾讯云

【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或95716。

文档目录

端渲染 SDK 接入（2D形象）
2D 端渲染场景概述

最近更新时间：2026-03-30 18:49:58

一、产品简介

腾讯云智能 2D 小样本数智人客户端渲染 SDK（以下简称 2D 端渲染 SDK）是一款高效的数智人终端集成解决方案。该 SDK 支持在终端设备上完成数智人的实时驱动与渲染；集成后，人像和口型渲染可以在用户终端设备上完成，适合并发量大的数智人交互场景。

核心特性：

- **终端渲染：** 人像与口唇驱动渲染均在用户设备本地完成。
- **实时驱动：** 通过输入流式音频，可实时驱动数智人口型。
- **灵活集成：** 提供透明背景的视频流输出，便于客户端自定义背景与布局。

授权模式概览：

授权模式	2D 端渲染 SDK 授权使用年包-按应用	2D 端渲染 SDK 授权使用年包-按设备
适合设备	支持移动端（Android/iOS），建议设备： 1. 智能手机 2. 平板电脑（8 - 13英寸） 3. 小尺寸智慧屏（16英寸以下）	支持大屏端（Android）设备，建议设备： 1. 智慧大屏 2. 大屏广告机
支持的分辨率	540P、720P	1080P
支持的设备数	购买1个授权，即可覆盖一个应用（Bundle ID/Application ID）在 Android 与 iOS 双端不限设备数 的使用。	根据实际部署的终端设备数量按需购买，1个授权对应1台设备。

说明：

- 该 SDK 授权不包含 TTS（语音合成）服务。可接入第三方音频以驱动数智人；如需使用数字人音色，需额外购买“通用音频合成资源”。（接入方式详见 TTS SDK 接入）
- 2D 端渲染 SDK 年包（按应用）在移动端使用时，可额外购买“轻量版云端服务小时包”，在设备性能不足时，使用云端渲染服务，扩展对低性能设备的兼容性。（接入方式详见 端云混合渲染方案）

二、SDK 主要信息及功能

功能	功能描述	
SDK 名称	2D 数智人客户端渲染 SDK	
SDK 开发提供商	腾讯云计算（北京）有限责任公司	
SDK 功能	数智人口型驱动	支持输入流式 PCM 音频，实时驱动数智人口型。
	数智人渲染	1. 在客户端加载并渲染数智人模型，输出带透明通道的视频流，便于与任意背景融合。 2. 支持在客户端灵活设置数智人背景、位置，并可自由缩放数智人大小，适配多样化的 UI 界面。
SDK 下载地址	登录 数智人平台 ->创建会话互动项目->选择对应的数字人类型->选择“端渲染”及其他配置信息->确定后，调整“形象设置”及音色->SDK 接入->下载 SDK （**注意：只有登录账号拥有数字人形象的前提下，才能创建会话互动项目**）	
SDK 当前版本	Android 1.7.x　iOS 1.7.x	
个人信息处理规则	2D 数智人客户端渲染 SDK 个人信息处理规则	
开发者合规使用指南	2D 数智人客户端渲染 SDK 合规使用指南	

三、购买及接入流程

1. 购买授权

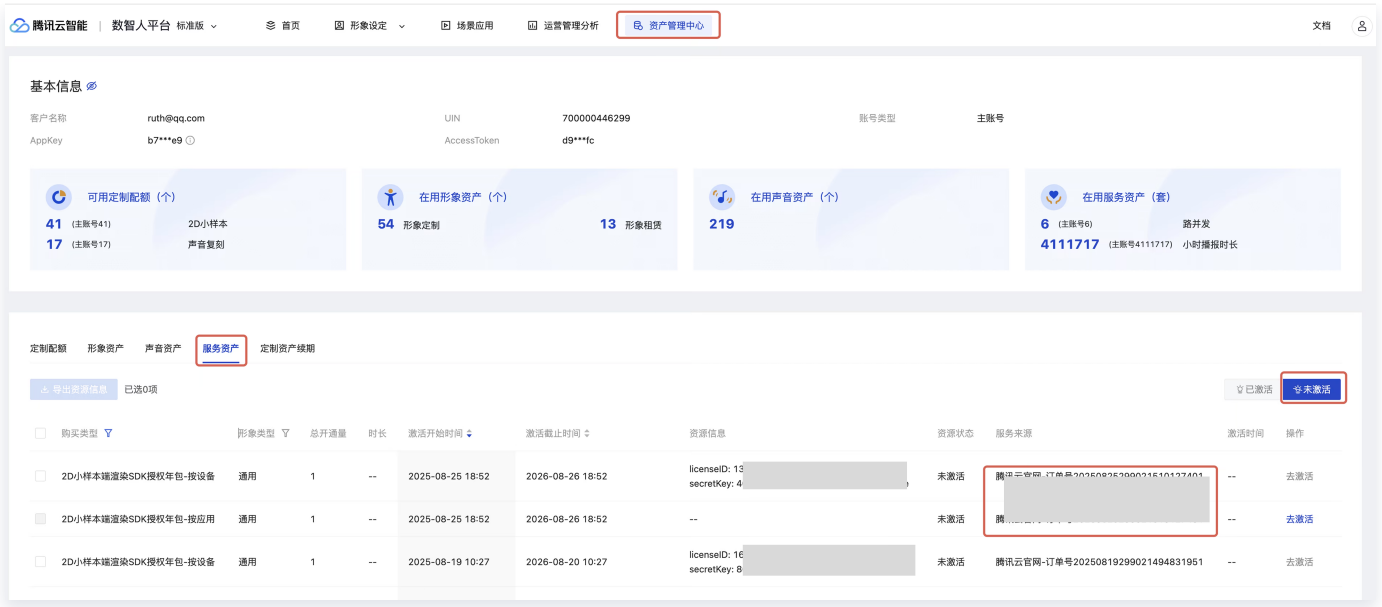
1）前往 官网下单 页面，在导航栏中选择 **2D 形象 > 2D 端渲染**，找到“2D 端渲染 SDK 授权使用年包”，根据您的业务场景选择按应用或按设备授权模式，点击立即购买并完成支付。如下图所示：

2）找到“**2D 端渲染 SDK 授权使用年包**”的资源包，选择合适的授权模式后，单击**立即购买**，如下图所示：

2. 获取 License 与 SDK

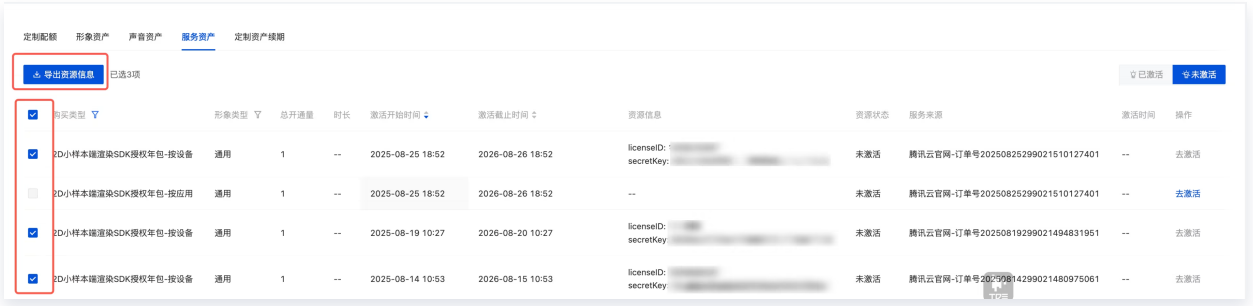
下单完成后，请登录 数智人平台，按以下路径获取授权信息（License）及 SDK 文件：

1）按照“**资产管理中心 > 服务资产 > 未激活**”步骤找到对应的订单信息，如下图：

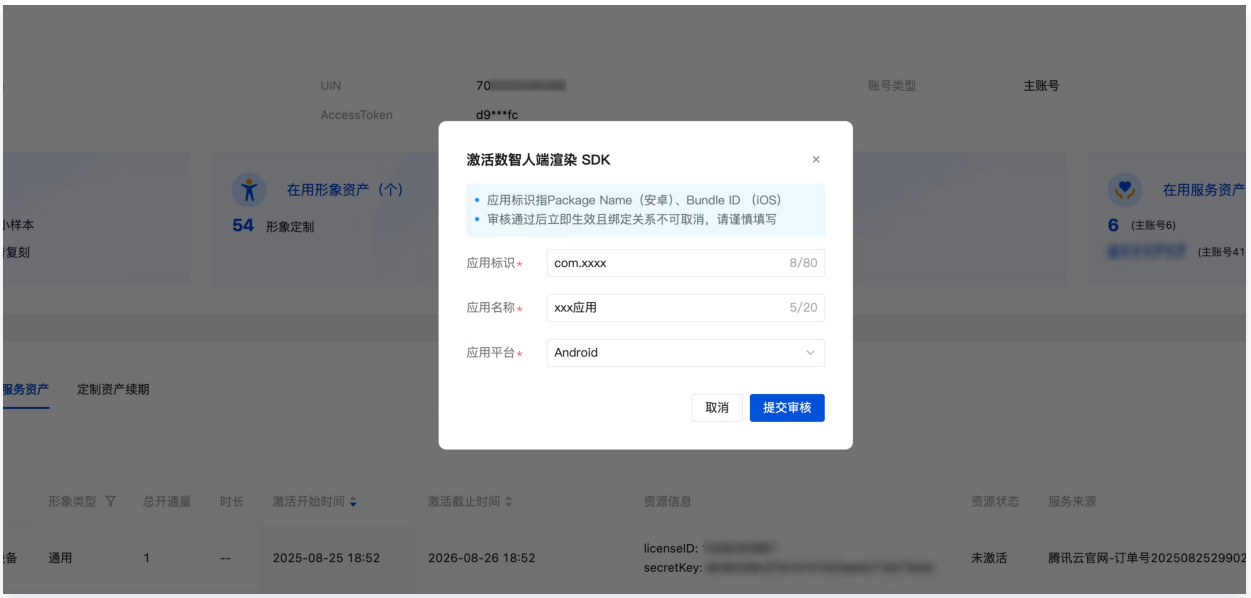


2）根据您购买的授权模式，操作流程有所不同：

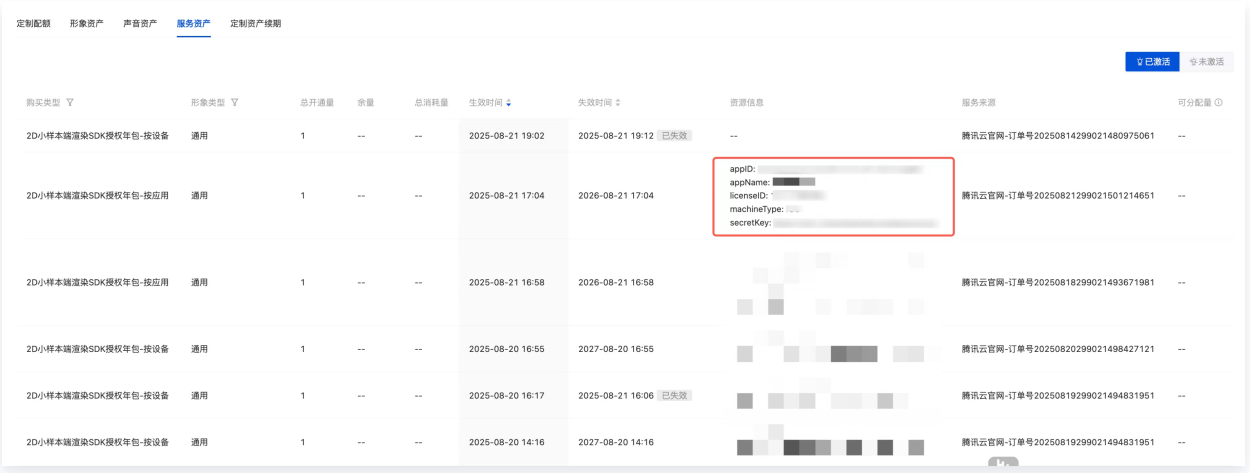
- **按设备授权模式：**
  - 系统将自动生成 licenseID 和 secretKey，您可直接复制这些信息，或通过复选框批量操作。如下图：
  - **激活机制：**License 的激活状态由设备端在联网状态下首次使用 SDK 时触发，平台端无需手动操作。激活后，可在已激活页面查看有效期等信息。



  - **激活机制：**License 的激活状态由设备端在联网状态下首次使用 SDK 时触发，平台端无需手动操作。激活后，可在已激活页面查看有效期等信息。
- **按应用授权模式：**
  - 您需要主动提交应用信息以供审核。
  - **具体流程：**单击**去激活 > 填写应用信息 > 提交审核**（如下图）。



- 审核通过后（通常为1个工作日内），平台将生成 LicenseID 和 SecretKey。
- 获取路径为：**资产管理中心 > 服务资产 > 已激活**。如下图：



3. 快速接入

获取授权和 SDK 文件后，请参考对应的接入文档进行集成开发：

- 移动端 Android SDK 接入指南：移动端 Android SDK 接入
- 移动端 iOS SDK 接入指南：移动端 iOS SDK 接入
- 大屏端 Android SDK 接入指南：大屏端 Android SDK 接入
- 端云混合渲染方案接入指南：端云混合渲染方案

四、其他相关服务

根据您的业务需求，您可选购以下资源：

1. 通用音频合成资源

若需使用数智人专属音色进行播报，需要额外购买此项服务。具体如下图所示：

2. 轻量版云端服务小时包

为解决部分终端设备性能不足导致的渲染问题，我们提供**端云混合渲染方案**。当设备性能充足时，使用本地渲染；当设备性能不足时，可自动切换至云端 CPU 进行推理，此时需购买此小时包。具体如下图所示：

> **说明：** 云端 CPU 推理支持的分辨率为720P。

五、资源占用说明

1. 基础 SDK 包体（渲染引擎）

此部分需集成到应用安装包中，直接影响应用初始下载体积。

平台	压缩前大小	压缩后大小（下载体积）	说明
Android（APK）	24MB	≈7.5MB	经 ZIP 压缩优化，减少下载体积。
iOS（IPA）	38.8MB	≈12.5MB	

2. 模型资源包

模型包为动态下载资源，按需加载至设备存储，不影响应用初始安装包的大小。

- **基础模型包（必备组件）**

类型	压缩前大小	压缩后大小（下载体积）	说明
2D 渲染引擎	82MB	≈58MB	支持所有数智人形象基础渲染能力的核心组件。

- **数智人形象包**

形象类型	压缩前大小	压缩后大小（下载体积）	分辨率	内容说明
照片数智人	38MB	≈25MB	540p	基于1张静态照片建模生成的数字人形象。
SDK 默认形象	60MB	≈43MB	540p	预置的公共形象，包含约1分30秒的动态视频素材。
用户定制形象	30 – 200MB	依定制内容而定	–	大小与定制素材分辨率、时长正相关，每个形象独立成包。

六、版本更新记录

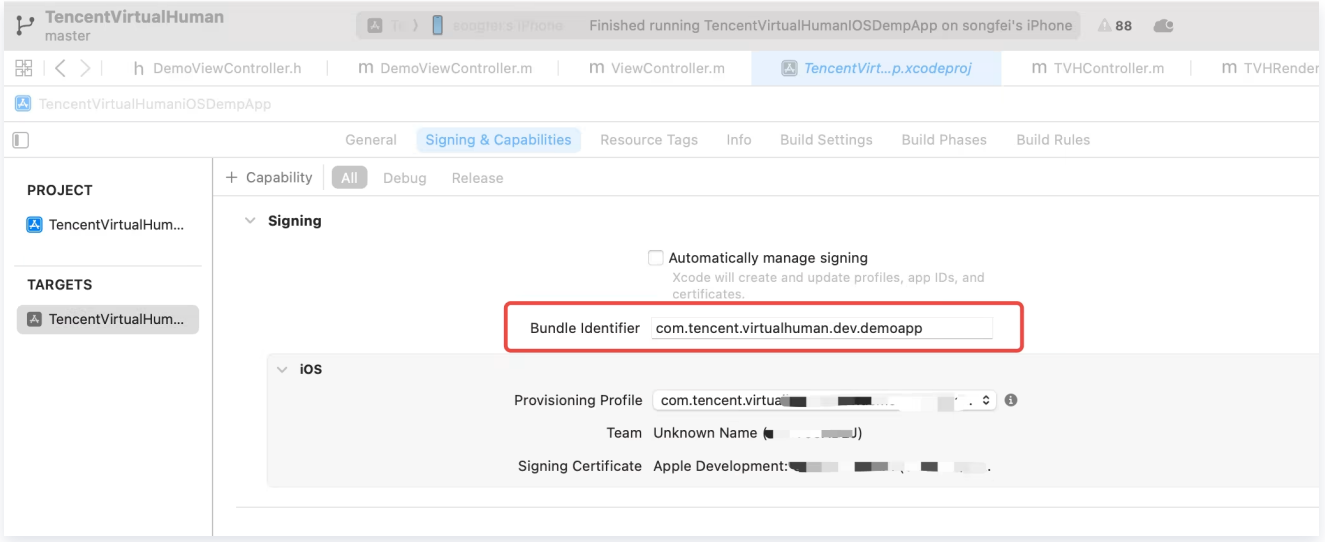
SDK 版本	主要优化与特性	适配说明
V1.7	• **新增功能：** 支持 2D 精品数智人，可以自主插入动作 • **问题修复：** 修复稳定性和内存问题 • **功能变更：** interrupt() 接口增加返回值，标识打断是否成功	SDK 版本号以 1.7.*结尾；需搭配版本号以 V3 结尾的形象包使用。
V1.6	• **新增功能：** – 支持性能更好的 480p 模型推理 – 通用模型包体积优化，通用模型包从 57.7MB 优化到 11.6MB • **问题优化：** 代码重构，解决了一些历史的崩溃和内存泄露问题	SDK 版本号以 1.6.*结尾；需搭配版本号以 V3 结尾的形象包使用。
V1.4	• **问题修复：** – 修复音频播放被外界终止时不能恢复的问题 – 修复静默区间问题	SDK 版本号以 1.4.*结尾；需搭配版本号以 V3 结尾的形象包使用。
V1.3	• **性能提升：** 首帧渲染延迟从1450ms 大幅降低至450ms。 • **体积优化：** 基础包体和模型包体积减少约50%。	SDK 版本号以 1.3.*结尾；需搭配版本号以 V3 结尾的形象包使用。
V1.1	• **稳定性增强：** 显著提升了 SDK 的运行稳定性与兼容性。	SDK 版本号以 1.1.*结尾；需搭配版本号以 V2 结尾的形象包使用。
V1.0	• **基础功能：** 提供数智人驱动与渲染	SDK 版本号以 1.0.*结尾；需搭配版本号以 V1

	的基础能力。	结尾的形象包使用。

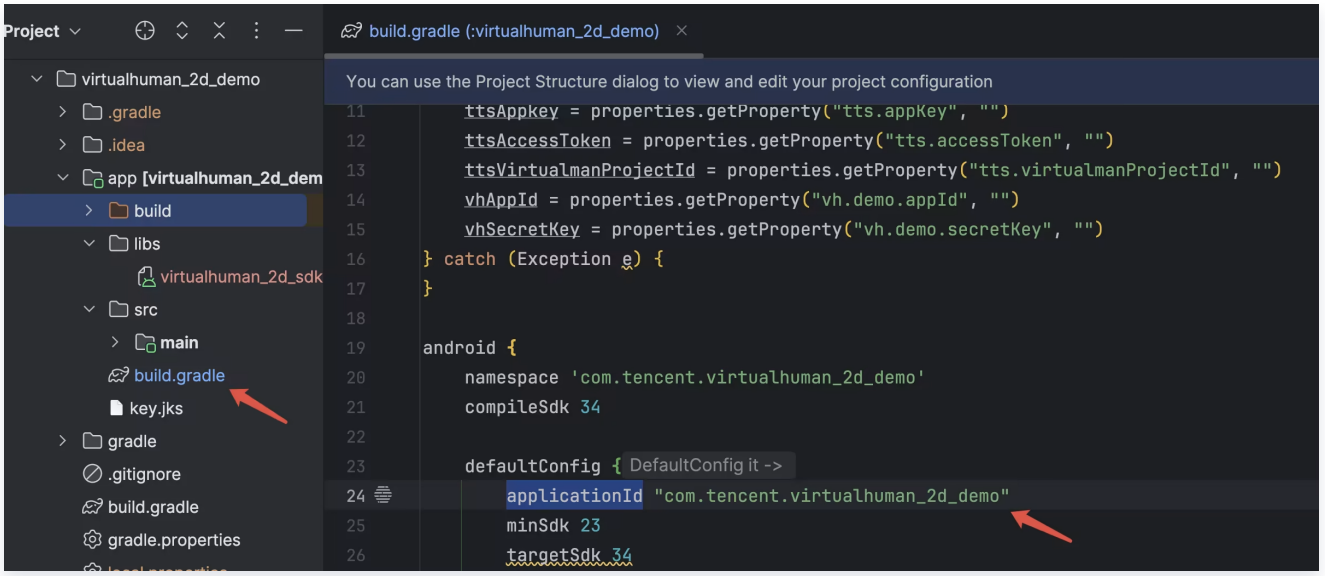
七、常见问题（FAQ）

1. 如何获取 iOS 的 Bundle ID 和 Android 的 Application ID?

- iOS 设备的 Bundle ID，可以在 Xcode 的 App 设置， Signing & Capabilities 中找到，如下图所示：



- Android 设备的 Application ID, 可以在工程目录的 build.gradle 文件中找到，如下图所示：



2. 2D 小样本端渲染是否支持定制形象?

- 支持。目前以下3种定制类型均可用于端渲染 SDK：
  - 2D 小样本通用口型
  - 2D 小样本专属口型
  - 2D 小样本照片
- 定制方式：

- 自助采购和定制 2D 小样本数字人：按 定制资产指引 完成形象定制。
- 定制完成后，可以通过发邮件（txc_avatar@tencent.com）线下获取形象模型包。

3. 如何进行日志管理以协助问题排查?

为便于客户集成与问题排查，数智人端渲染 SDK 提供了灵活的日志输出配置接口。请参考以下指南进行日志管理和问题排查：

- **日志等级说明**

日志等级	iOS 等级标识	安卓等级标识	适用场景	输出频率	存储占用
关闭日志	TVHLogLevelOff	0	不建议使用	无输出	无
错误日志	TVHLogLevelError	1	生产环境重大问题	极低	可忽略
警告日志	TVHLogLevelWarn	2	**推荐生产环境默认级别**	较低	可忽略
信息日志	TVHLogLevelInfo	3	**推荐日常调试级别**	适中	<1MB/天
调试日志	TVHLogLevelDebug	4	开发阶段问题定位	较高	1-5MB/天
追踪日志	TVHLogLevelTrace	5	极端情况问题复现	极高	>100MB/天

- **数智人 SDK 日志配置原则**
  - 初始化前必设：日志等级必须在 SDK 初始化前配置，运行时无法修改。
  - 推荐级别：日常使用建议设置为 WARN 或 INFO 级别，在保证关键日志输出的同时避免输出过多日志。
  - 慎用 TRACE：避免长期开启 Trace 级别，以防存储占用过大和性能下降。
  - 禁止长期关闭：生产环境不建议长期设置为 OFF 级别，否则无法获取关键问题日志。
  - 问题反馈：用户反馈问题时，请提供 INFO 级别日志，以便协助定位问题。
  - 云控调试：特殊情况下，可通过云控临时开启 TRACE 级别日志，复现问题后上报日志文件。
  - 日志收集：请实现日志上传功能，在用户反馈问题时，可自动或引导用户上报日志文件至后台进行分析。

移动端 Android SDK 接入

最近更新时间：2026-04-02 14:45:51

1. 软件和硬件要求

系统	支持 Android 6 以上系统
处理器架构	支持 arm64-v8a CPU 架构
处理器性能要求	支持高通骁龙8+、海思麒麟 980+、联发科 mt6878+ 等
开发 IDE	Android Studio
内存要求	大于 500MB

2. 模型包说明

模型包分为**基础模型包**和**人物形象包**两种，具体介绍如下：

基础模型包

基础模型包与形象无关， 是运行 2D 端渲染的必备条件，模型包目录名称为： `common_model/` 如果有多个人物模型，可以共用一个基础模型包。

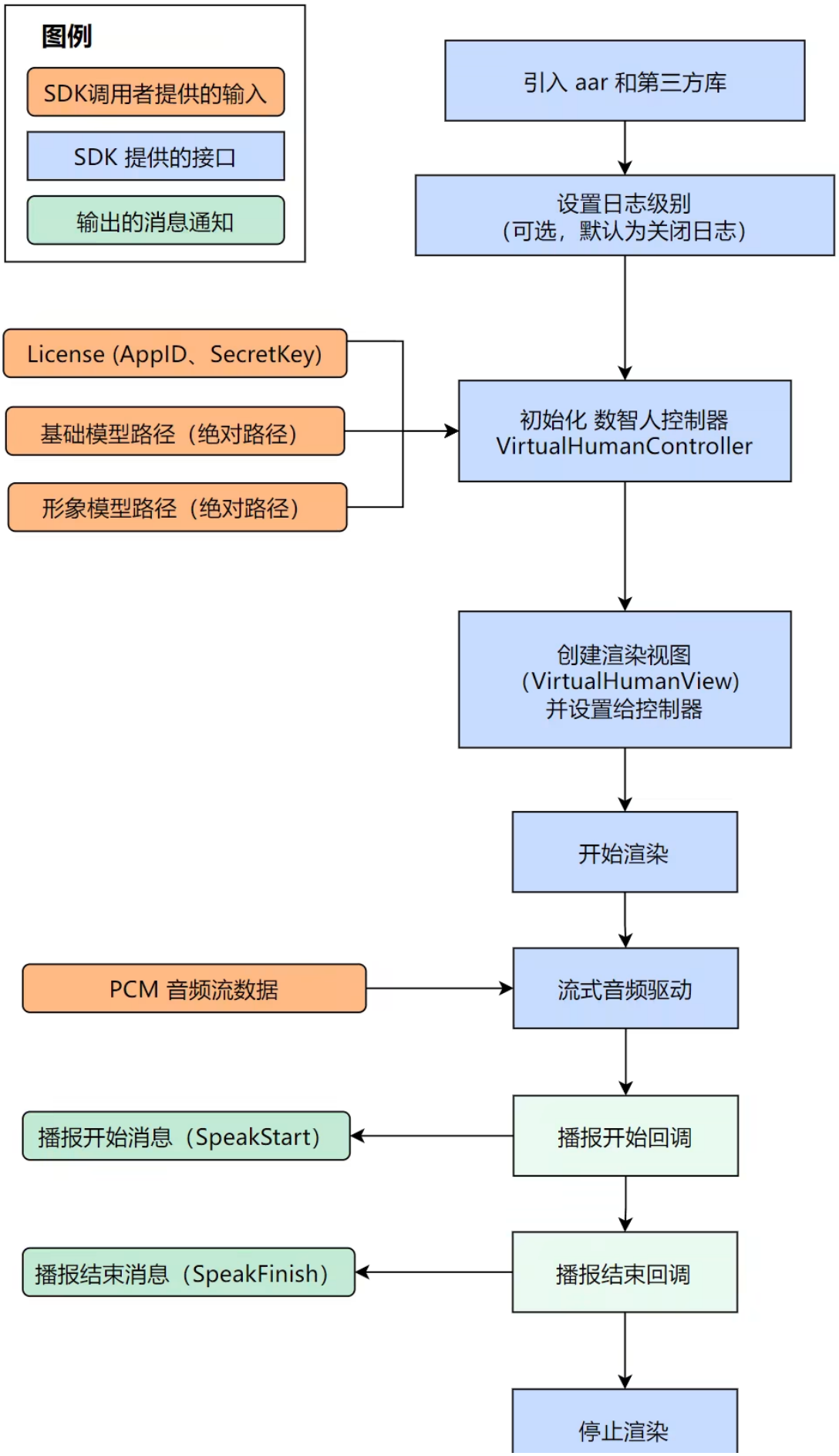
人物形象包

人物形象包里包含了 2D 数智人驱动的**图像数据**和**推理模型**。每个人物形象一个模型包， 加载不同的人物模型包，即可更换形象。模型包目录名称为： `human_xxxxx_540p_v2/` ，其中 xxxxx 为定制形象的名称。
目前 SDK 包里含一个公共形象，包名是：human_bingwen_540p_v3，您可直接使用。

> ⚠ **注意：**
> - SDK 接入方需要将模型包放到 App 的沙盒中，初始化 SDK 时需要将2个模型的绝对路径作为参数传递给 SDK。
> - 模型包可以动态下载，动态下载和更新的逻辑，需要接入方 App 自行实现。

3. 数智人客户端渲染 SDK 接入

数智人 SDK 的调用流程如下：


图例
SDK调用者提供的输入
SDK 提供的接口
输出的消息通知
引入 aar 和第三方库
设置日志级别
（可选，默认为关闭日志）
License (AppID、SecretKey)
基础模型路径（绝对路径）
形象模型路径（绝对路径）
初始化 数智人控制器
VirtualHumanController
创建渲染视图
(VirtualHumanView)
并设置给控制器
开始渲染
PCM 音频流数据
流式音频驱动
播报开始消息（SpeakStart）
播报开始回调
播报结束消息（SpeakFinish）
播报结束回调
停止渲染

3.1 引入 AAR 包和第三方库

将数智人端渲染 SDK 的 AAR 包引导到项目中， AAR 包文件命名为： virtualhuman_2d_sdk-release-202603061641_1.6.3_app.aar
由于使用了 OkHttp 库，所以需要在项目的 build.gradle（Module:app）文件中添加 OkHttp 的依赖。

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

    // 添加依赖
    implementation 'androidx.appcompat:appcompat:1.7.0'
    implementation 'com.squareup.okhttp3:okhttp:4.9.0'
    implementation 'com.parse.bolts:bolts-tasks:1.4.0'
}
```

3.2 配置权限

在 AndroidManifest.xml 中添加：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
// 设备绑定时需要添加此权限
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

3.3 基础使用示例

```
// 1. 创建参数
VirtualHumanParam param = new VirtualHumanParam();
param.appId = "your_app_id";
param.secretKey = "your_secret_key";
param.commonModelPath = "/path/to/common_model";
param.humanModelPath = "/path/to/human_model";

// 2. 设置回调
param.eventCallback = (code, message) -> {
    Log.i("VH", "Event: " + code + ", " + message);
};
```

```
param.errorCallback = (code, message) -> {
    Log.e("VH", "Error: " + code + ", " + message);
};

// 3. 创建控制器
VirtualHumanController controller = new VirtualHumanController(context,
param);

// 4. 初始化
int result = controller.initHuman();
if (result != 0) {
    Log.e("VH", "Init failed: " + result);
    return;
}

// 5. 设置渲染视图
VirtualHumanSurfaceView view = new VirtualHumanSurfaceView(context);
controller.setRenderView(view);

// 6. 启动渲染
controller.startHuman();

// 7. 推送音频数据
byte[] audioData = ...; // PCM 16bit 16kHz
controller.appendAudioData(audioData, false);

// 8. 停止并释放
controller.stop();
```

4. 核心类

VirtualHumanController

数智人渲染控制器，管理 Native 渲染引擎。

```
public VirtualHumanController(Context context, VirtualHumanParam param)
```

参数：

- context: Android Context

- param: 数智人参数配置

initHuman()

初始化数智人引擎。

```
public int initHuman()
```

返回值：

- 0: 成功
- 20001: 模型加载错误
- 20010: License 鉴权失败
- 20011: 参数无效
- 20013: 模型尺寸不支持
- 20014: 缺少 READ_PHONE_STATE 权限（仅设备绑定鉴权场景）

示例：

```
int result = controller.initHuman();
if (result == 0) {
    Log.i("VH", "初始化成功");
} else if (result == 20010) {
    Log.e("VH", "License 失效");
}
```

setRenderView()

设置渲染视图。

```
public void setRenderView(IVirtualHumanRenderView view)
```

参数：

- view: 实现 IVirtualHumanRenderView 接口的视图（VirtualHumanSurfaceView 或 VirtualHumanTextureView）

示例：

```
VirtualHumanSurfaceView view = new VirtualHumanSurfaceView(context);
```

```
controller.setRenderView(view);
```

startHuman()

启动数智人渲染。

```
public void startHuman()
```

说明：

- 必须在 initHuman() 成功后调用

appendAudioData()

推送音频数据驱动数智人播放。

```
public void appendAudioData(byte[] audioData, boolean isEnd)
public void appendAudioData(byte[] audioData, boolean isEnd, String
metaData)
```

参数：

- audioData: 音频数据（PCM 16bit 16kHz 单声道）
- isEnd: 是否为最后一帧
- metaData: 元数据（可选，用于同步自定义信息）

音频格式要求：

- 采样率：16000Hz
- 位深度：16bit
- 声道：单声道（Mono）
- 编码：PCM

```
// 流式推送
controller.appendAudioData(chunk1, false, "metadata_1");
controller.appendAudioData(chunk2, false, "metadata_2");
controller.appendAudioData(chunk3, true, "metadata_final");

// 一次性推送
byte[] audioData = loadAudioFile();
controller.appendAudioData(audioData, true);
```

appendAction()

插入动作。

> **说明：**
> - sdkVersion 1.7.0+ 支持。
> - 仅精品模式下生效。
> - 动作执行结果通过事件回调通知（10006开始、10007完成、10008失败）。

```
public void appendAction(String actionCode, long timestampMs)
```

参数：

- actionCode: 动作标识，对应模型中的 action_code（可通过 getAppendableActionList() 查询可用列表）。
- timestampMs: 目标时间戳（毫秒），该动作执行到一半时的时间点。

示例：

```
// 查询可用动作列表后，在指定时刻插入动作
String actionList = controller.getAppendableActionList();
// 如果想当前片段播完后立即执行该动作，可以加上该动作时长的一半时间。
long targetTs = controller.getRunningDurationMs() + 2500;
// 动作标识在actionList列表可看到
controller.appendAction("heart", targetTs);
```

getAppendableActionList()

获取当前模型支持的可插入动作列表。

> **说明：**
> - sdkVersion 1.7.0+ 支持。
> - 仅精品模式下生效。

```
public String getAppendableActionList()
```

返回值：

- JSON 字符串，格式：[{"action_code":"heart","duration_ms":2400}, ...]
- 引擎未初始化时返回 "[]"

示例：

```
String json = controller.getAppendableActionList();
// 解析示例
JSONArray actions = new JSONArray(json);
for (int i = 0; i < actions.length(); i++) {
    JSONObject action = actions.getJSONObject(i);
    String code = action.getString("action_code");     // 动作标识
    int durationMs = action.getInt("duration_ms");     // 动作时长（毫秒）
    Log.i("VH", "动作: " + code + ", 时长: " + durationMs + "ms");
}
```

getRunningDurationMs()

获取数智人运行时长。

> **说明：**
> sdkVersion 1.7.0+ 支持。

```
public long getRunningDurationMs()
```

返回值：

- 基于帧流已输出帧数计算的运行时长（每帧固定 40ms），与内部帧时间戳保持一致
- 与系统时钟无关，不受实际渲染耗时影响
- 未启动时返回 0

示例：

```
long durationMs = controller.getRunningDurationMs();
Log.i("VH", "已运行: " + durationMs + "ms");
```

interrupt()

打断当前播放。

```
public void interrupt()
```

说明：

- 立即停止当前播放内容
- 清空音频缓冲区

pause() / resume()

暂停/恢复渲染。

```
public void pause()
public void resume()
```

说明：

- pause(): 暂停渲染，但保持引擎运行状态
- resume(): 恢复渲染

setMuted()

设置静音状态。

```
public void setMuted(boolean muted)
```

参数：

- muted: true 为静音，false 为取消静音

setFillMode()

设置渲染填充模式。

```
public void setFillMode(VirtualHumanViewType fillMode)
```

参数：

- fillMode: 填充模式枚举
  - VirtualHumanViewType.fill: 保持比例，留黑边

- VirtualHumanViewType.fit: 保持比例，裁切超出
- VirtualHumanViewType.stretch: 拉伸填充，可能变形

stop()

停止并释放所有资源。

```
public void stop()
```

说明：

- 停止渲染
- 释放 Native 资源
- 关闭线程池
- 调用后需要重新 initHuman() 才能使用

getFillMode()

获取当前渲染填充模式。

```
public VirtualHumanViewType getFillMode()
```

返回值：

- 当前 VirtualHumanViewType 枚举值

getRenderView()

获取当前绑定的渲染 View。

```
public IVirtualHumanRenderView getRenderView()
```

返回值：

- 当前渲染 View，未设置时返回 null

setLogLevel()

设置日志级别。

```
public static void setLogLevel(int level)
```

参数：

- VirtualHumanController.LOG_LEVEL_OFF: 关闭日志
- VirtualHumanController.LOG_LEVEL_ERROR: 错误
- VirtualHumanController.LOG_LEVEL_WARN: 警告
- VirtualHumanController.LOG_LEVEL_INFO: 信息
- VirtualHumanController.LOG_LEVEL_DEBUG: 调试
- VirtualHumanController.LOG_LEVEL_TRACE: 追踪

示例：

```
VirtualHumanController.setLogLevel(VirtualHumanController.LOG_LEVEL_INFO
);
```

setLogCallback()

设置日志回调。

```
public static void setLogCallback(IDataCallback logCallback)
```

参数：

- logCallback: 日志回调接口

示例：

```
VirtualHumanController.setLogCallback((level, message) -> {
    Log.i("VH", "Log[" + level + "]: " + message);
});
```

getVersion()

获取 SDK 版本。

```
public static String getVersion()
```

返回值：

- SDK 版本字符串

runPerformanceTest()

运行性能检测。

> ⚠ **注意：**
> 该方法基于设备实时硬件配置执行基准测试，测试结果由当前设备的 SOC型号、负载状态及运行环境共同决定，因此同一设备在不同时刻可能存在差异。

```
public static void runPerformanceTest(Context context, VirtualHumanParam
param, IDataCallback callback)
public static void runPerformanceTest(Context context, VirtualHumanParam
param, IDataCallback callback, boolean forceRefresh)
```

参数：

- context: Android Context
- param: 数智人参数配置
- callback: 检测结果回调
- forceRefresh: 是否强制刷新（忽略缓存）

回调参数：

- code:
- 0: 性能达标
- 1: 性能不足
- 负数: 检测失败
- message: 详细信息

说明：

- 使用内置测试音频检测设备性能
- 会自动初始化、测试、销毁，无需手动管理生命周期
- 支持缓存机制，默认使用缓存结果（除非 forceRefresh=true）

示例：

```
VirtualHumanController.runPerformanceTest(context, param, (code,
message) -> {
    if (code == 0) {
        Log.i("VH", "性能达标");
    } else if (code == 1) {
```

```
            Log.w("VH", "性能不足");
        } else {
            Log.e("VH", "检测失败: " + message);
        }
});

// 强制刷新，忽略缓存
VirtualHumanController.runPerformanceTest(context, param, callback,
true);
```

checkDeviceWhitelist()

检查设备黑白名单。

> ⚠ **注意：**
> 由于 Android 设备碎片化程度较高，黑白名单结果仅供参考，我们将持续维护并定期更新数据，以确保其准确性与可靠性。

```
public static void checkDeviceWhitelist(Context context,
VirtualHumanParam param, IDataCallback callback)
public static void checkDeviceWhitelist(Context context,
VirtualHumanParam param, IDataCallback callback, boolean forceRefresh)
```

参数：

- context: Android Context
- param: 数智人参数配置
- callback: 检测结果回调
- forceRefresh: 是否强制刷新（忽略缓存）

回调参数：

- code:
- 0: 白名单（性能达标）
- 1: 黑名单（性能不足）
- 2: 未知设备（不在黑白名单中）
- -1: SOC 获取失败
- -2: 网络请求失败或初始化失败
- message: 详细信息

说明：

- 根据设备 SOC 和模型参数判断设备是否在黑白名单中
- 会请求远程配置并自动初始化引擎
- 支持内存缓存机制（进程级别），默认使用缓存（除非 forceRefresh=true）

示例：

```
VirtualHumanController.checkDeviceWhitelist(context, param, (code,
message) -> {
    switch (code) {
        case 0:
            Log.i("VH", "白名单：设备性能达标");
            break;
        case 1:
            Log.w("VH", "黑名单：设备性能不足");
            break;
        case 2:
            Log.i("VH", "未知设备：不在黑白名单中");
            break;
        case -1:
            Log.e("VH", "SOC 获取失败");
            break;
        case -2:
            Log.e("VH", "检测失败： " + message);
            break;
    }
});

// 强制刷新，忽略缓存
VirtualHumanController.checkDeviceWhitelist(context, param, callback,
true);
```

VirtualHumanParam

数智人参数配置类。

字段

```
// ===== 必填字段 =====

// License 鉴权
public String appId;                // 应用 ID
public String secretKey;            // 密钥

// 模型路径
public String commonModelPath;      // 通用模型路径
public String humanModelPath;       // 人物模型路径

// ===== 可选字段 =====

// 设备标识（仅设备绑定授权场景需要）
public String deviceName;           // 设备唯一标识，必须保持固定不变

// 性能优化
public int skipFrameInterval = 7; // 跳帧间隔（0=不跳帧，7=每8帧跳1帧）
public boolean dualGen = false;     // 是否启用双线程推理

/*
 * 数字人在静默时无额外手部动作，交互更真实自然。
 * 注：如果形象本身不支持该特性，则该配置项不生效。
 * 默认值为true
 */
public boolean enableSilenceList = true;

// 回调
public IDataCallback eventCallback;   // 事件回调
public IDataCallback errorCallback;   // 错误回调
```

示例

```
VirtualHumanParam param = new VirtualHumanParam();

// 必填
param.appId = "your_app_id";
param.secretKey = "your_secret_key";
param.commonModelPath = "/sdcard/models/common_model";
param.humanModelPath = "/sdcard/models/human_001";
```

```
// 可选：设备标识（仅设备绑定授权场景需要）
// 重要：同一台设备必须始终使用相同的 deviceName，否则会导致鉴权失败
param.deviceName = "SN20240101001";  // 使用设备序列号（推荐）
// 或使用其他固定标识：
// param.deviceName = "ASSET-A-001";       // 资产编号
// param.deviceName = "00:11:22:33:44:55"; // MAC 地址

// 可选：性能优化
param.skipFrameInterval = 7;  // 跳帧优化
param.dualGen = true;         // 双线程加速

// 可选：回调
param.eventCallback = (code, message) -> {
    Log.i("VH", "Event: " + code);
};
param.errorCallback = (code, message) -> {
    Log.e("VH", "Error: " + code + ", " + message);
};
```

说明：

- APP 绑定授权（bindDevice=false）：deviceName 可以为空或不设置
- 设备绑定授权（bindDevice=true）：
  - deviceName 必须设置，用于设备唯一性标识
  - 同一台物理设备必须始终使用相同的 deviceName
  - 如果 deviceName 变化，服务器会认为是不同设备，导致鉴权失败或消耗额外授权额度
  - 建议使用设备出厂铭牌上的序列号、资产编号等固定不变的标识
  - 不要使用随机值或每次变化的值

VirtualHumanSurfaceView

基于 SurfaceView 的渲染视图。

特点

- ✓ 性能更好，适合全屏渲染
- ✓ 独立的 Surface Layer
- ✗ UI 控件叠加需要额外处理（setZOrderOnTop 后普通 View 会被遮挡）

使用示例

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.tencent.virtualhuman_2d_sdk.VirtualHumanSurfaceView
        android:id="@+id/vh_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

```
VirtualHumanSurfaceView view = findViewById(R.id.vh_view);
controller.setRenderView(view);
```

VirtualHumanTextureView

基于 TextureView 的渲染视图。

特点

- ✅ 支持透明背景
- ✅ UI 控件可以正常叠加显示
- ✅ 支持动画和变换
- ❌ 性能稍逊于 SurfaceView

使用示例

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- 背景层 -->
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/background" />

    <!-- 渲染层 -->
    <com.tencent.virtualhuman_2d_sdk.VirtualHumanTextureView
```

```
        android:id="@+id/vh_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- 叠加控件层 -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="操作按钮" />

</FrameLayout>
```

```
VirtualHumanTextureView view = findViewById(R.id.vh_view);
controller.setRenderView(view);
```

VirtualHumanViewType

渲染填充模式枚举。

枚举值

```
public enum VirtualHumanViewType {
    fill,       // 保持比例，留黑边（默认）
    fit,        // 保持比例，裁切超出
    stretch     // 拉伸填充，可能变形
}
```

示例

```
// 保持比例，留黑边
controller.setFillMode(VirtualHumanViewType.fill);

// 保持比例，裁切超出
controller.setFillMode(VirtualHumanViewType.fit);

// 拉伸填充
controller.setFillMode(VirtualHumanViewType.stretch);
```

5. 回调接口

IDataCallback

通用数据回调接口。

```
public interface IDataCallback {
    void onDataCallback(int code, String message);
}
```

用途

1. 事件回调（param.eventCallback）
2. 错误回调（param.errorCallback）
3. 日志回调（setLogCallback）
4. 性能检测回调（runPerformanceTest）
5. 黑白名单检测回调（checkDeviceWhitelist）

6. 回调码

错误码

错误码	常量	说明
20001	ERROR_INIT_MODEL	模型加载失败
20010	ERROR_LICENSE_INVALID	License 鉴权失败
20011	ERROR_INIT_PARAM_INVALID	初始化参数无效
20012	ERROR_PERFORMANCE	设备性能不足
20013	ERROR_MODEL_SIZE_NOT_SUPPORTED	模型尺寸不支持（小屏版 SDK 不支持高尺寸模型）
20014	ERROR_PERMISSION_DENIED	缺少必要权限（设备绑定鉴权需要 READ_PHONE_STATE）

事件码

事件码	说明
10001	播放开始
10002	播放结束

10003	元数据开始
10004	元数据结束
10005	首帧渲染完成
50001	性能统计信息（包含平均耗时、最大耗时等）

移动端 iOS SDK 接入

最近更新时间：2025-09-17 18:34:21

1. 软件和硬件要求

系统	iOS 15.6 及以上系统
处理器架构	arm64 CPU 架构
支持终端设备	• A12+ 处理器，M1+ 处理器 • iPhone XS，iPhone XR，iPhone SE2，iPhone 11+ 及更新设备 • iPad 8+，iPad Mini5+，iPad Air3+，iPad Pro3+，New iPad Pro1+ 及更新设备 （2018 年以后发布的 iPhone，2019 年以后发布的 iPad 设备）
开发 IDE	XCode
内存要求	大于500M

2. 模型包说明

模型包分为两种：

基础模型包

基础模型包与形象无关， 是运行 2D 端渲染必须的，模型包目录名称为： `common_model/` 如果有多个人物模型，可以共用一个基础模型包。

人物形象包

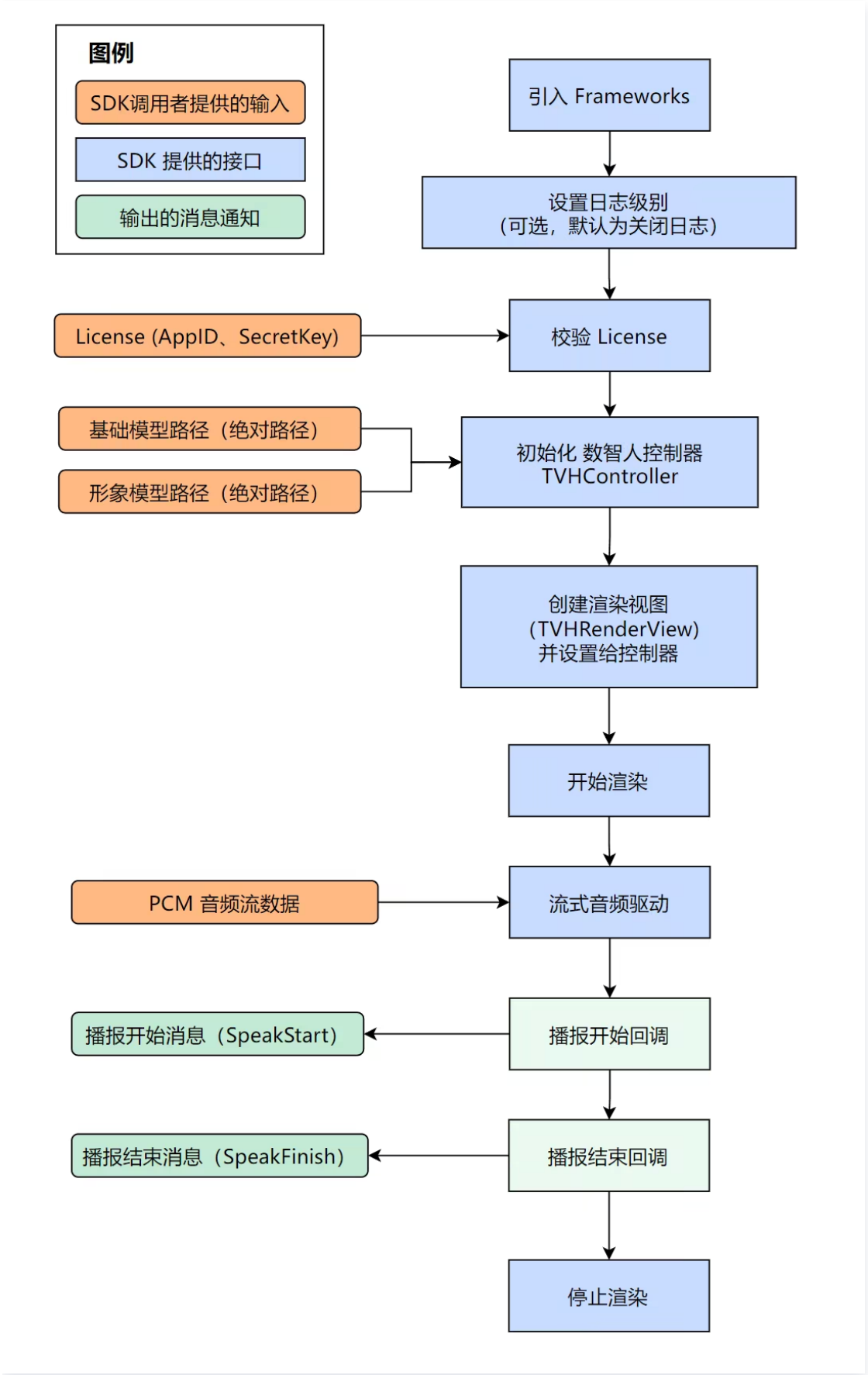
- 人物形象包里包含了 2D 数智人驱动的图像数据和推理模型。每个人物形象一个模型包， 加载不同的人物模型包，即可更换形象。模型包目录名称为： `human_xxxxx_540p_v3/` ，其中 xxxxx 为定制形象的名称。
- 人物形象包支持 540p (960 x 540) 和 720p (1280 x 720) 分辨率。

目前 SDK 包里含一个公共形象，包名是：human_yunxi_540p_v3，您可直接使用。

> ⚠ **注意：**
> - SDK 接入方需要将模型包放到 App 的沙盒中，初始化 SDK 时需要将2个模型的绝对路径作为参数传递给 SDK。
> - 模型包可以动态下载，动态下载和更新的逻辑，需要接入方 App 自行实现。

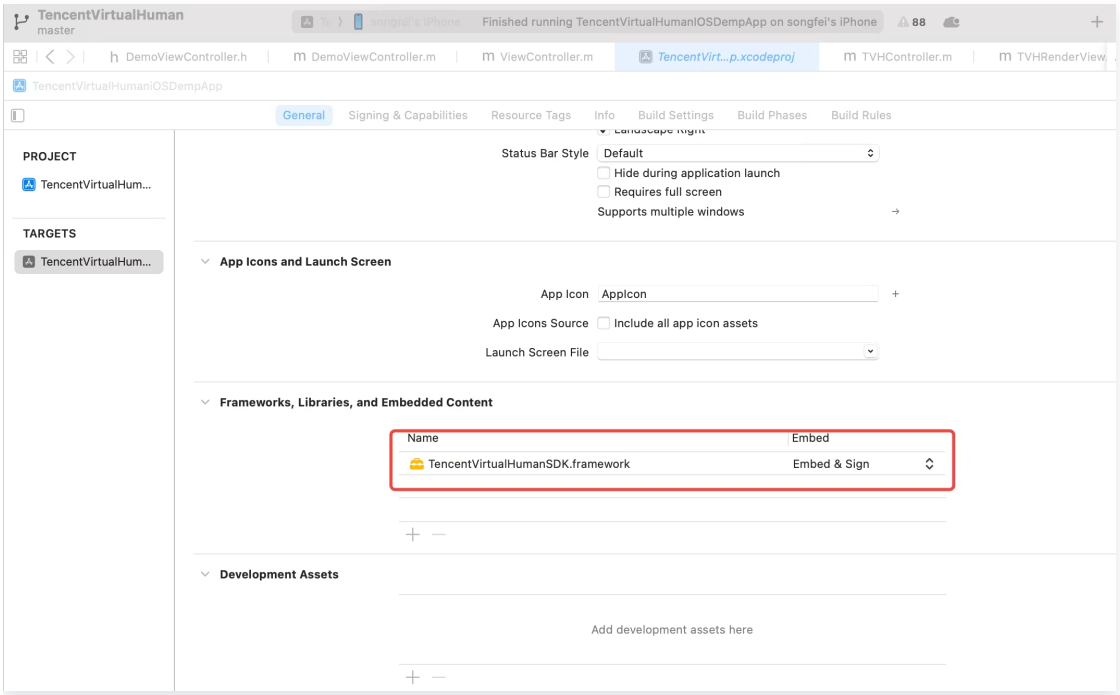
3. 数智人客户端渲染 SDK 接入

数智人 SDK 的调用流程如下：



3.1 引入 Framework

将 Framework 添加到工程中， 嵌入方式选择： Embed & Sign 。



并引入头文件：
在 Objective-C 中使用时，直接引入头文件， 在 Swift 中使用时， 需要在 xxxxx-Bridging-Header.h 文件中引入头文件。

```
#import <TencentVirtualHumanSDK/TencentVirtualHumanSDK.h>
```

3.2 设置日志级别（可选）

SDK 日志默认输出到标准输出 stdio 中， 可设置日志输出等级。 默认等级为 `TVHLogLevelOff`

> ⚠ **注意：**
> 日志等级仅可以设置一次， 多次调用仅第一次调用生效。

Objective-C

```
[[TVHLogManager shareInstance] setLogLevel:TVHLogLevelInfo];
```

Swift

```
TVHLogManager.shareInstance().setLogLevel(TVHLogLevelInfo)
```

日志输出等级：

日志等级	等级标识
关闭日志	TVHLogLevelOff
错误日志	TVHLogLevelError
警告日志	TVHLogLevelWarn
信息日志	TVHLogLevelInfo
调试日志	TVHLogLevelDebug
追踪日志	TVHLogLevelTrace

3.3 License 校验

在使用数智人 SDK 前，先调用授权方法，否则其他所有类的初始化方法都会失败。
鉴权可以在 App 启动时，也可以在第一次打开数智人界面时。鉴权模块为单例，可以调用多次，以最后一次鉴权结果为准。

Objective-C

```
int result = [[TVHLicenseManager shareInstance]
authWithAppID:@"0000000000"
andSecretKey:@"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"];
NSLog(@"license result: %d", result);
```

Swift

```
let result = TVHLicenseManager.shareInstance().auth(withAppID:
"0000000000", andSecretKey: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
print("license result: \(result)")
```

返回 0 则表示鉴权成功，返回其他值为鉴权失败。

3.4 创建数智人控制器（TVHController）

通过控制器，可以初始化数智人，控制数智人开始和停止渲染， 控制数智人张嘴说话。

Objective-C

```
// 请事先将通用模型和形象模型放入沙盒 Documents 目录下
NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *documentsPath = [paths firstObject];
NSString* common_model = [NSString stringWithFormat:@"%@/common_model",
documentsPath];
NSString* human_model = [NSString
stringWithFormat:@"%@/human_yunxi_540p_v3", documentsPath];

// 初始化控制器
self.controller = [[TVHController alloc]
initWithCommonModelPath:common_model humanModelPath:human_model];
self.controller.delegate = self;

// 开始渲染
[self.controller start];
```

Swift

```
// 请事先将通用模型和形象模型放入沙盒 Documents 目录下
let paths = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true)
let documentsPath = paths.first!
let commonModel = "\(documentsPath)/common_model"
let humanModel = "\(documentsPath)/human_youyou3_720p"

// 初始化控制器
controller = TVHController(commonModelPath: commonModel, humanModelPath:
humanModel)
if(controller != nil) {
    controller.delegate = self;

    // 开始渲染
    controller.start()
}
```

3.5 创建渲染视图（TVHRenderView）

数智人 SDK 使用 Metal 实现了高性能、背景透明的渲染界面，数智人的图像画面会被绘制到此 `RenderView` 上，您可以和使用普通 View 一样设置此 `RenderView` 的大小位置等，可以将此 `RenderView` 作为子 View 添加到其他 View。

> **说明：**
> 如果接入方有特殊需求，可以自行实现渲染 View， 需要自行实现 `TVHRenderViewDelegate` 来获取 RGBA 原始数据进行绘制。
> **我们强烈建议您使用 SDK 内封装的 `TVHRenderView` 以获得最好的渲染效果**

Objective-C

```
self.renderView = [[TVHRenderView alloc]
initWithFrame:self.view.bounds];
self.renderView.fillMode =TVHMetalViewContentModeFit;

[self.view addSubview:self.renderView];

// 将 renderView 设置给控制器
self.controller.renderView = self.renderView;
```

Swift

```
renderView = TVHRenderView(frame: self.view.bounds)
renderView.fillMode = .fit

view.addSubview(renderView)

// 将 renderView 设置给控制器
controller.renderView = renderView
```

可以设置渲染 View 的填充模式：

TVHMetalViewContentModeStretch	拉伸图像
TVHMetalViewContentModeFit	裁切图像
TVHMetalViewContentModeFill	保留黑边（透明边）

3.6 流式输入音频驱动数智人

流式输入音频，让数智人开口说话。
输入的音频格式为 s16le (signed 16 bits little endian, 有符号16位小端) PCM， 采样率16000， 单声道。
可以多次调用 `appendAudioData` 流式输入音频数据，输入的音频会被保存到音频缓冲队列中，每次调用时输入的音频数据时长可以是任意值。

> ⚠ **注意：**
> - 输入数据的实时率必须大于 1.0， 要保证输入数据的数量要比实时播放音频消耗的数据多， 如果音频数据不足可能导致等待音频数据，数智人播报过程中卡住不动。
> - 输入音频最后的一片数据需要设置 isFinal 为 YES，否则 SDK 会认为音频没有结束，等待音频导致数智人播报时卡住不动。

Objective-C

```
// 此代码片段演示了从文件读取 PCM 数据，模拟流式分段发送给控制器

NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *documentsPath = [paths firstObject];
NSString* pcmPath = [NSString stringWithFormat:@"%@/test.pcm",
documentsPath];

// 读取音频数据
NSData* data = [NSData dataWithContentsOfFile:pcmPath];

// 模拟流式，分片发送数据（仅供演示，丢掉了最后一个分片）
int packageSize = 1280;
int packageCount =(int)[data length] / packageSize;
for(int i=0; i<packageCount; i++) {
    BOOL isFinal = (i == packageCount -1);
    [self.controller appendAudioData:[data
subdataWithRange:NSMakeRange(i*packageSize, packageSize)] metaData:@""
isFinal:isFinal];
}
```

Swift

```
// 此代码片段演示了从文件读取 PCM 数据，模拟流式分段发送给控制器

let paths = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true)
guard let documentsPath = paths.first else { return }
let pcmPath = "\(documentsPath)/test.pcm"

// 读取音频数据
guard let data = try? Data(contentsOf: URL(fileURLWithPath: pcmPath))
else { return }

// 模拟流式，分片发送数据（仅供演示，丢掉了最后一个分片）
let packageSize = 1280
let packageCount = data.count / packageSize
for i in 0..<packageCount {
    let range = i * packageSize..<(i * packageSize + packageSize)
    let subData = data.subdata(in: range)
    let isFinal = (i == packageCount - 1)
    controller.appendAudioData(subData, metaData: "", isFinal: isFinal)
}
```

3.7 打断播报

在数智人播报过程中，可以随时打断播报进入静默状态， 调用控制器的 `interrupt` 方法，数智人就会立刻闭嘴并停止音频播放。

Objective-C

```
[self.controller interrupt]
```

Swift

```
controller.interrupt()
```

> ⚠ **注意：**
> 打断后，也会收到 `speakFinish` 消息。

> 打断调用后数智人会立刻闭嘴并停止音频播放，但是需要等1 – 2秒的时间，才会收到 `speakFinish` 消息， 在收到 `speakFinish` 消息之前，调用 `appendAudioData` 时，也会等到 `speakFinish` 后才会继续开始播报。

3.8 处理 App 切换前后台时暂停和恢复数智人

对于大部分 App 来说，当应用被切换到后台时，数智人的渲染和播报应暂停，待应用重新被激活回到前台后，渲染和播报应自动恢复继续运行。
控制器提供了 `pause` 和 `resume` 两个方法，用来控制数智人暂停和恢复。

在 iOS 平台，可以使用 `NSNotificationCenter` 注册监听器，获得应用进入后台和进入前台的消息。

Objective-C

```
@implementation DemoViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //  ... 其他代码

    // 注册通知
    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(appDidEnterBackground:)

name:UIApplicationDidEnterBackgroundNotification
                                              object:nil];

    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(appWillEnterForeground:)

name:UIApplicationWillEnterForegroundNotification
                                              object:nil];
}

- (void)dealloc{
    // 在销毁时取消委托
    [[NSNotificationCenter defaultCenter] removeObserver:self];
```

```
}

- (void)appDidEnterBackground:(NSNotification *)notification {
    // 处理暂停
    [self.controller pause];
}

- (void)appWillEnterForeground:(NSNotification *)notification {
    // 处理恢复
    [self.controller resume];
}

@end
```

Swift

```
class DemoViewController2: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        //  ... 其他代码

        // 注册通知
        NotificationCenter.default.addObserver(
            self,
            selector: #selector(appDidEnterBackground(_:)),
            name: UIApplication.didEnterBackgroundNotification,
            object: nil
        )

        NotificationCenter.default.addObserver(
            self,
            selector: #selector(appWillEnterForeground(_:)),
            name: UIApplication.willEnterForegroundNotification,
            object: nil
        )
    }
```

```
    deinit {
        // 在销毁时取消委托
        NotificationCenter.default.removeObserver(self)
    }


    @objc private func appDidEnterBackground(_ notification:
Notification) {
        // 处理暂停
        controller.pause()
    }

    @objc private func appWillEnterForeground(_ notification:
Notification) {
        // 处理恢复
        controller.resume()
    }
}
```

3.9 处理数智人的通知消息

数智人渲染通知消息，通过委托（Delegate）模式实现。实现 `TVHControllerDelegate` 委托，在控制器中注册委托对象，就可以收到消息通知。

Objective-C

```
// 实现 TVHControllerDelegate
@interface DemoViewController () <TVHControllerDelegate>
// ... 其他代码
@end


@implementation DemoViewController

// 渲染开始
- (void)renderStart {
    NSLog(@"render start");
}
```

```objc
// 播报开始
- (void)speakStart {
    NSLog(@"speak start");
}

// 播报完成
- (void)speakFinish {
    NSLog(@"speak finish");
}

// 播报错误
- (void)speakError:(NSInteger)errorCode message:(NSString*)message {
    NSLog(@"speak error code:%ld, message:%@", errorCode, message);
}
@end
```

Swift

```swift
// 实现 TVHControllerDelegate
extension DemoViewController2 : TVHControllerDelegate {
    // 渲染开始
    nonisolated func renderStart() {
        print("render start");
    }

    // 播报开始
    nonisolated func speakStart() {
        print("speak start");
    }

    // 播报完成
    nonisolated func speakFinish() {
        print("speak finish");
    }

    // 播报错误
    nonisolated func speakError(_ errorCode: Int, message: String!) {
        print("speak error code:\(errorCode), message:\(message)");
    }
```

```
}
```

3.10 处理 Meta 信息

给数智人输入音频后，有时需要获取播放到某个特定时刻的时机来处理一些逻辑。 比较常见的应用场景是在数智人播报的时候，同时显示字幕。
数智人 SDK 引入了一个 MetaData（附加信息）的概念，在输入音频片段的时候，可以附加一个字符串到音频片段的开头，当数智人驱动到该音频片段时，就会通过 delegate 的方式通知调用者。
如果需要有格式的信息，可以将 JSON 序列化为字符串传入。

输入音频时携带 MetaData

Objective-C

```
[self.controller appendAudioData:data metaData:@"这里是附加信息"
isFinal:isFinal];
```

Swift

```
controller.appendAudioData(data, metaData: "这里是附加信息", isFinal:
isFinal)
```

获取通知

Objective-C

```
// 实现 TVHControllerDelegate
@interface DemoViewController () <TVHControllerDelegate>
// ... 其他代码
@end


@implementation DemoViewController

// 播报 Meta信息 开始
- (void)speakMetaStart:(NSString *)metaData {
    NSLog(@"speak meta start: %@", metaData);
}
```

```
// 播报 Meta信息 完成
- (void)speakMetaFinish:(NSString *)metaData {
    NSLog(@"speak meta finish: %@", metaData);
}

@end
```

Swift

```
// 实现 TVHControllerDelegate
extension DemoViewController2 : TVHControllerDelegate {
    // 播报 Meta信息 开始
    nonisolated func speakMetaStart(_ metaData: String!) {
        print("speak meta start: \(metaData)");
    }

    // 播报 Meta信息 完成
    nonisolated func speakMetaFinish(_ metaData: String!) {
        print("speak meta finish: \(metaData)");
    }
}
```

3.11 针对早期设备的降级方案

端渲染数智人 SDK 是使用客户端的计算能力，完成 AI 算法推理和图像合成，对移动设备的性能有一定的要求。我们支持 2018 年以后发布的 iPhone 和 2019 年以后发布的 iPad 设备，对于较早期的设备需要进行降级，提示用户不支持或者用其他产品方案代替。
可以使用 `canSmoothlyRun` 方法判断当前设备是否支持端渲染 SDK。

Objective-C

```
BOOL canRun = [[TVHDeviceManager shareInstance] canSmoothlyRun]
```

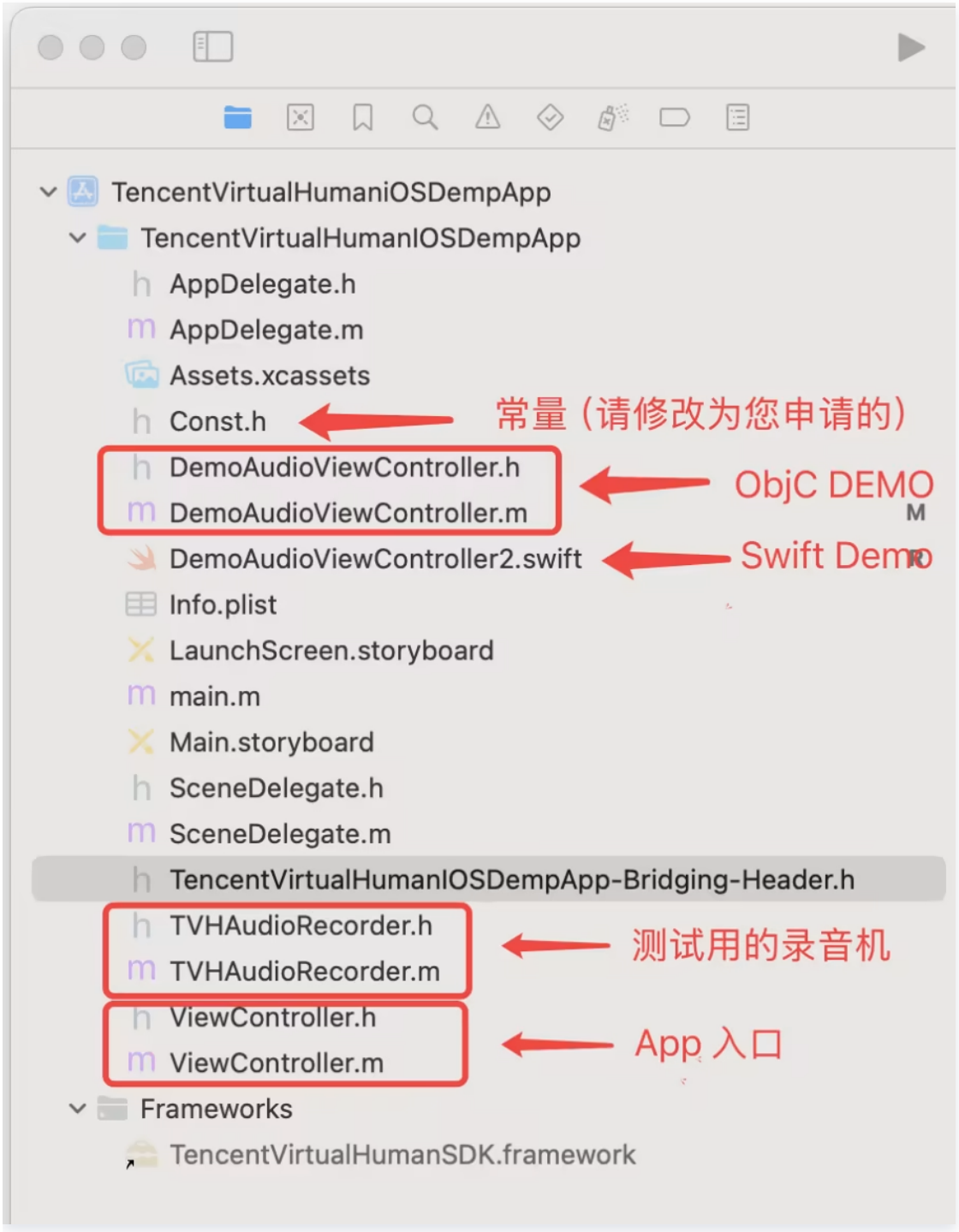
Swift

```
let canRun = TVHDeviceManager.shareInstance().canSmoothlyRun()
```

> ⚠ **注意：**
>
> 1. 此API 是基于 SDK 发布前的性能测试给出的建议结果，不是实际运行的测试。 实际运行结果会受客户程序抢占 CPU 的影响导致不准确。
> 2. 结果偏保守，返回 NO 的部分机型，可能也可以流畅运行。

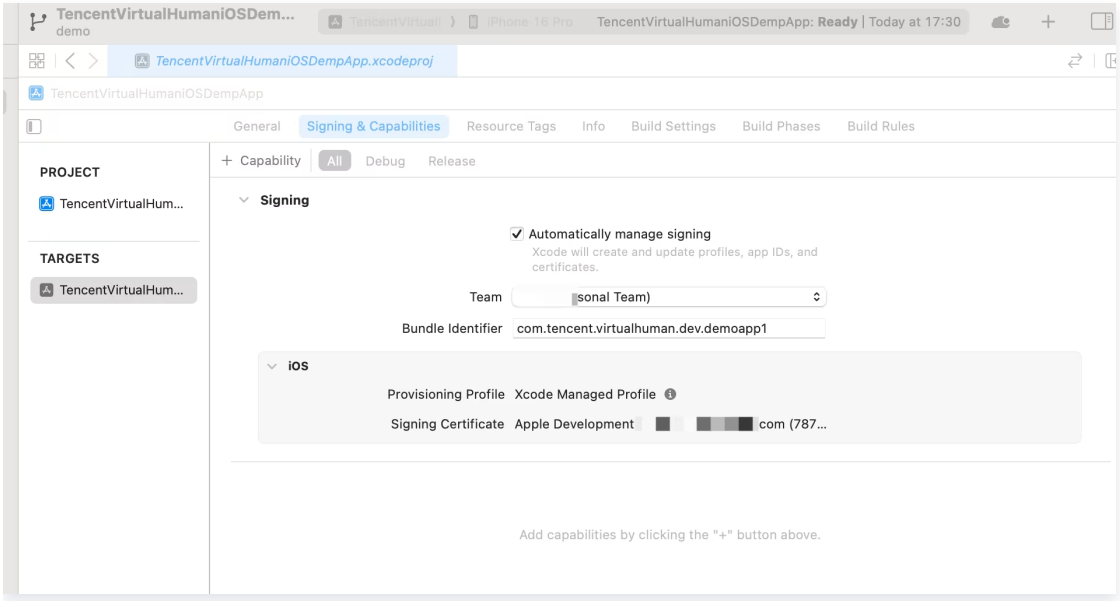
4. Demo App 运行说明

随 SDK 附带了一个能完整运行的 Demo App。 项目工程结构如下：



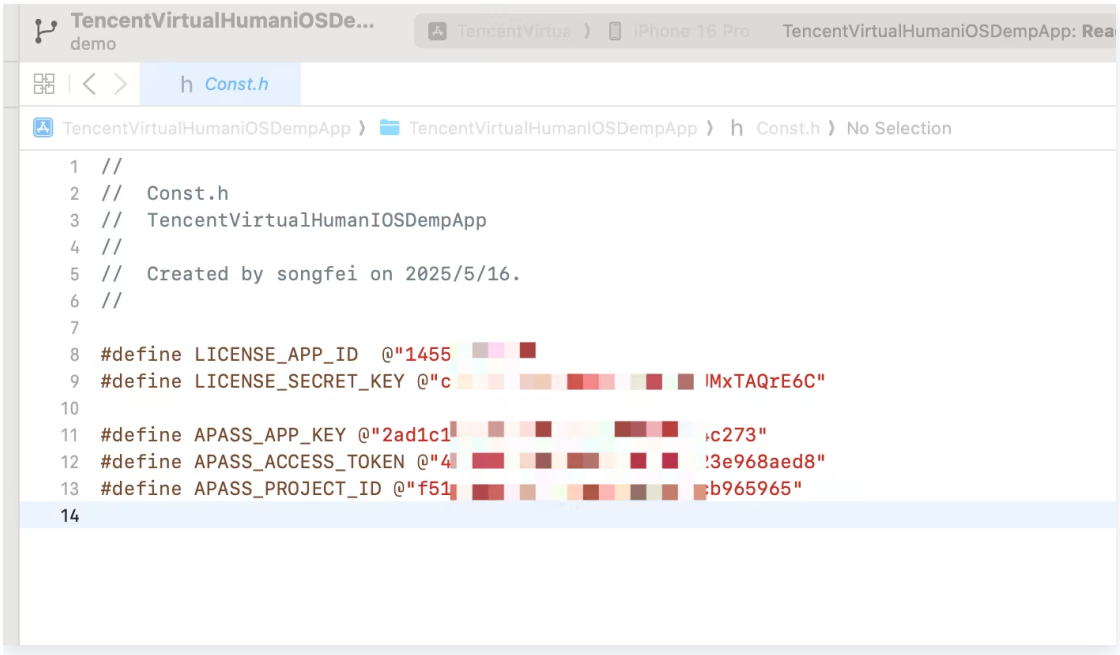
4.1 修改 Bundle ID

请修改 Bundle ID 保证和申请 License 提供的 Bundle ID 一致， 并更新设置证书和签名。



4.2 修改 Const.h 文件

Const.h 文件包含了所有用到的 AppKey, Secret 等密钥， 请修改为客户申请的。



4.3 拷贝模型到沙盒

先运行一次 App，然后在 Mac 电脑的 Finder 应用中，拷贝模型到手机。

4.4 运行 Demo

单击运行按钮，即可运行 2D 端渲染 Demo 应用。

大屏端 Android SDK 接入

最近更新时间：2026-04-02 14:45:51

1. 软件和硬件要求

系统	支持 Android6 以上系统
处理器架构	支持 arm64-v8a CPU 架构
处理器性能要求	支持瑞芯微 RK3588 等
开发 IDE	Android Studio
内存要求	大于500MB

2. 模型包说明

模型包分为**基础模型包**和**人物形象包**两种，具体介绍如下：

基础模型包

基础模型包与形象无关， 是运行 2D 端渲染的必备条件，模型包目录名称为： `common_model/` 如果有多个人物模型，可以共用一个基础模型包。

人物形象包

人物形象包里包含了 2D 数智人驱动的**图像数据**和**推理模型**。每个人物形象一个模型包， 加载不同的人物模型包，即可更换形象。模型包目录名称为： `human_xxxxx_540p_v2/` ，其中 xxxxx 为定制形象的名称。
目前 SDK 包里含一个公共形象，包名是：human_yunxi_540p_v2，您可直接使用。

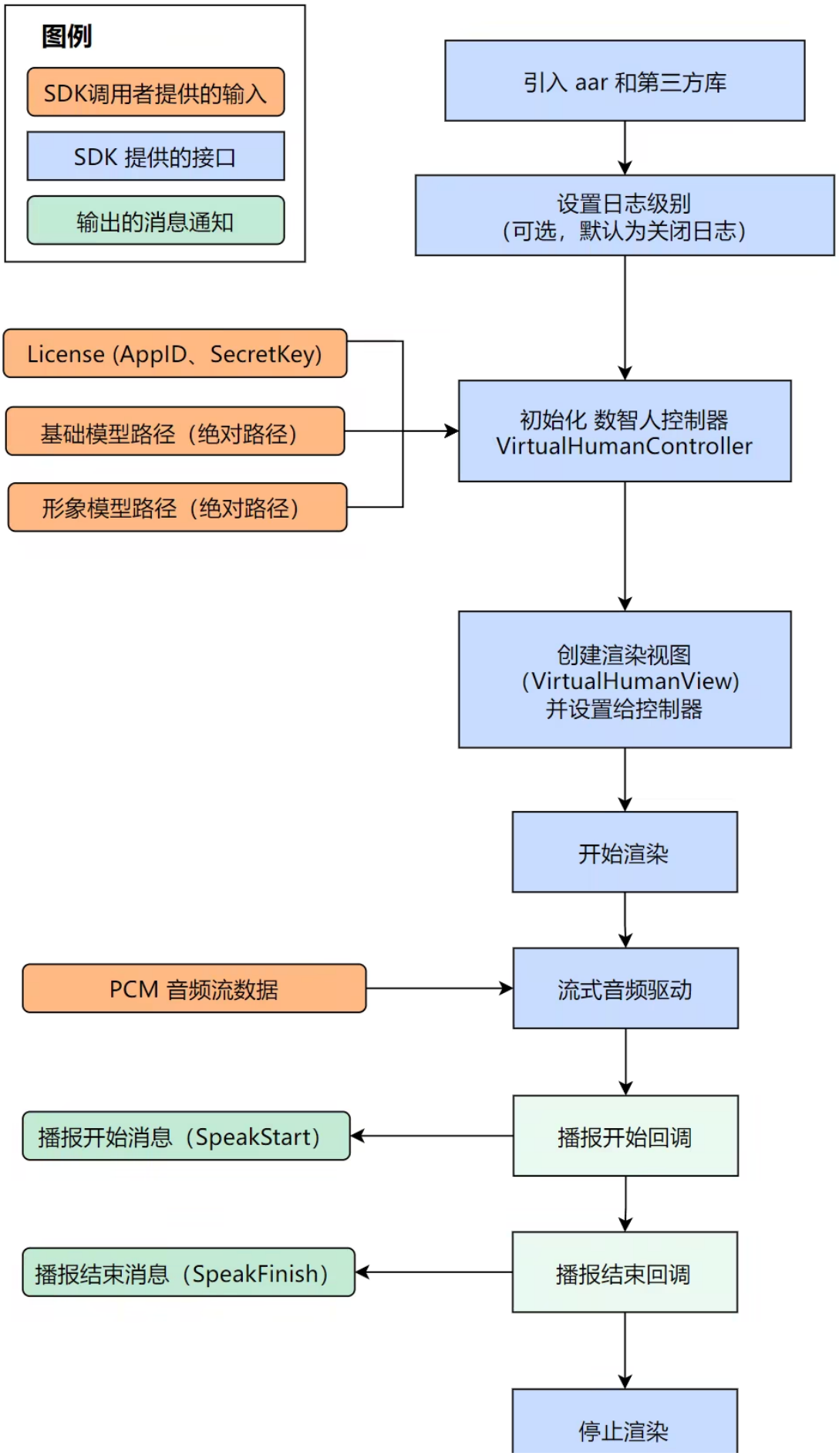
> ⚠ **注意：**
> - SDK 接入方需要将模型包放到 App 的沙盒中，初始化 SDK 时需要将2个模型的绝对路径作为参数传递给 SDK。
> - 模型包可以动态下载，动态下载和更新的逻辑，需要接入方 App 自行实现。

3. 授权方式

大屏端渲染 SDK 采用按设备数授权的方式，请确认申请的 License 为大屏端渲染类型。
新设备第一次启动，会消耗一个 License 配额，请保证 License 有剩余的配额，否则会导致鉴权失败。

4. 数智人客户端渲染 SDK 接入

数智人 SDK 的调用流程如下：


图例
SDK调用者提供的输入
SDK 提供的接口
输出的消息通知
引入 aar 和第三方库
设置日志级别
（可选，默认为关闭日志）
License (AppID、SecretKey)
基础模型路径（绝对路径）
形象模型路径（绝对路径）
初始化 数智人控制器
VirtualHumanController
创建渲染视图
(VirtualHumanView)
并设置给控制器
开始渲染
PCM 音频流数据
流式音频驱动
播报开始消息（SpeakStart）
播报开始回调
播报结束消息（SpeakFinish）
播报结束回调
停止渲染

4.1 引入 AAR 包和第三方库

将数智人端渲染 SDK 的 AAR 包引导到项目中， AAR 包文件命名为： virtualhuman_2d_sdk-release-202603061641_1.6.3_app.aar
由于使用了 OkHttp 库，所以需要在项目的 build.gradle（Module:app）文件中添加 OkHttp 的依赖。

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

    // 添加依赖
    implementation 'androidx.appcompat:appcompat:1.7.0'
    implementation 'com.squareup.okhttp3:okhttp:4.9.0'
    implementation 'com.parse.bolts:bolts-tasks:1.4.0'
}
```

4.2 配置权限

在 AndroidManifest.xml 中添加：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
// 设备绑定时需要添加此权限
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

4.3 基础使用示例

```
// 1. 创建参数
VirtualHumanParam param = new VirtualHumanParam();
param.appId = "your_app_id";
param.secretKey = "your_secret_key";
param.commonModelPath = "/path/to/common_model";
param.humanModelPath = "/path/to/human_model";

// 2. 设置回调
param.eventCallback = (code, message) -> {
    Log.i("VH", "Event: " + code + ", " + message);
```

```
};
param.errorCallback = (code, message) -> {
    Log.e("VH", "Error: " + code + ", " + message);
};

// 3. 创建控制器
VirtualHumanController controller = new VirtualHumanController(context,
param);

// 4. 初始化
int result = controller.initHuman();
if (result != 0) {
    Log.e("VH", "Init failed: " + result);
    return;
}

// 5. 设置渲染视图
VirtualHumanSurfaceView view = new VirtualHumanSurfaceView(context);
controller.setRenderView(view);

// 6. 启动渲染
controller.startHuman();

// 7. 推送音频数据
byte[] audioData = ...; // PCM 16bit 16kHz
controller.appendAudioData(audioData, false);

// 8. 停止并释放
controller.stop();
```

5. 核心类

VirtualHumanController

数智人渲染控制器，管理 Native 渲染引擎。

```
public VirtualHumanController(Context context, VirtualHumanParam param)
```

参数：

- context: Android Context
- param: 数智人参数配置

initHuman()

初始化数智人引擎。

```
public int initHuman()
```

返回值：

- 0: 成功
- 20001: 模型加载错误
- 20010: License 鉴权失败
- 20011: 参数无效
- 20013: 模型尺寸不支持
- 20014: 缺少 READ_PHONE_STATE 权限（仅设备绑定鉴权场景）

示例：

```
int result = controller.initHuman();
if (result == 0) {
    Log.i("VH", "初始化成功");
} else if (result == 20010) {
    Log.e("VH", "License 失效");
}
```

setRenderView()

设置渲染视图。

```
public void setRenderView(IVirtualHumanRenderView view)
```

参数：

- view: 实现 IVirtualHumanRenderView 接口的视图（VirtualHumanSurfaceView 或 VirtualHumanTextureView）

示例：

```
VirtualHumanSurfaceView view = new VirtualHumanSurfaceView(context);
```

```
controller.setRenderView(view);
```

startHuman()

启动数智人渲染。

```
public void startHuman()
```

说明：

- 必须在 initHuman() 成功后调用

appendAudioData()

推送音频数据驱动数智人播放。

```
public void appendAudioData(byte[] audioData, boolean isEnd)
public void appendAudioData(byte[] audioData, boolean isEnd, String
metaData)
```

参数：

- audioData: 音频数据（PCM 16bit 16kHz 单声道）
- isEnd: 是否为最后一帧
- metaData: 元数据（可选，用于同步自定义信息）

音频格式要求：

- 采样率：16000Hz
- 位深度：16bit
- 声道：单声道（Mono）
- 编码：PCM

```
// 流式推送
controller.appendAudioData(chunk1, false, "metadata_1");
controller.appendAudioData(chunk2, false, "metadata_2");
controller.appendAudioData(chunk3, true, "metadata_final");

// 一次性推送
byte[] audioData = loadAudioFile();
controller.appendAudioData(audioData, true);
```

appendAction()

插入动作。

> **说明：**
> - sdkVersion 1.7.0+ 支持。
> - 仅精品模式下生效。
> - 动作执行结果通过事件回调通知（10006 开始、10007 完成、10008 失败）。

```
public void appendAction(String actionCode, long timestampMs)
```

参数：

- actionCode: 动作标识，对应模型中的 action_code（可通过 getAppendableActionList() 查询可用列表）。
- timestampMs: 目标时间戳（毫秒），该动作执行到一半时的时间点。

示例：

```
// 查询可用动作列表后，在指定时刻插入动作
String actionList = controller.getAppendableActionList();
// 如果想当前片段播完后立即执行该动作，可以加上该动作时长的一半时间。
long targetTs = controller.getRunningDurationMs() + 2500;
// 动作标识在actionList列表可看到
controller.appendAction("heart", targetTs);
```

getAppendableActionList()

获取当前模型支持的可插入动作列表。

> **说明：**
> - sdkVersion 1.7.0+ 支持。
> - 仅精品模式下生效。

```
public String getAppendableActionList()
```

返回值：

- JSON 字符串，格式：[{"action_code":"heart","duration_ms":2400}, ...]
- 引擎未初始化时返回 "[]"

示例：

```
String json = controller.getAppendableActionList();
// 解析示例
JSONArray actions = new JSONArray(json);
for (int i = 0; i < actions.length(); i++) {
    JSONObject action = actions.getJSONObject(i);
    String code = action.getString("action_code");     // 动作标识
    int durationMs = action.getInt("duration_ms");     // 动作时长（毫秒）
    Log.i("VH", "动作: " + code + ", 时长: " + durationMs + "ms");
}
```

getRunningDurationMs()

获取数智人运行时长。

> **说明：**
> sdkVersion 1.7.0+ 支持。

```
public long getRunningDurationMs()
```

返回值：

- 基于帧流已输出帧数计算的运行时长（每帧固定 40ms），与内部帧时间戳保持一致
- 与系统时钟无关，不受实际渲染耗时影响
- 未启动时返回 0

示例：

```
long durationMs = controller.getRunningDurationMs();
Log.i("VH", "已运行: " + durationMs + "ms");
```

interrupt()

打断当前播放。

```
public void interrupt()
```

说明：

- 立即停止当前播放内容
- 清空音频缓冲区

pause() / resume()

暂停/恢复渲染。

```
public void pause()
public void resume()
```

说明：

- pause(): 暂停渲染，但保持引擎运行状态
- resume(): 恢复渲染

setMuted()

设置静音状态。

```
public void setMuted(boolean muted)
```

参数：

- muted: true 为静音，false 为取消静音

setFillMode()

设置渲染填充模式。

```
public void setFillMode(VirtualHumanViewType fillMode)
```

参数：

- fillMode: 填充模式枚举
  - VirtualHumanViewType.fill: 保持比例，留黑边

- VirtualHumanViewType.fit: 保持比例，裁切超出
- VirtualHumanViewType.stretch: 拉伸填充，可能变形

stop()

停止并释放所有资源。

```
public void stop()
```

说明：

- 停止渲染
- 释放 Native 资源
- 关闭线程池
- 调用后需要重新 initHuman() 才能使用

getFillMode()

获取当前渲染填充模式。

```
public VirtualHumanViewType getFillMode()
```

返回值：

- 当前 VirtualHumanViewType 枚举值

getRenderView()

获取当前绑定的渲染 View。

```
public IVirtualHumanRenderView getRenderView()
```

返回值：

- 当前渲染 View，未设置时返回 null

setLogLevel()

设置日志级别。

```
public static void setLogLevel(int level)
```

参数：

- VirtualHumanController.LOG_LEVEL_OFF: 关闭日志
- VirtualHumanController.LOG_LEVEL_ERROR: 错误
- VirtualHumanController.LOG_LEVEL_WARN: 警告
- VirtualHumanController.LOG_LEVEL_INFO: 信息
- VirtualHumanController.LOG_LEVEL_DEBUG: 调试
- VirtualHumanController.LOG_LEVEL_TRACE: 追踪

示例：

```
VirtualHumanController.setLogLevel(VirtualHumanController.LOG_LEVEL_INFO
);
```

setLogCallback()

设置日志回调。

```
public static void setLogCallback(IDataCallback logCallback)
```

参数：

- logCallback: 日志回调接口

示例：

```
VirtualHumanController.setLogCallback((level, message) -> {
    Log.i("VH", "Log[" + level + "]: " + message);
});
```

getVersion()

获取 SDK 版本。

```
public static String getVersion()
```

返回值：

- SDK 版本字符串

runPerformanceTest()

运行性能检测。

> ⚠ **注意：**
> 该方法基于设备实时硬件配置执行基准测试，测试结果由当前设备的 SOC型号、负载状态及运行环境共同决定，因此同一设备在不同时刻可能存在差异。

```
public static void runPerformanceTest(Context context, VirtualHumanParam
param, IDataCallback callback)
public static void runPerformanceTest(Context context, VirtualHumanParam
param, IDataCallback callback, boolean forceRefresh)
```

参数：

- context: Android Context
- param: 数智人参数配置
- callback: 检测结果回调
- forceRefresh: 是否强制刷新（忽略缓存）

回调参数：

- code:
- 0: 性能达标
- 1: 性能不足
- 负数: 检测失败
- message: 详细信息

说明：

- 使用内置测试音频检测设备性能
- 会自动初始化、测试、销毁，无需手动管理生命周期
- 支持缓存机制，默认使用缓存结果（除非 forceRefresh=true）

示例：

```
VirtualHumanController.runPerformanceTest(context, param, (code,
message) -> {
    if (code == 0) {
        Log.i("VH", "性能达标");
    } else if (code == 1) {
```

```
            Log.w("VH", "性能不足");
        } else {
            Log.e("VH", "检测失败: " + message);
        }
    });

    // 强制刷新，忽略缓存
    VirtualHumanController.runPerformanceTest(context, param, callback,
    true);
```

checkDeviceWhitelist()

检查设备黑白名单。

> ⚠ **注意：**
> 由于 Android 设备碎片化程度较高，黑白名单结果仅供参考，我们将持续维护并定期更新数据，以确保其准确性与可靠性。

```
public static void checkDeviceWhitelist(Context context,
VirtualHumanParam param, IDataCallback callback)
public static void checkDeviceWhitelist(Context context,
VirtualHumanParam param, IDataCallback callback, boolean forceRefresh)
```

参数：

- context: Android Context
- param: 数智人参数配置
- callback: 检测结果回调
- forceRefresh: 是否强制刷新（忽略缓存）

回调参数：

- code:
- 0: 白名单（性能达标）
- 1: 黑名单（性能不足）
- 2: 未知设备（不在黑白名单中）
- -1: SOC 获取失败
- -2: 网络请求失败或初始化失败
- message: 详细信息

说明：

- 根据设备 SOC 和模型参数判断设备是否在黑白名单中
- 会请求远程配置并自动初始化引擎
- 支持内存缓存机制（进程级别），默认使用缓存（除非 forceRefresh=true）

示例：

```
VirtualHumanController.checkDeviceWhitelist(context, param, (code,
message) -> {
    switch (code) {
        case 0:
            Log.i("VH", "白名单：设备性能达标");
            break;
        case 1:
            Log.w("VH", "黑名单：设备性能不足");
            break;
        case 2:
            Log.i("VH", "未知设备：不在黑白名单中");
            break;
        case -1:
            Log.e("VH", "SOC 获取失败");
            break;
        case -2:
            Log.e("VH", "检测失败： " + message);
            break;
    }
});

// 强制刷新，忽略缓存
VirtualHumanController.checkDeviceWhitelist(context, param, callback,
true);
```

VirtualHumanParam

数智人参数配置类。

字段

```
// ===== 必填字段 =====
```

```
// License 鉴权
public String appId;                // 应用 ID
public String secretKey;            // 密钥

// 模型路径
public String commonModelPath;      // 通用模型路径
public String humanModelPath;       // 人物模型路径

// ===== 可选字段 =====

// 设备标识（仅设备绑定授权场景需要）
public String deviceName;           // 设备唯一标识，必须保持固定不变

// 性能优化
public int skipFrameInterval = 7; // 跳帧间隔（0=不跳帧，7=每8帧跳1帧）
public boolean dualGen = false;     // 是否启用双线程推理

/*
 * 数字人在静默时无额外手部动作，交互更真实自然。
 * 注：如果形象本身不支持该特性，则该配置项不生效。
 * 默认值为true
 */
public boolean enableSilenceList = true;

// 回调
public IDataCallback eventCallback;   // 事件回调
public IDataCallback errorCallback;   // 错误回调
```

示例

```
VirtualHumanParam param = new VirtualHumanParam();

// 必填
param.appId = "your_app_id";
param.secretKey = "your_secret_key";
param.commonModelPath = "/sdcard/models/common_model";
param.humanModelPath = "/sdcard/models/human_001";

// 可选：设备标识（仅设备绑定授权场景需要）
```

```
// 重要：同一台设备必须始终使用相同的 deviceName，否则会导致鉴权失败
param.deviceName = "SN20240101001";  // 使用设备序列号（推荐）
// 或使用其他固定标识：
// param.deviceName = "ASSET-A-001";       // 资产编号
// param.deviceName = "00:11:22:33:44:55"; // MAC 地址

// 可选：性能优化
param.skipFrameInterval = 7;  // 跳帧优化
param.dualGen = true;         // 双线程加速

// 可选：回调
param.eventCallback = (code, message) -> {
    Log.i("VH", "Event: " + code);
};
param.errorCallback = (code, message) -> {
    Log.e("VH", "Error: " + code + ", " + message);
};
```

说明：

- APP 绑定授权（bindDevice=false）：deviceName 可以为空或不设置
- 设备绑定授权（bindDevice=true）：
  - deviceName 必须设置，用于设备唯一性标识
  - 同一台物理设备必须始终使用相同的 deviceName
  - 如果 deviceName 变化，服务器会认为是不同设备，导致鉴权失败或消耗额外授权额度
  - 建议使用设备出厂铭牌上的序列号、资产编号等固定不变的标识
  - 不要使用随机值或每次变化的值

VirtualHumanSurfaceView

基于 SurfaceView 的渲染视图。

特点

- ✓ 性能更好，适合全屏渲染
- ✓ 独立的 Surface Layer
- ✗ UI 控件叠加需要额外处理（setZOrderOnTop 后普通 View 会被遮挡）

使用示例

```
<FrameLayout
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.tencent.virtualhuman_2d_sdk.VirtualHumanSurfaceView
        android:id="@+id/vh_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

```
VirtualHumanSurfaceView view = findViewById(R.id.vh_view);
controller.setRenderView(view);
```

VirtualHumanTextureView

基于 TextureView 的渲染视图。

特点

- ✅ 支持透明背景
- ✅ UI 控件可以正常叠加显示
- ✅ 支持动画和变换
- ❌ 性能稍逊于 SurfaceView

使用示例

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- 背景层 -->
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/background" />

    <!-- 渲染层 -->
    <com.tencent.virtualhuman_2d_sdk.VirtualHumanTextureView
        android:id="@+id/vh_view"
        android:layout_width="match_parent"
```

```
        android:layout_height="match_parent" />

    <!-- 叠加控件层 -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="操作按钮" />

</FrameLayout>
```

```
VirtualHumanTextureView view = findViewById(R.id.vh_view);
controller.setRenderView(view);
```

VirtualHumanViewType

渲染填充模式枚举。

枚举值

```
public enum VirtualHumanViewType {
    fill,       // 保持比例，留黑边（默认）
    fit,        // 保持比例，裁切超出
    stretch     // 拉伸填充，可能变形
}
```

示例

```
// 保持比例，留黑边
controller.setFillMode(VirtualHumanViewType.fill);

// 保持比例，裁切超出
controller.setFillMode(VirtualHumanViewType.fit);

// 拉伸填充
controller.setFillMode(VirtualHumanViewType.stretch);
```

6. 回调接口

IDataCallback

通用数据回调接口。

```
public interface IDataCallback {
    void onDataCallback(int code, String message);
}
```

用途

1. 事件回调（param.eventCallback）
2. 错误回调（param.errorCallback）
3. 日志回调（setLogCallback）
4. 性能检测回调（runPerformanceTest）
5. 黑白名单检测回调（checkDeviceWhitelist）

7. 回调码

错误码

错误码	常量	说明
20001	ERROR_INIT_MODEL	模型加载失败
20010	ERROR_LICENSE_INVALID	License 鉴权失败
20011	ERROR_INIT_PARAM_INVALID	初始化参数无效
20012	ERROR_PERFORMANCE	设备性能不足
20013	ERROR_MODEL_SIZE_NOT_SUPPORTED	模型尺寸不支持（小屏版 SDK 不支持高尺寸模型）
20014	ERROR_PERMISSION_DENIED	缺少必要权限（设备绑定鉴权需要 READ_PHONE_STATE）

事件码

事件码	说明
10001	播放开始
10002	播放结束
10003	元数据开始

10004	元数据结束
10005	首帧渲染完成
50001	性能统计信息（包含平均耗时、最大耗时等）

TTS SDK 接入

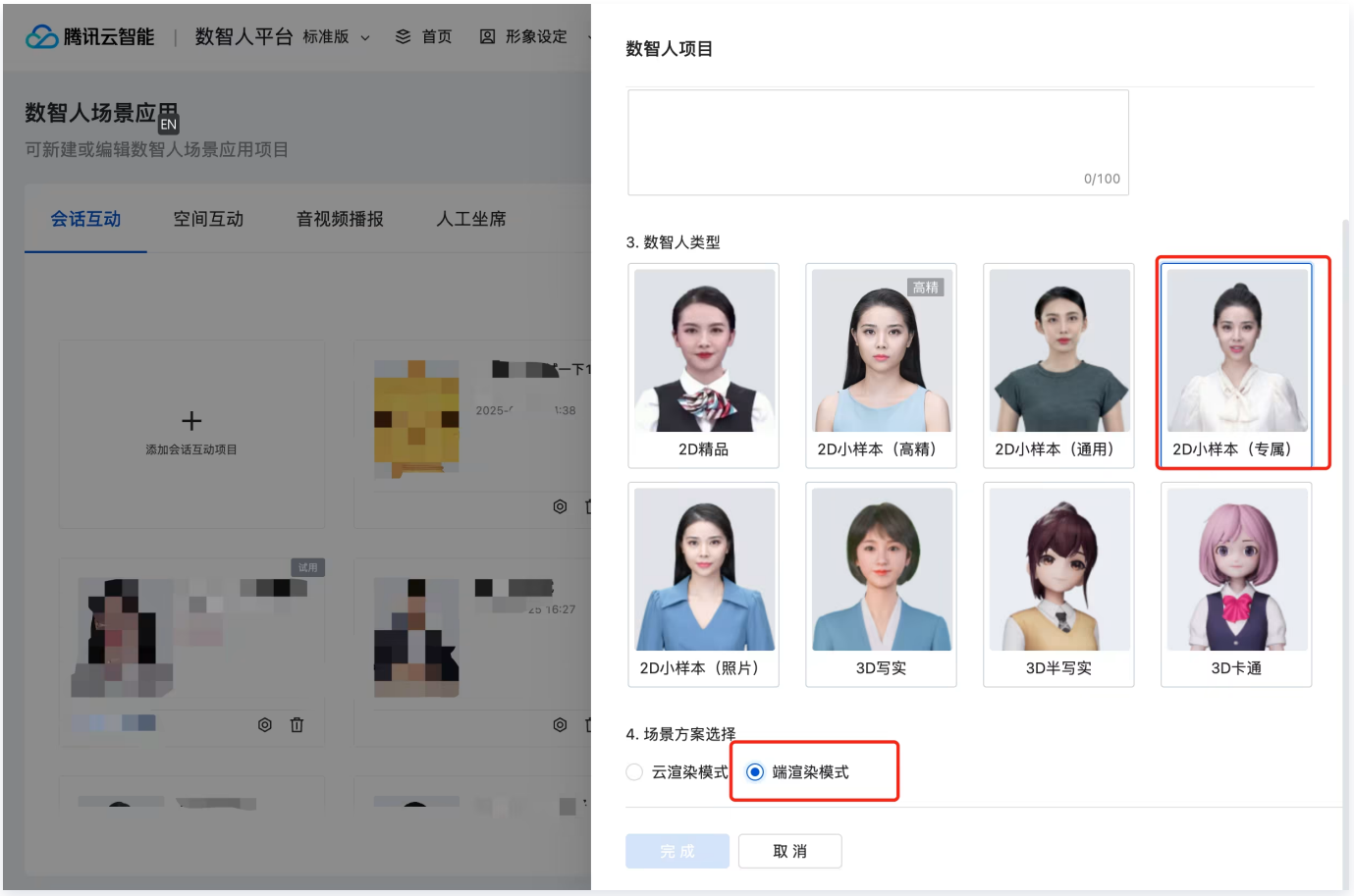
最近更新时间：2025-09-17 18:34:21

获取 SDK 文件

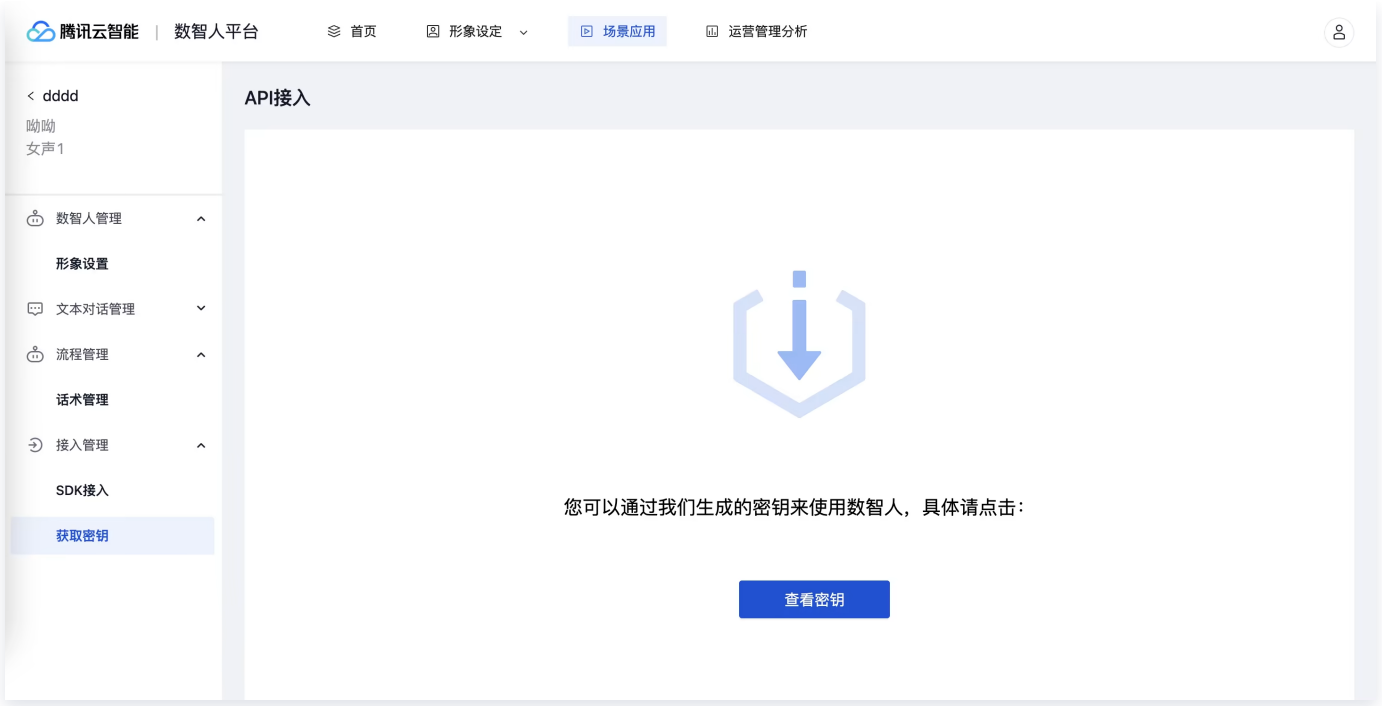
TTS SDK 文件在线下提供给客户的文件压缩包中，以“virtualhuman_apaas_sdk”开头的 aar 文件。

获取密钥

1. 进入 数智人平台 > 场景应用 > 会话互动，创建项目。



2. 在获取密钥菜单中，单击“**查看密钥**”，即可获取 appKey， accessToken 和 virtualmanProjectId 。



VirtualHumanApaas 接口说明

初始化接口

java

```
VirtualHumanApaas vha = VirtualHumanApaas.getInstance();
VirtualHumanApaasParam vhap = new VirtualHumanApaasParam();
vhap.appKey = BuildConfig.ttsAppkey;
vhap.accessToken = BuildConfig.ttsAccessToken;
vhap.virtualmanProjectId = BuildConfig.ttsVirtualmanProjectId;
vha.init(vhap);
```

swift

```
let vha = VirtualHumanApaas.shared
let vhap = VirtualHumanApaasParam(
    appKey: APASS_APP_KEY,
    accessToken: APASS_ACCESS_TOKEN,
    virtualmanProjectId: APASS_PROJECT_ID
)
vha.initApaas(vhap)
vha.callbackDelegate = self
```

oc

```
VirtualHumanApaas* vha = [VirtualHumanApaas shared];
VirtualHumanApaasParam* vhap = [[VirtualHumanApaasParam alloc] init];
vhap.appKey = APASS_APP_KEY;
vhap.accessToken = APASS_ACCESS_TOKEN;
vhap.virtualmanProjectId = APASS_PROJECT_ID;
[vha initApaas:vhap];
vha.callbackDelegate = self;
```

设置回调

java

```
// error回调
vha.setErrorCallback((code, message) -> {
    Log.e(TAG, code + ", " + message);
});
// 数据回调
vha.setDataCallback((result) -> {
    String type = result.getDriverRspType();
    if (type.equals("SPEECH")) {
        VirtualHumanApaas.Result.SpeechRsp speechRsp = result.speechRsp;
        // controller是数智人控制器，这里是结合端渲染sdk的使用
        if (controller != null) {
            controller.appendAudioData(speechRsp.getAudio(),
speechRsp.isFinal());
        }
    } else {
        VirtualHumanApaas.Result.ReplyRsp replyRsp = result.replyRsp;
        Log.i(TAG, replyRsp.getReplyDisplay());
    }
});
```

swift

```
extension DemoTTSViewController: CallbackDelegate {
```

```
    nonisolated func onData(_ result: VirtualHumanApaas.Result) {
        let driverRspType = result.driverRspType
        if (driverRspType != "SPEECH") {
            print("tts data: \(result.replyRsp?.replyType ?? ""), \
(result.replyRsp?.replyDisplay ?? "")")
        }
        else {
            let data = result.speechRsp?.audio ?? Data()
            let isFinal = result.speechRsp?.isFinal ?? true
            DispatchQueue.main.async { [weak self] in
                if (self?.controller != nil) {
                    self?.controller!.appendAudioData(data, metaData:
"", isFinal: isFinal)
                }
            }
        }
    }

    nonisolated func onError(_ code: Int, _ message: String) {
        print("==========")
        print("\(code): \(message)")
    }
}
```

OC

```
- (void)onData:(Result*)result {
    NSString* driverRspType = [result driverRspType];
    if (![driverRspType isEqualToString:@"SPEECH"]) {
        NSLog(@"tts data: %@, %@", [[result replyRsp] replyType],
[[result replyRsp] replyDisplay]);
    } else {
        NSData* data = [[result speechRsp] audio] ?: [NSData data];
        BOOL isFinal = [[result speechRsp] isFinal];
        dispatch_async(dispatch_get_main_queue(), ^{
            if (self.controller != nil) {
                [self.controller appendAudioData:data metaData:@""
isFinal:isFinal];
            }
```

```
        });
    }
}

- (void)onError:(int)code message:(NSString*)message {
    NSLog(@"==========");
    NSLog(@"%d: %@", code, message);
}
```

驱动接口

参数

参数名称	数据类型	必选	说明
reqId	String	是	单次请求唯一标识，长度为32的 UUID ，流式输入时要保证同一个流 reqId 一致
seq	Int	是	流式文本序号，从1开始
isFinal	Boolean	是	流式文本分片的结束标记（每一段流式文本结束必须传入结束标记）
text	String	是	文本内容

java

```
String reqId = UUID.randomUUID().toString().replace("-", "");
vha.sendTTS(reqId, 1, true, "这是一句话。");
```

swift

```
let reqId = UUID().uuidString.replacingOccurrences(of: "-", with:
"").lowercased()
let text = "你好"
vha.sendTTS(reqId: reqId, seq: 1, isFinal: true, text: text)
```

oc

```
NSString* reqId = [[[NSUUID UUID] UUIDString]
stringByReplacingOccurrencesOfString:@"-"
withString:@""].lowercaseString;
NSString* text = @"你好";
[self.vha sendTTSWithReqId:reqId seq:1 isFinal:YES text:text];
```

对话接口

> ⚠ **注意：**
> 创建项目时，必须是对话项目，否则接口会返回错误。

参数

参数名称	数据类型	必选	说明
reqId	String	是	单次请求唯一标识，长度为32的 UUID
streamId	String	是	会话 ID，用于区分多轮会话，长度为32的 UUID（不包含‘-’）
text	String	是	文本内容

java

```
String reqId = UUID.randomUUID().toString().replace("-", "");
String text = "Hi~ 你好。";
String streamId = UUID.randomUUID().toString().replace("-", "");
vha.sendChat(reqId, streamId, text);
```

swift

```
let reqId = UUID().uuidString.replacingOccurrences(of: "-", with:
"").lowercased()
let streamId = UUID().uuidString.replacingOccurrences(of: "-", with:
"").lowercased()
let text = "你好"
vha.sendChat(reqId: reqId, streamId: streamId, text: text)
```

oc

```
NSString* reqId = [[[NSUUID UUID] UUIDString]
stringByReplacingOccurrencesOfString:@"-"
withString:@""].lowercaseString;
NSString* streamId = [[[NSUUID UUID] UUIDString]
stringByReplacingOccurrencesOfString:@"-"
withString:@""].lowercaseString;
NSString* text = @"你好";
[self.vha sendChatWithReqId:reqId streamId:streamId text:text];
```

停止中断

java

```
vha.stopTTS();
```

swift

```
vha.stopTTS()
```

oc

```
[vha stopTTS];
```

设置音色

> ⚠ **注意：**
> 无效的音色值，会引发接口错误。

参数

数据类型	必选	说明
String	是	音色资产 ID

java

```
vha.setTimbreKey("音色资产ID");
```

swift

```
vha.setTimbreKey("音色值")
```

oc

```
[vha setTimbreKey:@"音色值"];
```

音色资产 ID 获取方式： 数智人平台 > 资产管理中心 > 声音资产 > 资产 ID

设置播放速度

参数

数据类型	必选	说明
Float	是	0.5f 到 2.0f 范围内

java

```
vha.setSpeed(0.5f);
```

swift

```
vha.setSpeed(0.5)
```

oc

```
[vha setSpeed:0.5f];
```

其他参数说明

VirtualHumanParam 参数说明

参数名称	数据类型	必选	说明
appKey	String	是	获取密钥里的参数
accessToken	String	是	获取密钥里的参数
virtualmanProjectId	String	是	获取密钥里的参数
disableTls	Boolean	否	是否禁用 TLS 协议，默认值 false
serverDomain	String	否	设置私有化域名，默认公有云域名

VirtualHumanApaas 回调函数参数 Result 格式说明

参数名称	数据类型	说明
reqId	String	单次请求 ID，和入参一致
driverRspType	String	响应类型 • REPLY：返回 ReplyRsp，对应会话信息 • SPEECH：返回 SpeechRsp，对应音频信息
replyRsp	ReplyRsp	会话响应，当 driverRspType 为 REPLY 时返回
speechRsp	SpeechRsp	音频信息响应，当 driverRspType 为 SPEECH 时返回

ReplyRsp

参数名称	数据类型	说明
replyType	String	回复语类型。 • cloudAiGpt：腾讯云大模型对话 • yunxiaowei：云小微客服对话 • cloudAiWaiting：首包超时等待话术 • cloudAiTimeOut：超时未响应话术，会话结束 • sensitive：输入文本或回复语中包含敏感内容时，返回的固定话术 • input：InputText 为纯文本或流式文本时输入的内容 • enhanceText：未配置对话服务时，匹配到话术管理的内容

		• cloudAiThought：腾讯云大模型思考过程内容
replyPro	String	播报内容，包含 SSML 标签
replyDisplay	String	展示内容，包含富文本标签
interactionType	String	特殊消息类型
interactionContent	String	特殊消息内容，用于下发弹窗、图片等非文本类的特殊消息
uninterrupt	Boolean	当前播报内容是否可打断
muted	Boolean	当前播报内容，是否关闭收音
seqNo	Int	子句序号，当 replyType 为 cloudAiGpt 时，正常回复语序号从1开始，其余固定话术从0开始
contentType	Int	回复语内容类型 • 0：未知 • 1：普通字符串 • 2：有序列表 • 3：无序列表 • 4：图片链接 • 5：http链接 • 6：表格 • 8：标题 • 9：SSML
ttsSupport	Boolean	当前子句是否播报
isFinal	Boolean	是否为最后一句
isHighLight	Boolean	是否需要高亮展示

SpeechRsp

参数名称	数据类型	说明
audio	Android：byte[] iOS：Data/NSData	PCM 音频数据
isFinal	Boolean	整句结束标识
sampling	Int	采样率

seqNo	Int	子句序号
subtitleInfo	Array of [SubtitleInfo]	字幕信息

SubtitleInfo

参数名称	类型	描述
Word	String	对应单词
Start	String	起始时间点，单位为0.1us，该数值/10000为ms
End	String	结束时间点，单位为0.1us，该数值/10000为ms
PosStart	String	文本中的起始 unicode 位置，注意为左闭右开形式 [PosStart,PosEnd)
PosEnd	String	文本中的结束 unicode 位置，注意为左闭右开形式 [PosStart,PosEnd)

接入示例

Android

将数智人端渲染 SDK 的 aar 包引导到项目中，aar 包文件命名格式为：virtualhuman_tts_sdk_版本信息.aar。
由于使用了 OkHttp 库，所以需要在项目的 build.gradle（Module:app）文件中添加 OkHttp 的依赖。

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

    // 添加OkHttp的依赖
    implementation("com.squareup.okhttp3:okhttp:4.9.0")
}
```

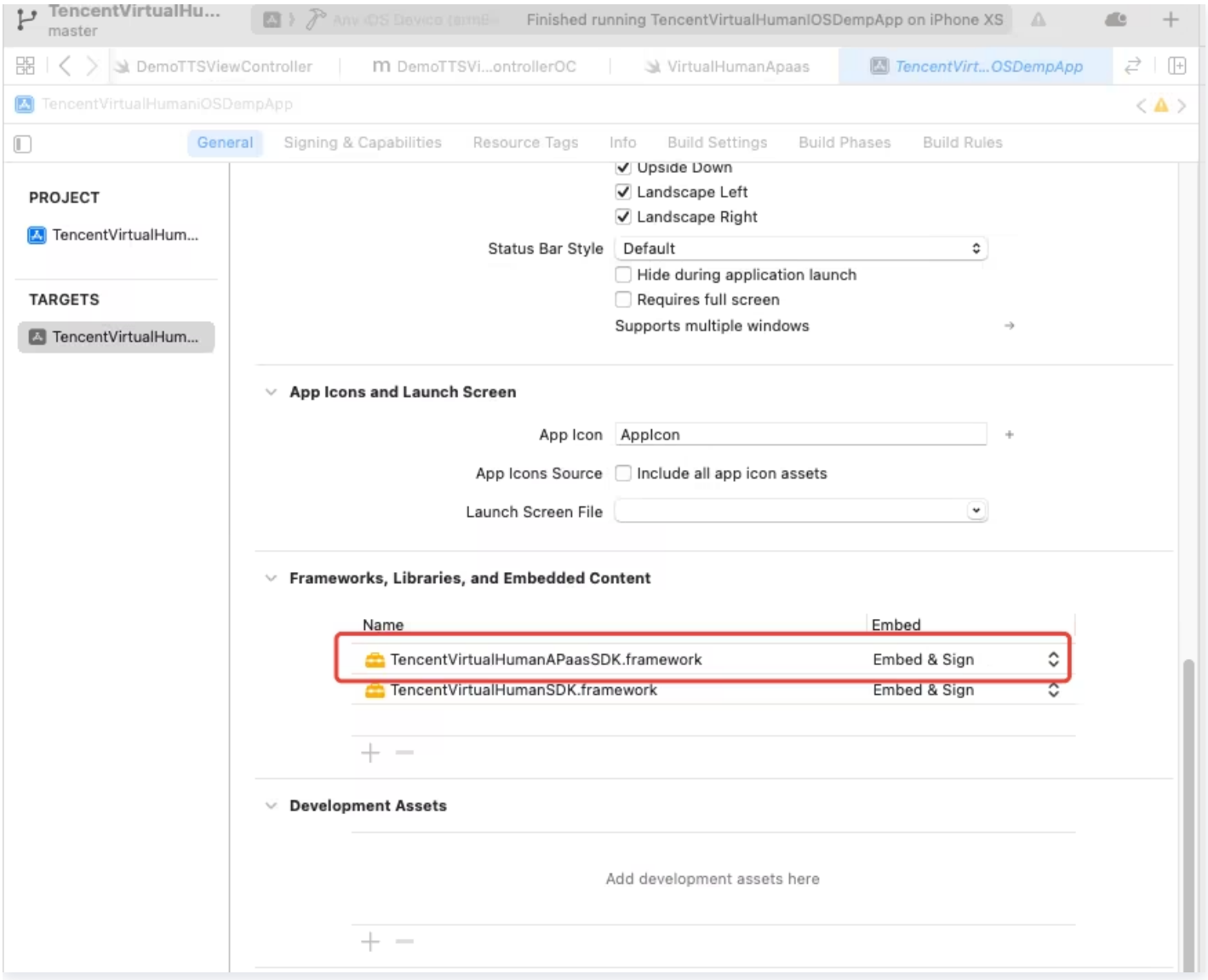
示例

```
VirtualHumanApaas vha = VirtualHumanApaas.getInstance();
VirtualHumanApaasParam vhap = new VirtualHumanApaasParam();
```

```
vhap.appKey = BuildConfig.ttsAppkey;
vhap.accessToken = BuildConfig.ttsAccessToken;
vhap.virtualmanProjectId = BuildConfig.ttsVirtualmanProjectId;
// error回调
vha.setErrorCallback((code, message) -> {
    Log.e(TAG, code + ", " + message);
});
// 数据回调
vha.setDataCallback((result) -> {
    String type = result.getDriverRspType();
    if (type.equals("SPEECH")) {
        VirtualHumanApaas.Result.SpeechRsp speechRsp = result.speechRsp;
        // controller是数智人控制器，这里是结合端渲染sdk的使用
        if (controller != null) {
            controller.appendAudioData(speechRsp.getAudio(),
speechRsp.isFinal());
        }
    } else {
        VirtualHumanApaas.Result.ReplyRsp replyRsp = result.replyRsp;
        Log.i(TAG, replyRsp.getReplyDisplay());
    }
});
vha.init(vhap);
```

iOS

将 Framework 添加到工程中， 嵌入方式选择： Embed & Sign 。



并引入头文件：

在 Objective-C 中使用时，直接引入头文件， 在 Swift 中使用时， 需要在 xxxxx-Bridging-Header.h 文件中引入头文件。

```
#import <TencentVirtualHumanAPaasSDK/TencentVirtualHumanAPaasSDK-
Swift.h>
```

示例

swift

```
import UIKit
import TencentVirtualHumanAPaasSDK

class DemoTTSViewController: UIViewController {
```

```
    let vha = VirtualHumanApaas.shared

    override func viewDidLoad() {
        super.viewDidLoad()

        let vhap = VirtualHumanApaasParam(
            appKey: APASS_APP_KEY,
            accessToken: APASS_ACCESS_TOKEN,
            virtualmanProjectId: APASS_PROJECT_ID
        )
        vha.initApaas(vhap)
        vha.callbackDelegate = self
    }
}

extension DemoTTSViewController: CallbackDelegate {

    nonisolated func onData(_ result: VirtualHumanApaas.Result) {
        // 处理回调数据
    }

    nonisolated func onError(_ code: Int, _ message: String) {
        print("\(code): \(message)")
    }
}
```

OC

```
#import <UIKit/UIKit.h>
#import <TencentVirtualHumanAPaasSDK/TencentVirtualHumanAPaasSDK-
Swift.h>

@interface DemoTTSViewController : UIViewController <CallbackDelegate>

@end

@implementation DemoTTSViewController
```

```
- (void)viewDidLoad {
    [super viewDidLoad];

    VirtualHumanApaas* vha = [VirtualHumanApaas shared];
    VirtualHumanApaasParam* vhap = [[VirtualHumanApaasParam alloc]
init];
    vhap.appKey = APASS_APP_KEY;
    vhap.accessToken = APASS_ACCESS_TOKEN;
    vhap.virtualmanProjectId = APASS_PROJECT_ID;
    [vha initApaas:vhap];
    vha.callbackDelegate = self;
}

#pragma mark - CallbackDelegate

- (void)onData:(Result*)result {
    // 处理回调数据
}

- (void)onError:(int)code message:(NSString*)message {
    NSLog(@"%d: %@", code, message);
}

@end
```

端云混合渲染方案

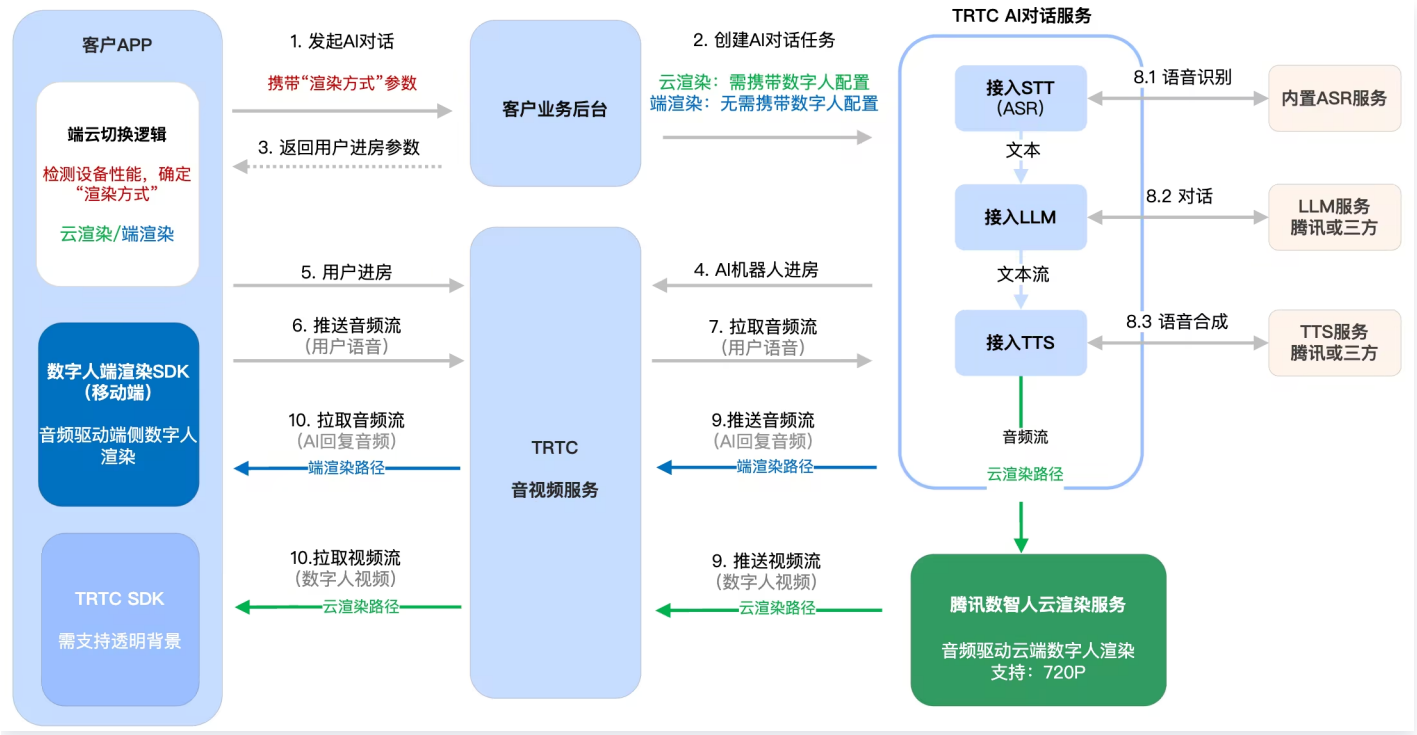
最近更新时间：2025-12-30 12:01:40

一、方案概述

在2D数字人端渲染的实际落地中，部分移动端设备可能因硬件性能限制，难以稳定、流畅地进行数字人渲染。为解决终端设备兼容性与性能差异问题，我们推出端云混合渲染方案。通过内置的“端云切换逻辑”，提前判断设备性能，走不同的渲染路径。

- 云渲染路径：设备性能不足时，使用云端渲染服务，保障流畅稳定的交互体验。
- 端渲染路径：设备性能充足时，使用端侧渲染服务，充分发挥本地算力优势。

本方案需要结合腾讯云实时音视频（TRTC）服务一起使用，整体架构如下：



说明：

1. 本方案需使用**支持透明背景的专版 TRTC SDK**，具体获取路径请参阅下文“客户端代码示例”中的 README 文档。
2. **端渲染 SDK**：该方案当前仅支持移动端设备使用。
3. **云渲染服务**：支持 720P 分辨率的数字人视频输出。

二、核心机制：端云切换逻辑

客户端通过“端云切换逻辑”动态判断当前会话应使用云渲染还是端渲染路径，确保渲染效果与设备性能相匹配。该机制采用“静态名单筛查 + 动态性能检测”双重策略：

1. **静态名单筛查：**系统优先校验设备型号是否在预置 “性能不兼容黑名单” 中，若命中，则直接锁定云渲染路径。
2. **动态性能压测：**通过名单筛查的设备，将执行一次无界面静默性能测试，进一步判断性能：
   - 若测试触发20012错误码，判定为设备性能不足，需采用云渲染路径；
   - 若测试全程无异常，则默认采用端渲染路径。

> **说明：**具体细节可参考下文代码示例。

三、快速开始

我们提供了两种方式，帮助您快速体验和集成端云混合渲染方案：

1. 体验演示应用（APK）

为方便您直观了解端云切换效果，我们提供了开箱即用的 Android 演示应用。您可通过邮件申请获取 APK，直接在真实设备上体验完整流程。

> **端云切换 Android 示例 APK 申请流程（邮件）**
>
> - 邮箱地址：txc_avatar@tencent.com
> - 邮件主题：2D 端渲染端云切换 Android 示例 APK 试用申请
> - 邮件内容（请详细提供如下信息）：
>   - 腾讯云主账号：
>   - 客户/公司名称：
>   - 计划使用场景：

说明：此演示包集成的后端服务接口（如黑白名单拉取、云端推理等）**仅用于功能演示**，不可直接用于生产环境。正式业务部署中，所有后端服务需您自行搭建。

2. 获取并运行代码示例

如果您希望直接集成代码，我们提供了完整的客户端与服务端示例，助您快速开发验证。
客户端代码示例：展示端云切换的核心实现逻辑，包含“黑白名单筛查”与“运行时性能检测”机制。
下载 Android 代码示例（iOS 代码示例规划中）
服务端代码示例：基于腾讯音视频AI服务能力构建，展示服务端实时对话接口的实现方案。
下载服务端代码示例

四、所需采购的服务

1. 数智人相关资源

序号	资源类型	资源用途	获取方式

1	2D 端渲染 SDK 授权使用年包-按应用	用于端渲染	详见：2D 端渲染场景概述→购买及接入流程
2	轻量版云端服务10小时包	• 用于云渲染的资源消耗 • 仅支持 720P	详见：2D 端渲染场景概述→其他相关服务

2. TRTC 相关服务

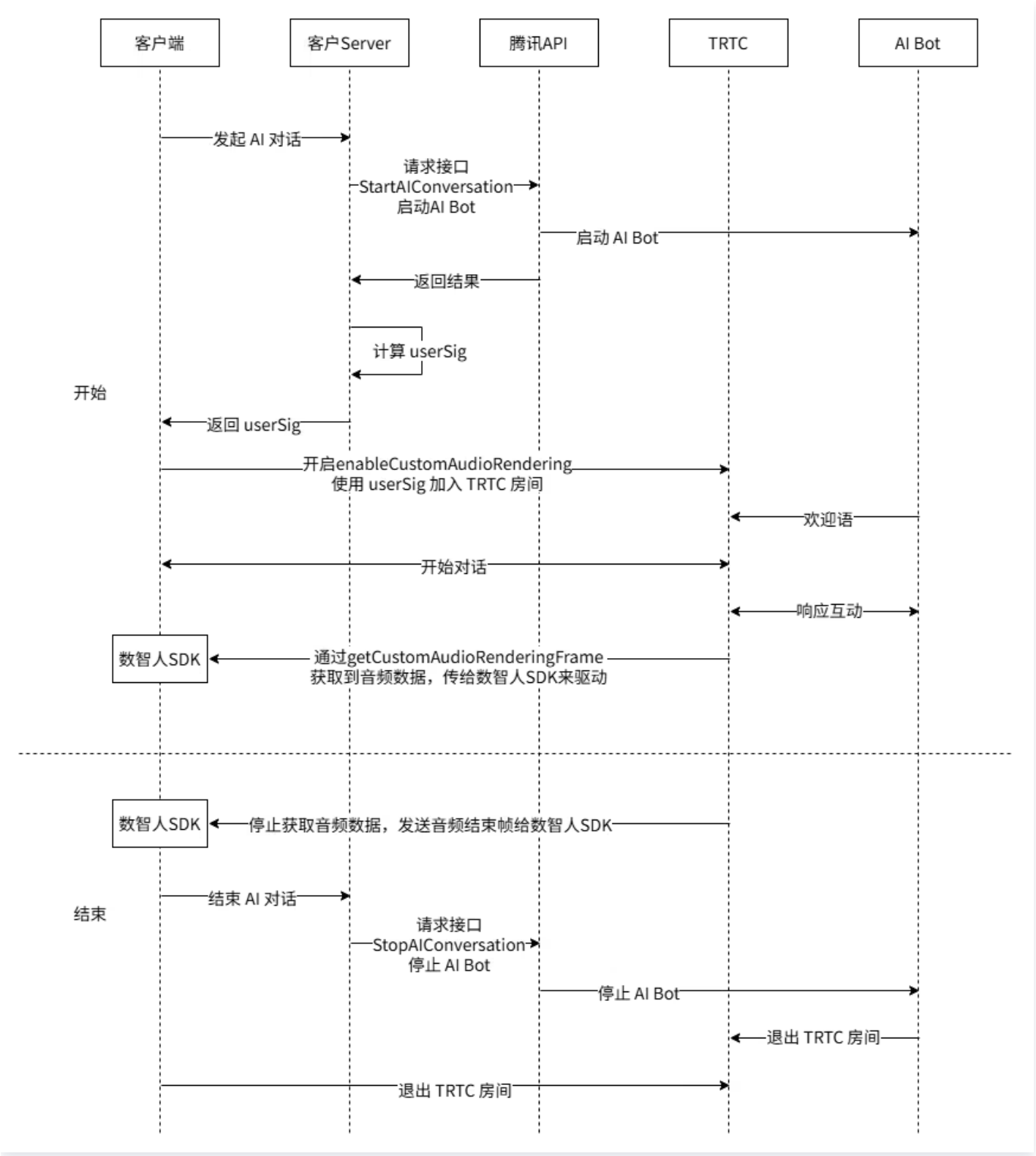
方案主要用到了“TRTC AI 实时对话服务”，费用由三部分组成：

序号	资源类型	资源详情	购买方式
1	音频通话费用	真人与 AI 通话的音频通话费用，详细计费参见：音视频时长计费说明	新账号可以免费领取10000分钟的免费音视频时长包。领取入口
2	语音转文字费用	需购买 AI 智能识别套餐包（轻量版 / 基础版 / 尊享版）。详情计费参见：AI 智能识别计费说明	新用户可以购买50元体验套餐。购买入口
3	AI实时对话服务	使用 AI 实时对话需要支付对应的服务费用。详细计费参见：AI 实时对话计费说明	只有后付费方式，无需提前购买。具体接入步骤请参见 AI 对话 接入说明

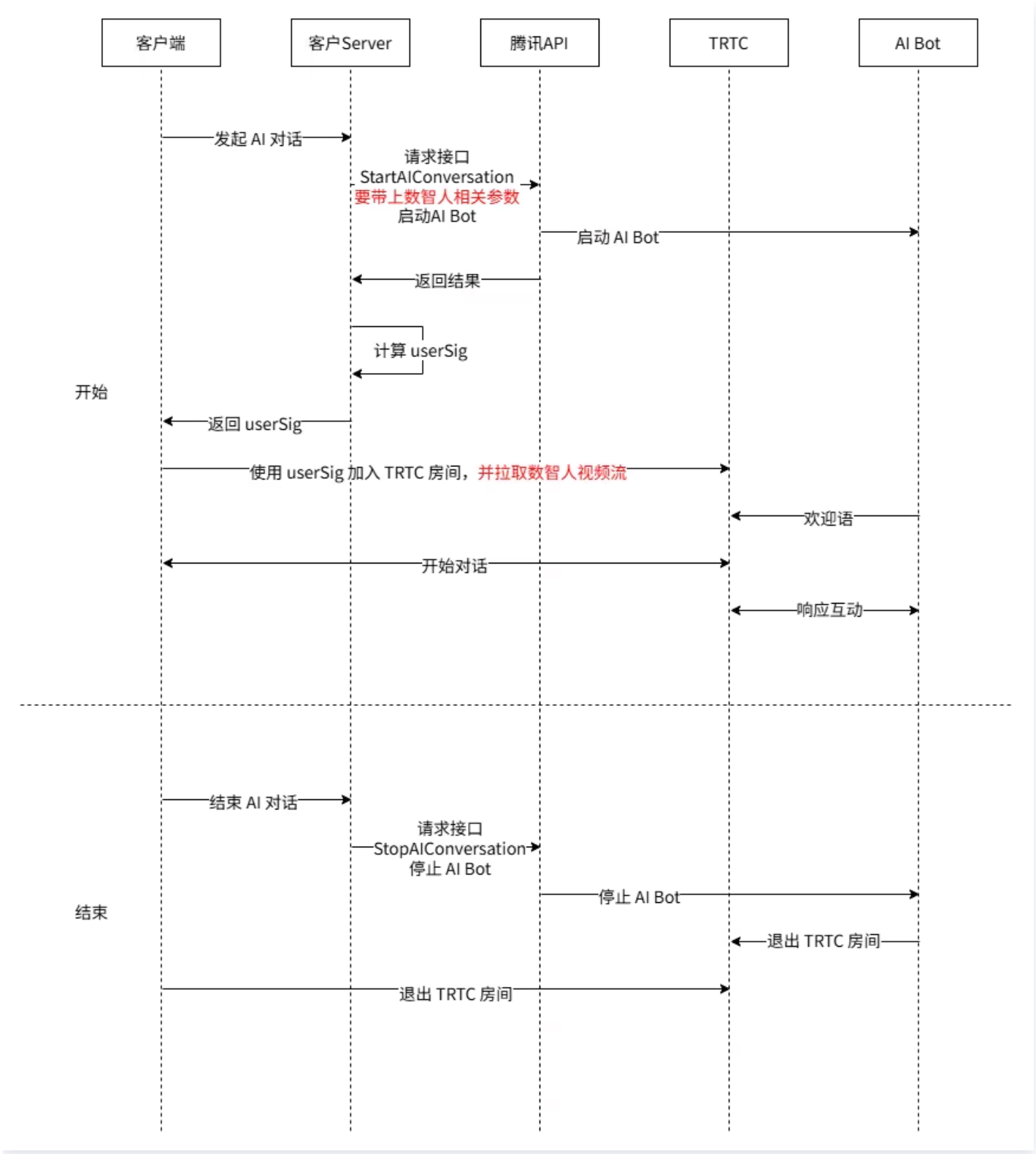
> **说明：**
> 详细说明参见 TRTC AI 实时对话 文档。

五、技术架构图

1. 端渲染流程图


客户端
客户Server
腾讯API
TRTC
AI Bot
发起 AI 对话
请求接口
StartAIConversation
启动AI Bot
启动 AI Bot
返回结果
计算 userSig
开始
返回 userSig
开启enableCustomAudioRendering
使用 userSig 加入 TRTC 房间
欢迎语
开始对话
响应互动
数智人SDK
通过getCustomAudioRenderingFrame
获取到音频数据，传给数智人SDK来驱动
数智人SDK
停止获取音频数据，发送音频结束帧给数智人SDK
结束
结束 AI 对话
请求接口
StopAIConversation
停止 AI Bot
停止 AI Bot
退出 TRTC 房间
退出 TRTC 房间

2. 云端渲染流程图


客户端
客户Server
腾讯API
TRTC
AI Bot
发起 AI 对话
请求接口
StartAIConversation
要带上数智人相关参数
启动AI Bot
启动 AI Bot
返回结果
计算 userSig
开始
返回 userSig
使用 userSig 加入 TRTC 房间，并拉取数智人视频流
欢迎语
开始对话
响应互动
结束 AI 对话
请求接口
StopAIConversation
停止 AI Bot
停止 AI Bot
结束
退出 TRTC 房间
退出 TRTC 房间

轻量版-端云混合方案（不包含 ASR、AI 对话等能力）

最近更新时间：2026-04-02 15:46:53

应用场景

产品定义：我们为客户提供了轻量版云渲染形象（其实是端渲染形象在训练时候产生的特殊版本，表示其既可以用于端渲染也可以用于云渲染），我们在下文中统一称为轻量版形象。
基于产品定义，我们可以发现，端云混合方案就是为轻量版形象打造的一个独特场景。用户产品硬件性能足够的情况下，可以通过本地 CPU 进行端上渲染，最大限度减少带宽等成本；当用户产品硬件性能比较差的时候，可以通过云端服务的 CPU 来完成图像的渲染，此时会增加一个云服务渲染成本，但整体形象模型包还是同一个，保证渲染效果的同质化。通过此方案可以最大程度的适应各种终端硬件的差异化所带来的覆盖率问题。

服务资源清单

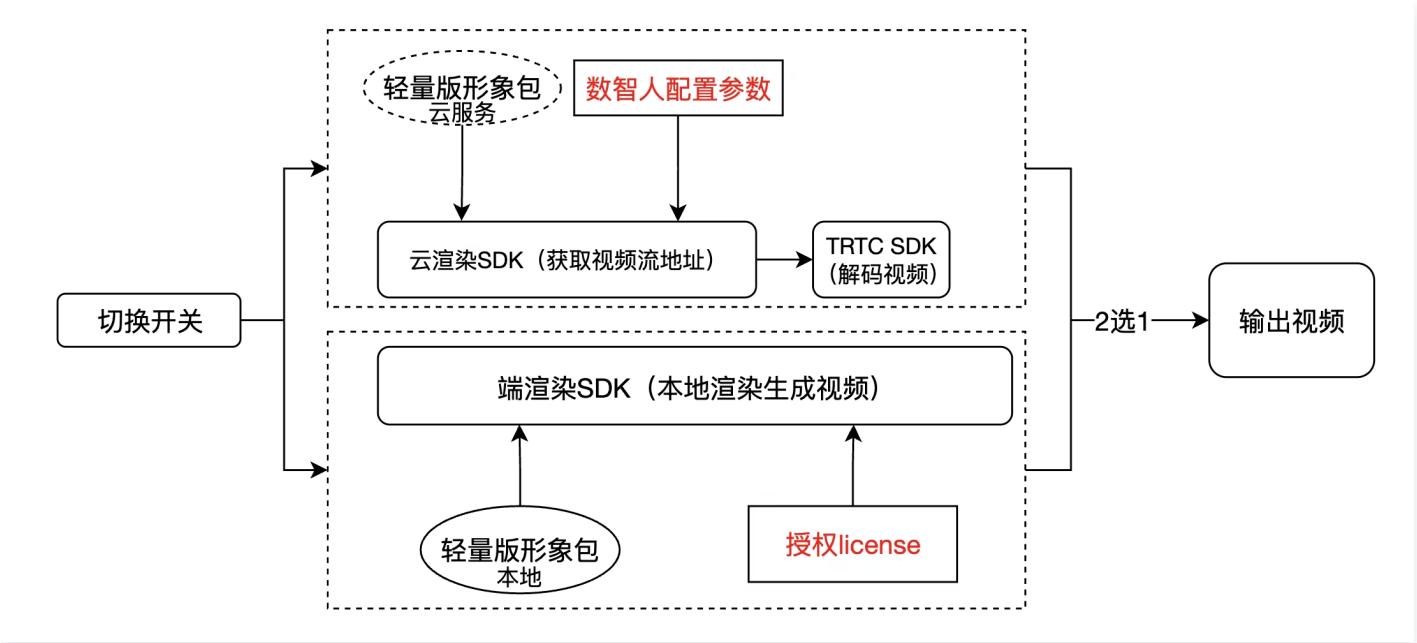
登录 购买页 购买相关服务资源，清单如下：

资源名目	数量	购买页示意图	备注指引信息
轻量版形象 必选	1		小样本通用口型、小样本专属口型、小样本照片三选一，购买完定制配额后，在 数智人平台 发起端渲染训练即可生成轻量版形象。 方式1：购买即训练，可单击**新增定制任务**，在后续流程中选择**端渲染**类型，训练的就包含有轻量版形象。 方式2：配额如已使用但未用于训练端渲染，可通过补训方式生成轻量版形象，找到已使用的训练单子，单击“+”补训“端渲染”。 **注意：**如果没有“+”，说明配额已使用超过半年以上，无法进行补训。 详细流程参考： 2D 端渲染形象定制及下载流程
轻量版云端服务小时包 可选	1		主要用于端云混合方案的云渲染场景下消耗云服务的费用抵扣。如果不打算使用云渲染，可以不购买。

端渲染 SDK License（订阅版） 可选	1		主要用于端云混合方案的端渲染场景下保证端渲染 SDK 可以正常运行。如果不打算使用端渲染，可以不购买。
轻量版形象续费 可选	1		云渲染场景下使用，需要保证轻量版形象在云端服务的有效期不能过期，可以通过购买形象续期来延长。（仅限 2D 小样本通用口型和 2D 小样本专属口型，2D 小样本照片无此限制）

调用方式

我们提供了专门的端云混合 Demo，供开发者快速体验、调试。



Demo 结构图

说明：

- 端云混合场景的云渲染场景，只能使用 720P 分辨率形象；端渲染场景，所有分辨率的形象均可使用。

- 驱动形式目前只支持音频驱动。如果需要文本驱动，则需要借助 TTS SDK 接入 。本 Demo 暂未集成。

下载 Demo

点此下载 switchdemo_android.apk

使用说明

1、购买或申请可用的端渲染 License 信息，包括 Licenseid 和 SecretKey，联系商务获取测试 License。
2、登录 数智人平台 ，在资源管理中心获取 AppKey、AccessToken 信息；继续选择形象资产，获取形象的 virtualmanKey 信息。（**注意，一定要选择渲染类型为“端渲染/云渲染”的形象信息。**）
3、将以上信息配置到 Demo 中即可开始体验。由于配置信息多，可以使用 Demo 内嵌的扫码工具，通过扫码配置，点击访问 二维码生成地址 。

注意：
端渲染数字人模型会在第一次渲染时自动下载，但我们对模型下载做了限制，默认下载期限为模型训练生成后7天内。若7天后才体验，会提示无法下载，导致渲染失败。