腾讯特效 SDK

Web 美颜特效

产品文档

腾讯云

文档目录

Web 美颜特效
产品概述

最近更新时间：2024-05-08 16:30:10

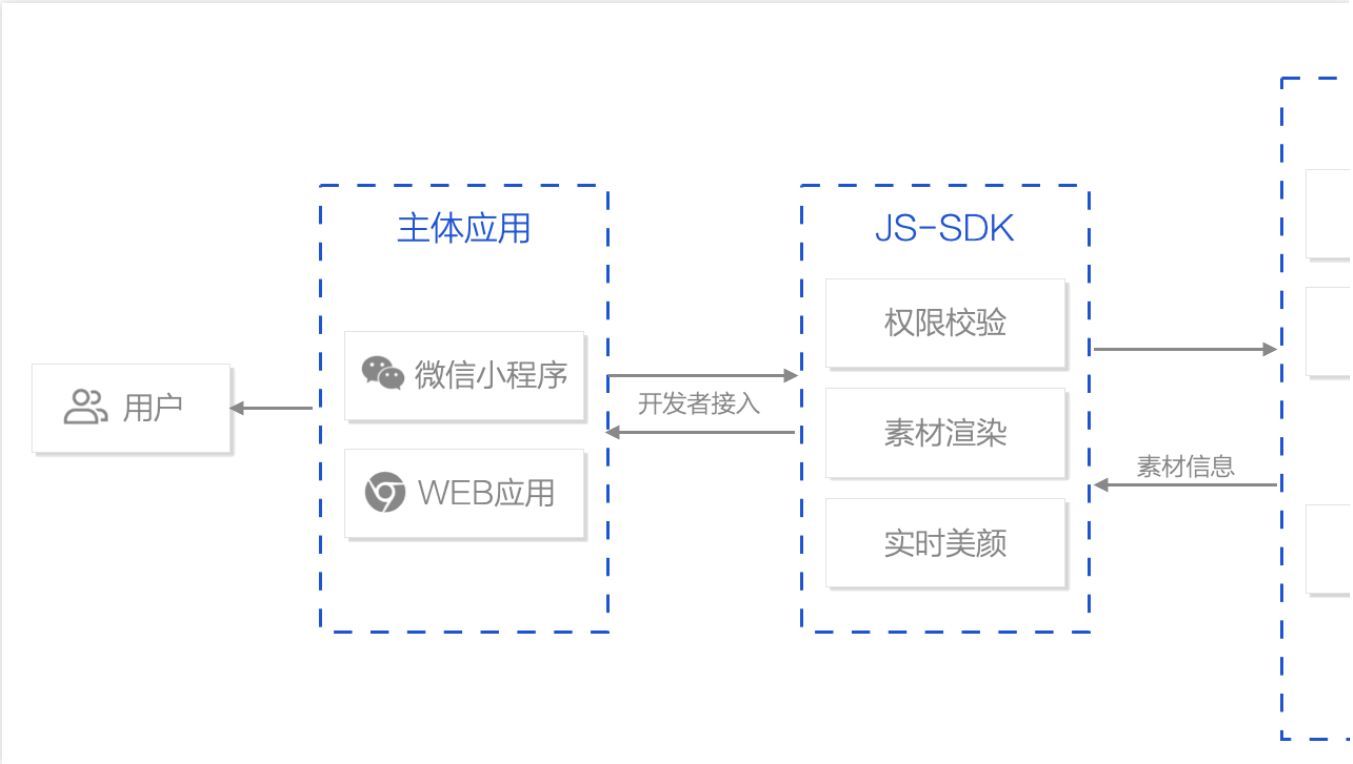
腾讯云视立方·Web 美颜特效是音视频终端 SDK（腾讯云视立方）的重要组成部分，提供在 Web 与小程序中实现美颜特效等 AR 效果的整体解决方案。
您可以使用本产品提供的 SDK 以及素材生产管理工具，轻松将 AI 美颜、滤镜、美妆、趣味贴纸、Animoji 表情、虚拟形象等 AR 效果添加到您的小程序、PC Web、移动 Web 页面中。

商业化版本

产品自2023年02月份起推出商业化版本，包含基础版和高级版，您可根据自身业务需求购买相应版本的License。
注意：
推广期免费申请的 License 到期后不再提供续期服务，如仍使用需要产品服务，请购买正式版本。

套餐类型	功能	域名支持	计费类型
基础版	美颜、美妆、滤镜、2D贴纸、虚拟背景	精准域名	按月订阅
高级版	美颜、美妆、滤镜、2D贴纸、虚拟背景 3D贴纸、Animoji 表情、虚拟形象	精准域名&泛域名	按年订阅

架构图



体验 Demo

平台	体验地址	环境说明
Web	单击体验	PC 环境推荐使用桌面端 Chrome 90+ 移动端使用微信扫码直接内置浏览器打开，建议您使用最新版本微信客户端
微信小程序		建议您使用最新版本微信客户端

快速接入

参考 快速上手 文档在几分钟内即可快速跑通 Demo 应用，体验 Web 美颜特效能力。

技术支持

有任何接入问题，请 联系我们。

快速上手

最近更新时间：2024-07-11 17:33:19

Web 美颜特效支持在 Web 网站以及微信小程序中实现美颜、滤镜、美妆、贴纸等功能。本文档将引导您快速地在本地跑通一个支持实时美颜的 Web 应用和微信小程序，您可以在此基础上根据相关文档实现更多功能。

注意：

此项目为测试项目，仅供本地测试使用。正式上线前请购买正式版License，并在产品控制台配置业务网站域名，详情参见 创建正式 LIcense。

前期准备事项

已开通 腾讯云服务。

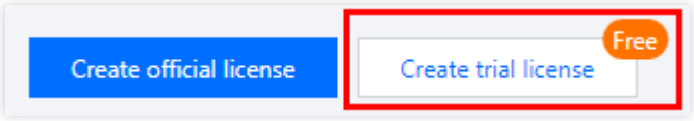
步骤1：创建 License

创建测试 License

登录 腾讯云视立方控制台 > Web端 License，单击**新建测试 License**。填写项目名称，域名输入本地开发用地址，此处以 `localhost:8090` 为例，单击确定完成创建。

Web License 针对域名和小程序 APPID 授权，此处为了帮助您快速在本地跑通 Demo 所以使用 `localhost` 作为本地开发用地址，实际项目中请填写业务站点域名。

如果需要跑通微信小程序，请填入自己的微信小程序 APPID（获取路径：**微信公众平台 > 登录小程序账号 > 设置 > 帐号信息**）。



注意：

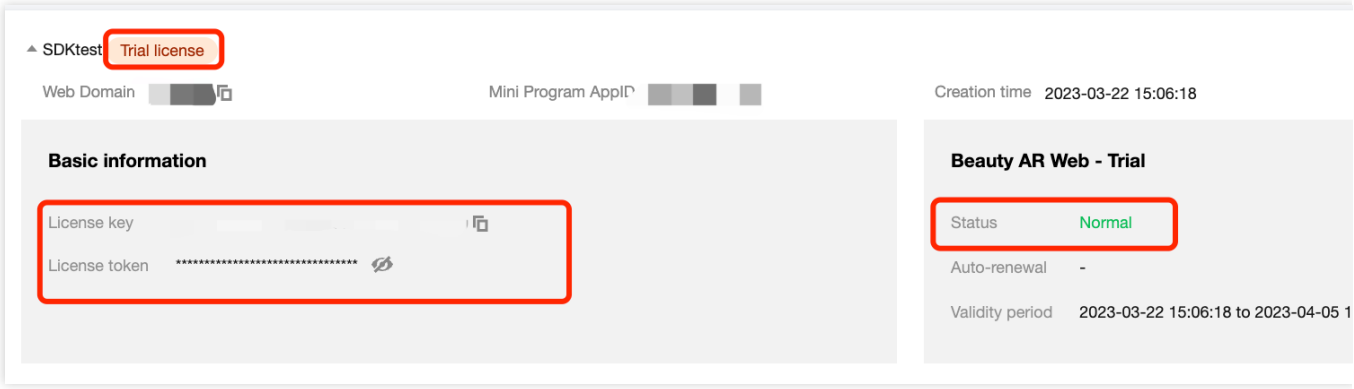
测试 License 有效期仅支持 14 天（可续期一次增加14天，共28天），最多可创建三个项目的测试License，正式项目请前往 Web License购买页 按需购买。

获取 License Key 和 Token

创建完成后可以看在项目列表中看到创建好的测试项目信息，获取 Web 美颜特效服务对应的**密钥 Token**，以及本测试项目的 **License Key**。

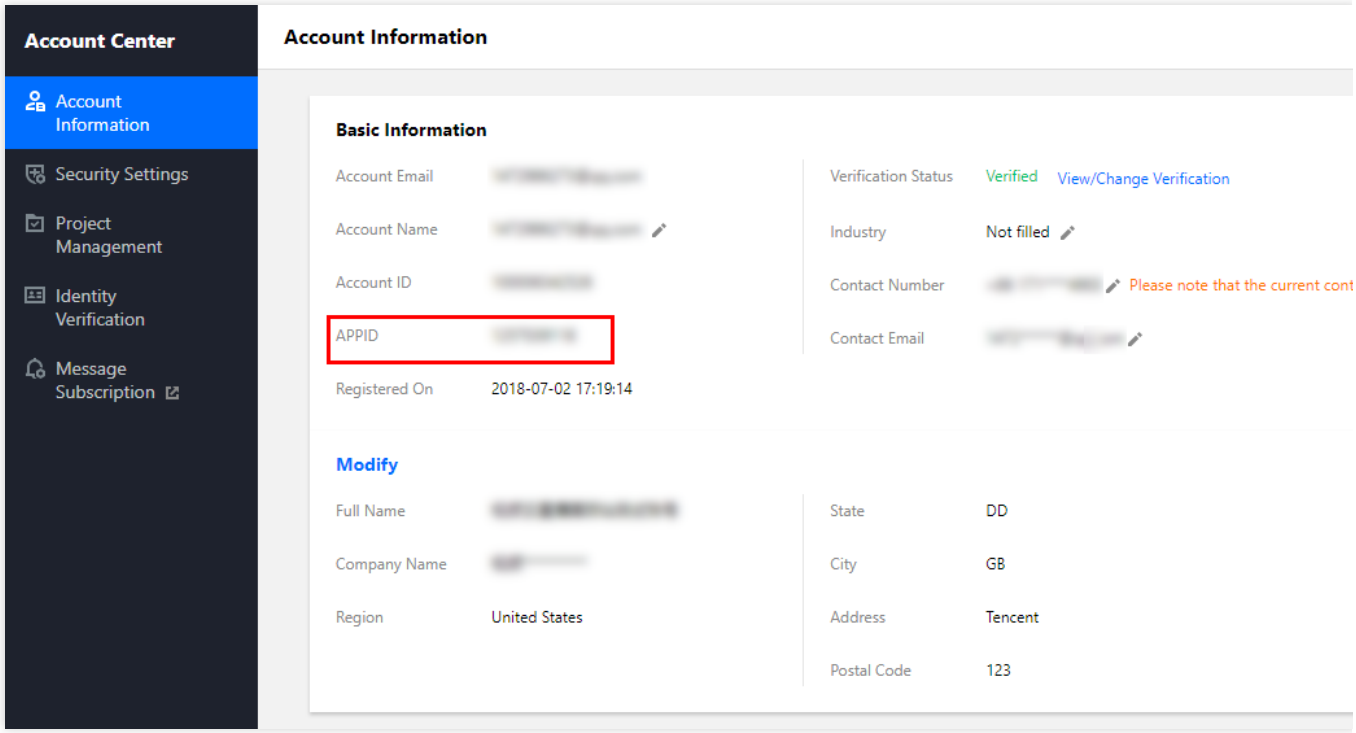
注意：

密钥 Token 用于计算鉴权签名，请一定妥善保管，此处在前端使用 Token 计算签名仅是为了帮助您在本地跑通 Demo，正式环境中需要迁移到服务端。



获取 APPID

从 腾讯云账号中心 获取 APPID。



步骤2：本地运行

Web License 同时支持 Web 网站和微信小程序，此处您可以根据自己的业务场景选择进行以下步骤。

微信小程序

1. 进入 微信公众平台 > **登录小程序账号** > **设置** > **帐号信息**，获取微信小程序 APPID。
2. 在 微信公众平台 添加小程序插件：**视立方 WEBAR**。
3. 拉取 示例项目 代码并导入**微信小程序开发工具**，并填入上述微信小程序对应的 APPID。

4. 进入 `miniprogram` 目录执行 `npm i` 安装依赖，然后进入微信开发者工具，选择 **工具 >构建 npm**（可参考 小程序使用 npm）。
5. 将 步骤1 中拿到的 License Key、Token 和 APPID 以及 第一步 中拿到的微信小程序 APPID 按下图所示替换到 `pages/camera/camera.js` 的指定配置项中。

```
/** ------ 鉴权配置 ------ */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APPID = '您的appid';

/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube/we
 */
const LICENSE_KEY = '您的licenseKey';

/**
 * 计算签名用的秘钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过接口提
 * [签名方法](https://https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.
 */
const token = '您的token';
```

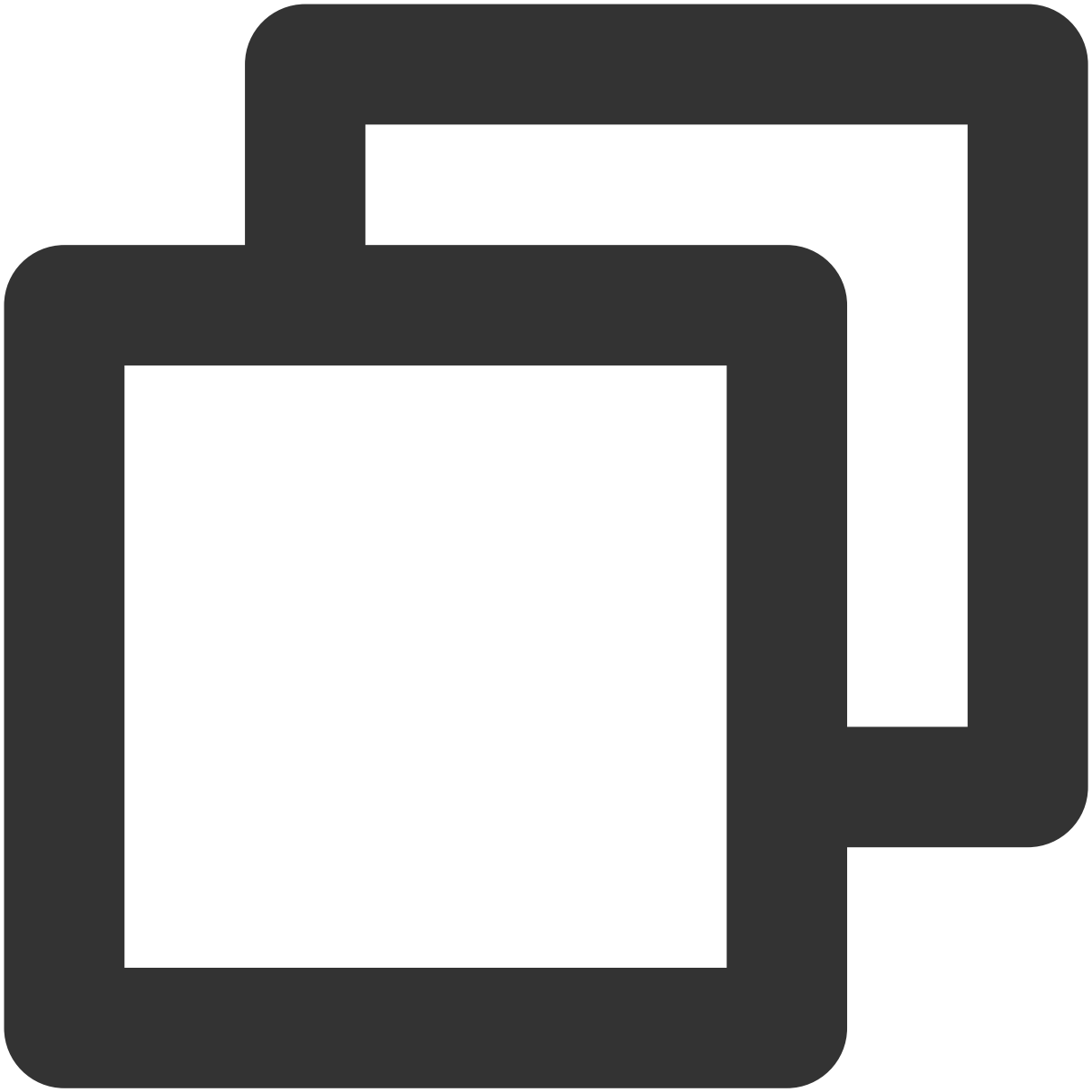
6. 单击**编译预览**，用手机扫码即可开始体验。

注意：

不要在开发者工具的模拟器中直接运行。由于 SDK 使用 webgl canvas 进行了重量级的渲染工作，目前**开发者工具不支持调试**，因此建议直接通过**编译** > **预览** > **手机扫码**的方式进行调试。

Web 网站

1. 拉取示例项目代码。



```
git clone https://github.com/tencentcloud-webar/web-demo-en.git
cd quick-start
```

2. 将 步骤1 中拿到的License Key、Token 和 APPID 按下图所示替换到 `index.js` 配置项中。

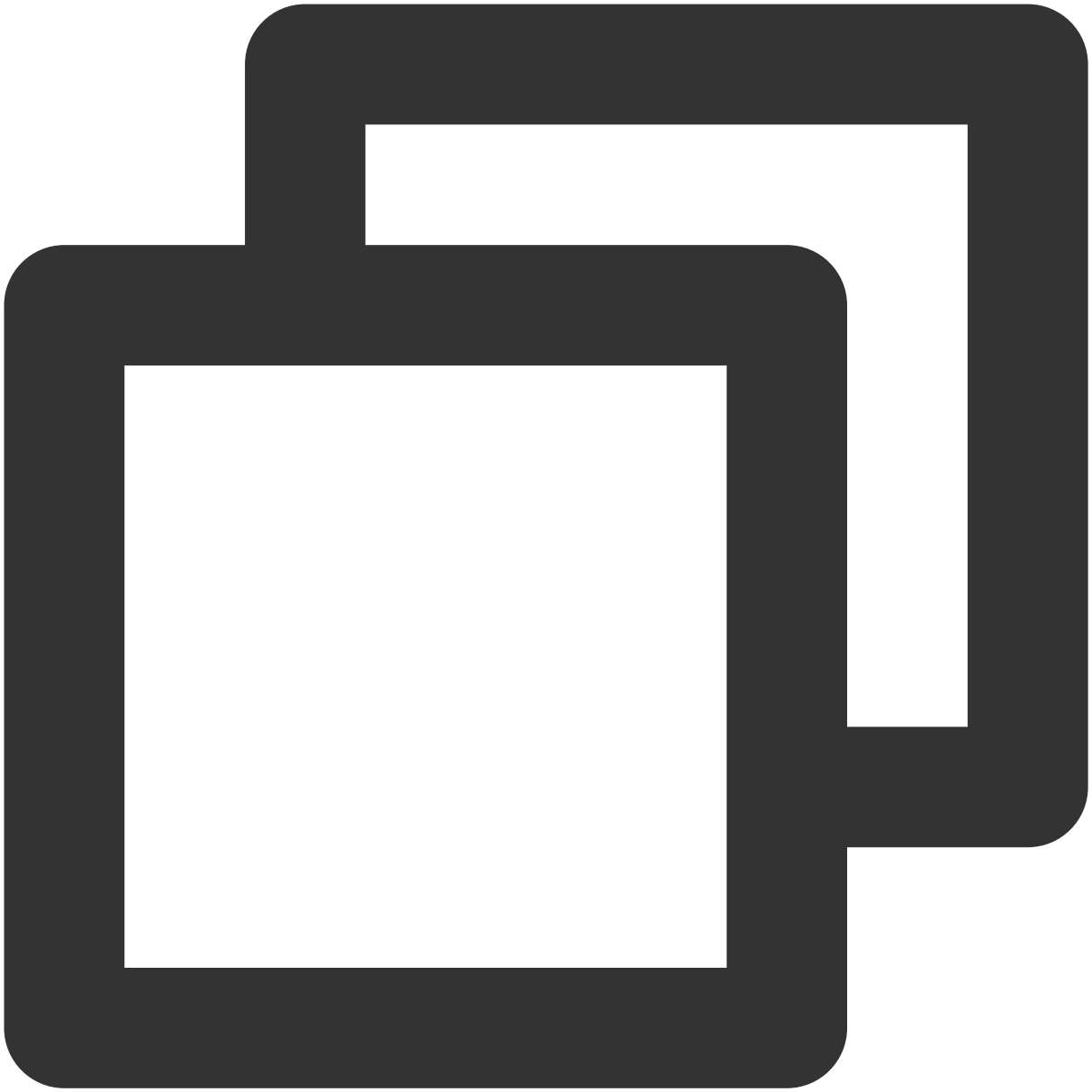
```
/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APPID = '您的appid';

/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube/we
 */
const LICENSE_KEY = '您的licenseKey';

/**
 * 计算签名用的秘钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过接口提
 * [签名方法](https://https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.
 */
const token = '您的token';
```

3. 本地开发环境运行。

注意：

本地运行前保证当前设备已安装 `nodejs` 环境。

项目目录下依次执行以下命令后，打开浏览器访问 `localhost:8090` 即可开始体验 Web 美颜特效能力。



```
# 安装依赖
npm i

# 编译运行
npm run dev
```

完成以上流程后，就可以分别在微信小程序或 PC 浏览器中体验 Web SDK 的美颜特效功能，您可以根据 SDK 接入指南，使用内置的素材体验各种美妆滤镜效果，或使用更多 Web 美颜特效的能力，如自定义贴纸、自定义美妆、自定义滤镜等，请参见 素材制作 制作属于您的特效素材 。

SDK 接入

开始

最近更新时间：2024-05-08 16:30:10

本文档指导您快速、安全地接入 **Web 美颜特效**，并使用相关功能。在接入使用过程中，您有任何疑问可进群进行技术咨询。

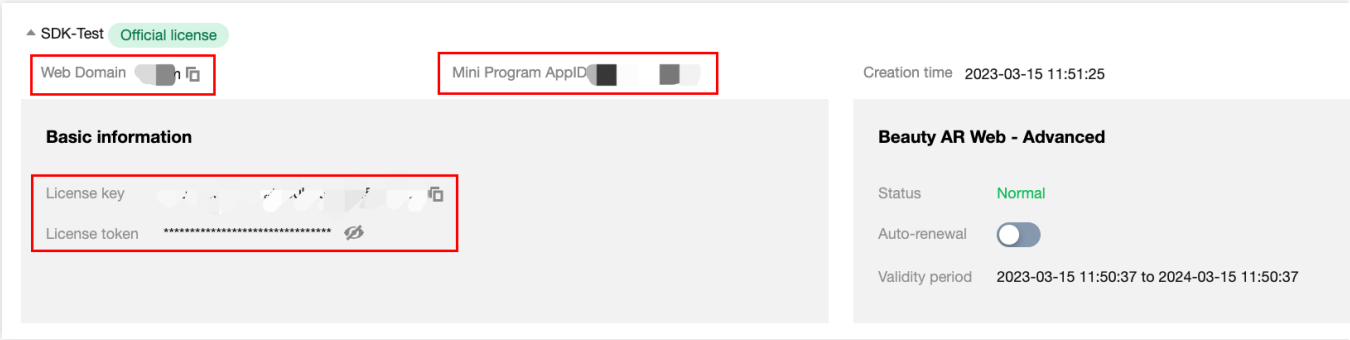
前期准备

准备接入 SDK 使用之前，需要确保您已购买 Web License 并创建项目，详情参见 Web端 License 新增与续期。

获取参数信息

获取 License Key 和 Token

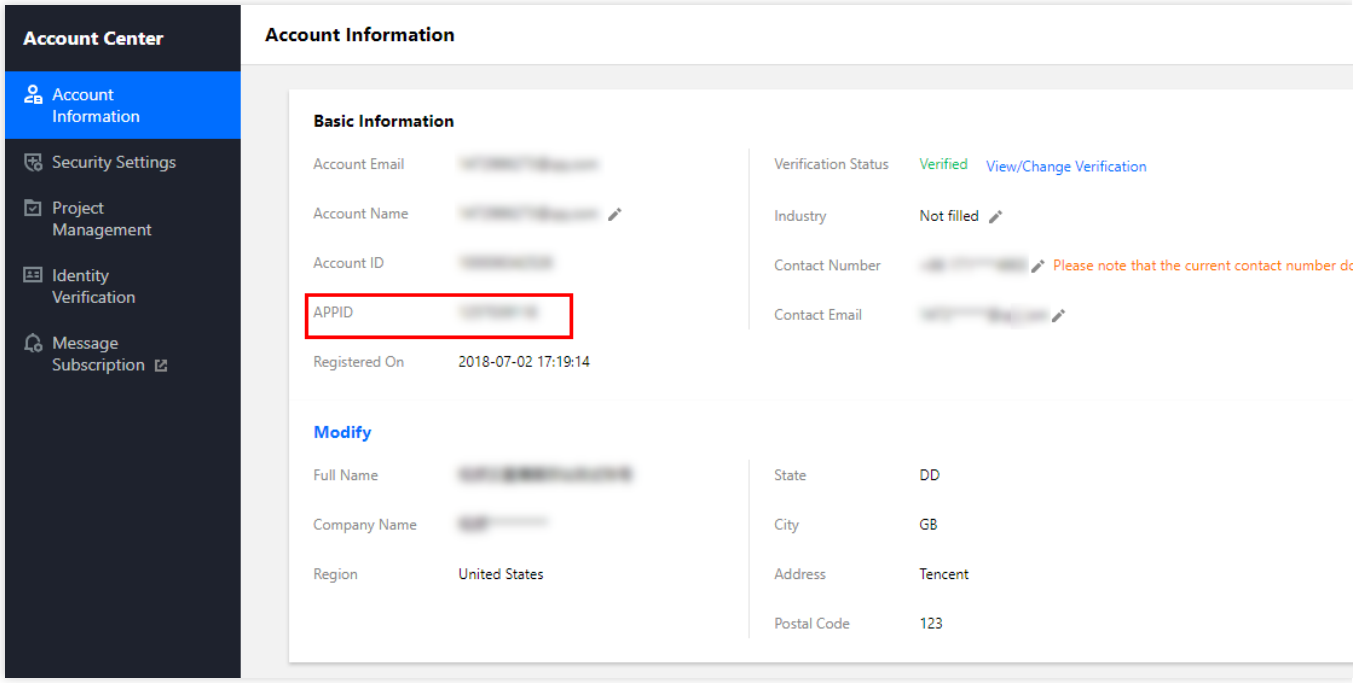
进入音视频终端 SDK 控制台 > License 管理 > Web License管理，查看您已创建的 Web License 并复制其 License Key 和 Token。



Web Domain：创建项目时填写的域名信息，只可以在该域名下使用此 License。
小程序 APPID：创建项目时填写的小程序信息，只可以在该小程序上使用此 License。
注意：
请确保使用与开发 Domain 和小程序匹配的 License Key 和 Token，否则鉴权会失败，无法正常初始化SDK。

获取 APPID

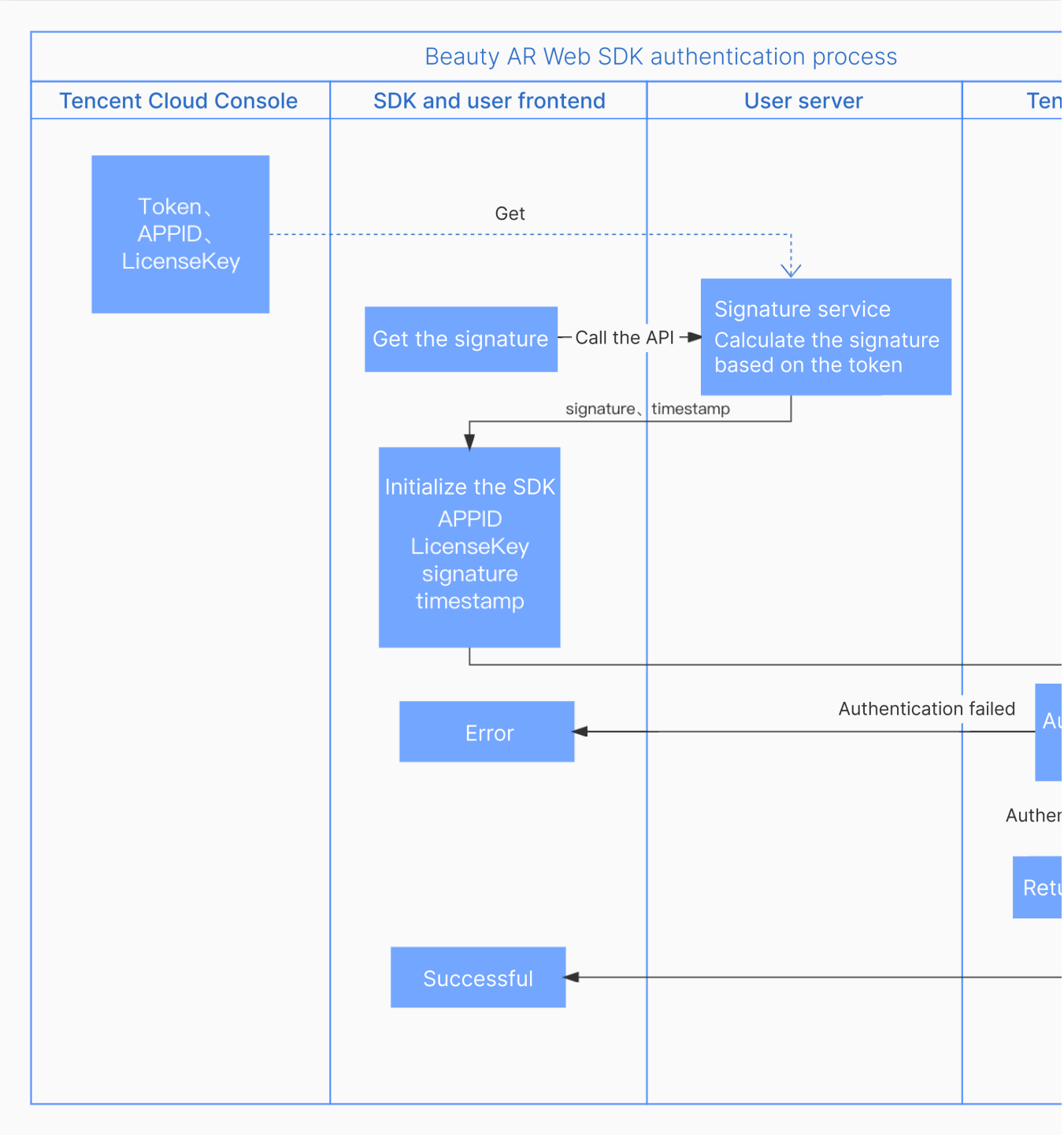
您可以登录腾讯云控制台，进入账号信息 > 基本信息 查看 APPID。



准备签名信息

除了需要上述 License Key 授权 SDK 外，还需要使用 Token 对 SDK 中调用的接口进行签名。

签名方法鉴权流程



Token：用于 SDK 接口签名，是您身份的唯一标识。

appid：即腾讯云控制台 账号信息 > 基本信息 展示的 APPID。

timestamp：当前时间戳，精确到秒（10位数字）。

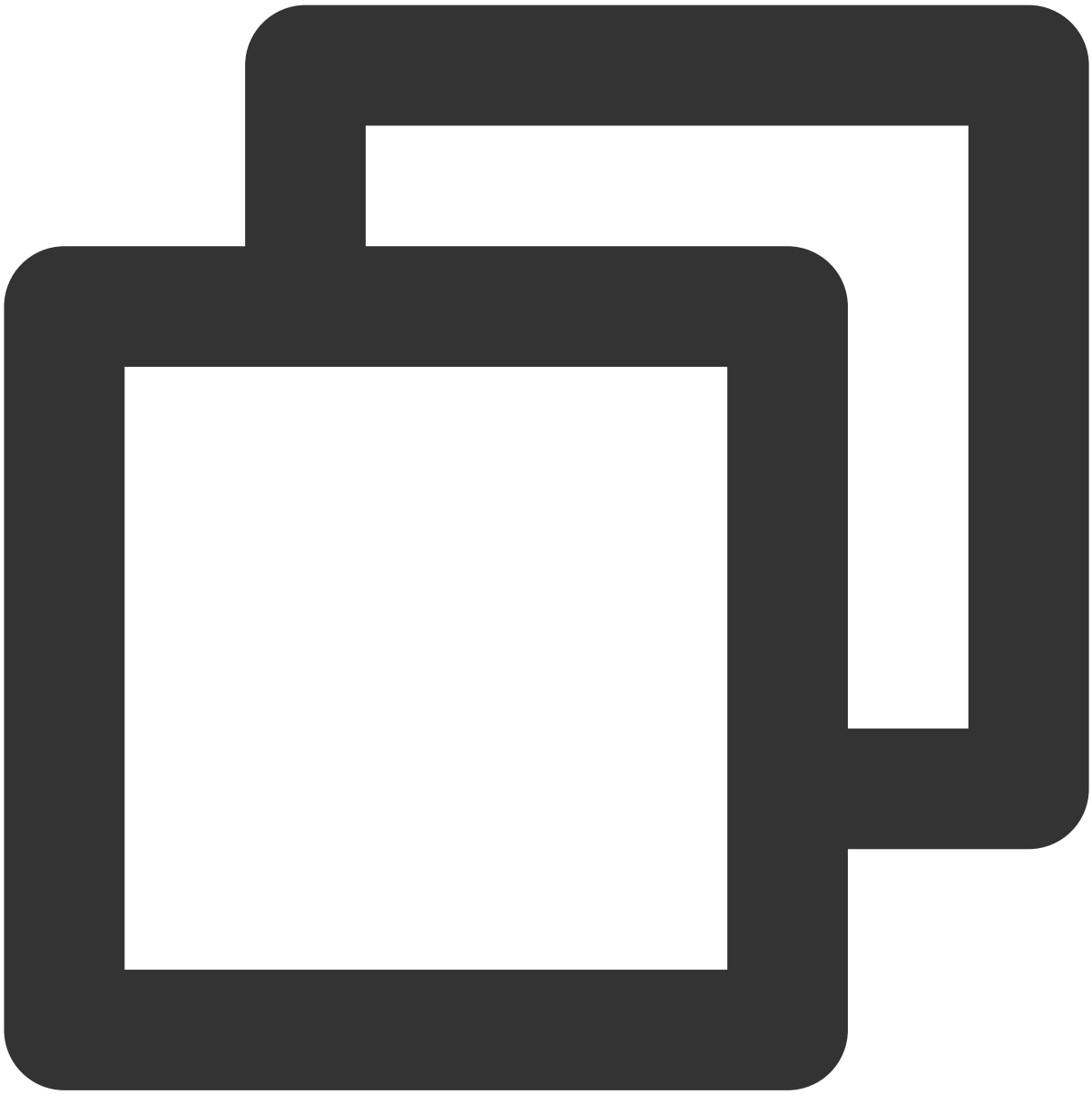
signature：签名（签名有时效性，目前5分钟过期）。

部署签名服务

由于 signature 有时效性，且需要防止 Token 泄露，您需要部署一个生成签名的服务。

注意：

如果 Token 泄露您的身份会被盗用，您的资源会泄露。
生成签名的方法放在前端会导致 Token 泄露，为了避免利益受损，建议您不要将生成签名的方法放在前端。



```
// 以express后台为例
// 签名方法： sha256(timestamp+token+appid+timestamp)

const { createHash } = require('crypto');
const config = {
    appid: '您的腾讯云APPID',
    token: '您的Token',
}
```

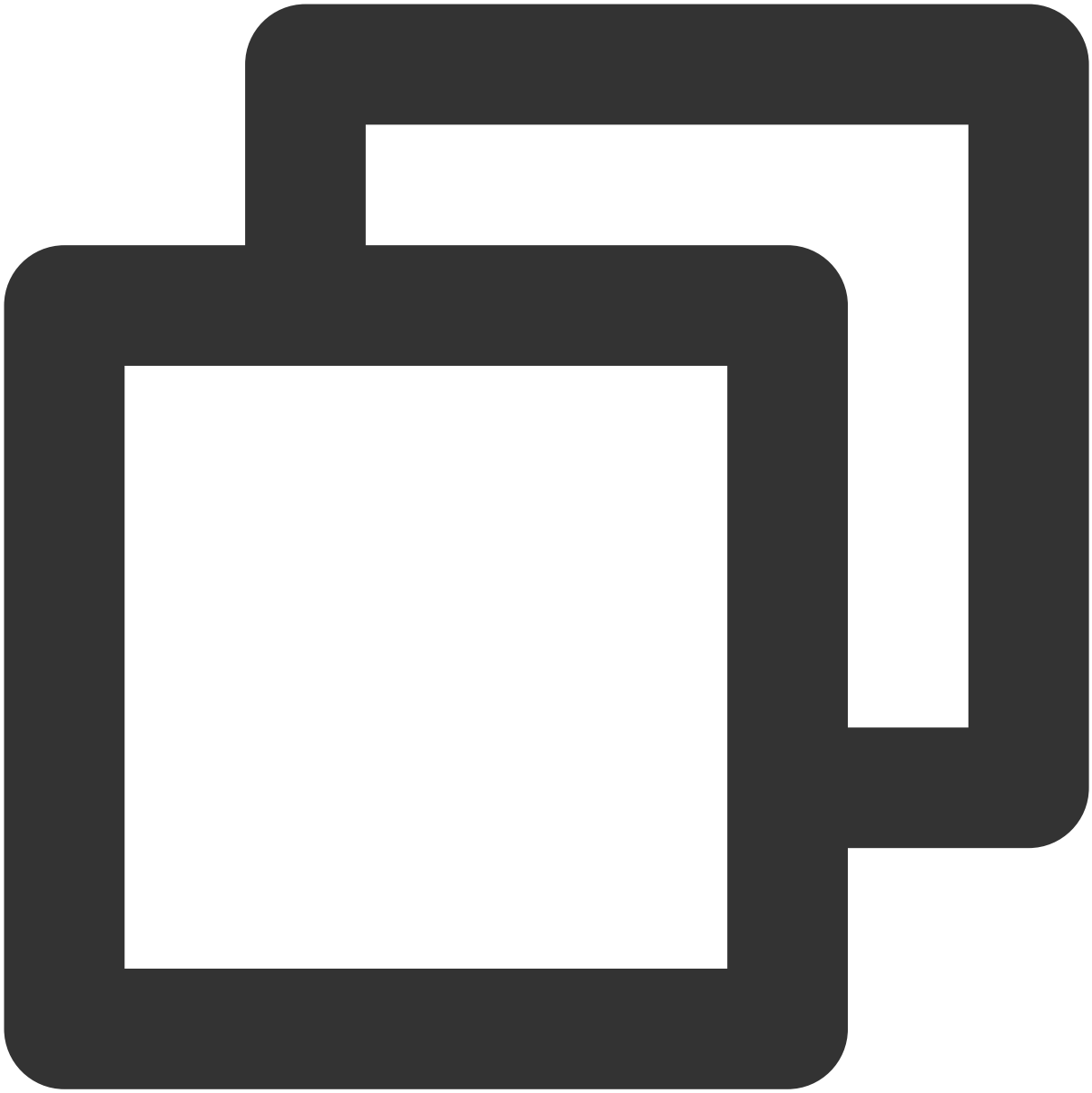
```
const sha256 = function(str) {
    return createHash('sha256')
        .update(str)
        .digest('hex');
}

const genSignature = function() {
    const timestamp = Math.round(new Date().getTime() / 1000);
    const signature = sha256(timestamp + config.token + config.appid + timestamp).t
    return { signature, timestamp };
}

app.get("/get-ar-sign", (req, res) => {
    const sign = genSignature();
    res.setHeader('Access-Control-Allow-Origin','*');
    res.setHeader('Access-Control-Allow-Methods', 'GET, OPTIONS');
    res.send(sign);
})
```

前端调用签名服务

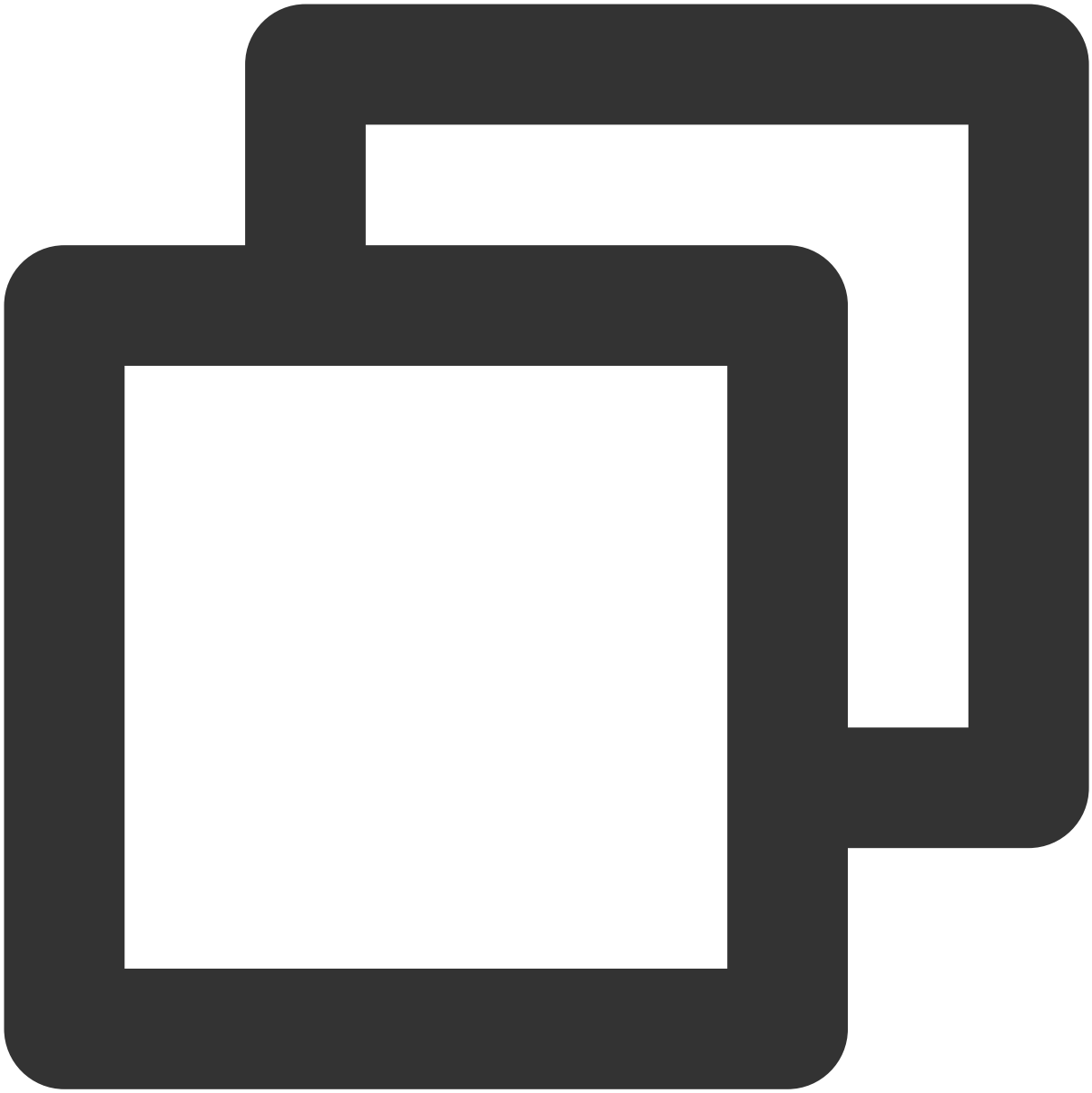
获取签名的服务部署完成后，在您的 Web 页面或小程序中，添加一个获取签名的方法供 SDK 接入调用。

Web

小程序



```
async function getSignature() {
    const res = await fetch('您的域名/get-ar-sign')
    const authdata = await res.json()
    console.log('authdata',authdata)
    return authdata
}
```



```
async function getSignature() {
    return new Promise((resolve,reject)=>{
      wx.request({
        url: '您的域名/get-ar-sign',
        method: 'GET',
        success(res) {
          console.log('getSignature ok', res)
          resolve(res.data);
        },
        fail(e){
            console.log('getSignature error', e)
```

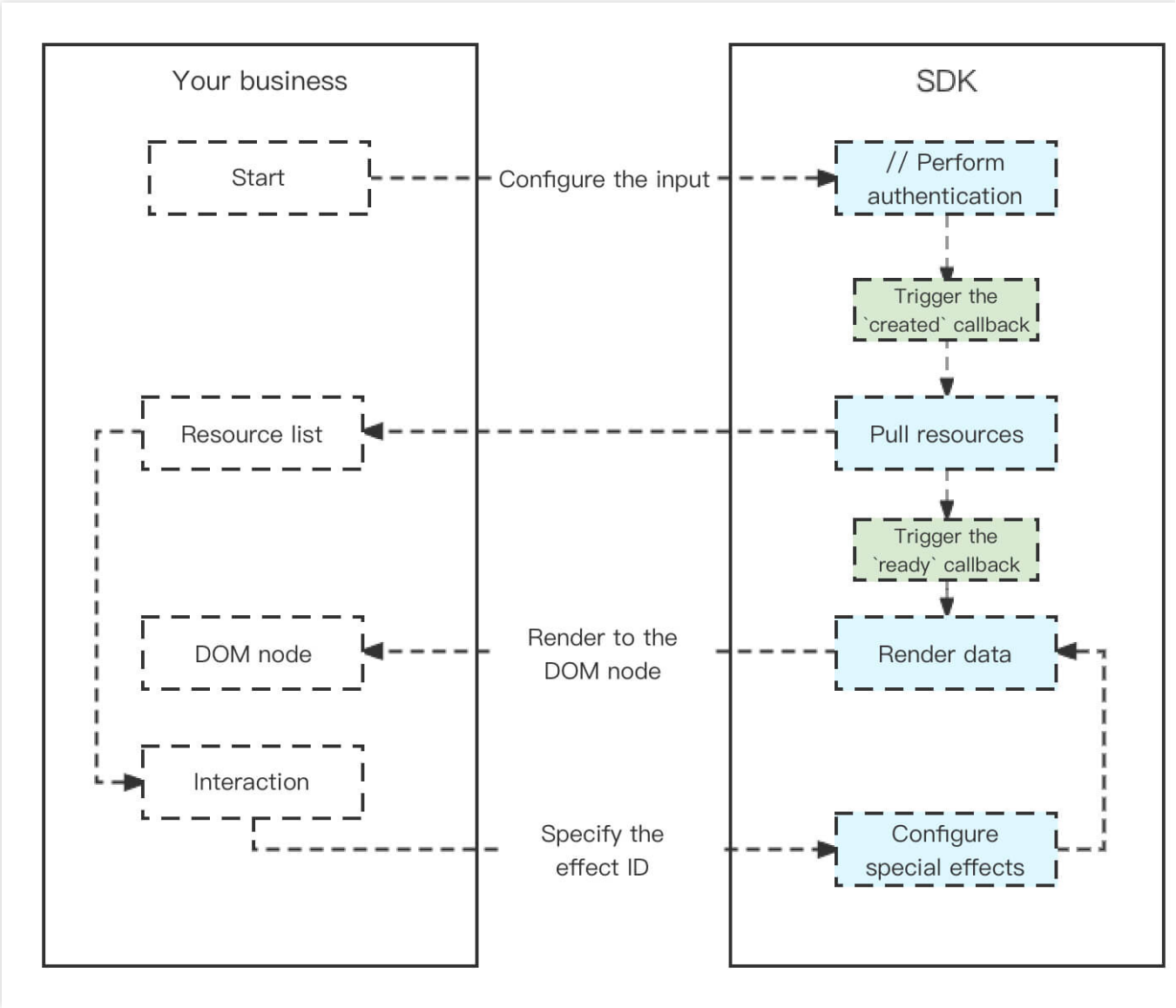
```
            }
          })
        })
    }
```

SDK 接入

上述准备工作完成后，便可根据如下流程接入并使用 SDK。

流程说明

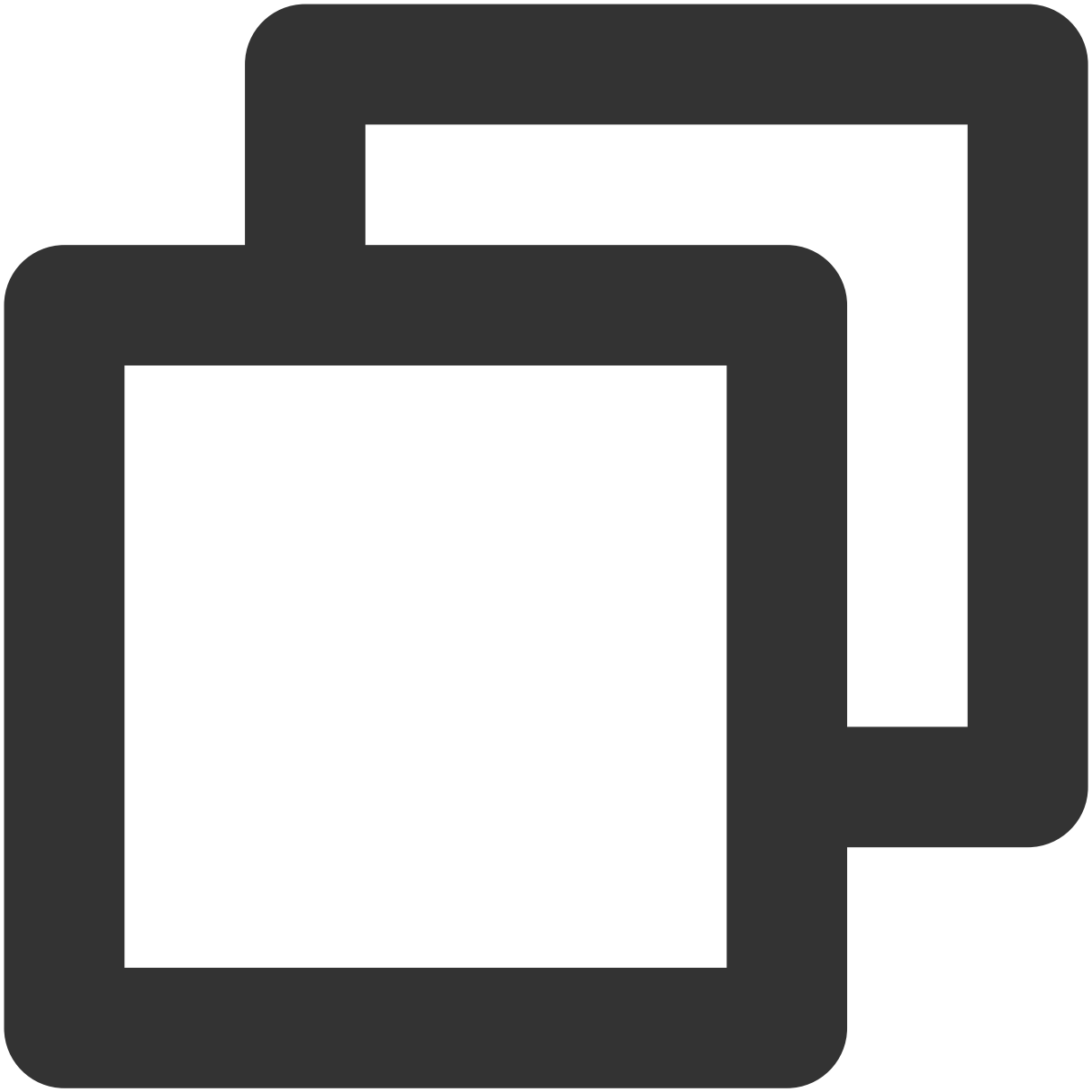
SDK 实现了简洁低侵入性的接口提供接入，您只需要初始化实例，将渲染节点添加到自己的页面即可快速实现 **Web 美颜特效**功能。



安装 SDK

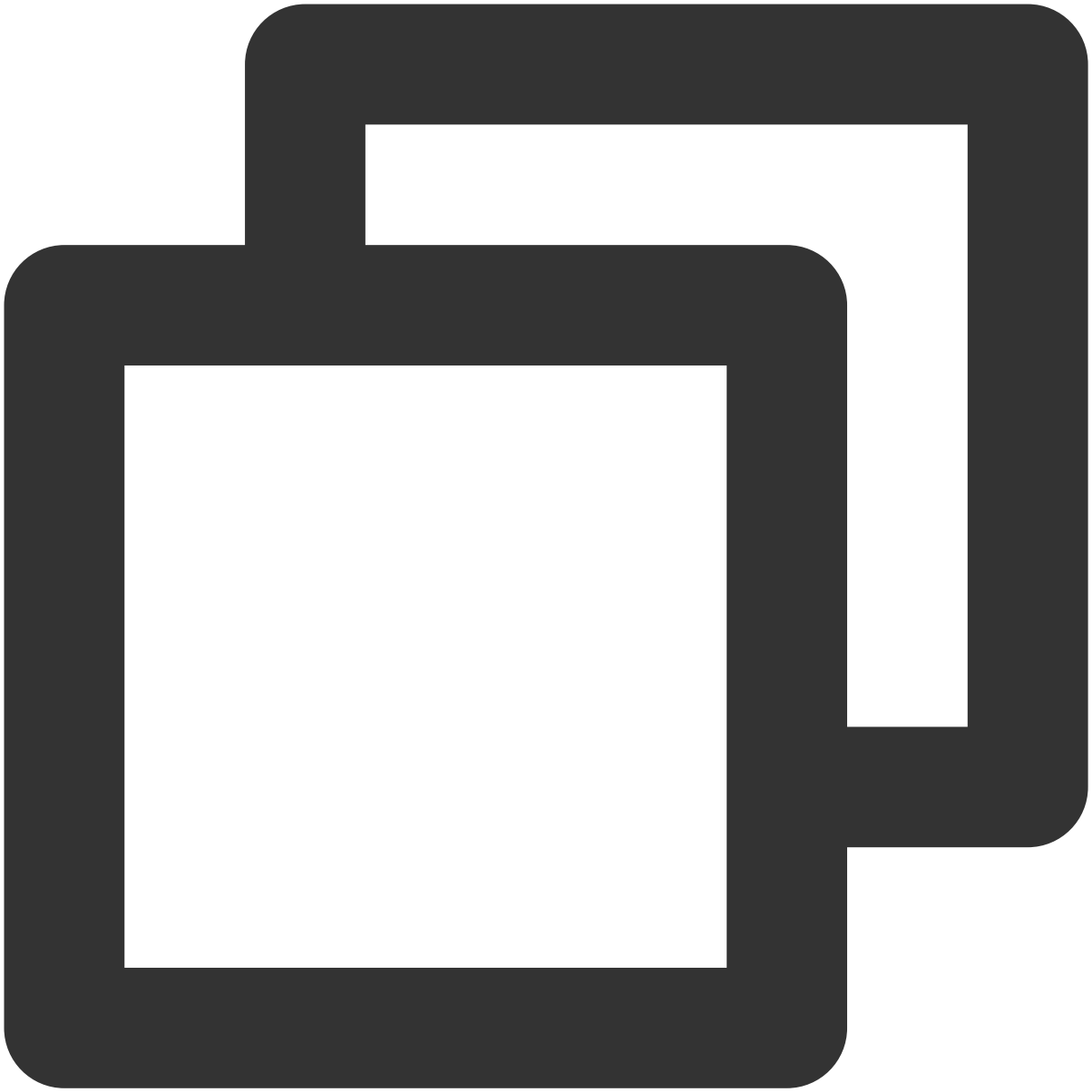
本 SDK 使用 npm 包提供 Web 端与微信小程序的 SDK。

Web端 SDK



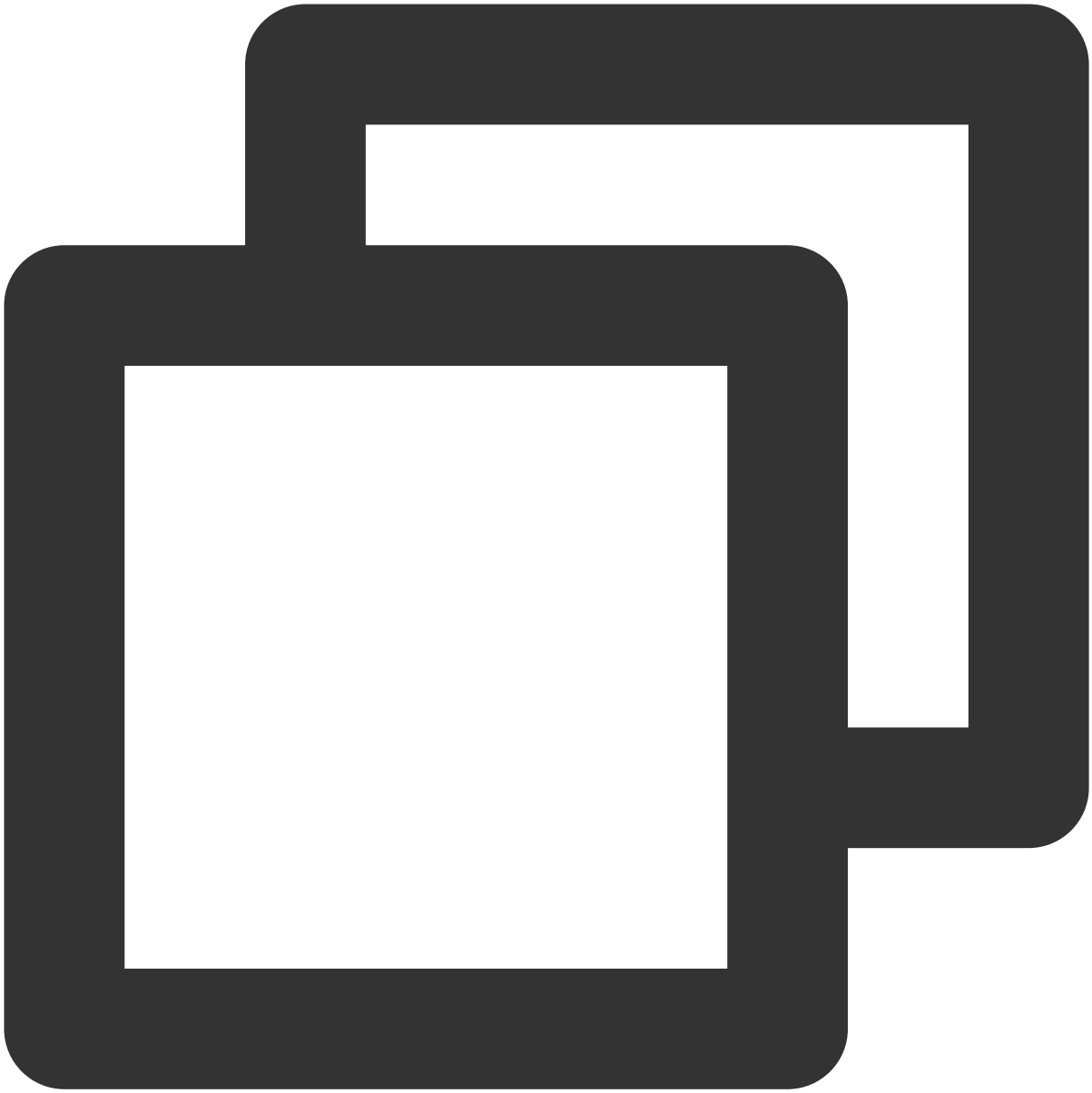
```
npm install tencentcloud-webar
```

此外，也可以通过引入JS的方式使用，可以根据自身项目灵活选择。



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

微信小程序端 SDK



```
npm install tencentcloud-webar-wx
```

初始化 SDK

本 SDK 在微信小程序和 Web 端的初始化方式略有差异。

小程序接入请参见 小程序端接入。

Web 端我们提供了内置 Camera 与自定义流两种初始化方式。

内置相机与播放器：使用了内置的摄像头与播放器封装，调用简单迅速，交互功能丰富。

自定义流接入：适用于有自己维护媒体流的需求或者要求更高的灵活性与可控性的场景。

使用 SDK

设置美颜和特效

SDK 的所有素材均兼容微信小程序端与 Web 端，调用方式一致，详情可参见 设置美颜和特效。

人像分割（0.2.0版本新增）

开启人像分割可以支持设置背景功能，仅在 Web 端支持，详情请参见 使用人像分割。

3D特效（0.3.0版本新增）

同时支持微信小程序端与 Web 端，调用方式同设置特效一致，详情请参见 设置美颜和特效。

Animoji 表情与虚拟形象（0.3.0版本新增）

依赖 WebGL2 运行环境，目前仅支持 Web 端，详情请参见 使用表情和虚拟形象。

参数与方法说明

请参见 API 文档。

代码示例

请参见 快速上手。

集成 Web 美颜特效

最近更新时间：2023-04-11 16:11:42

独立集成

Web网页接入可参见 Web 端接入
小程序接入可参见 小程序端接入

场景化集成

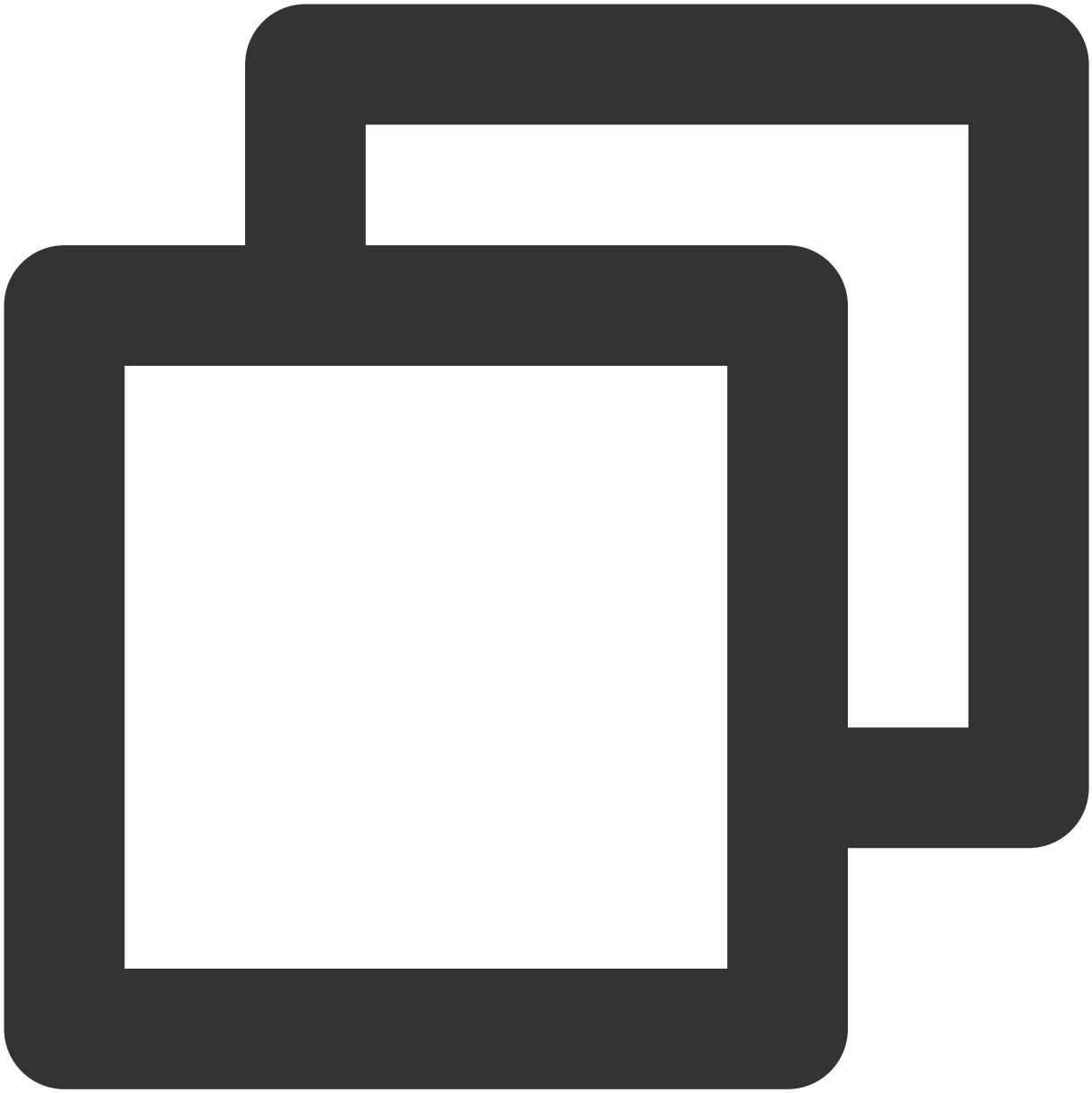
结合WebRTC、TRTC 接入 Web 美颜特效可参见以下文档：
结合 WebRTC 的推流
结合 WebRTC 的推流（预初始化方案）
结合 TRTC 推流

Web 端接入
内置相机

最近更新时间：2023-03-28 16:37:22

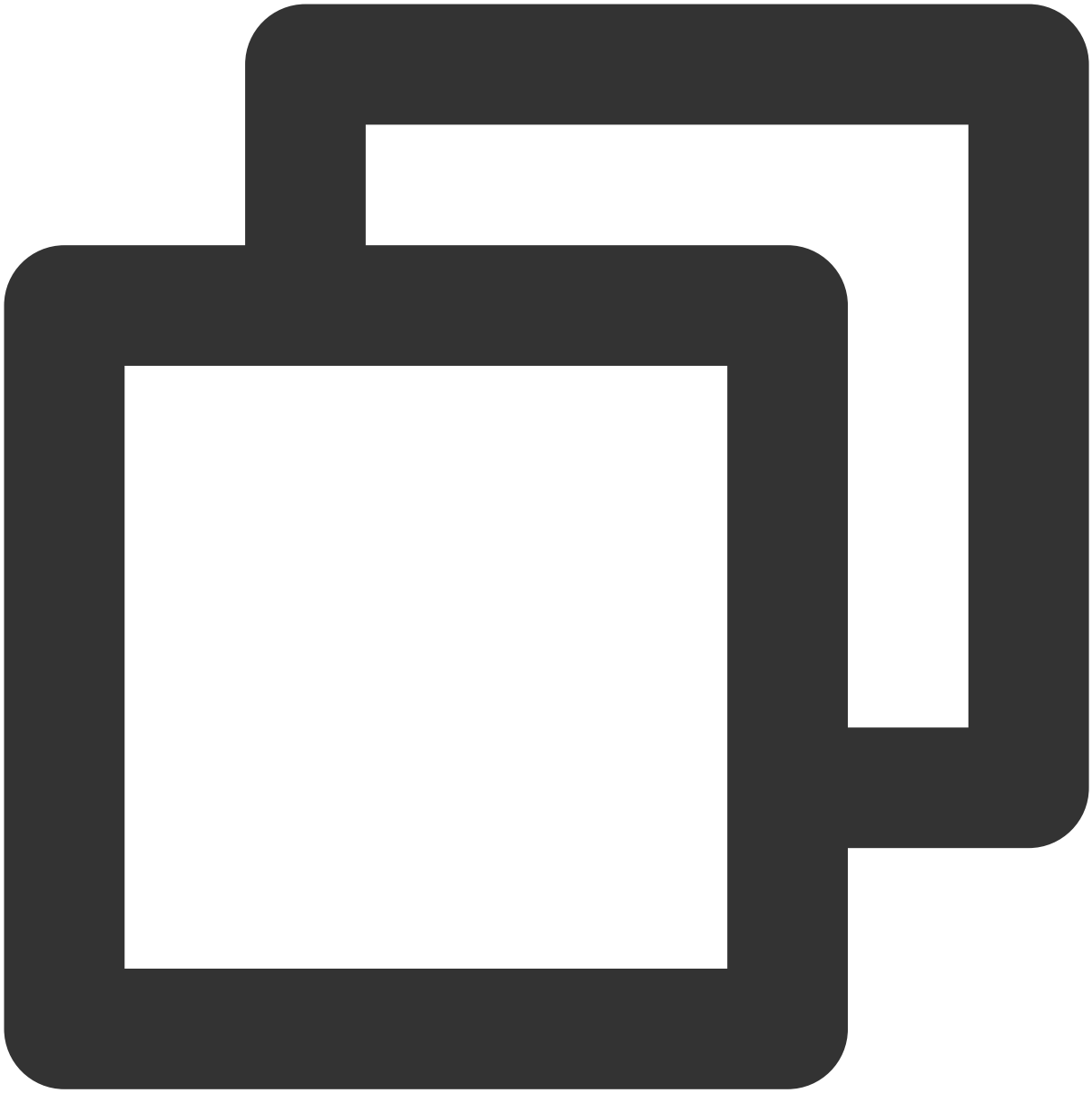
SDK 提供了内置相机，如开发者需要快速调起摄像头，或对摄像头有较多交互控制，推荐使用内置相机方式。

步骤1：引入 SDK



```
import { ArSdk } from 'tencentcloud-webar';// SDK 类
```

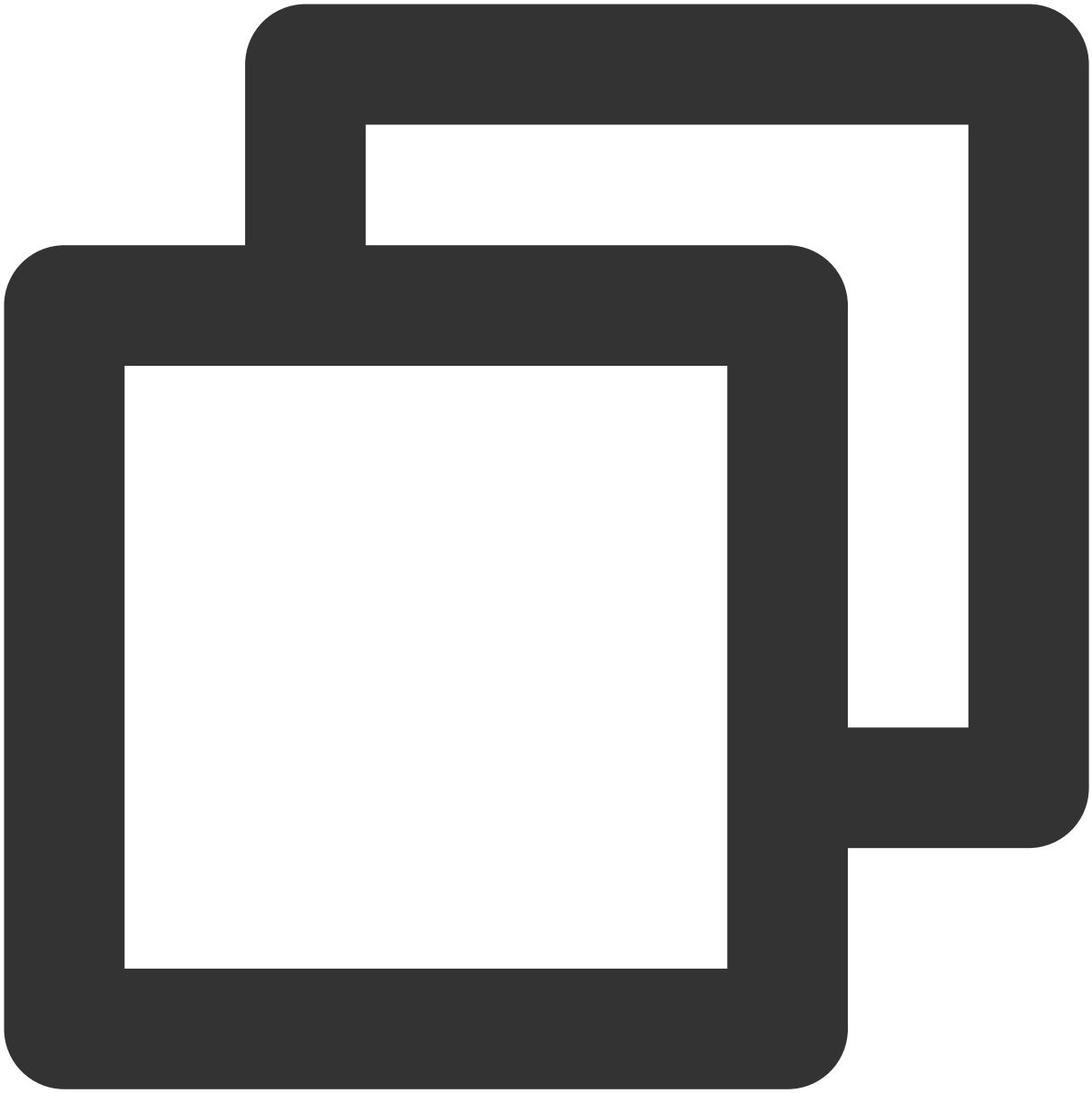
如果项目无需编译，则可以直接以如下方式引用：



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

步骤2：初始化实例

接下来初始化一个 SDK 实例用于显示：



```
// 获取鉴权参数
const authData = {
    licenseKey: 'xxxxxxxxx',
    appId: 'xxx',
    authFunc: authFunc // 详见「License 配置与使用 - 签名方法」
};

const config = {
    module: { // 0.2.0版本新增
        beautify: true, // 是否启用美颜模块，启用后可以使用美颜、美妆、贴纸等功能
        segmentation: true // 是否启用人像分割模块，启用后可以使用背景功能
```

```
    },
    auth: authData, // 鉴权参数
    camera: { // 传camera配置调起内置相机
        width: 1280,
        height: 720
    },
    beautify: { // 初始化美颜参数(可选)
        whiten: 0.1,
        dermabrasion: 0.3,
        eye: 0.2,
        chin: 0,
        lift: 0.1,
        shave: 0.2
    }
}
const sdk = new ArSdk(
    // 传入一个 config 对象用于初始化 sdk
    config
)

let effectList = [];
let filterList = [];

sdk.on('created', () => {
    // 在 created 回调中可拉取特效和滤镜列表供页面展示，详见「SDK接入 - 参数与方法」
    sdk.getEffectList({
        Type: 'Preset',
        Label: '美妆',
    }).then(res => {
        effectList = res
    });
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})

// 在 ready 回调中调用 setBeautify/setEffect/setFilter 等渲染方法
// 详情可参见「SDK接入 - 设置美颜和特效」
sdk.on('ready', () => {
    // 设置美颜
    sdk.setBeautify({
        whiten: 0.2
    });
    // 设置特效
    sdk.setEffect({
        id: effectList[0].EffectId,
        intensity: 0.7
```

```
    });
    // 设置滤镜
    sdk.setFilter(filterList[0].EffectId, 0.5)
})
```

注意

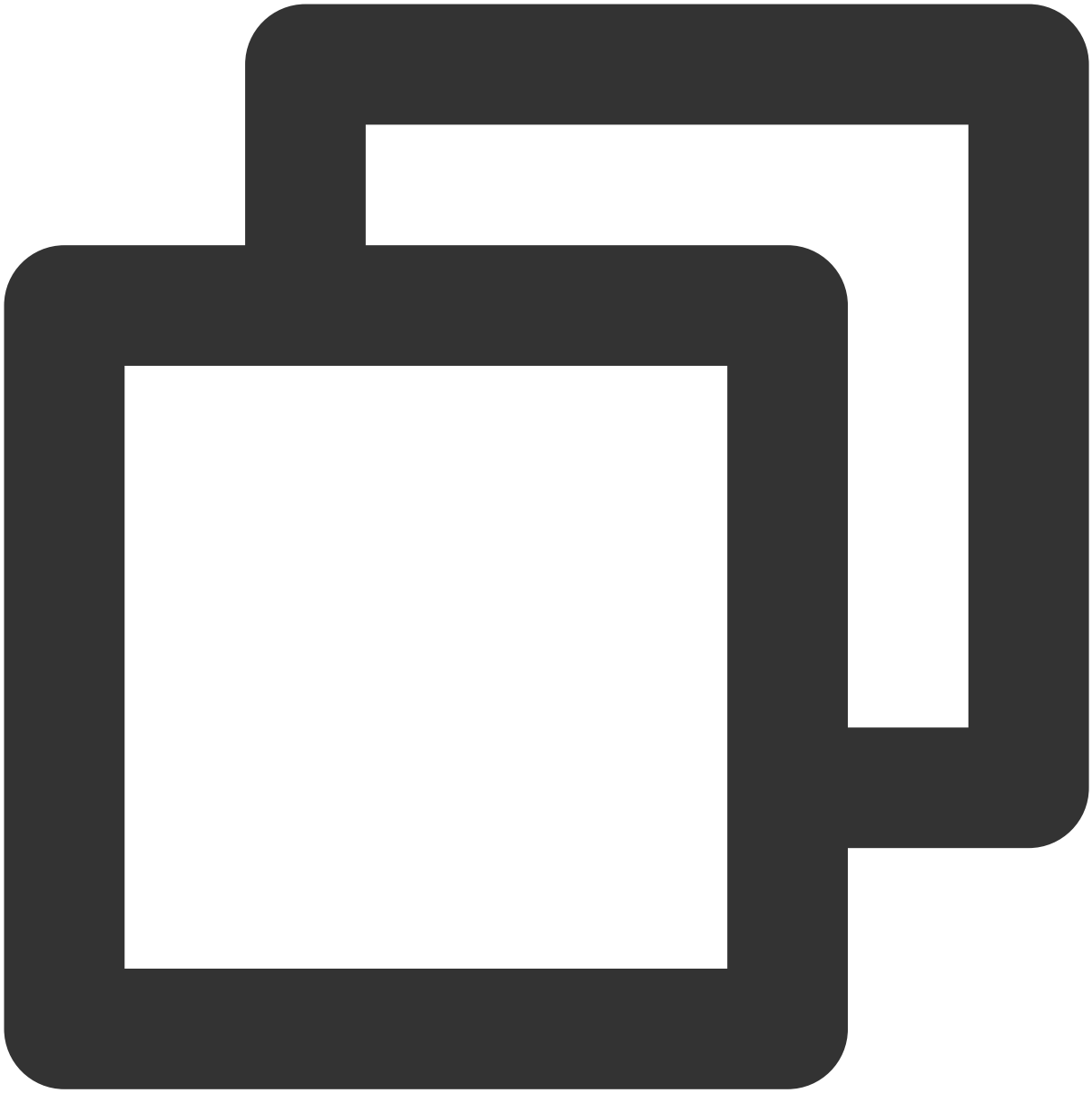
由于美颜检测和人像分割均有一定的加载耗时和资源消耗，初始化配置中提供了模块配置以供选择需要的功能，关闭的模块将不会进行预加载和初始化。

config 中的 camera 参数表示将由 SDK 来采集当前设备的摄像头媒体流作为输入。当设置了 camera 参数，SDK 还提供一系列基础设备管理方法，详情请参见 设备控制。

步骤3：播放流

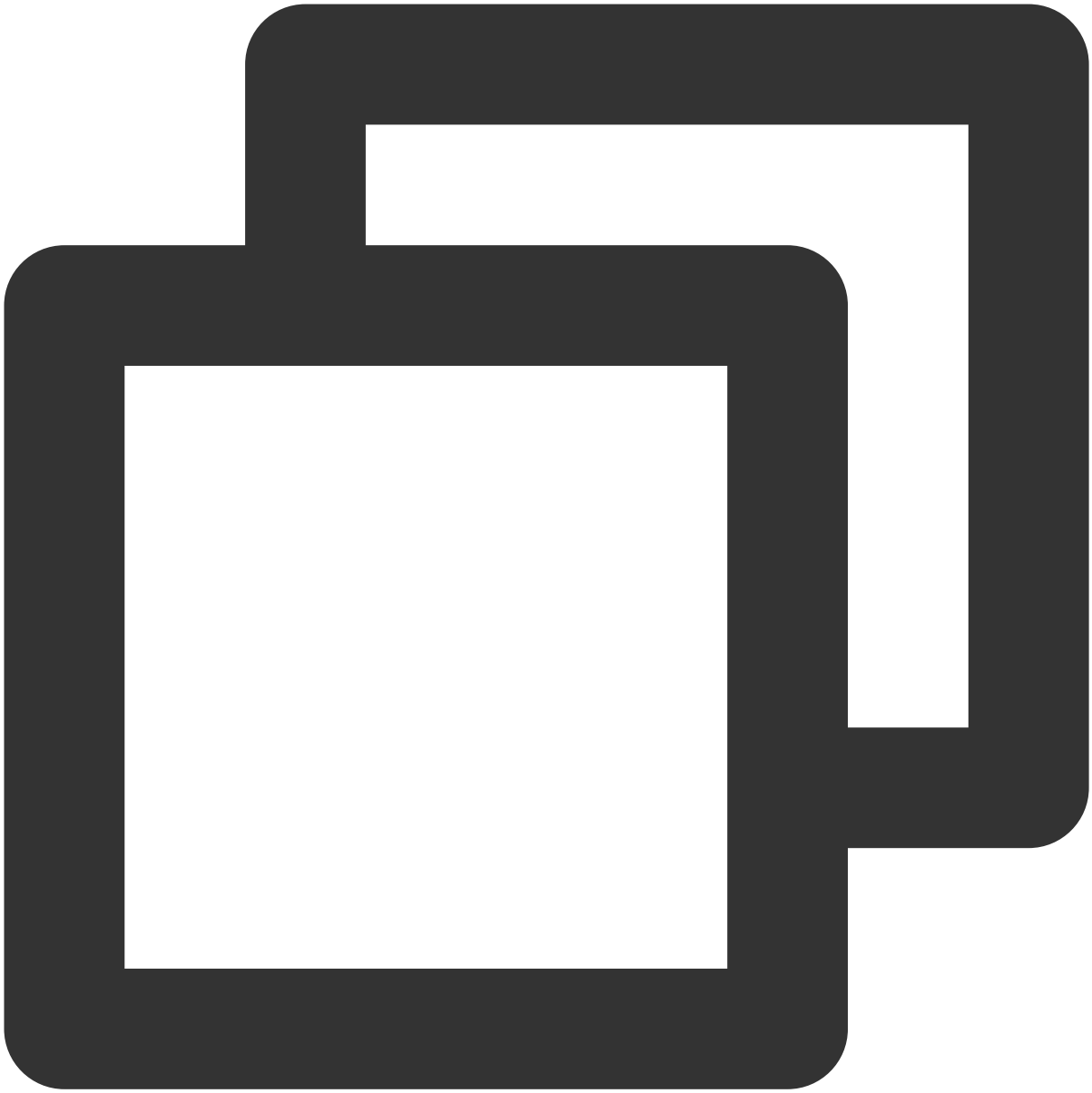
SDK 提供了一个快速在页面预览输出效果的播放器，可用于本地预览效果场景，不同的回调事件中获取的 player 有些许差别，适用于不同的场景，根据自身业务需求选择一种处理方式即可。

如果对画面展示的时效性有需求，可以在 `cameraReady` 回调中获取播放器，此时 SDK 尚未开始资源加载和检测的初始化，仅能展示原始画面。



```
sdk.on('cameraReady', async () => {
  // 初始化一个SDK的player，其中my-dom-id表示您需要放置播放器的容器id
  const player = await sdk.initLocalPlayer('my-dom-id')
  // 直接播放
  await player.play()
})
```

如果需要检测初始化完成，且美颜生效之后再播放，则可以在 ready 回调获取播放器。



```
sdk.on('ready', async () => {
  // 初始化一个SDK的player，其中my-dom-id表示您需要放置播放器的容器id
  const player = await sdk.initLocalPlayer('my-dom-id')
  // 直接播放
  await player.play()
})
```

注意

initLocalPlayer 获取到的播放器默认静音，如果取消静音则可能产生回声。

获取到的 player，内部封装了 `sdk.getOutput()` 方法，方便用户使用。

SDK initLocalPlayer 后，其 player 对象封装了一些常用接口，支持以下方法：

方法名	说明	入参	返回值
play	播放	-	Promise;
pause	暂停（流不会停止，可恢复）	-	-
stop	停止（会把流也停止）	-	-
mute	静音	-	-
unmute	取消静音	-	-
setMirror	设置镜像	true|false	-
getVideoElement	获取内置 video 对象	-	HTMLVideoElement
destroy	销毁	-	-

注意

播放器会默认跟随 camera 设备控制 的变化。您可以简单理解为 camera 的设备控制是总开关，而 localPlayer 的播放控制是一个子开关；例如：

调用 `camera.muteVideo` 后，此时禁用了设备视频流，此时 localPlayer 即使再调用 play 也相当于处于停止播放状态。

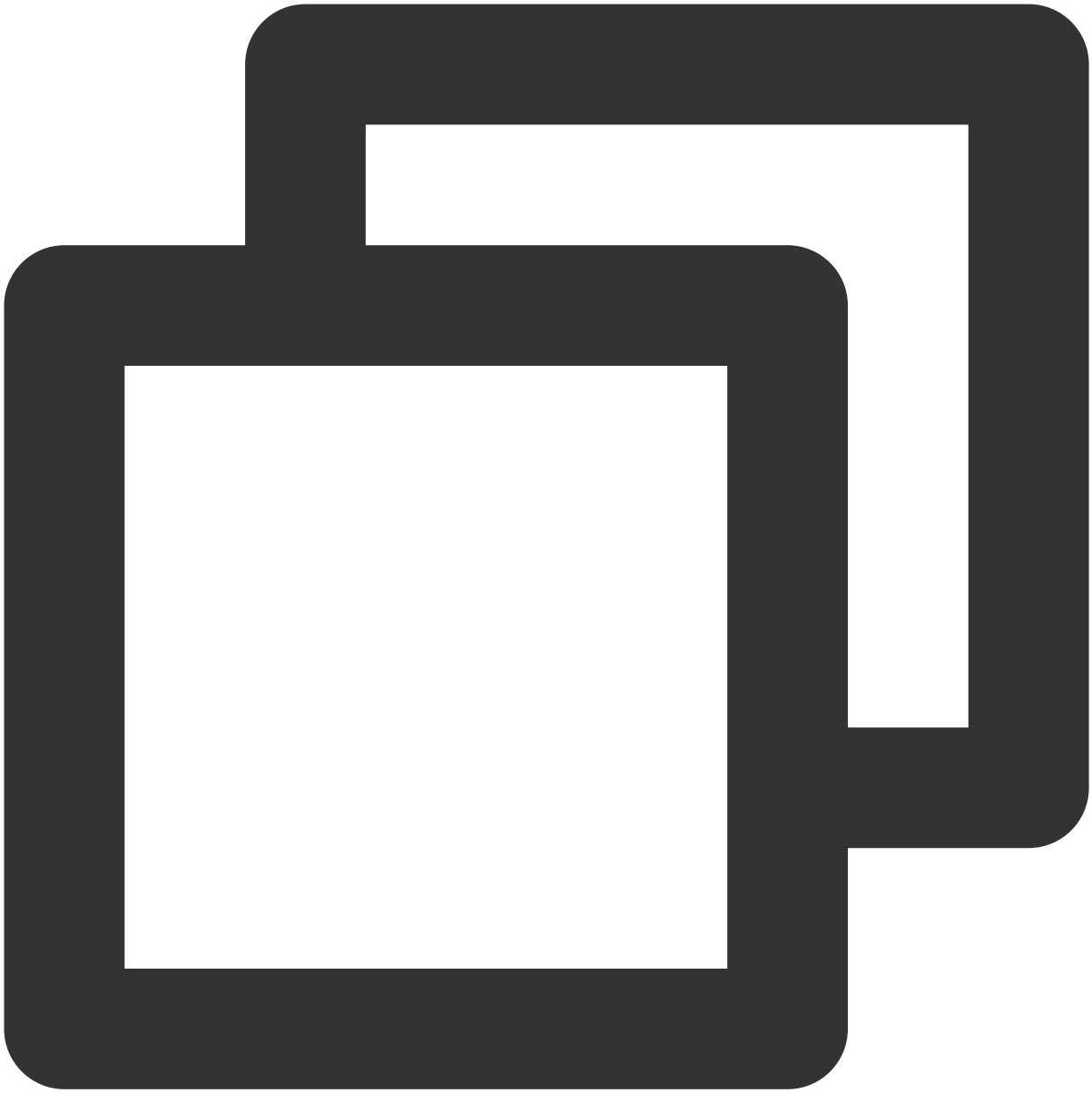
调用 `camera.unmuteVideo` 后，重新启用了视频流，此时 localPlayer 会默认自动恢复播放状态。

因此启用 camera 配置情况下您无须再手动控制 localPlayer 的状态，管理好 camera 设备状态即可。

步骤4：获取输出

如有推流相关需求，可使用 getOutput 方法获取输出的媒体流。

拿到输出的 MediaStream 之后，可以结合第三方 SDK（如 TRTC Web SDK，快直播 Web SDK）进行推流等后续处理。



```
const output = await sdk.getOutput()
```

注意

使用内置相机初始化时，getOutput 输出的媒体流均为 `MediaStream` 类型。

输出的媒体流中 video 轨道是 ArSdk 实时处理的，如有 `audio` 轨道则保持不变。

getOutput 方法是异步方法，会等到sdk执行完一系列初始化工作并且可以生成流之后返回。

getOutput 方法支持传入一个 FPS 参数，表示设置输出的帧率为 FPS（例如15），不传则默认取输入流的帧率。

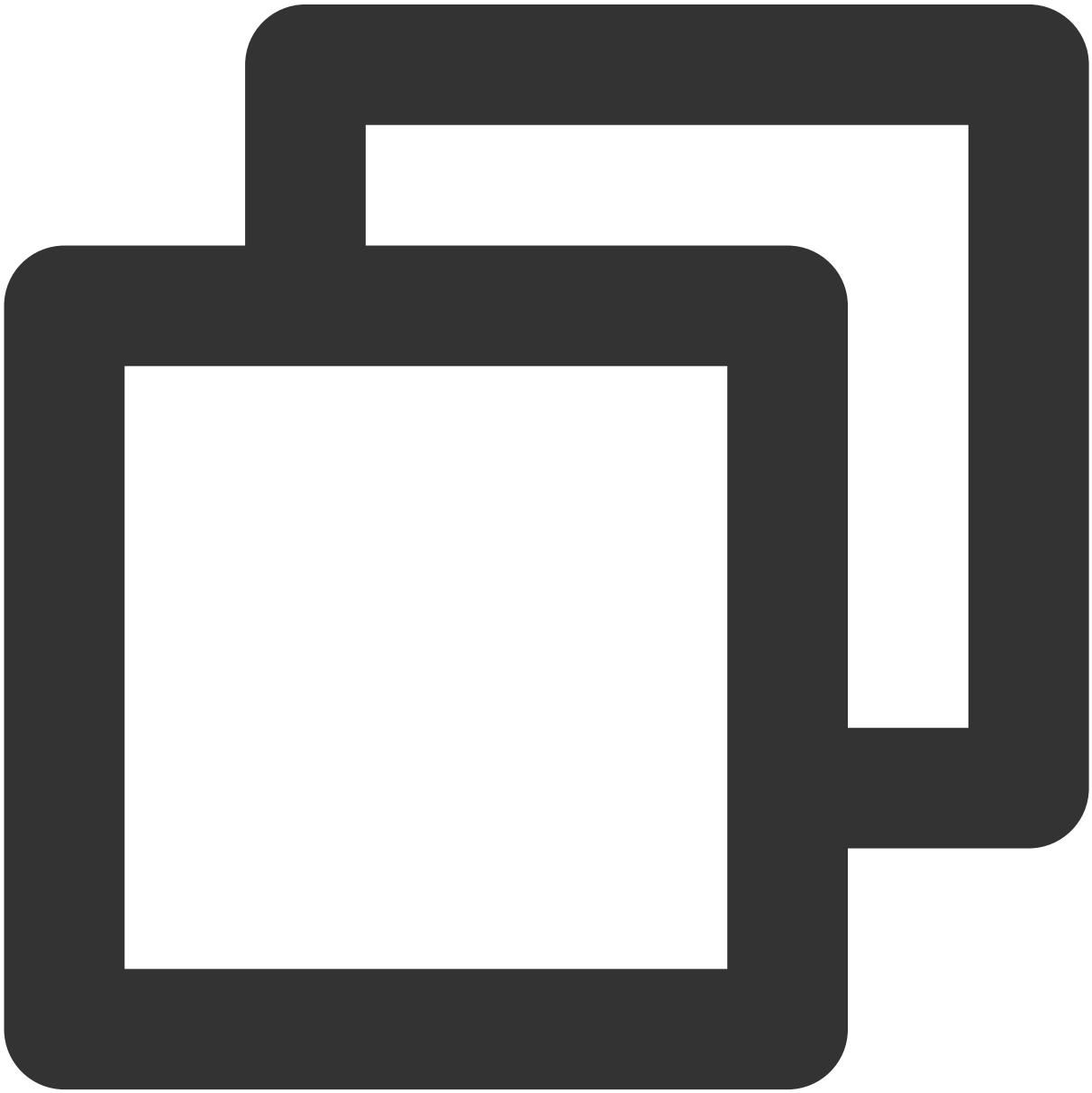
推流等具体操作参见 结合 TRTC 推流 和 结合 WebRTC 推流。

步骤5：设置美颜和特效

SDK 的所有素材均兼容微信小程序端与 Web 端，调用方式一致，详情请参见 设置美颜和特效。

步骤6：设备控制

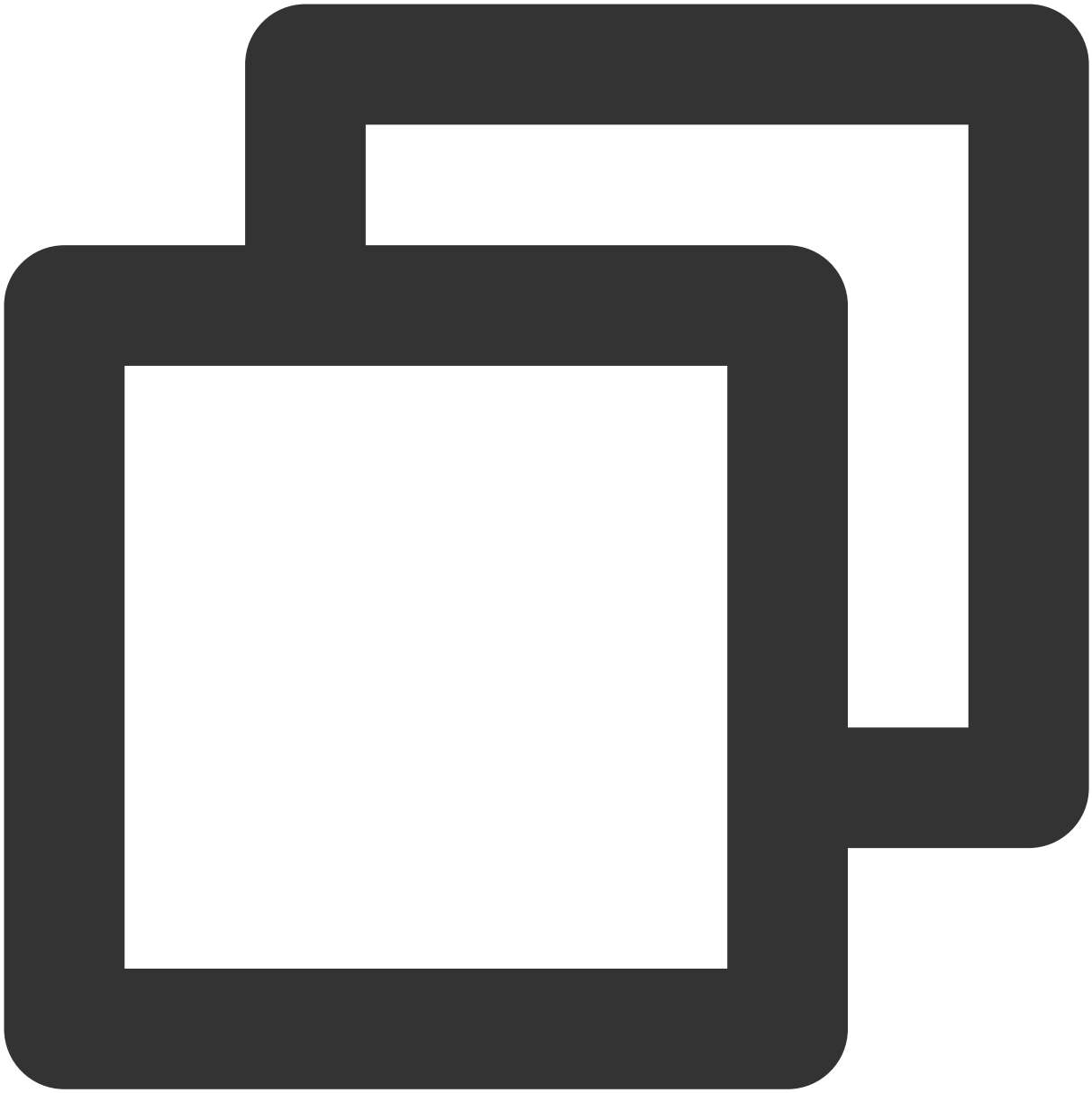
如果您需要进行摄像头的启停等操作，可通过 sdk.camera 实例来实现，示例代码如下：



```
const output = await sdk.getOutput()
```

```
// todo 这里省略若干业务逻辑
// ...

// getOutput之后sdk.camera已经初始化完毕了可以直接取
const cameraApi = sdk.camera

// 获取当前设备列表
const devices = await cameraApi.getDevices()
console.log(devices)
// 禁用视频轨道
// cameraApi.muteVideo()
// 启用视频轨道
// cameraApi.unmuteVideo()
// 如果有多个摄像头可以根据设备id切换
// await cameraApi.switchDevices('video', 'your-device-id')
```

如果需要在第一时间取到 `sdk.camera`，可以在初始化时，cameraReady 事件中获取：



```
// todo 这里省略若干初始化参数
// ...
const sdk = new ArSdk(
    config
)

let cameraApi;
sdk.on('cameraReady', async () => {
    cameraApi = sdk.camera
    // 获取当前设备列表
    const devices = await cameraApi.getDevices()
```

```
    console.log(devices)
    // 禁用视频轨道
    // cameraApi.muteVideo()
    // 启用视频轨道
    // cameraApi.unmuteVideo()
    // 如果有多个摄像头可以根据设备id切换
    // await cameraApi.switchDevices('video', 'your-device-id')
})
```

为了方便设备控制，内置的 `camera` 提供了以下方法调用：

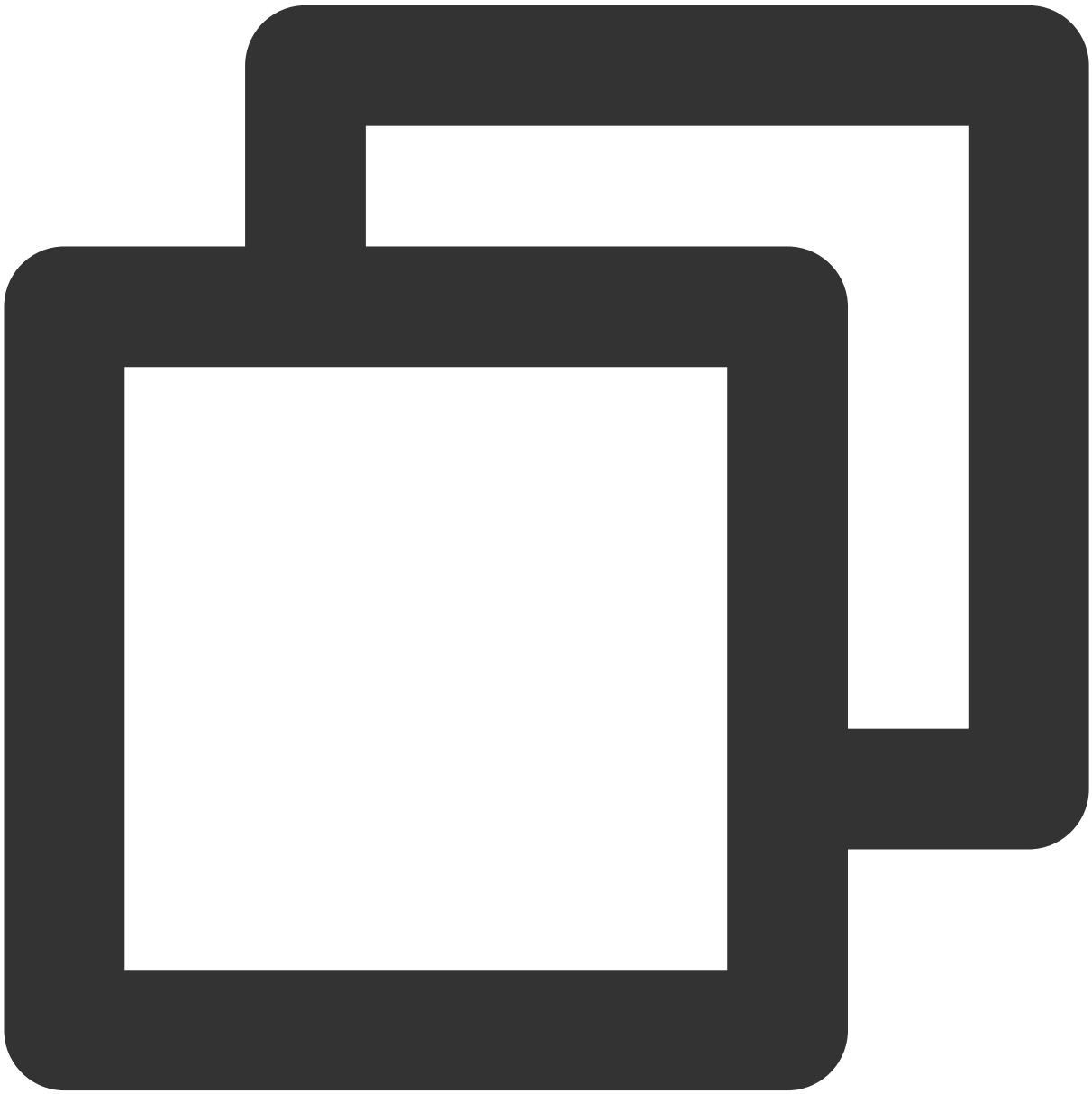
方法名	说明	入参	返回值
getDevices	获取当前所有设备列表	-	Promise<Array<MediaDeviceInfo>>
switchDevice	切换设备	type:string, deviceId:string	Promise
muteAudio	静音当前摄像头流	-	-
unmuteAudio	恢复静音	-	-
muteVideo	禁用当前摄像头流内画面，此时流仍存在只是视频轨道已禁用	-	-
unmuteVideo	启用当前摄像头流内画面	-	-
stopVideo	停止当前摄像头设备，视频流会停止（音频流还存在）	-	-
restartVideo	重启当前摄像头，在 stopVideo 之后使用	-	Promise
stop	停止当前摄像头视频以及音频设备	-	-

自定义流

最近更新时间：2023-03-28 16:37:22

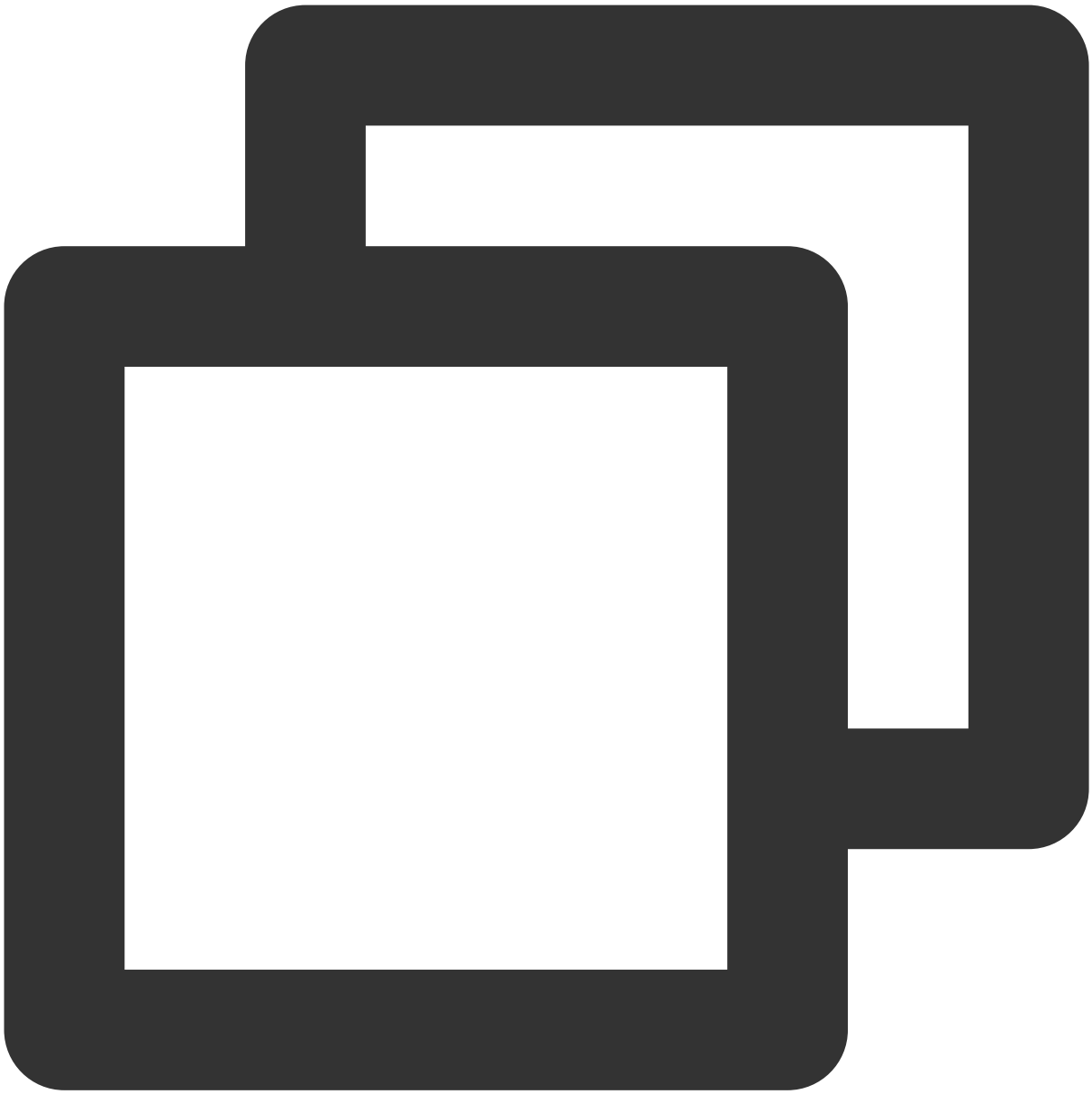
自定义流方式适用于手动维护媒体流或者要求更高的灵活性与可控性的场景。

步骤1：引入 SDK



```
import { ArSdk } from 'tencentcloud-webar';// SDK 类
```

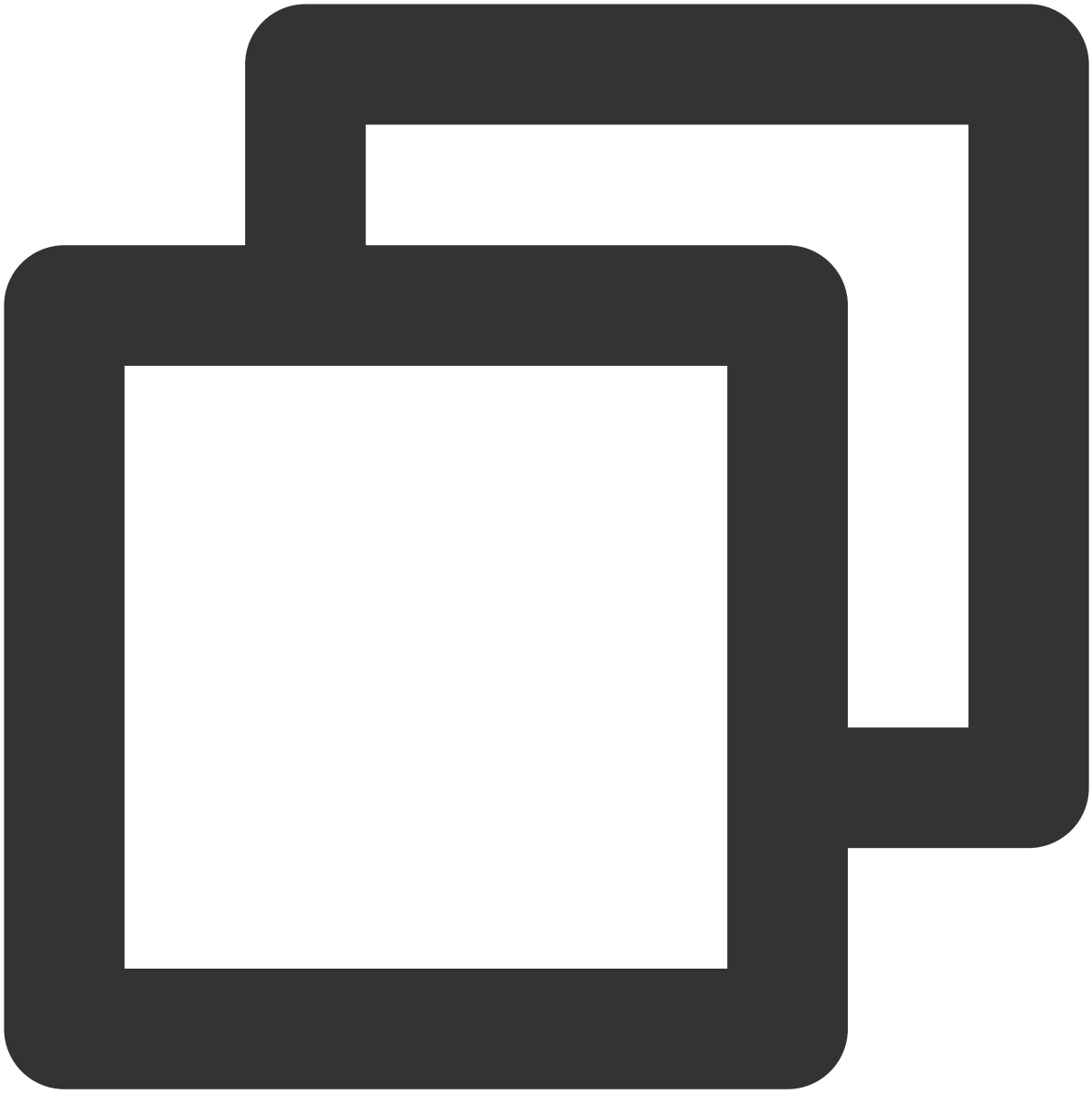
如果项目无需编译，则可以直接以如下方式引用：



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

步骤2：初始化实例

1. 初始化一个 SDK 实例。



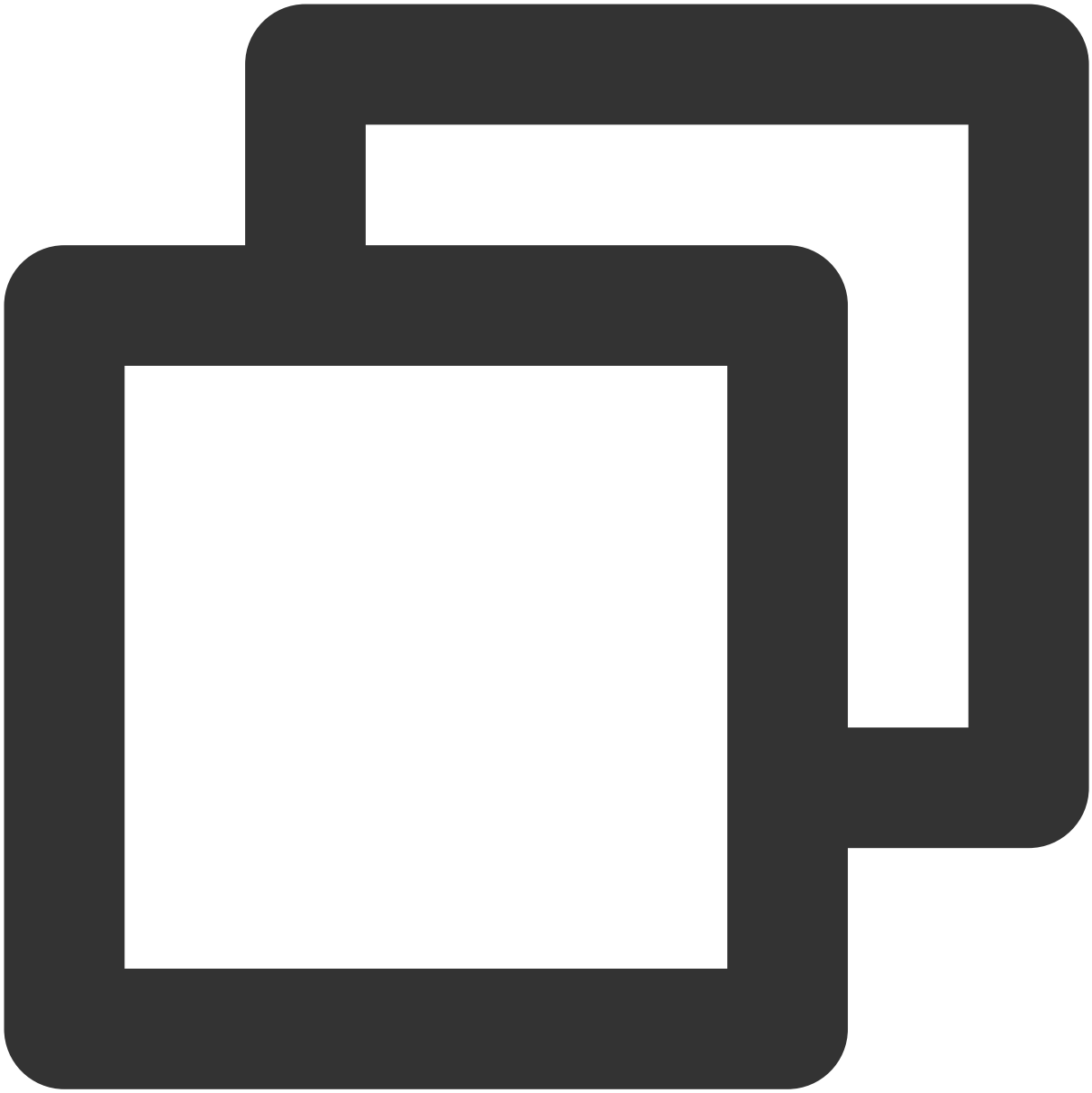
```
// 获取鉴权参数
const authData = {
 licenseKey: 'xxxxxxxxx',
 appId: 'xxx',
 authFunc: authFunc // 详见「License 配置与使用 - 签名方法」
};
// 输入SDK的流
const stream = await navigator.mediaDevices.getUserMedia({
 audio: true,
 video: { width: w, height: h }
})
```

```
const config = {
 module: { // 0.2.0版本新增
     beautify: true, // 是否启用美颜模块，启用后可以使用美颜、美妆、贴纸等功能
     segmentation: true // 是否启用人像分割模块，启用后可以使用背景功能
 },
 auth: authData, // 鉴权参数
 input: stream, // input传输入流
 beautify: { // 初始化美颜参数(可选)
     whiten: 0.1,
     dermabrasion: 0.3,
     eye: 0.2,
     chin: 0,
     lift: 0.1,
     shave: 0.2
 }
}
const sdk = new ArSdk(
 // 传入一个 config 对象用于初始化 sdk
 config
)
```

注意

由于美颜检测和人像分割均有一定的加载耗时和资源消耗，初始化配置中提供了模块配置以供选择需要的功能，关闭的模块将不会进行预加载和初始化，也无法设置相关的效果。

2. 输入除了媒体流外，也支持传 `string|HTMLImageElement` 来处理图片。



```
const config = {
 auth: authData, // 鉴权参数
 input: 'https://xxx.png', // input传输入流
}
const sdk = new ArSdk(
 // 传入一个 config 对象用于初始化 sdk
 config
)

// 在 created 回调中可拉取特效和滤镜列表供页面展示，详见「SDK接入 - 参数与方法」
sdk.on('created', () => {
```

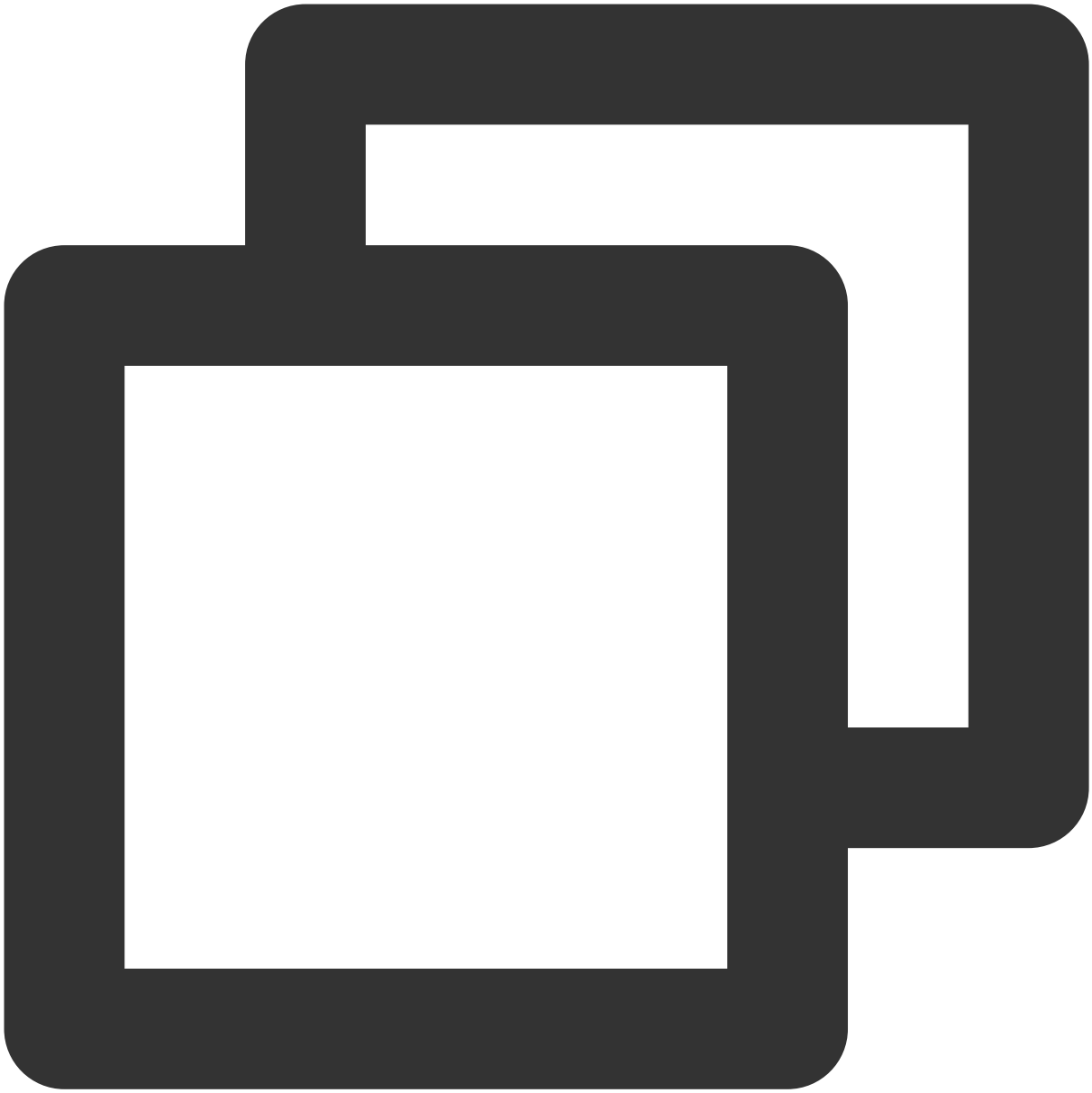
```
  // 获取内置美妆
  sdk.getEffectList({
      Type: 'Preset',
      Label: '美妆',
  }).then(res => {
      effectList = res
  });
  // 获取内置滤镜
  sdk.getCommonFilter().then(res => {
      filterList = res
  })
})

// 在 ready 回调中调用 setBeautify/setEffect/setFilter 等渲染方法
// 详情可参见「SDK接入 - 设置美颜和特效」
sdk.on('ready', () => {
  // 设置美颜
  sdk.setBeautify({
      whiten: 0.2
  });
  // 设置特效
  sdk.setEffect({
      id: effectList[0].EffectId,
      intensity: 0.7
  });
  // 设置滤镜
  sdk.setFilter(filterList[0].EffectId, 0.5)
})
```

步骤3：播放流

可以使用 `ArSdk.prototype.getOutput` 方法获取输出的媒体流。

不同的回调事件中获取的输出流有些许差别，适用于不同的场景，根据自身业务需求选择一种处理方式即可。

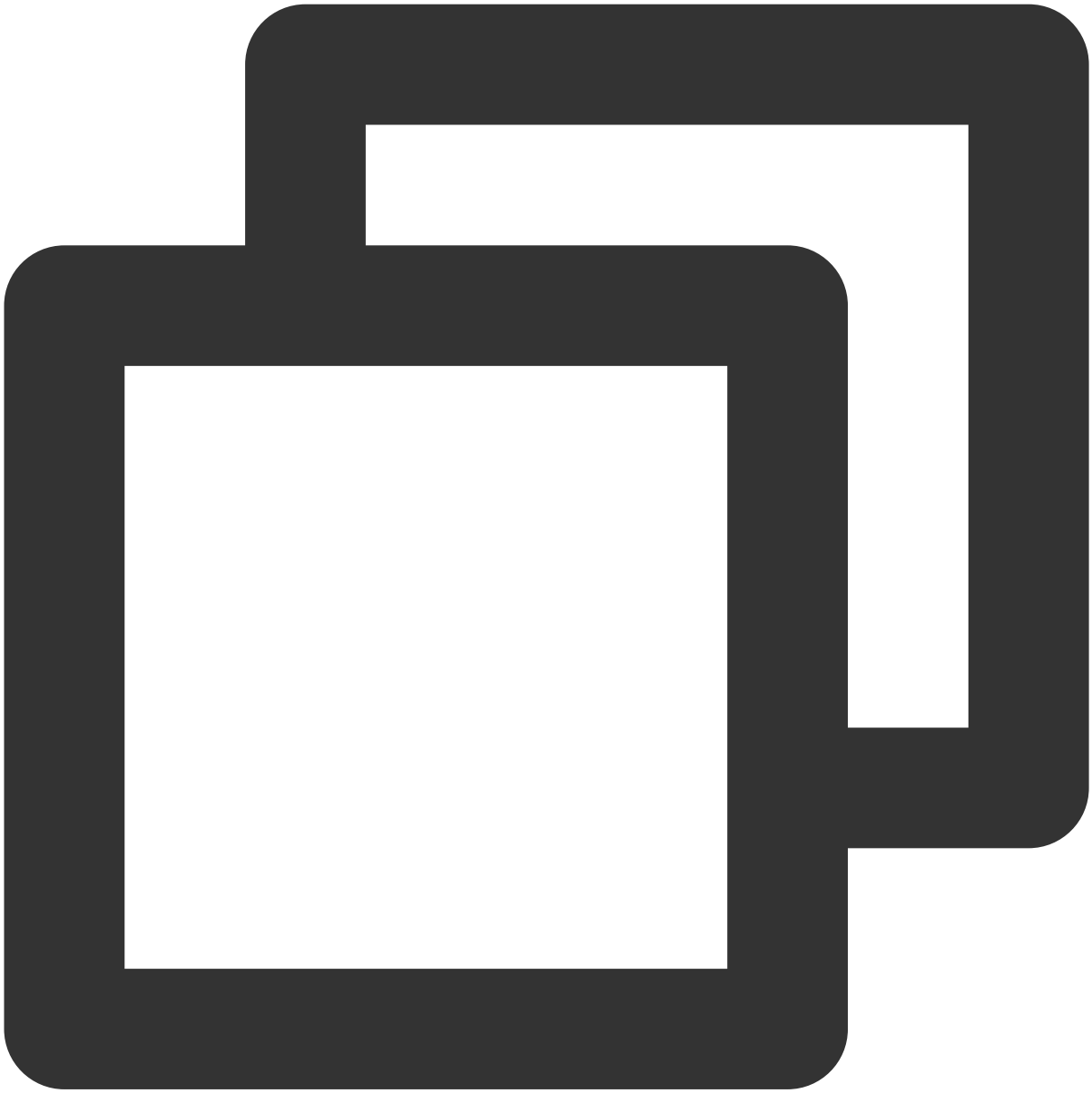
如果对画面展示的时效性有需求，可以在 `cameraReady` 回调中获取并播放输出流，此时 SDK 尚未开始资源加载和检测的初始化，仅能展示原始画面。



```
sdk.on('cameraReady', async () => {
  // 在 cameraReady 回调中可以更早地获取输出画面，此时初始化传入的美颜参数还未生效，同输入画面
  // 适用于需要尽早地展示画面，但不要求画面一出现就有美颜的场景
  // 后续美颜生效后无需更新stream，SDK内部已处理
  const output = await ar.getOutput();
  // 使用video播放输出的媒体流，预览效果
  const video = document.createElement('video')
  video.setAttribute('playsinline', '');
  video.setAttribute('autoplay', '');
  video.srcObject = output
  document.body.appendChild(video)
```

```
  video.play()
})
```

如果需要检测初始化完成，且美颜生效之后再播放，则可以在 ready 回调获取输出并播放。

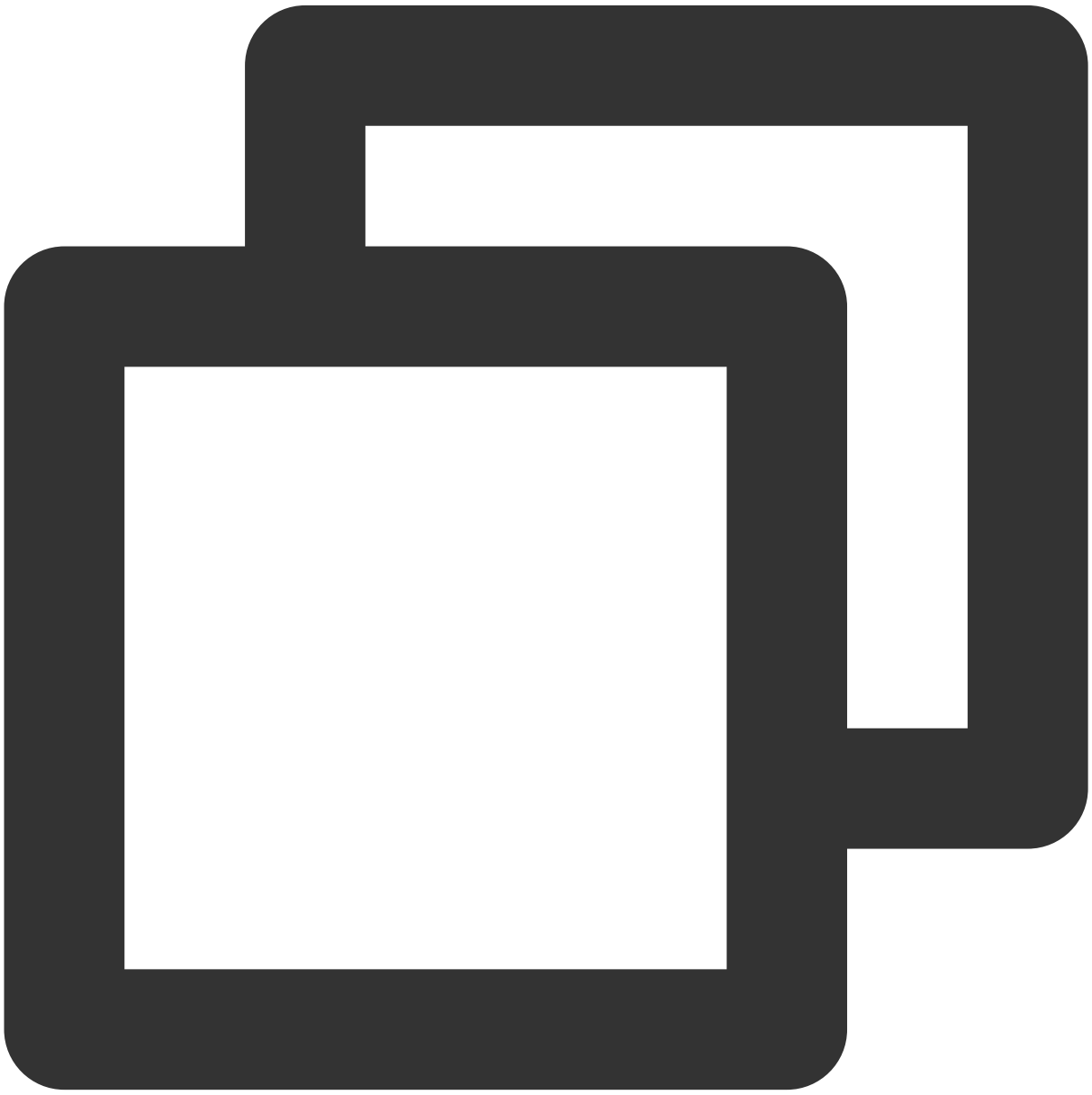


```
sdk.on('ready', async () => {
  // 在 ready 回调中获取输出画面，此时初始化传入的美颜参数已生效，获取的 output stream 已带美颜
  // 区别上述 cameraReady 中获取 output，此回调执行时机稍晚一些，适用于画面一展示就要有美颜的场
  const output = await ar.getOutput();
  // 使用video播放输出的媒体流，预览效果
  const video = document.createElement('video')
  video.setAttribute('playsinline', '');
```

```
    video.setAttribute('autoplay', '');
    video.srcObject = output
    document.body.appendChild(video)
    video.play()
})
```

步骤4：获取输出

拿到输出的 `MediaStream` 之后，可以结合第三方 SDK（如 TRTC Web SDK，快直播 Web SDK）进行推流等后续处理。



```
const output = await sdk.getOutput()
```

推流等操作参见 结合 TRTC 推流 和 结合 WebRTC 推流。

注意

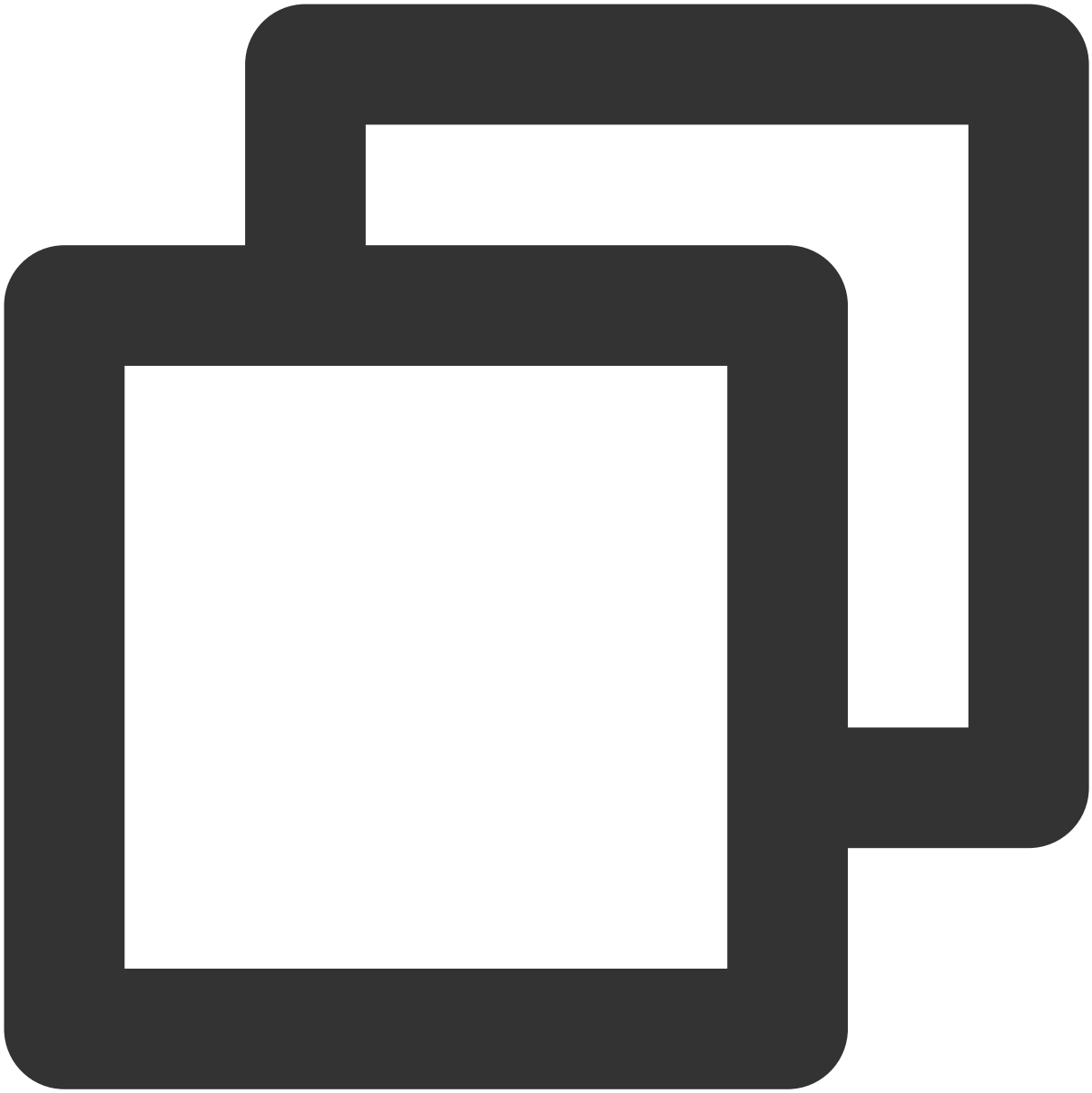
如果传入的 input 是图片，则返回为 string 类型的 DataURL，其他场景均返回 `MediaStream` 类型。
输出的媒体流中 `video` 轨道是 ArSdk 实时处理的，如有 `audio` 轨道则保持不变。
getOutput 方法是异步方法，会等到 SDK 执行完一系列初始化工作并且可以生成流之后返回。
getOutput 方法支持传入一个 FPS 参数，表示设置输出的帧率为 FPS（例如15），不传则默认取输入流的帧率。
getOutput 可以执行多次，每次执行会产生一个新的媒体流，可用于输出不同帧率媒体流的场景（例如预览时使用高帧率流，推流时使用低帧率流）。

步骤5：设置美颜和特效

SDK 的所有素材均兼容微信小程序端与 Web 端，调用方式一致，详情请参见 设置美颜和特效。

更新输入流（0.1.19版本后开始支持）

如果您有切换设备、启停摄像头等需求，需要获取新的流输入sdk，请勿重复初始化多次sdk，可以直接调用 `sdk.updateInputStream` 切换输入流。
下述以切换电脑默认摄像头及外置摄像头为例，介绍 updateInputStream 的用法：

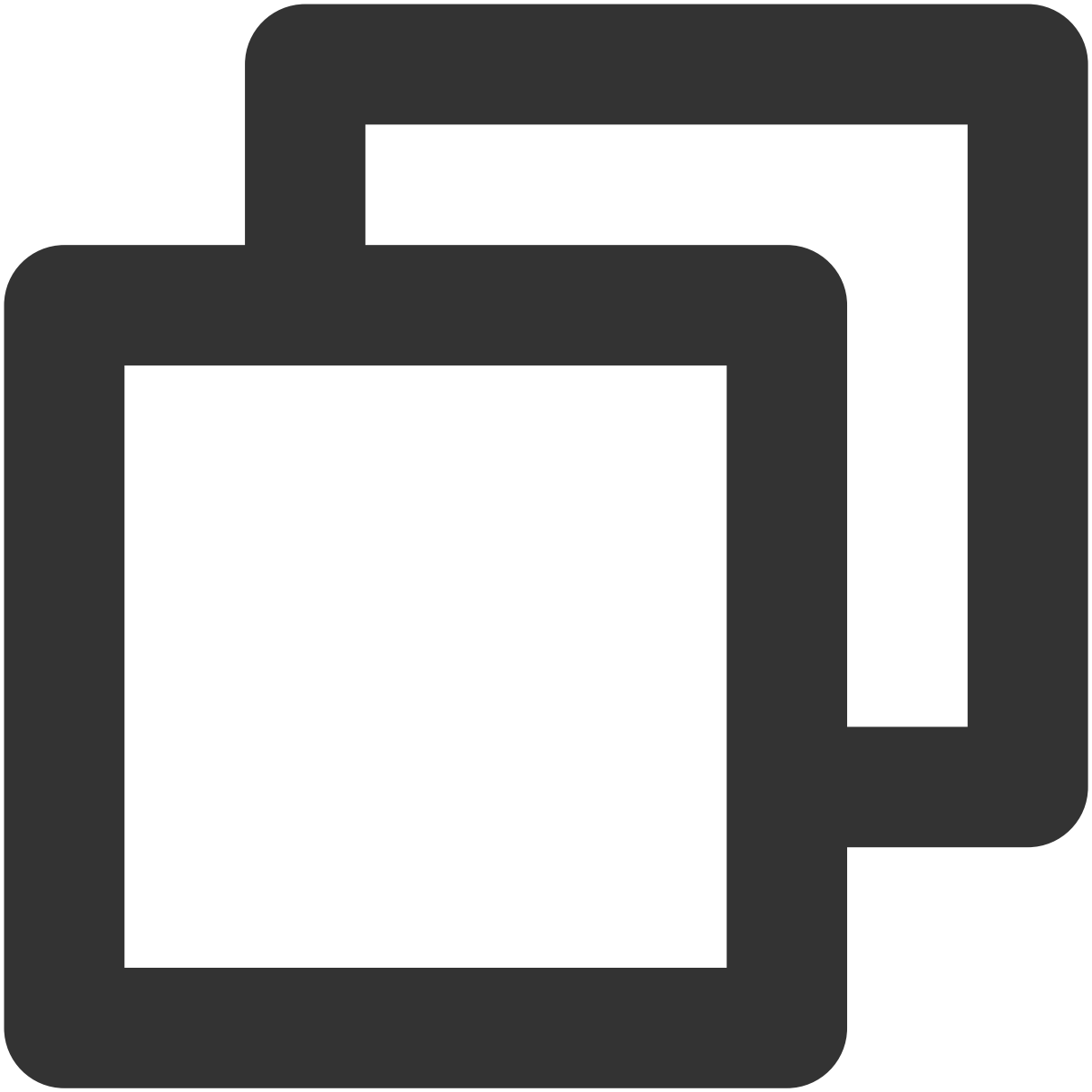


```
async function getVideoDeviceList(){
    const devices = await navigator.mediaDevices.enumerateDevices()
    const videoDevices = []
    devices.forEach((device)=>{
        if(device.kind === 'videoinput'){
            videoDevices.push({
                label: device.label,
                id: device.deviceId
            })
        }
```

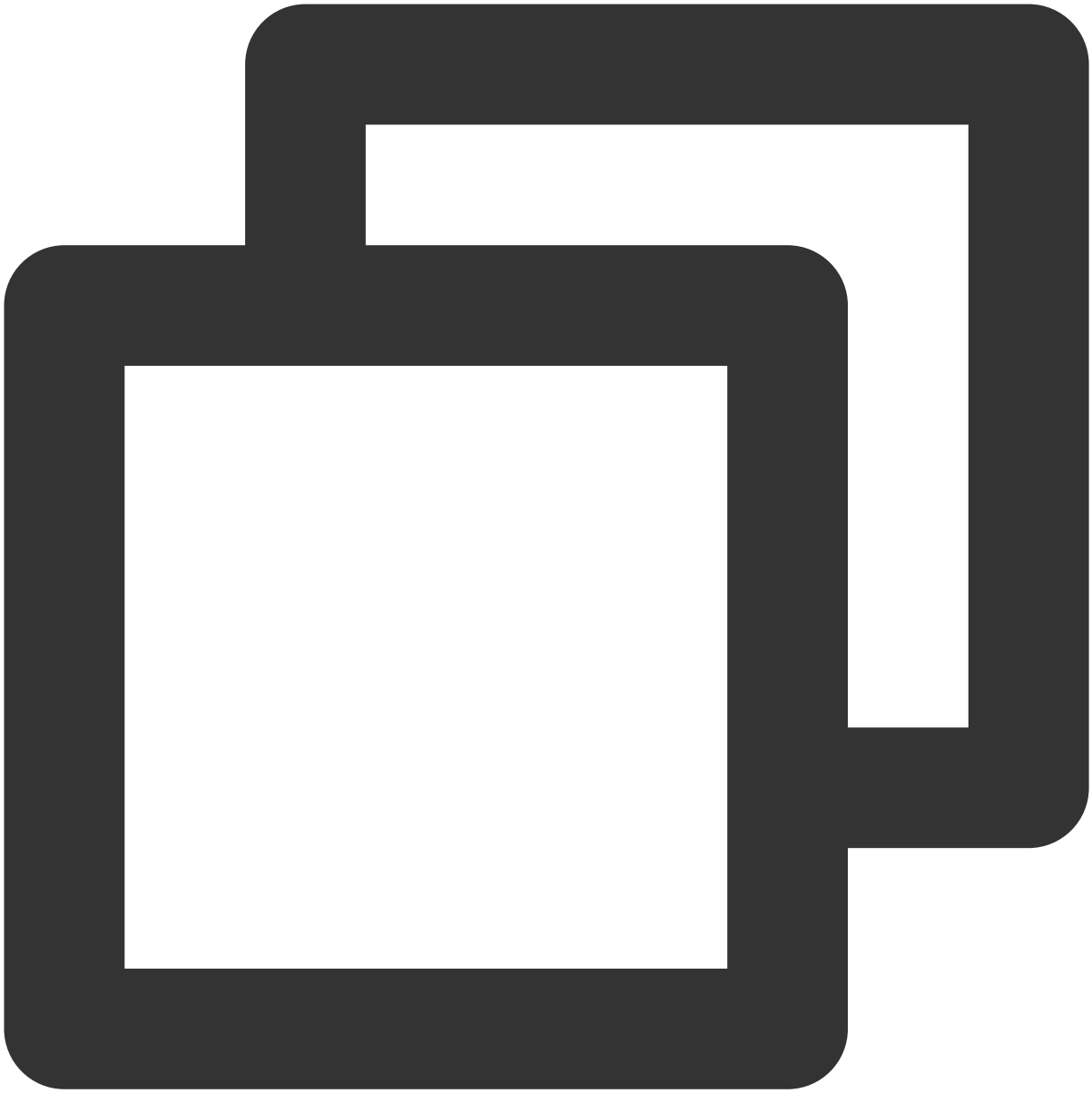
```
    })
    return videoDevices
}

async function initDom(){
    const videoDeviceList = await getVideoDeviceList()
    let dom = ''
    videoDeviceList.forEach(device=>{
        dom = `${dom}
        <button id=${device.id} onclick='toggleVideo("${device.id}")'>${device.labe
        `
    })
    const div = document.createElement('div');
    div.id = 'container';
    div.innerHTML = dom;
    document.body.appendChild(div);
}

async function toggleVideo(deviceId){
    const stream = await navigator.mediaDevices.getUserMedia({
        video: {
            deviceId,
            width: 1280,
            height: 720,
          }
    })
    // 使用sdk提供的更新输入流接口，内部已处理旧轨道的 stop
    // 输入流更新后无需再次调用 getOutput ，SDK内部已处理
    sdk.updateInputStream(stream)
}

initDom()
```

暂停与恢复检测

暂停检测可以节省 CPU 占用，如果业务逻辑中有需要暂时停止检测可以调用 disable 和 enable 接口进行手动启停。



```
<button id="disable">停止检测</button>
<button id="enable">启动检测</button>
```



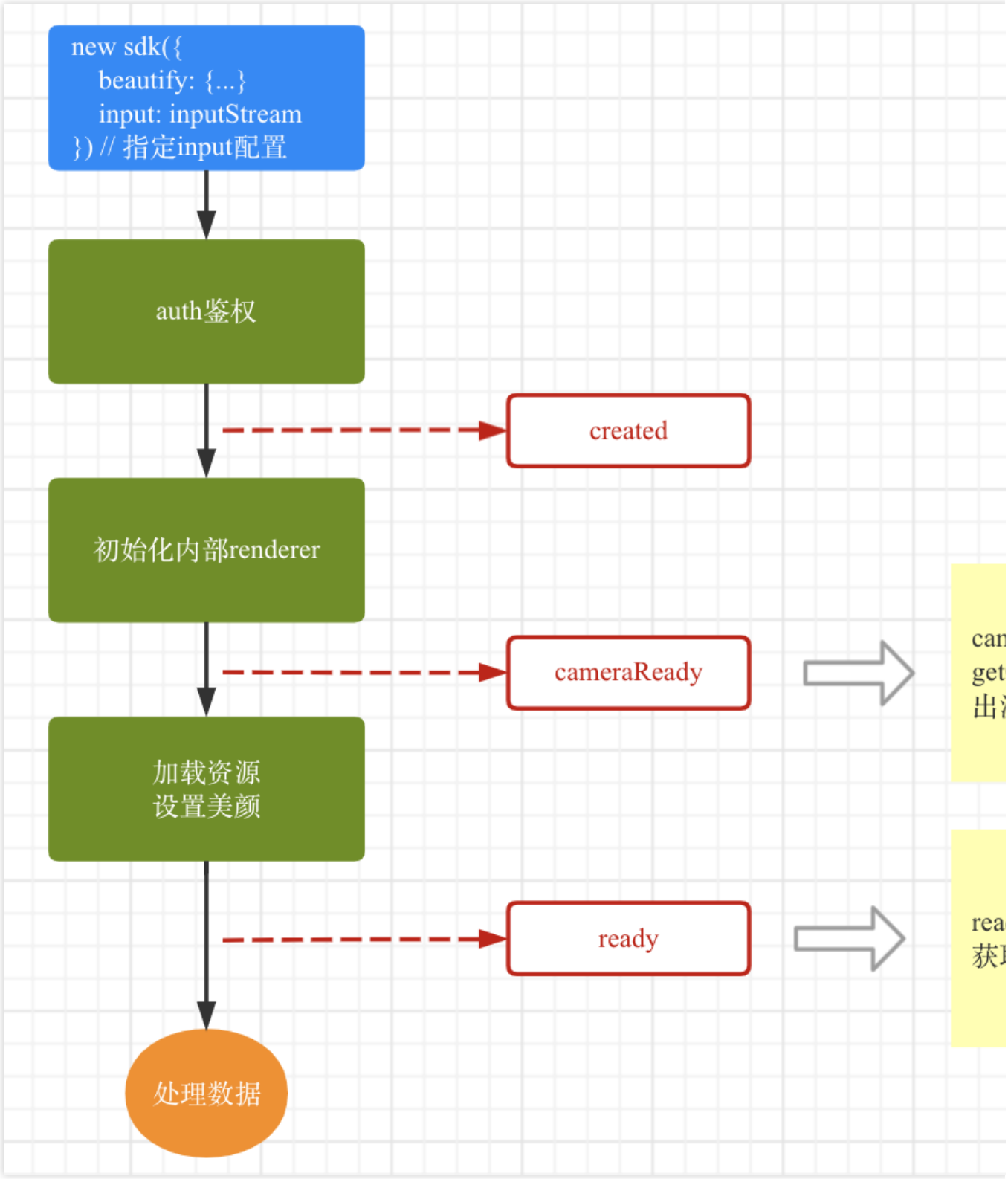
```
// 停止检测，输出原始画面
disableButton.onClick = () => {
    sdk.disable()
}
// 启动检测，输出美颜等特效生效后的画面
enableButton.onClick = () => {
    sdk.enable()
}
```

加载优化

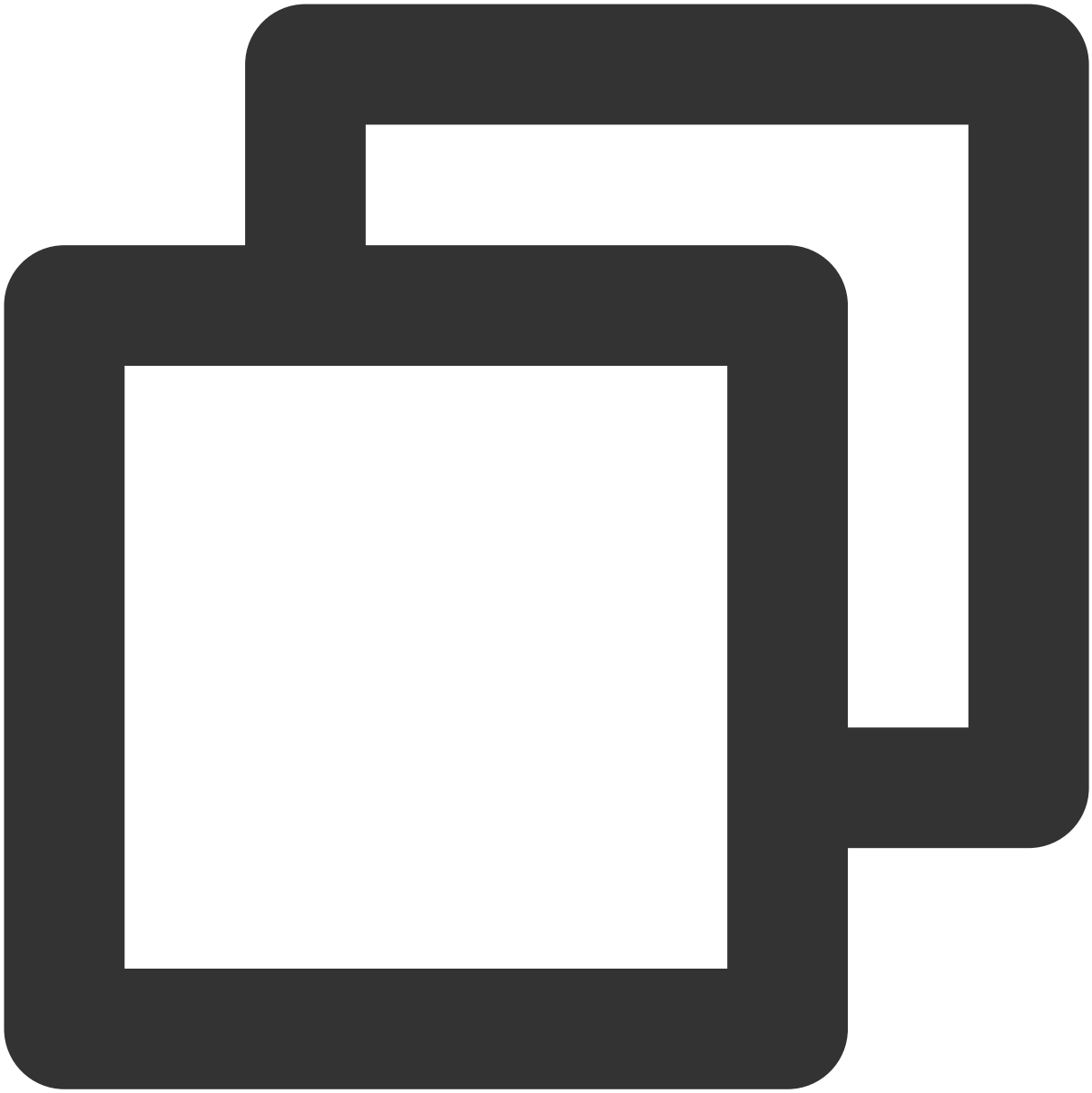
最近更新时间：2024-05-08 16:30:10

普通模式

普通模式下可以在 `cameraReady` 回调中通过 getOutput 接口获取输出流，此时获取的输出流数据与传递给 SDK 的输入流数据是一样的，美颜效果还未生效，可以理解为 SDK 将输入数据原封不动地吐出，后续相关美颜效果准备就绪后会自动叠加到此输出流中，无需重新调用 getOutput 获取，这种方式适用于页面初始化就初始化 SDK，且需要尽早地展示画面，但不要求画面一展示就有美颜效果的场景。
相应的，也可以在 `ready` 回调中通过 getOutput 接口获取输出流，与上述 `cameraReady` 不同的是，此时获取的输出流数据已经带有美颜特效，由于该回调接口触发时机略晚于 `cameraReady` 接口，因此这种方式适用于画面一展示就要有美颜效果，但不要求尽早地展示画面的场景。


new sdk({
beautify: {...}
input: inputStream
}) // 指定input配置
auth鉴权
created
初始化内部renderer
cameraReady
加载资源
设置美颜
ready
处理数据

示例代码如下：



```
// 获取鉴权参数
const authData = {
    licenseKey: 'xxxxxxxxx',
    appId: 'xxx',
    authFunc: authFunc // 详见「License 配置与使用 - 签名方法」
};

// 初始化sdk时传input或camera参数，按照普通模式加载
const config = {
    auth: authData, // 鉴权参数
    beautify: { // 美颜参数
```

```
            whiten: 0.1,
            dermabrasion: 0.5,
            lift: 0,
            shave: 0,
            eye: 0.2,
            chin: 0,
        },
        input: inputStream // 构建传递给SDK的 input 流数据，详见「参数与方法 - 初始化参数」
}
const sdk = new ArSdk(
    // 传入一个 config 对象用于初始化 sdk
    config
)
// sdk通过鉴权后会立刻触发created事件
sdk.on('created', () => {
    // 在 created 回调中拉取特效和滤镜列表供页面展示
    sdk.getEffectList({
        Type: 'Preset',
        Label: '美妆',
    }).then(res => {
        effectList = res
    });
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})

// 不同回调中getOutput获取的数据差别如下，根据自身需求选择其中一种即可
sdk.on('cameraReady', async () => {
    const output = await sdk.getOutput() // 美颜参数未生效
    // 播放
    ...
})
sdk.on('ready', async () => {
    const output = await sdk.getOutput() // 美颜参数已生效
    // 播放
    ...
    // 在ready回调中调用setEffect/setBeautify/setFilter等效果
})
```

预初始化（0.2.0版本后开始支持）

在 SDK 初次加载时需要下载远程静态资源后才能够完成检测模型的初始化，加载的耗时受到网络的影响。在一些要求画面快速出现的业务场景中，SDK 提供了预初始化的加载方案来提前加载静态资源。

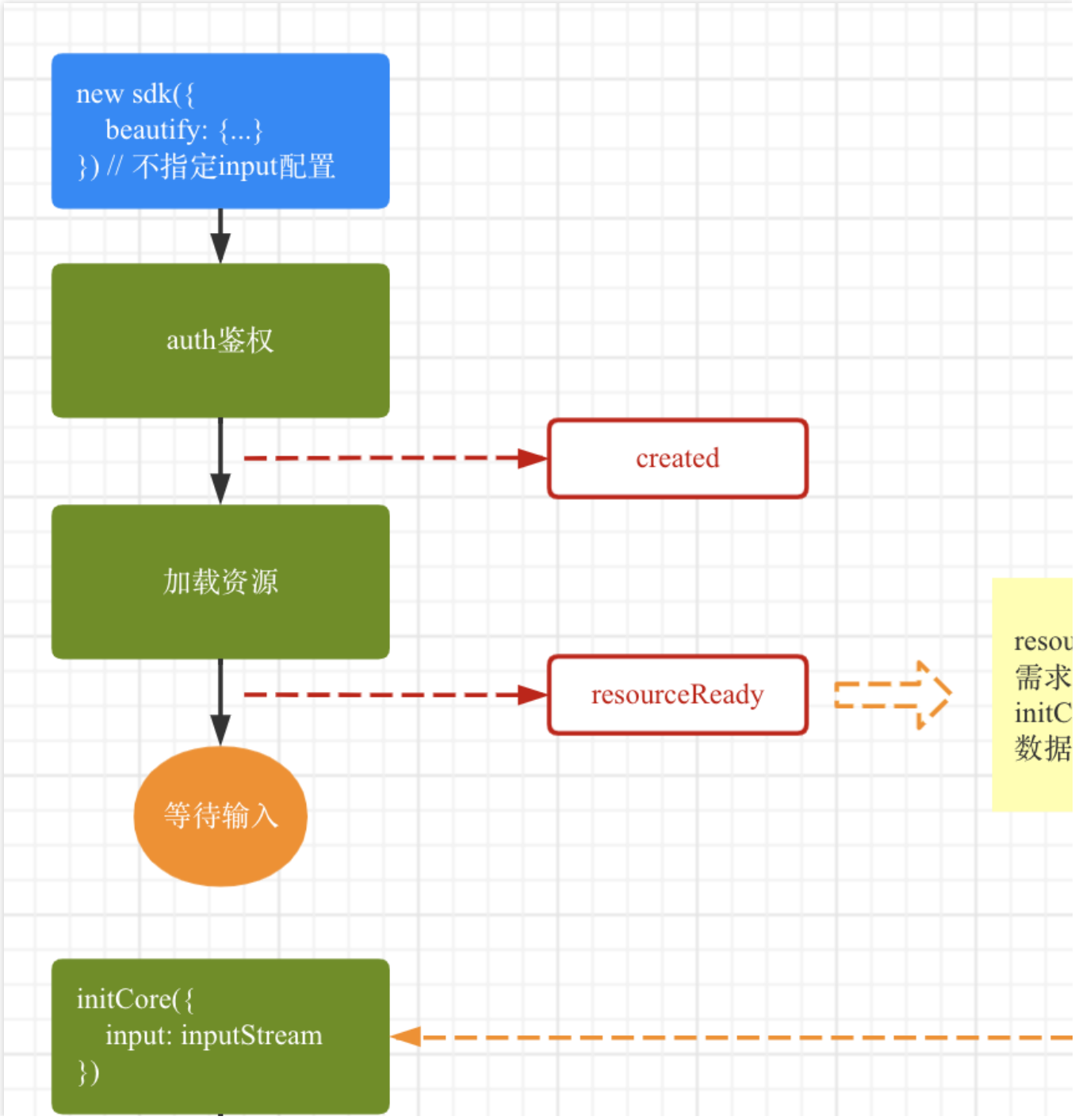
预初始化的应用场景

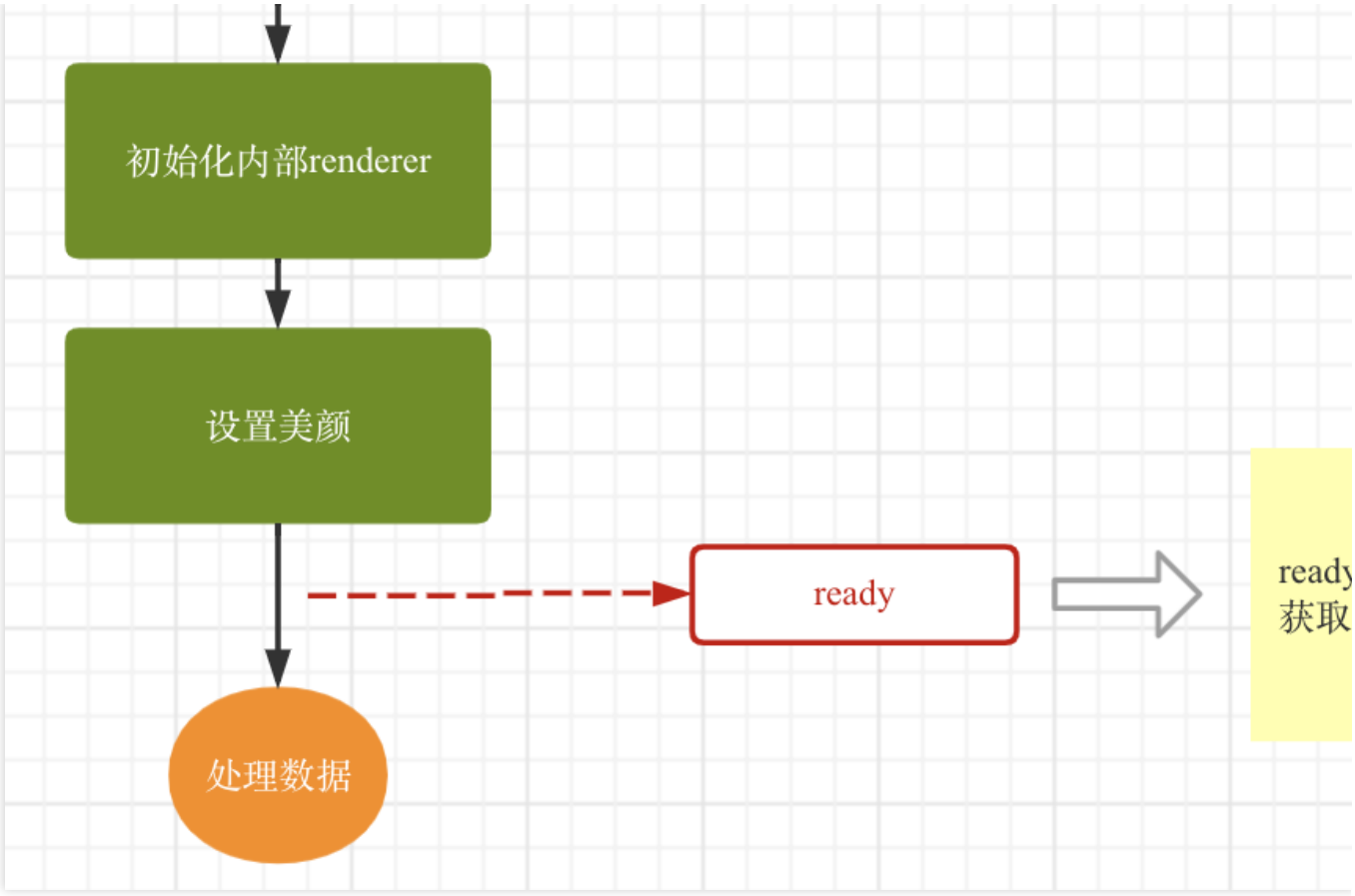
场景一：页面初始化加载时不需要美颜 SDK，需要用户某些操作后才出现视频画面。

场景二：B 页面中涉及美颜效果，而 B 页面跳转自 A 页面。

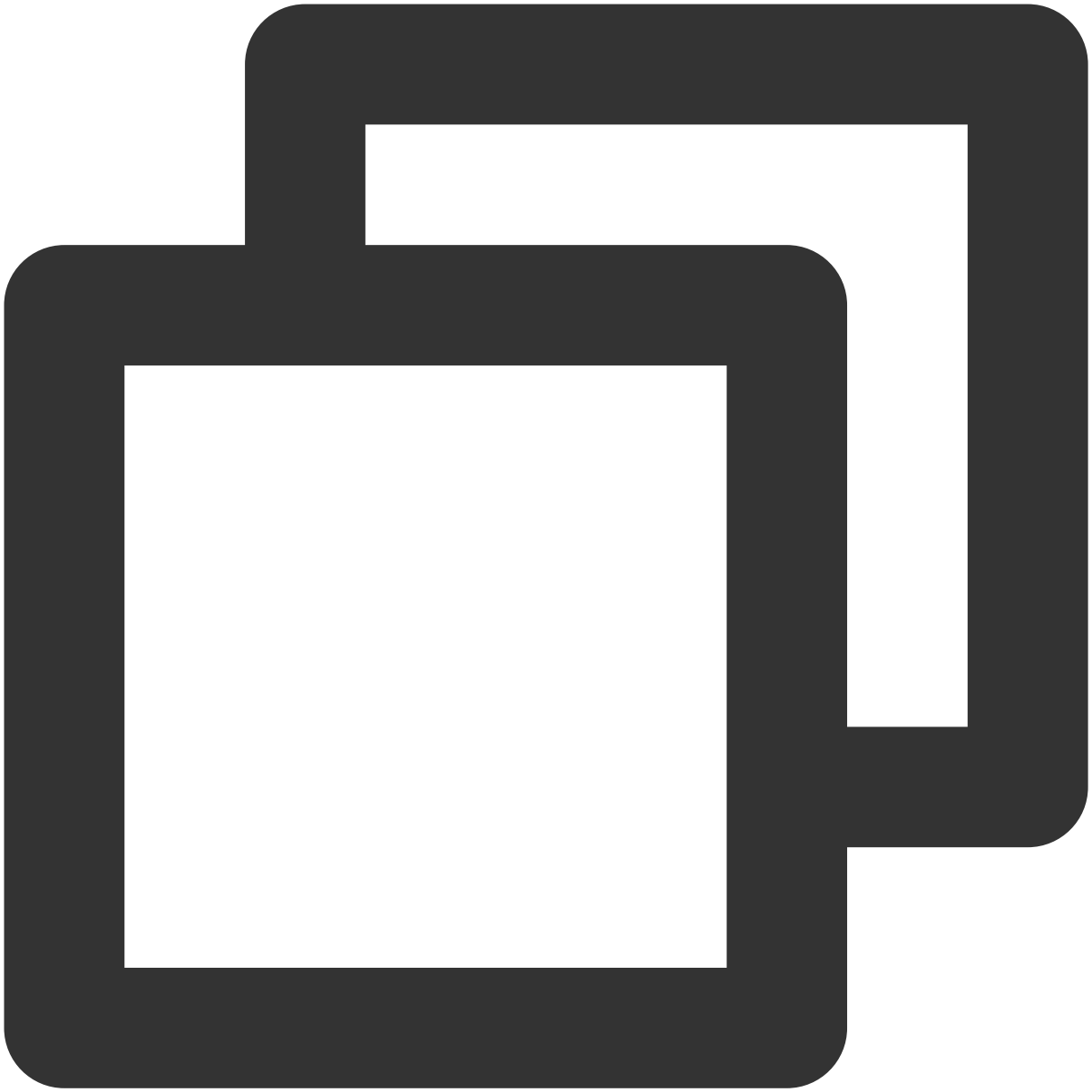
类似的情况都可以先执行资源加载（尽可能早地），之后结合业务需求，在合适的时机为 SDK 提供输入流数据，再获取输出流展示效果。

例如：场景一可以在页面初始化的时候加载资源；场景二可以在 A 页面获取 SDK 实例，传递给 B 使用。

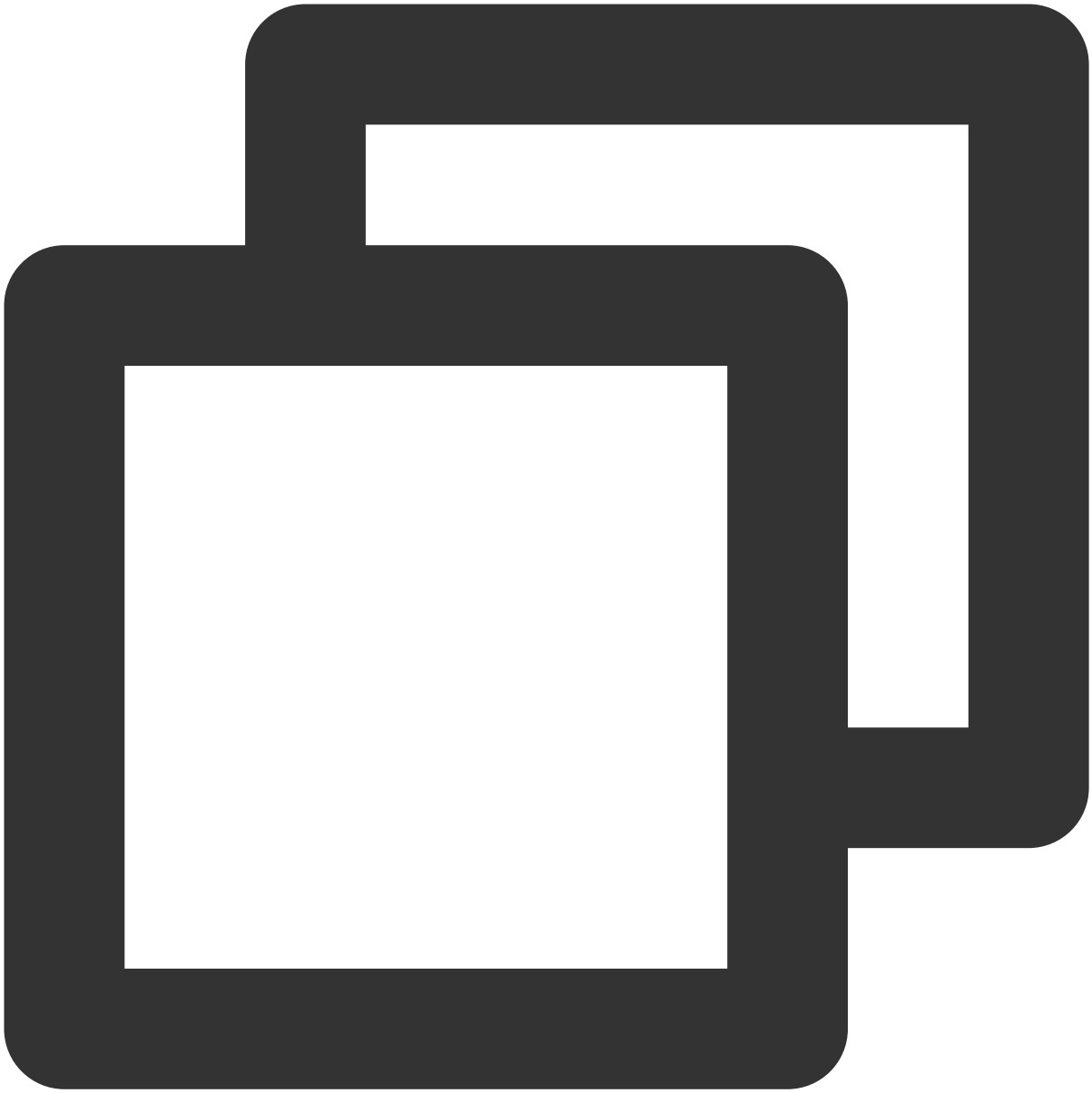




以场景一为例，代码如下：



```
<button id="start">开启摄像头</button>
```



```
// 获取鉴权参数
const authData = {
    licenseKey: 'xxxxxxxxx',
    appId: 'xxx',
    authFunc: authFunc // 详见「License 配置与使用 - 签名方法」
};

// 初始化sdk时不传input或camera参数，实例化后sdk会开始预加载所需的资源
const config = {
    auth: authData, // 鉴权参数
    beautify: { // 美颜参数
```

```
        whiten: 0.1,
        dermabrasion: 0.5,
        lift: 0,
        shave: 0,
        eye: 0.2,
        chin: 0,
    },
}
const sdk = new ArSdk(
    // 传入一个 config 对象用于初始化 sdk
    config
)
// sdk通过鉴权后会立刻触发created事件
sdk.on('created', () => {
    // 在 created 回调中拉取特效和滤镜列表供页面展示
    sdk.getEffectList({
        Type: 'Preset',
        Label: '美妆',
    }).then(res => {
        effectList = res
    });
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})
// resourceReady 意味着相关资源已就绪，可以在此回调之后调用initCore提供输入流数据
sdk.on('resourceReady', () => {
})
// 此模式下，用户显式调用initCore之后，才会触发 ready 回调
sdk.on('ready', async () => {
    const output = await sdk.getOutput() // 美颜已生效
    // 播放
    ...
    // 在ready回调中调用setEffect/setBeautify/setFilter等效果
})
// 用户点击开启摄像头时给SDK传递输入流数据
document.getElementById('start').onclick = async function(){
    const devices = await navigator.mediaDevices.enumerateDevices()
    const cameraDevice = devices.find(d=>d.kind === 'videoinput')
    navigator.mediaDevices.getUserMedia({
        audio: false,
        video: {
            deviceId: cameraDevice.deviceId
            ... // 其他配置
        }
    }).then(mediaStream => {
        // 此模式下，请确保在上述resourceReady事件之后调用initCore方法
```

```
            sdk.initCore({
                input: mediaStream // 构建传递给SDK的 input 流数据，详见「参数与方法 - 初始化参
            })
        })
    }
```

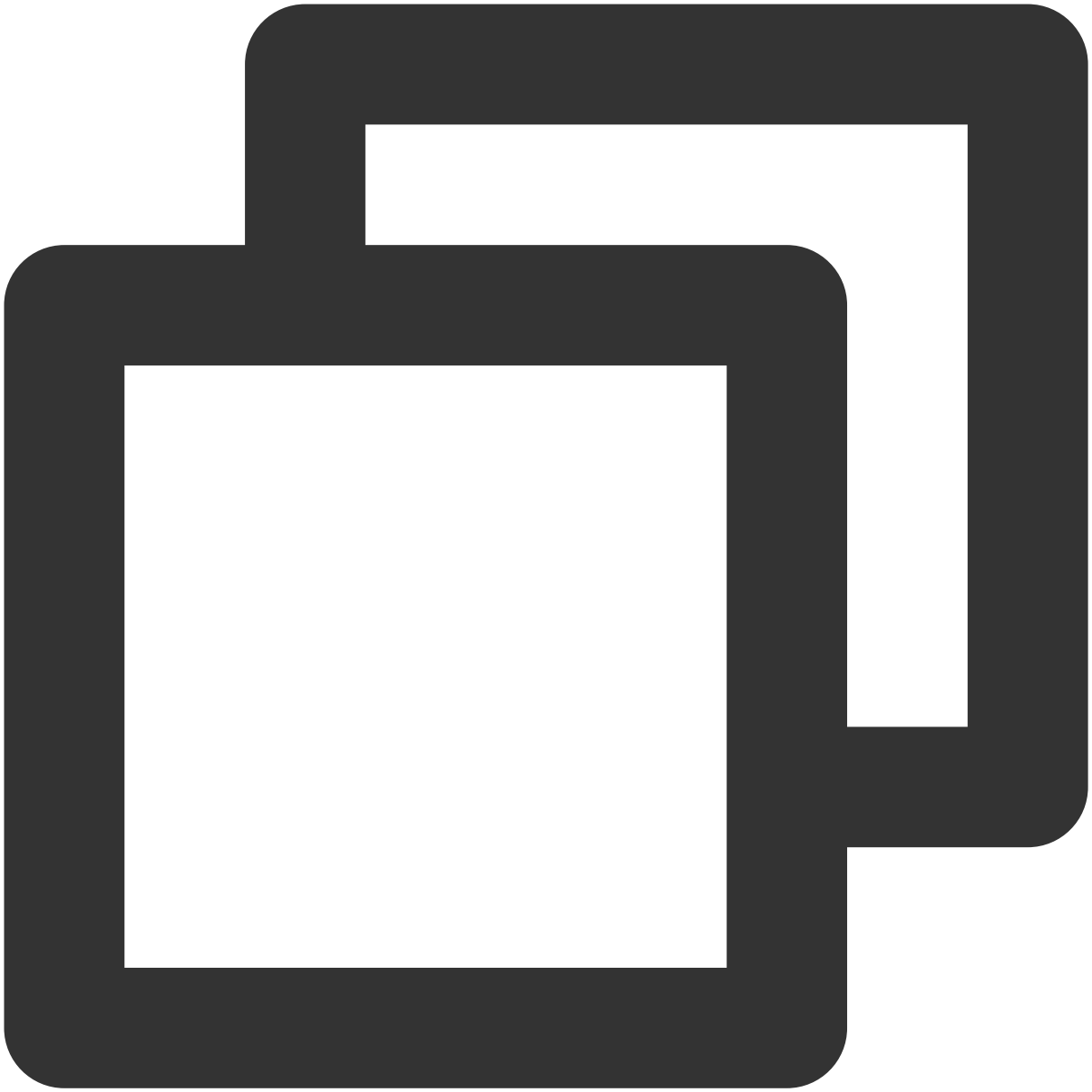
设置美颜和特效

最近更新时间：2024-05-08 16:30:10

SDK 内置提供**美颜**、**滤镜**、**特效**几类效果，均支持 Web 端与微信小程序；其中滤镜和特效需要先获取到素材列表再通过素材的 EffectId 在 SDK 中设置。

美颜

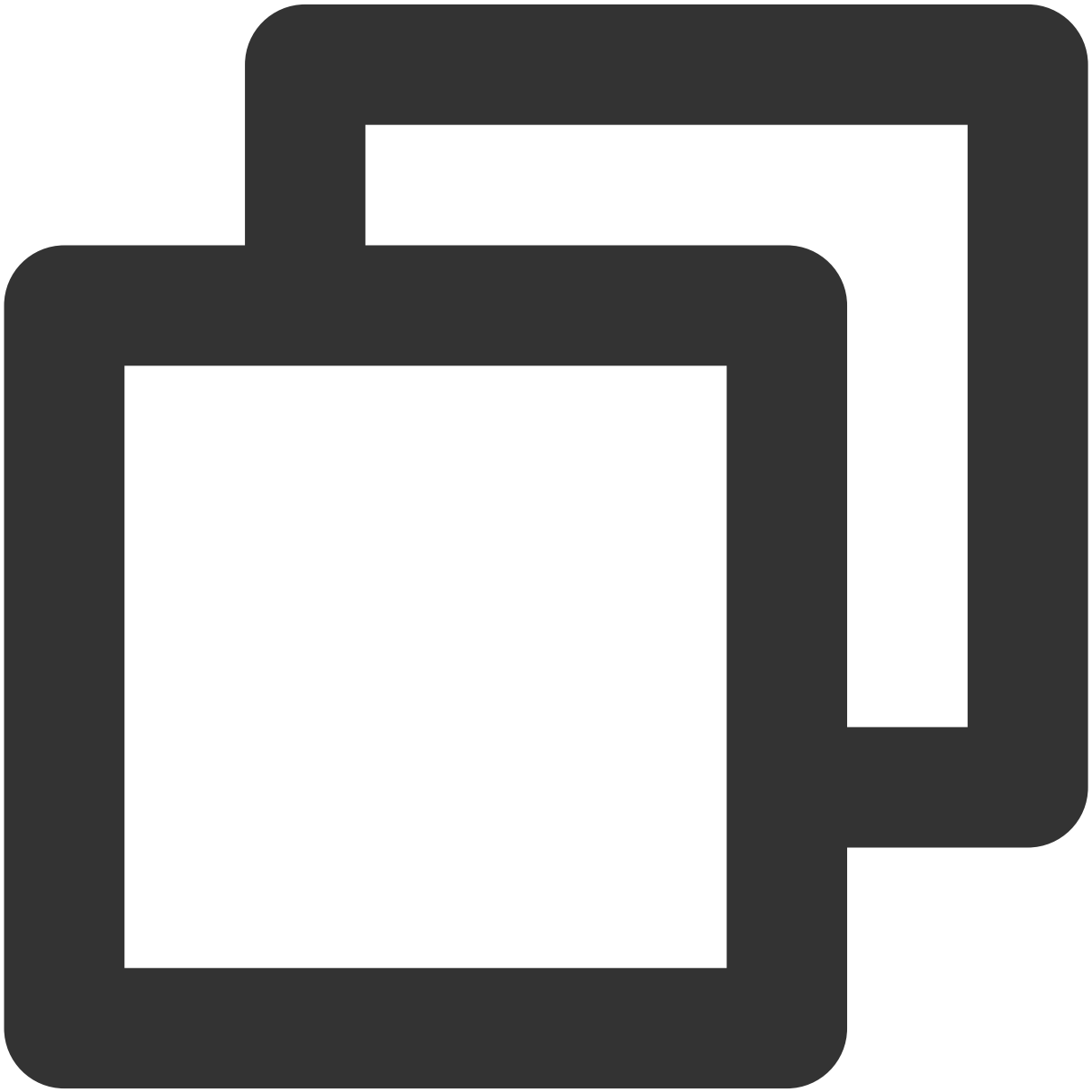
除了初始化时可以传入美颜参数，也可以通过 ArSdk 的 `setBeautify` 方法设置美颜。
SDK 目前支持的美颜效果有：



```
type BeautifyOptions = {
    whiten?: number; // 美白 0-1
    dermabrasion?: number; // 磨皮0-1
    lift?: number; // 窄脸0-1
    shave?: number; // 削脸0-1
    eye?: number; // 大眼0-1
    chin?: number; // 下巴0-1
    // 注意：以下参数仅在1.0.11及以上版本可用
    darkCircle?: number; // 黑眼圈0-1
    nasolabialFolds?: number; // 法令纹0-1
    cheekbone?: number; // 颧骨0-1
```

```
    head?: number; // 小头0-1
    eyeBrightness?: number; // 亮眼0-1
    lip?: number; //嘴唇 -1 - 1
    forehead?: number; //发际线 0-1
    nose?: number; // 鼻子 -1 - 1
    usm?: number; // 清晰 0-1
}
```

调用 SDK 实例的 `setBeautify` 方法设置美颜参数：



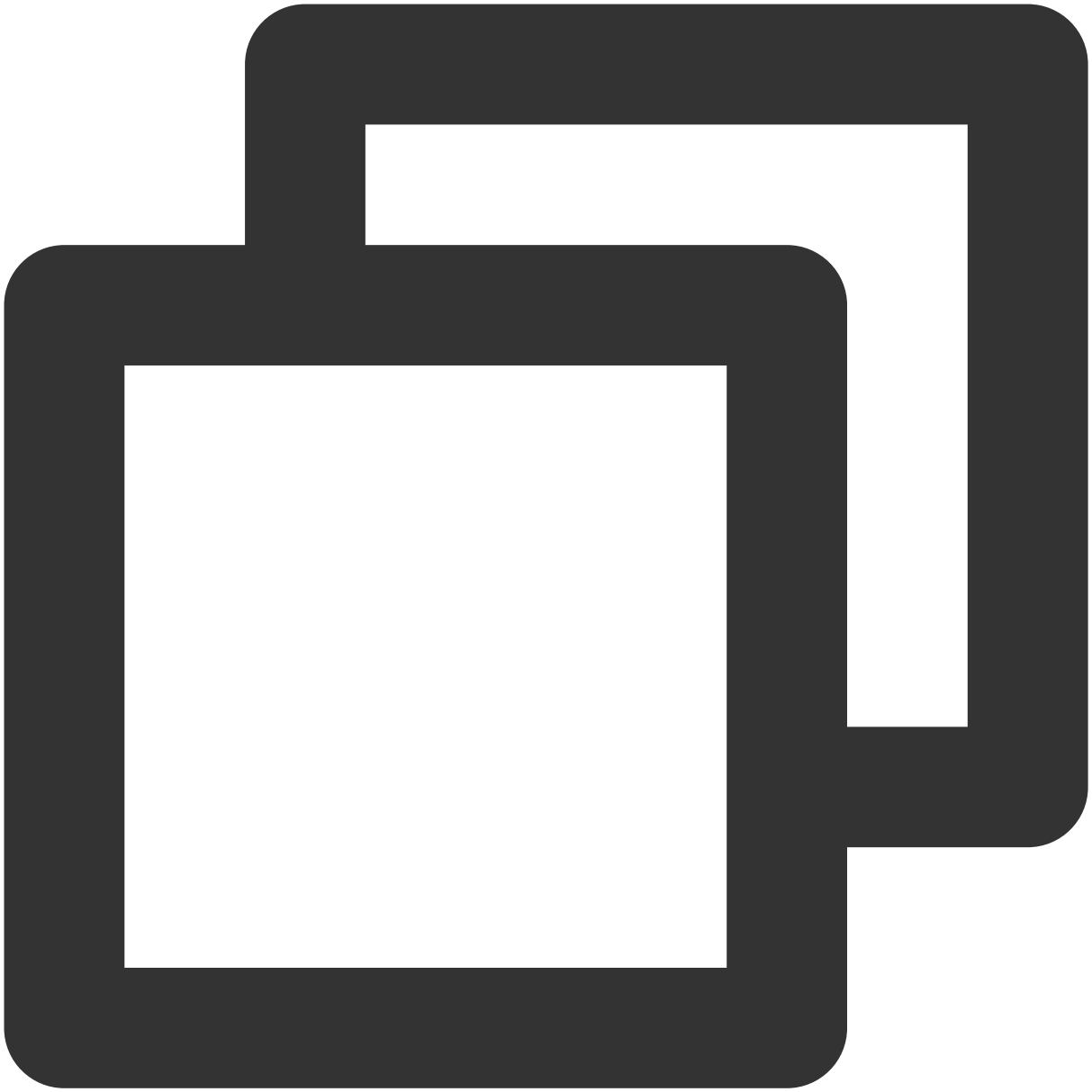
```
sdk.setBeautify({
```

```
    whiten: 0.2,
    dermabrasion: 0,
    lift: 0.3,
    shave:0.1,
    eye: 0.9,
    chin: 0,
    ……
})
```

滤镜

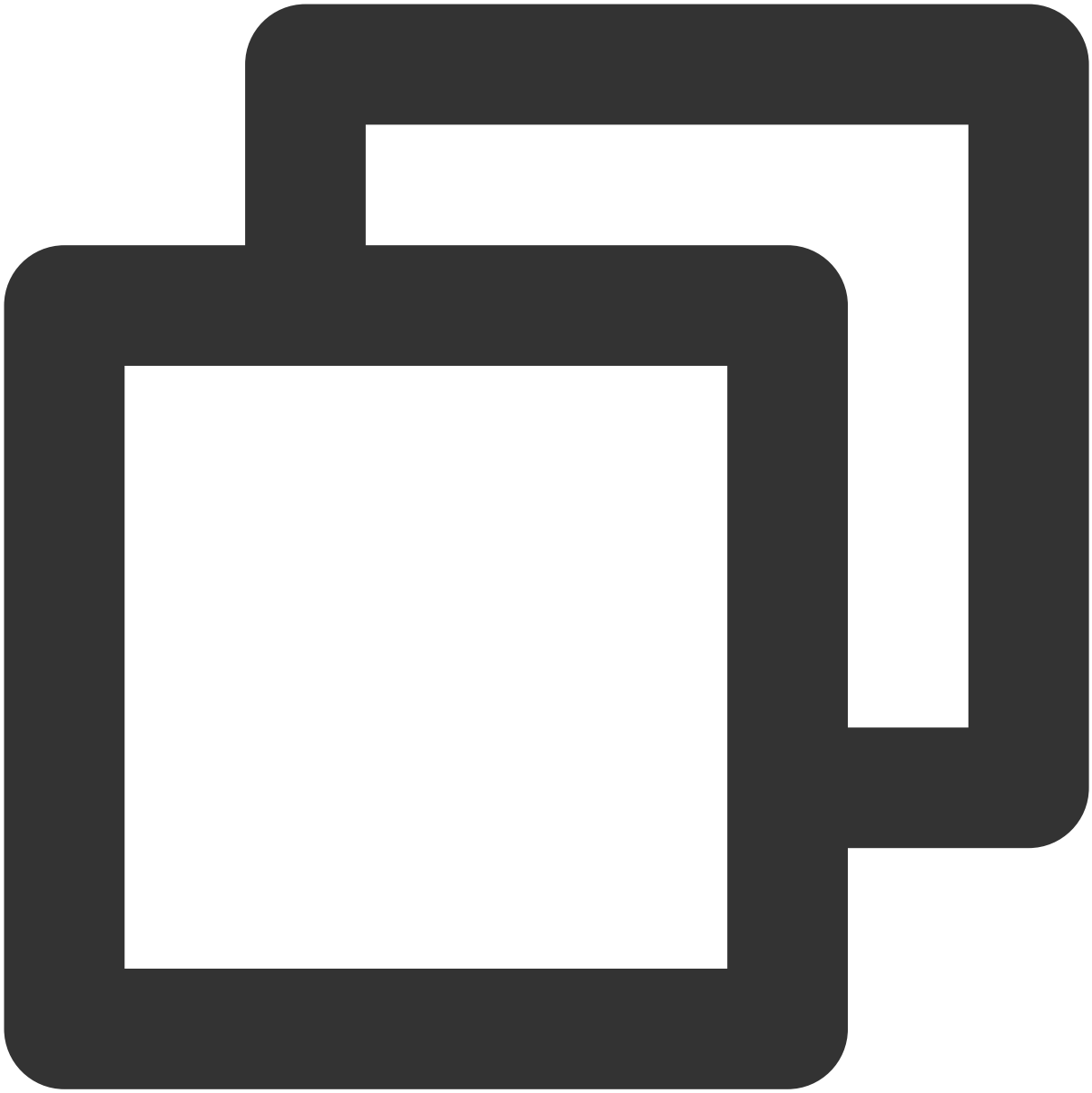
由于制作滤镜图的成本较高，我们提供了一些内置滤镜供您直接使用。

1. 获取内置滤镜列表：



```
const filterList = await sdk.getCommonFilter()
```

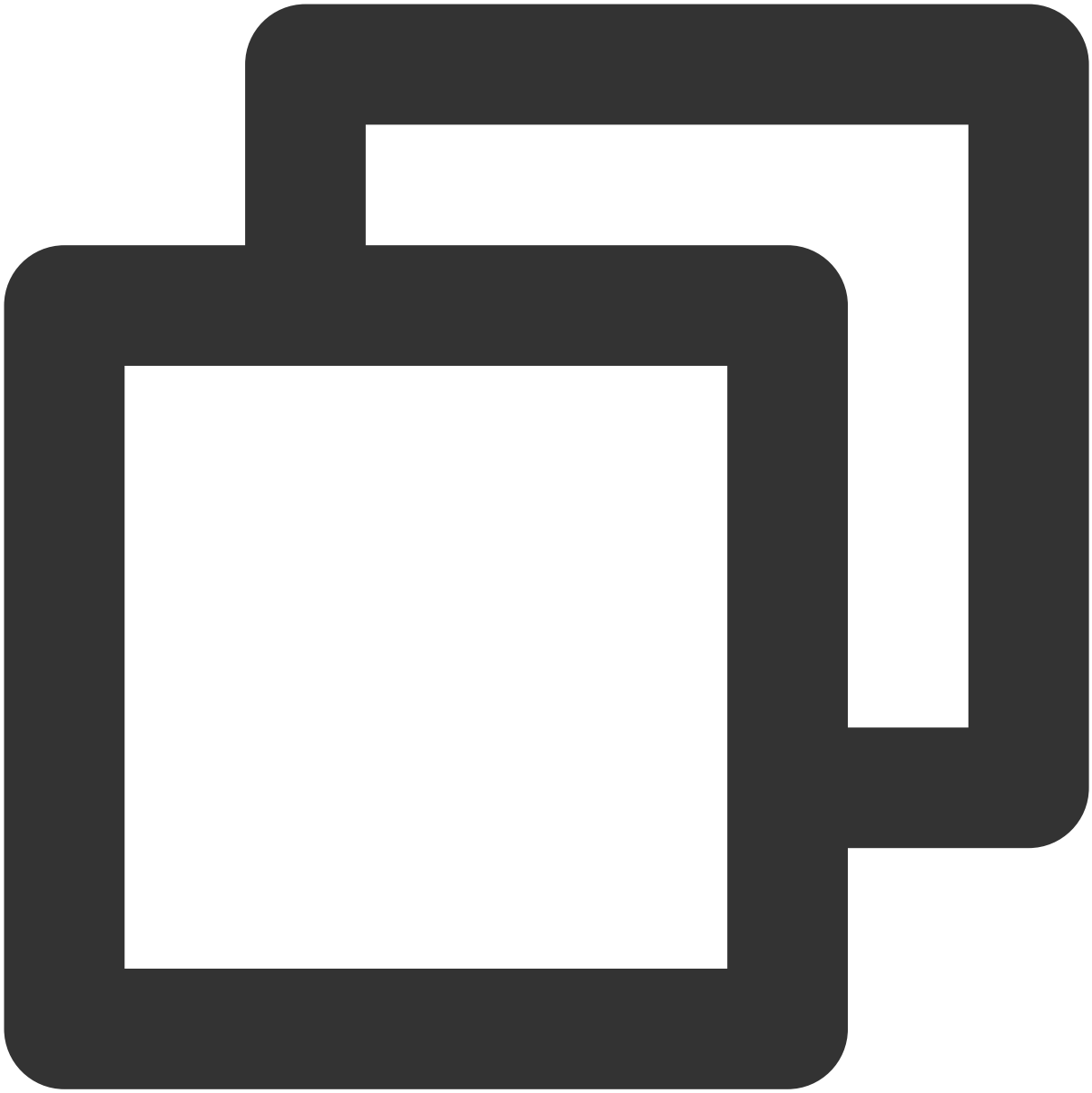
2. 设置滤镜：



```
sdk.setFilter(filterList[0].EffectId, 0.5)
```

特效

您可以直接在 SDK 中设置您在 控制台 上制作的特效，也可以使用我们提供的内置效果。相关教程请参见 素材制作基础。



```
// 获取内置特效
// 默认返回内置美妆和贴纸素材，通过Label参数可以指定返回特定类型的素材
const presetEffectList = await sdk.getEffectList({
    Type: 'Preset'
    // Label: ['贴纸'] 只返回内置贴纸
})

// 获取自制特效
const customEffectList = await sdk.getEffectList({
    Type: 'Custom'
})
```

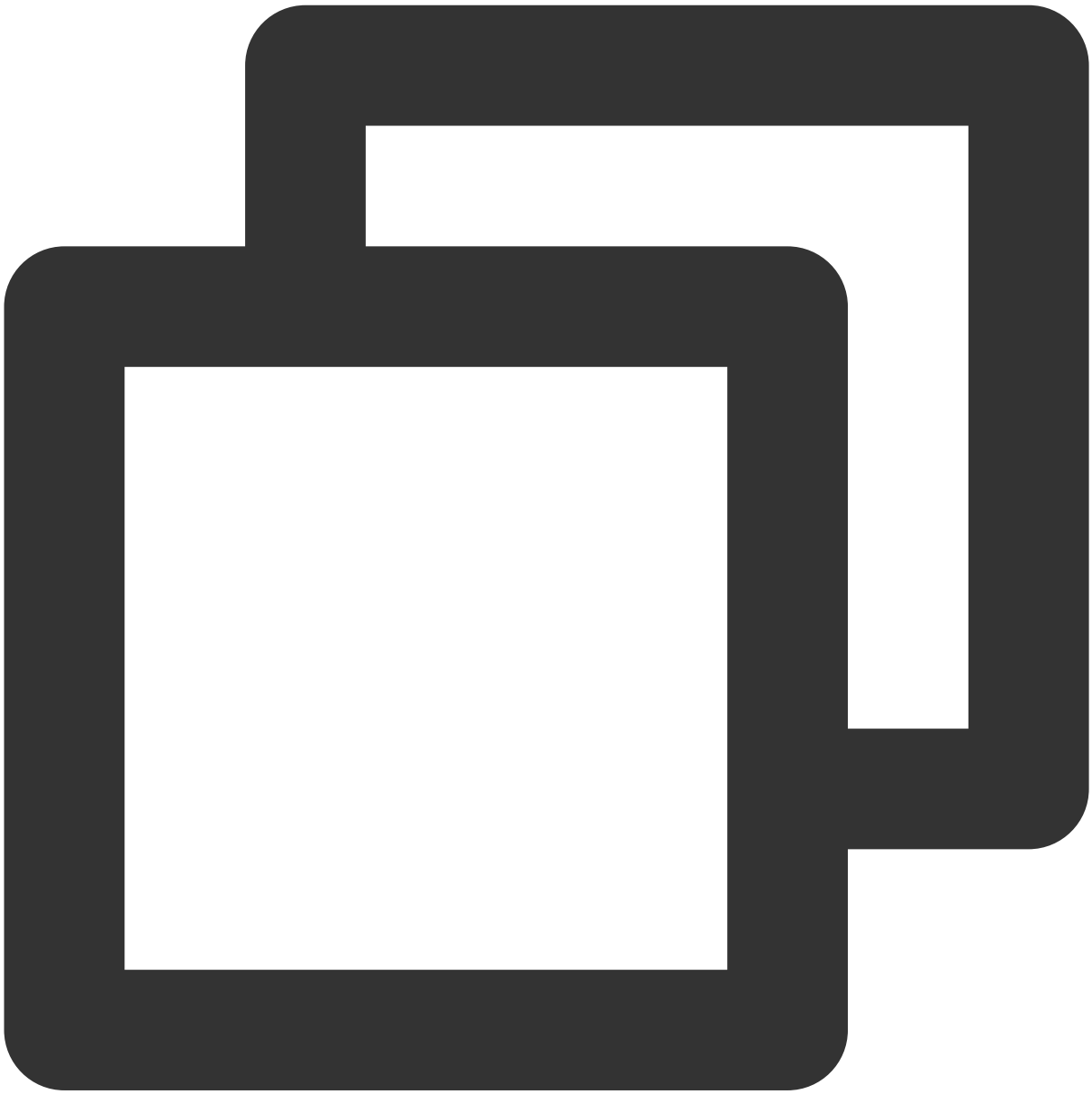
```
// 传入列表请求参数
const lipList = await sdk.getEffectList({
    PageNumber: 0,
    PageSize: 10,
    Name: '',
    Label: ['唇妆'], // 用特效的标签筛选
    Type: 'Custom'
})
const eyeList = await sdk.getEffectList({
    PageNumber: 0,
    PageSize: 10,
    Name: '',
    Label: ['眼妆'], // 用特效的标签筛选
    Type: 'Custom'
})
// 设置特效
sdk.setEffect([lipList[0].EffectId, eyeList[0].EffectId])

// 设置特效+强度
sdk.setEffect([
{
    id: lipList[0].EffectId,
    intensity: 0.5
},
{
    id: eyeList[0].EffectId,
    intensity: 0.7
})
// 单独设置特效中的滤镜强度
sdk.setEffect([
{
    id: lipList[0].EffectId,
    intensity: 0.5,
    filterIntensity: 0
},
{
    id: eyeList[0].EffectId,
    intensity: 0.7,
    filterIntensity: 1
})
```

注意：

由于特效制作工具支持在特效中添加滤镜，当多个特效叠加时，滤镜默认采用叠加处理，如需要单独控制，可使用 filterIntensity 单独控制。

关闭美颜

由于 AI 检测的性能消耗较大，为了尽可能节约资源。当 SDK 没有设置任何美颜与特效效果的时候，会自动关闭 AI 检测，再次设置效果后自动开启。



```
// 清空SDK设置的所有效果即可关闭检测
sdk.setBeautify({
    whiten: 0,
    dermabrasion: 0,
    lift: 0,
```

```
    shave:0,
    eye: 0,
    chin: 0,
    ……
})
sdk.setEffect('')
```

使用人像分割（虚拟背景）

最近更新时间：2023-04-11 16:31:24

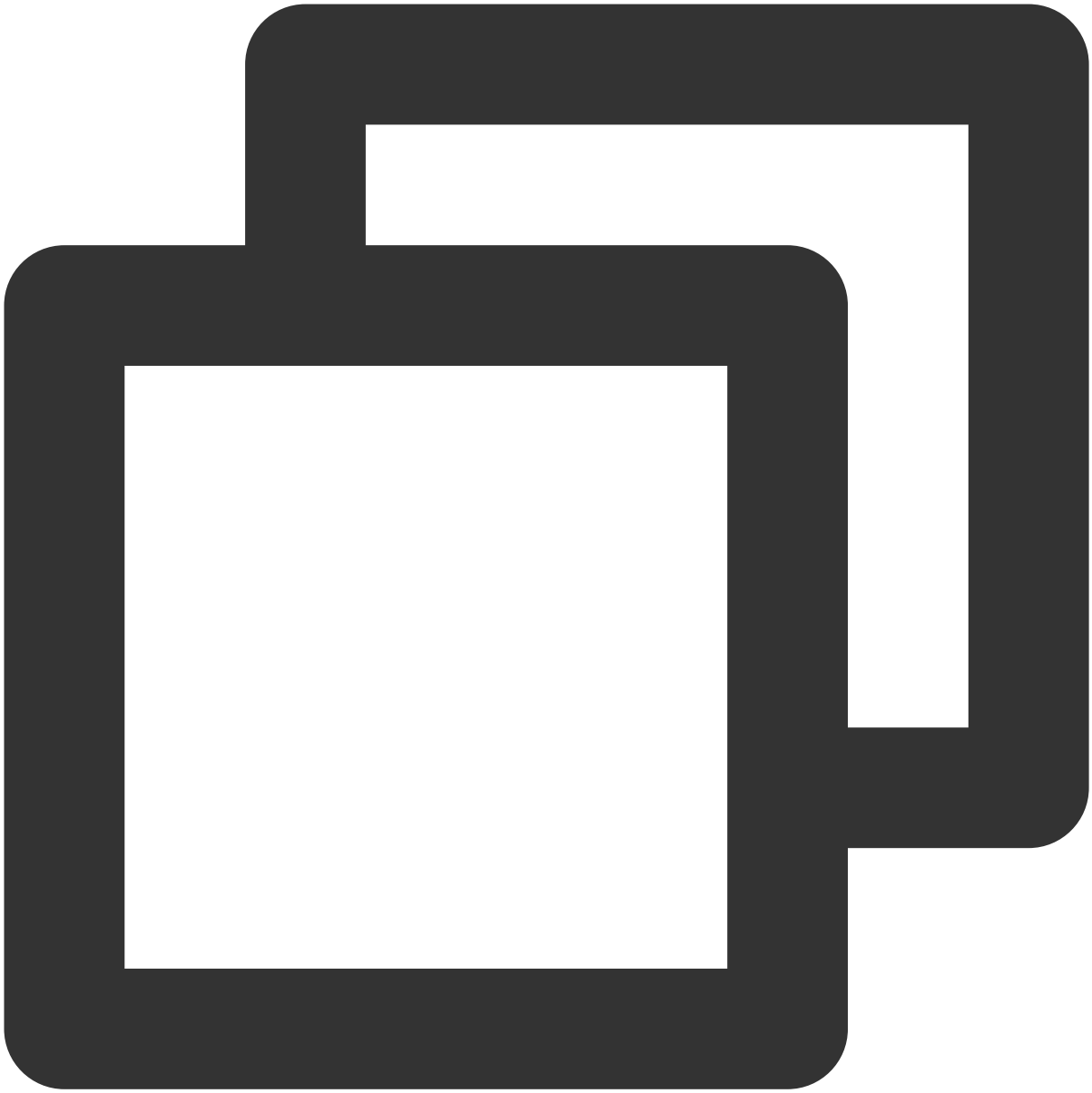
实现虚拟背景需要在初始化时启用人像分割模块，详情请参见 自定义流、内置相机。

注意：

虚拟背景暂不支持小程序。

设置背景

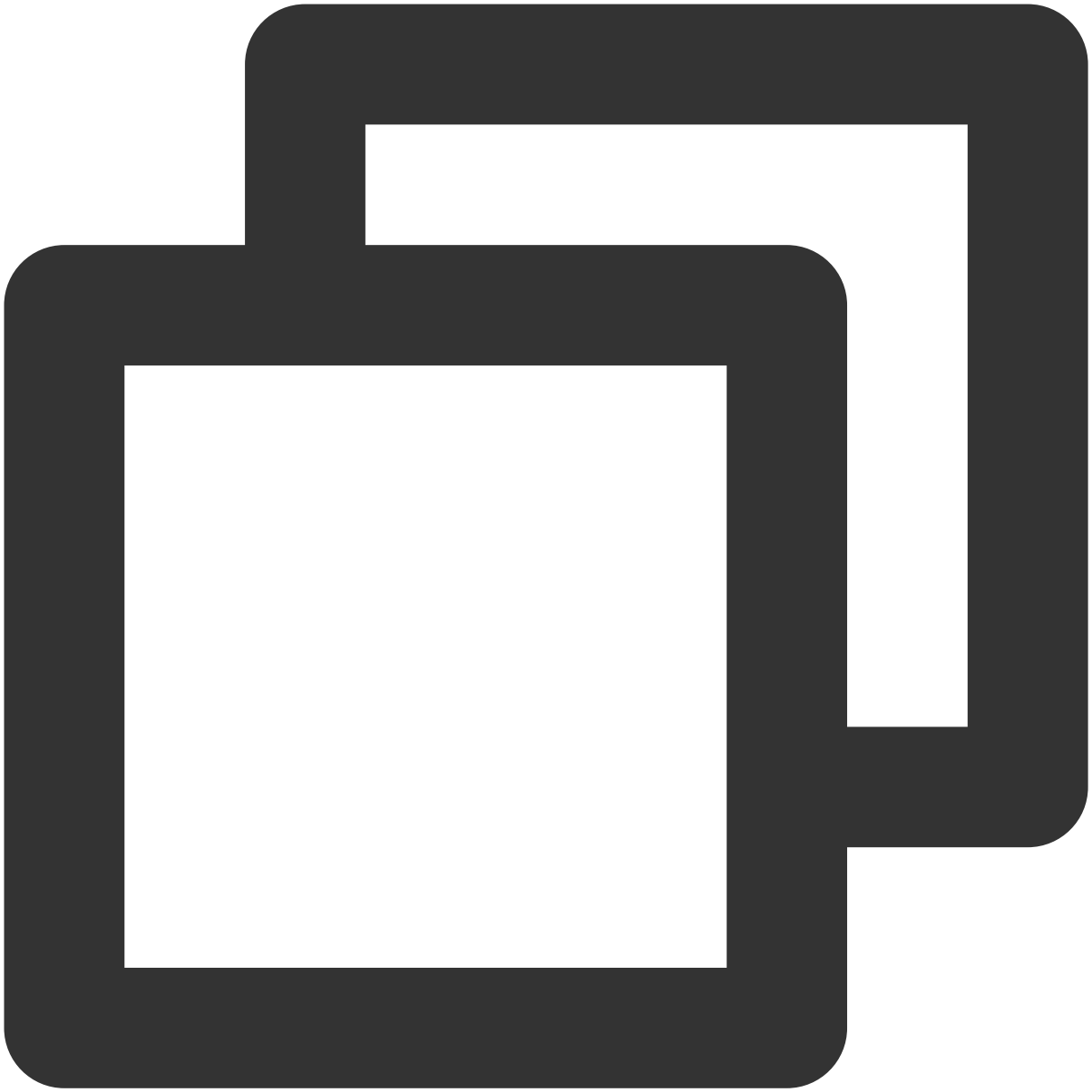
SDK 支持设置模糊背景、图片背景，支持在初始化参数中透传：



```
const config = {
  module: {
      beautify: true, // 是否启用美颜模块，启用后可以使用美颜、美妆、贴纸等功能
      segmentation: true // 是否启用人像分割模块，启用后可以使用背景功能
  },
  auth: authData, // 鉴权参数
  input: stream, // input传输入流
  beautify: { // 初始化美颜参数(可选)
      whiten: 0.1,
      dermabrasion: 0.3,
      eye: 0.2,
```

```
        chin: 0,
        lift: 0.1,
        shave: 0.2
    },
    background: {
        type: 'blur' // 模糊背景
    }
}
const sdk = new ArSdk(
    // 传入一个 config 对象用于初始化 sdk
    config
)
```

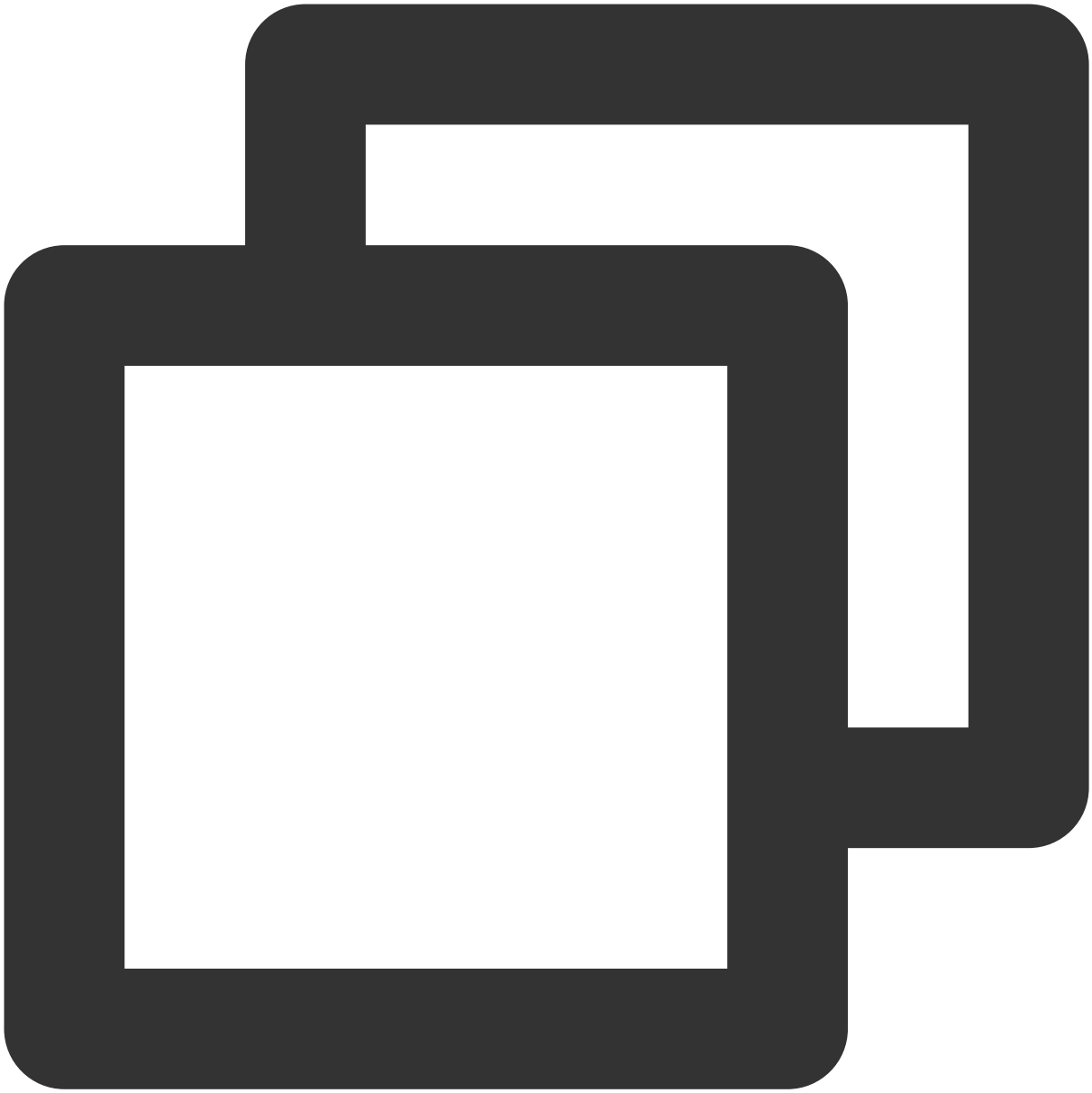
SDK 也支持动态修改背景：



```
sdk.setBackground({
  type: 'image', // 图片背景
  src: 'https://webar-static.tencent-cloud.com/assets/background/1.jpg'
})
```

透明背景

SDK 在部分浏览器中支持透明背景：



```
sdk.setBackground({
    type: 'transparent'
})
```

注意

由于浏览器的兼容性问题，请注意以下几点：

人像分割同时支持移动端及桌面端浏览器。

仅能在本地处理并展示透明背景，WebRTC 不支持编码透明通道，推流会使透明背景失效。

透明背景效果仅支持桌面端的 Chrome/Firefox，桌面端 Safari 及 iOS 端均不支持。

使用表情和虚拟形象

最近更新时间：2023-04-11 16:32:01

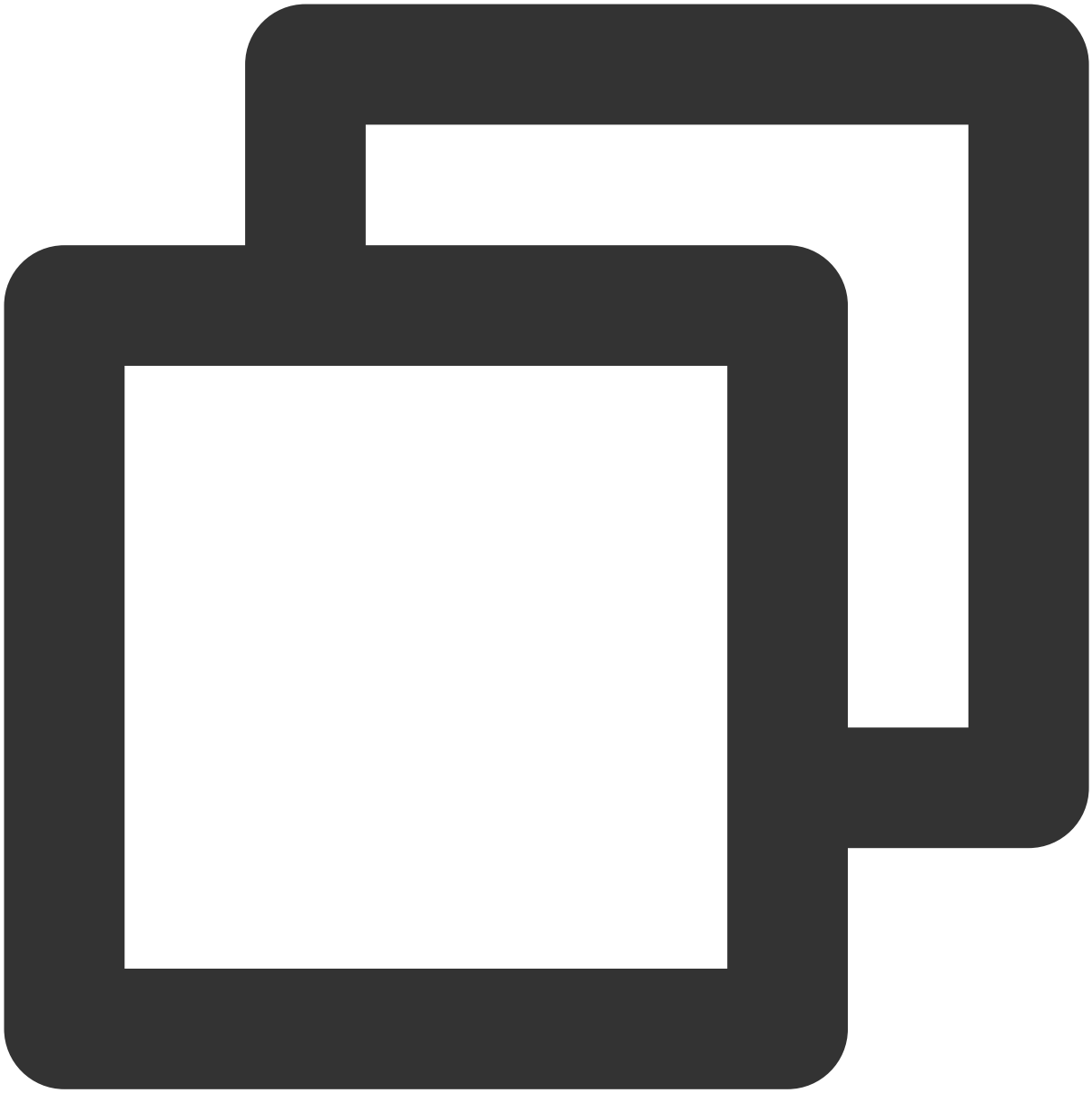
美颜特效 SDK 从0.3.0版本支持 Animoji 表情和 VR 虚拟形象功能。

注意：

Animoji 表情和虚拟形象暂不支持小程序。

检测是否支持

Animoji 表情和 VR 虚拟形象仅可在支持 WebGL2 的环境下使用，SDK 提供了检测的静态方法以供判断。

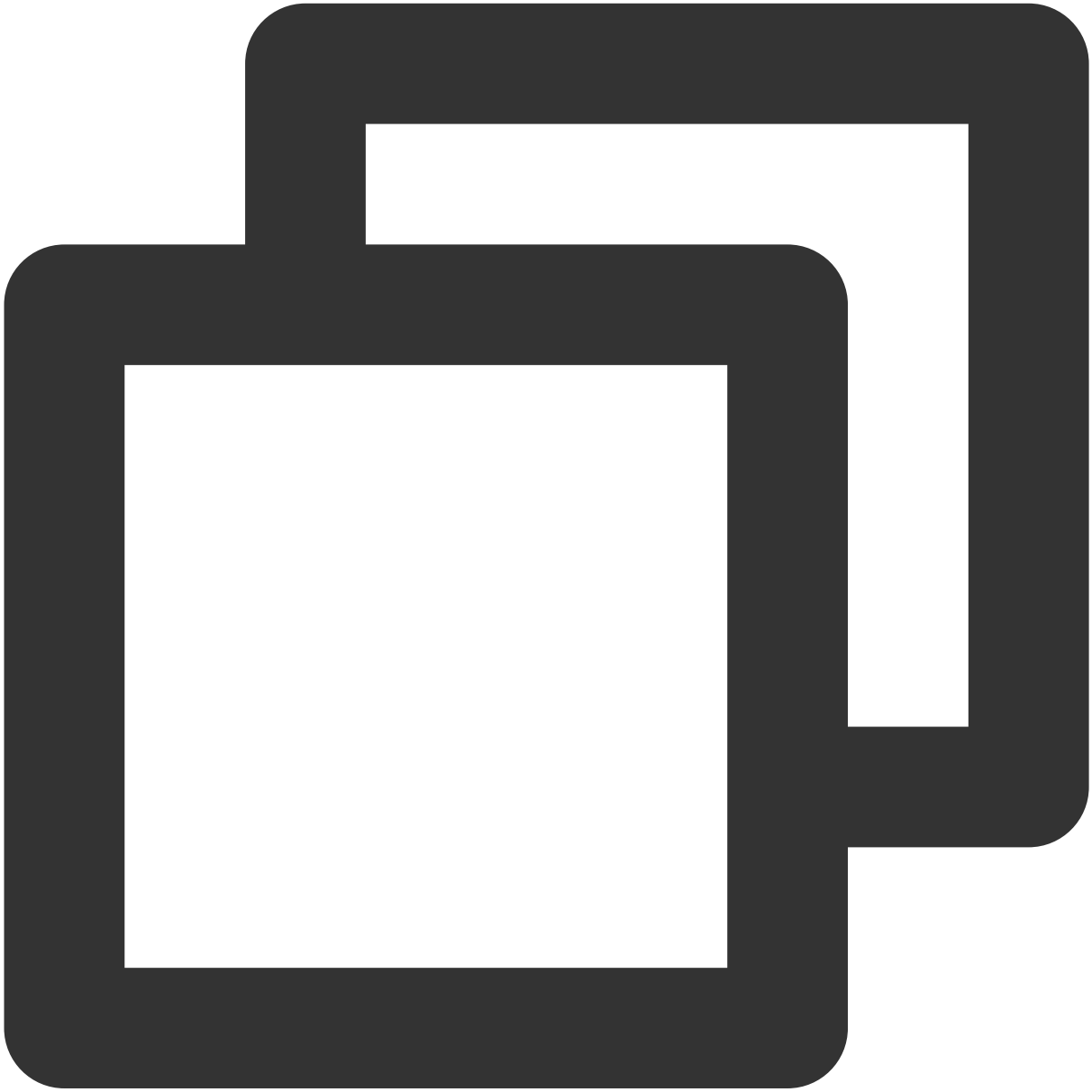


```
import {ArSdk} from 'tencentcloud-webar'
if (ArSdk.isAvatarSupported()) {
    // 初始化相关功能

} else {
    alert('当前浏览器不支持虚拟形象')
    // 隐藏相关功能
}
```

Animoji 表情

获取模型列表

SDK 初始化完成之后即可获取内置的模型列表，目前 SDK 内置了4个表情头像。



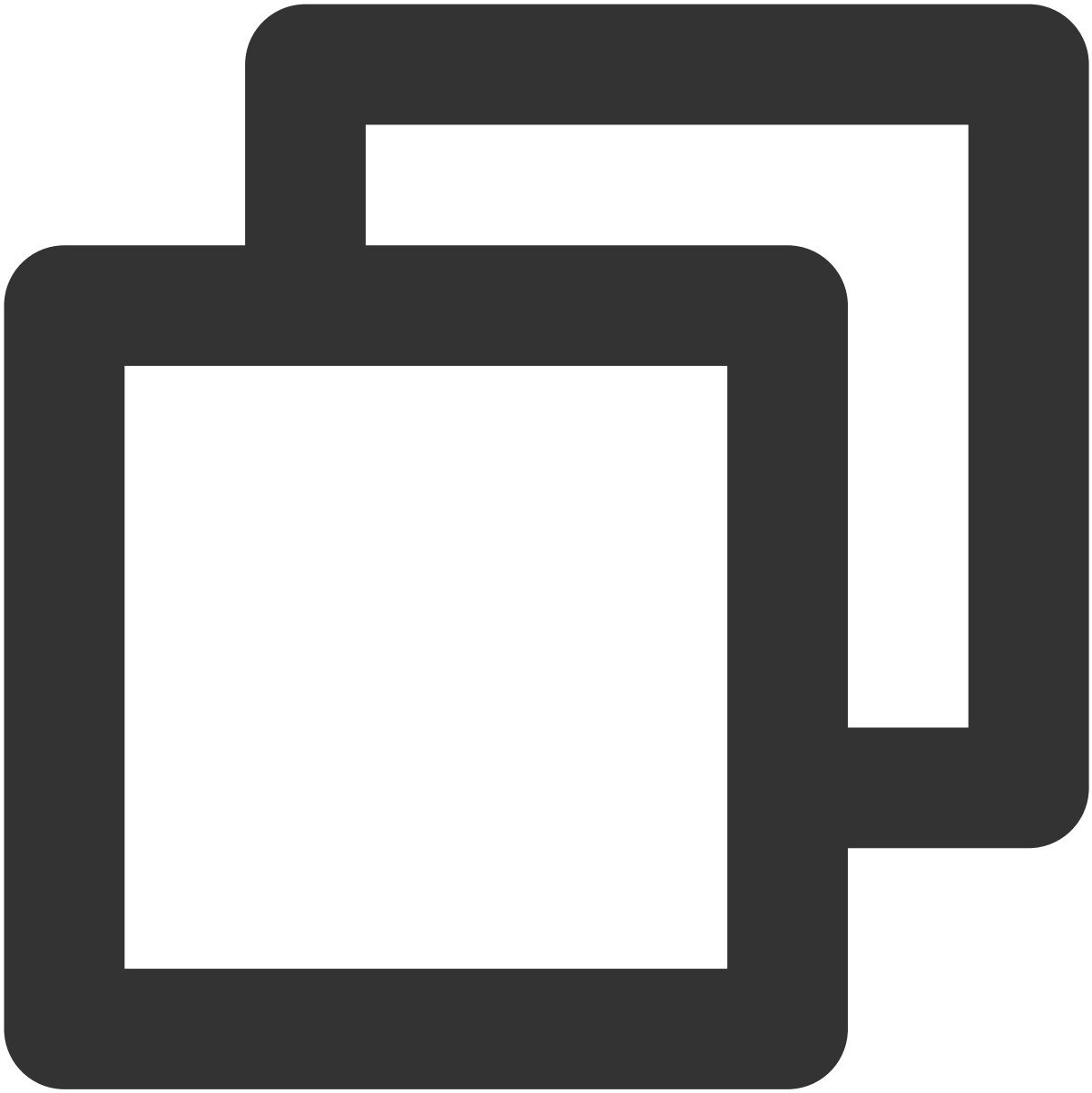
```
const avatarARList = await sdk.getAvatarList('AR')
```

注意

设置 Animoji 表情与虚拟形象会自动清除美妆、贴纸等效果，同理设置美妆与贴纸会清除表情或虚拟形象效果。

设置模型

拿到列表之后可以通过 EffectId 设置 Animoji 表情效果。



```
ar.setAvatar({
  mode: 'AR', // 模式设置为VR
  effectId: avatarARList[0].EffectId// 传内置id
}, () => {
  // success callback

});
```

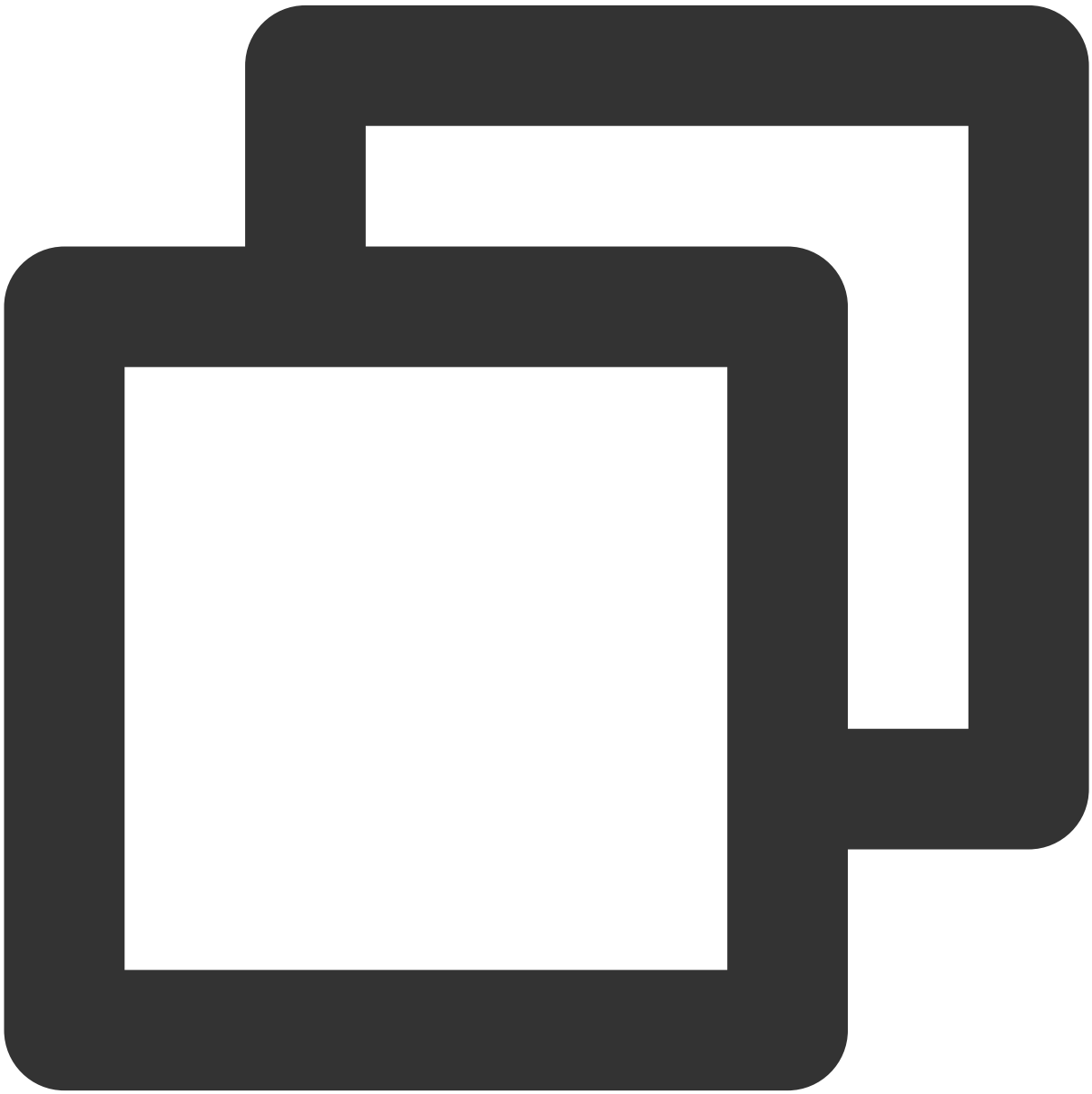
自定义模型

如有自定义模型的要求请 联系我们。

VR 虚拟形象

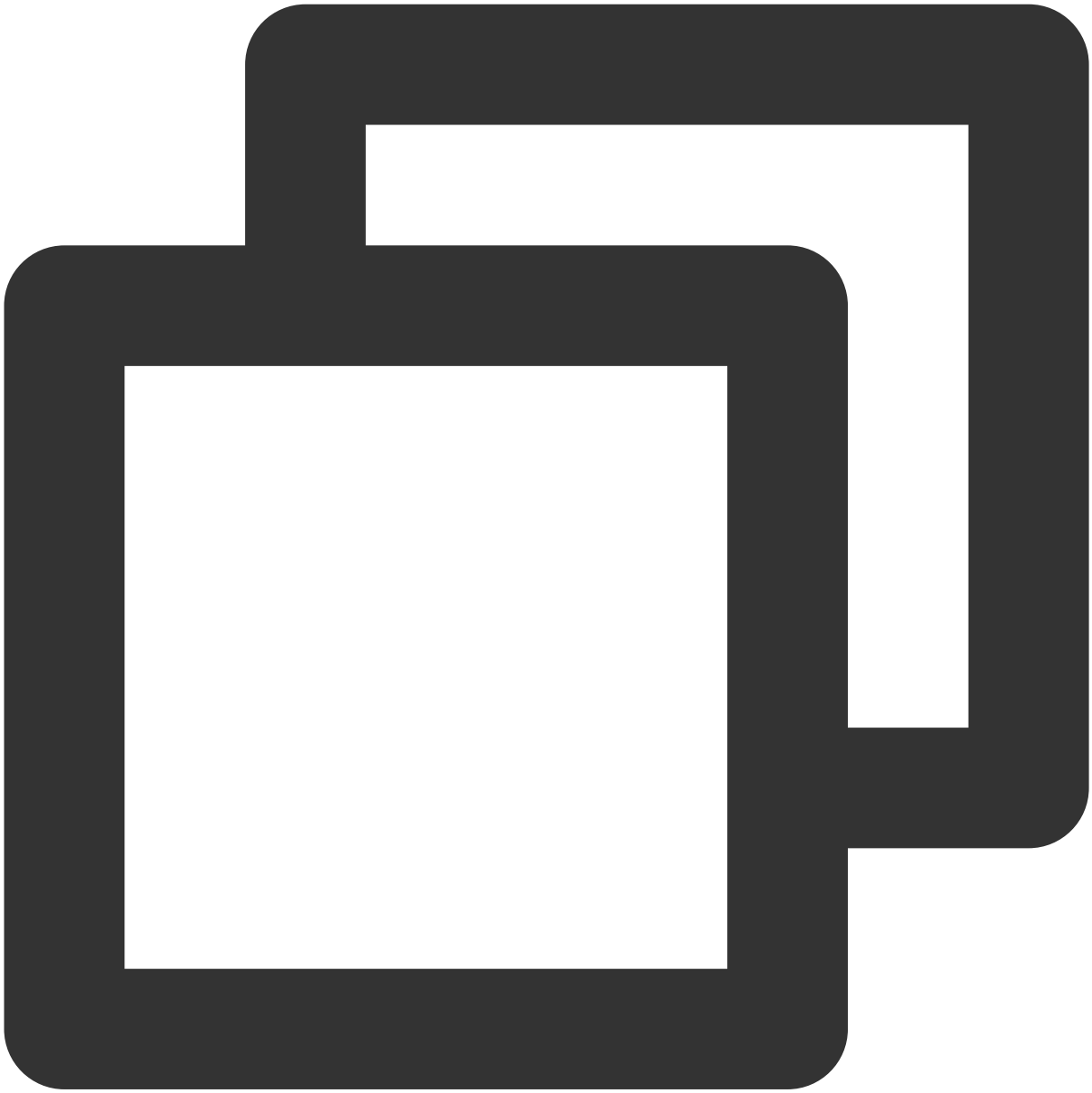
获取模型列表

SDK 初始化完成之后即可获取内置的模型列表，目前 SDK 内置了10个虚拟形象。



```
const avatarVRList = await sdk.getAvatarList('VR')
```

设置场景



```
ar.setAvatar({
  mode: 'VR', // mode传VR
  effectId: avatarVRList[0].EffectId, // 传内置id
  backgroundUrl: 'https://webar-static.tencent-cloud.com/assets/background/1.jpg',
}, () => {
```

```
    // success callback


});
```

注意

设置 VR 场景需要同步设置背景图片 URL，缺省则默认为黑色背景。

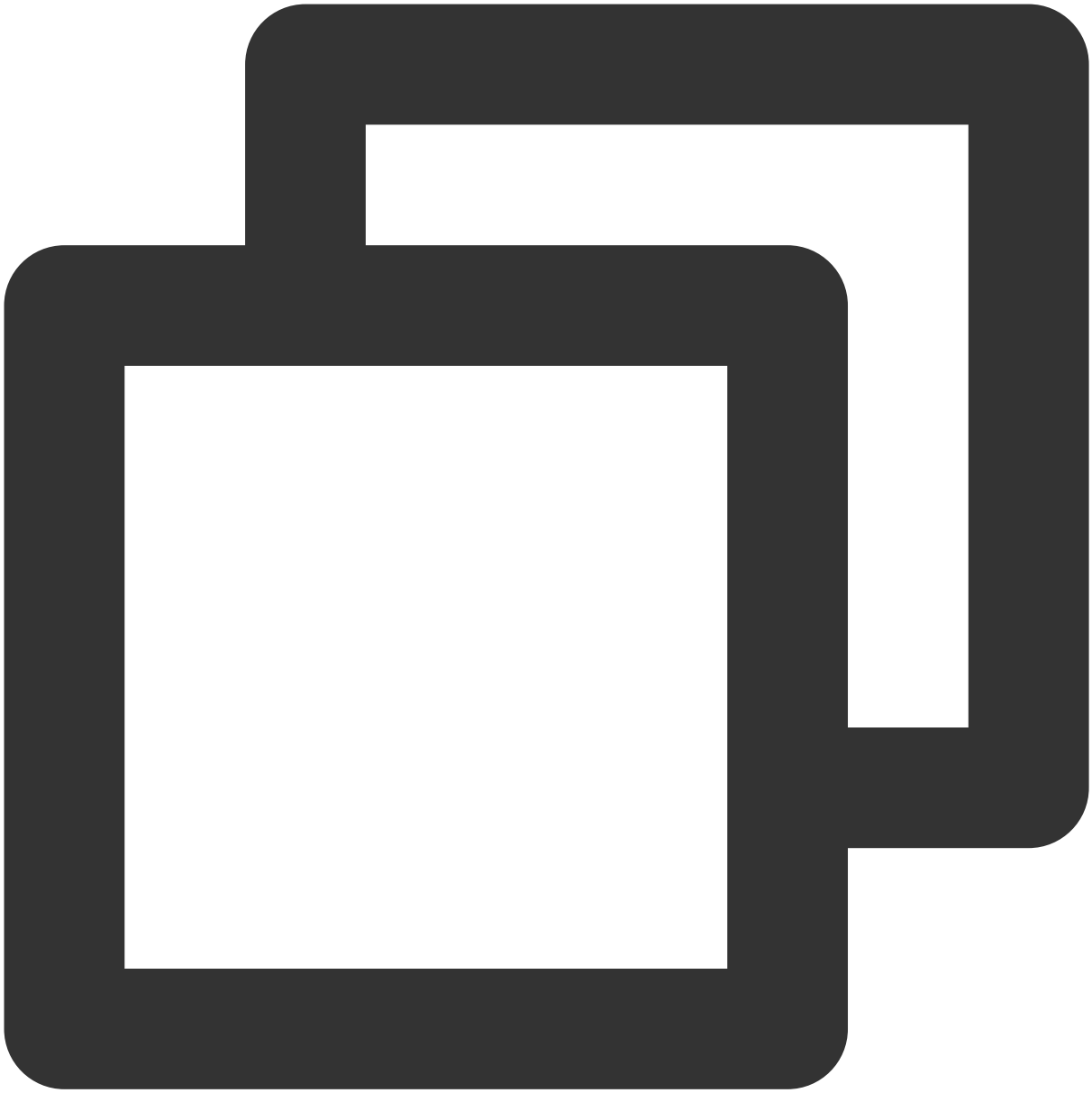
自定义模型

有两种方式可以快速定制自己的虚拟形象并直接在 SDK 中进行使用。

方式一：`readyplayer.me`

方式二：Vroid

两种方式导出的模型需自行上传至 CDN，使用 URL 设置 SDK。



```
ar.setAvatar({
  mode: 'VR', // mode传VR
  url: 'https://xxxx.glb', // 传内置id
  backgroundUrl: 'https://webar-static.tencent-cloud.com/assets/background/1.jpg',
}, () => {
    // success callback

});
```

目前自定义模型仅支持 GLB 与 VRM 格式。

小程序端接入

最近更新时间：2023-04-21 15:35:52

小程序端接入需要自备小程序，详细请参见 微信小程序文档。

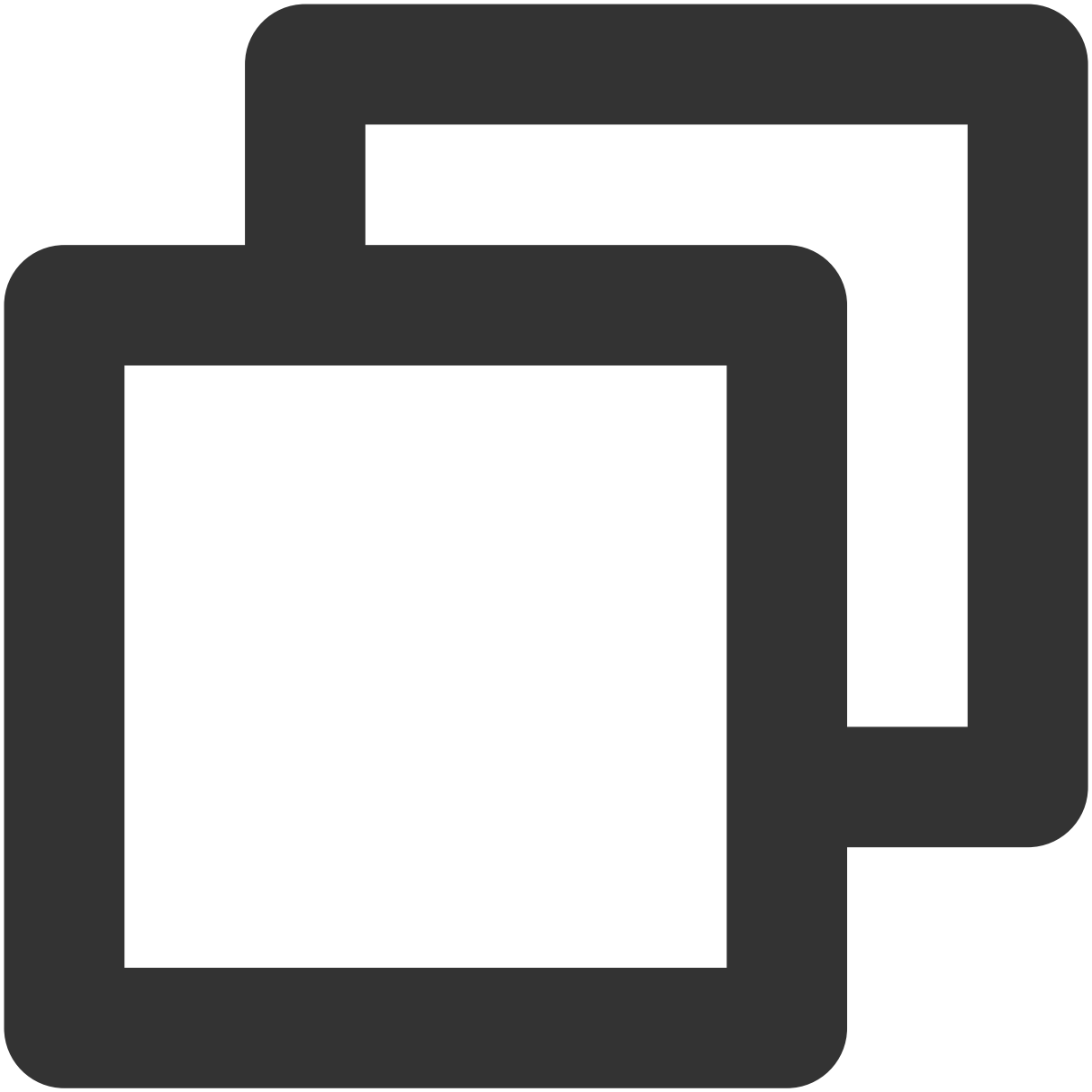
接入小程序项目

步骤1：配置域名白名单

SDK 内部会请求后台进行鉴权和资源加载，需要在小程序后台配置域名白名单。

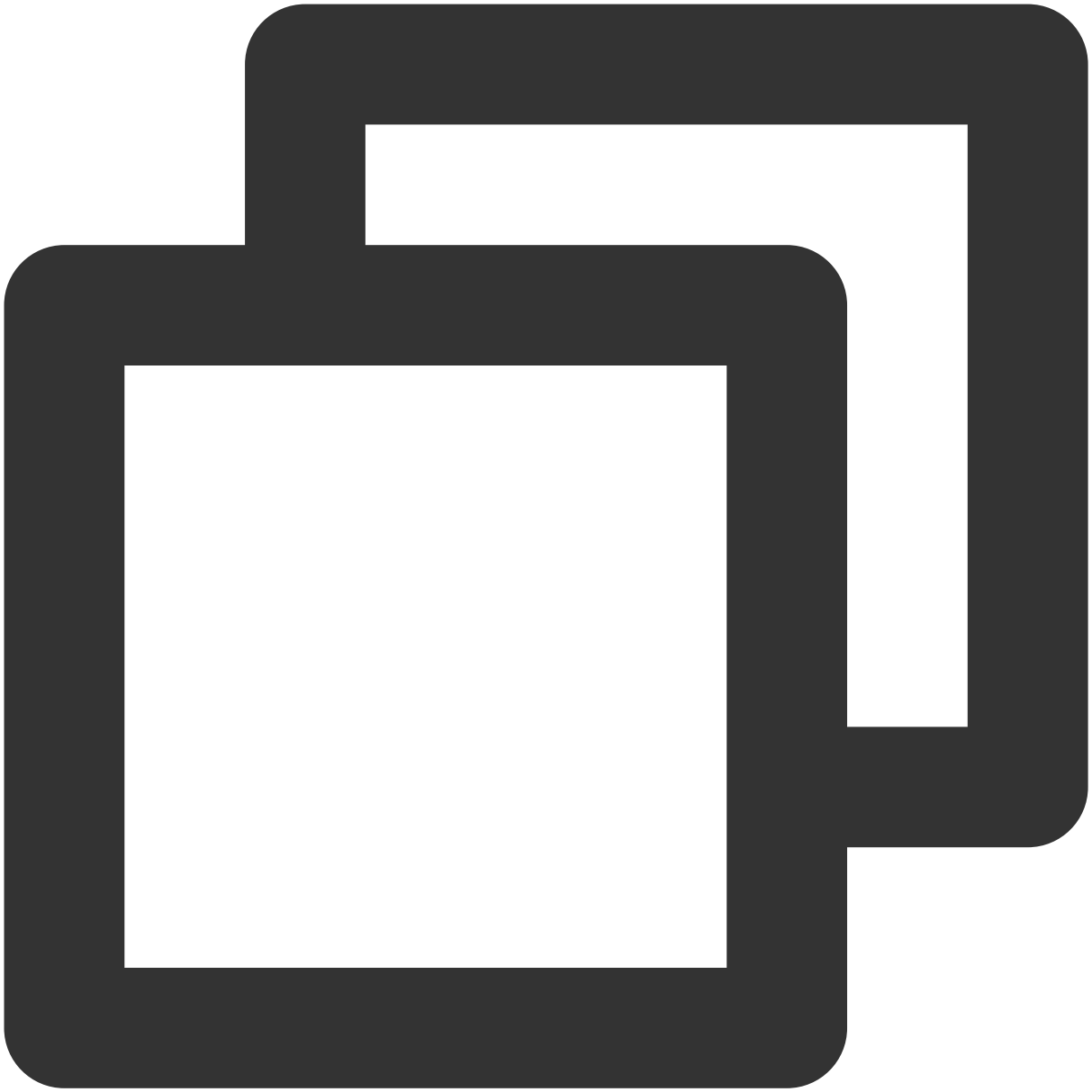
1. 进入 小程序后台，进入 **开发 > 开发管理 > 开发设置 > 服务器域名**。
2. 单击**修改**，配置以下域名并保存。

请求域名：



```
https://webar.qcloud.com;
https://webar-static.tencent-cloud.com;
https://aegis.qq.com;
以及上述签名接口（get-ar-sign）的地址
```

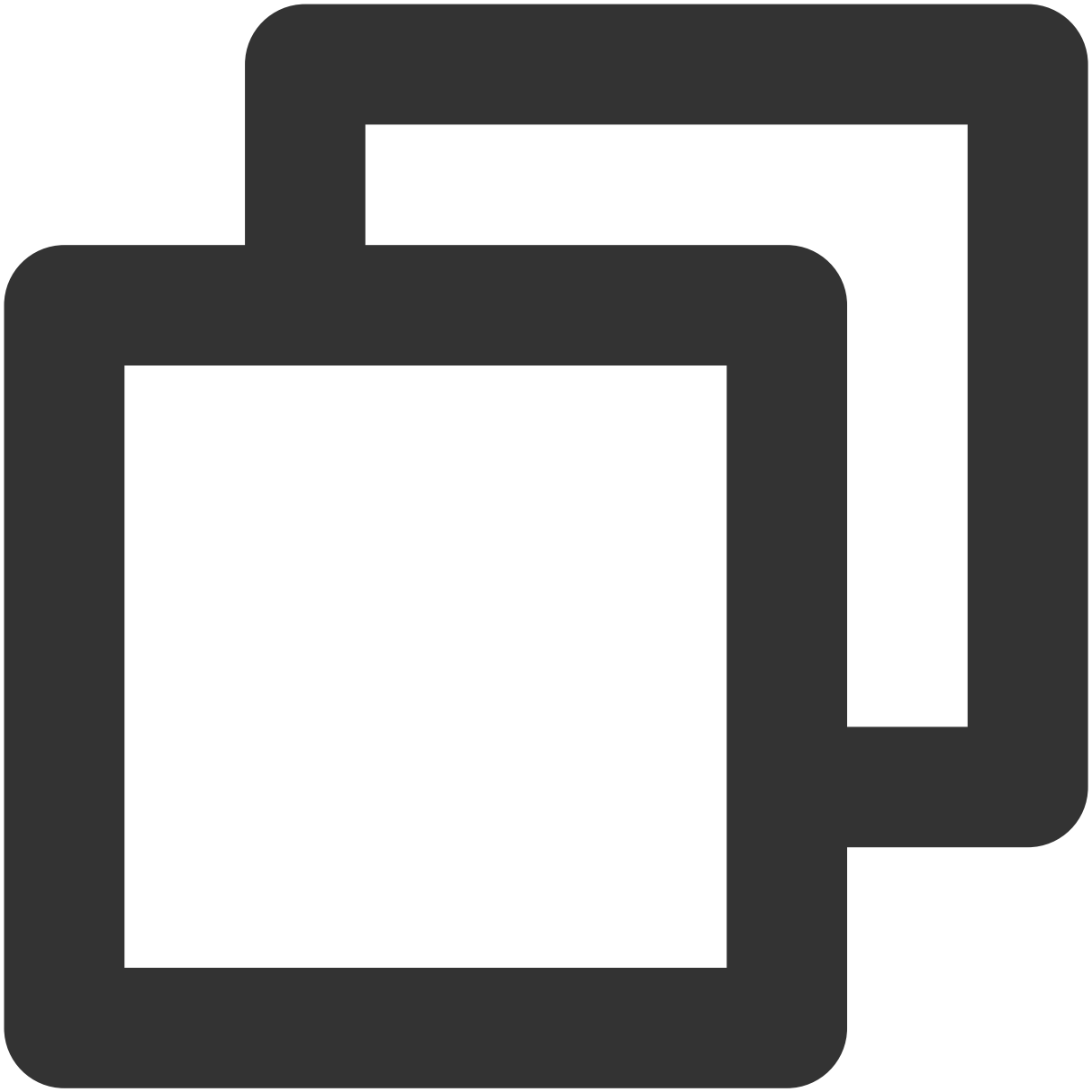
downloadFile 域名：



```
https://webar-static.tencent-cloud.com
```

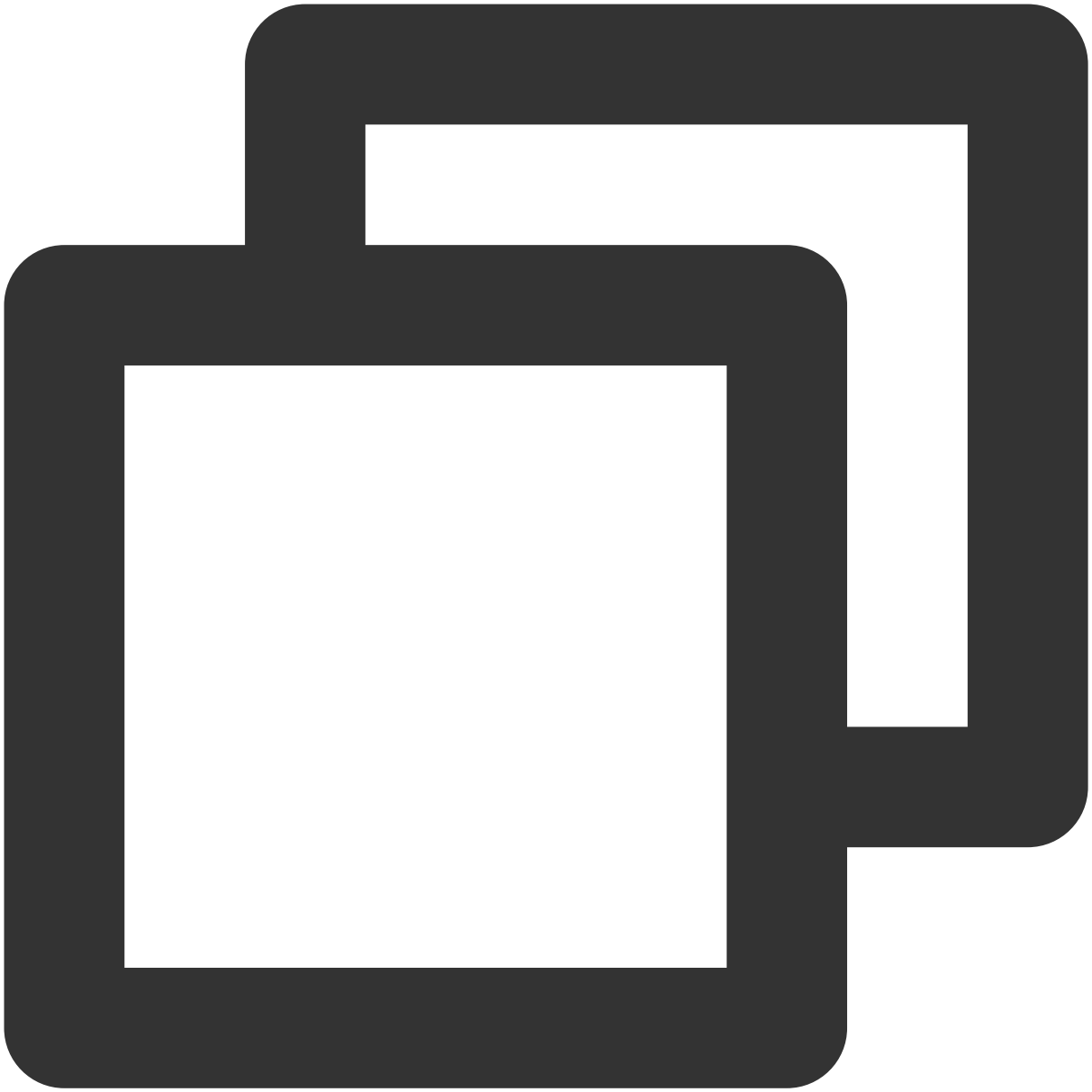
步骤2：安装并构建 npm

1. 安装：



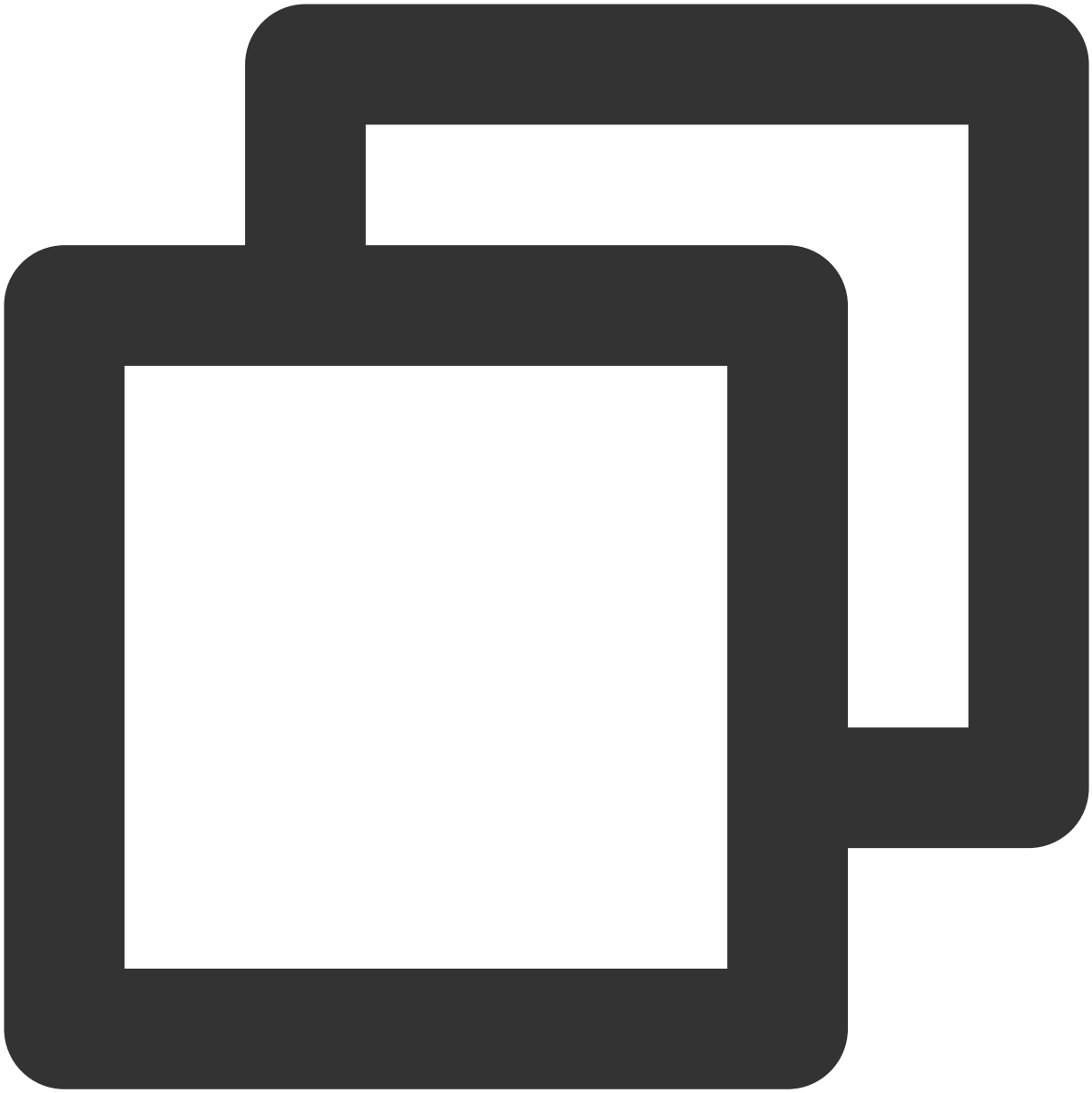
```
npm install tencentcloud-webar
```

2. 单击开发者工具**菜单** > **工具** > **构建 npm**。
3. 在 app.json 中配置 workers 路径：



```
"workers": "miniprogram_npm/tencentcloud-webar/worker"
```

步骤3：引入文件



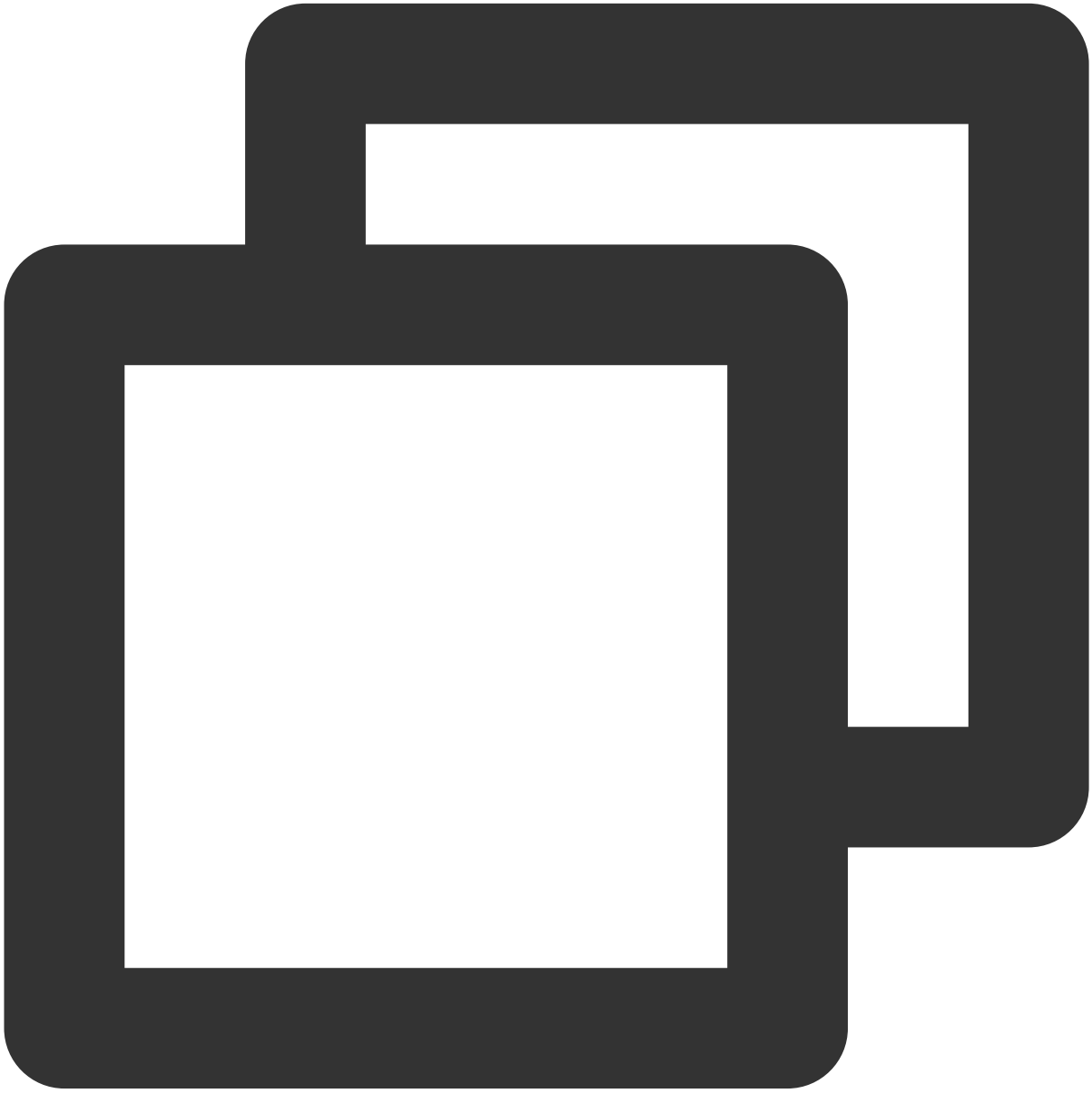
```
// 0.3.0之前版本引用方式（1个文件）
// import "../../miniprogram_npm/tencentcloud-webar/lib.js";
// 0.3.0及之后版本引用方式（2个文件 + 按需初始化3d模块）
import '../../miniprogram_npm/tencentcloud-webar/lib.js';
import '../../miniprogram_npm/tencentcloud-webar/core.js';
// 按需初始化3d插件，如果不需要3d则可以不引用
import '../../miniprogram_npm/tencentcloud-webar/lib-3d.js';
import { plugin3d } from '../../miniprogram_npm/tencentcloud-webar/plugin-3d'
// 导入 ArSdk
import { ArSdk } from "../../miniprogram_npm/tencentcloud-webar/index.js";
```

注意

小程序有单文件不超过 500kb 的限制，因此 SDK 分为多个 js 文件提供。

0.3.0版本之后，对 SDK 进行了进一步的拆分，新增3D支持，针对3D模块提供按需加载方式，导入前请确认当前使用的 SDK 版本信息，选择对应的导入方式。

步骤4：初始化实例



```
// wxml
//打开相机
<camera
```

```
    device-position="{{'front'}}"
    frame-size="large" flash="off" resolution="medium"
    style="width: 750rpx; height: 134rpx;position:absolute;top:-9999px;"
/>
//sdk 将处理完的画面实时输出到此 canvas 上
<canvas
    type="webgl"
    canvas-id="main-canvas"
    id="main-canvas"
    style="width: 750rpx; height: 1334rpx;">
</canvas>
//拍照将 ImageData 对象绘制到此 canvas 上
<canvas
    type="2d"
    canvas-id="photo-canvas"
    id="photo-canvas"
    style="position:absolute;width:720px;height:1280px;top:-9999px;left:-9999px;">
</canvas>
// js
Component({
    methods: {
        async getCanvasNode(id) {
            return new Promise((resolve) => {
                this.createSelectorQuery()
                    .select(`#${id}`)
                    .node()
                    .exec((res) => {
                        const canvasNode = res[0].node;
                        resolve(canvasNode);
                    });
            });
        },

        // 初始化相机类型
        async initSdkCamera() {
            // 获取在屏的 canvas, sdk 会将处理完的画面实时输出到 canvas 上
            const outputCanvas = await this.getCanvasNode("main-canvas");
            const sdk = new ArSdk({
                camera: {
                    width:720,
                    height:1280,
                },
                output: outputCanvas,
                loading: {
                    enable: false,
                },
                auth: {
```

```
                    licenseKey: LICENSE_KEY,
                    appId: APP_ID,
                    authFunc: authFunc
                },
                plugin3d: plugin3d // 0.3.0之后版本写法，按需初始化3d模块，如不需要3d模块，
            });
            this.sdk = sdk
            sdk.setBeautify({
                whiten: 0.2
            });
            sdk.on('created', () => {
                // 可以在回调中处理业务逻辑，详见[参数与方法]()
            })
        }
    }
})
```

注意

小程序初始化 SDK 前须在控制台配置小程序 APPID。

由于小程序的特性与 Web 不同，因此小程序支持的 input 参数目前只有 `string` 类型的图片链接。

相机配置与 Web 相同，input 参数不传，直接设置 camera 参数，前提是页面中必须插入 camera 标签。

小程序不支持 getOutput，需要自行传入一个在屏的 WEBGL canvas，SDK 直接输出到此 canvas。

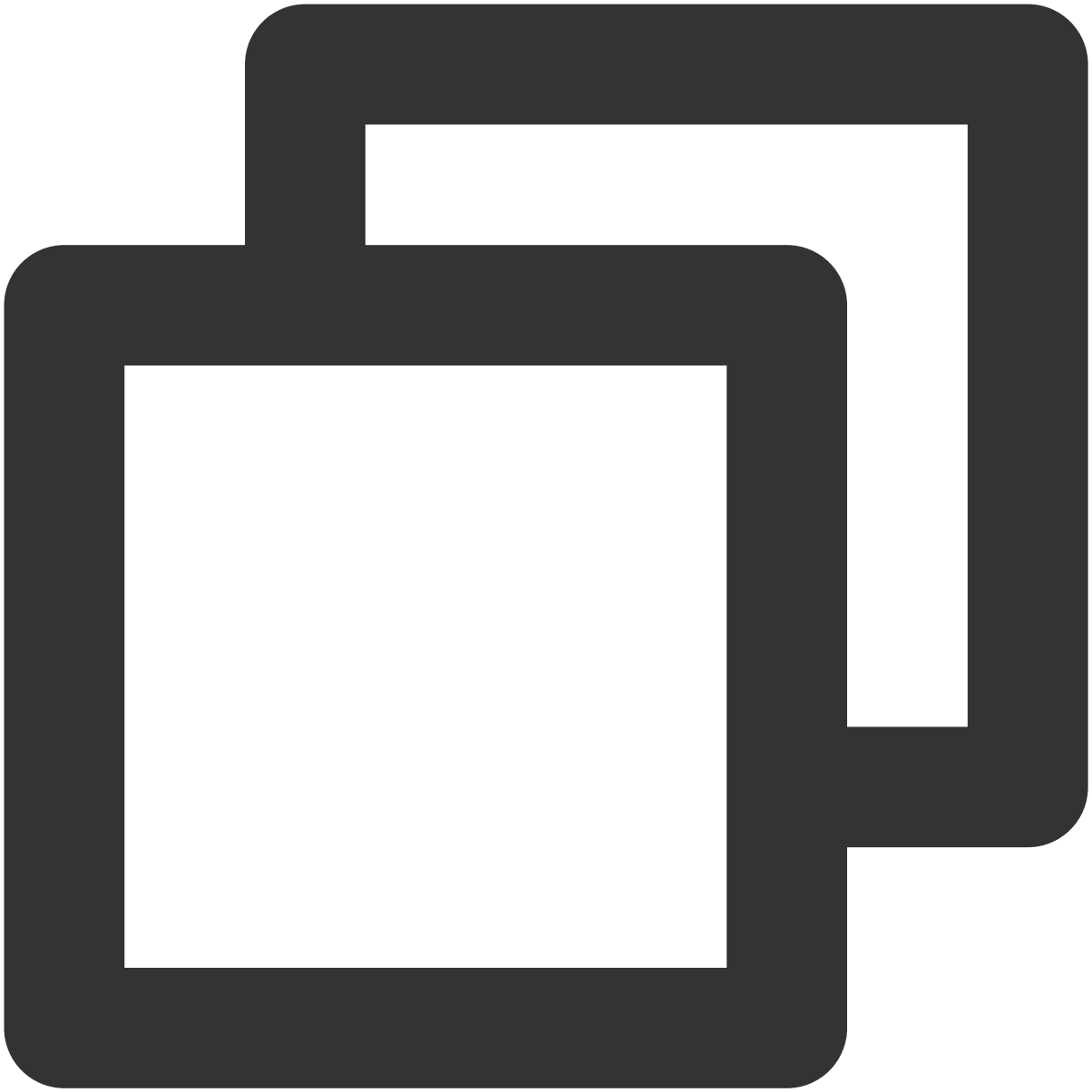
拍照与录像

小程序端支持拍照和录像功能。

拍照

录像

SDK 会返回包含宽高和 buffer 数据的对象，用户可以通过自己页面内的 2d canvas 绘制此数据并导出为图片文件。

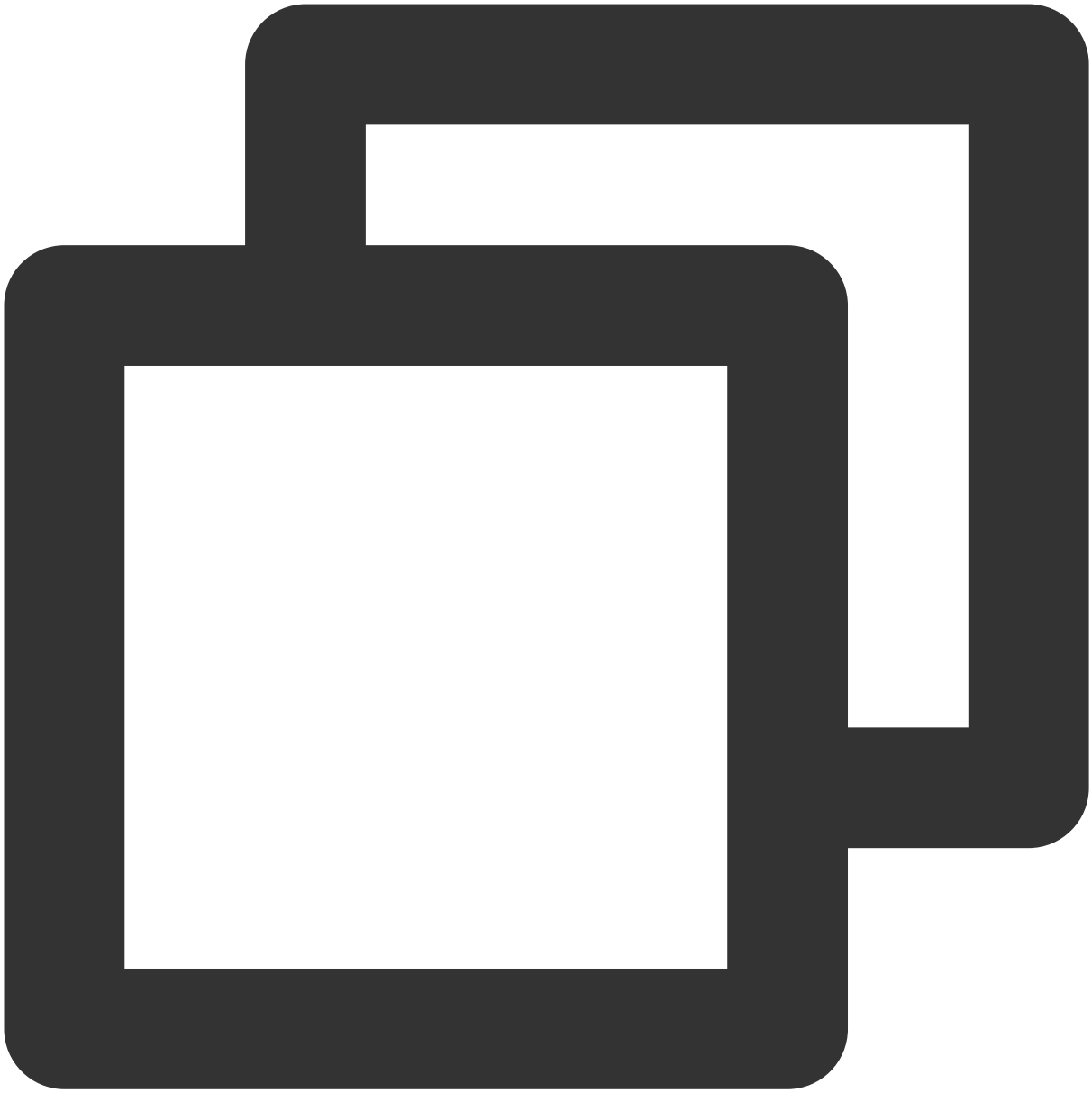


```
Component({
    ...
    async takePhoto() {
        const {uint8ArrayData, width, height} = this.sdk.takePhoto(); // takePhoto
        const photoCanvasNode = await this.getCanvasNode('photo-canvas');
        photoCanvasNode.width = parseInt(width);
        photoCanvasNode.height = parseInt(height);
        const ctx = photoCanvasNode.getContext('2d');
        // 用 sdk 返回的数据创建 ImageData 对象
        const imageData = photoCanvasNode.createImageData(uint8ArrayData, width, he
        // 将 ImageData 对象绘制到 canvas 上
```

```
            ctx.putImageData(imageData,0,0,0,0,width,height);
            // 将 canvas 保存为本地图片
            wx.canvasToTempFilePath({
                canvas: photoCanvasNode,
                x: 0,
                y: 0,
                width: width,
                height: height,
                destWidth: width,
                destHeight: height,
                success: (res) => {
                    // 保存照片到本地
                    wx.saveImageToPhotosAlbum({
                        filePath: res.tempFilePath
                    });
                }
            })
        }
    })
```

注意

当小程序切换后台或检测到手机锁屏时，需要调用 stopRecord 停止录像，否则可能出错。



```
Component({
    methods: {
        // 开始录像
        startRecord() {
            this.sdk.startRecord()
        }
        // 结束录像
        async stopRecord() {
            const res = await this.sdk.stopRecord();
            // 保存录像到本地
            wx.saveVideoToPhotosAlbum({
```

```
                    filePath: res.tempFilePath
                })
            }
        }
    })
```

更新日志

最近更新时间：2024-05-08 16:30:10

SDK Version 1.0.21 @ 2023-12-2

优化了性能问题。
修复了一些其他问题。

SDK Version 1.0.19 @ 2023-11-20

支持设置背景分割精确度 level。
优化了性能问题。
修复了一些其他问题。

SDK Version 1.0.16 @ 2023-10-24

支持小程序推流组件接入美颜特效 SDK。
优化了性能问题。
修复了一些其他问题。

SDK Version 1.0.11 @ 2023-07-17

新增发际线、清晰等多种美颜参数。
优化了性能问题。
修复了一些其他问题。

SDK Version 1.0.0 @ 2023-02-27

推出商业化版本

SDK Version 0.3.0 @ 2022-10-14

新增 3D特效、支持 Web 和小程序
新增 Animoji 表情和虚拟形象，支持 Web 平台
修复了内存泄漏问题
修复了其他问题

SDK Version 0.2.5 @ 2022-08-01

修复 Android 端微信浏览器闪退的问题
修复 disable 背景不消失的 bug
修复了其他问题

SDK Version 0.2.3 @ 2022-07-20

修复了切换输入流自适应失败的问题
支持自动检测启停人脸检测
修复了其他问题

SDK Version 0.2.0 @ 2022-06-29

Web端支持人像分割
优化了接入体验

SDK Version 0.1.18 @ 2022-06-17

优化内置相机配置，完善了摄像头的管理能力，新增本地播放器
新增了延迟初始化配置
优化了滤镜强度设置以及叠加效果

SDK Version 0.1.12 @ 2022-05-11

SDK 性能优化
新增 FPS 属性，支持设置渲染帧率

SDK Version 0.1.9 @ 2022-04-28

新增内置贴纸
修复移动 Web 切后台黑屏问题

SDK Version 0.1.1 @ 2022-04-13

正式发布
支持 Web 和小程序

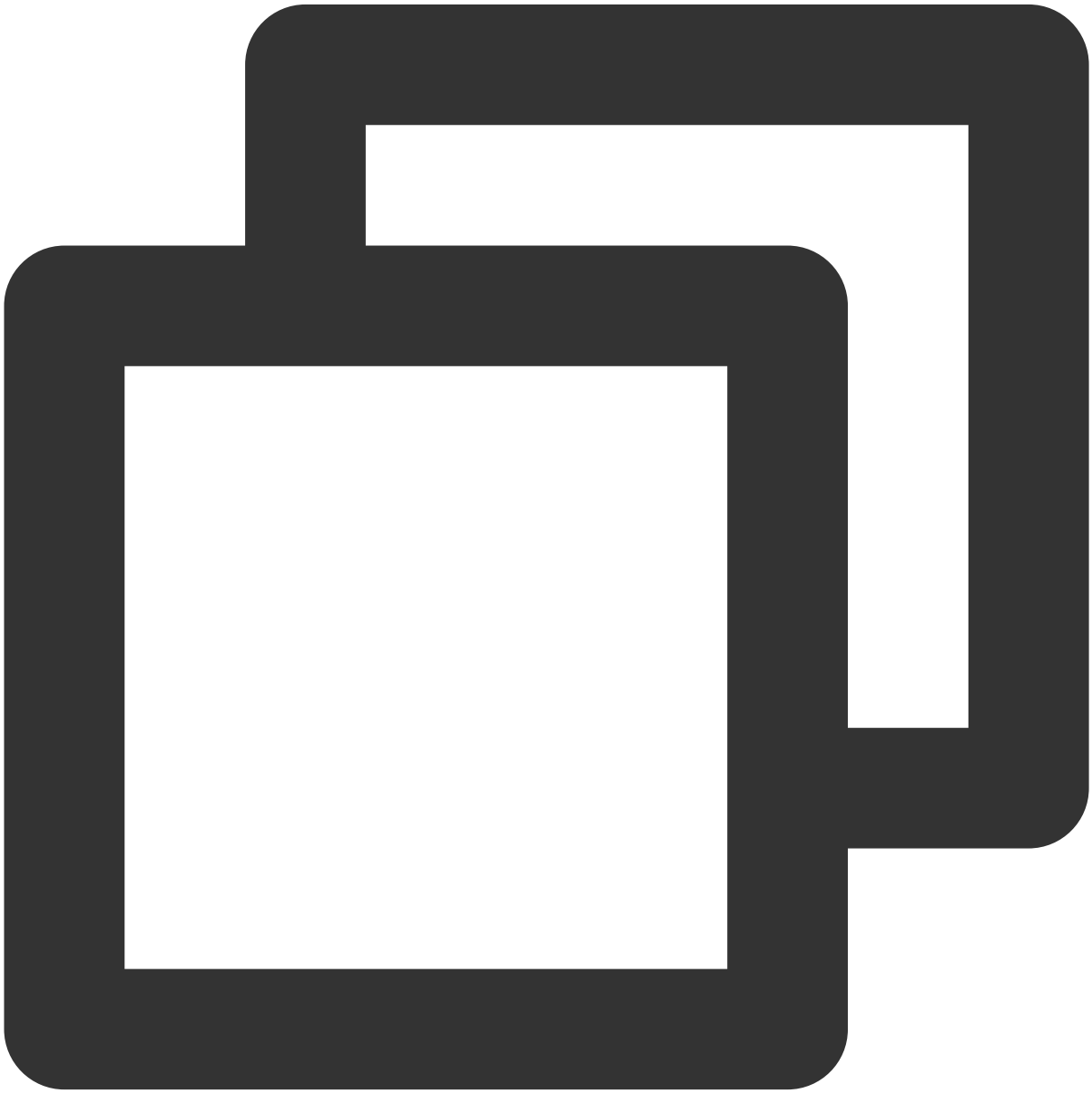
接口错误码

最近更新时间：2023-04-11 16:34:12

SDK 内置接口错误码

如： `https://webar.qcloud.com/sdk/xxx` 等接口的错误码。

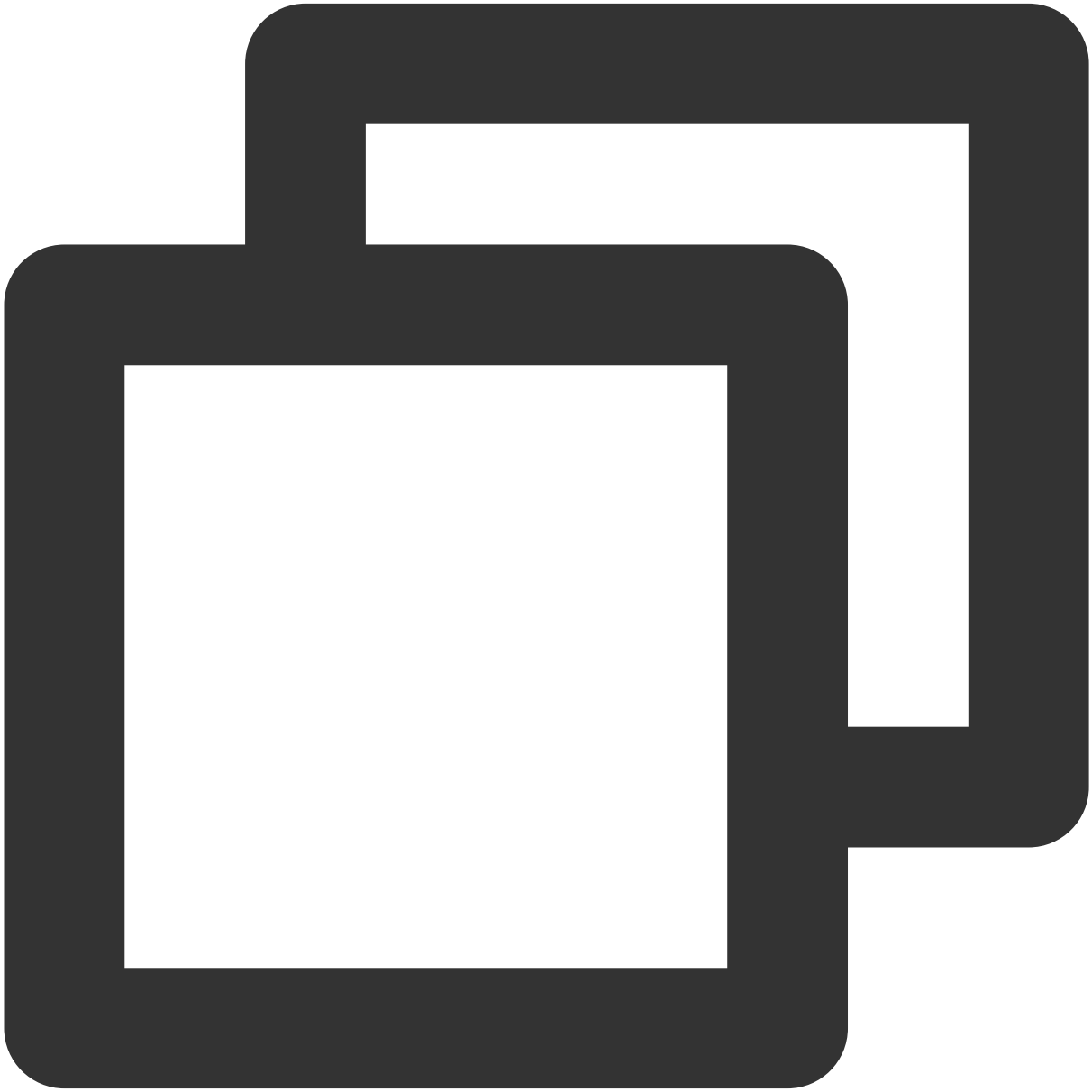
Response 返回结构：



```
{
    "Code": xxx,
    "Message": "xxx"
}
```

成功

Code 为 0 表示成功，如果有数据，Data 会有相应数据：

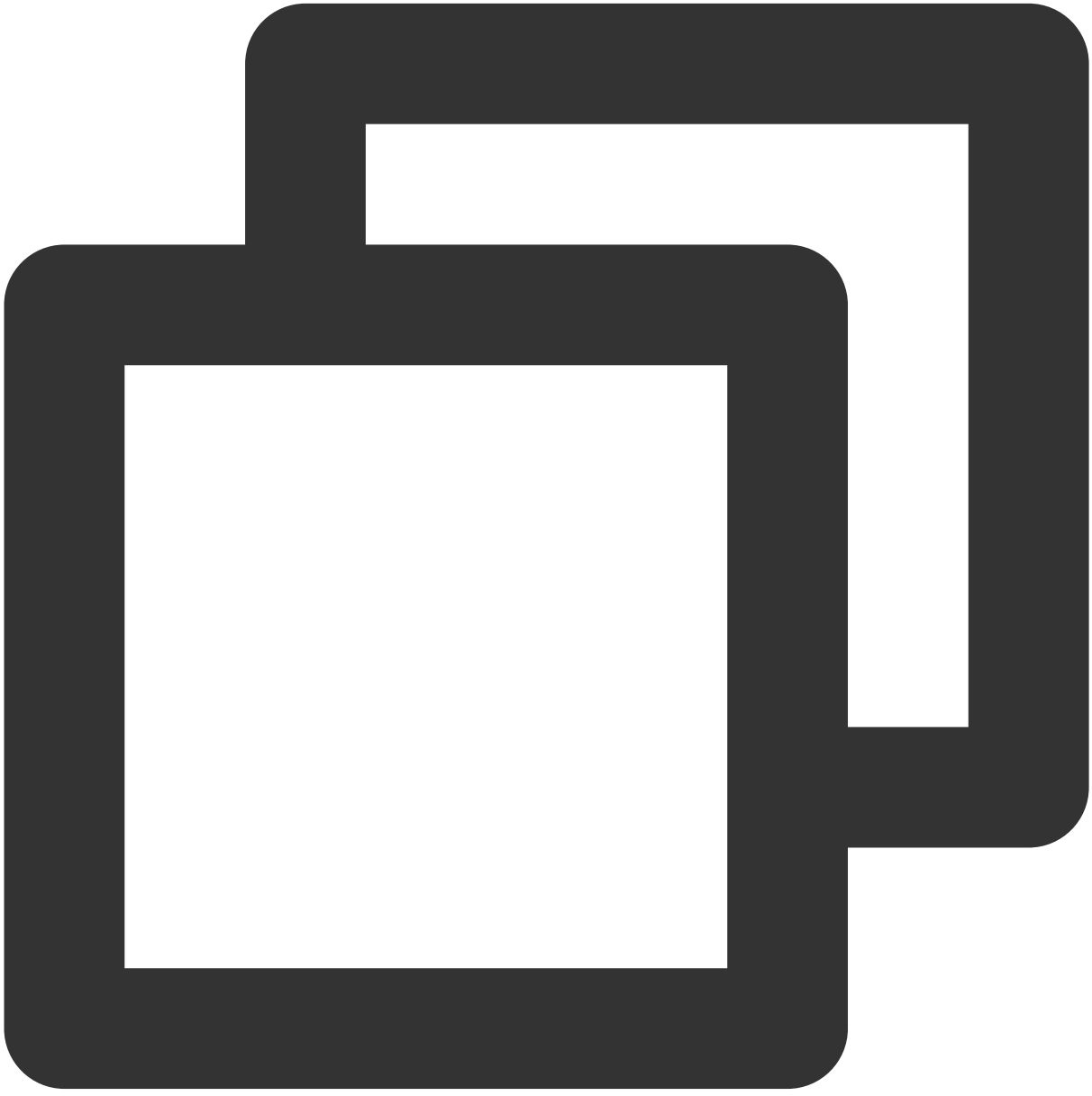


```
{
```

```
    Code: 0,
    Data:{...}
}
```

鉴权错误

Code在 100 -- 104 是鉴权错误，response Status 为 401：

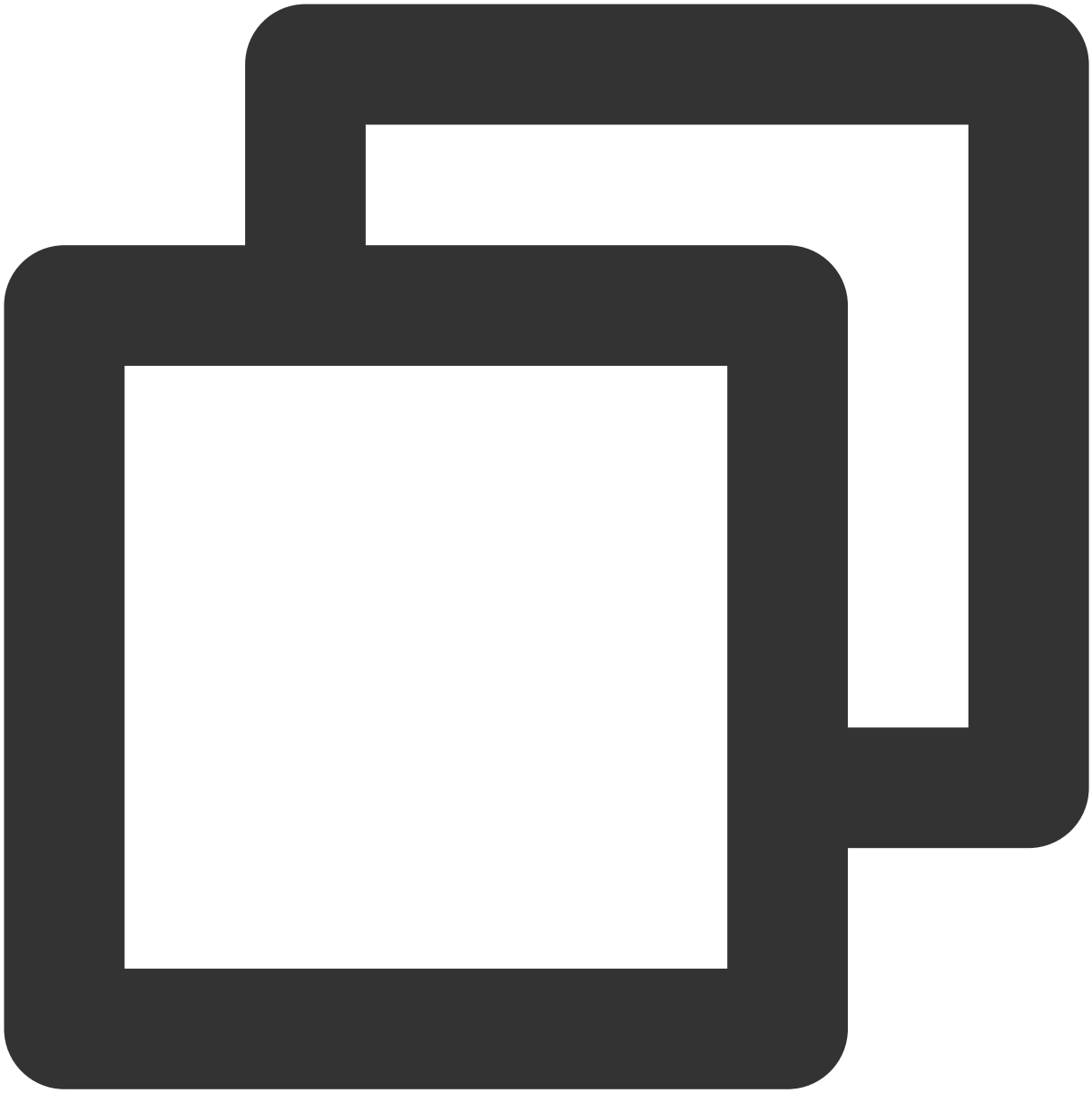


```
{
  "100":{
    zh: "缺少鉴权参数"
```

```
    },
    "101":{
      zh: "signature 超时"
    },
    "102":{
      zh: "未找到此用户"
    },
    "103":{
      zh: "signature 错误"
    },
    "104":{
      zh: "referer 或者 WeChatAppId 不匹配"
    }
  }
```

入参错误

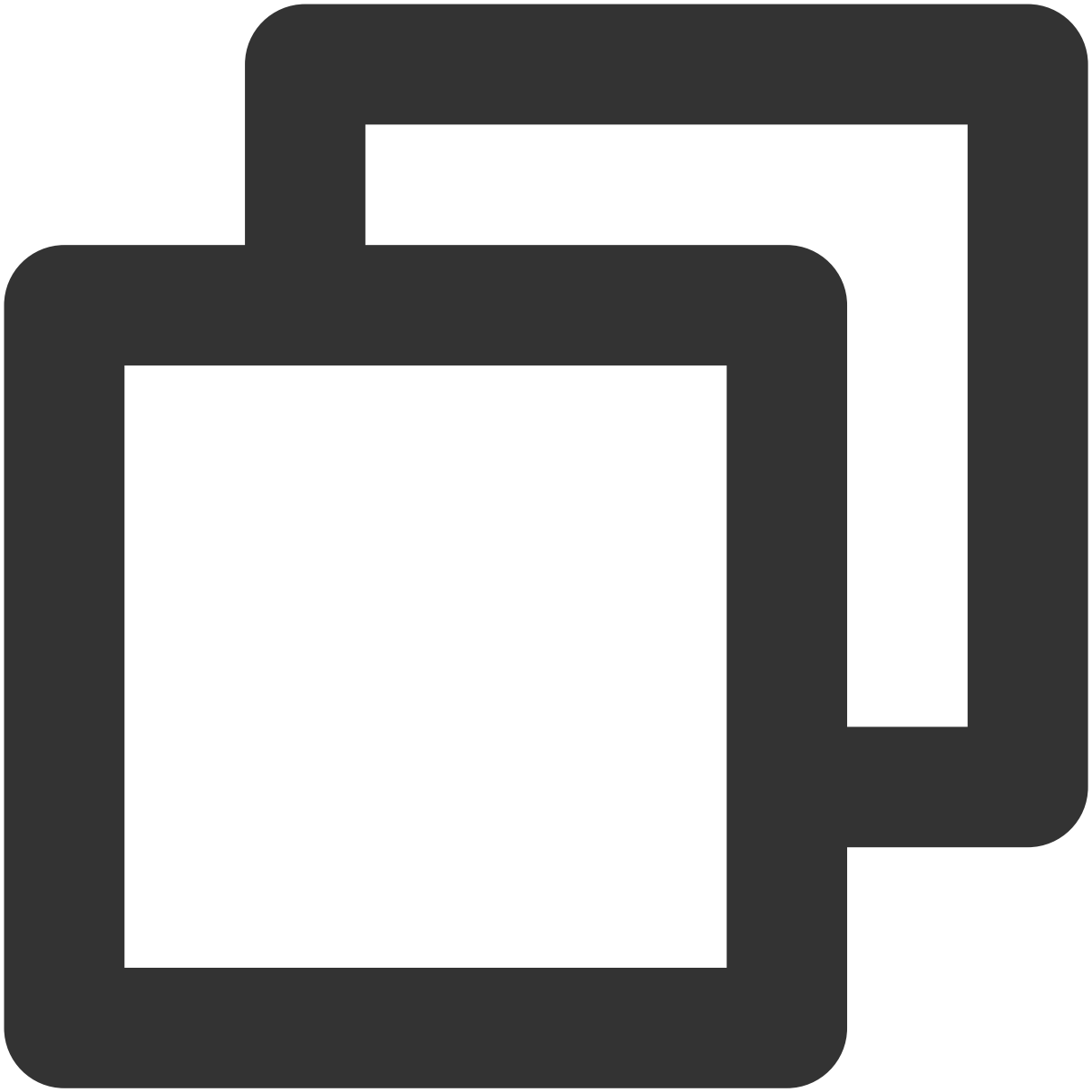
Code = -2 是入参校验错误，Message 会有相应的说明：



```
{
    "Code": -2,
    "Message": "LicenseKey字段必须是字符串"
}
```

业务校验错误

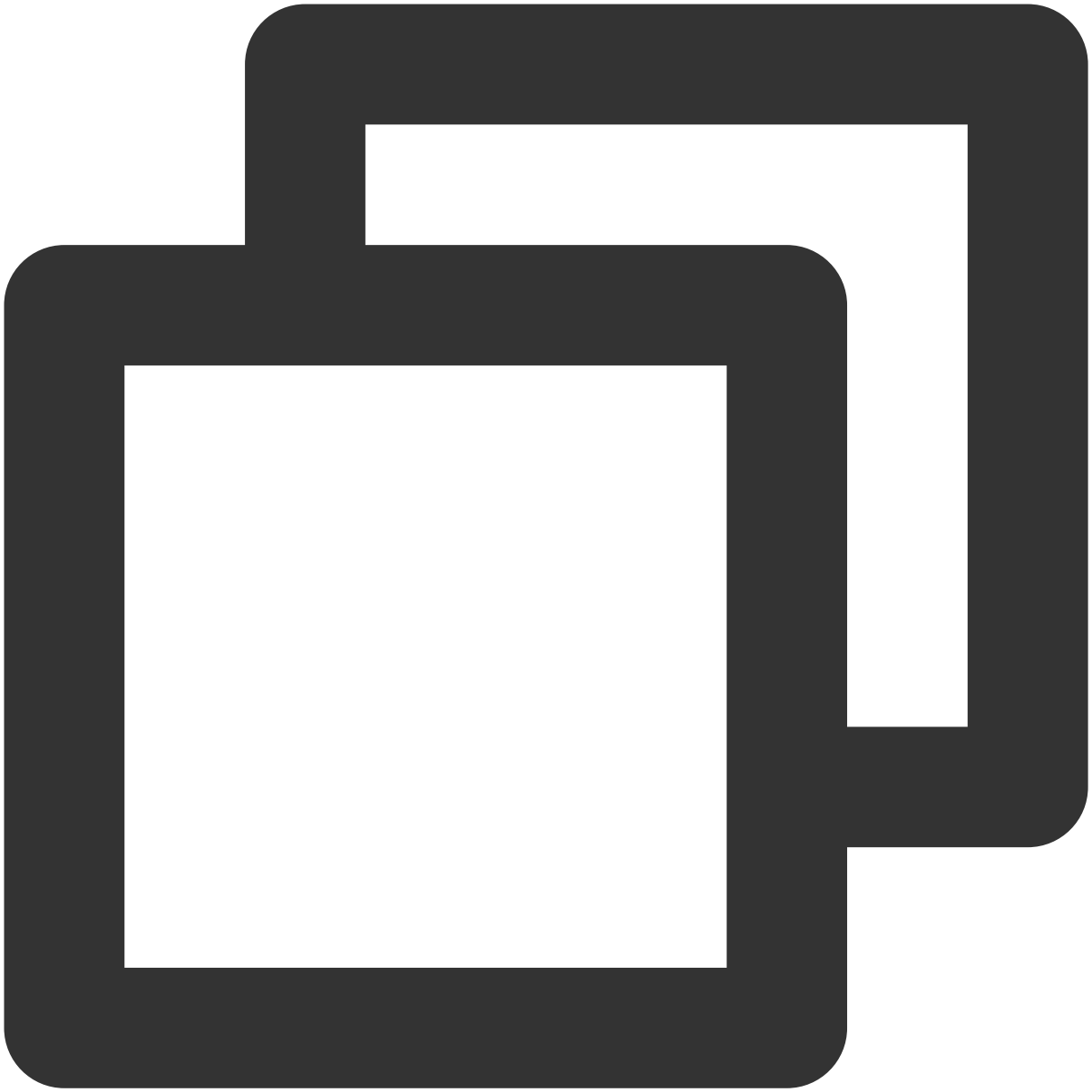
Code > 1000 的是业务校验错误，Message 会有相应说明：



```
{
    "Code": 2007,
    "Message": "此项目不存在"
}
```

未知错误

Code = -1 是未知错误：



```
{
    "Code": -1,
    "Message": "未知错误"
}
```

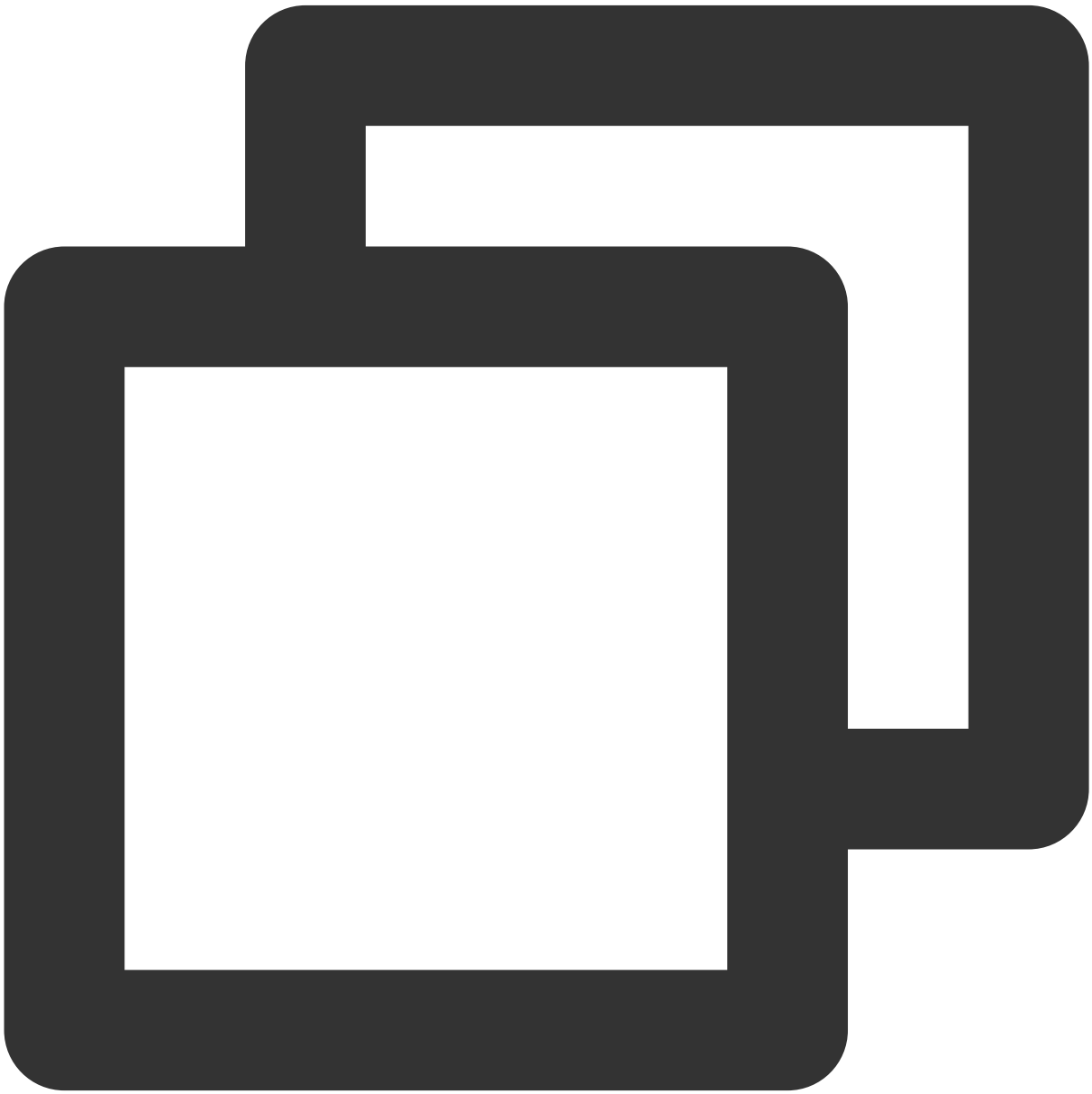
API 文档

最近更新时间：2024-05-08 18:31:50

本文档将介绍 Web 美颜特效 SDK 核心参数及方法。

说明：

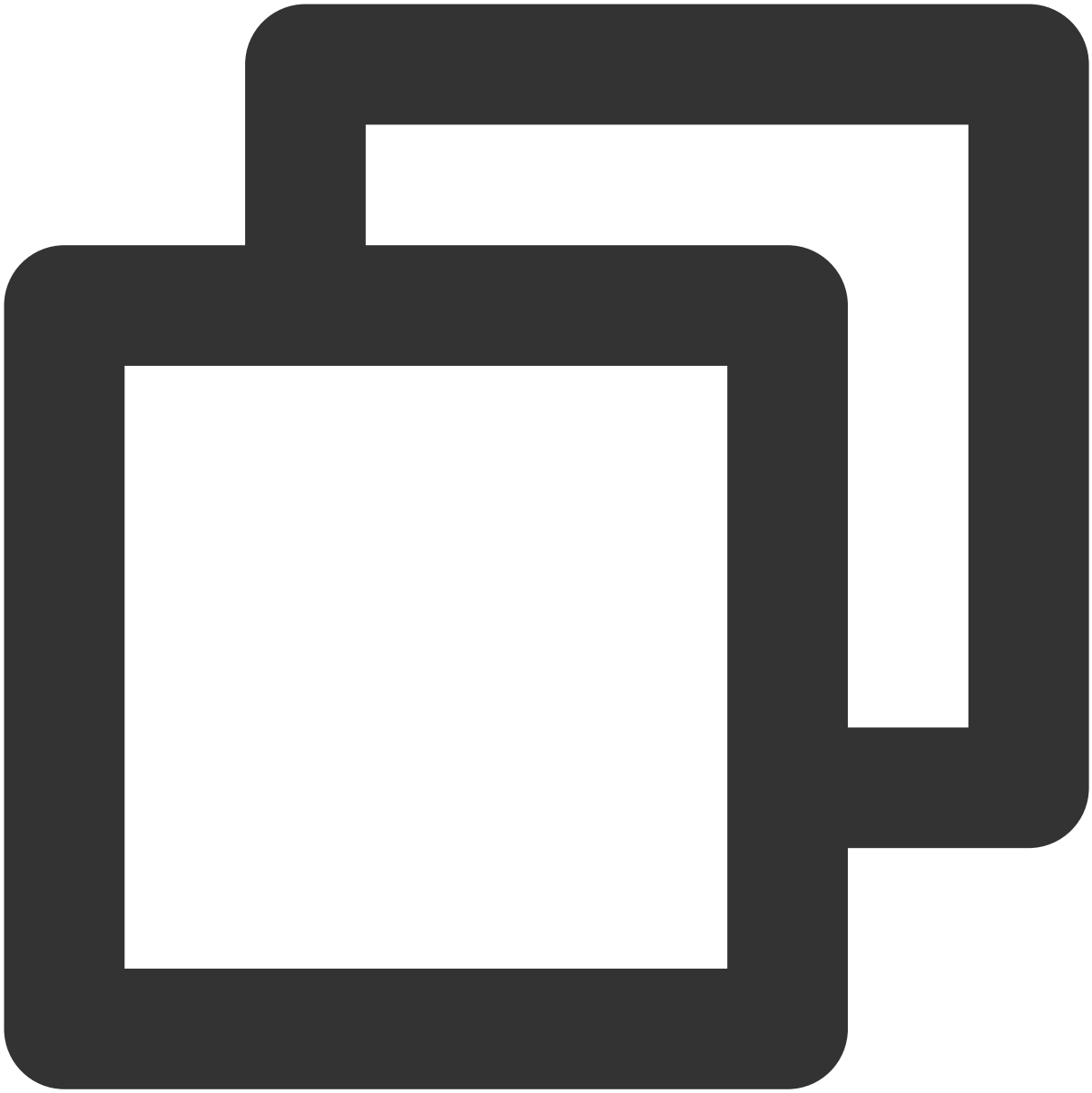
Web 美颜特效 SDK 需要浏览器支持并开启硬件加速才能够流畅渲染（小程序端无需判断），因此 SDK 提供了检测方法供业务提前判断，如不支持则进行屏蔽处理。



```
import {ArSdk, isWebGLSupported} from 'tencentcloud-webar'
```

```
if(isWebGLSupported()) {
    const sdk = new ArSdk({
    ...
})
} else {
    // 屏蔽逻辑
}
```

初始化参数

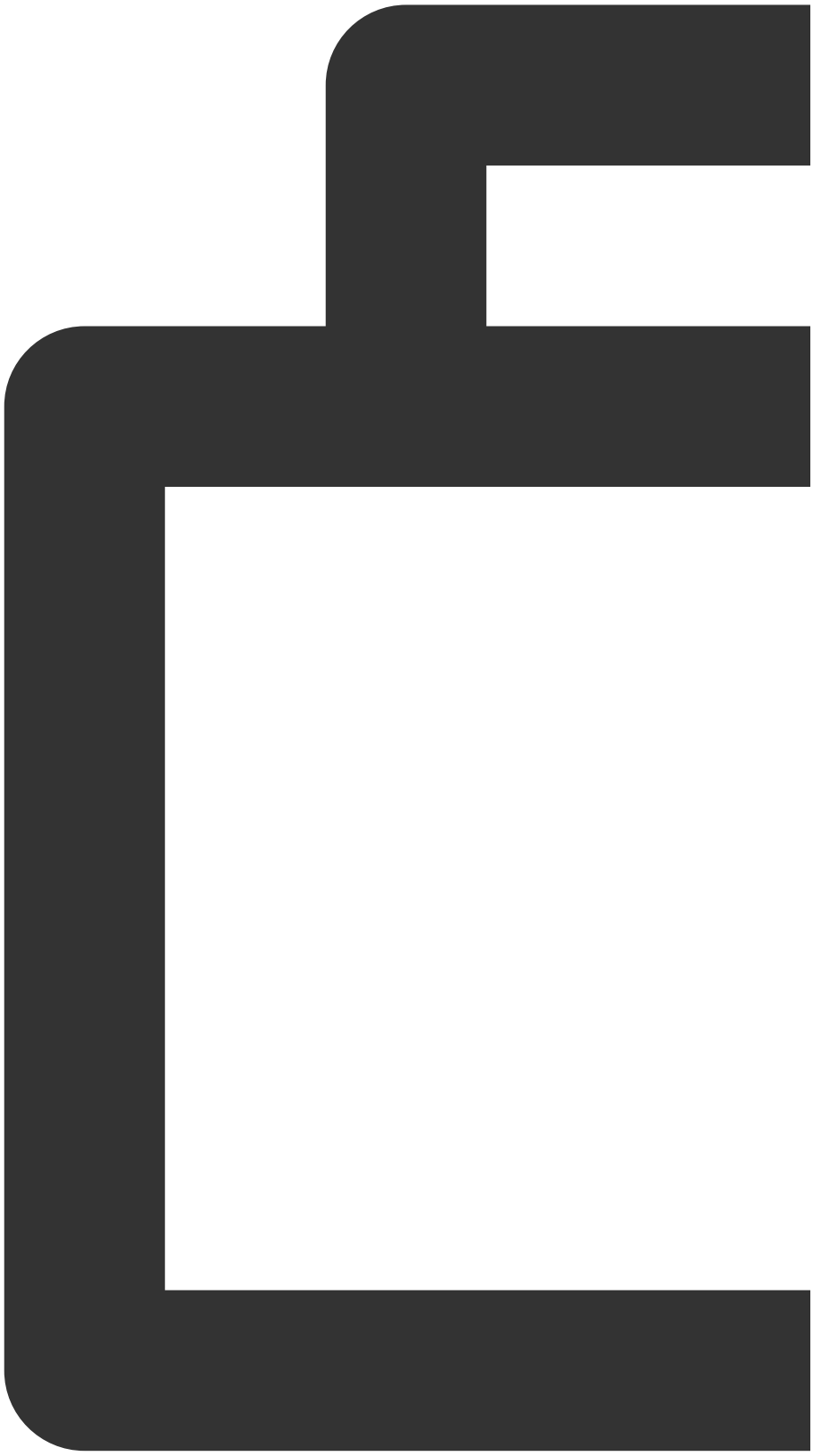


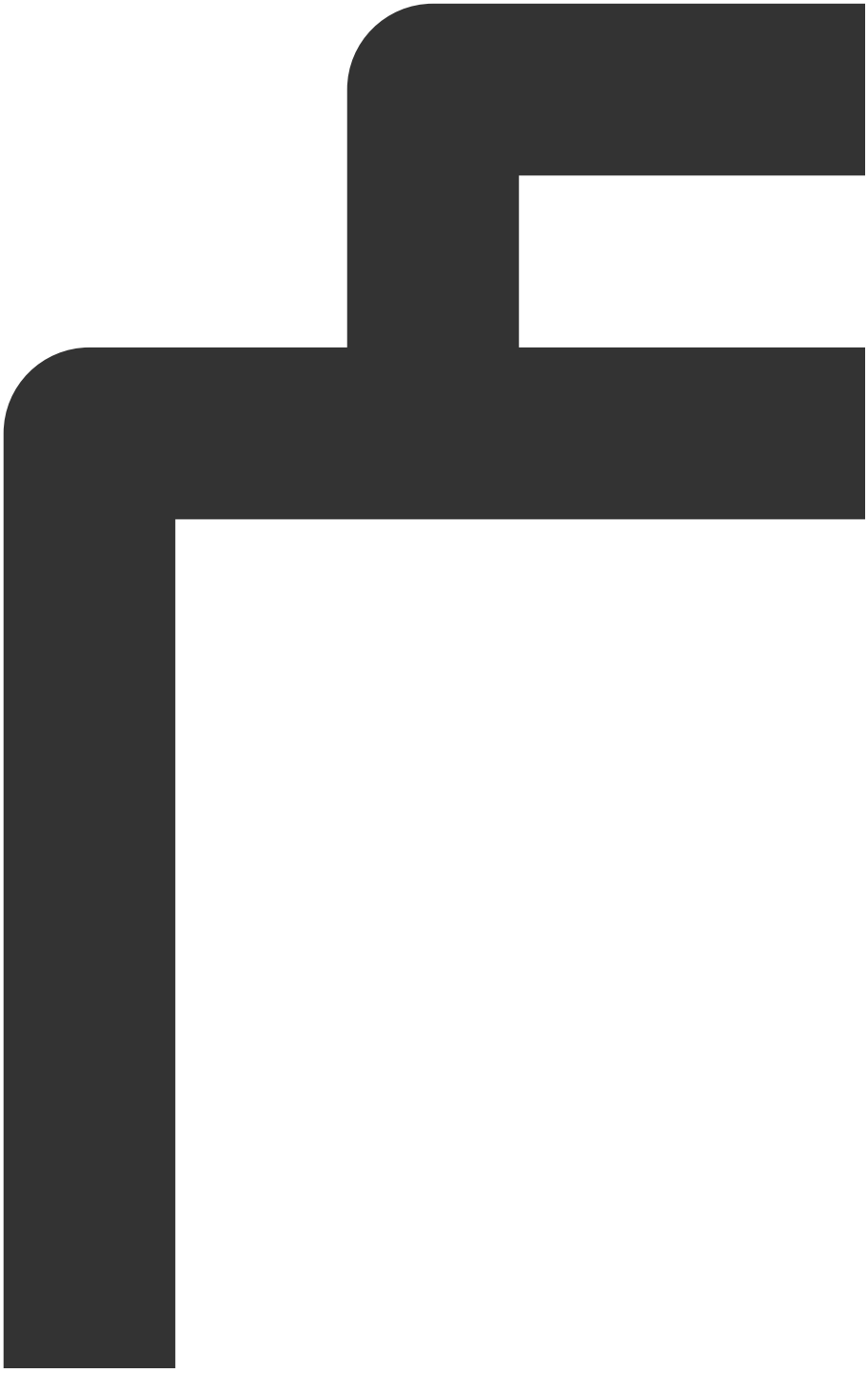
```
import { ArSdk } from 'tencentcloud-webar'
// 初始化SDK
const sdk = new ArSdk({
...
})
```

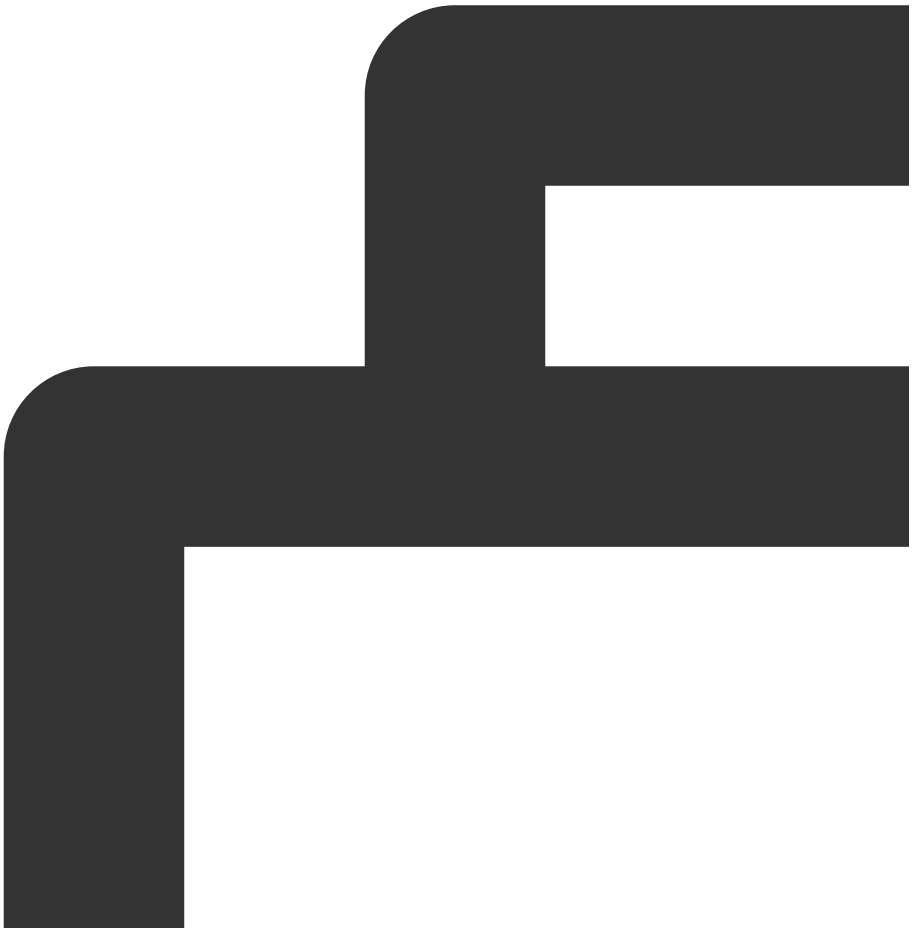
初始化 SDK 的 Config 支持以下参数：

参数	说明	类型
module	模块配	

置

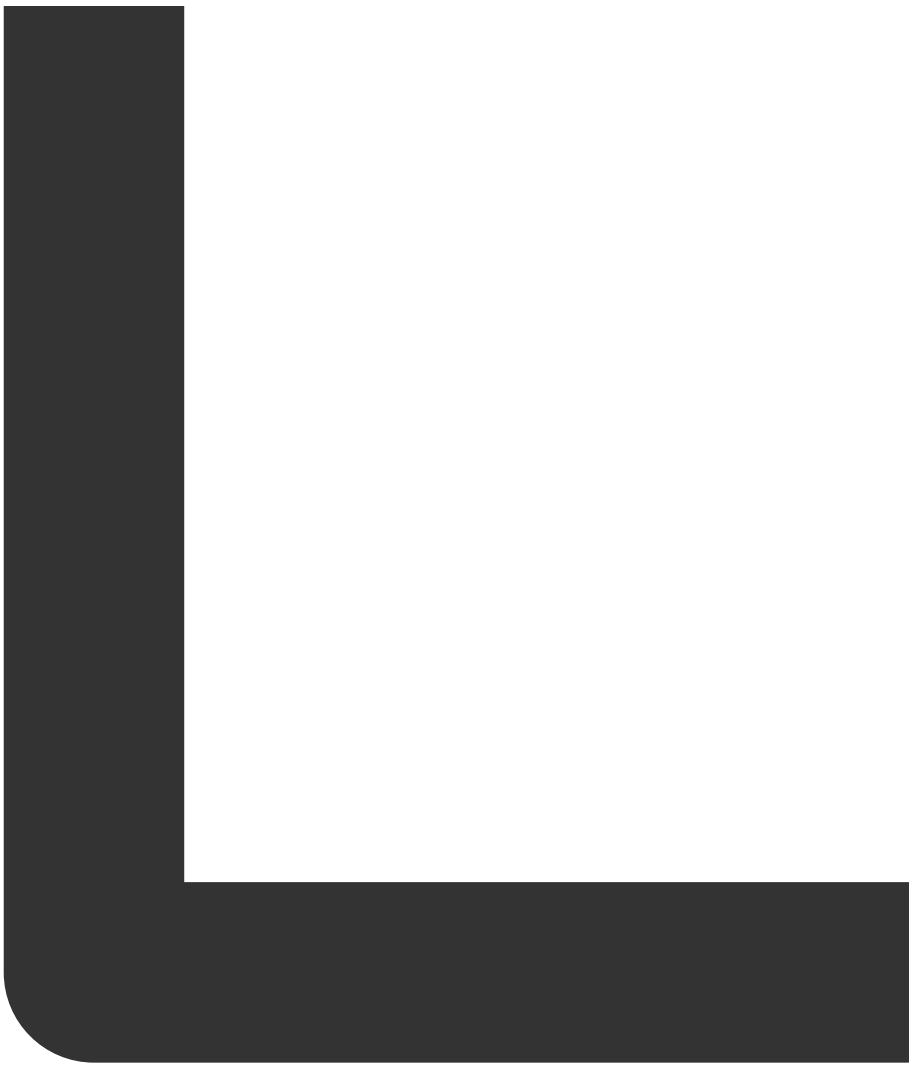


		`type SegmentationLevel = 0 | 1 | 2 // 1.0.19 之后的版本支持传参选` `type ModuleConfig = {` `  beautify: boolean // 是否开启美颜，默认为true` `  segmentation: boolean // 是否开启背景分割，默认为false` `  segmentationLevel: SegmentationLevel // 背景分割精确度等级，默认` `}`
auth	鉴权参数	

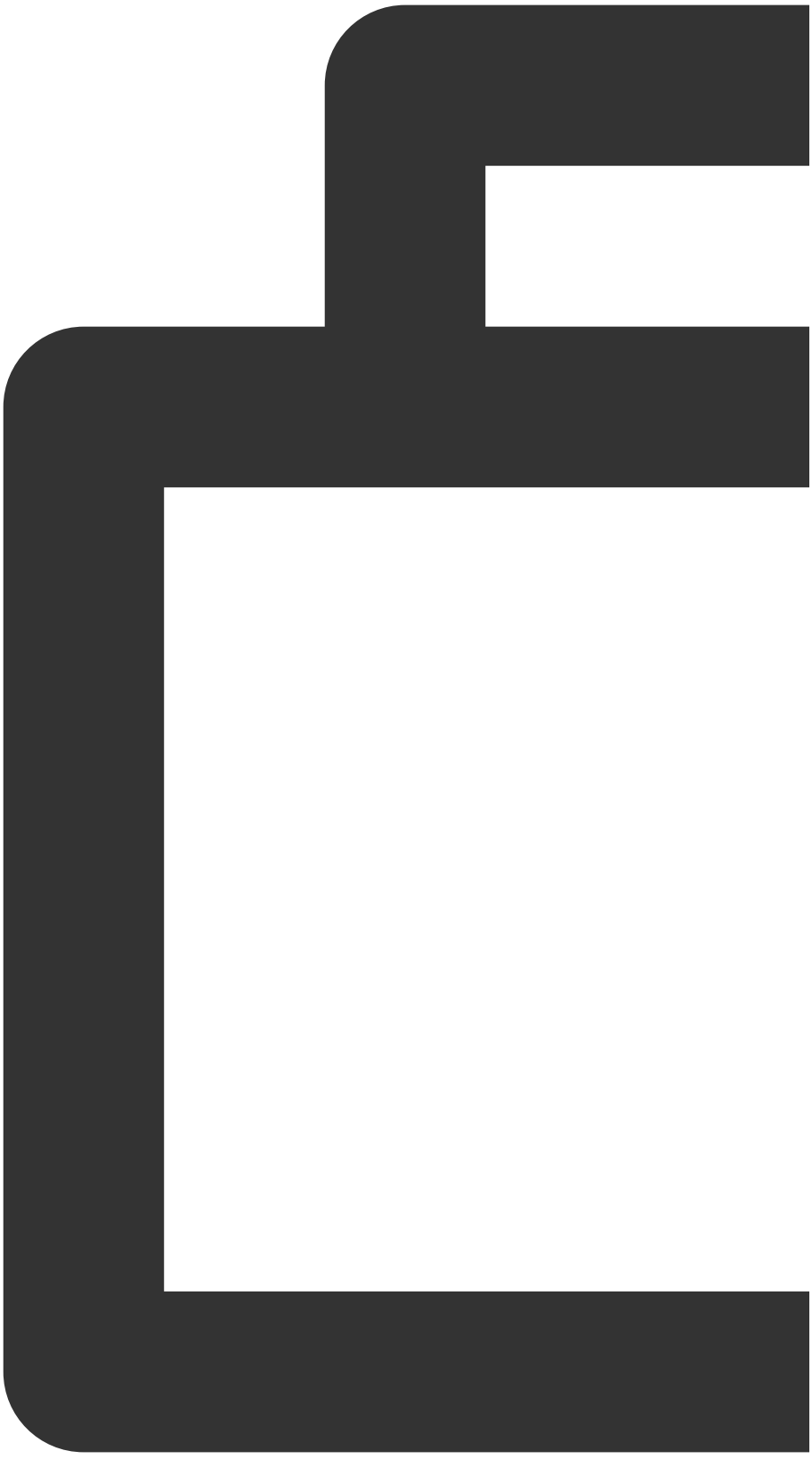
		 `type AuthConfig = {` `licenseKey: string // 控制台 Web License 管理 获取` `appId: string // 控制台 账号信息 > 基本信息 查看 APPID` `authFunc:() => Promise<{` `signature:string,` `timestamp:string` `}> // 请参见 License 配置使用` `}`
input	输入源	MediaStream|HTMLImageElement|String
camera	内置相机	

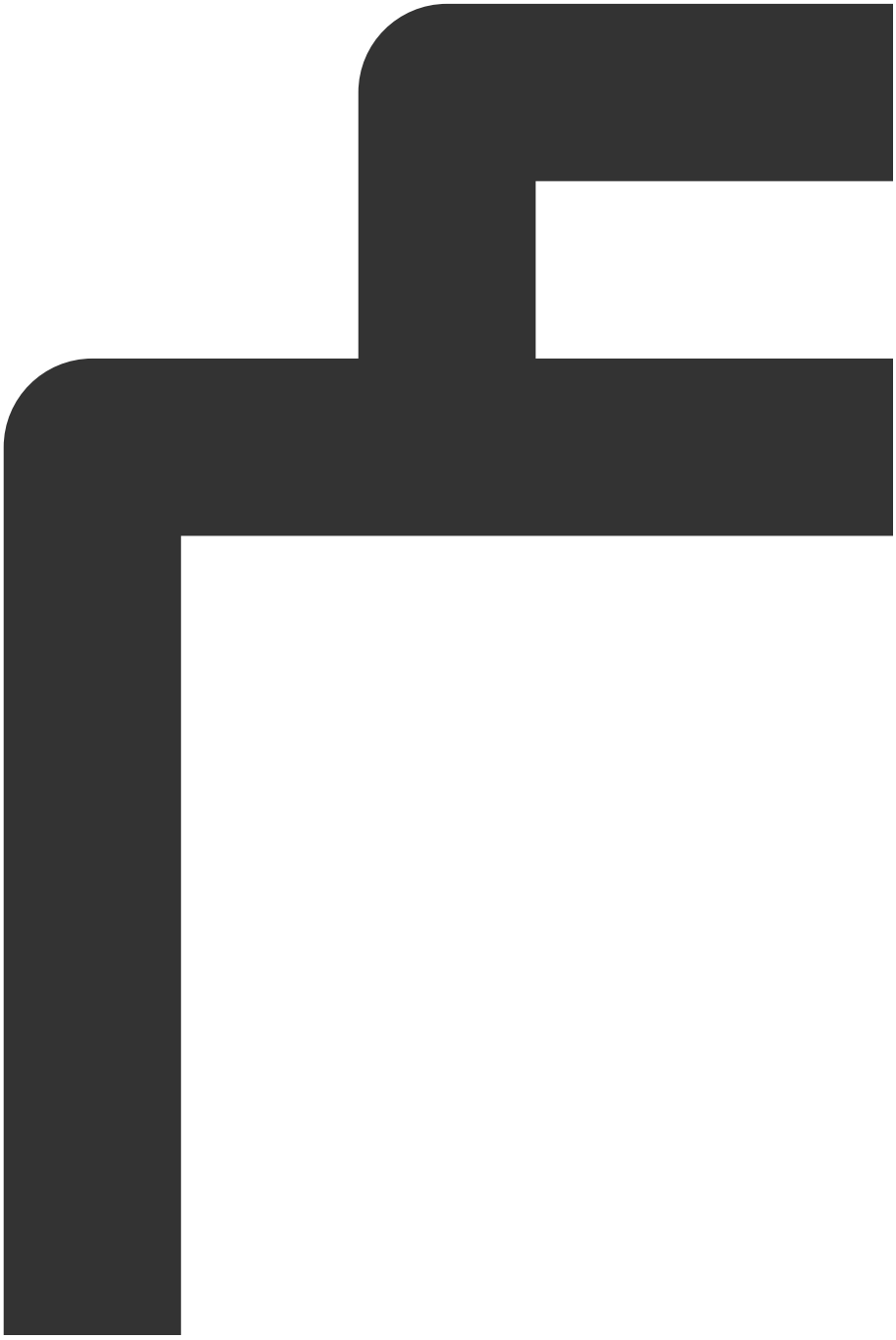
```
type CameraConfig = {
    width: number, // 拍摄画面宽度
    height: number, // 拍摄画面高度
    mirror: boolean, // 是否开启左右镜像
    frameRate: number // 画面采集帧率
}
```

beautify	美颜参数	



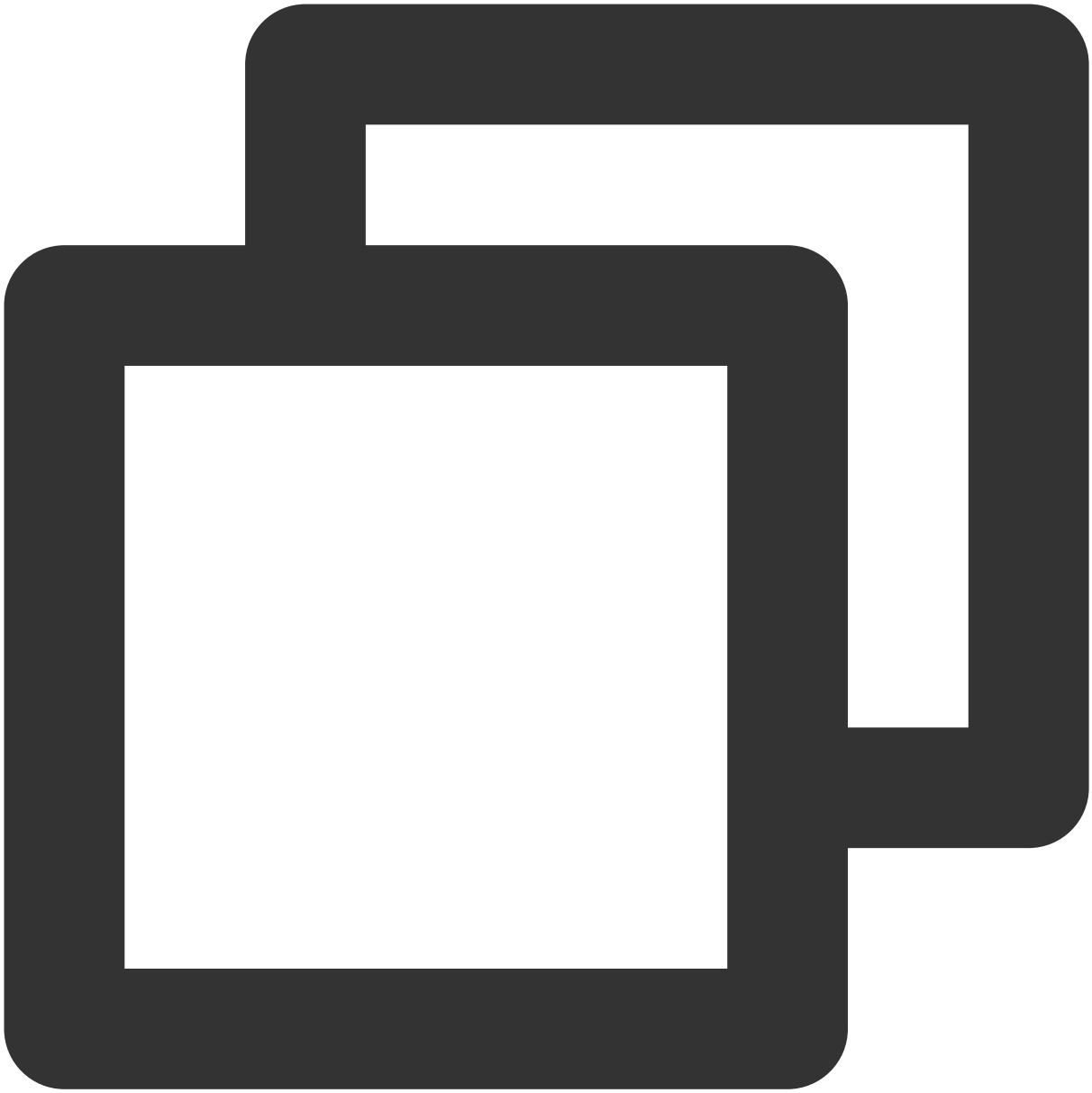
```
type BeautifyOptions = {
    whiten?: number; // 美白 0-1
    dermabrasion?: number; // 磨皮0-1
    lift?: number; // 窄脸0-1
    shave?: number; // 削脸0-1
    eye?: number; // 大眼0-1
    chin?: number; // 下巴0-1
    // 注意：以下参数仅在1.0.11及以上版本可用
    darkCircle?: number; // 黑眼圈0-1
    nasolabialFolds?: number; // 法令纹0-1
    cheekbone?: number; // 颧骨0-1
    head?: number; // 小头0-1
    eyeBrightness?: number; // 亮眼0-1
    lip?: number; //嘴唇 -1 - 1
    forehead?: number; //发际线 0-1
    nose?: number; // 鼻子 -1 - 1
```

		usm?: number; // 清晰 0-1 }
background	背景参数	

		`` type BackgroundOptions = { type: 'image' | 'blur' | 'transparent', src?: string } ```
loading	内置 loading icon 配置	

		 ``` type loadingConfig = { enable: boolean, size?: number lineWidth?: number strokeColor?: number } ```
language	国际化对应（1.0.6 版本开始支持），中文（zh）和英文（en）	String: zh | en

回调事件



```
let effectList = [];
let filterList = [];
// sdk 的回调用法
sdk.on('created', () => {
    // 在 created 回调中拉取特效和滤镜列表供页面展示
    sdk.getEffectList({
        Type: 'Preset',
        Label: '美妆',
    }).then(res => {
        effectList = res
    });
```

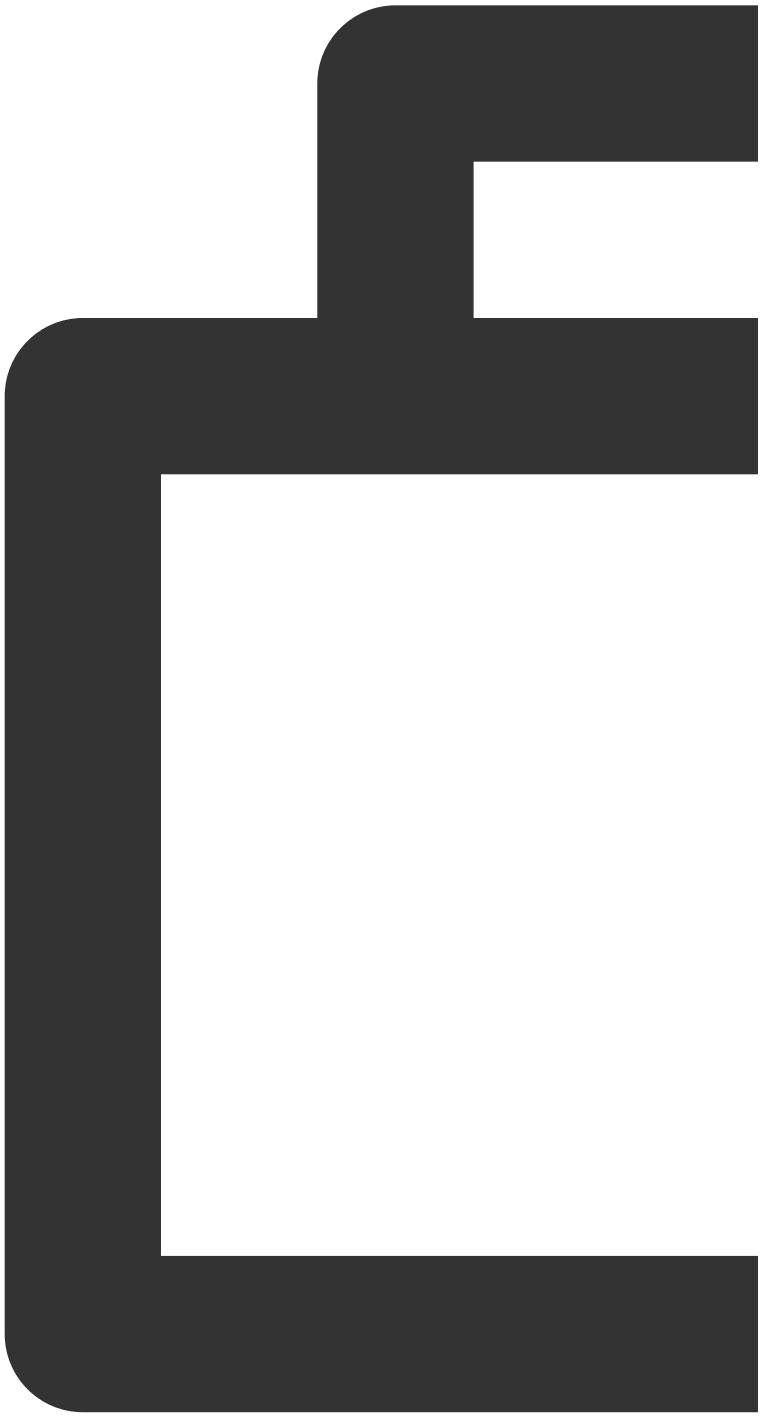
```
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})
sdk.on('cameraReady', async () => {
    // 在 cameraReady 回调中可以更早地获取输出画面，此时初始化传入的美颜参数还未生效
    // 适用于需要尽早地展示画面，但不要求画面一展示就有美颜的场景
    // 后续美颜生效后无需更新stream，SDK内部已处理
    const arStream = await ar.getOutput();
    // 本地播放
    // localVideo.srcObject = arStream

})
sdk.on('ready', () => {
    // 在 ready 回调中获取输出画面，此时初始化传入的美颜参数已生效
    // 区别上述cameraReady中获取output，适用于画面一展示就要有美颜的场景，但不要求尽早地展示画面
    // 根据自身业务需求选择一种处理方式即可
    const arStream = await ar.getOutput();
    // 本地播放
    // localVideo.srcObject = arStream

    // 在 ready 回调中调用 setBeautify/setEffect/setFilter 等渲染方法
    sdk.setBeautify({
        whiten: 0.3
    });
    sdk.setEffect({
        id: effectList[0].EffectId,
        intensity: 0.7
    });
    sdk.setEffect({
        id: effectList[0].EffectId,
        intensity: 0.7,
        filterIntensity: 0.5 // 0.1.18及以上版本支持单独设置effect中滤镜的强度，不传则默认与
    });
    sdk.setFilter(filterList[0].EffectId, 0.5)
})
```

事件	说明	回调参数
created	SDK 鉴权完成并成功创建实例时触发	-
cameraReady	SDK 的画面生成时触发，此时 output 已有画面但美颜仍无法生效	-
ready	SDK 内部检测初始化完成时触发，此时 output 画面已有美颜，也可以设置新的特效	-

error	SDK 发生错误时触发	error 对象

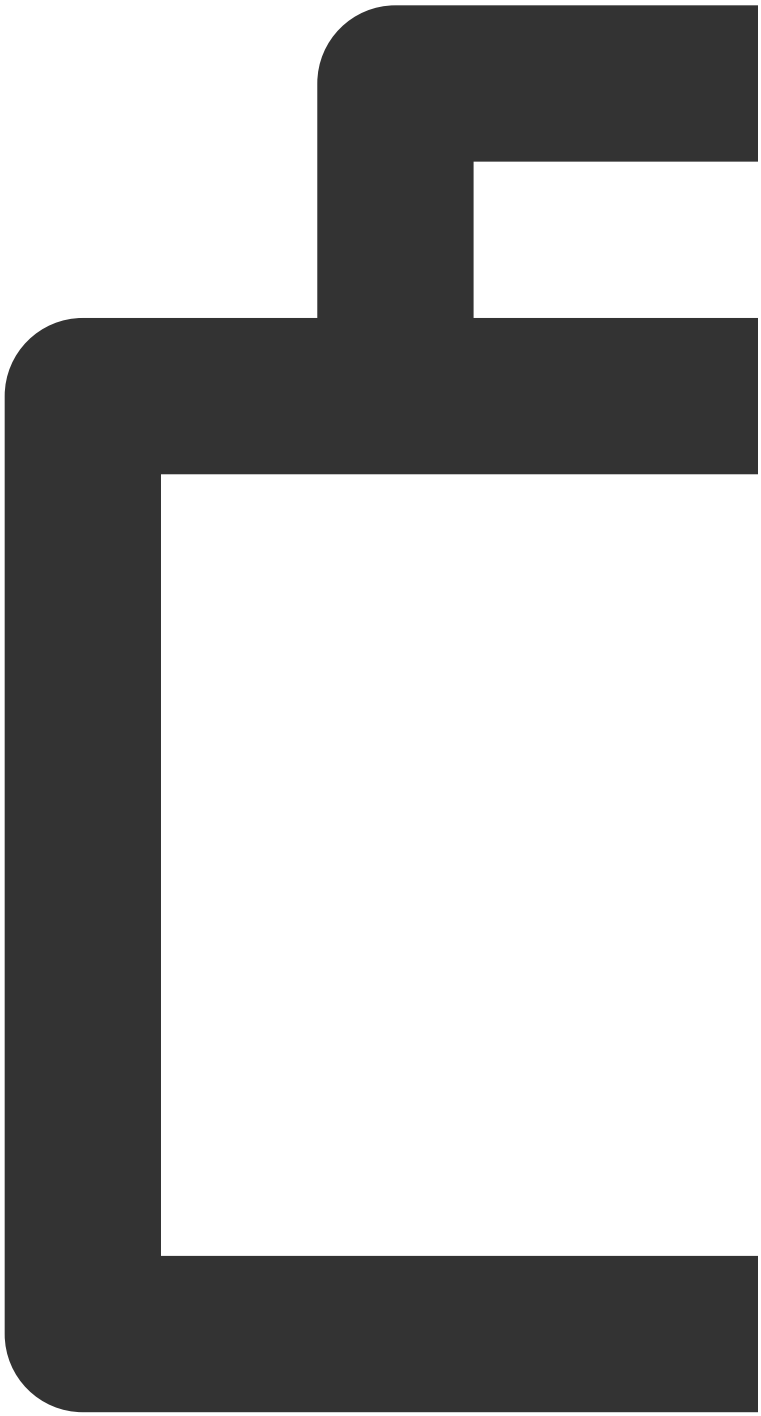
对象方法

接口	参数
async initCore()	



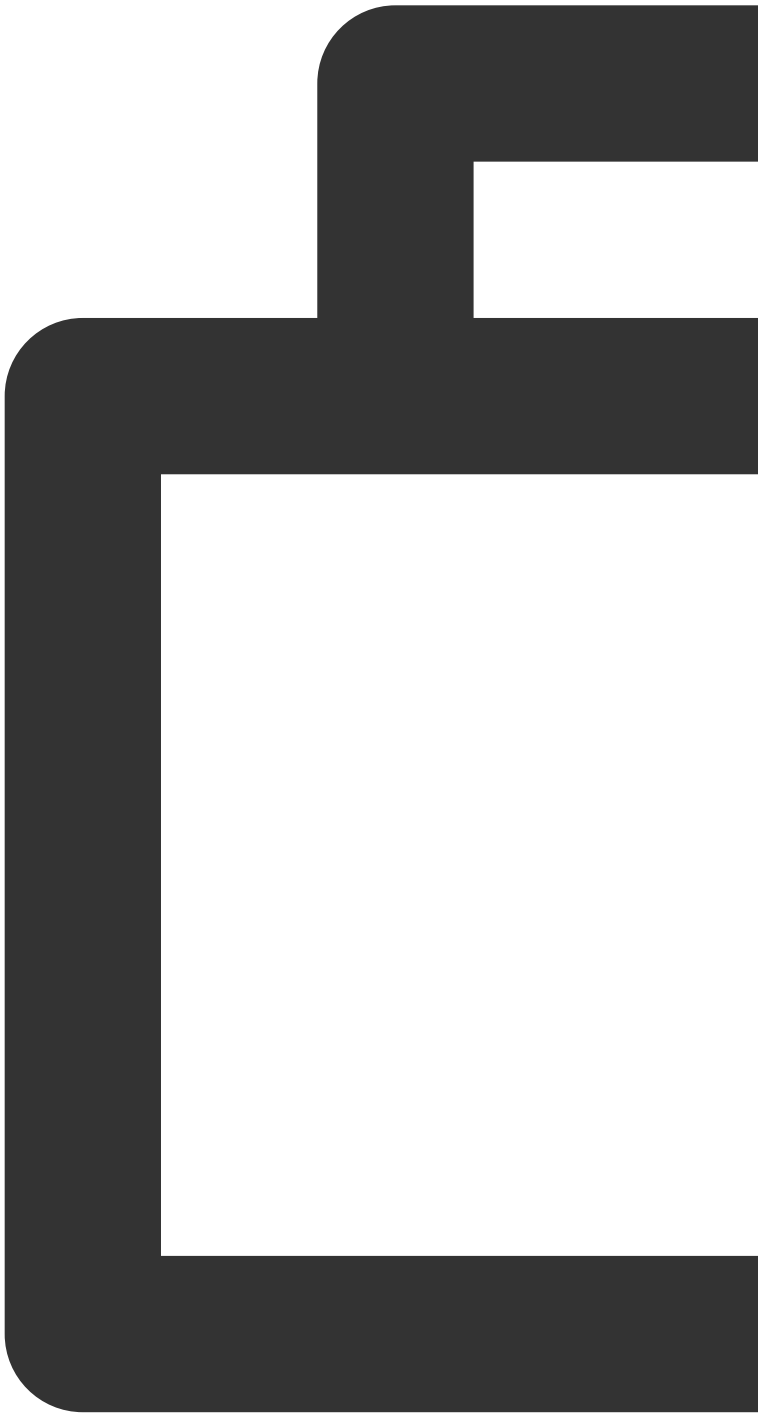
```
{
  input?:MediaStream|HTMLImageElement|String; // 输
  camera?:CameraConfig; // 摄像头模式
  mirror?:boolean;  // 是否镜像
```

	} }
async getOutput(fps)	fps：设置输出的媒体流帧率，默认无须设置
setBeautify(options)	

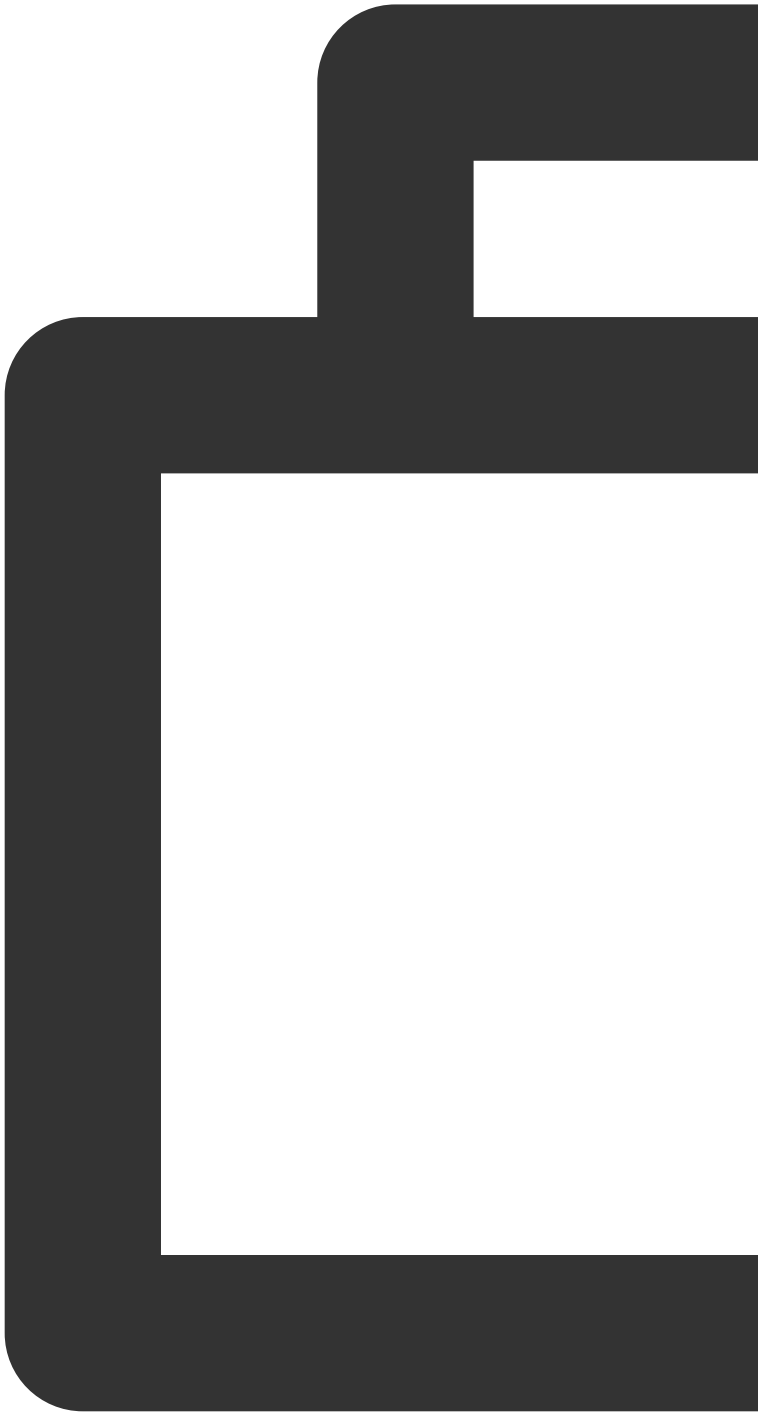


```
type BeautifyOptions = {
    whiten?: number; // 美白 0-1
    dermabrasion?: number; // 磨皮0-1
    lift?: number; // 窄脸0-1
    shave?: number; // 削脸0-1
```

	eye?: number; // 大眼0-1 chin?: number; // 下巴0-1 // 注意：以下参数仅在1.0.11及以上版本可用 darkCircle?: number; // 黑眼圈0-1 nasolabialFolds?: number; // 法令纹0-1 cheekbone?: number; // 颧骨0-1 head?: number; // 小头0-1 eyeBrightness?: number; // 亮眼0-1 lip?: number; //嘴唇 -1 - 1 forehead?: number; //发际线 0-1 nose?: number; // 鼻子 -1 - 1 usm?: number; // 清晰 0-1 }
setEffect(effects, callback)	effects：特效 ID | effect 对象 | 特效 ID / effect 数组

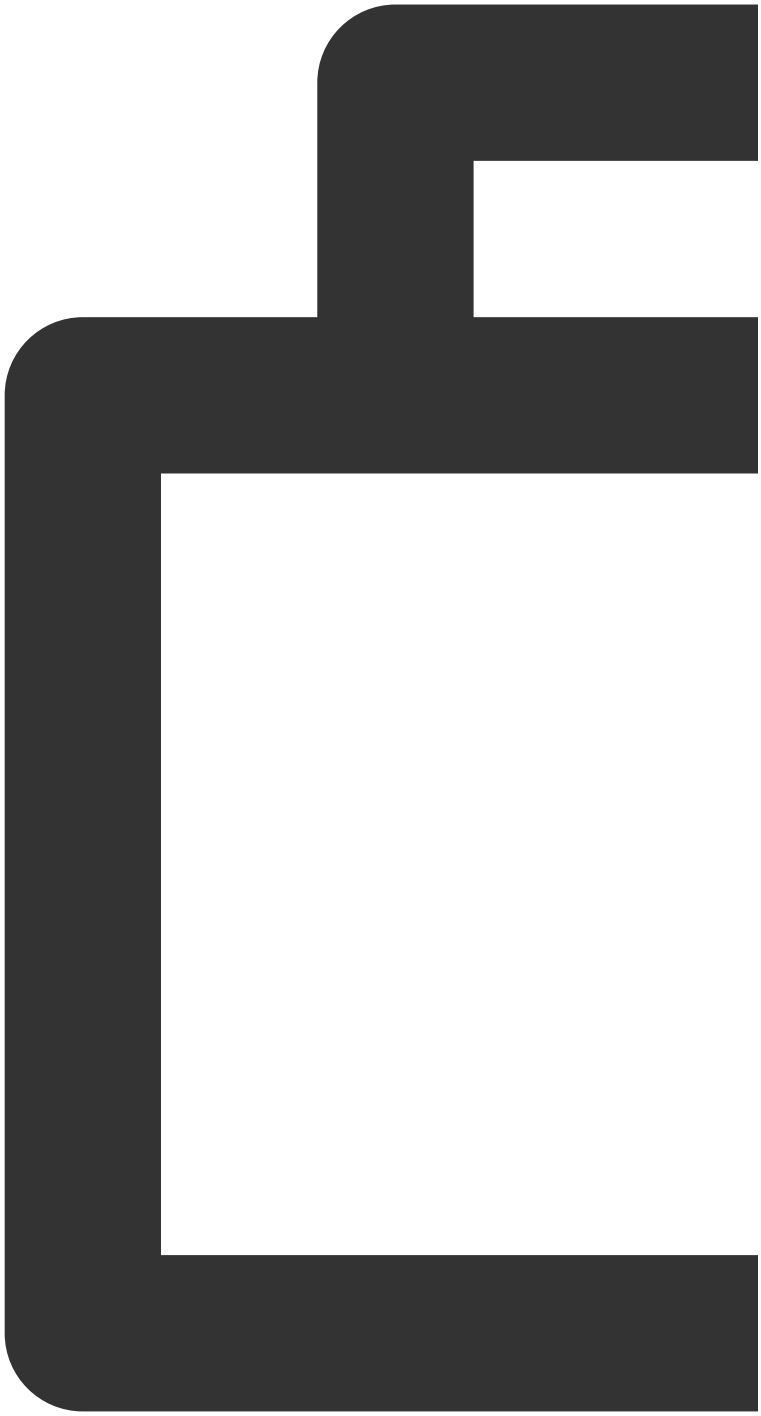


```
effect:{
    id: string,
    intensity: number, // 特效强度，默认1，范围0-1
    filterIntensity: number // 单独控制特效中的滤镜强度
}
```

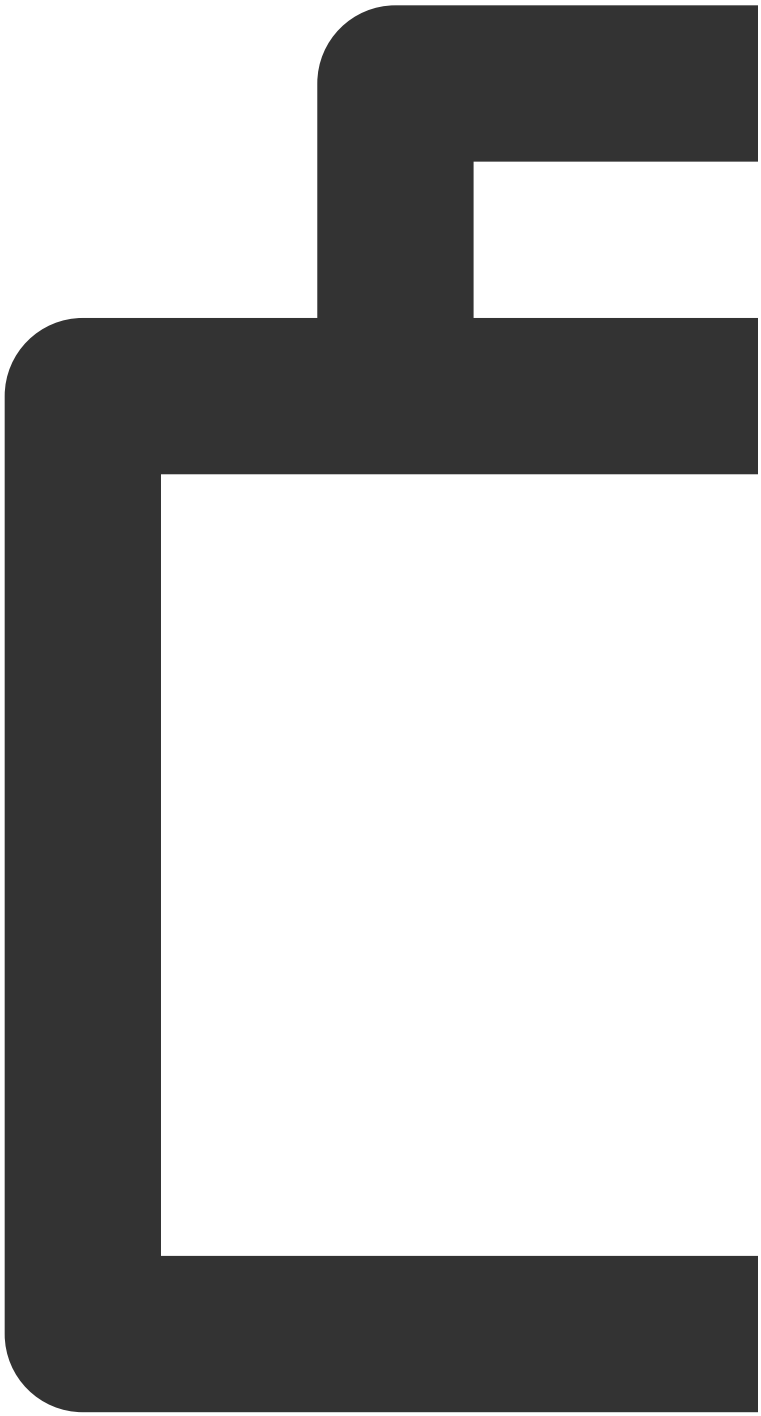
	callback：设置成功的回调
setAvatar(params)	 { mode: 'AR' | 'VR',

```
    effectId?: string, // 透传effectId使用内置模型
    url?: string, // 透传url使用自定义模型
    backgroundUrl?: string, // 背景图片链接，仅在VR模式
}
```

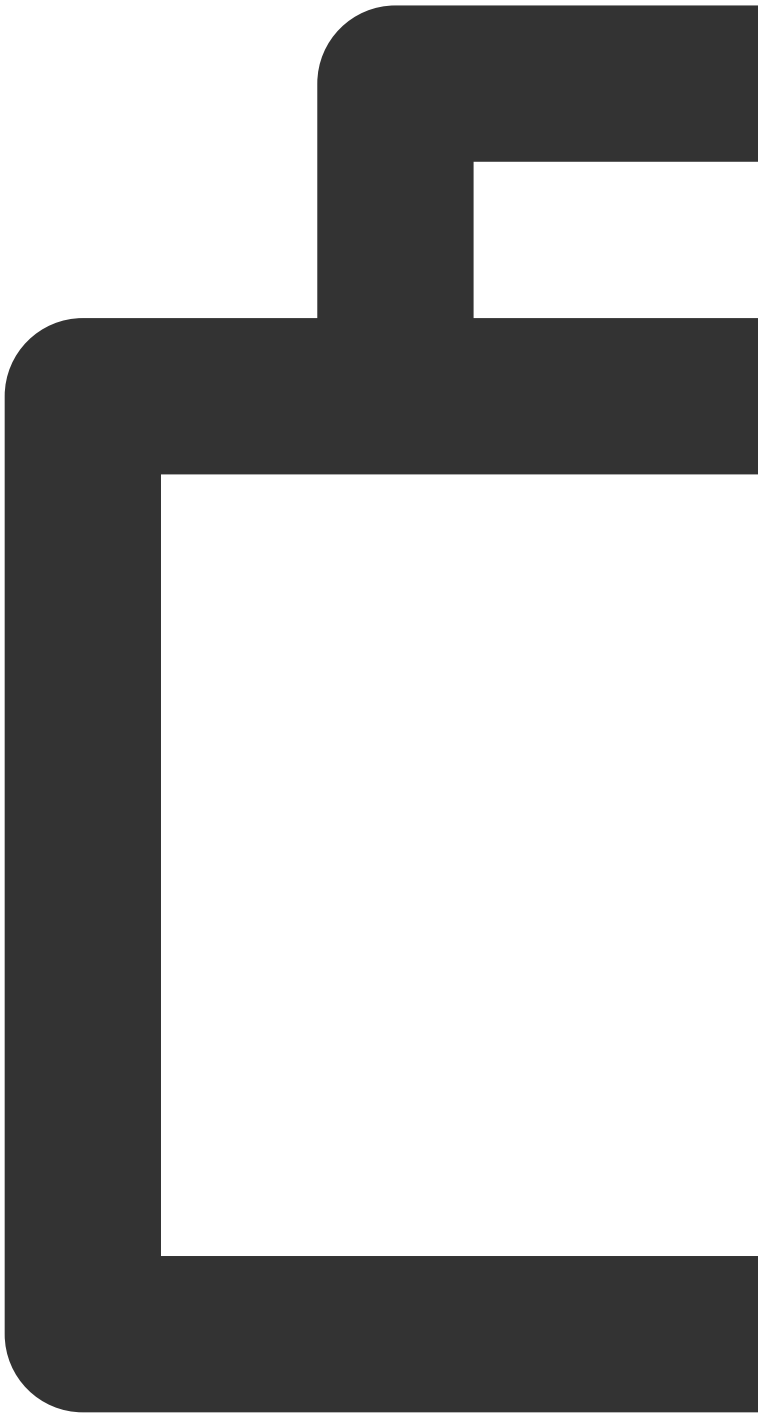
setBackground(options)



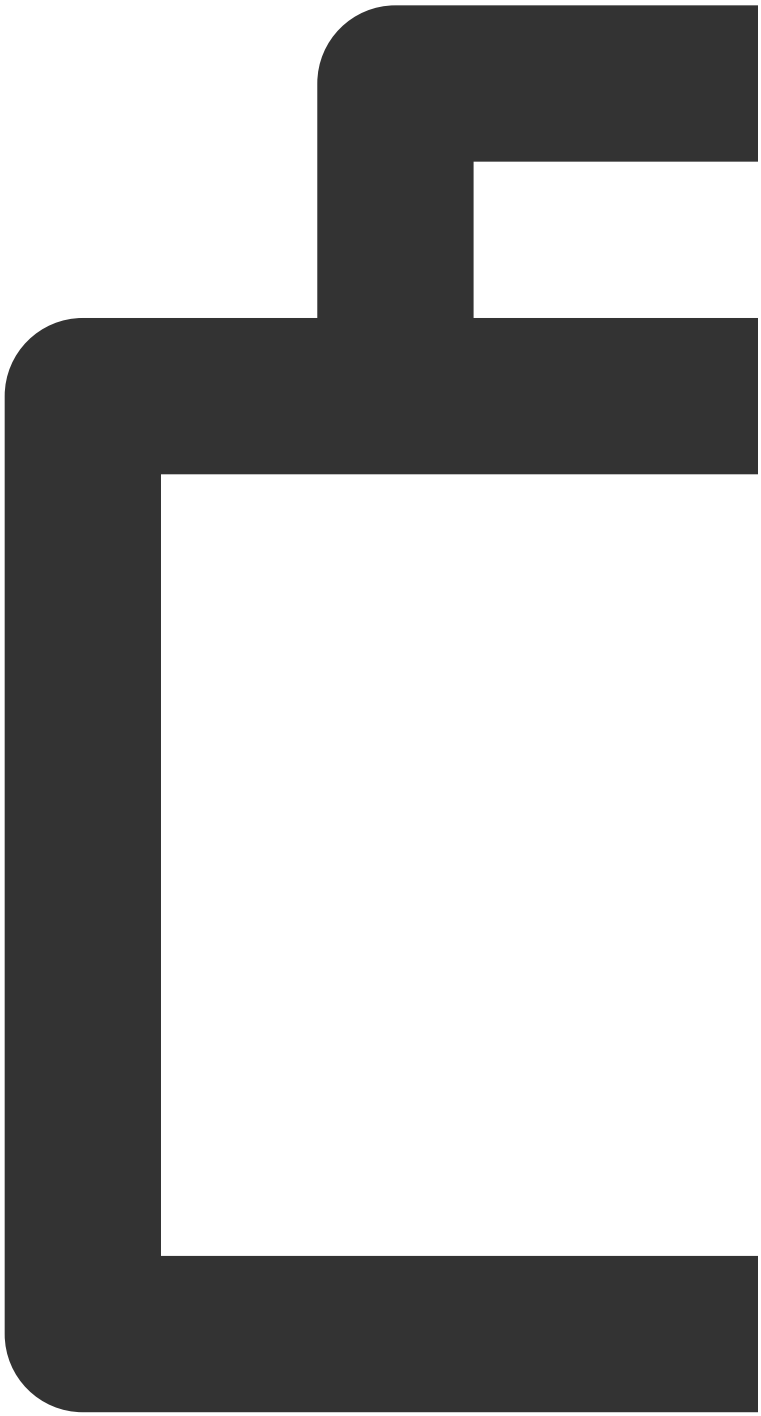
	`` { type: 'image|blur|transparent', src: string // 仅image类型需要 } ```
setSegmentationLevel(level)	level: 0 | 1 | 2
setFilter(id, intensity, callback)	id：滤镜 ID intensity：滤镜强度，范围0 - 1 callback：设置成功回调
getEffectList(params)	



```
{
    PageNumber: number,
    PageSize: number,
    Name: '',
    Label: Array,
```

	Type: 'Custom|Preset' }
getAvatarList(type)	 type = 'AR' | 'VR'

getEffect(effectId)	effectId：特效 ID
getCommonFilter()	-
async updateInputStream(src:MediaStream) **（0.1.19版本后支持）**	src：新的输入流MediaStream
disable()	-
enable()	-
async startRecord()	-
async stopRecord()	

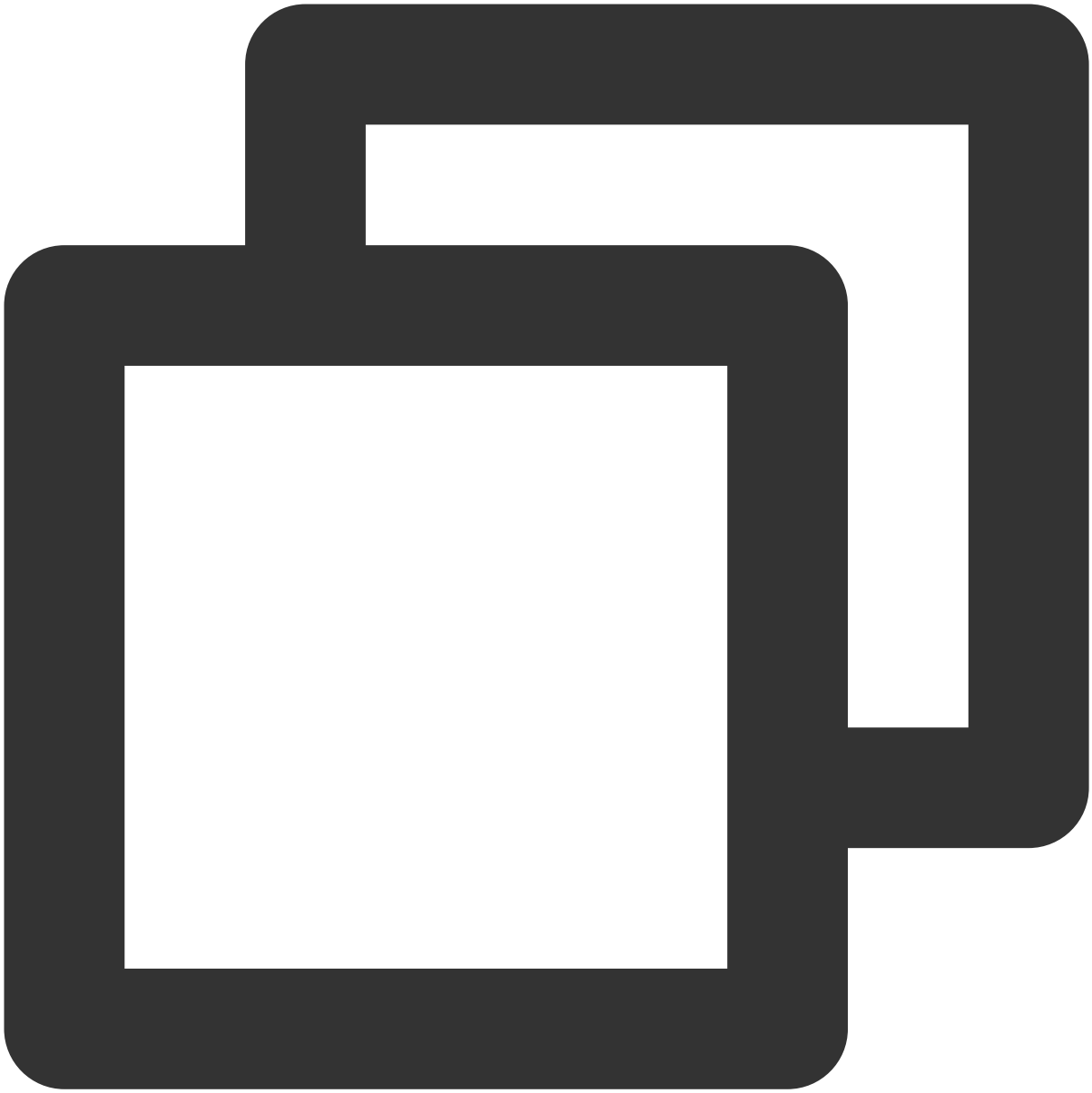


```
{
    useOriginAudio: boolean, // 是否录制视频原声
    musicPath: string, // 背景音乐地址，useOriginAudi
}
```

async takePhoto()	-
destroy()	-

错误处理

在 error 回调返回的对象中包含错误码与错误信息以方便进行错误处理。



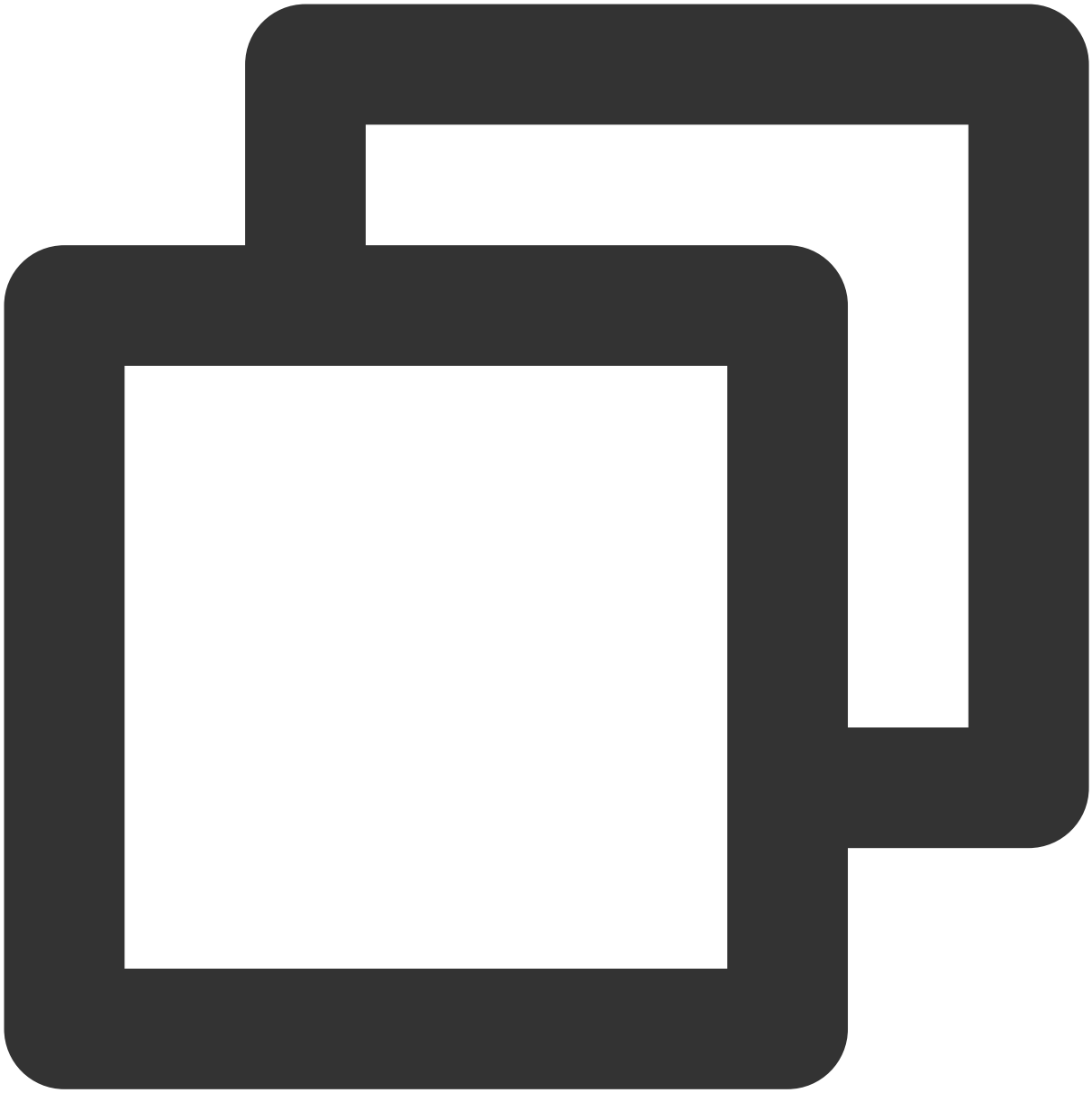
```
sdk.on('error', (error) => {
    // 在 error 回调中处理错误
    const {code, message} = error
    ...
})
```

错误码	含义	备注
10000001	当前浏览器环境不兼容	建议用户使用 Chrome、Firefox、Safari、微信浏览器访问

10000002	当前渲染上下文丢失	-
10000003	渲染耗时长	考虑降低视频分辨率或屏蔽功能
10000005	输入源解析错误	-
10000006	浏览器特性支持不足，可能会出现卡顿情况	建议用户使用 Chrome、Firefox、Safari、微信浏览器访问
10001101	设置特效出错	-
10001102	设置滤镜出错	-
10001103	特效强度参数不正确	-
10001201	调起用户摄像头失败	-
10001202	摄像头中断	-
10001203	没有获取到摄像头权限	需要开启摄像头权限，设置-隐私-相机开启
20002001	缺少鉴权参数	-
20001001	鉴权失败	请确认是否创建 License，请确认签名是否正确
20001002	接口请求失败	回调会回传接口返回的数据，具体信息请参见 接口错误码
20001003	设置特效接口鉴权失败	无权访问的接口，基础版 License 无法使用高级版License 功能
30000001	小程序 startRecord 失败	-
30000002	小程序 stopRecord 失败	-
40000000	未捕获的异常	-
40000001	当前使用 SDK 版本过低，部分特效无法正确展示，请升级 SDK 版本	-
50000002	分辨率改变导致特效丢失	需要重新设置特效

处理当前渲染上下文丢失

部分 PC 在长期切后台的场景可能触发处理 contextlost 错误，可以调用 `ArSdk.prototype.resetCore(input: MediaStream)` 恢复渲染上下文。



```
sdk.on('error', async (error) => {
    if (error.code === 10000002) {
        const newInput = await navigator.mediaDevices.getUserMedia({...})
        await sdk.resetCore(newInput)
    }
})
```

控制台指南
素材制作
3D特效

最近更新时间：2022-12-01 17:41:10

SDK 从0.3.0版本开始支持3D特效功能，使用前请确认版本信息。

3D特效指用户根据标准的头模，在相关3D软件中（如 Blender，3DSMax 等）制作相关3D模型，上传至控制台中，实现更加真实的效果。

素材规范

为了使模型更加贴合，效果更加真实，请下载我们提供的 标准头模，根据标准头模制作3D素材。

类型	规范
模型格式	GLB 文件或 GLTF 文件夹
模型大小	考虑到加载时间，单个模型大小请控制在5M以内
模型面数	考虑到包体积，单个模型请控制在10W面以内
纹理贴图	正方形 PNG 格式，单张图不超过1024*1024，常用256、512、1024这三种尺寸 在保证清晰度的同时，请尽量压缩贴图大小
材质属性	表（曲）面使用原理化 BSDF（Principled BSDF），否则会导致材质贴图渲染异常

制作步骤

步骤1：创作素材

1. 下载3D素材用标准头模，导入3D软件中（此处以 Blender 为例），制作贴合头模的3D素材，导出为 GLB 或 GLTF 类型。

注意：

- 为了使素材更加贴合拍摄者视角，请勿随意更改默认头模的位置、缩放等变换属性。
- 为了降低特效包的体积，更快地加载资源，推荐使用 GLB 格式作为导出首选项。

步骤2：导入控制台

1. 打开控制台素材制作页面，默认展示的是2D特效用内置模特，单击顶部的**3D特效**，会切换为3D编辑用头模，与上述标准头模为同一模型。
2. 单击右侧的**选择模型**，会弹出模型类型选项窗口，根据不同的数据类型选择不同的入口即可。此处以 GLB 模型为例，演示导入流程。

注意：

- 仅可添加一个3D特效场景，如果需要导入多个3D模型，可通过单击左侧列表中的 + 添加。
- 单击左侧节点切换当前编辑的3D模型。

步骤3：调整素材参数

理论情况下，模型是根据上述默认头模制作而成，导入控制台中即可直接使用，无需进行二次调整。特殊情况下若需调整模型的相关参数，可通过单击顶部的**切换编辑属性**，在下方编辑区域拖拽定位，或直接修改右侧相关属性参数值进行调整。

注意：
若导入的是现有的素材，并非根据默认头模制作而成，可以根据模型类型调整对应的素材的追踪点信息，以实现更精准的定位效果。如眼镜类模型可以选择鼻梁作为追踪点，胡子类类型可以选择人中作为追踪点等。

步骤4：预览效果

右侧的预览窗口可以预览素材置于内置模特人脸之上的效果，也可以切换为电脑摄像头，查看自身面部效果。

步骤5：导出

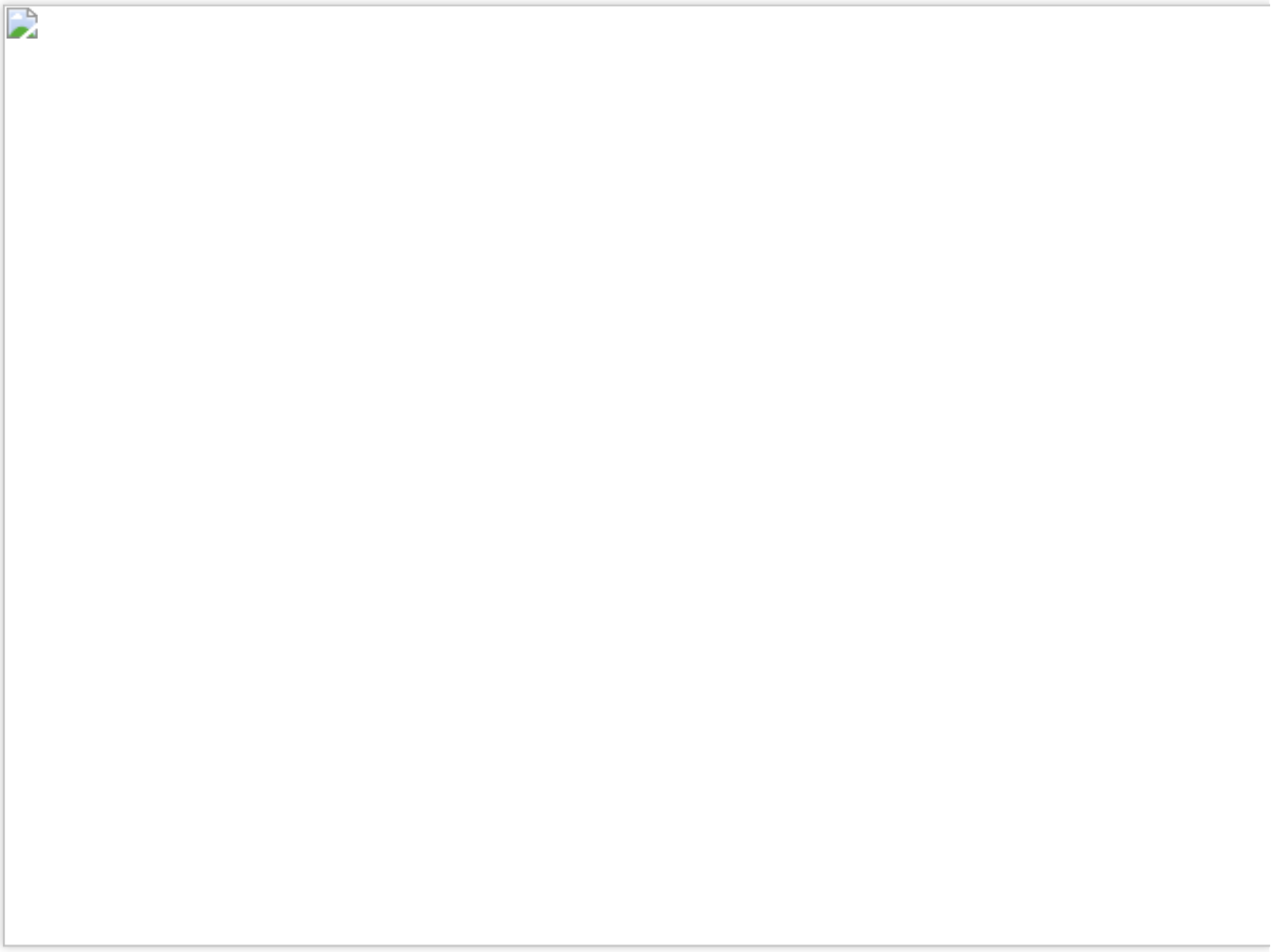
确认效果后，单击右上角**导出素材**，弹出框中选择封面、填写特效名称等信息后即可**发布**，已发布的素材可在**素材管理**列表中查看。

素材制作基础

最近更新时间：2023-03-28 16:59:17

素材制作面板入门

准备工作完成，申请 Web License 后就可以进入素材制作了，素材制作页面布局如下：

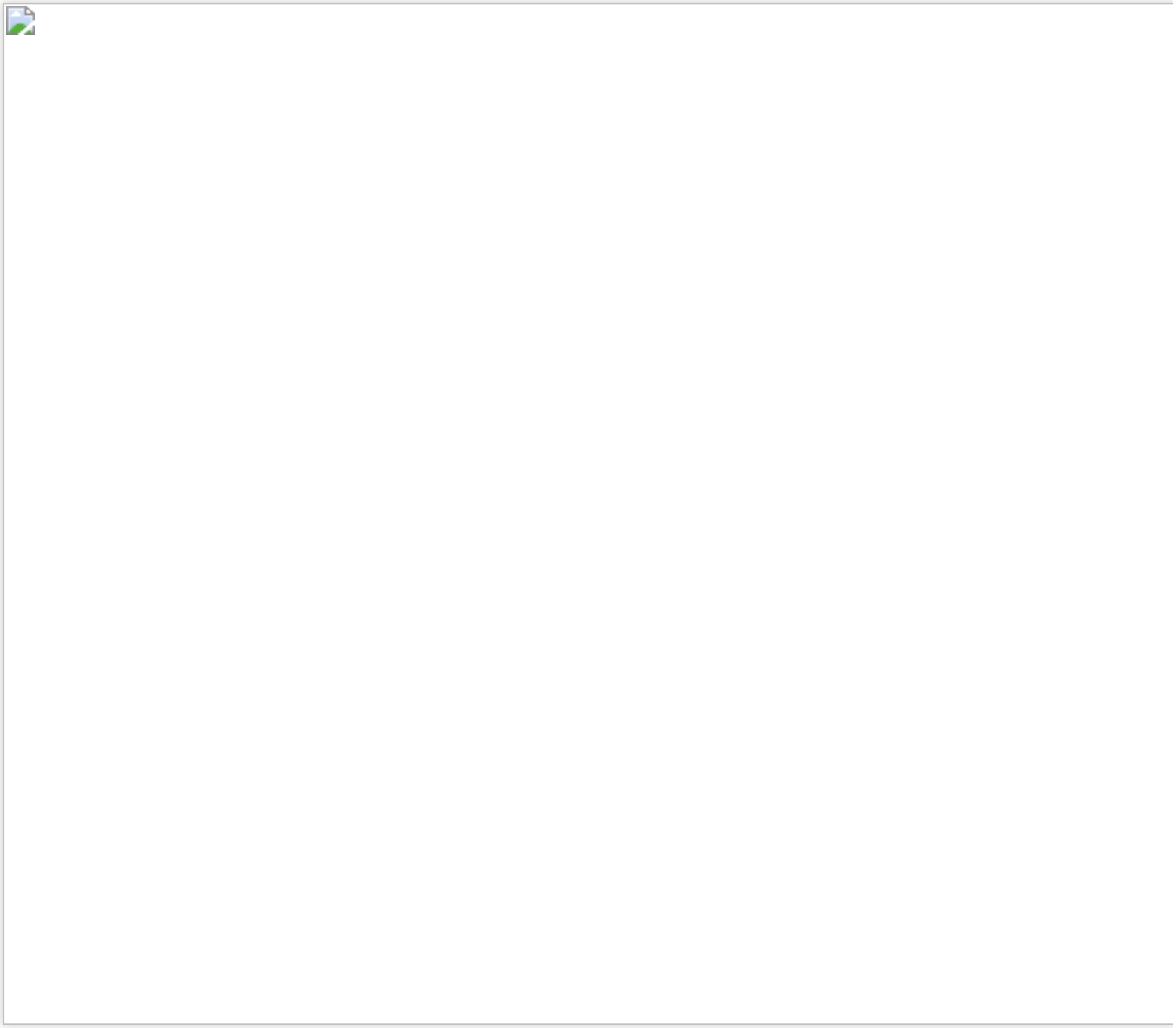


整体分为上下两部分，顶部为操作按钮区，其下方为主制作区。主制作区自左至右主要分为 ①—素材信息区、②—操作面板区、③—参数配置区 及 ④—效果预览区。

素材信息区

素材信息区展示当前使用的素材列表，类常规文件夹风格，右键功能提供常用的复制、粘贴等操作，也可以使用快捷键。**组合**功能可以将多个素材归纳到一个分组中，方便操作。单击列表或分组最右边的

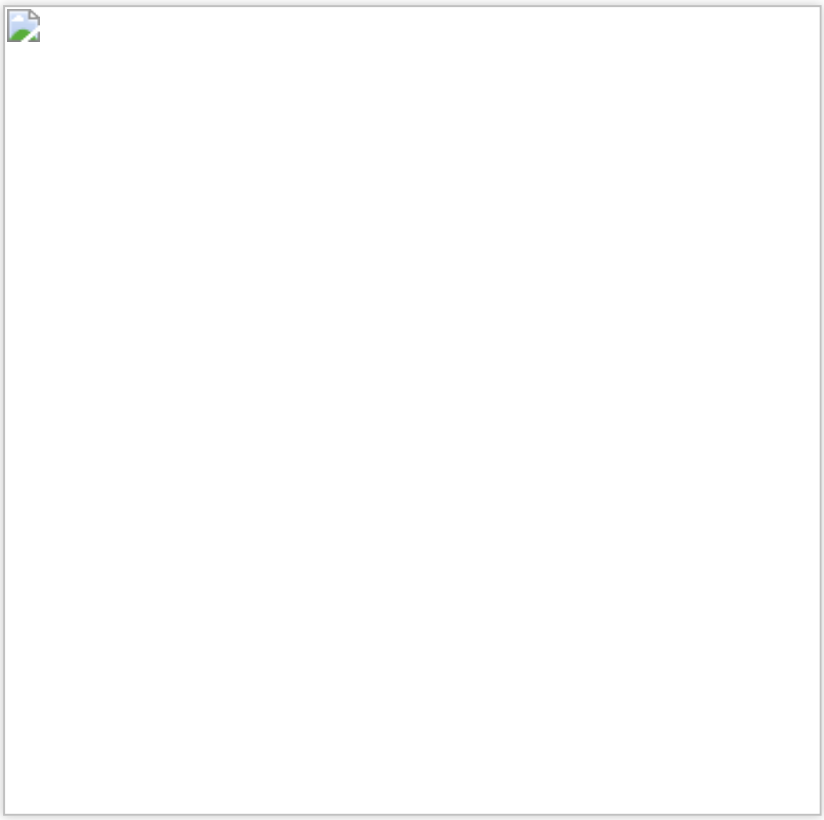
按钮可以控制素材的显示/隐藏效果。选中一个素材后，可以长按拖动改变位置，或者编辑素材的内容。



操作面板区

操作面板区是素材制作的主要操作区域，将各素材的参数设置可视化，可以放大、缩小操作区，按住拖动改变操作区位置等。

针对贴纸类素材，支持拖动图像边框的角，以调整图像的大小。以及支持拖动图像，以改变贴纸的位置。



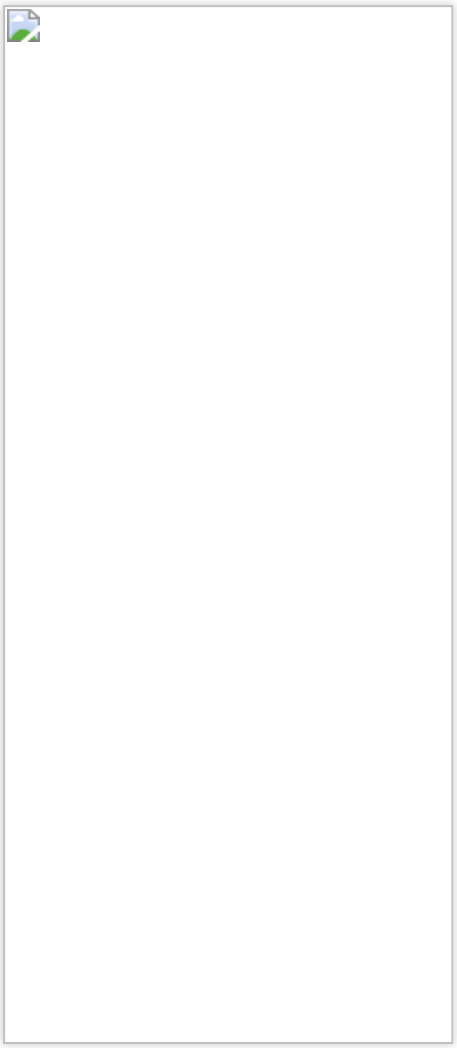
参数配置区

参数配置区支持编辑选定素材的属性，常见的如透明度、混合模式等。贴纸素材额外支持设置位置，大小，锚点位置属性。

每种素材都有对应的 PSD 模板，鼠标悬浮到**选择素材**旁边的按钮，即可下载使用。

效果预览区

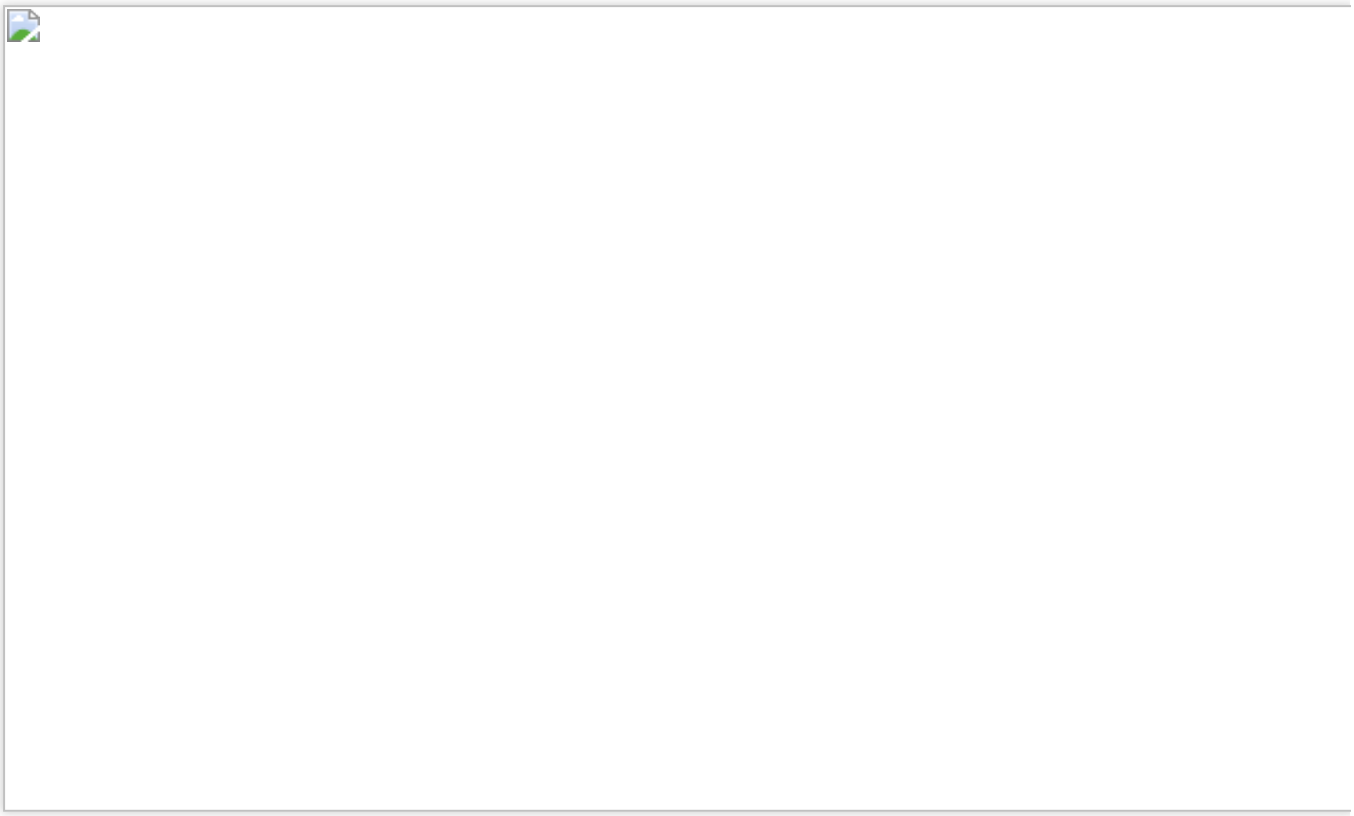
效果预览区可以将制作的特效实时展示出来，有男/女两个内置模特，也可以选择电脑摄像头查看真实效果。



制作素材四步骤

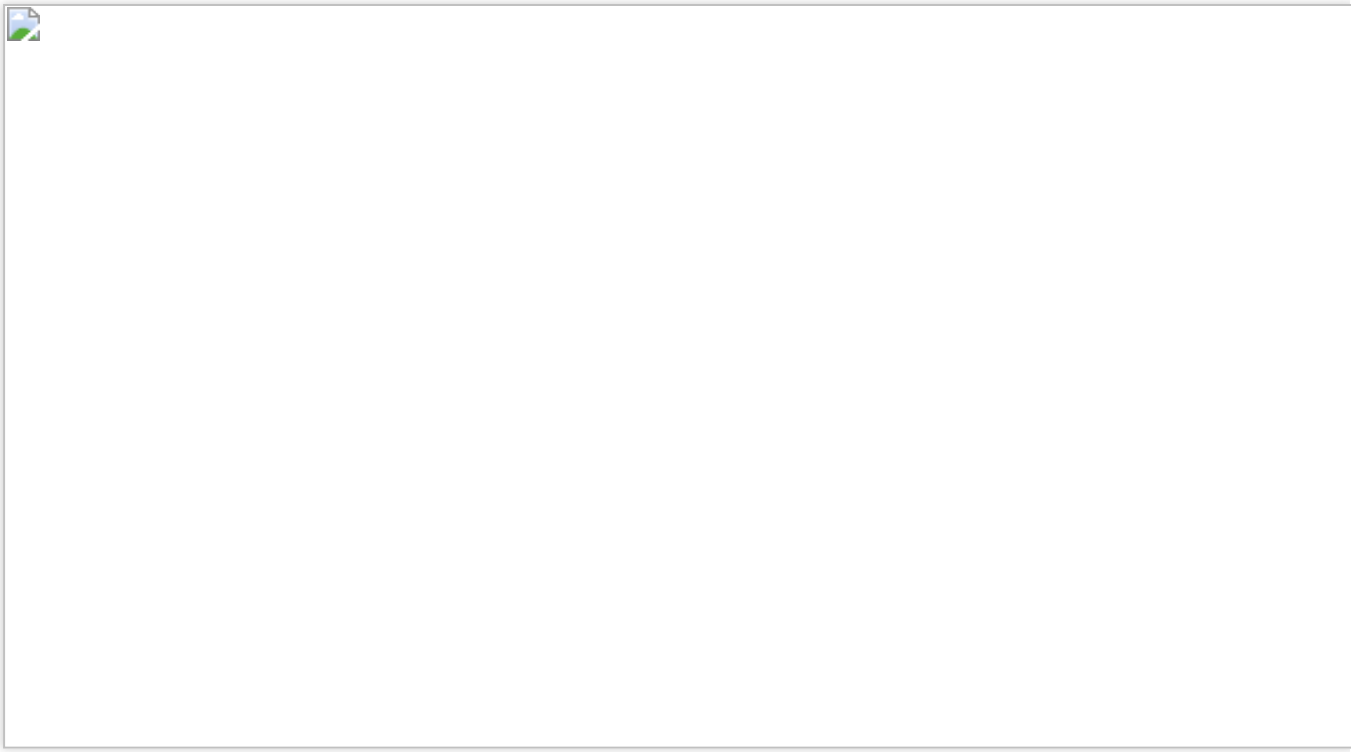
步骤1：模板下载

1. 做一个能够与人脸贴合的特效素材，需要下载官方的**专用人脸模型**，导入 PS、AE 等软件，进行贴合的妆容绘制。
2. 单击**美妆效果** > **眉妆**或**眼妆**，在**参数配置区**单击**模板下载**，即可下载标准的模板。



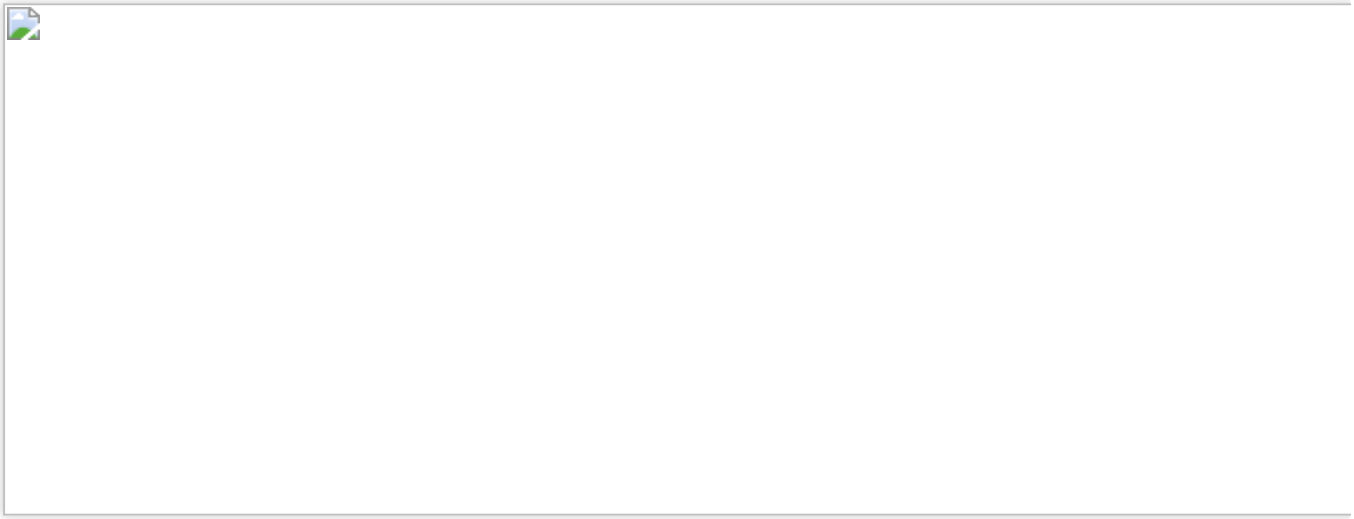
说明

一个标准的人脸模型 PSD 打开后如下所示，包含两部分：模型、面具。

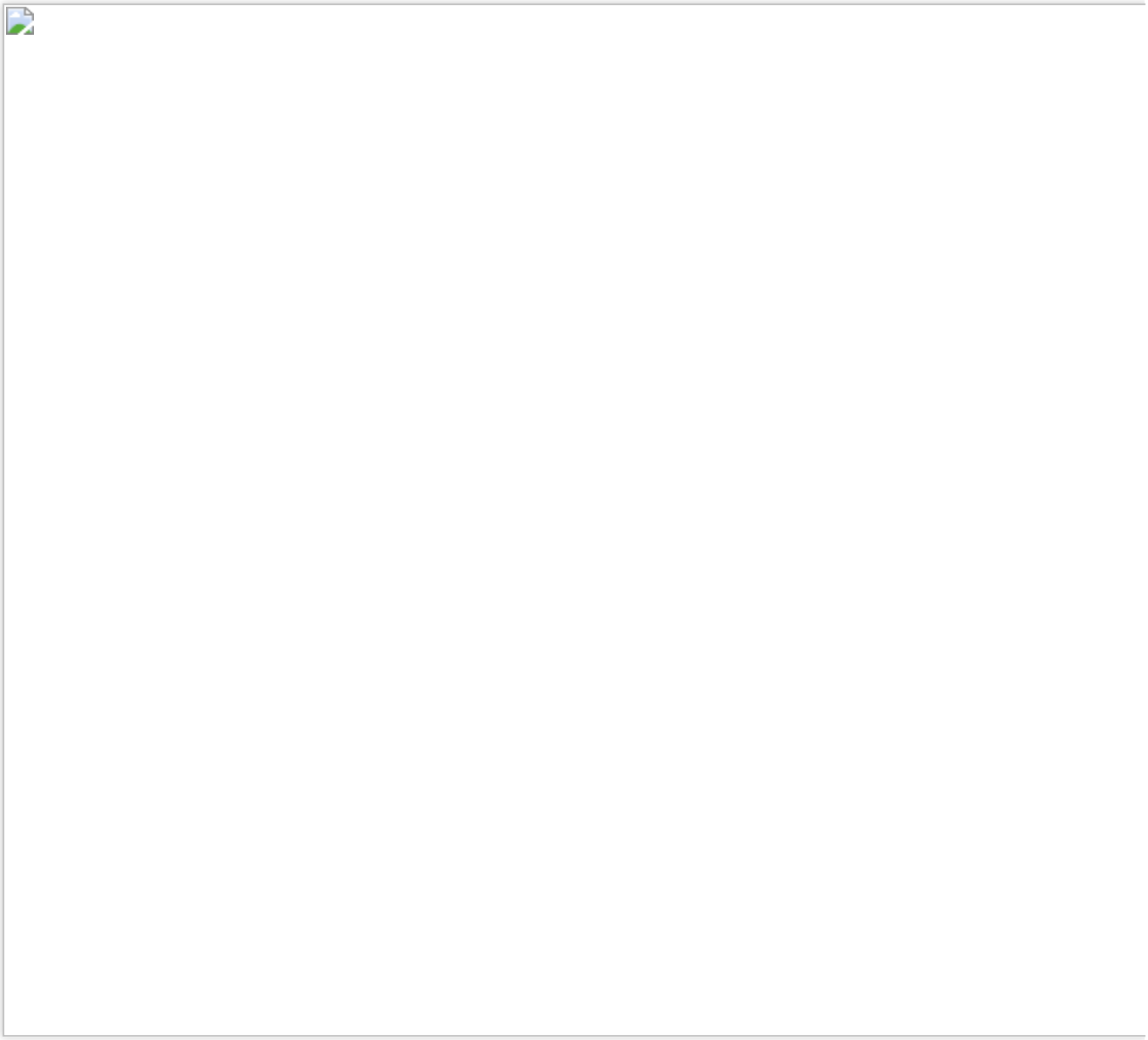


步骤2：素材制作

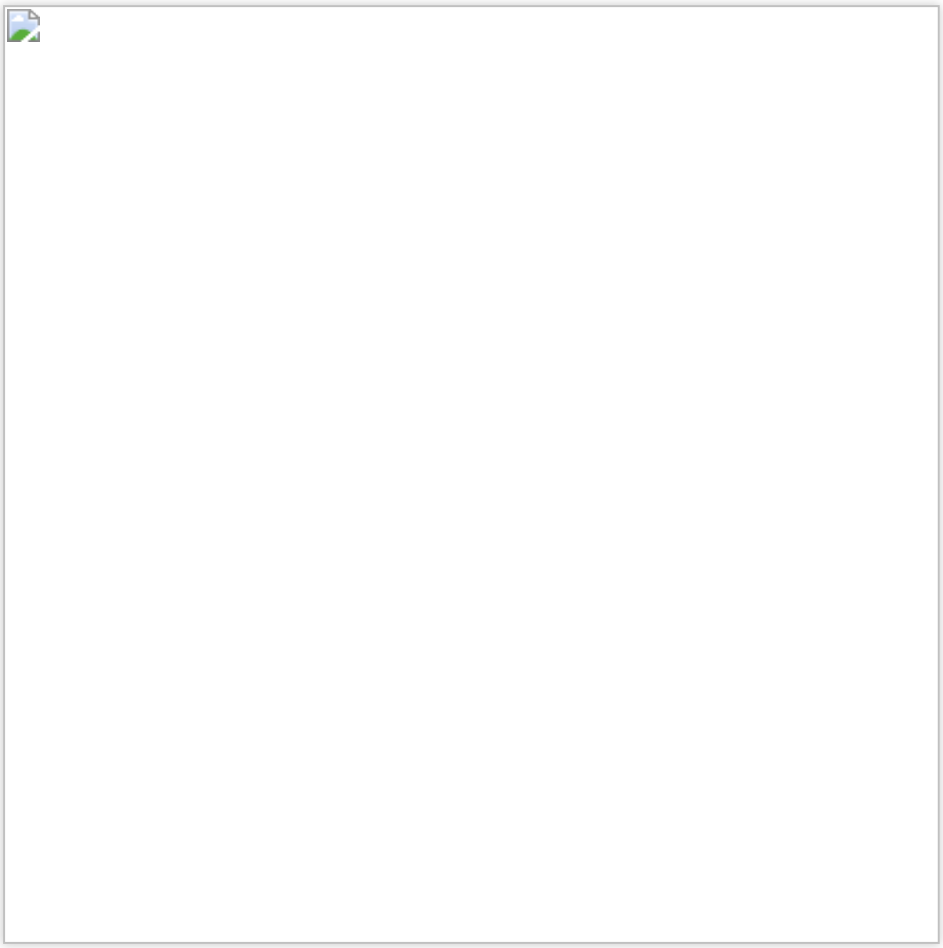
通常情况下，一套完整的特效素材会包含以下几部分：美妆，贴纸，滤镜。



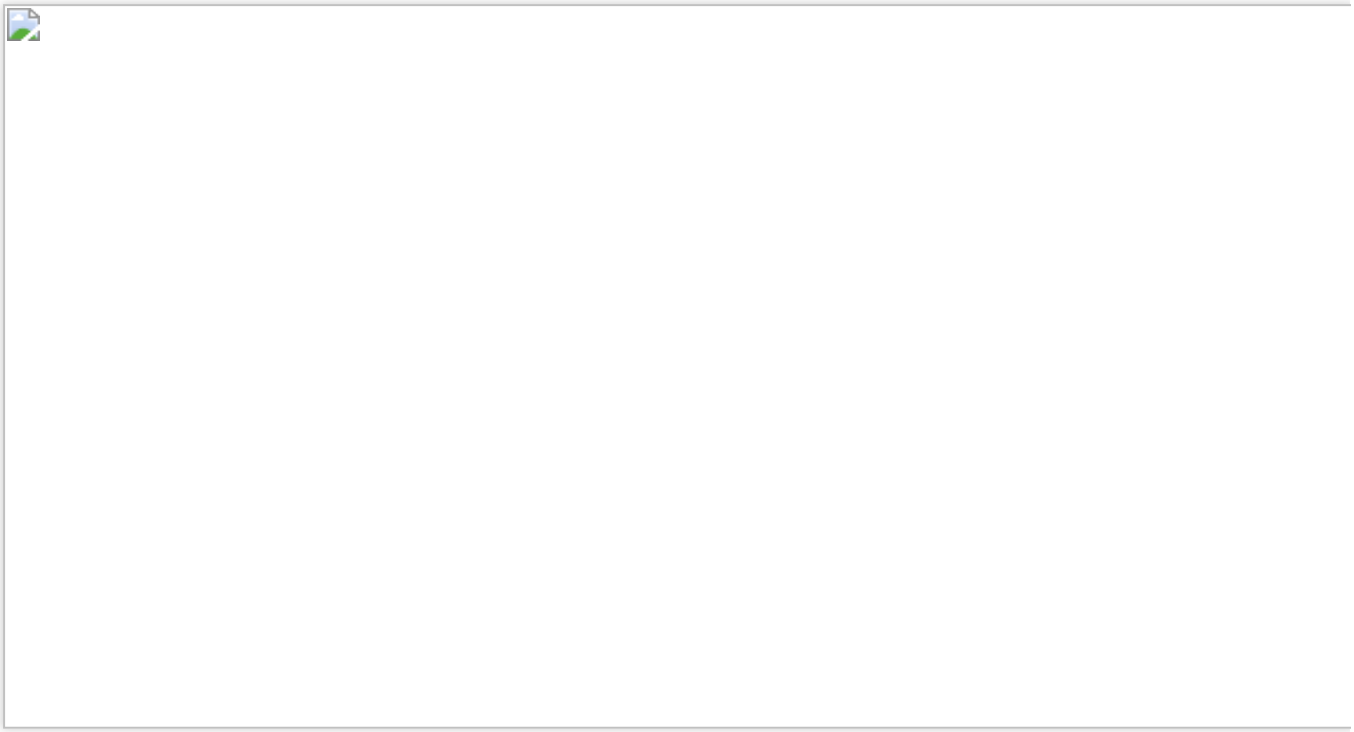
除了上述**专用人脸模型**之外，不同的素材类型（美瞳、滤镜）可能会有不同的 PSD 模板，可以单击顶部**美妆效果**按钮添加相应类型的素材，然后下载**参数配置区**对应的模板。



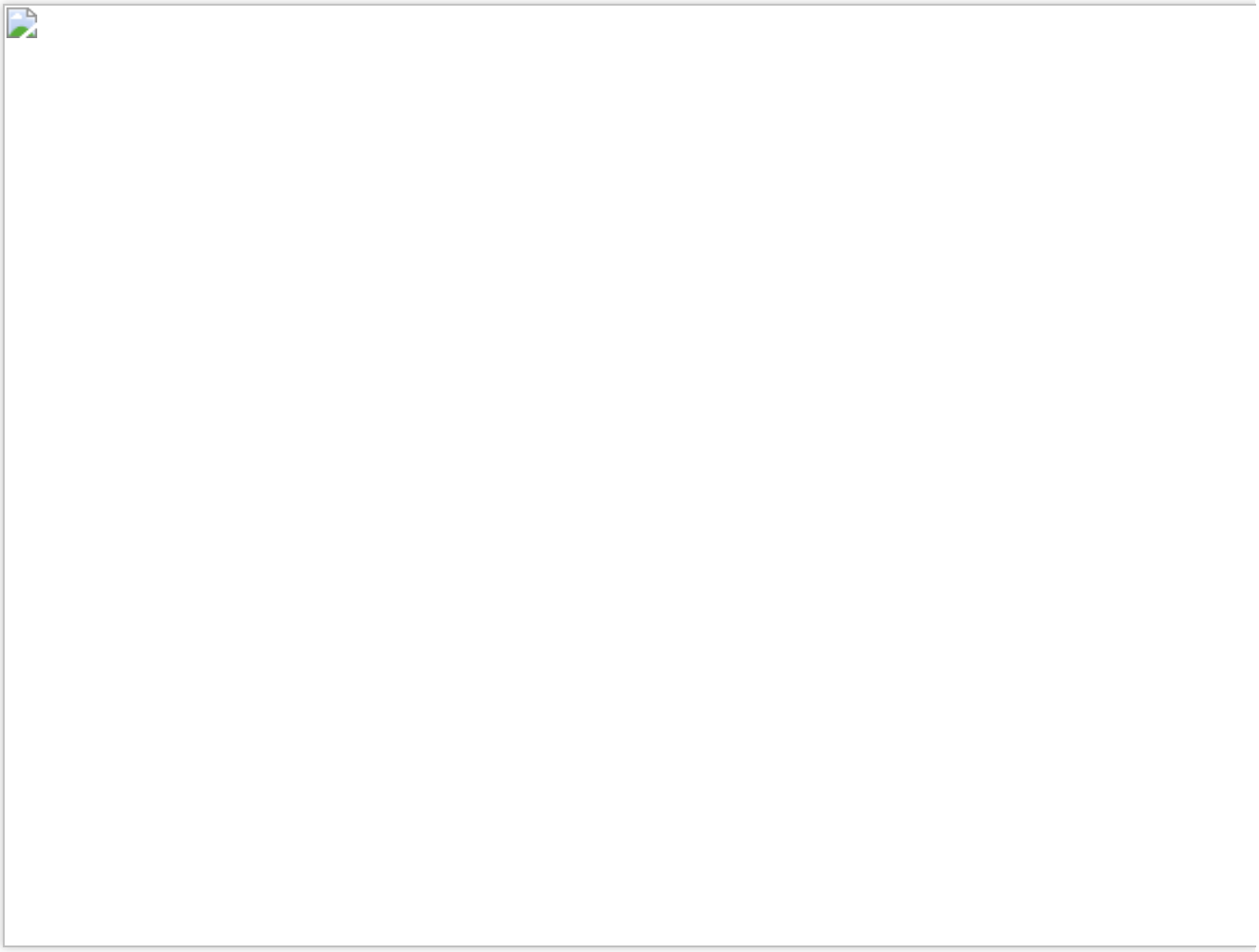
以 Photoshop 为例，打开上述标准 PSD 模板后，您可以在以下几个位置创作绘制。标准模板提供了多个参考点，为了保证最终的贴合效果，请确保素材制作时与模板人脸对应点位紧密贴合。



美瞳类 PSD 模板文件如下所示：



滤镜类 PSD 模板文件如下所示：



贴纸类素材不提供特定的模板，可以根据自身业务需要生成。

注意

不同部位的素材有不同的要求，请严格参考如下标准：

面部妆容（唇妆、眼妆、眉妆、面具）为标准的800×800的 PNG 图。

美瞳不强制要求800×800，只需要满足正方形即可，推荐150×150。

贴纸推荐小于1000×1000，超过的会被等比例压缩，总数小于100帧。

滤镜请使用模板提供的标准色卡，手机端可以在 VSCO 等滤镜调色软件里调整参数和想要的效果，然后将色卡导出在电脑中转成标准 LUT PNG。当然如果您擅长调色，也可以直接在 PS 里调色，最终输出的都是标准512×512的 LUT 色卡图。

步骤3：添加素材

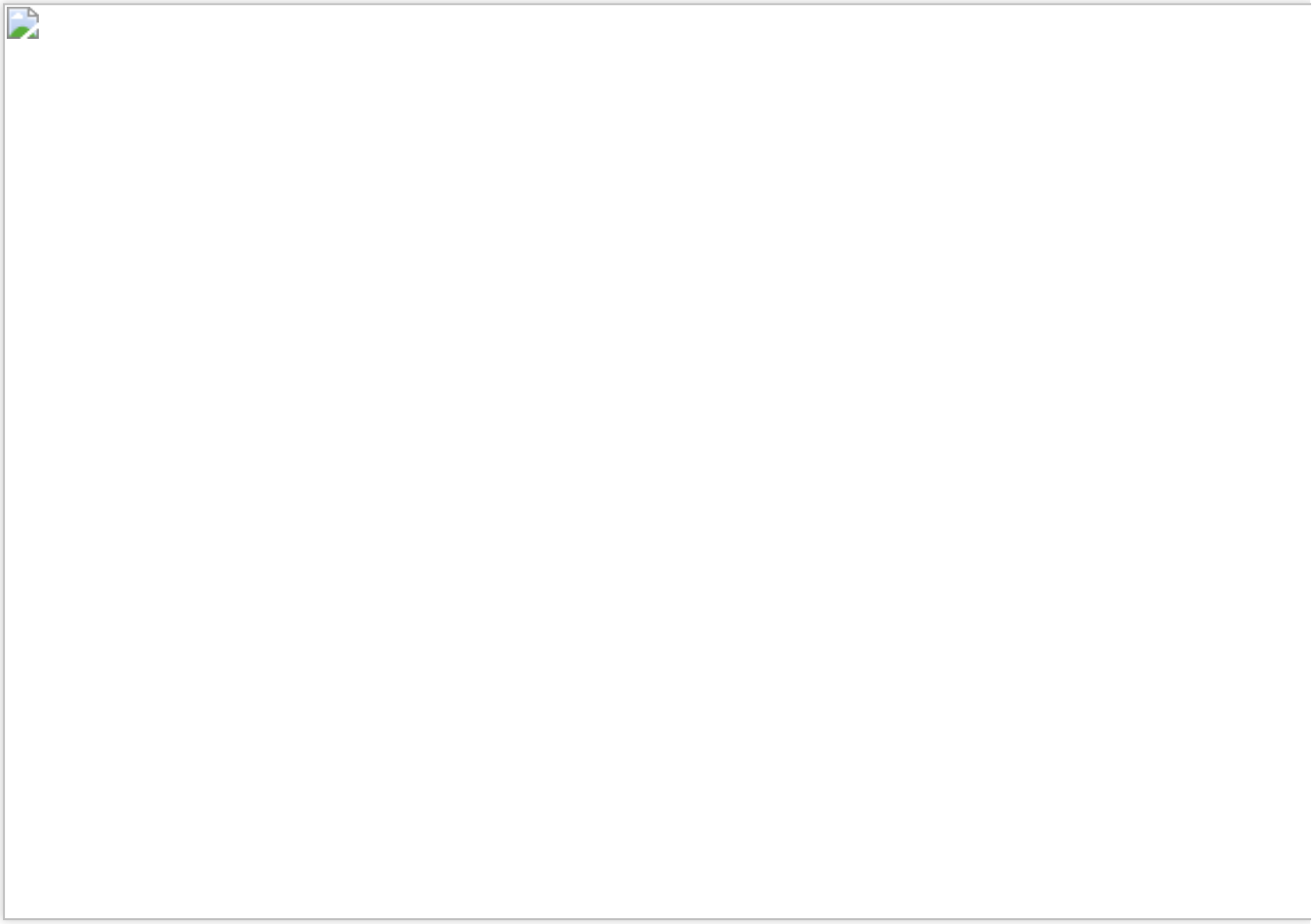
1. 素材制作完成后，可以根据不同的部位对号入座，导入到控制台中。

单击顶部**美妆效果**，添加对应的面部妆容，在**参数配置区**中替换对应的文件即可预览妆容效果。

单击顶部**滤镜效果**，添加默认滤镜，在**参数配置区**中替换对应的 LUT 文件即可预览滤镜效果。

单击顶部**人脸贴纸**，添加默认贴纸，在**参数配置区**中导入提前制作好的序列帧图片列表，即可预览贴纸效果。

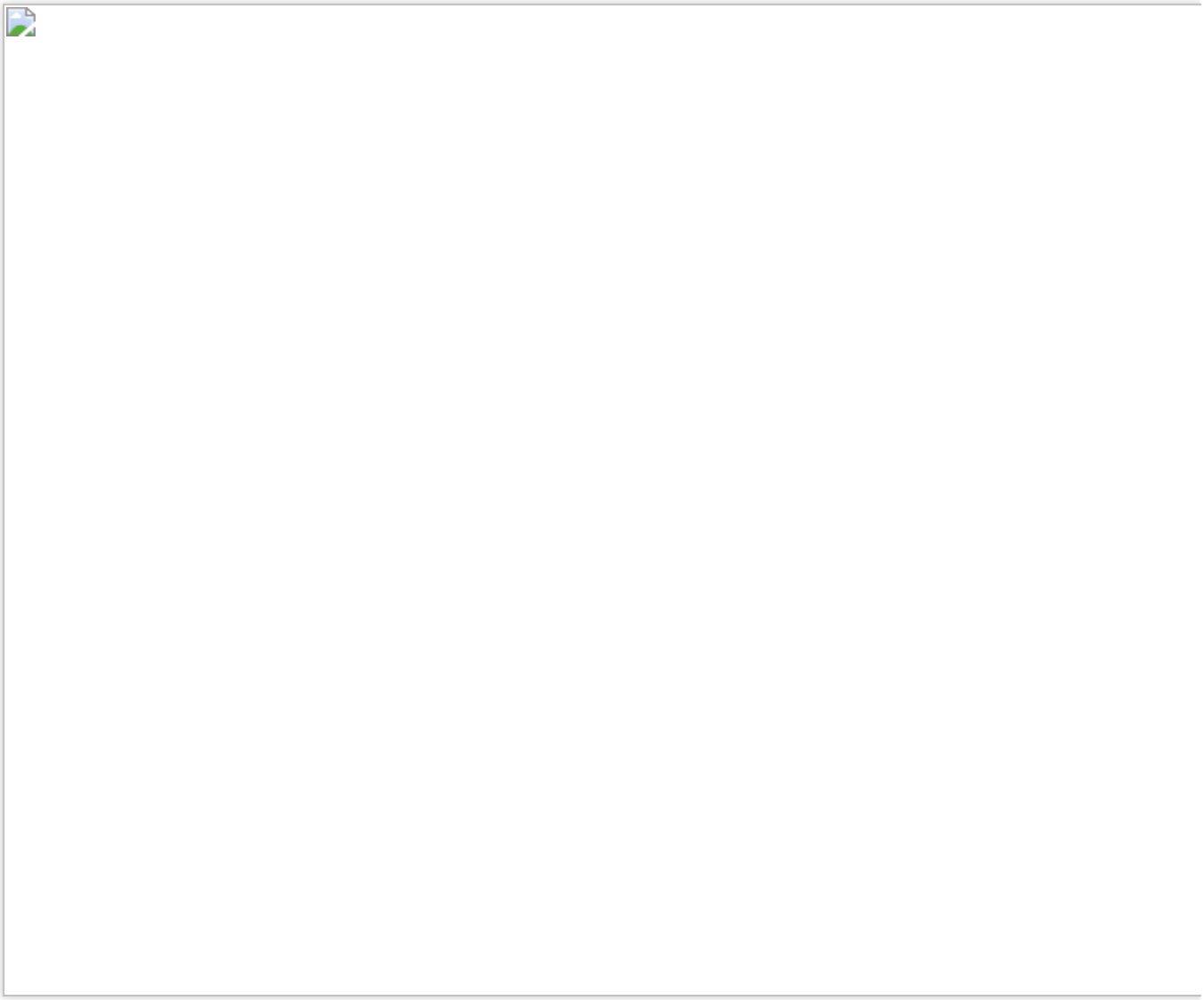
以眼妆为例，**素材信息区**列表选中相应素材后，替换成新制作的素材。



说明

除此之外，控制台内置了**素材库**，里面包含了较为通用基础的滤镜、美妆等，您可以把它们导入，成为自己素材的一部分。

2. 每添加一个素材，**参数配置区**都会出现该素材的**参数信息**，您可以对素材进行一些调整，例如**混合模式**（参考制作素材时 PS 的混合模式）、强度（即透明度）等，贴纸类素材还可以调整其位置大小及播放速率（FPS）等信息。



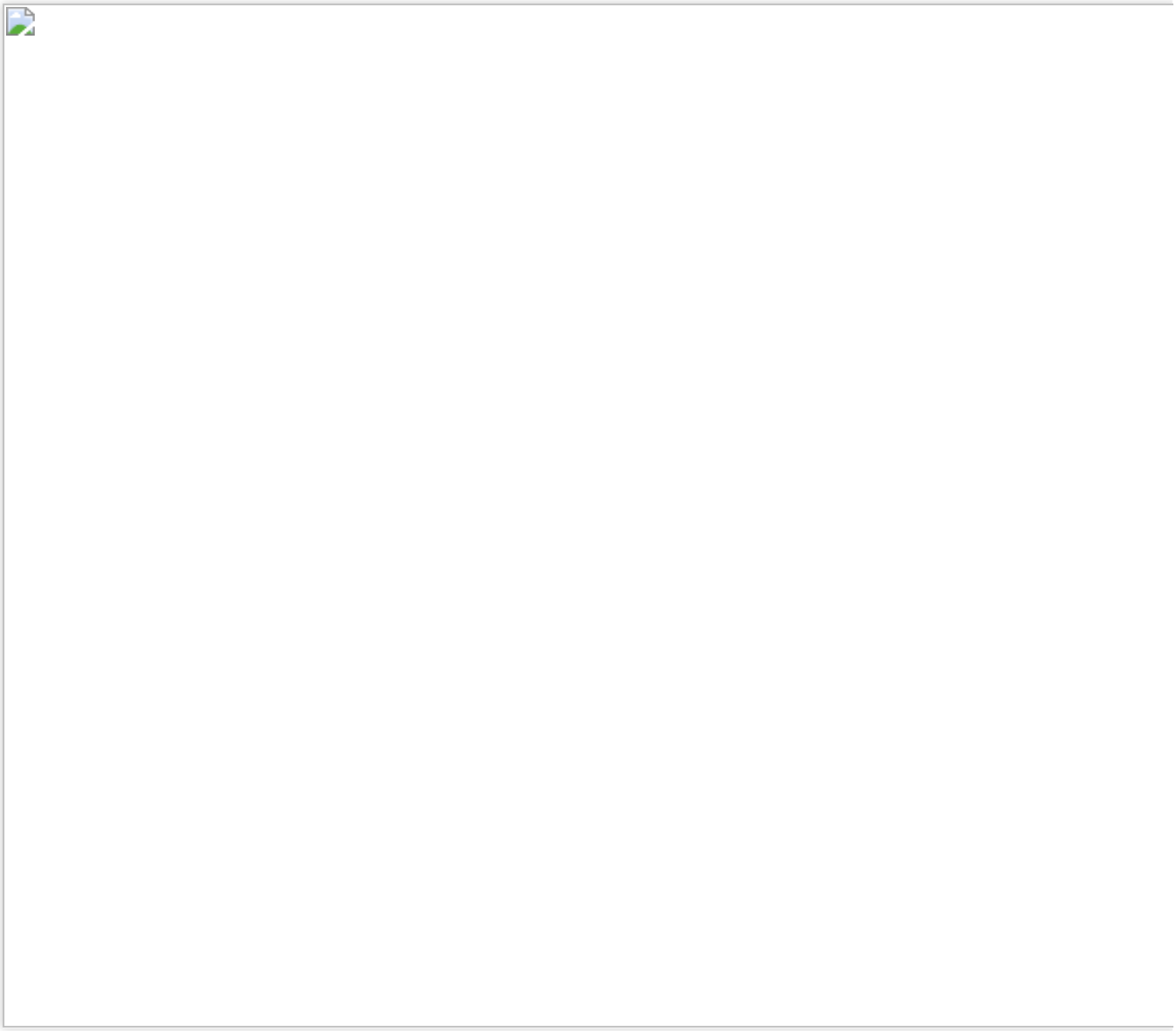
步骤4：导出素材

1. 素材设置完后，可以在**效果预览区**通过默认模特或者摄像头预览效果。
2. 确认后，单击控制台右上角**导出素材**按钮，填写下列素材信息后，单击**发布**即可导出。

上传素材封面。

填写素材名称。

选择已有标签或添加新标签。合理地使用标签属性可以起到素材分类的效果，例如根据项目类型为素材打上不同的标签（`Web 项目`、`小程序项目`等），以便于在特定的项目中根据标签过滤出相应的素材列表。



人脸贴纸

最近更新时间：2023-04-21 15:36:56

人脸贴纸功能，指通过上传 PNG / PNG 序列帧素材，实现素材跟随拍摄者头部移动的效果。

素材规范

贴纸类素材不提供标准的 PSD 模板，用户可以自行发挥，满足以下规范即可：

类型	规范
格式	透明背景 PNG / PNG 序列帧图，静态素材为单张 PNG，动态素材为**成组编号**的 PNG 序列帧，例如 `001.PNG` 、 `002.PNG` 等
尺寸	推荐小于1000 × 1000

帧数	不超过100帧
大小	单张素材不超过 1M

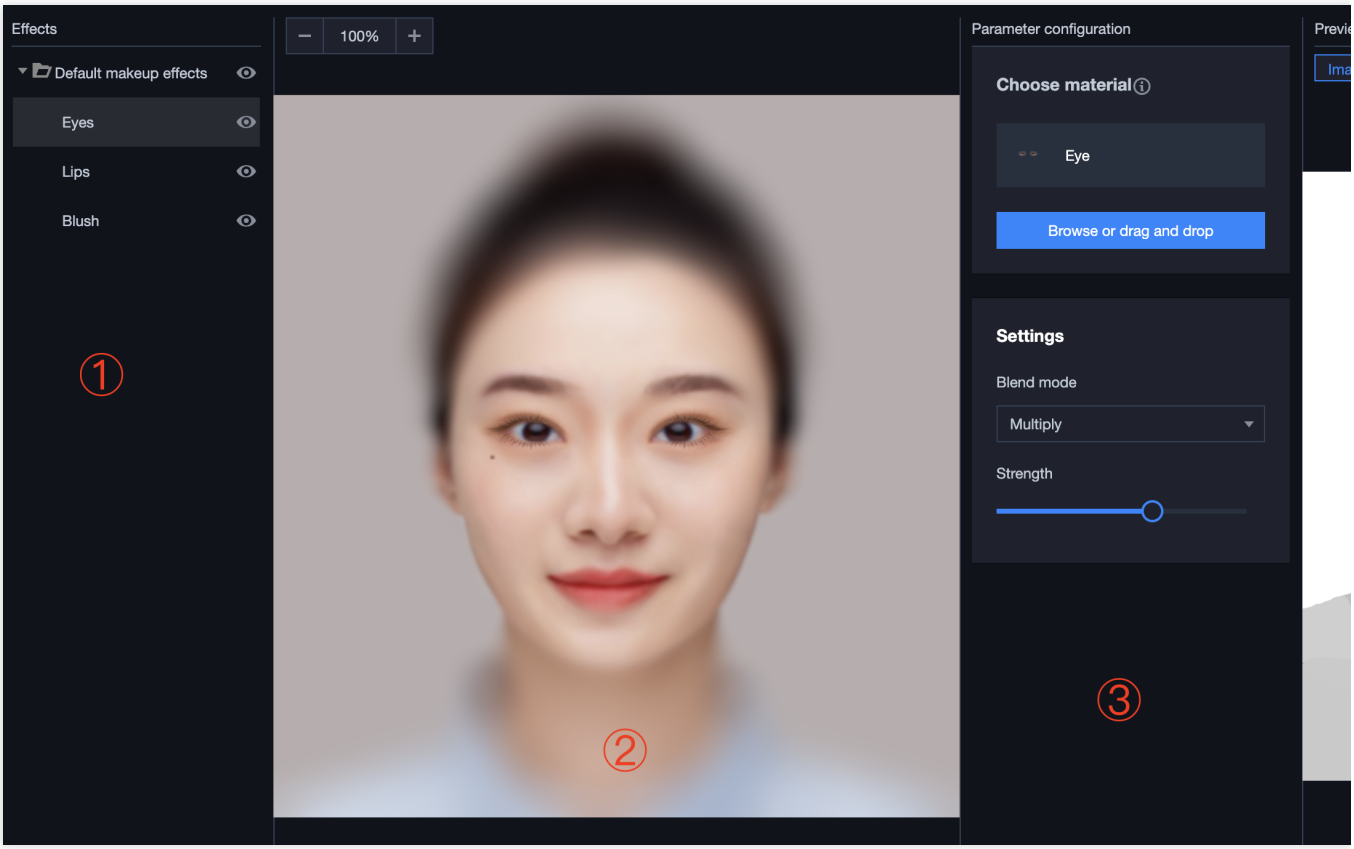
制作步骤

步骤1：创作素材

根据上述 素材规范，制作符合要求的 PNG / PNG 序列帧。

步骤2：导入控制台

1. 打开控制台**素材制作**，单击顶部**人脸贴纸**，添加一个默认的贴纸。
2. 在右侧**参数面板**的**素材资源**处，拖入或单击**选择**上传制作好的 PNG 素材图片，即可将素材导入控制台。



步骤3：调整素材参数

支持通过下述两种方法进行调整：

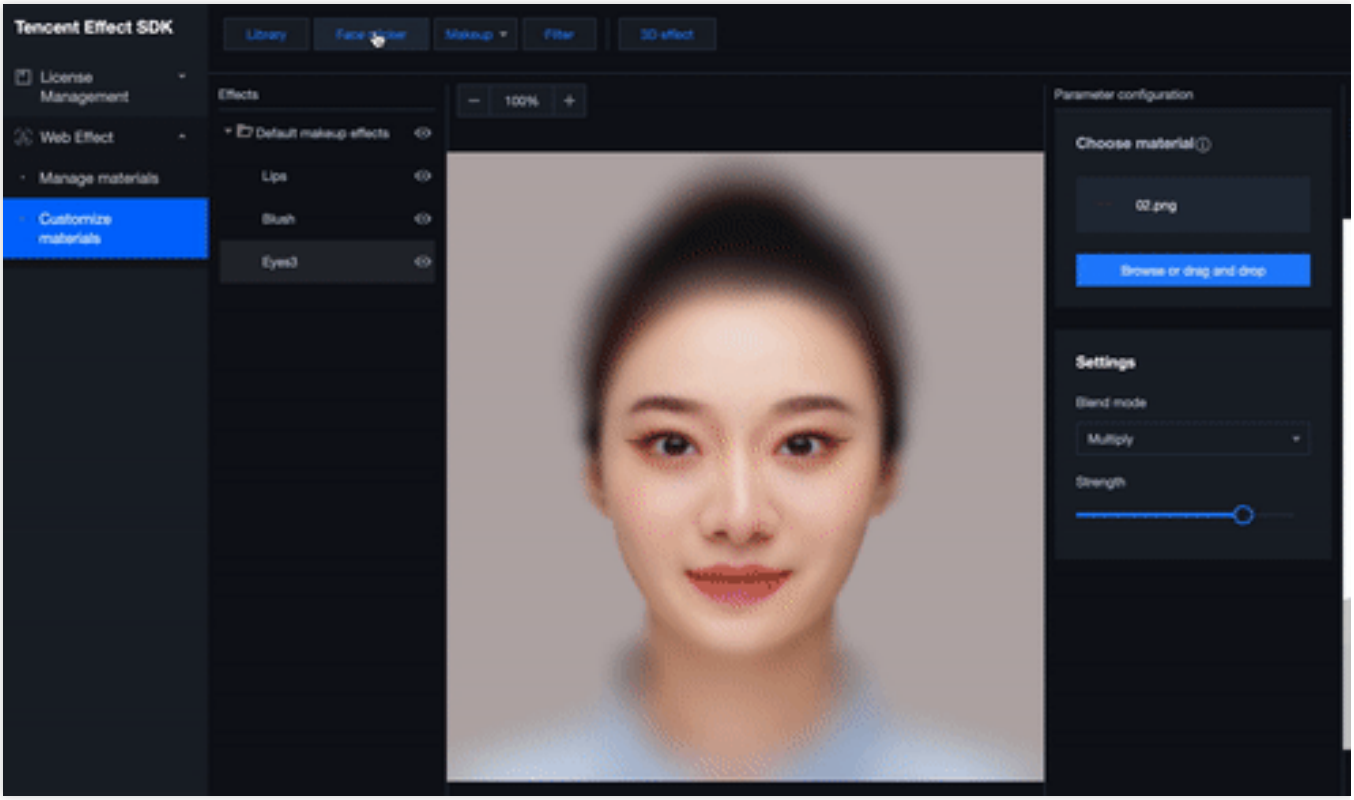
在操作面板区域，通过观察素材与面部跟踪点的相对位置，可视化调整面部贴纸素材与人脸的跟踪展示效果。

在参数面板内，调整该素材的**混合模式**、**强度（透明度）**、**位置大小**、**播放速率（序列帧）**、**锚点位置**等参数，可达到相应效果。

说明

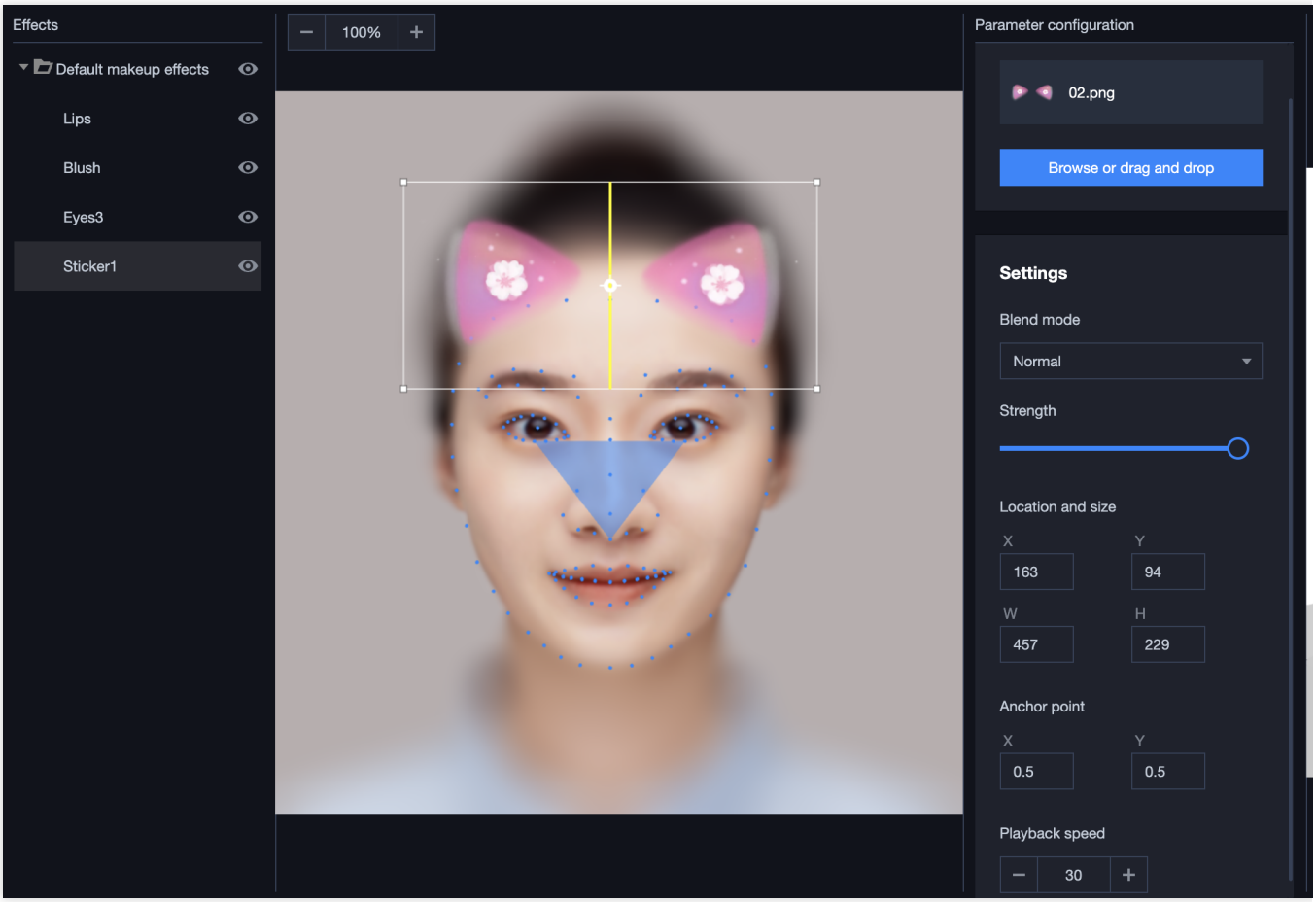
建议播放速率（序列帧）参数设置不超过30FPS。

锚点位置是贴纸位置的参考点，默认位置是贴纸的正中心，贴纸的锚点位置参数 `(x, y)` 是以贴纸左上角为基准的相对位置系数， `x` 为相对贴纸宽度的系数， `y` 为相对贴纸高度的系数，例如默认的锚点位置参数（贴纸中心点）为 `(0.5, 0.5)` ，若期望以贴纸左上角顶点为锚点，则设置锚点位置为 `(0, 0)` 。合适的锚点位置可以实现更好的贴纸跟踪效果，例如头饰类贴纸锚点适合落在前额位置，脸部贴纸如眼镜锚点适合落在鼻子中庭位置，具体可以多尝试下，找到最合适的位置。



步骤4：预览效果

右侧的预览窗口可以预览到该贴纸跟随人脸的效果，也可切换为电脑摄像头，真实地体验贴纸跟随自己头部运动的效果。


Effects
Default makeup effects
Lips
Blush
Eyes3
Sticker1
100%
Parameter configuration
02.png
Browse or drag and drop
Settings
Blend mode
Normal
Strength
Location and size
X
163
Y
94
W
457
H
229
Anchor point
X
0.5
Y
0.5
Playback speed
30

素材管理

最近更新时间：2023-06-21 16:56:27

登录**音视频终端 SDK控制台** > **Web 美颜特效** > 素材管理，该页面展示您在控制台制作的特效素材，提供包括查看、筛选、新增、删除、编辑等素材管理功能。

查看素材

素材表格中可以预览素材的封面图，并展示名称、标签、创建时间等属性。

筛选素材

素材表格支持按照素材的属性进行筛选，目前支持**名称**和**标签**两个维度，名称筛选支持模糊匹配。

编辑标签

素材支持添加多个标签，制作好的素材在素材管理页面可以再次编辑标签。鼠标移动到素材展示卡片上时，右上角显示编辑按钮，单击即可对素材标签进行新增或删除。标签可以帮助您更便捷地管理素材，良好的标签设置同时可以帮助您在页面中更好地展示素材和更方便地通过 SDK 使用素材。

删除素材

制作好的素材支持删除，鼠标移动到素材展示卡片上时，右上角显示删除按钮，单击并二次确认后即可删除当前素材，删除素材意味着永久销毁素材数据，无法找回。

Demo 体验

最近更新时间：2023-04-11 16:26:55

音视频终端 SDK（腾讯云视立方）提供的 Web 美颜特效 SDK 支持 PC Web、H5、微信小程序等多种平台，您可单击/扫码体验Demo。

平台	体验地址	环境说明
Web	单击体验	PC 环境推荐使用桌面端 Chrome 90+ 二维码可使用微信扫一扫直接内置浏览器打开，建议您使用最新版本微信客户端
微信小程序		建议您使用最新版本微信客户端

实践教程
结合 WebRTC 的推流

最近更新时间：2024-07-09 16:06:24

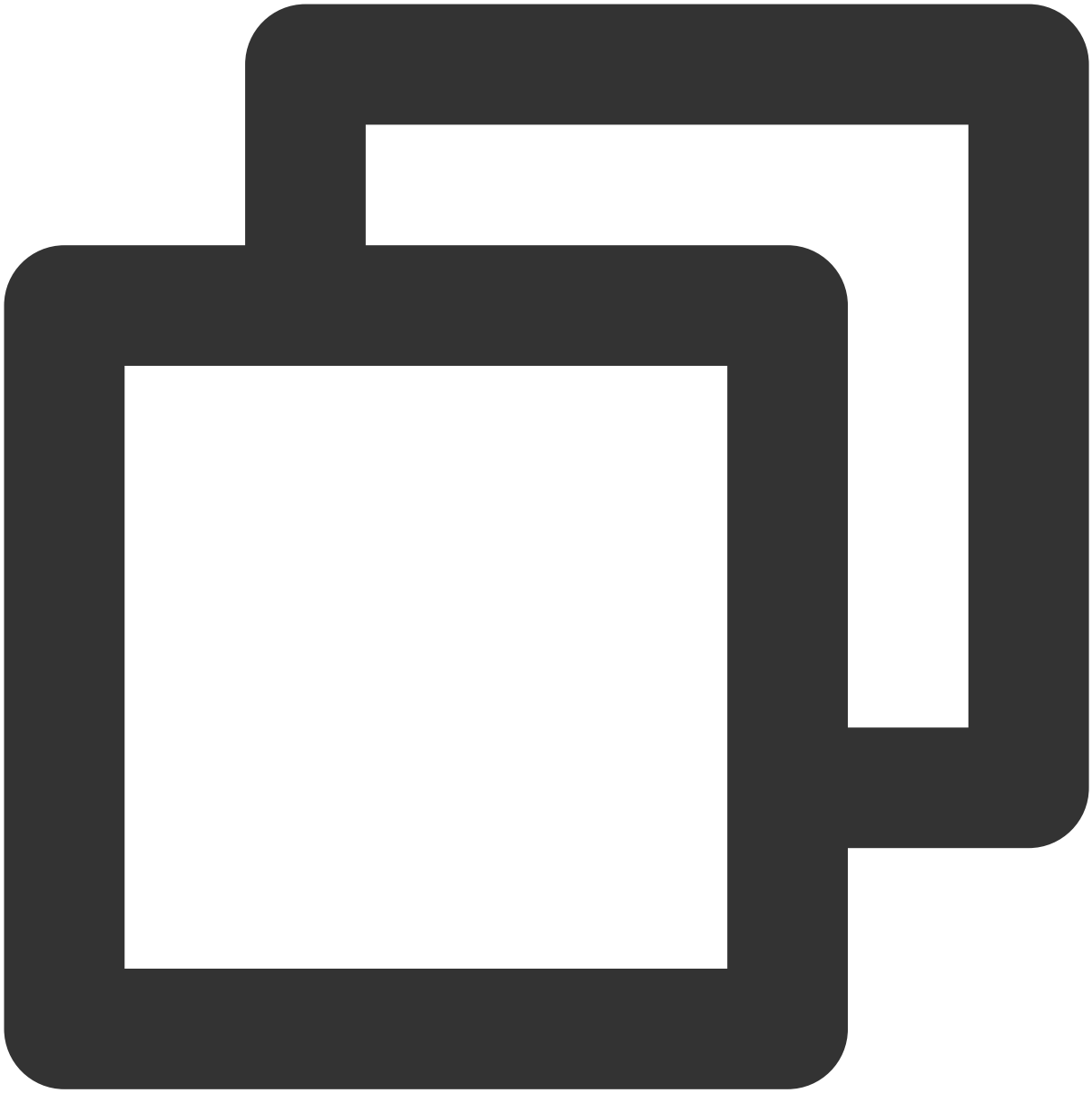
准备工作

请阅读 Web 美颜特效 SDK 接入指南，熟悉 SDK 基本用法。
请阅读云直播 入门文档 以及 WebRTC 推流，了解 WebRTC 推流工具基本用法，并完成直播基础设置。

开始使用

步骤1：Web 美颜特效 SDK 引入

在需要直播推流的页面（PC Web 端）中引入 js 脚本：



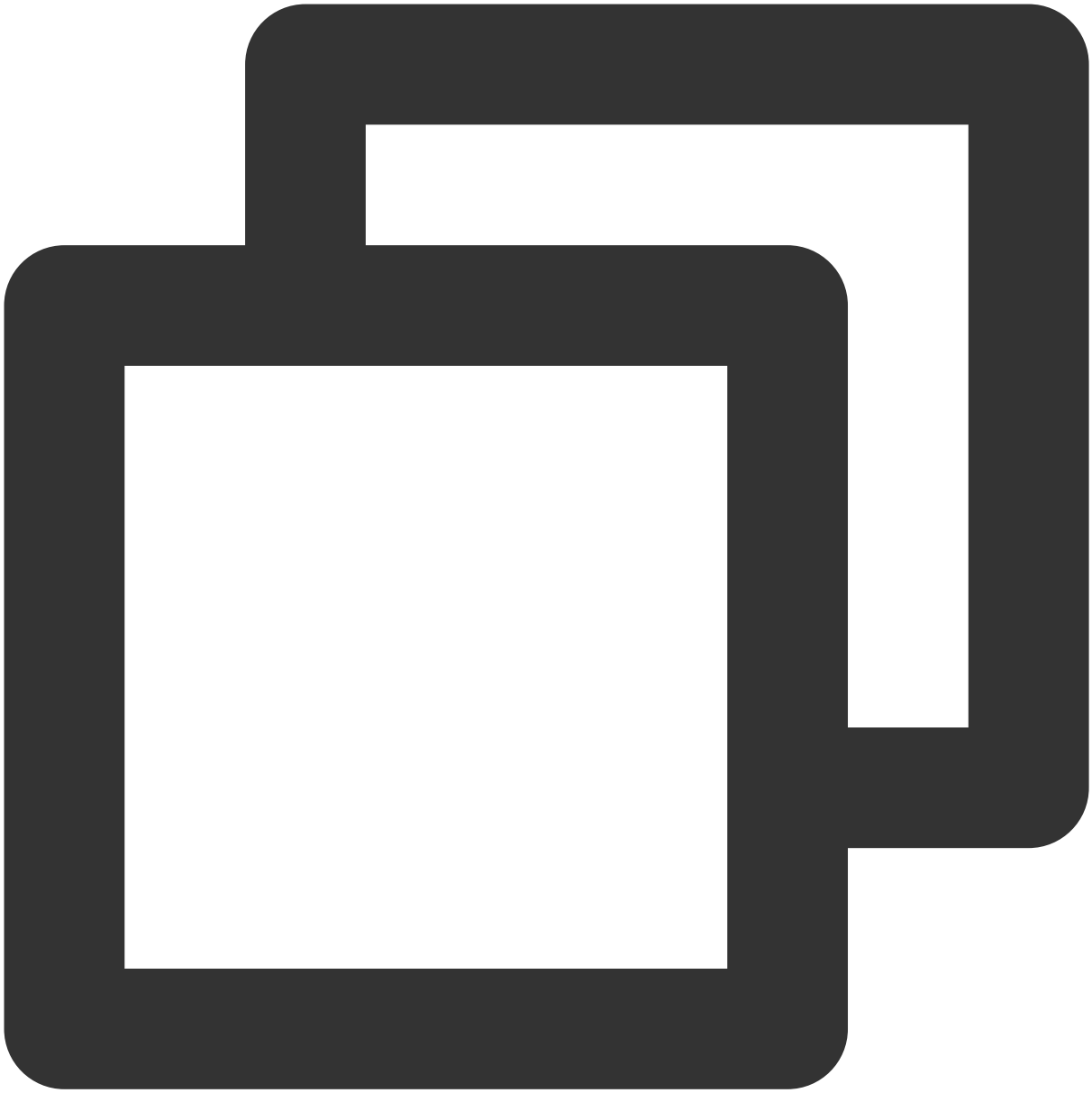
```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

注意：

这里是示例项目，为了方便使用 script 标签方式引入，您也可以参考 接入指南 中的方法，用 npm 包的方式引入。

步骤2：WebRTC 推流资源引入

在需要直播推流的页面（PC Web 端）中引入 js 脚本：



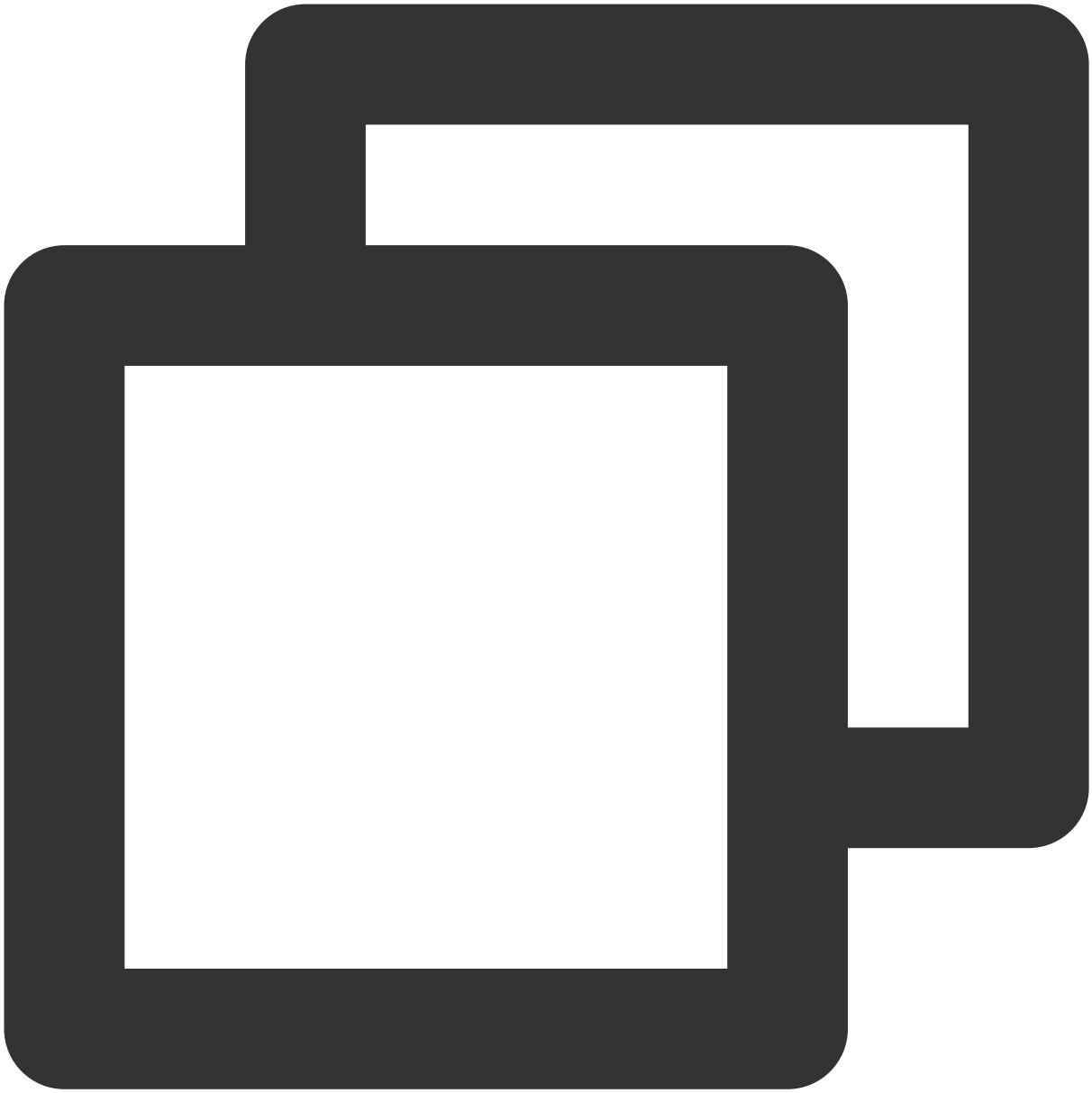
```
<script src="https://video.sdk.qcloudecdn.com/web/TXLivePusher-2.0.0.min.js" charse
```

注意：

请在 HTML 的 body 部分引入上述脚本，如果在 head 部分引入会报错。

步骤3：初始化 Web 美颜特效 SDK

示例代码如下：



```
const { ArSdk } = window.AR;

/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.tencentcloud.com/developer) 即可查看 APPID
 */
const APPID = ''; // 此处请填写您自己的参数
```

```
/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.tencentcloud.com/vcube/
 */
const LICENSE_KEY = ''; // 此处请填写您自己的参数

/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过
 * [签名方法](https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8
 */
const token = ''; // 此处请填写您自己的参数

/** ---------------------- */

/**
 * 定义获取签名方法
 *
 * 注意：此处方案仅适用于 DEMO 调试，正式环境中签名方法推荐在服务端实现，通过接口提供，前端调用拉
 * 如：
 * async function () {
 *  return fetch('http://xxx.com/get-ar-sign').then(res => res.json());
 * };
 */
const getSignature = function () {
    const timestamp = Math.round(new Date().getTime() / 1000);
    const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
    return { signature, timestamp };
};

let w = 720;
let h = 480;

// 获取输入流
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: { width: w, height: h }
})

// ar sdk 基础配置参数
const config = {
    input: stream,
    auth: {
        licenseKey: LICENSE_KEY,
        appId: APPID,
```

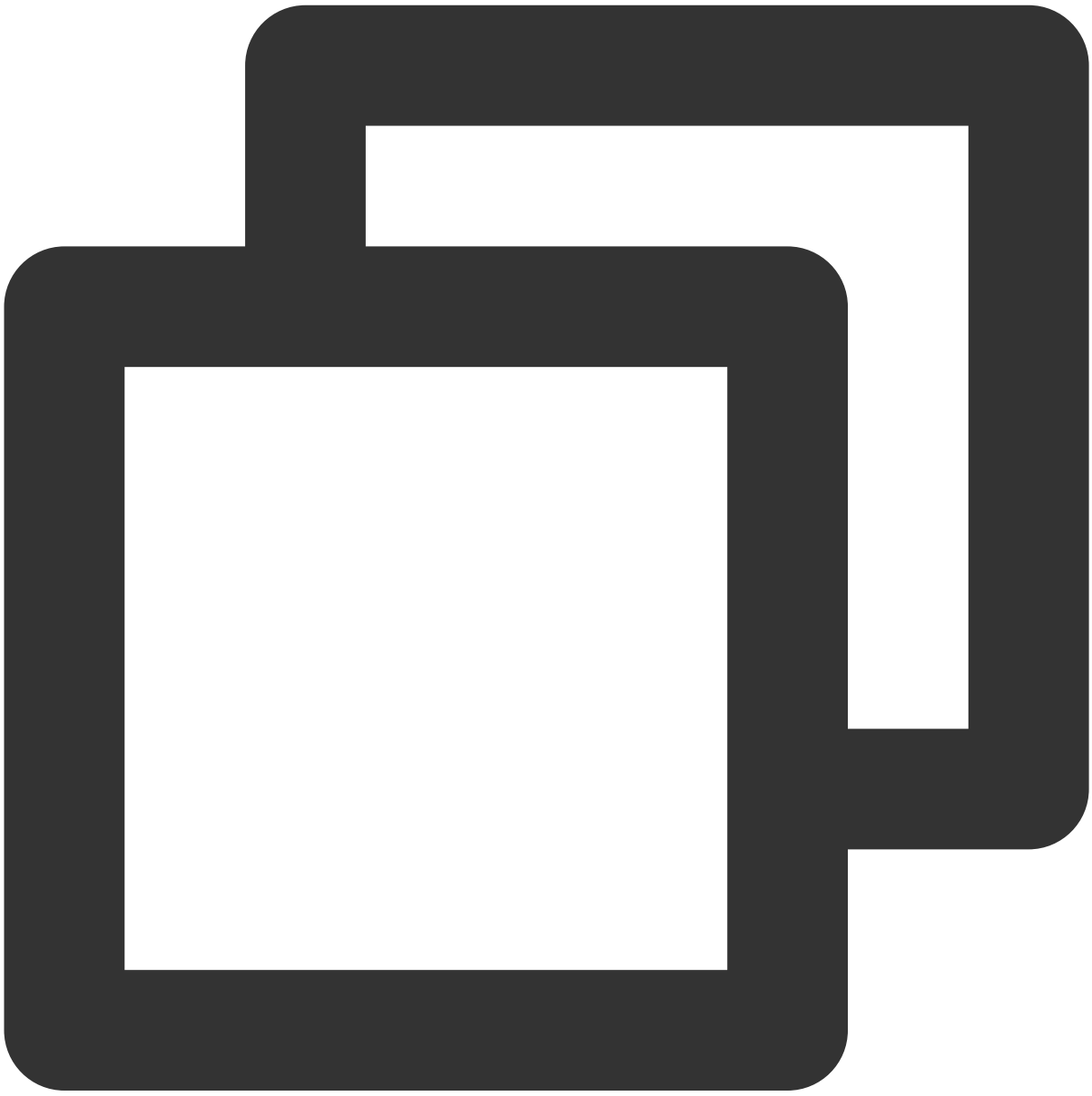
```
        authFunc: getSignature
    },
    // 初始美颜效果（可选参数）
    beautify: {
        whiten: 0.1, // 美白 0-1
        dermabrasion: 0.5, // 磨皮 0-1
        lift: 0.3, // 瘦脸 0-1
        shave: 0, // 削脸 0-1
        eye: 0, // 大眼 0-1
        chin: 0, // 下巴 0-1
    }
}

// config 传入 ar sdk
const ar = new ArSdk(config);

// created回调里可以获取内置特效与滤镜列表进行界面展示
ar.on('created', () => {
    // 获取内置美妆、贴纸
    ar.getEffectList({
        Type: 'Preset'
    }).then((res) => {
        const list = res.map(item => ({
            name: item.Name,
            id: item.EffectId,
            cover: item.CoverUrl,
            url: item.Url,
            label: item.Label,
            type: item.PresetType,
        }));

        const makeupList = list.filter(item=>item.label.indexOf('美妆')>=0)
        const stickerList = list.filter(item=>item.label.indexOf('贴纸')>=0)
        // 渲染美妆、贴纸列表视图
        renderMakeupList(makeupList);
        renderStickerList(stickerList);

    }).catch((e) => {
        console.log(e);
    });
    // 内置滤镜
    ar.getCommonFilter().then((res) => {
        const list = res.map(item => ({
            name: item.Name,
            id: item.EffectId,
            cover: item.CoverUrl,
            url: item.Url,
```

```
            label: item.Label,
            type: item.PresetType,
        }));

        // 渲染滤镜列表视图
        renderFilterList(list);

    }).catch((e) => {
        console.log(e);
    });
});

ar.on('ready', async (e) => {

    // 在 ready 回调里及该事件之后，可使用三种方法来设置美颜特效：setBeautify/setEffect/setF

    // 例如使用range input（滑动控件）设置美颜效果
    $('#dermabrasion_range_input').change((e) => {
        ar.setBeautify({
            dermabrasion: e.target.value, // 磨皮 0-1
        });
    });

    // 通过created回调中创建的美妆、贴纸列表交互设置效果(setEffect的输入参数支持三种格式，详见s
    $('#makeup_list li').click(() => {
        ar.setEffect([{id: effect.id, intensity: 1}]);
    });
    $('#sticker_list li').click(() => {
        ar.setEffect([{id: effect.id, intensity: 1}]);
    });

    // 通过created回调中创建的滤镜列表交互设置滤镜效果(setFilter第二个参数1表示强度，详见SDK接
    ar.setFilter(filterList[0].id, 1);
    $('#filter_list li').click(() => {
        ar.setFilter(filter.id, 1);
    });

    // 获取ar sdk 输出的流在下一步中进行 WebRTC 推流
    const arStream = await ar.getOutput();

});

ar.on('error', (e) => {
    console.log(e);
});
```

更细致的 UI 控制用法您可以通过下载文末提供的代码包来进一步查看。

步骤4：开始推流

完成上一步之后，在 SDK ready 回调中获取输出流即可进行 WebRTC 推流，示例代码如下：



```
let livePusher = new TXLivePusher()
// 设置直播推流基础参数 begin
let DOMAIN = '您的推流域名'
let AppName = '您的appName'
let StreamName = '您的streamName'
let txSecret = '您的txSecret'
```

```
let txTime = '您的txTime'
// 设置直播推流基础参数 end

let pushUrl = `webrtc://${DOMAIN}/${AppName}/${StreamName}?txSecret=${txSecret}&txT

// 可选：设置预览界面元素
livePusher.setRenderView('id_local_video');
// 捕获流内容
livePusher.startCustomCapture(arStream);
// 立刻开始推流，您也可以通过其他函数来控制推流时机
livePusher.startPush(pushUrl)
```

其中 txSecret 和 txTime 都需要计算，为了方便您也可以通过 **直播控制台** 的 **地址生成器** 快速生成这些参数和推流 URL，具体请参见 地址生成器。

推流（startPush）成功后，您应该就能看到应用了美颜特效的直播流的效果了。

步骤5：查看效果

注意：

示例项目需您自行启动本机 Web 服务，并保证通过端口号可访问到 HTML 文件。

若您有一个已备案成功的播放域名，可参考 直播播放 查看实际的播放效果。

若您无播放域名，可在**直播控制台**的 流管理 中预览当前流的画面。

代码示例

您可以下载 示例代码 解压后查看 `AR_LEB_WEB` 代码目录。

结合 WebRTC 的推流（预初始化方案）

最近更新时间：2024-07-09 16:06:24

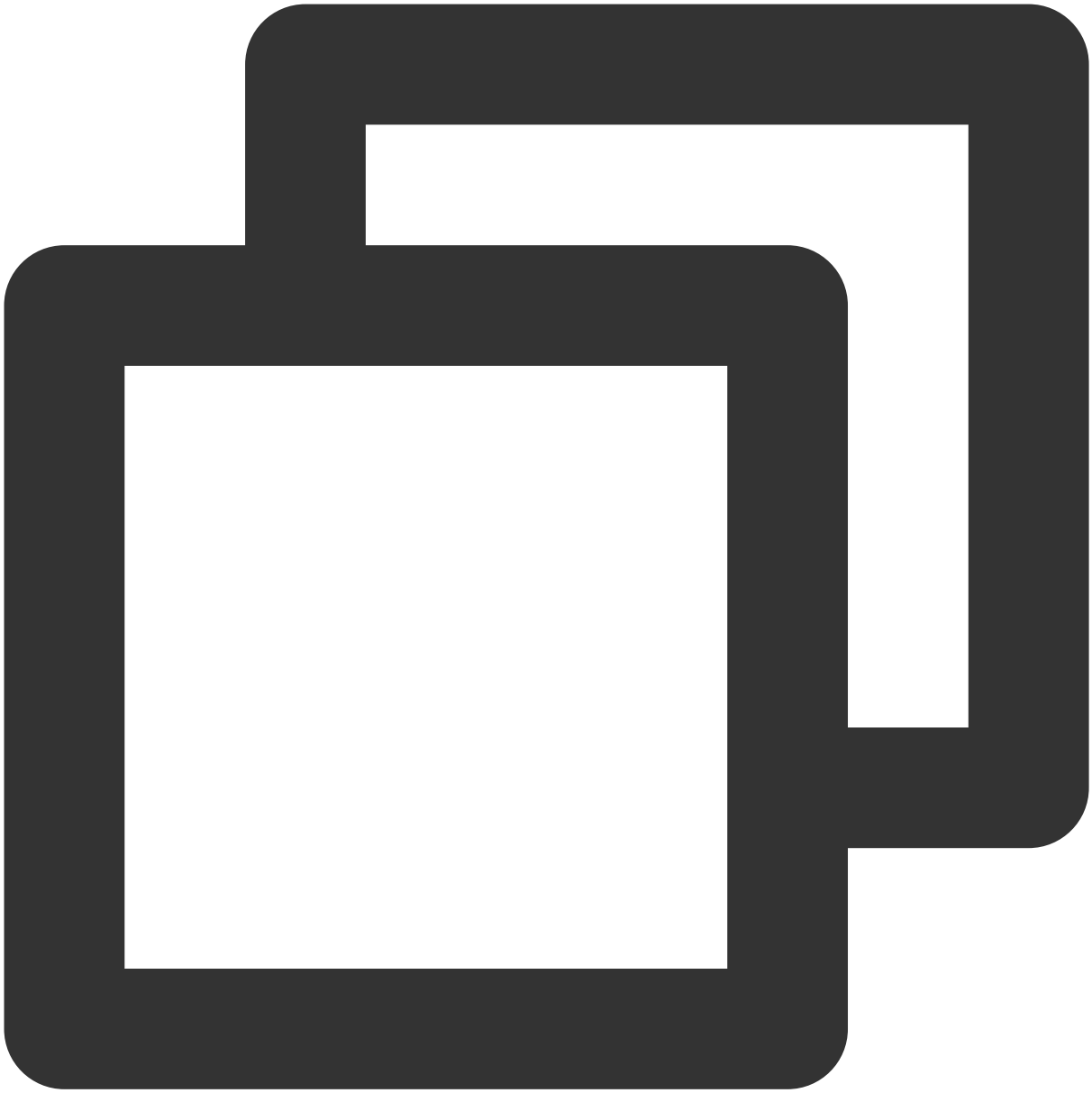
准备工作

请阅读 Web 美颜特效 SDK 接入指南，熟悉 SDK 基本用法。
WebRTC 的推流相关请参见 结合 WebRTC 的推流，本文重点介绍采取预初始方案时代码及流程差异部分。
预初始化方案相关介绍，请参见 加载优化。

开始使用

预初始化方案与常规加载方案相比，最大的差异在于初始化 SDK 时不需要指 input 或 camera 属性，即初始化时不为 SDK 指定输入数据，后续根据业务需求在适当的位置调用 initCore 接口指定输入数据。这样做的好处是将 SDK 依赖的资源提前加载，后续调用 initCore 后，SDK 的 ready 事件就会更快地触发，便于获取输出流展示等。以下为关键代码示例：

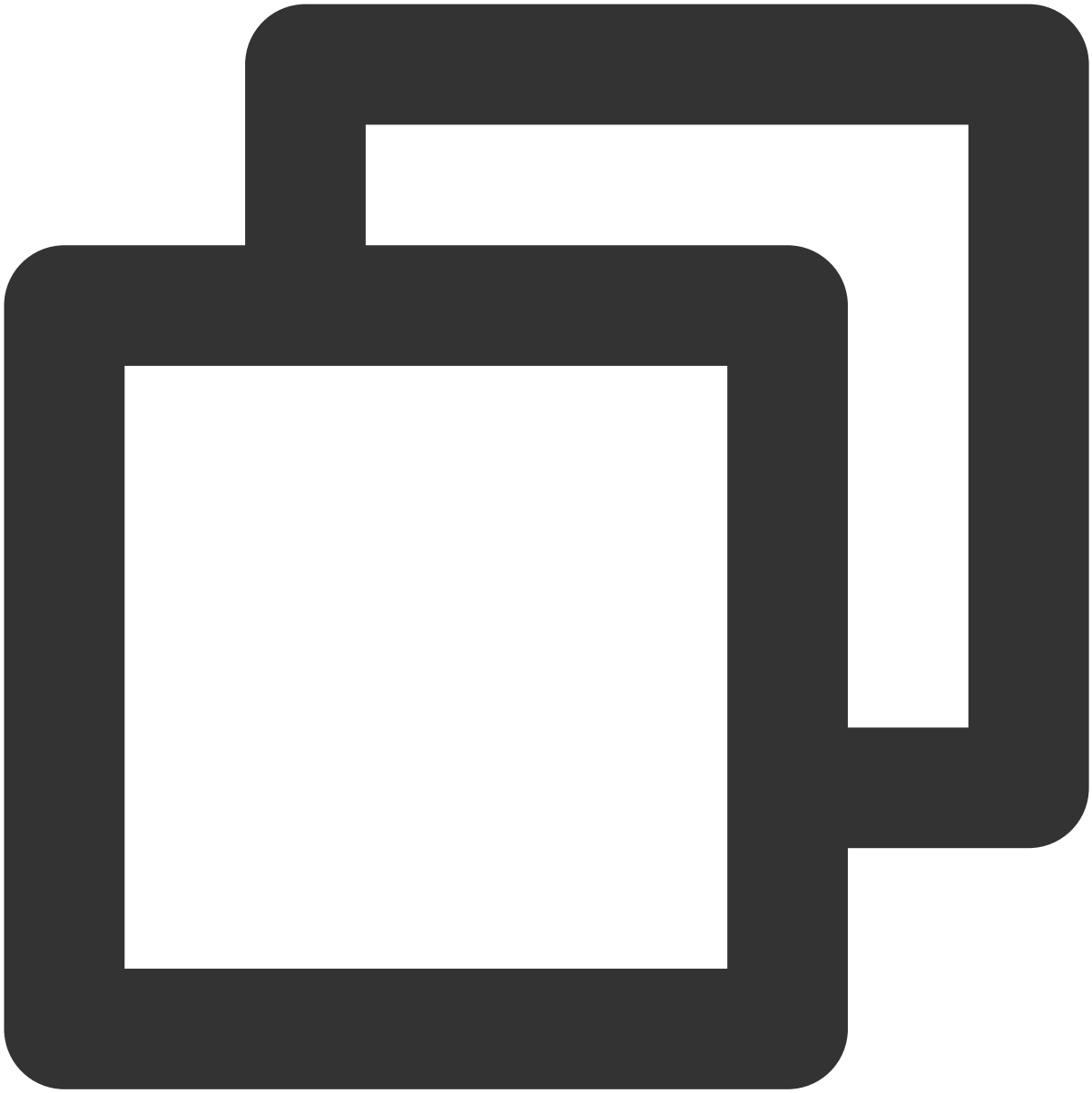
初始化SDK



```
...
let resourceReady = false
// ar sdk 基础配置参数
const config = {
    // input: stream, // 不指定input
    auth: {
        licenseKey: LICENSE_KEY,
        appId: APPID,
        authFunc: getSignature
    },
    // 初始美颜效果（可选参数）
```

```
    beautify: {
        whiten: 0.1, // 美白 0-1
        dermabrasion: 0.5, // 磨皮 0-1
        lift: 0.3, // 瘦脸 0-1
        shave: 0, // 削脸 0-1
        eye: 0, // 大眼 0-1
        chin: 0, // 下巴 0-1
    }
}

// config 传入 ar sdk
const ar = new ArSdk(config);
// resourceReady 回调事件触发，意味着相关资源已加载完成，等待 initCore 提供输入
ar.on('resourceReady', () => {
    resourceReady = true
})
// 调用 initCore 后会触发 ready 事件
ar.on('ready', () => {
    // 获取 ar sdk 输出流数据
    const arStream = await ar.getOutput();
    // 处理输出流
    ...
})
...
```

用户操作触发 initCore 事件



```
// 此处以用户点击【开启摄像头】为例，介绍预初始化方案设置输入流的方式
function onClickStartCamera(){
    let w = 1280;
    let h = 720;

    // 获取设备输入流
    const arInputStream = await navigator.mediaDevices.getUserMedia({
        audio: true,
        video: {
            width: w,
            height: h
```

```
            }
        });
        if(!resourceReady){ // 此模式下，resourceReady未触发时，调用initCore无意义，业务可以做
            return
        }
        // 设置 ar sdk 输入流数据
        ar.initCore({
            input: arInputStream
        })
    }
```

代码示例

您可以下载 示例代码 解压后查看 `AR_LEB_WEB` 代码目录中的 `AR_and_LEB_Preload.html` 文件。

结合 TRTC 推流

最近更新时间：2024-07-09 16:06:24

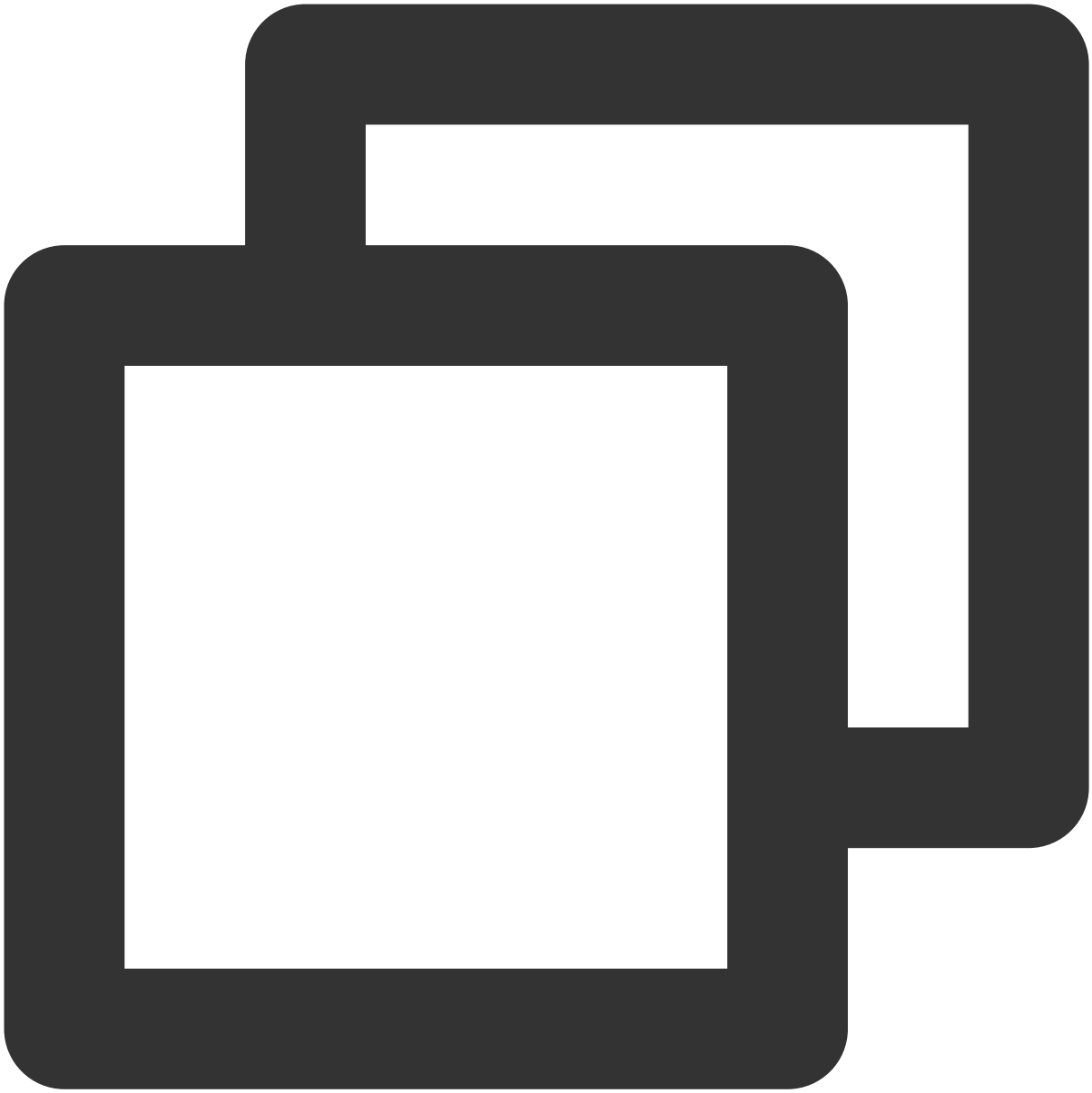
准备工作

请阅读 Web 美颜特效 SDK 接入指南，熟悉 SDK 基本用法。
请阅读 TRTC 快速集成(Web)，了解 TRTC Web SDK 基本用法，并完成基础设置。
请阅读 TRTC 快速跑通 Web Demo，并先尝试在本地项目中运行 TRTC Web Demo。

开始使用

步骤1：Web 美颜特效 SDK 引入

由于 TRTC 的 Web Demo 项目已经做的比较完善，我们在此基础上进行少量改造即可。
在页面（PC Web 端）中引入 js 脚本：



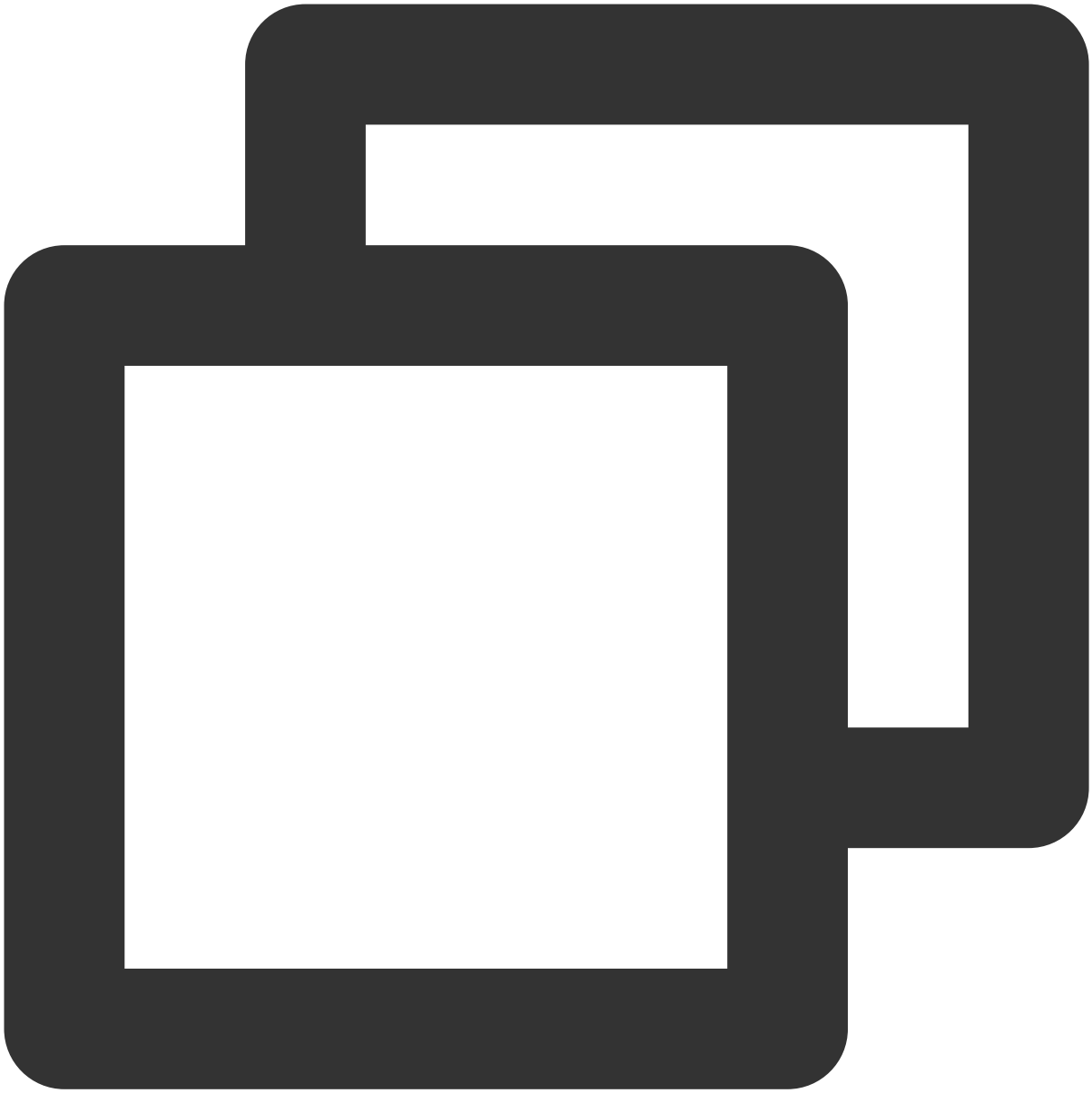
```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

注意：

这里是示例项目，为了方便使用 script 标签方式引入，您也可以参见 接入指南 中的方法，用 npm 包的方式引入。

步骤2：理解 TRTC 流初始化逻辑

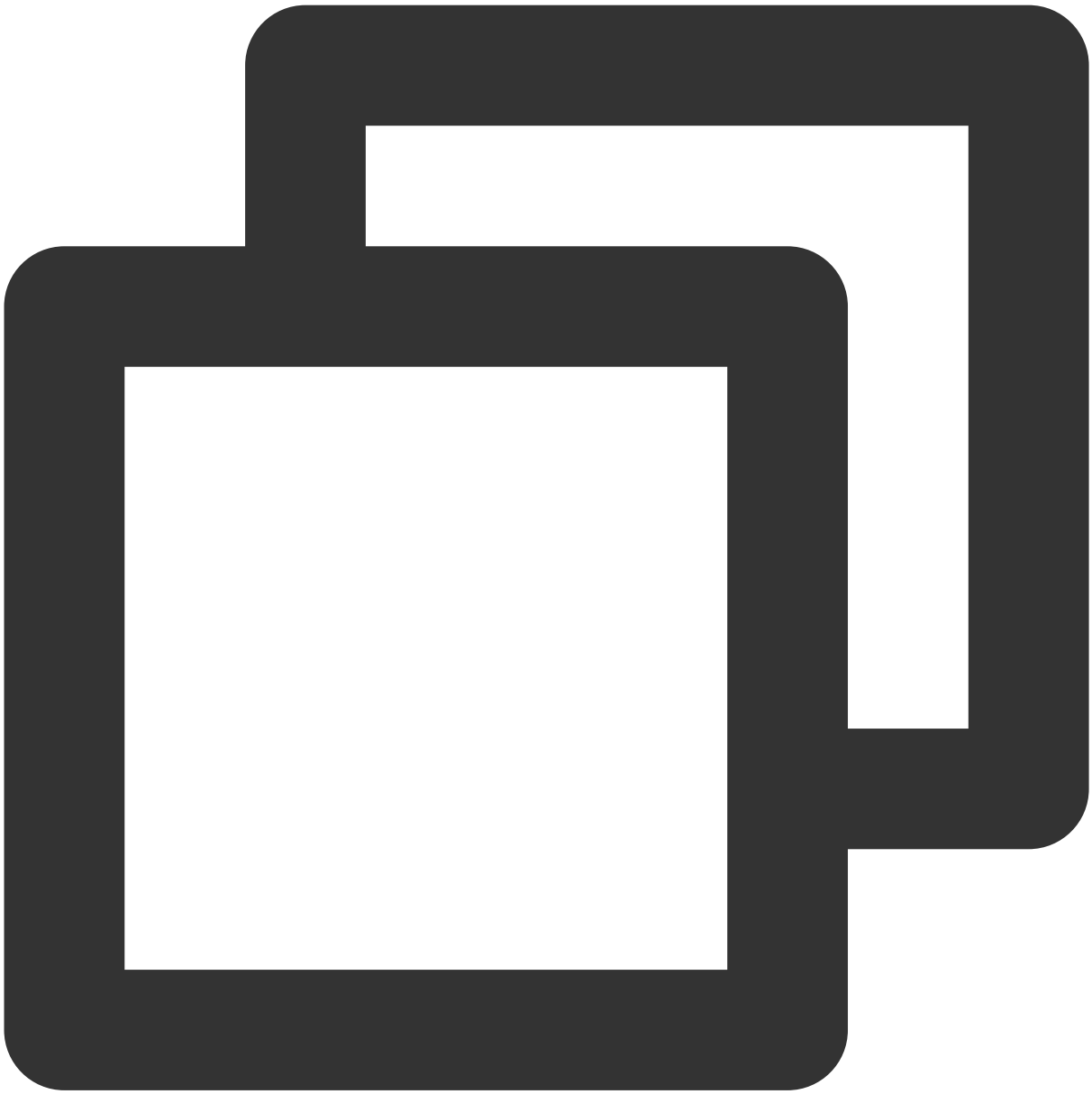
1. 在 TRTC 的 Demo 项目里，查看 TRTC 本地流的初始化过程，TRTC 通过 createStream 方法创建流对象，可以指定使用 TRTC SDK 的默认采集方式，如下：



```
// 从麦克风和摄像头采集本地音视频流
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
 console.log('initialize localStream success');
 // 本地流初始化成功，可通过Client.publish(localStream)发布本地音视频流
}).catch(error => {
 console.error('failed initialize localStream ' + error);
});
```

以上为最基本的采集本地音视频流的初始化方式。

2. 为了便于开发者对音视频流进行预处理，TRTC.createStream 也支持从外部音视频源创建本地流，通过这种方式创建本地流，开发者可以实现自定义处理（例如对视频进行美颜处理），以下为示例代码：



```
// 获取自定义的stream
const stream = await this.ar.getOutput();

// 从指定的音视频源创建本地音视频流
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];
const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSourc
localStream.initialize().then(() => {
```

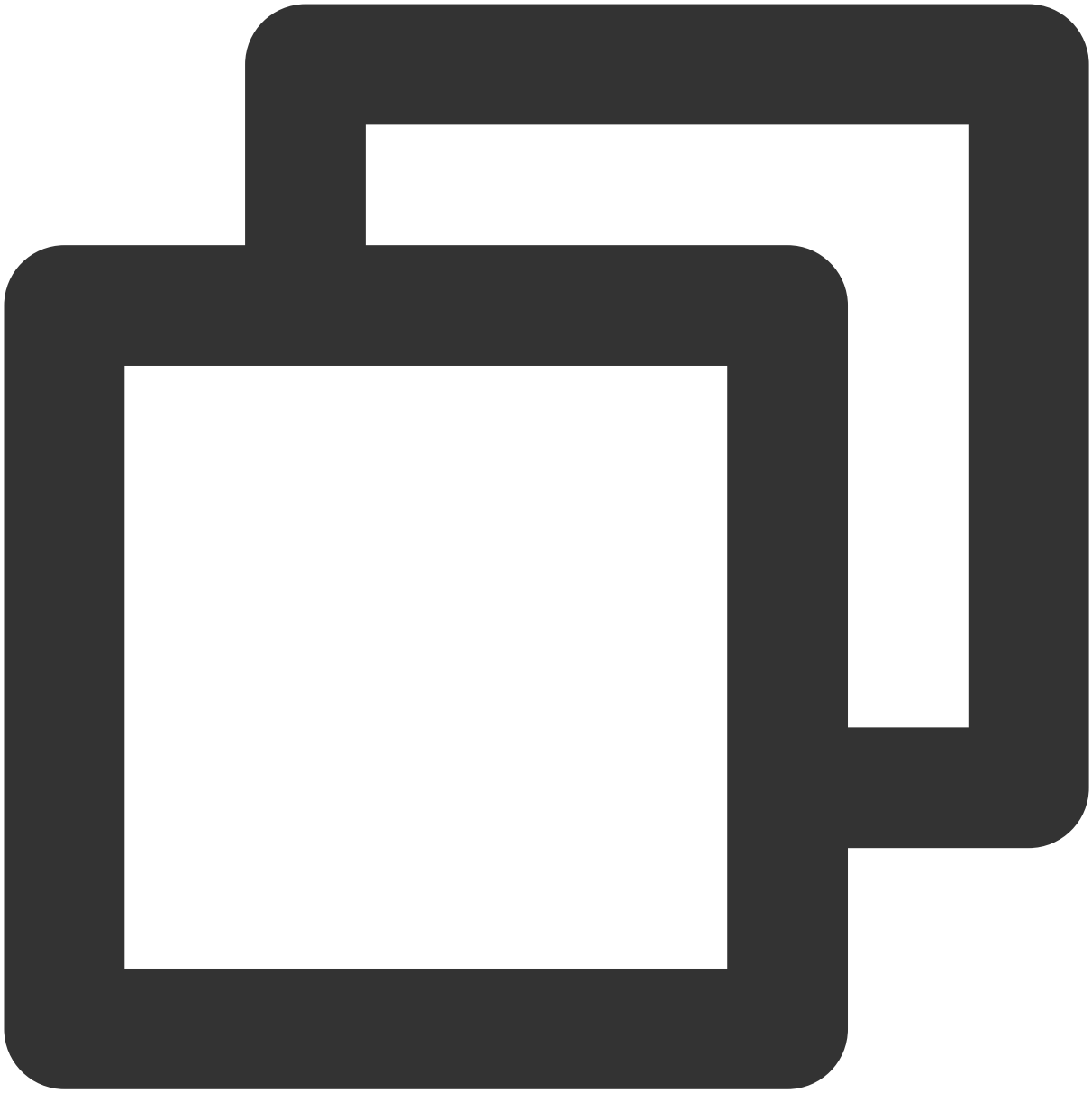
```
  console.log('initialize localStream success');
  // 本地流初始化成功，可通过Client.publish(localStream)发布本地音视频流
}).catch(error => {
  console.error('failed initialize localStream ' + error);
});
```

createStream 方法的使用详情可见 TRTC SDK 文档。

3. 为了获取经过美颜处理的自定义流（即上面的 getMyStream 方法），我们需要先 初始化 Web 美颜特效 SDK。

步骤3：初始化 Web 美颜特效 SDK

示例代码如下：



```
const { ArSdk } = window.AR

/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.tencentcloud.com/developer) 即可查看 APPID
 */
const APPID = ''; // 此处请填写您自己的参数
```

```
/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.tencentcloud.com/vcube/
 */
const LICENSE_KEY = ''; // 此处请填写您自己的参数

/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过
 * [签名方法](https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8
 */
const token = ''; // 此处请填写您自己的参数

/** ---------------------- */

/**
 * 定义获取签名方法
 *
 * 注意：此处方案仅适用于 DEMO 调试，正式环境中签名方法推荐在服务端实现，通过接口提供，前端调用拉
 * 如：
 * async function () {
 *  return fetch('http://xxx.com/get-ar-sign').then(res => res.json());
 * };
 */
const getSignature = function () {
    const timestamp = Math.round(new Date().getTime() / 1000);
    const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
    return { signature, timestamp };
};

let w = 1280;
let h = 720;

// ar sdk 基础配置参数
const config = {
    camera: { //这个参数表示从本机摄像头开始采集流
        width: 1280,
        height:720
    },
    auth: {
        licenseKey: LICENSE_KEY,
        appId: APPID,
        authFunc: getSignature
    },
    // 初始美颜效果（可选参数）
```

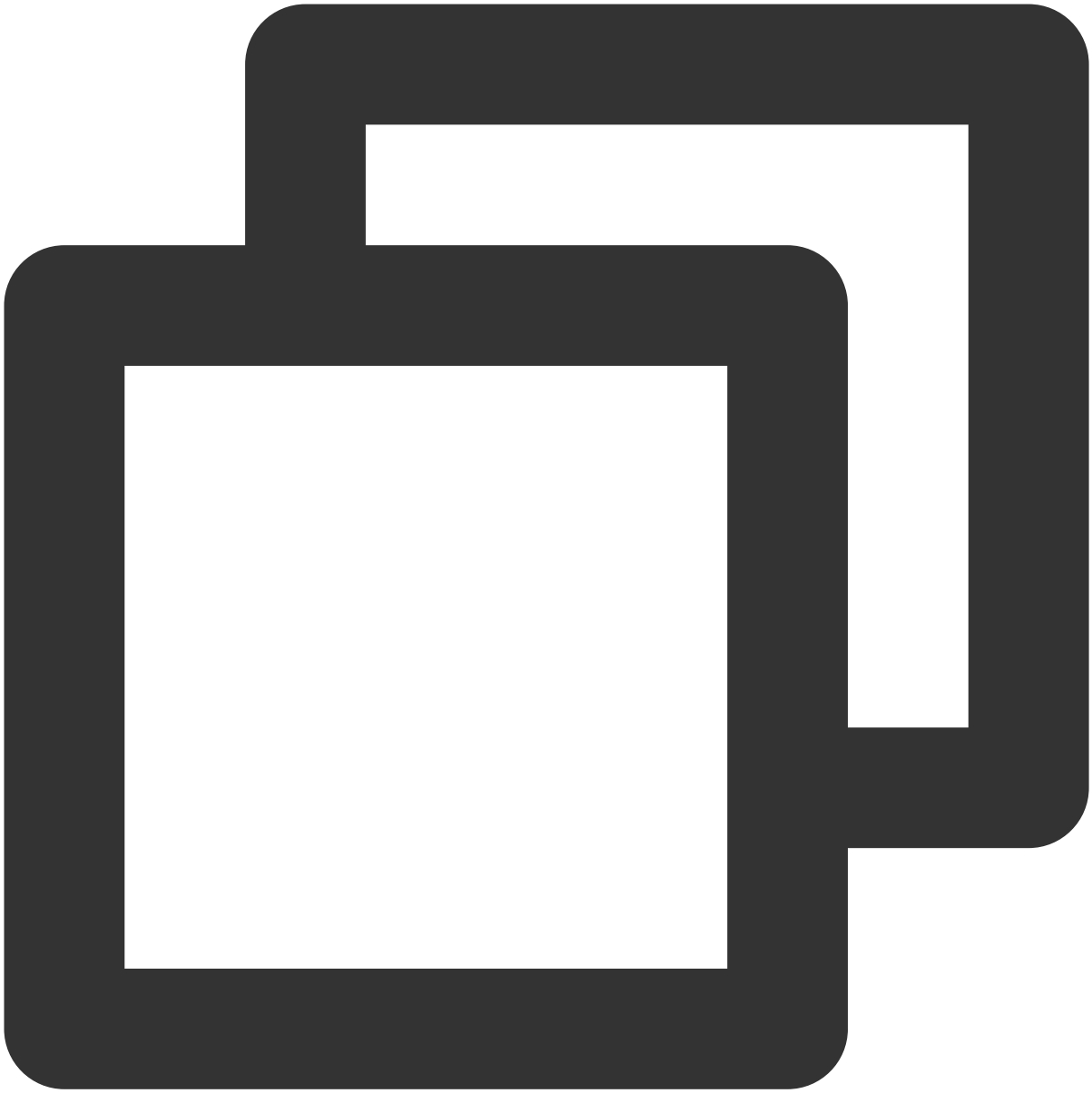
```
    beautify: {
        whiten: 0.1, // 美白 0-1
        dermabrasion: 0.5, // 磨皮 0-1
        lift: 0.3, // 瘦脸 0-1
        shave: 0, // 削脸 0-1
        eye: 0, // 大眼 0-1
        chin: 0, // 下巴 0-1
    }
}

// config 传入 ar sdk
const ar = new ArSdk(config);

// created回调里可以获取内置特效与滤镜列表进行界面展示
ar.on('created', () => {
    // 获取内置美妆、贴纸
    ar.getEffectList({
        Type: 'Preset'
    }).then((res) => {
        const list = res.map(item => ({
            name: item.Name,
            id: item.EffectId,
            cover: item.CoverUrl,
            url: item.Url,
            label: item.Label,
            type: item.PresetType,
        }));
        const makeupList = list.filter(item=>item.label.indexOf('美妆')>=0)
        const stickerList = list.filter(item=>item.label.indexOf('贴纸')>=0)
        // 渲染美妆、贴纸列表视图
        renderMakeupList(makeupList);
        renderStickerList(stickerList);

    }).catch((e) => {
        console.log(e);
    });
    // 内置滤镜
    ar.getCommonFilter().then((res) => {
        const list = res.map(item => ({
            name: item.Name,
            id: item.EffectId,
            cover: item.CoverUrl,
            url: item.Url,
            label: item.Label,
            type: item.PresetType,
        }));
```

```
        // 渲染滤镜列表视图
        renderFilterList(list);

    }).catch((e) => {
        console.log(e);
    });
});

ar.on('ready', (e) => {

    // 在 ready 回调里及该事件之后，可使用三种方法来设置美颜特效：setBeautify/setEffect/setF

    //  例如使用range input（滑动控件）设置美颜效果，例如
    $('#dermabrasion_range_input').change((e) => {
        ar.setBeautify({
            dermabrasion: e.target.value, // 磨皮 0-1
        });
    });

    // 通过created回调中创建的美妆、贴纸列表交互设置效果(setEffect的输入参数支持三种格式，详见s
    $('#makeup_list li').click(() => {
        ar.setEffect([{id: effect.id, intensity: 1}]);
    });
    $('#sticker_list li').click(() => {
        ar.setEffect([{id: effect.id, intensity: 1}]);
    });

    // 通过created回调中创建的滤镜列表交互设置滤镜效果(setFilter第二个参数1表示强度，详见SDK接
    ar.setFilter(filterList[0].id, 1);
    $('#filter_list li').click(() => {
        ar.setFilter(filter.id, 1);
    });
});

ar.on('error', (e) => {
    console.log(e);
});
```

上述代码对 Web 美颜特效 SDK 进行了初始化配置，且在 ready 事件内进行了一些简单的美颜设置交互处理，更细致的 UI 交互可以通过下载文末提供的代码包来进一步查看。

步骤4：修改 TRTC stream 初始化过程

Web 美颜特效 SDK 初始化完成后就可以使用 `getOutput` 方法获取输出的流，再根据 步骤2 的介绍完成 TRTC 流初始化，示例代码如下：



```
// 获取 ar sdk 输出流
const arStream = await this.ar.getOutput();

const audioSource = arStream.getAudioTracks()[0];
const videoSource = arStream.getVideoTracks()[0];

// create a local stream with audio/video from custom source
this.localStream_ = TRTC.createStream({
    audioSource,
    videoSource
});
```

步骤5：查看效果

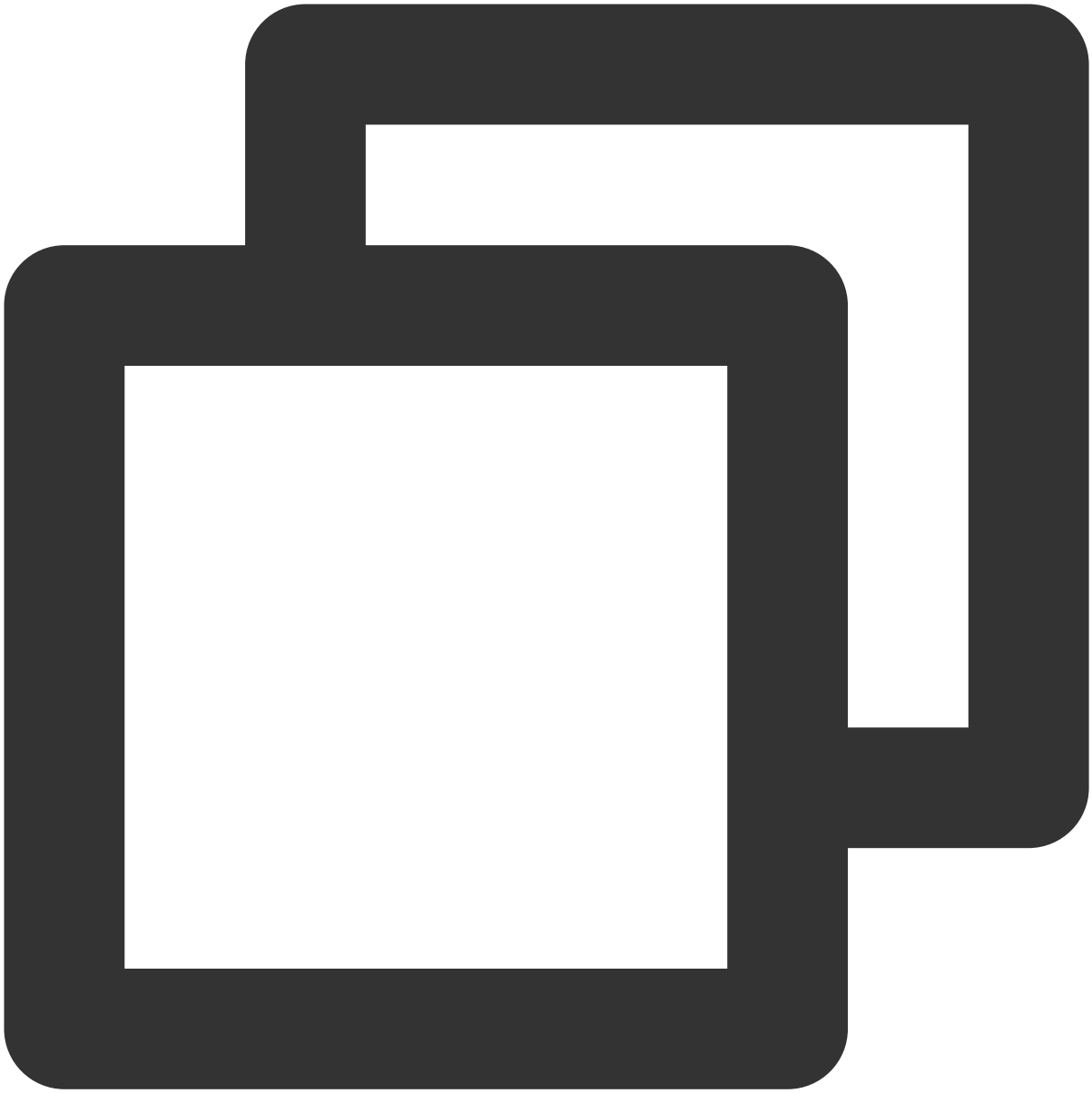
注意：

示例项目需您自行启动本机 Web 服务，并保证通过指定端口号可访问到 HTML 文件。

您在进入房间后可以很快查看实际的播放效果（运行示例代码中的 `index_AR.html` ），成功后可以新开浏览器标签页进入刚才创建的房间模拟其他人加入房间的效果。

步骤6：查看设备控制能力

在输入类型为 camera 的情况下，Web 美颜特效 SDK 还提供了一些设备控制能力，示例代码如下：

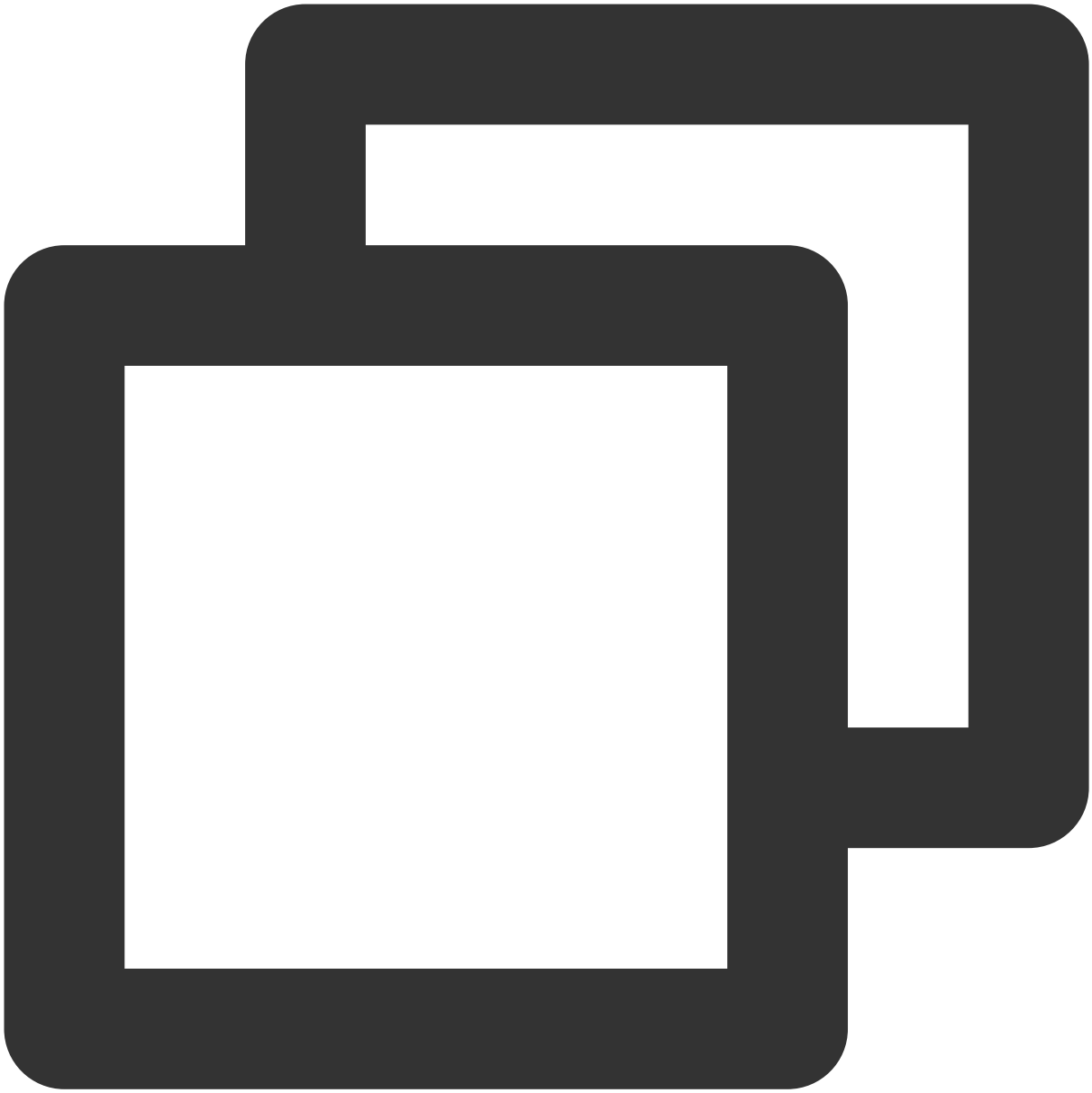


```
const cameraApi = this.ar.camera;
// 支持获取设备列表
const devices = await cameraApi.getDevices()
console.log(devices)
// 支持切换不同摄像头
// await cameraApi.switchDevice('video', 'your-video-device-id')
// 禁用视频轨道
// cameraApi.muteVideo()
// 恢复视频轨道
// cameraApi.unmuteVideo()
// 禁用音频轨道
```

```
// cameraApi.muteAudio()
// 恢复音频轨道
// cameraApi.unmuteAudio()
// 停止摄像头
// cameraApi.stop()
// 重启摄像头
// await cameraApi.restart()
```

步骤7：查看本地预览能力

在输入类型为 camera 的情况下，Web 美颜特效 SDK 还提供了本地预览播放的能力，示例代码如下：



```
// 使用SDK内置player，其中my-dom-id表示您需要放置播放器的容器id
const player = await sdk.initLocalPlayer('my-dom-id')
// 直接播放
await player.play()
// 暂停
player.pause()
```

示例代码

您可以下载 示例代码 解压后查看，主要改动部分在 `index_AR.html` 和 `rtc-client-with-webar.js`，美颜特效的交互逻辑代码在 `base-js/js/ar_interact.js`，您的 TRTC 密钥信息配置在 `base-js/js/debug/GenerateTestUserSig.js`。

美颜特效小程序拍摄实践

最近更新时间：2023-11-06 15:37:27

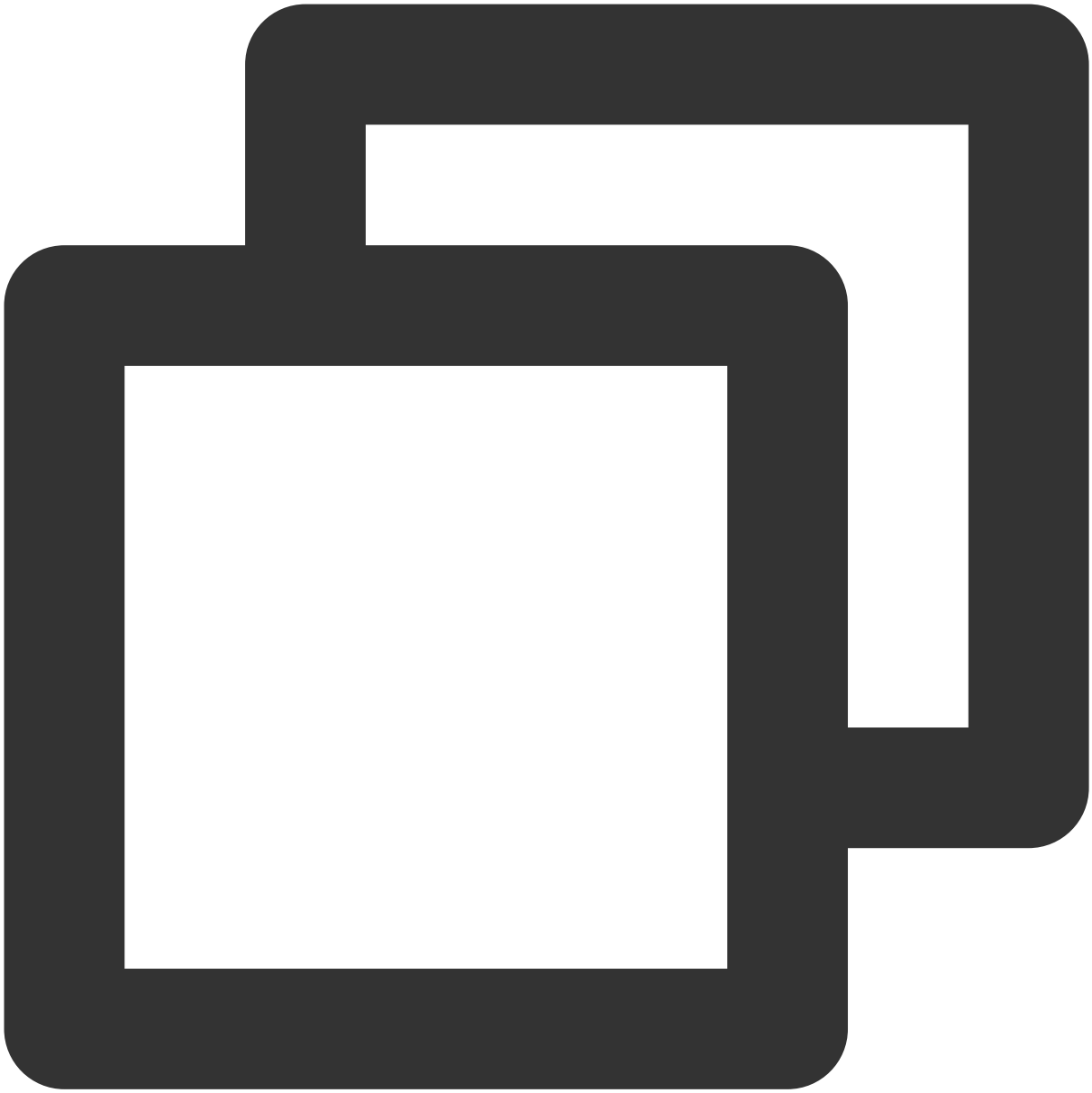
准备工作

小程序开发入门请参见 微信小程序文档。
请阅读 Web 美颜特效 SDK 接入指南，熟悉 SDK 基本用法。

开始使用

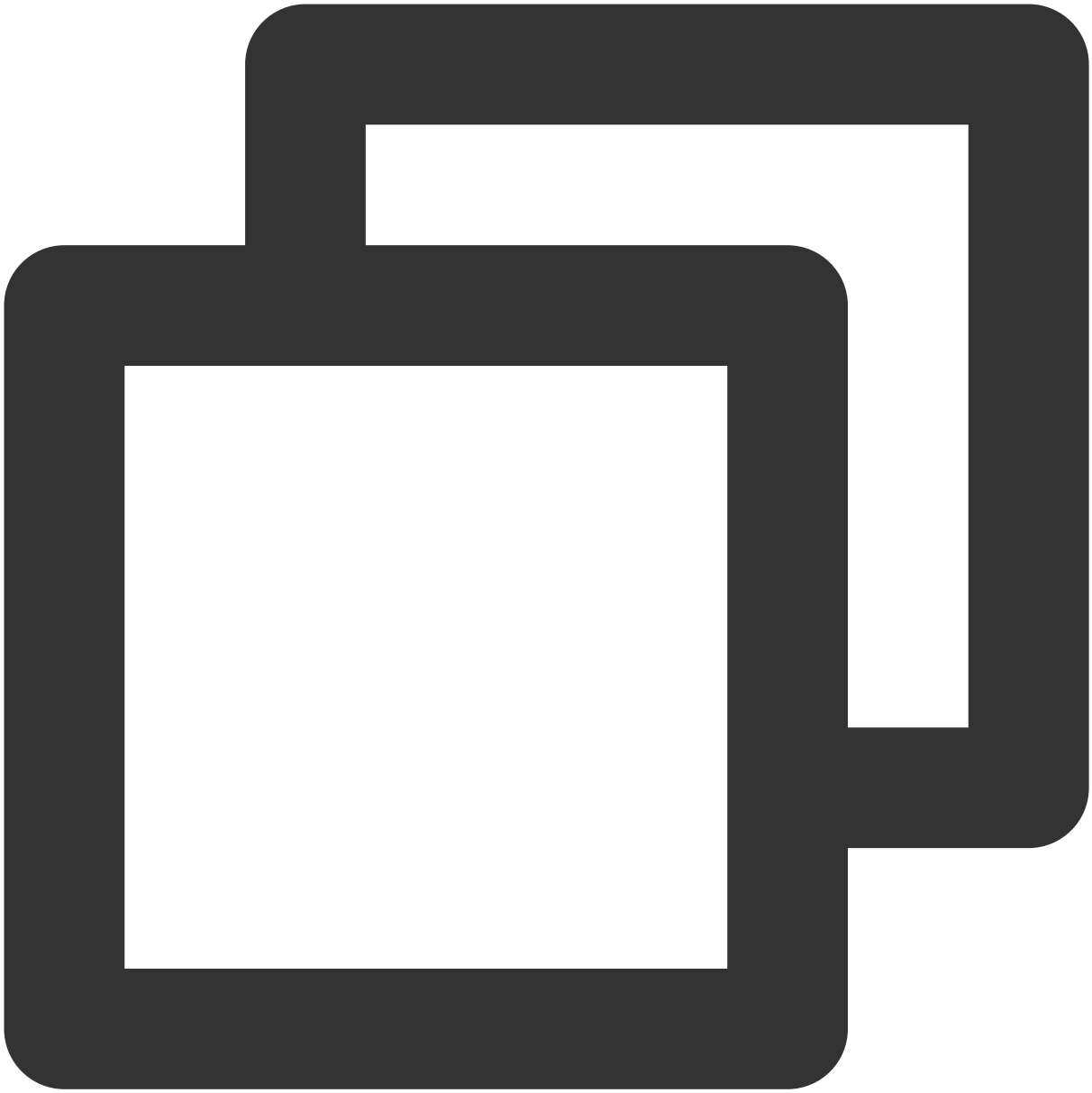
步骤1：小程序后台配置域名白名单

SDK 内部会请求后台进行鉴权和资源加载，因此小程序创建完后，需要在小程序后台配置域名白名单。
1. 打开 小程序后台，进入 **开发 > 开发管理 > 开发设置 > 服务器域名**。
2. 单击**修改**，配置以下域名并保存。
请求域名：



```
https://webar.qcloud.com;
https://webar-static.tencent-cloud.com;
https://aegis.qq.com;
以及鉴权签名接口（get-ar-sign）的地址
```

downloadFile 域名：

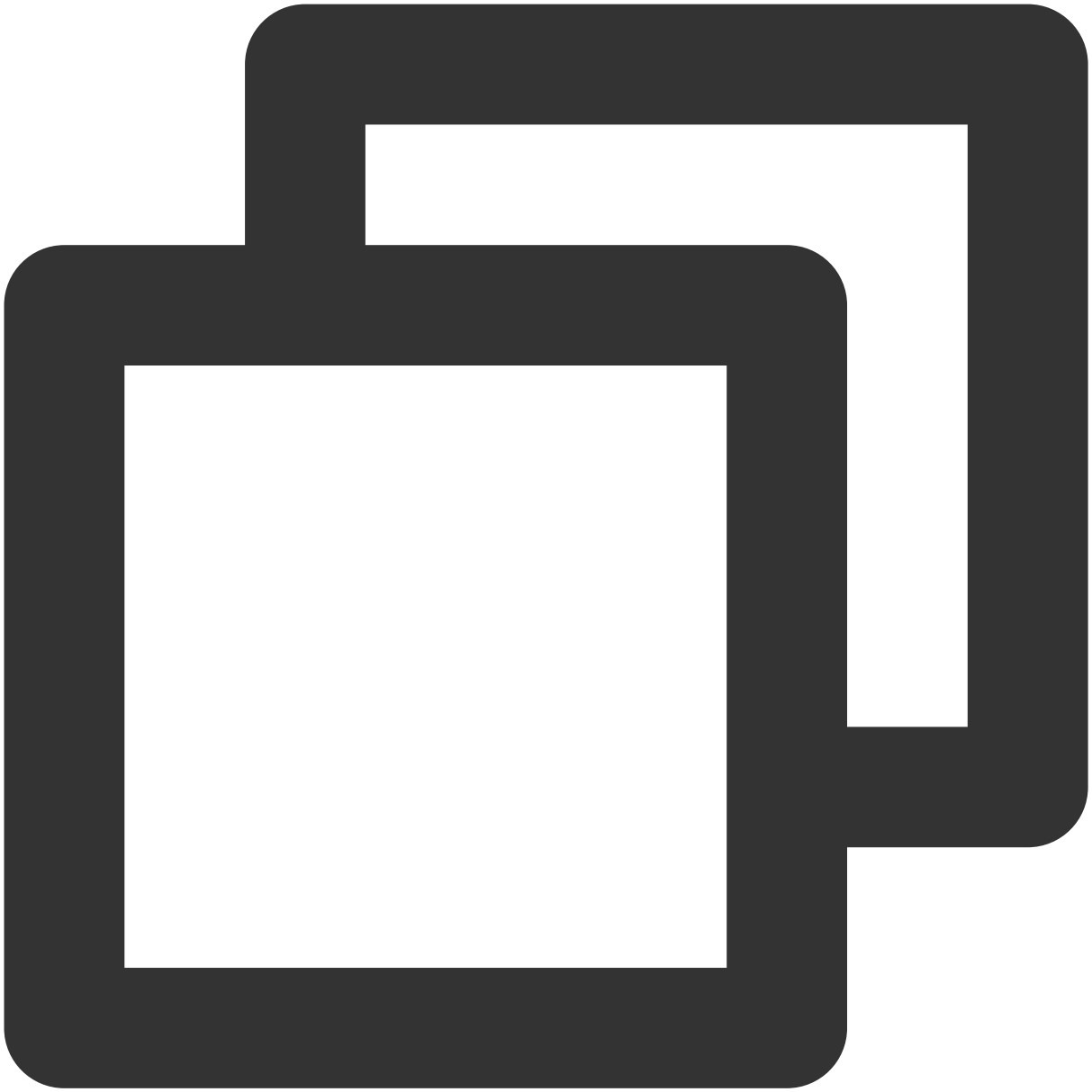


```
https://webar-static.tencent-cloud.com
```

步骤2：安装并构建 npm

小程序 npm 相关请参见 小程序使用 npm。

1. 安装：

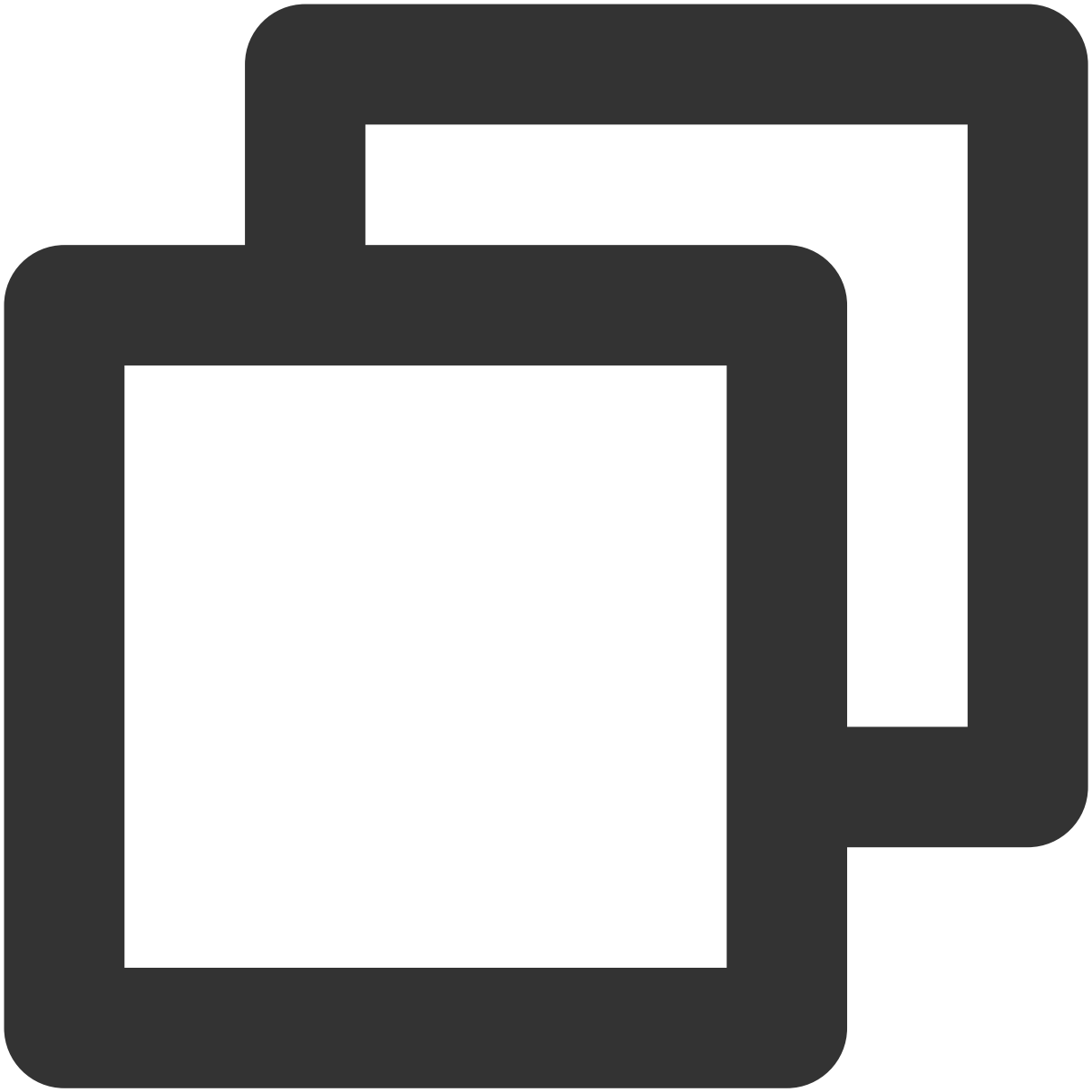


```
npm install tencentcloud-webar
```

2. 构建：

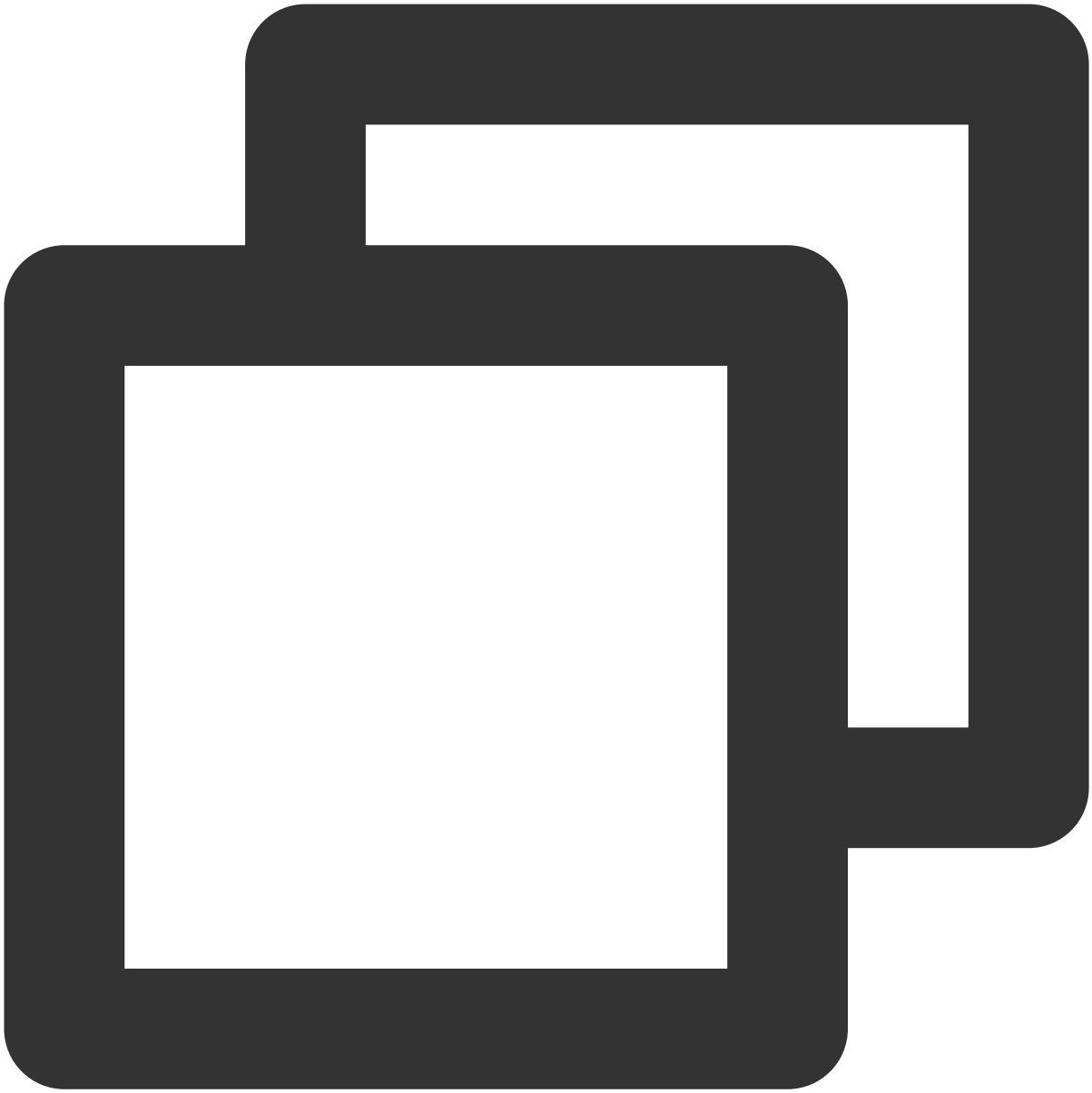
打开小程序开发者工具，顶部菜单选择 **工具 > 构建 npm**。

3. 在 app.json 中配置 workers 路径：



```
"workers": "miniprogram_npm/tencentcloud-webar/worker"
```

步骤3：引入文件



```
// 0.3.0之前版本引用方式（1个文件）
// import "../../miniprogram_npm/tencentcloud-webar/lib.js";
// 0.3.0及之后版本引用方式（2个文件 + 按需初始化3d模块）
import '../../miniprogram_npm/tencentcloud-webar/lib.js';
import '../../miniprogram_npm/tencentcloud-webar/core.js';
// 按需初始化3d插件，如果不需要3d则可以不引用
import '../../miniprogram_npm/tencentcloud-webar/lib-3d.js';
import { plugin3d } from '../../miniprogram_npm/tencentcloud-webar/plugin-3d'
// 导入 ArSdk
import { ArSdk } from "../../miniprogram_npm/tencentcloud-webar/index.js";
```

注意

小程序有单文件不超过 500kb 的限制，因此 SDK 分为两个 js 文件提供。

0.3.0版本之后，对 SDK 进行了进一步的拆分，新增3D支持，针对3D模块提供按需加载方式，导入前请确认当前使用的 SDK 版本信息，选择对应的导入方式。

步骤4：初始化 SDK

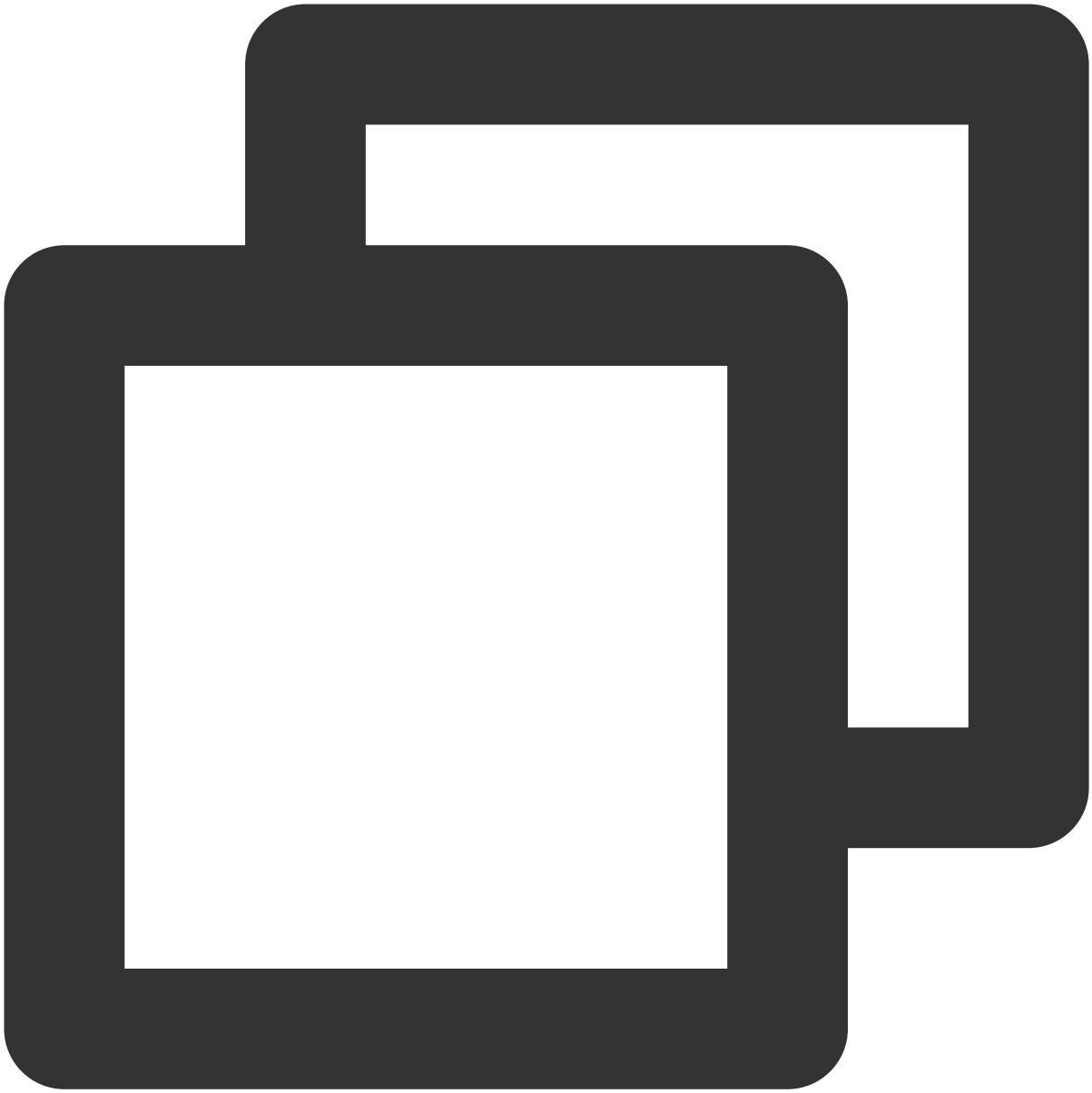
注意

小程序初始化 SDK 前须在控制台配置小程序 APPID，请参见 快速上手。

需在页面中插入 camera 标签来打开相机，然后设置 camera 参数，参数配置详情请参见 接入指南。

小程序不支持 getOutput，需要自行传入一个在屏的 webgl canvas，SDK 直接输出画面到此 canvas 上。

示例代码如下：



```
// wxml
//打开相机，通过position使相机不展示
<camera
  device-position="{{'front'}}"
  frame-size="large" flash="off" resolution="medium"
  style="width: 750rpx; height: 134rpx;position:absolute;top:-9999px;"
/>
//sdk 将处理完的画面实时输出到此 canvas 上
<canvas
  type="webgl"
  canvas-id="main-canvas"
```

```
  id="main-canvas"
  style="width: 750rpx; height: 1334rpx;">
</canvas>
//拍照将 ImageData 对象绘制到此 canvas 上
<canvas
  type="2d"
  canvas-id="photo-canvas"
  id="photo-canvas"
  style="position:absolute;width:720px;height:1280px;top:-9999px;left:-9999px;">
</canvas>
// js
/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APPID = ''; // 此处请填写您自己的参数

/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube
 */
const LICENSE_KEY = ''; // 此处请填写您自己的参数

/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过
 * [签名方法](https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8I
 */
const token = ''; // 此处请填写您自己的参数

Component({
    data: {
        makeupList: [],
        stickerList: [],
        filterList: [],
        recording: false
    },
    methods: {
        async getCanvasNode(id) {
            return new Promise((resolve) => {
                this.createSelectorQuery()
                    .select(`#${id}`)
```

```
                .node()
                .exec((res) => {
                    const canvasNode = res[0].node;
                    resolve(canvasNode);
                });
        });
    },
    getSignature() {
        const timestamp = Math.round(new Date().getTime() / 1000);
        const signature = sha256(timestamp + token + APPID + timestamp).toUpper
        return { signature, timestamp };
    },
    // 初始化相机类型
    async initSdkCamera() {
        // 获取在屏的 canvas，sdk 会将处理完的画面实时输出到 canvas 上
        const outputCanvas = await this.getCanvasNode("main-canvas");
        // 获取鉴权参数
        const auth = {
            licenseKey: LICENSE_KEY,
            appId: APP_ID,
            authFunc: this.getSignature
        };
        // 构造SDK初始化参数
        const config = {
            auth,
            camera: {
                width:720,
                height:1280,
            },
            output: outputCanvas,
            // 初始美颜效果（可选）
            beautify: {
                whiten: 0.1, // 美白 0-1
                dermabrasion: 0.3, // 磨皮 0-1
                lift: 0, // 瘦脸 0-1
                shave: 0, // 削脸 0-1
                eye: 0.2, // 大眼 0-1
                chin: 0, // 下巴 0-1
            }
        };
        const ar = new ArSdk(config);
        // created回调里可以开始获取内置特效与滤镜列表
        ar.on('created', () => {
            // 获取内置美妆、贴纸列表
            ar.getEffectList({
                Type: 'Preset'
            }).then((res) => {
```

```
                const list = res.map(item => ({
                    name: item.Name,
                    id: item.EffectId,
                    cover: item.CoverUrl,
                    url: item.Url,
                    label: item.Label,
                    type: item.PresetType,
                }));
                const makeupList = list.filter(item=>item.label.indexOf('美妆')>
                const stickerList = list.filter(item=>item.label.indexOf('贴纸')
                // 渲染特效列表
                this.setData({
                    makeupList,
                    stickerList
                });
            }).catch((e) => {
                console.log(e);
            });
            // 内置滤镜
            ar.getCommonFilter().then((res) => {
                const list = res.map(item => ({
                    name: item.Name,
                    id: item.EffectId,
                    cover: item.CoverUrl,
                    url: item.Url,
                    label: item.Label,
                    type: item.PresetType,
                }));
                // 渲染滤镜列表
                this.setData({
                    filterList: list
                });
            }).catch((e) => {
                console.log(e);
            });
        });
        // ready 回调中可以设置美颜、滤镜、特效效果
        ar.on('ready', (e) => {
            this._sdkReady = true
        });

        ar.on('error', (e) => {
            console.log(e);
        });

        this.ar = ar
    },
```

```
        // 改变美颜参数，需要确保sdk ready
        onChangeBeauty(val){
            if(!this._sdkReady) return
            // 可以通过setBeautify设置美颜效果，支持6种属性，详见SDK接入指南
            this.ar.setBeautify({
                dermabrasion: val.dermabrasion, // 磨皮 0-1
            });
        },
        // 改变美妆，需要确保sdk ready
        onChangeMakeup(id, intensity){
            if(!this._sdkReady) return
            // 使用setEffect设置特效，setEffect的输入参数支持三种格式，详见SDK接入指南
            this.ar.setEffect([{id, intensity}]);
        },
        // 改变贴纸，需要确保sdk ready
        onChangeSticker(id, intensity){
            if(!this._sdkReady) return
            // 使用setEffect设置特效，setEffect的输入参数支持三种格式，详见SDK接入指南
            this.ar.setEffect([{id, intensity}]);
        },
        // 改变滤镜，需要确保sdk ready
        onChangeFilter(id, intensity){
            if(!this._sdkReady) return
            // 使用setFilter设置滤镜，第二个参数表示滤镜强度（范围0-1）
            this.ar.setFilter(id, 1);
        }
    }
})
```

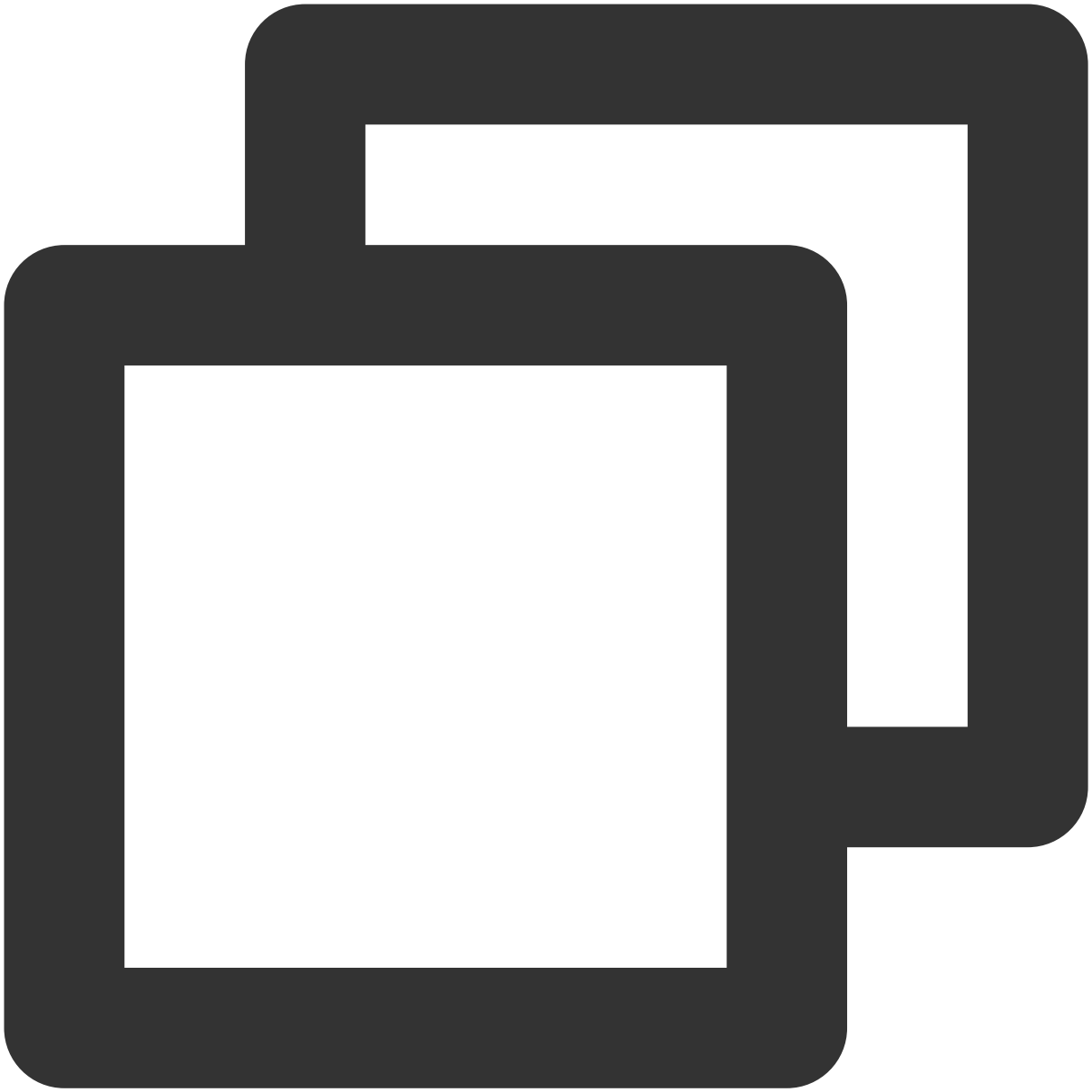
步骤5：拍照和录像功能实现

示例代码如下：

拍照

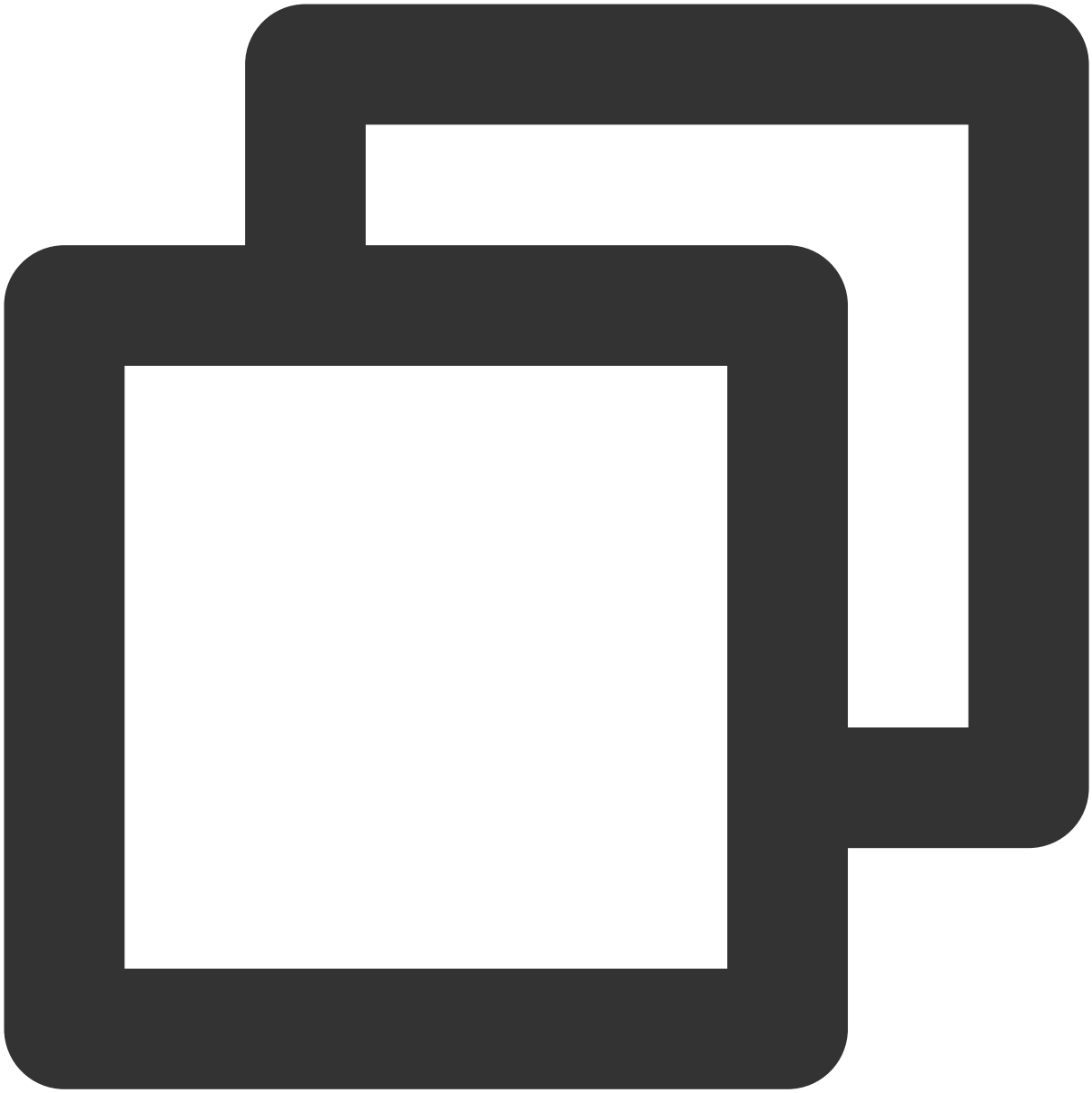
录像

SDK 会返回包含宽高和 buffer 数据的对象，用户可以通过自己页面内预设的 2d canvas（上述代码中id为photo-canvas）绘制此数据并导出为图片文件。



```
async takePhoto() {
    const {uint8ArrayData, width, height} = this.ar.takePhoto(); // takePhoto 方法返
    const photoCanvasNode = await this.getCanvasNode('photo-canvas');
    photoCanvasNode.width = parseInt(width);
    photoCanvasNode.height = parseInt(height);
    const ctx = photoCanvasNode.getContext('2d');
    // 用 sdk 返回的数据创建 ImageData 对象
    const imageData = photoCanvasNode.createImageData(uint8ArrayData, width, height
    // 将 ImageData 对象绘制到 canvas 上
    ctx.putImageData(imageData,0,0,0,0,width,height);
    // 将 canvas 保存为本地图片
```

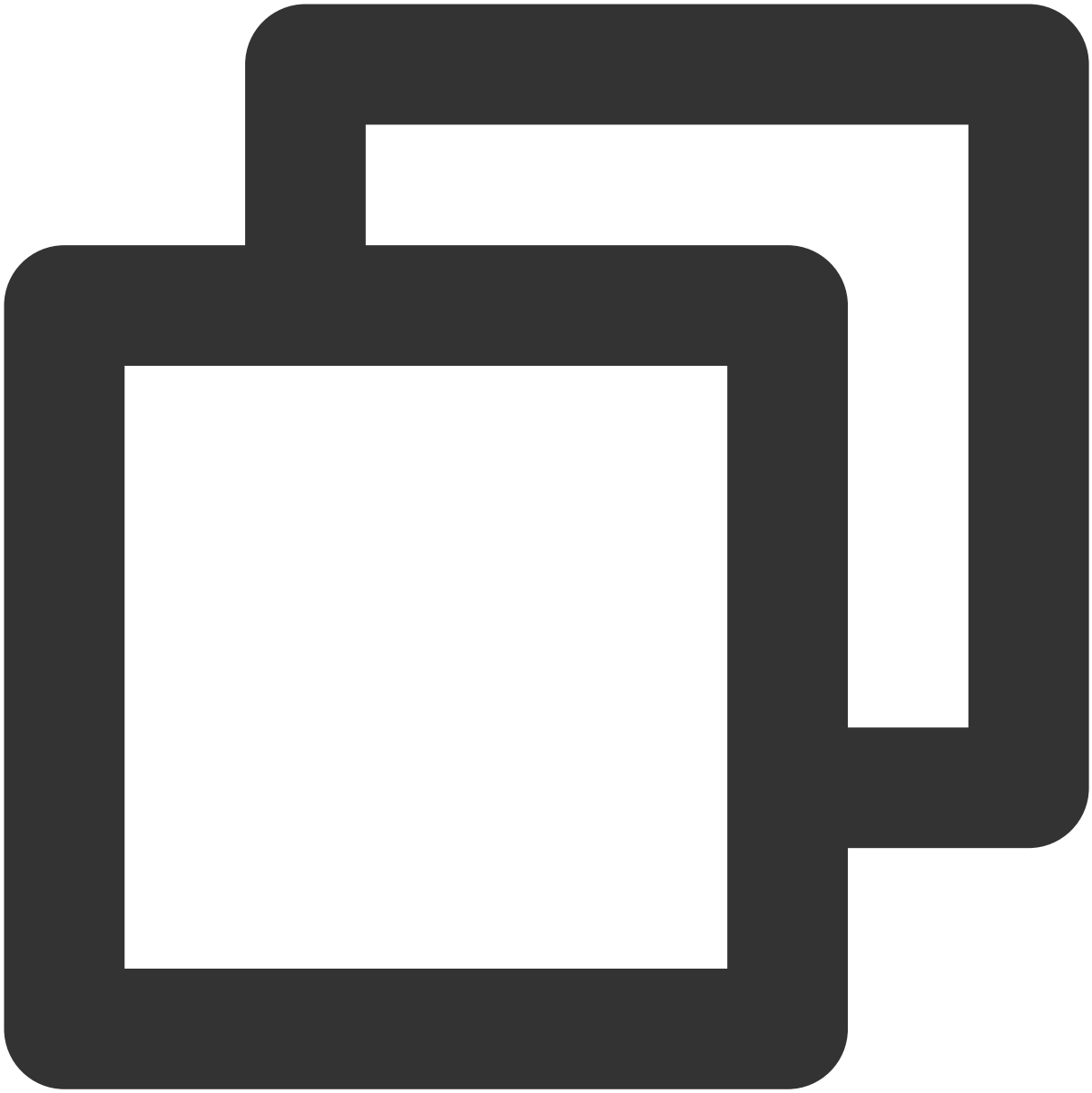
```
    wx.canvasToTempFilePath({
        canvas: photoCanvasNode,
        x: 0,
        y: 0,
        width: width,
        height: height,
        destWidth: width,
        destHeight: height,
        success: (res) => {
            // 保存照片到本地
            wx.saveImageToPhotosAlbum({
                filePath: res.tempFilePath
            });
        }
    })
}
```



```
Component({
    methods: {
        // 开始录像
        startRecord() {
            this.setData({
                recording: true
            });
            this.ar.startRecord()
        }
        // 结束录像
        async stopRecord() {
```

```
            const res = await this.ar.stopRecord();
            // 保存录像到本地
            wx.saveVideoToPhotosAlbum({
                filePath: res.tempFilePath
            });
            this.setData({
                recording: false
            });
        }
    }
})
```

当小程序切换后台或检测到手机锁屏时，需要调用 stopRecord 停止录像，再次回到页面时重新开启SDK即可。



```
onShow() {
    this.ar && this.ar.start();
},
onHide() {
    this.ar && this.ar.stop();
},
async onUnload() {
    try {
        this.ar && this.ar.stop();
        if (this.data.recording) {
            await this.ar.stopRecord({
```

```
                destroy: true,
            });
        }
    } catch (e) {
    }
    this.ar && this.ar.destroy();
}
```

代码示例

您可以下载 示例代码 解压后查看 `ar-miniprogram` 代码目录。

常见问题

最近更新时间：2023-04-11 16:26:55

在使用 Web 美颜特效的过程中，您可能会遇见以下问题，可以参考本文的解答。

在 Chrome 中运行 Demo 发现画面颠倒且卡顿

SDK 使用 GPU 进行加速，您需要在浏览器设置中找到**使用硬件加速模式**并启用。

现在有一个 Web 端推流直播应用，能否使用 Web 美颜特效 SDK 来美化推流画面？

可以，SDK 可以作为一个中间渲染处理器，支持多种输入和输出源，您可以参考 结合WebRTC的推流 以及 结合 TRTC 推流 轻松扩展您的线上应用，快速实现美颜特效功能。

我的签名服务会不会被频繁调用？

不会，SDK 内部针对签名有缓存机制，您也可以在自己的 `getSignature` 方法中自定义返回逻辑，只要符合方法签名即可。

控制台 - 制作工具中预览的效果和实际调用 SDK 时是否会有出入？

不会，SDK 使用时的渲染结果和您在制作工具中预览的效果一致，即制作时期望的预览效果即您实际生产环境中的效果。

如何使用 localhost 本地开发？

可以通过创建测试 License，Domain 中指定 localhost 及端口号（限制端口）。
或者购买正式版 License，正式版有效期内可以使用 localhost 本地开发（不限制端口）。

快直播推流失败，打开浏览器 console 看到提示“streamurl authentication failed”？

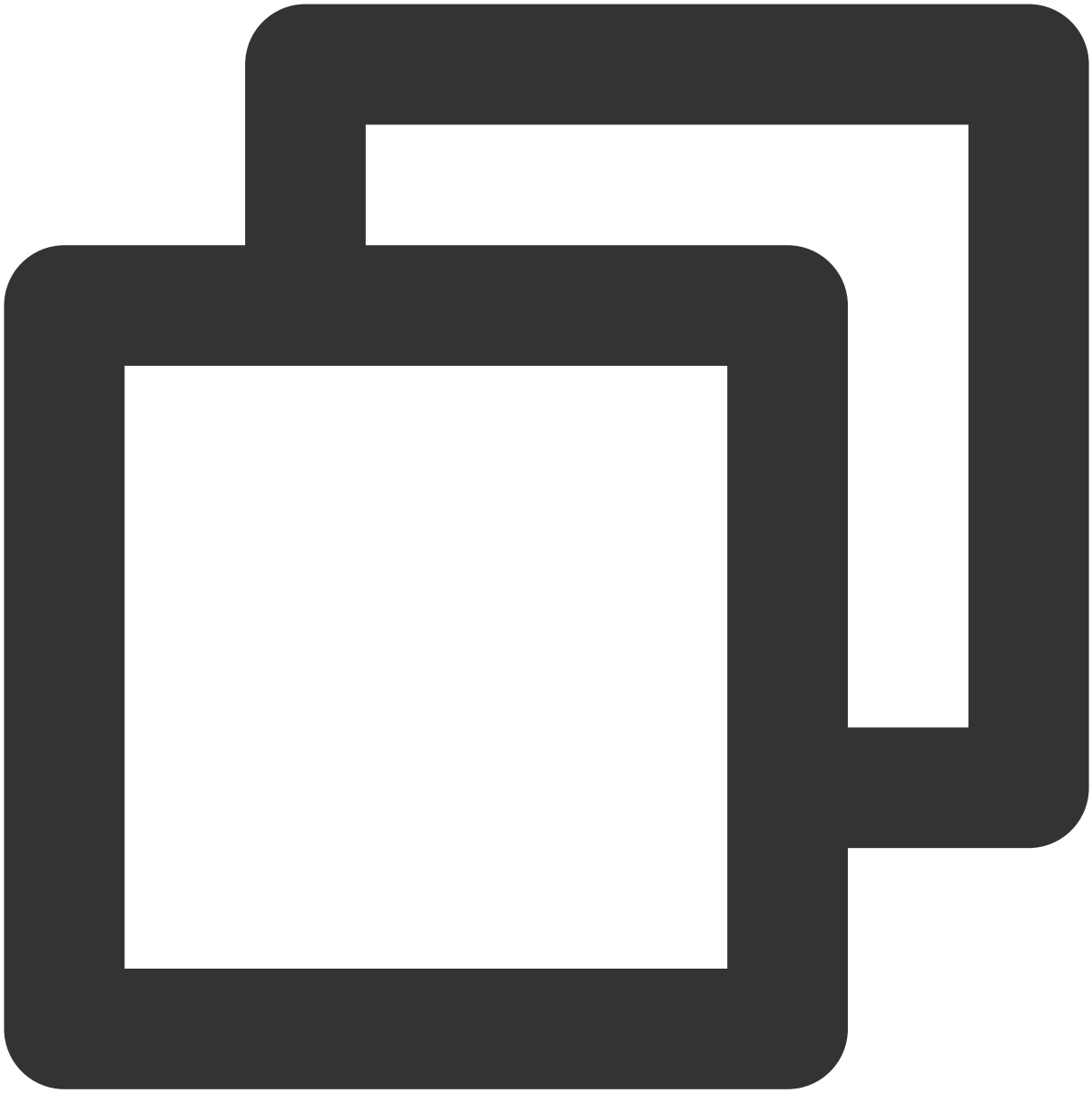
一般是因为签名已过期，请重新计算直播推流签名，推流 URL 拼装方法请参见 自主拼装直播 URL。

getEffectList 拉取素材资源时报错是什么原因？

一般是调用时机错误导致的报错，注意**在 SDK `created` 回调或之后调用 `getEffectList`，此时就可以根据素材数据完成业务交互逻辑**，具体使用案例请参见 最佳实践文档。

Web 接入 SDK 的内置相机后，预览视频效果时出现回音？

当本地预览开启了声音并且外放时，播放的声音会被麦克风采集作为内置相机的声音源，则会造成回音问题，解决方案将预览的 video 静音。



```
const output = await sdk.getOutput()
const video = document.createElement('video')
document.body.appendChild(video)

video.setAttribute('muted', '')
video.volume = 0
video.srcObject = output
```

SDK 提示鉴权失败，接口返回401？

SDK 内部会通过参数透传的 `getSignature` 方法请求签名并向后台进行鉴权，当签名的时间戳过期会自动重试一次，重试失败则抛错并拦截后续的所有流程。开发者可以检查 getSignature 的逻辑，查看是否时间戳过期（5分钟有效）或者签名生成逻辑不正确。

小程序目前都支持哪些能力？

截止 SDK 0.3.0版本，小程序支持美颜、美妆、滤镜、贴纸、3D特效等功能，**虚拟背景、Animoji 表情及虚拟形象能力暂不支持**，后续得到支持后会第一时间通知您。