人脸核身

接入文档

产品文档

腾讯云

文档目录

接入文档
开始集成
活体人脸比对 (移动端H5) 接入指引
接入流程说明
H5兼容性与模式切换说明
活体人脸比对 (App SDK) 接入指引
接入流程说明
SDK接口API概述
Android 接口概述
iOS 接口概述
SDK自定义能力说明
Android 自定义能力
iOS 自定义能力
常见问题与修复指引
Identity Verification(App SDK) 接入指引
接入流程说明
SDK 接口 API 概述
Identity Verification Android 接口概述文档
Identity Verification iOS 接口概述文档
常见问题与修复指引
Identity verification（移动端 H5）接入指引
接入流程说明
活体人脸比对（纯 API）接入指引
接入流程说明
其他指引
三分钟跑通接口
连接腾讯云 API 接口
错误码

接入文档
开始集成

最近更新时间：2024-07-24 17:26:27

将腾讯云 eKYC 服务与你的应用程序集成，可以按照下图所示三个阶段进行：准备阶段、集成阶段和测试阶段。

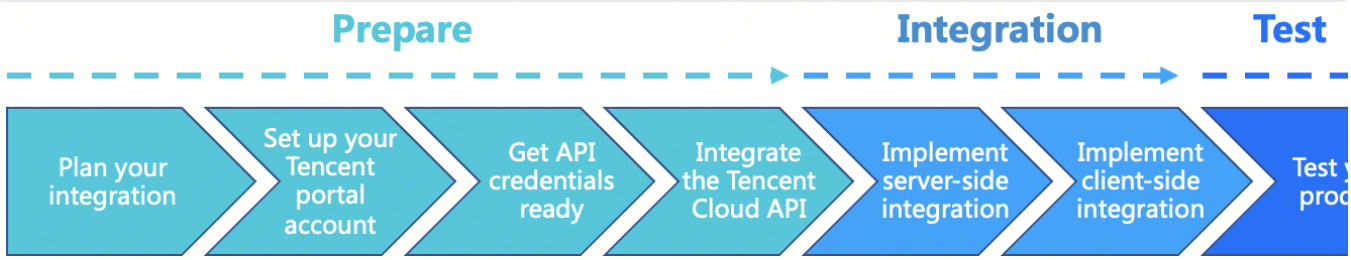


Figure 1. Integration flow

如上图所示，集成 eKYC 分为三个不同的阶段，下面将分阶段进行详细介绍：

准备阶段（Prepare）

集成 eKYC 准备阶段，可以按照下面的步骤执行：

1. 做好计划（Plan your integration）

在您采取任何实际行动之前，请根据所需要的功能选择合适的产品：

子产品名称	功能简介	功能明细	支持的国家/地域	支持接入的方式
活体人脸比对	判断视频上的用户是否为真人，并与客户后台传入的照片比对判断是否为同人。	●人脸姿态实时检测 ●活体检测（支持配置：静默、动作、光线、眨眼+光线模式） ●与客户后台自传照片进行人脸比对	全地域支持	●APP SDK ●移动端H5 ●纯API
Identity verification	提供一整套自动化身份认证的方案，通过引导用户拍摄证件照和录制人脸视频，判	●证件文字识别 ●自动提取证件人像照 ●人脸姿态实时检测	●中国香港 ●马来西亚 ●菲律宾 ●印度尼西亚	APP SDK

	断用户是否为真人、是否为本人。	●活体检测（支持配置：静默、动作、光线、眨眼+光线模式） ●与证件人像照进行人脸比对	●新加坡（如还需求其他更多地域的证件识别，可联系我们进行定制）	

腾讯云 eKYC 提供了不同的集成方式供您与不同的产品进行集成。目前支持4种集成方式：

A. 活体人脸比对

活体人脸比对（App SDK）模式：如果您希望将 eKYC 的 活体人脸比对服务与您的本机移动应用程序（iOS/Android）集成，请使用此模式。
活体人脸比对（移动端 H5）模式：如果您希望将 eKYC 的活体人脸比对服务与移动网络 (H5) 应用程序集成或通过现代移动网络浏览器进行交互，请使用此模式。
活体人脸比对（纯API）模式：如果您希望通过 API 将 eKYC 的活体人脸比对服务与您的服务器端应用程序集成，请使用此模式。

B. Identity verification

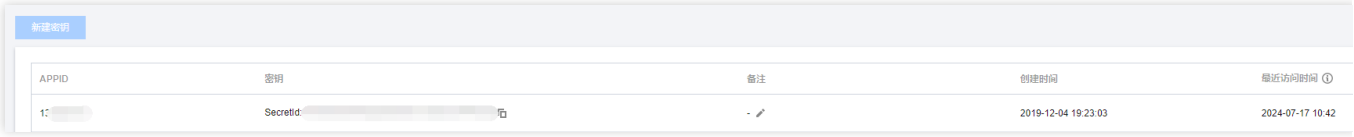
Identity verification (App SDK) 模式：如果您希望将 eKYC 的 Identity verification 服务与您的本机移动应用程序（iOS/Android）集成，请使用此模式。

2. 设置腾讯云账号（Set up your Tencent portal account）

A.注册腾讯云账号
在使用腾讯云 eKYC 服务前，您需要先注册一个腾讯云账号。请参考 注册腾讯云 教程，并进行企业实名认证。（如果您已注册，请跳过该步骤。）
B.开通 eKYC 服务
使用腾讯云账号，登录腾讯云 eKYC 控制台 开通服务。

3. 获取API秘钥（Get API credentials ready）

腾讯云通过 SecretId 和 SecretKey 对开发者的身份和权限进行校验，您可以参考以下步骤获取腾讯云 API 访问密钥：
A. 进入 云 API 密钥管理 页面，在左侧导航栏选择**访问管理 > API 密钥管理**，进入 API 密钥管理页面。
B. 单击**新建密钥**创建密钥，并保存 secretid 和 secretkey 供后续 API 调用传参。（如果您已经具有腾讯云密钥，请跳过本步骤。）



4. 集成腾讯云API（Integrate the Tencent Cloud API）

腾讯云 API 可以通过 API 3.0 Explorer 在线调试页面调用，您可以通过阅读三分钟跑通接口快速查看接口的入参以响应结果。集成腾讯云 API 之前，你必须集成腾讯云 SDK 以简化腾讯云 API 接入流程以及保证 API 接口的请求和响应都符合预期，更多的集成信息可以参考 连接腾讯云 API 接口。

集成阶段（Integration）

eKYC 提供了多种集成模式，不同的集成模式，其集成方案是不一样的，可以参考不同模式的集成指引文档完成对应模式的集成。
下面是不同模式下的接入指引：
活体人脸比对（App SDK）接入指引
活体人脸比对（移动端 H5）接入指引
活体人脸比对（纯API）接口文档
Identity verification（App SDK）接入指引

测试阶段（Test）

完成集成之后，您可以通过测试验证你的集成是否成功，强烈建议你在生产环境上线之前完成全流程测试，腾讯云目前不提供测试环境，可以通过腾讯云生产环境完成您的集成测试工作。

活体人脸比对 (移动端H5) 接入指引
接入流程说明

最近更新时间：2024-01-25 10:37:43

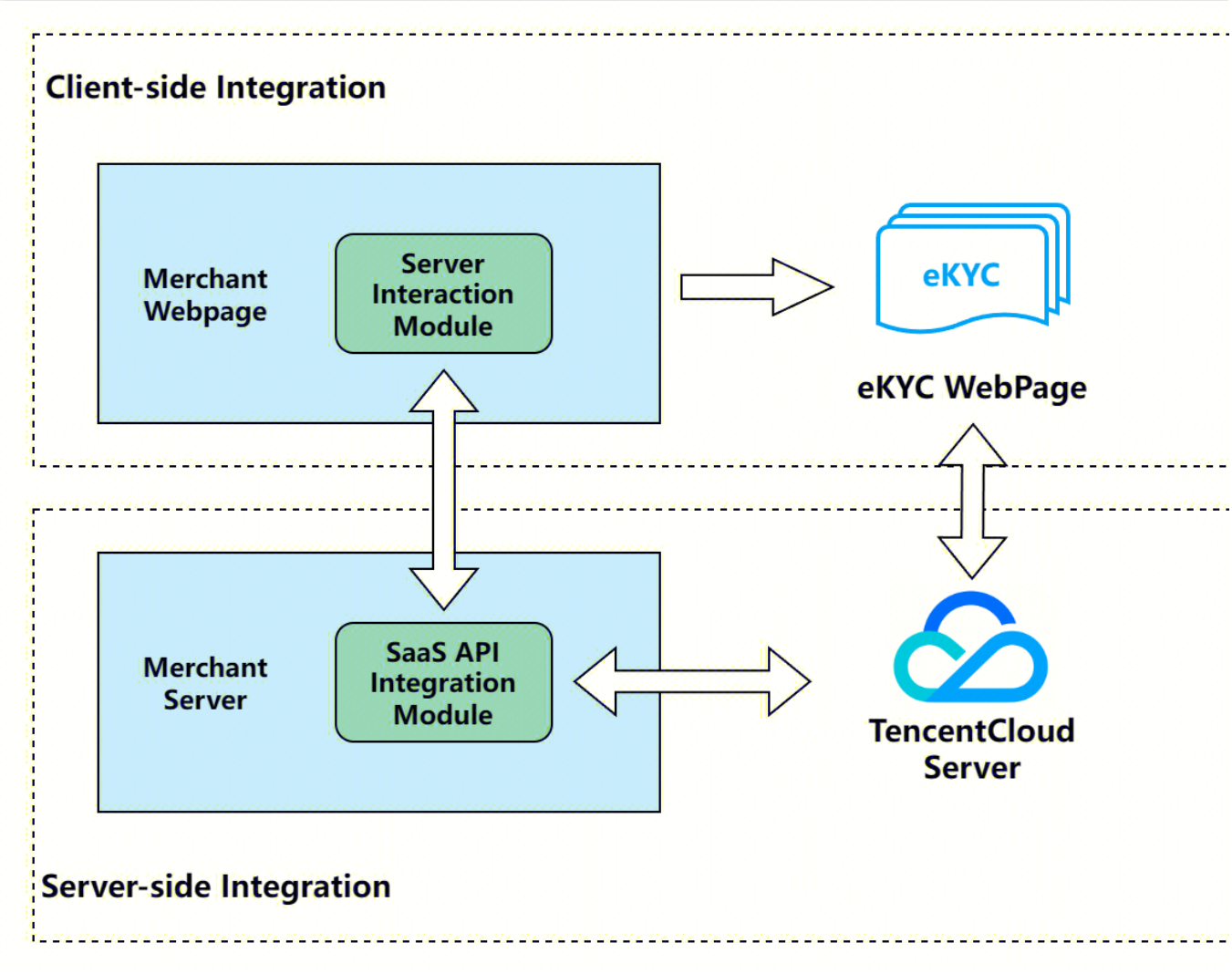
本文介绍 eKYC 活体比对（移动端 H5）整体接入的流程。

接入准备

注册腾讯云企业账号，请参见 注册指引。
完成企业实名认证，请参见 企业实名指引。
登陆 eKYC 控制台 开通服务。

整体架构图

下图为 eKYC 活体比对（移动端 H5）的架构图：



整体交互流程

下图展示了客户和腾讯云 eKYC 的整体交互逻辑，图中角色说明：

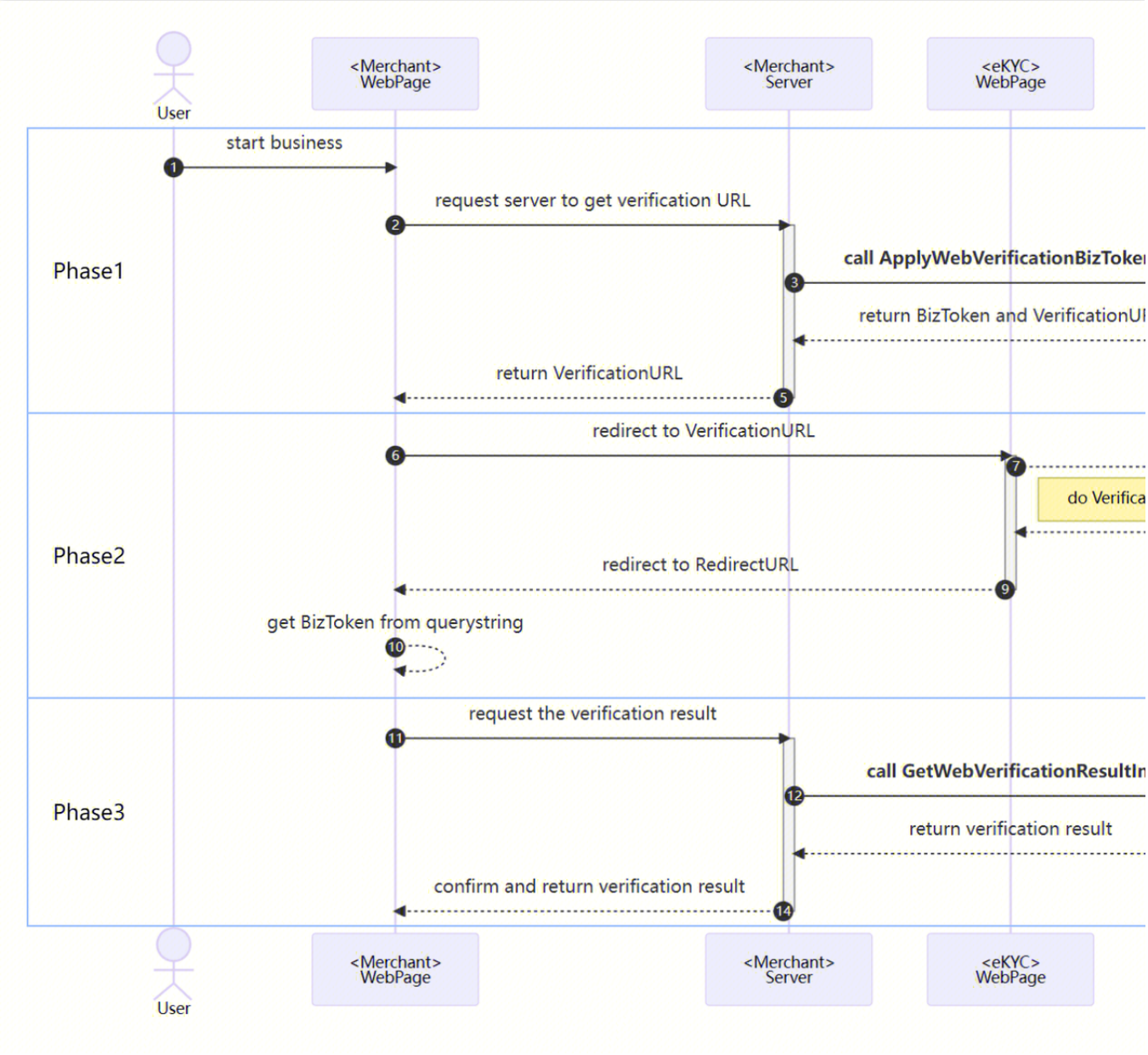
User：移动端 H5 用户

Merchant WebPage：客户前端页面

Merchant Server：客户后端服务

eKYC WebPage：eKYC 前端页面

eKYC Server：eKYC 后端服务



具体的推荐交互流程详细交互如下：

阶段一

1. 用户端业务触发核验流程。
2. Merchant WebPage 发送请求到 Merchant Server，告知 Merchant Server 需要发起一次核验流程。
3. Merchant Server 传入相关参数调用 ApplyWebVerificationBizTokenIntl 接口，参考服务端集成第一步。
4. eKYC Server 接收到请求后，将该次核验流程的 BizToken和VerificationURL 返回给 Merchant Server。
5. Merchant Server 将获取到的 BizToken 进行保存，并将 VerificationURL 下发给 Merchant WebPage。

阶段二

1. Merchant WebPage 跳转到 VerificationURL，打开 eKYC WebPage，参考前端集成第一步。
2. 用户在 eKYC WebPage 上进行核验流程。

3. 核验完成后，eKYC Server 将核验结果下发给 eKYC WebPage，Merchant WebPage 展示结果页面。
4. 用户点击下一步后，eKYC WebPage 会跳转回 RedirectURL 地址，并在 URL 上拼接 token 参数。
5. Merchant WebPage 通过地址获取到当前核验流程的 token 参数，参考前端集成第二步。

阶段三

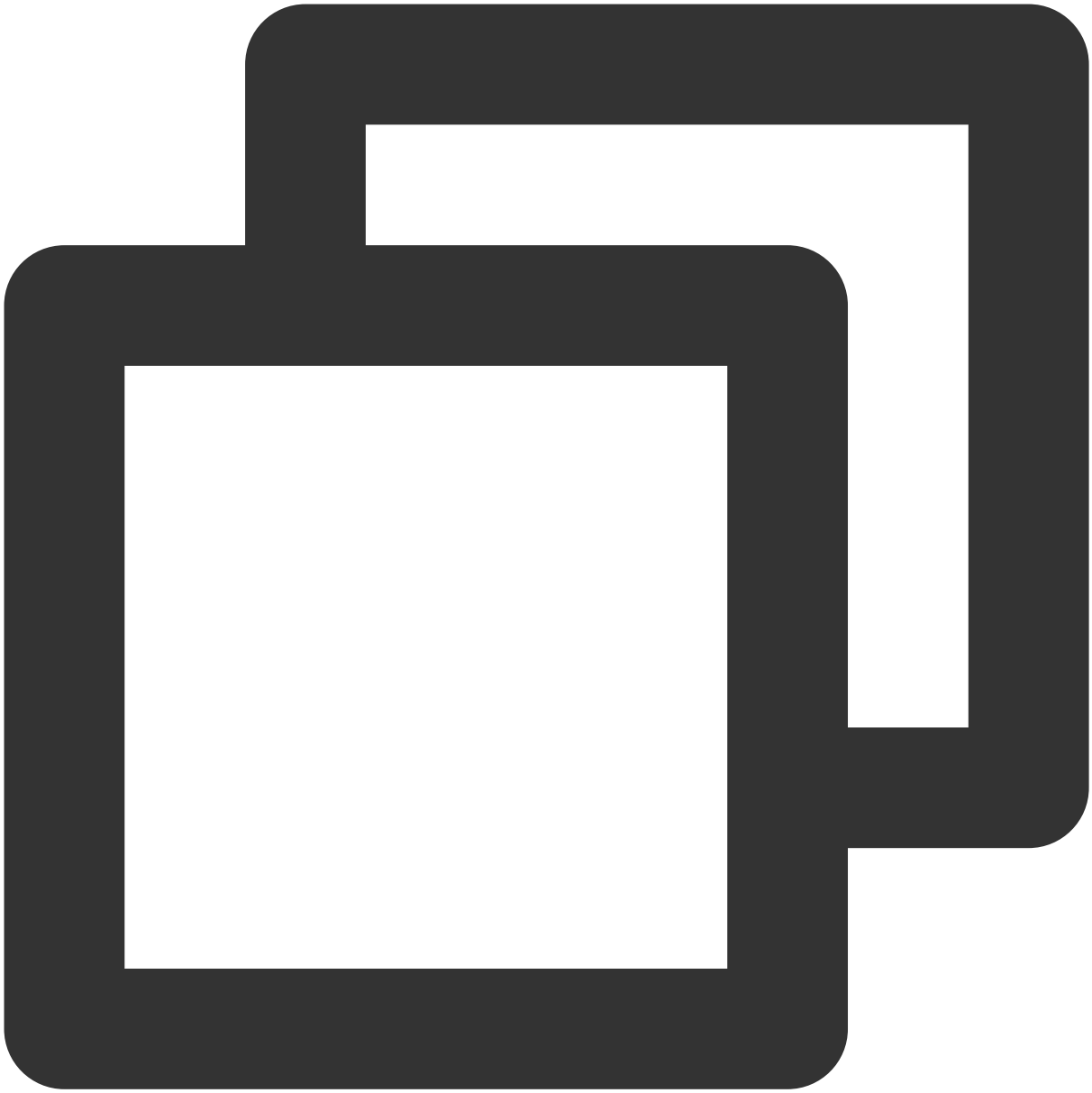
1. Merchant WebPage 发送请求到 Merchant Server，告知 Merchant Server 获取核验的结果信息。
2. Merchant Server 传入相关参数调用 GetWebVerificationResultIntl 接口，参考服务端集成第二步。
3. eKYC Server 接收到请求后，将该次核验流程的详细信息返回给 Merchant Server。
4. Merchant Server 将结果信息返回给 Merchant WebPage，Merchant WebPage 根据结果进行后续业务流程。

服务端集成

调用申请Web核验令牌生成核验地址（对应阶段一）

调用ApplyWebVerificationBizTokenIntl接口，获取 **BizToken** 和核验地址 **VerificationURL**。对应时序图的第3点。
CompareImageBase64：比对照片的 base64 编码字符串，base64编码后的字符串不超过8MB。
RedirectURL：核验结束后重定向的 Web 回跳地址，包含协议头，主机名和路径，如：https://www.tencentcloud.com/products/faceid。核验流程完成后，回跳地址会拼接当次流程的 BizToken，以 https://www.tencentcloud.com/products/faceid?token={BizToken} 的格式进行跳转。
Extra：业务透传参数，最大长度为1000个字符，会在 GetWebVerificationResultIntl 接口中返回，如无需要可以不传。
Config：自定义核验页面的相关配置，如无需要可以不传。
Config 数据结构如下：
AutoSkip：验证成功时，是否跳过结果展示页面，自动跳转到 RedirectURL，默认为 false。

接口调用代码示例



```
package main

import (
  "fmt"
  "os"

  "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common"
  "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/profile"
  "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/regions"
  faceid "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/faceid/v
  "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/errors"
```

```
)

func ApplyWebVerificationBizTokenIntl(imageBase64 string) {
  // 设置云API访问秘钥
  credential := common.NewCredential(
    os.Getenv("TENCENTCLOUD_SECRET_ID"),
    os.Getenv("TENCENTCLOUD_SECRET_KEY"),
  )
  cpf := profile.NewClientProfile()
  client, _ := faceid.NewClient(credential, regions.Singapore, cpf)
  request := faceid.NewApplyWebVerificationBizTokenIntlRequest()
  request.RedirectURL = common.StringPtr("https://www.tencentcloud.com/products/fac
  // 传入CompareImageBase64和Extra参数
  request.CompareImageBase64 = common.StringPtr(imageBase64)
  request.Extra = common.StringPtr("ExtraString")
  response, err := client.ApplyWebVerificationBizTokenIntl(request)
  if _, ok := err.(*errors.TencentCloudSDKError); ok {
    fmt.Printf("An API error has returned: %s", err)
    return
  }
  if err != nil {
    panic(err)
  }
  // 获取BizToken和verificationURL
  bizToken := *response.Response.BizToken
  verificationURL := *response.Response.VerificationURL
  fmt.Printf("BizToken: %s, VerificationURL: %s", bizToken, verificationURL)
}
```

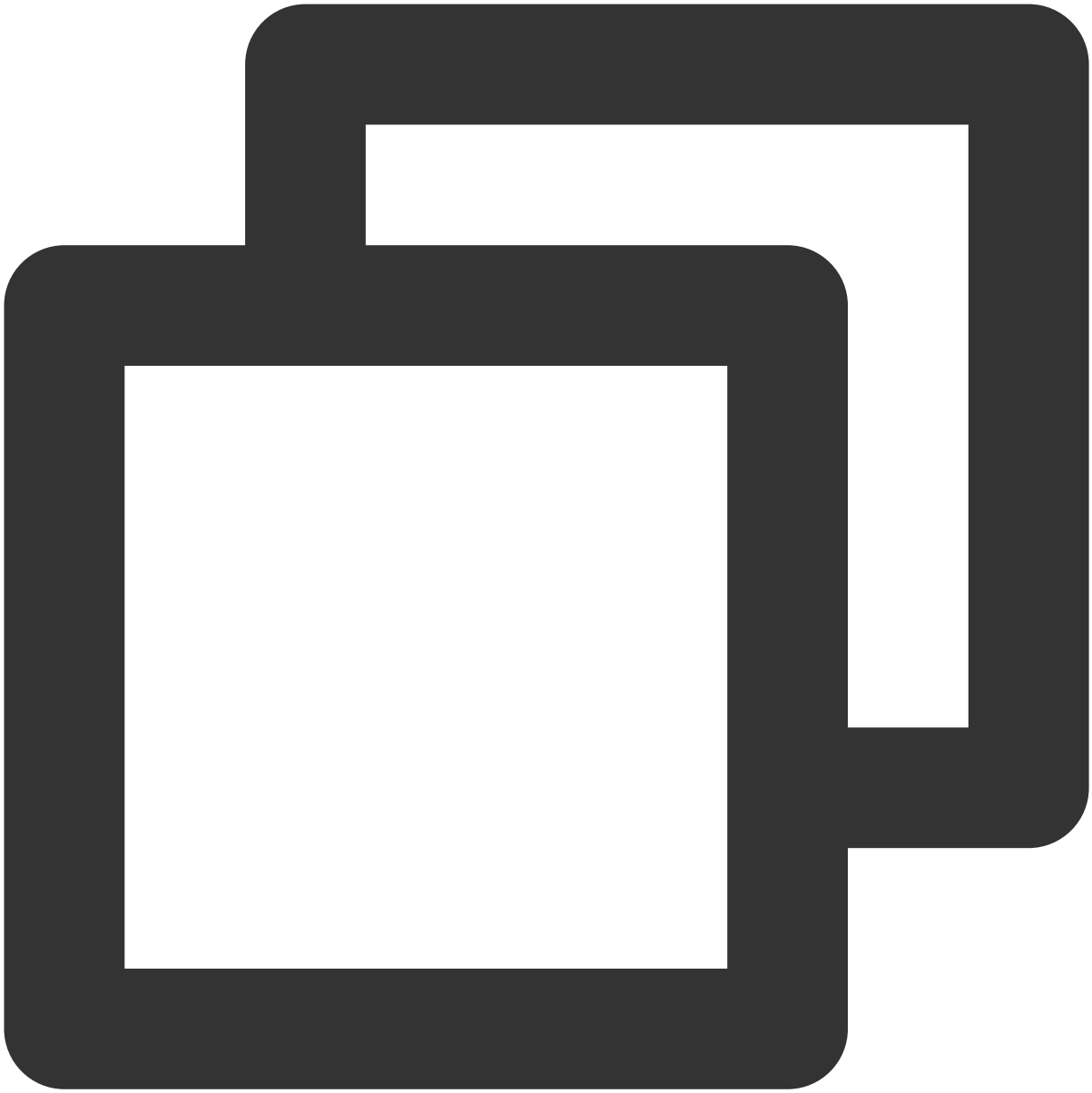
确认当次核验流程的结果（对应阶段三）

核验流程结束后，商户前端通知商户服务端获取本次核验的结果，商户服务端调用 GetWebVerificationResultIntl 接口，将最终的结果返回给前端页面。对应时序图的第12点。

本次核验的最终结果应以该接口返回的信息为准，当响应的 ErrorCode 字段为0时，视为本次核验流程通过，其他情况下视为未通过。详细的错误码列表请参考Liveness Detection and Face Comparison (Mobile HTML5) Error Codes。

BizToken：由 ApplyWebVerificationBizTokenIntl 接口生成的 BizToken，当次核验流程的唯一标识。

接口调用代码示例



```
package main

import (
  "fmt"
  "os"

  "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common"
  "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/profile"
  "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/regions"
  faceid "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/faceid/v
  "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/errors"
```

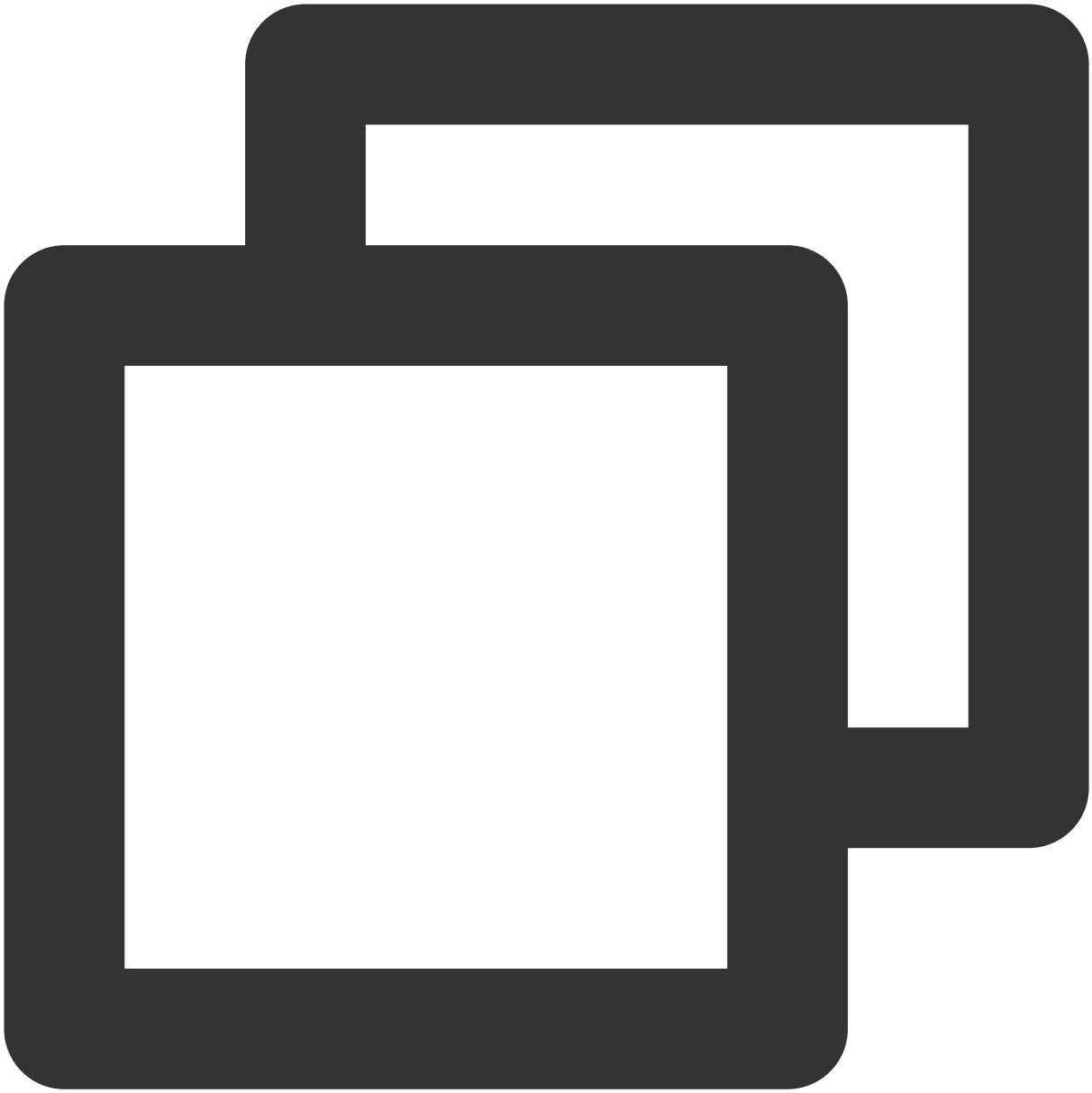
```
)

func GetWebVerificationResult(bizToken string) {
  // 设置云API访问秘钥
  credential := common.NewCredential(
    os.Getenv("TENCENTCLOUD_SECRET_ID"),
    os.Getenv("TENCENTCLOUD_SECRET_KEY"),
  )
  cpf := profile.NewClientProfile()
  client, _ := faceid.NewClient(credential, regions.Singapore, cpf)
  request := faceid.NewGetWebVerificationResultIntlRequest()
  // 传入BizToken参数
  request.BizToken = common.StringPtr(bizToken)
  response, err := client.GetWebVerificationResultIntl(request)
  if _, ok := err.(*errors.TencentCloudSDKError); ok {
    fmt.Printf("An API error has returned: %s", err)
    return
  }
  if err != nil {
    panic(err)
  }
  if response.Response.ErrorCode == nil {
    fmt.Print("the verification is uncompleted.")
    return
  }
  errorCode := *response.Response.ErrorCode
  errorMsg := *response.Response.ErrorMsg
  if errorCode == 0 {
        // 核验通过
    fmt.Print("Success")
  }else {
        // 核验不通过
    fmt.Printf("Fail: %s\\n", errorMsg)
  }
}
```

前端集成

获取 VerificationURL 并跳转，发起核验流程（对应阶段二）

客户前端页面获取服务端请求到的 VerificationURL，并跳转到该地址进入核验流程，用户按照提示完成活体比对流程。对应时序图的第6点。

代码示例

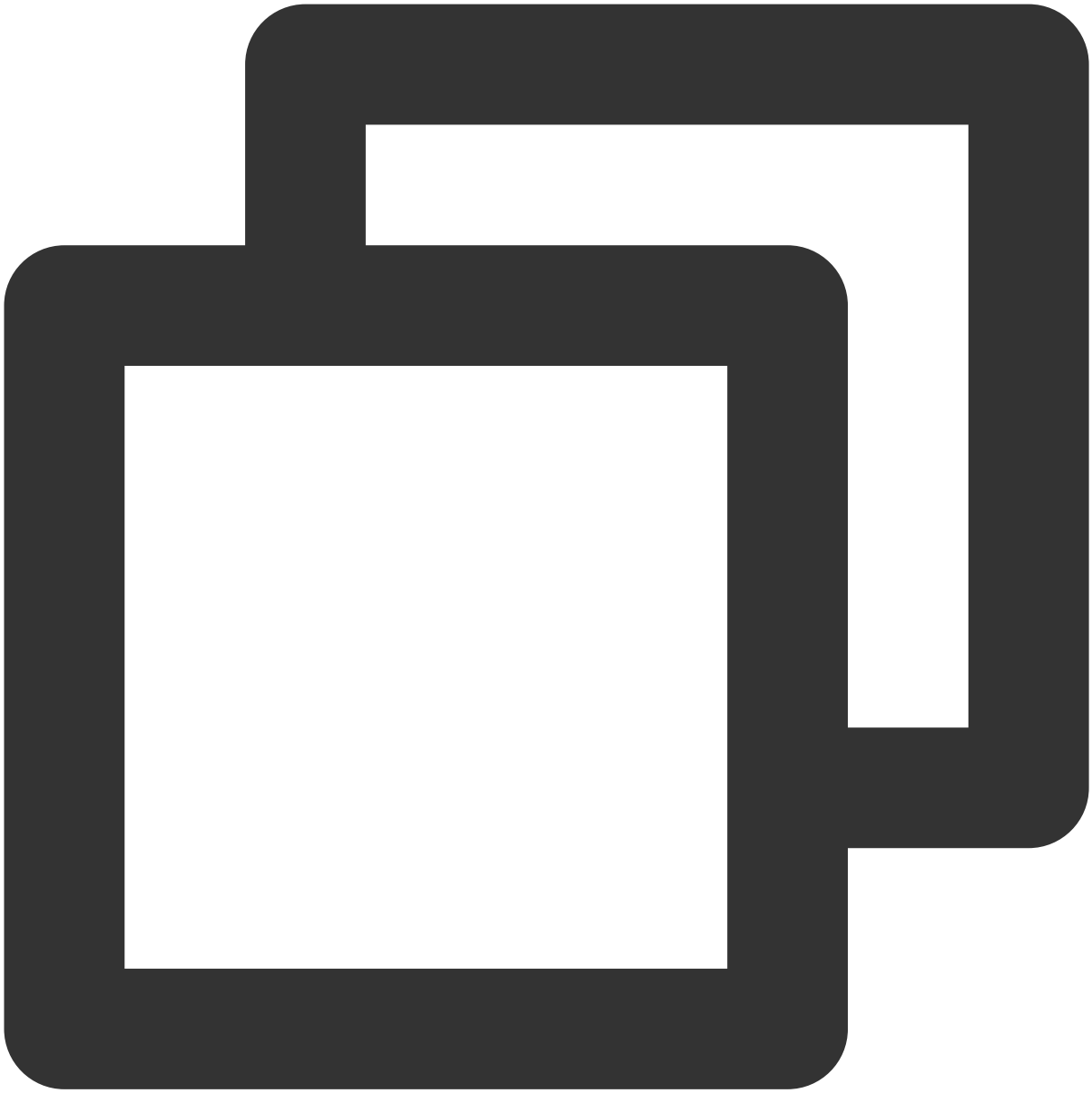


```
// 从服务端获取VerificationURL
const VerificationURL = 'https://sg.faceid.qq.com/reflect/?token=*****';
// 前端页面跳转
window.location.href = VerificationURL;
```

从回调地址获取 BizToken，向后端请求核验结果（对应阶段二）

在核验完成后，页面会跳转到 RedirectURL上。RedirectURL 会拼接当次流程的 **BizToken** 参数，通过解析 RedirectURL 获取 BizToken 参数，用于拉取本次活体比对的结果信息。对应时序图的第12点。

代码示例



```
// 获取RedirectURL
const RedirectURL = "https://*?token={BizToken}";

// 解析获得RedirectURL的BizToken参数，用于拉取本次活体比对的结果信息
const bizToken = getURLParameter(RedirectURL, "token");
if (bizToken) {
  // 使用bizToken用于拉取本次活体比对的结果信息
}
```

```
/**
/ * 解析url的参数
/* @params url 查询url
/* @params variable 查询参数
*/
function getURLParameter(url, variable) {
  const query = url.split('?')[1] || '';
  const vars = query.split('&');
  for (let i = 0; i < vars.length; i++) {
    const pair = vars[i].split('=');
    if (pair[0] == variable) {
      return pair[1];
    }
  }
  return (false);
}
```

H5兼容性与模式切换说明

最近更新时间：2024-07-29 11:45:20

兼容性说明

因 Web 实时音视频技术对浏览器和手机系统存在兼容性要求，实时音视频技术的兼容性情况如下：

手机平台	浏览器	兼容性要求
IOS	微信内嵌浏览器	iOS 系统版本14.3+，微信版本6.5+
	Safari 浏览器	iOS 系统版本11.1.2+，浏览器版本11+
	Chrome 浏览器	iOS 系统版本14.3+
Android	微信内嵌浏览器	支持
	自带浏览器	Android 系统版本7+ 华为、OPPO、VIVO、魅族、荣耀、三星等自带浏览器兼容性较好（支持率80%），小米自带浏览器兼容性一般（支持率30%）
	其他浏览器	Android 系统版本7+，支持 Chrome 浏览器，不支持QQ 、UC浏览器

注意：

由于 H.264 版权限制，华为系统的 Chrome 浏览器和以 Chrome WebView 为内核的浏览器,均不支持实时音视频技术的正常运行。

对于不支持实时音视频技术的情况，Web 人脸核身会将光线活体检测切换为视频录制模式,保证用户可正常完成核验流程。

模式切换说明

在 Web 人脸核验流程后，优先使用光线活体检测模式，但在不满足实时音视频兼容性要求下，会自动切换为静默(视频录制)模式。两种模式服务流程如下：

1. 光线活体检测模式用户做 Web 人脸核验流程：

2. 静默模式用户做 Web 人脸核验流程：

光线活体摄像机授权说明

调用 Web 人脸核验流程时，光线活体检测模式需要用户摄像机的使用授权。如果用户拒绝授权,需要重新进入人脸核验流程和允许授权。部分浏览器会存在无法拉起授权界面，可以尝试清理浏览器的缓存。

返回信息	处理措施
Unable to access your camera/mic. Please make sure that there is no other app requesting access to them and try again.	建议用户检查所需的摄像机是否被占用
Mic and camera permissions of your device are required during the whole verification. Please clear your browser cache and try again.	建议用户重新进入并允许摄像机的授权
Please check whether the camera/mic can be accessed normally and try again	建议用户检查所需的摄像头设备是否正常

活体人脸比对 (App SDK) 接入指引

接入流程说明

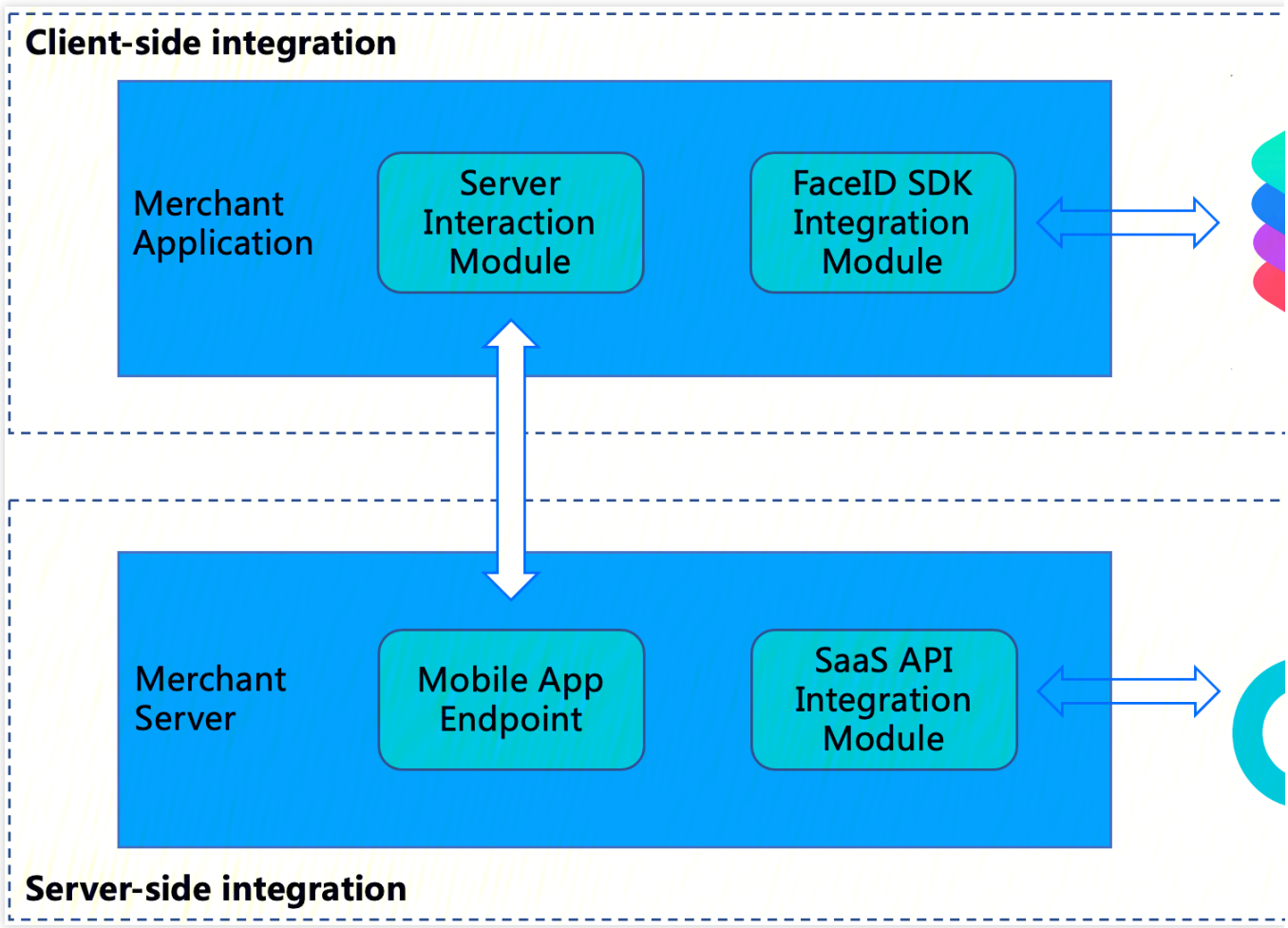
最近更新时间：2024-08-01 16:42:58

接入准备

注册腾讯云企业账号，请见 注册指引。
完成企业实名认证，请见 企业实名指引。
登录慧眼控制台 开通服务。
联系我们 获取最新的 SDK 及 License。

整体架构图

下图为活体人脸比对 SDK 集成的架构图：



慧眼 SDK 集成包括两部分：

客户端集成： 将慧眼 SDK 集成到客户终端业务 App 中。

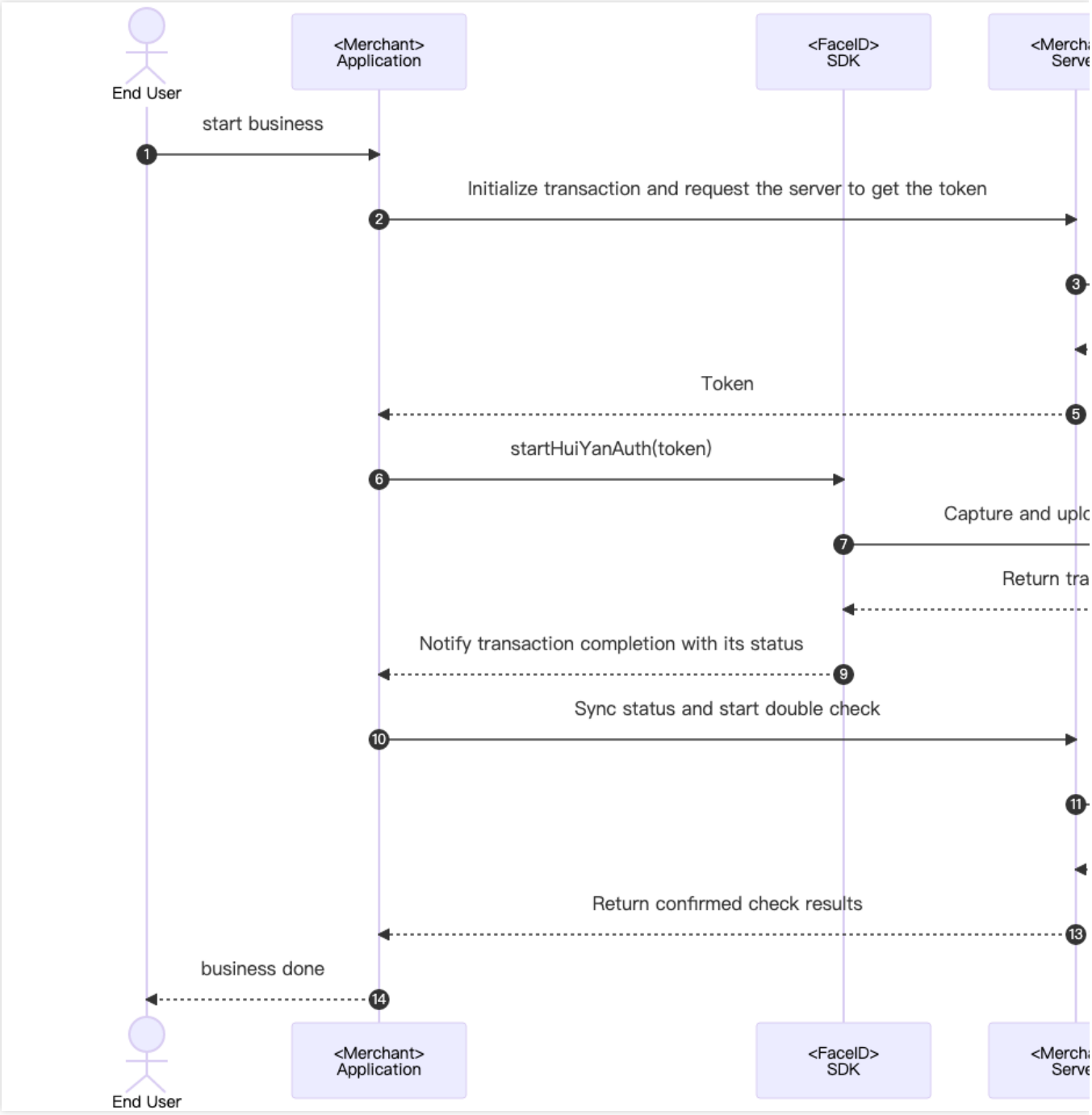
服务器端集成： 在您的（商家）服务器中公开您的（商家）应用程序的端点，以便商家应用程序可以与商家服务器交互，然后访问 FaceId SaaS API 以获取串联活体比对流程的 `SdkToken` 以及通过 `SdkToken` 拉取最终的核身结果。

整体交互流程

集成方只需要传入 Token 并启动对应的慧眼 SDK 的活体检测方法，便可以完成活体检测，并返回活体结果。

1. Token 的获取可以参考云 API 接口：GetFaceIdTokenIntl
2. 用户主动拉取活体结果的云 API 接口：GetFaceIdResultIntl

下图展示了 SDK、客户端以及服务器端的整体交互逻辑，图中负责模块解析：



具体的推荐交互流程详细交互如下：

1. 用户触发终端 Merchant Application 准备调用核身业务场景。
2. Merchant Application 发送请求到 Merchant Server，告知启动一次核身业务需要活体业务Token。
3. Merchant Server 传入相关参数调用云API接口GetFaceIdTokenIntl。
4. FaceID SaaS 接收到GetFaceIdTokenIntl调用后，下发此次业务的token给 Merchant Server。
5. Merchant Server可以将获取到的业务Token，下发给客户的 Merchant Application。
6. Merchant Application 调用慧眼SDK的启动接口**startHuiYanAuth**传入token与配置信息，开始核身验证。
7. FaceID SDK 捕捉并上传所需的用户数据，包括活体数据等，上传至 FaceID SaaS 。
8. FaceID SaaS 在完成核身检测（包括活体与比对流程）以后，会返回结果给 FaceID SDK 。

9. FaceID SDK 主动触发回调给 Merchant Application 通知核验完成以及核验状态。
10. Merchant Application 接收到回调以后，可以发送请求通知 Merchant Server去主动获取本次核身的结果，进行确认检查。
11. Merchant Server主动调用 FaceID SaaS 的接口GetFaceIdResultIntl传入相关参数以及本次业务的Token，去获取本次核身的结果。
12. FaceID SaaS 接收到GetFaceIdResultIntl调用后，会返回本次核身的结果到 Merchant Server。
13. Merchant Server接收到本次核身的结果后，可以下发需要的信息到 Merchant Application。
14. Merchant Application 展示最后的结果，呈现在 UI 界面上，告知用户核身的结果。

接入流程

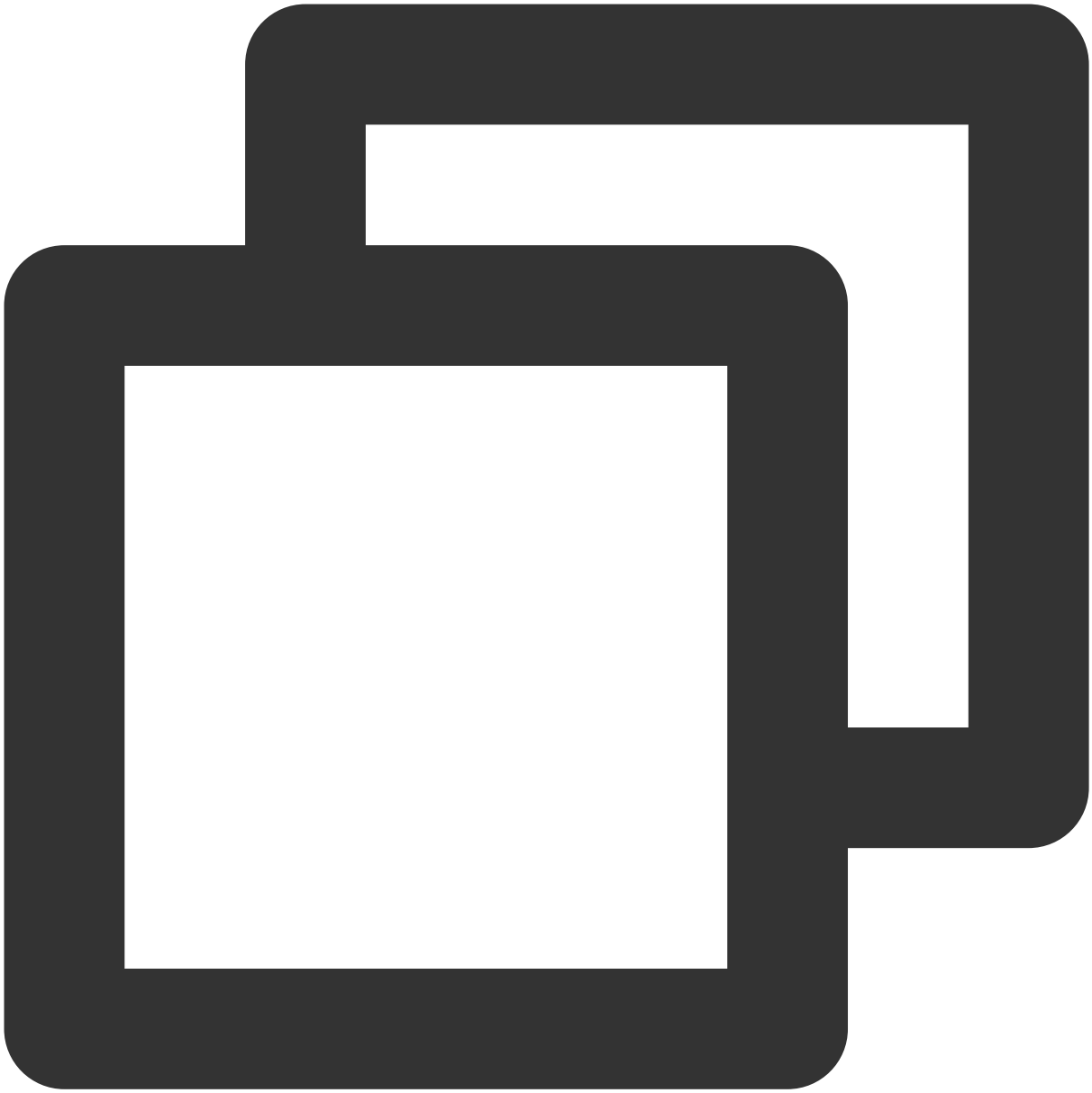
服务器端集成

1. 集成准备

在服务端集成之前，你需要按照获取API秘钥指引中的说明进行操作，开通腾讯云慧眼服务，并且拿到了腾讯云api访问秘钥SecretId和SecretKey。除此之外你还需要按照连接腾讯云API接口中的操作流程，引入你所熟悉开发语言的SDK包到你服务端模块中，以确保可以成功调用腾讯云API以及正确处理API请求和响应。

2. 开始集成

为了确保你的（商户）客户端应用程序能够跟你的（商户）服务端正常交互，商户服务端需要调用慧眼提供的API接口 **GetFaceIdTokenIntl** 获取SDKToken串联整个活体比对流程以及调用 **GetFaceIdResultIntl** 接口获取活体比对结果，商户服务端需要提供相应的端点给商户客户端调用，下面的示例代码使用 Golang语言作为案例，展示如何在服务端调用腾讯云API接口并拿到正确的响应结果。
注意： 该示例中仅仅演示商户服务端与腾讯云API服务交互所需要的处理逻辑，如果有需要的话你需要自己实现你的业务逻辑，比如：
当你通过 **GetFaceIdTokenIntl** 接口获取活体比对SDKToken之后，可以将客户端应用程序需要的其他响应同SDKToken一并返回给客户端。
当你通过 **GetFaceIdResultIntl** 接口获取活体比对结果之后，可以将返回的最佳帧照片保存起来，以便后续业务逻辑使用。



```
var FaceIdClient *faceid.Client

func init() {
    // 初始化客户端配置对下，您可以指定超时时间和其他配置项
    prof := profile.NewClientProfile()
    prof.HttpProfile.ReqTimeout = 60
    //  TODO 需要替换成您调用账号的 SecretId 和 SecretKey
    credential := cloud.NewCredential("SecretId", "SecretKey")
    var err error
    // 初始化调用慧眼人脸核身服务的客户端
    FaceIdClient, err = faceid.NewClient(credential, "ap-singapore", prof)
```

```
    if nil != err {
        log.Fatal("FaceIdClient init error: ", err)
    }
}

// GetFaceIdToken 获取人脸核身Token
func GetFaceIdToken(w http.ResponseWriter, r *http.Request) {
    log.Println("get face id token")
    // 步骤1: 解析请求参数
    _ = r.ParseForm()
    var SecureLevel = r.FormValue("SecureLevel")

    // 步骤2: 初始化请求对象，并给必要的参数赋值
    request := faceid.NewGetFaceIdTokenIntlRequest()
    request.SecureLevel = &SecureLevel
    // 步骤3: 通过FaceIdClient调用人脸核身服务
    response, err := FaceIdClient.GetFaceIdTokenIntl(request)

    // 步骤4: 处理腾讯云API的响应，并构造返回对象
    if nil != err {
        _, _ = w.Write([]byte("error"))
        return
    }
    SdkToken := response.Response.SdkToken
    apiResp := struct {
        SdkToken *string
    }{SdkToken: SdkToken}
    b, _ := json.Marshal(apiResp)

    // ... 其他业务处理代码

    //步骤5: 返回服务响应
    _, _ = w.Write(b)
}

//  GetFaceIdResult 获取人脸核身核验结果
func GetFaceIdResult(w http.ResponseWriter, r *http.Request) {
    // 步骤1: ... 解析请求参数
    _ = r.ParseForm()
    SdkToken := r.FormValue("SdkToken")
    // 步骤2: 初始化请求对象，并给必要的参数赋值
    request := faceid.NewGetFaceIdResultIntlRequest()
    request.SdkToken = &SdkToken
    // 步骤3: 通过FaceIdClient调用人脸核身服务
    response, err := FaceIdClient.GetFaceIdResultIntl(request)

    // Step 4: 处理腾讯云API的响应，并构造返回对象
```

```
    if nil != err {
        _, _ = w.Write([]byte("error"))
        return
    }
    result := response.Response.Result
    apiResp := struct {
        Result *string
    }{Result: result}
    b, _ := json.Marshal(apiResp)

    // ... 其他业务处理代码

    //步骤5：返回服务响应
    _, _ = w.Write(b)
}

func main() {
    // 注册http接口路径
    http.HandleFunc("/api/v1/get-token", GetFaceIdToken)
    http.HandleFunc("/api/v1/get-result", GetFaceIdResult)
    // 监听端口
    err := http.ListenAndServe(":8080", nil)
    if nil != err {
        log.Fatal("ListenAndServe error: ", err)
    }
}
```

3. 接口测试

当你完成了集成之后，可以通过postman或者curl命令来测试当前的集成是否正确，通过访问http://ip:port/api/v1/get-token 接口查看是否正常返回 `SdkToken` ，访问http://ip:port/api/v1/get-result 接口查看 `Result` 字段的响应是否为0以判断服务端集成是否成功，具体的响应结果详见API接口部分的介绍。

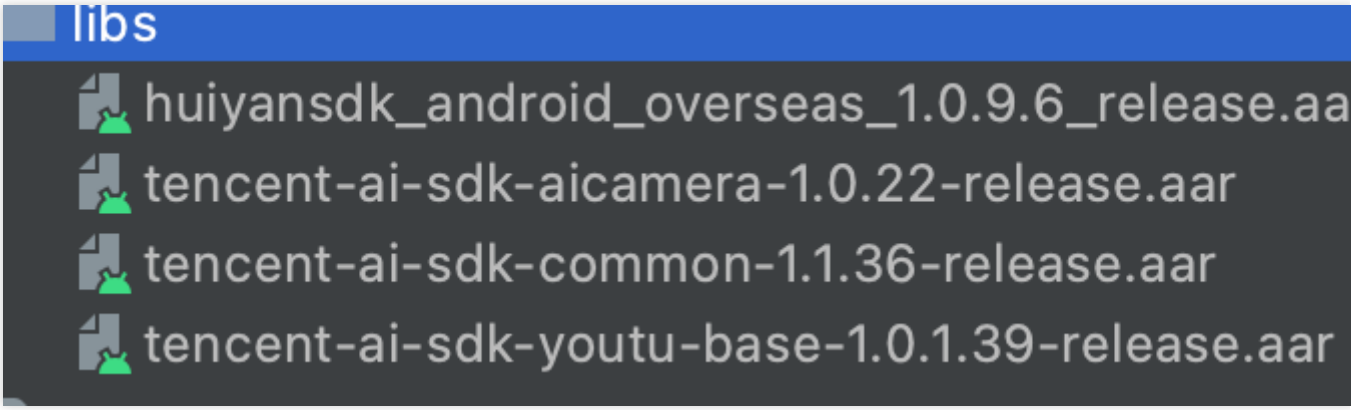
Android 端集成

1. 依赖环境

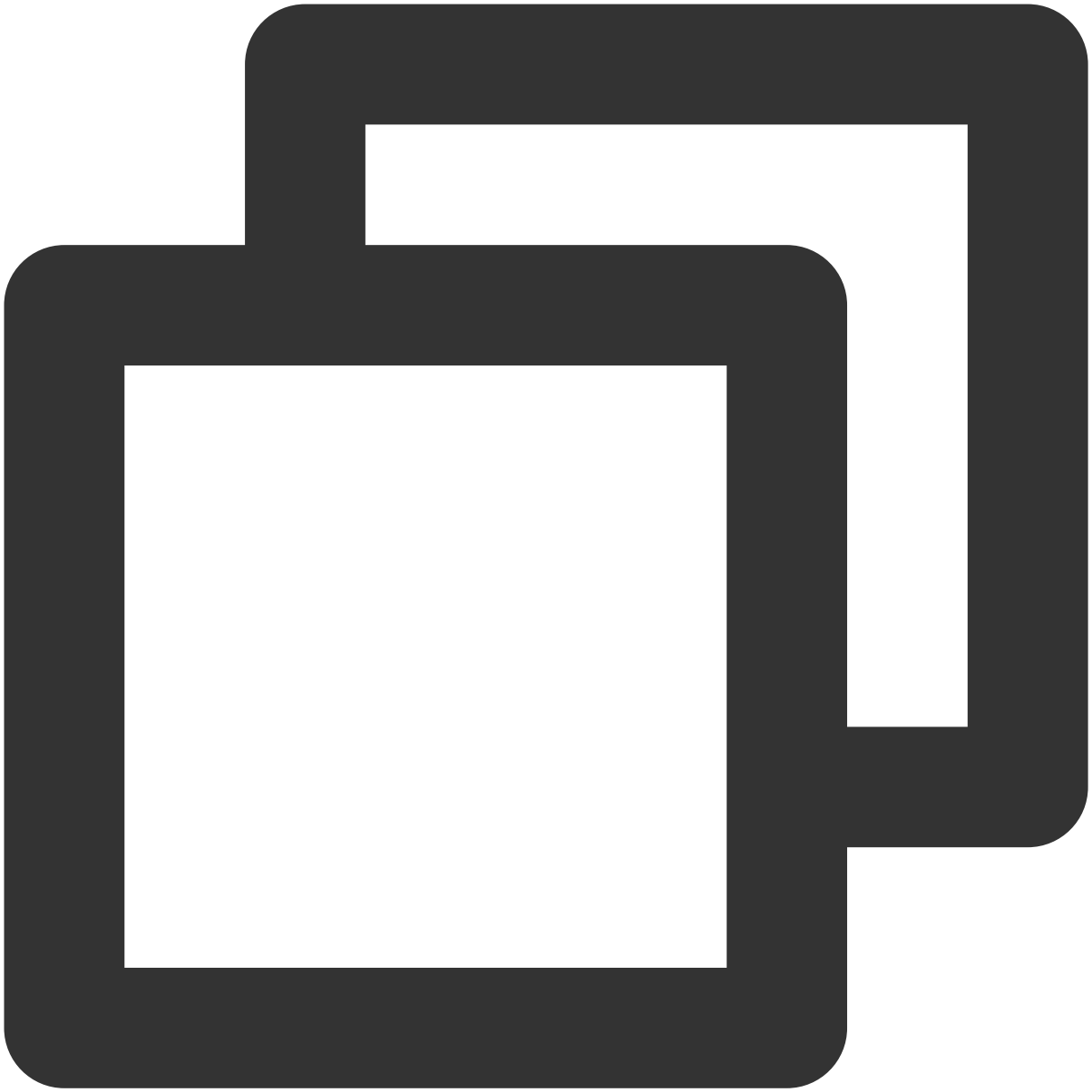
当前Android端慧眼SDK适用于API 19 (Android 4.4) 及以上版本。

2. SDK 接入步骤

1. 将**huiyansdk_android_overseas_1.0.9.6_release.aar**具体版本号以官网下载为准) 和 **tencent-ai-sdk-youtu-base-1.0.1.39-release.aar**、**tencent-ai-sdk-common-1.1.36-release.aar**、**tencent-ai-sdk-aicamera-1.0.22-release.aar**(具体版本号以最终提供为准) 添加到您工程的libs目录下。



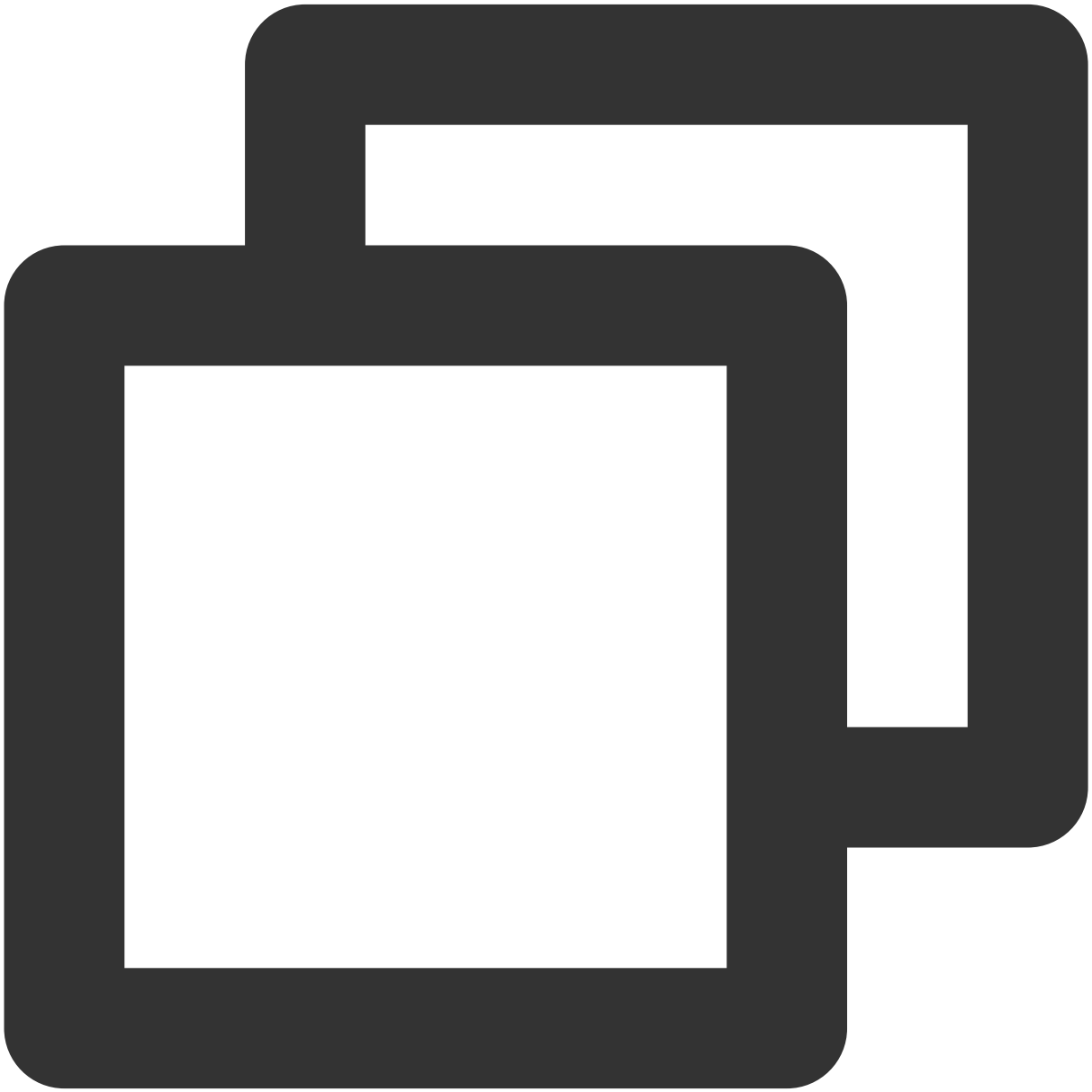
2. 在您工程的 **build.gradle** 中进行如下配置：



```
// 设置ndk so架构过滤(以armeabi-v7a为例)
ndk {
    abiFilters 'armeabi-v7a'
}

dependencies {
    // 引入慧眼SDK
    implementation files("libs/huiyansdk_android_overseas_1.0.9.5_release.aar")
    // 慧眼通用算法SDK
    implementation files("libs/tencent-ai-sdk-youtu-base-1.0.1.32-release.aar")
    // 通用能力组件库
```

```
    implementation files("libs/tencent-ai-sdk-common-1.1.27-release.aar")
    implementation files("libs/tencent-ai-sdk-aicamera-1.0.18-release.aar")

      // 慧眼SDK需要依赖的第三方库
    // gson
    implementation 'com.google.code.gson:gson:2.8.5'
}
```

3. 同时需要在AndroidManifest.xml文件中进行必要的权限声明



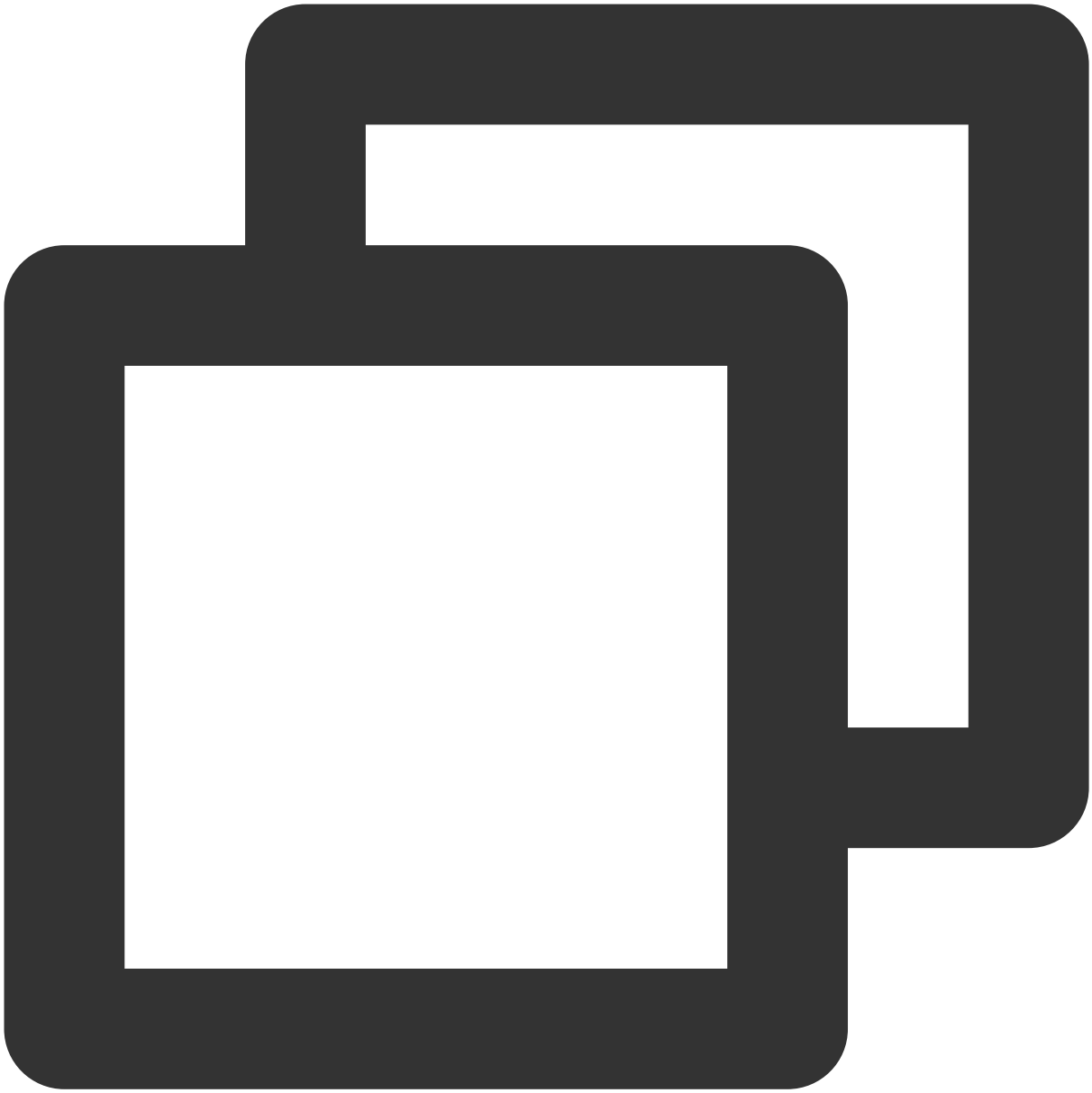
```
<!-- 摄像头权限 -->
<uses-permission android:name="android.permission.CAMERA" />
```

```
<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />
<uses-feature android:name="android.hardware.camera.autofocus" />

<!-- SDK需要的权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- SDK可选需要的权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

对于需要兼容Android 6.0以上的用户，以上权限除了需要在AndroidManifest.xml文件中声明权以外，还需使用代码动态申请权限。

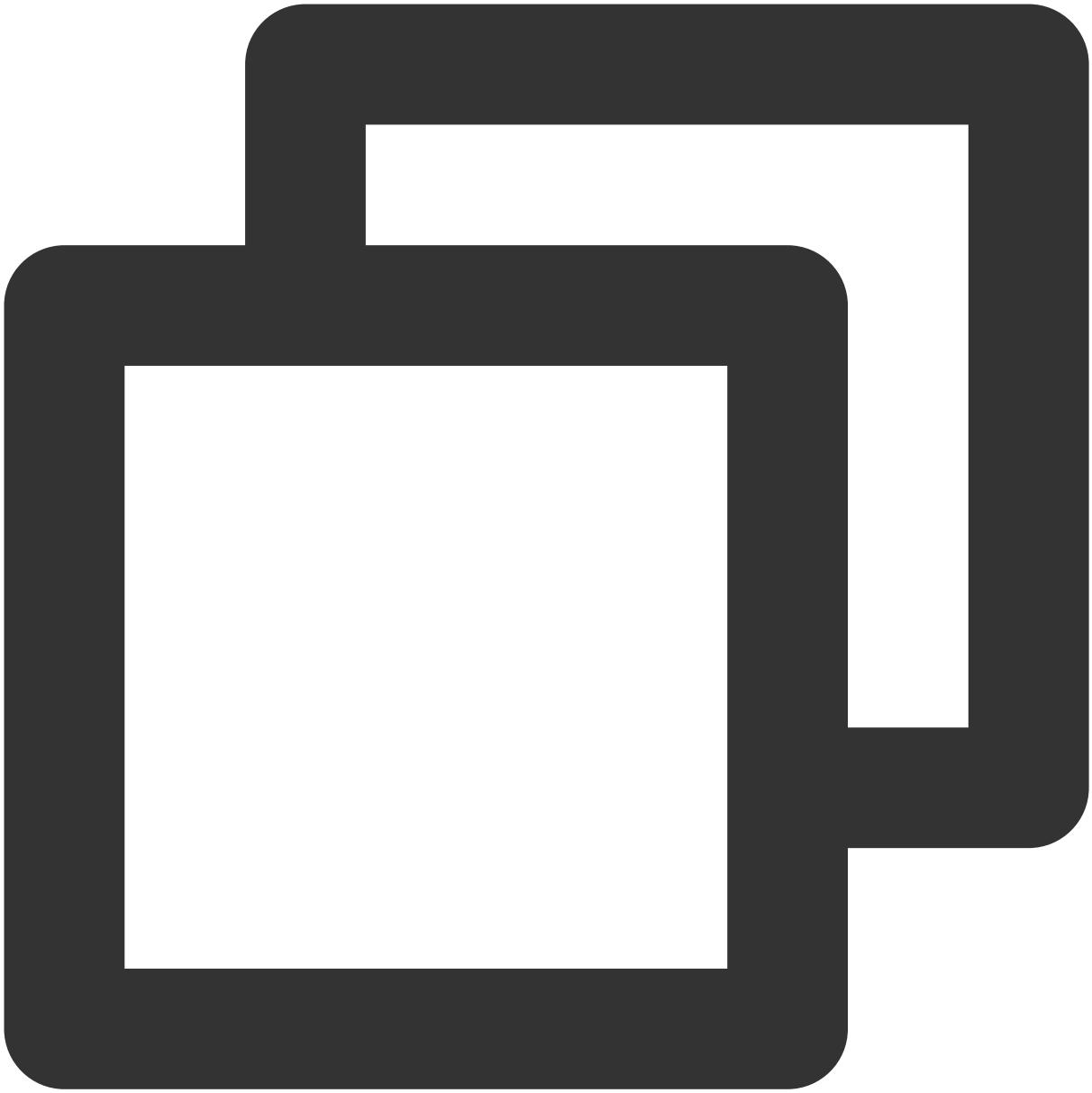
3. 初始化接口

在您APP初始化的时候调用，推荐在Application中调用，主要是进行一些SDK的初始化操作



```
@Override
public void onCreate() {
    super.onCreate();
    instance = this;
    // SDK需要在Application初始化时进行初始化
    HuiYanOsApi.init(getApp());
}
```

4. 启动核身检测接口

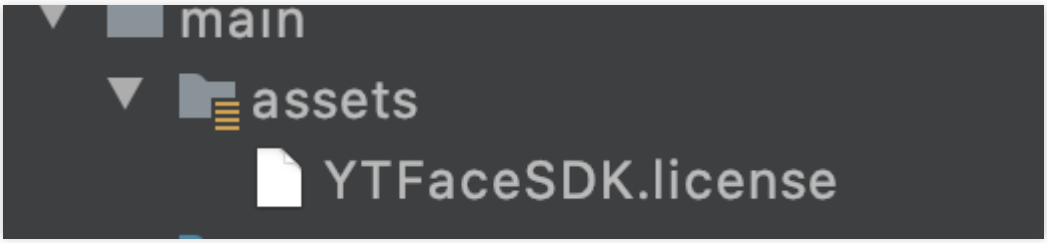


```
// HuiYanOs的相关参数
HuiYanOsConfig huiYanOsConfig = new HuiYanOsConfig();
// 此license文件存放在assets下
huiYanOsConfig.setAuthLicense("YTFaceSDK.license");
if (compatCheckBox.isChecked()) {
    huiYanOsConfig.setPageColorStyle(PageColorStyle.Dark);
}
// 是否需要返回最佳帧
if (needBestImageCB.isChecked()) {
    huiYanOsConfig.setNeedBestImage(true);
}
```

```
// 启动方法，开始的活体，currentToken为后台下发数据
HuiYanOsApi.startHuiYanAuth(currentToken, huiYanOsConfig, new HuiYanOsAuthCallBack(
    @Override
    public void onSuccess(HuiYanOsAuthResult authResult) {
        showToast("活体通过！");
        if (!TextUtils.isEmpty(authResult.getBestImage())) {
           CommonUtils.decryptBestImgBase64(authResult.getBestImage(), false);
        }
    }

    @Override
    public void onFail(int errorCode, String errorMsg, String token) {
        String msg = "活体失败 " + "code: " + errorCode + " msg: " + errorMsg + " to
        Log.e(TAG, "onFail" + msg);
        showToast(msg);
    }
});
```

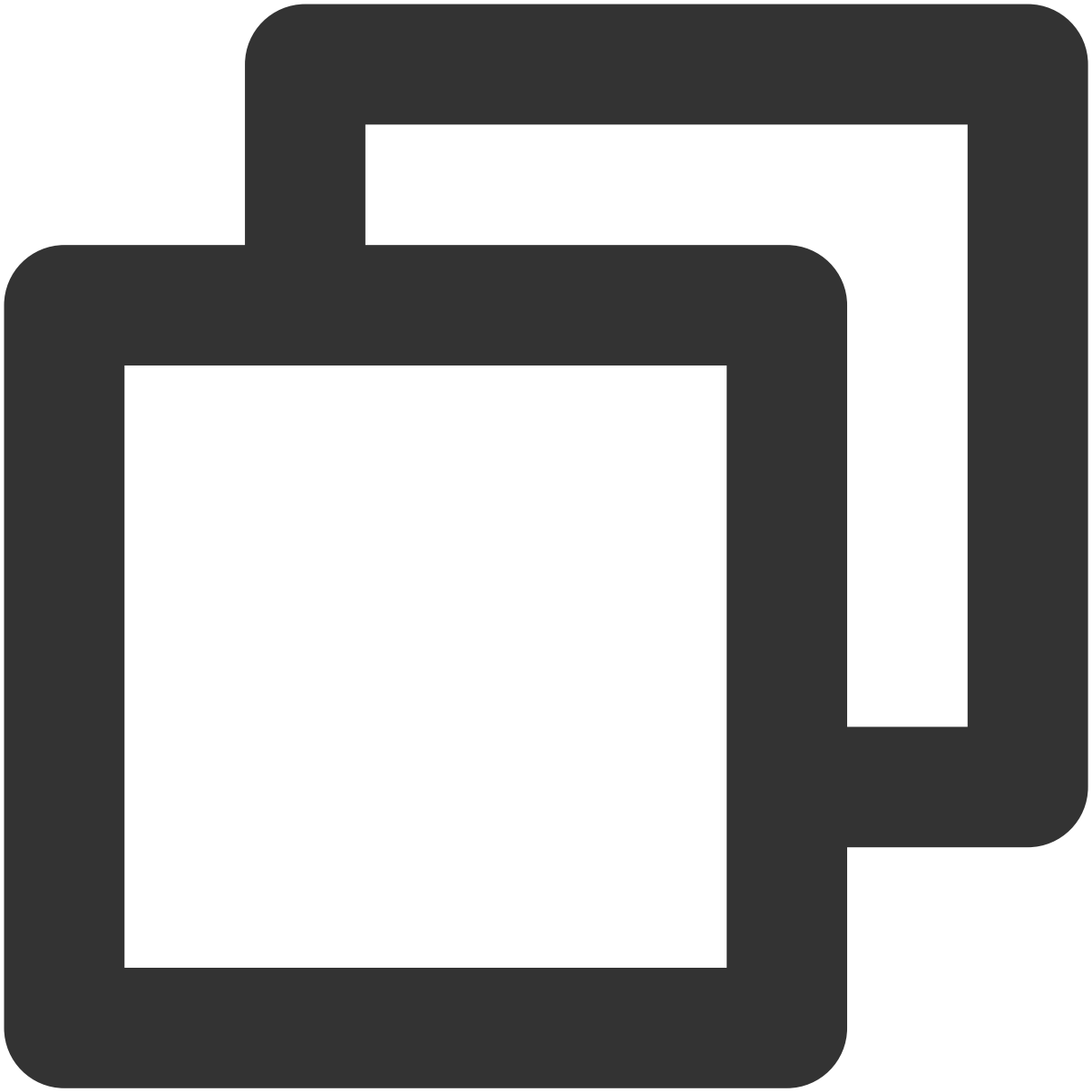
HuiYanOsAuthResult为活体成功的返回结果, 最终的核身结果可以通过token，访问GetFaceIdResultIntl获取。

注意：当前的"YTFaceSDK.license"文件是需要您主动申请的，暂时您可以联系客服人员进行license申请。将申请完成后的license文件放到assets文件下。



5. SDK 资源释放

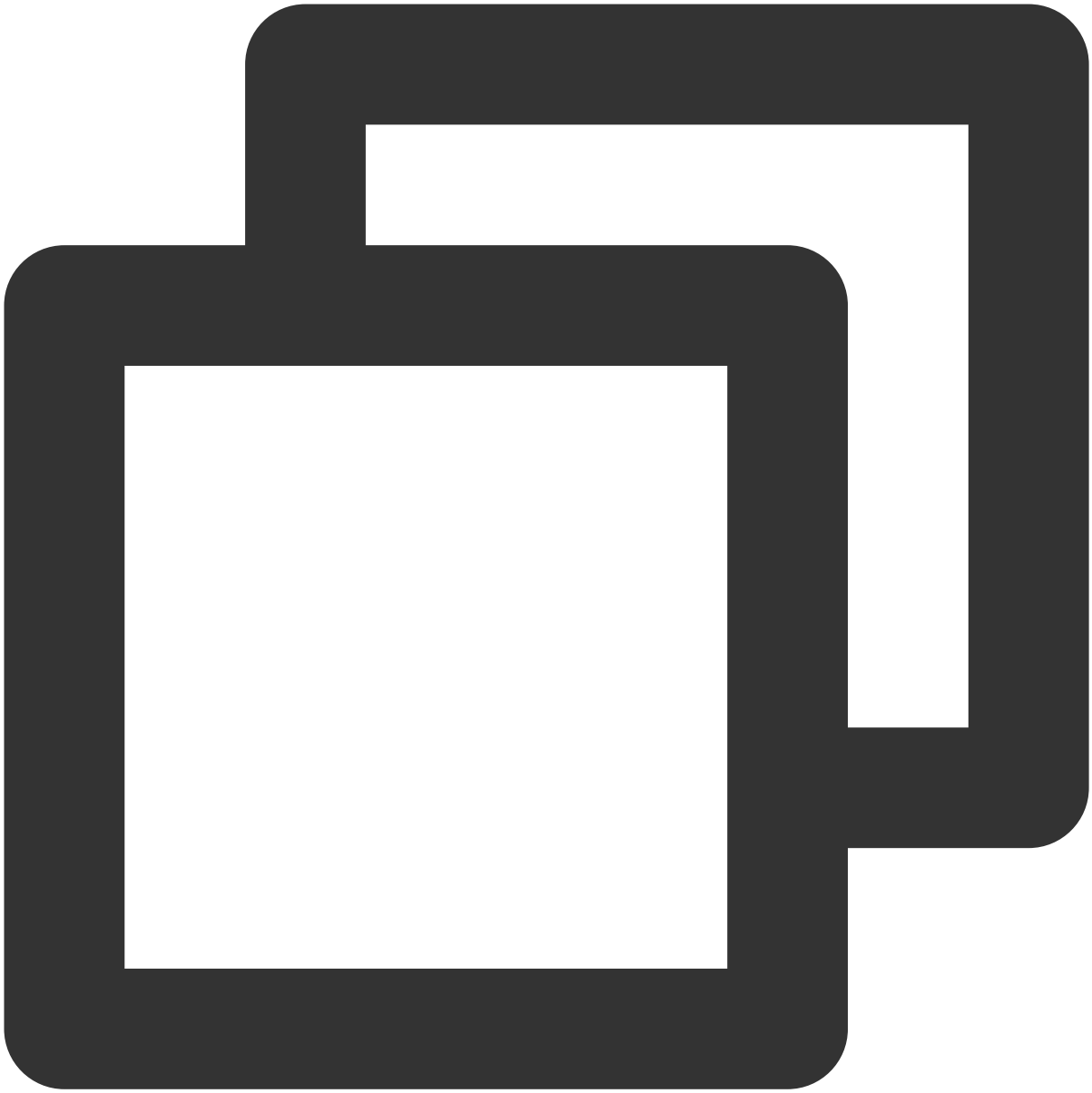
在您APP退出使用的时候，可以调用SDK资源释放接口



```
@Override
protected void onDestroy() {
    super.onDestroy();
    // 退出时做资源释放
    HuiYanOsApi.release();
}
```

6. 混淆规则配置

如果您的应用开启了混淆功能，为确保SDK的正常使用，请把以下部分添加到您的混淆文件中。



```
#慧眼SDK的混淆包含
-keep class com.tencent.could.huiyansdk.** {*;}
-keep class com.tencent.could.aicamare.** {*;}
-keep class com.tencent.could.component.** {*;}
-keep class com.tencent.youtu.** {*;}
-keep class com.tenpay.utils.SMUtils {*;}
```

iOS 端集成

1. 依赖环境

1. 开发环境 Xcode 11.0 或以上
2. 慧眼iOS SDK 适用于手机iOS9.0及以上版本

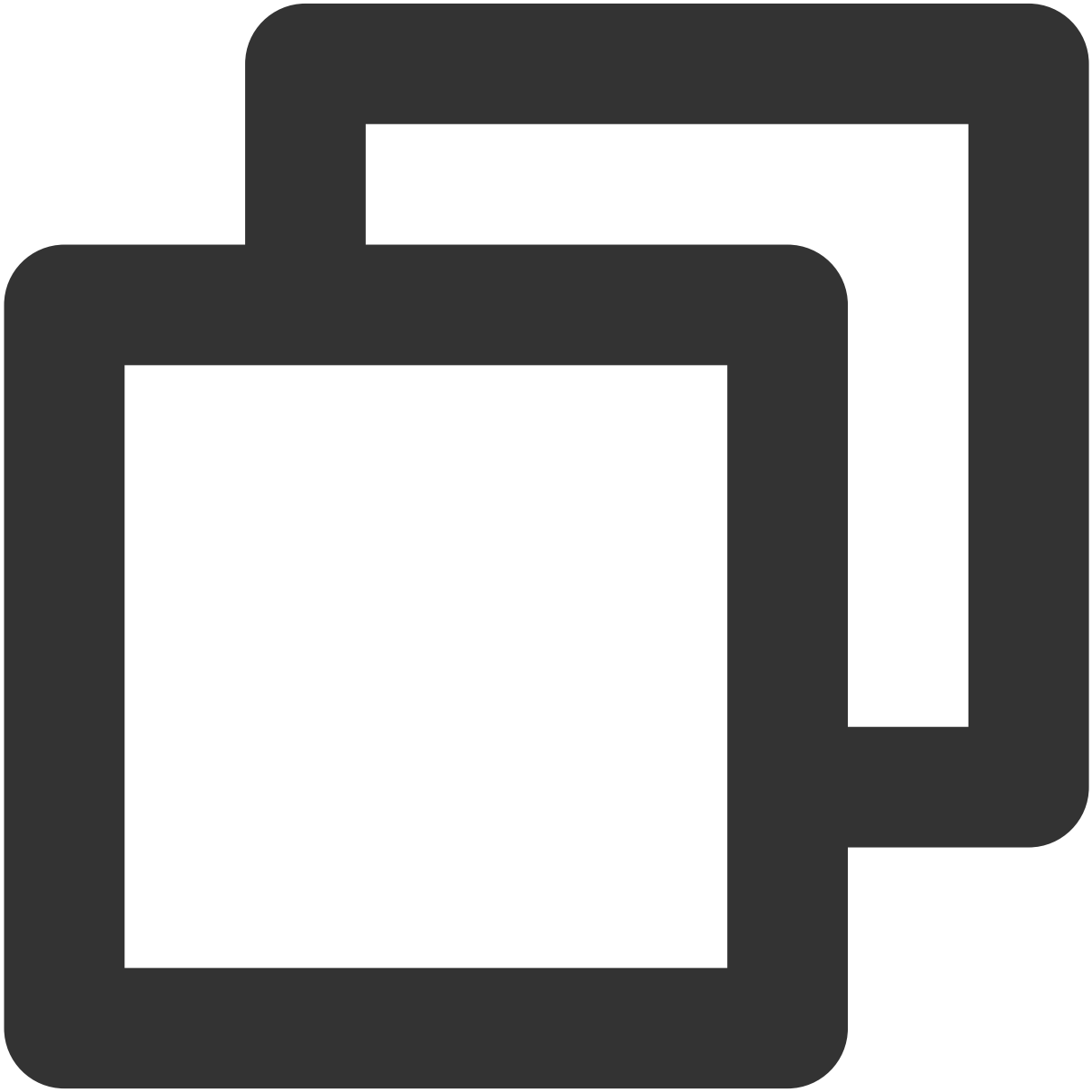
2. SDK接入步骤

手动接入方式

1. 导入相关库及文件

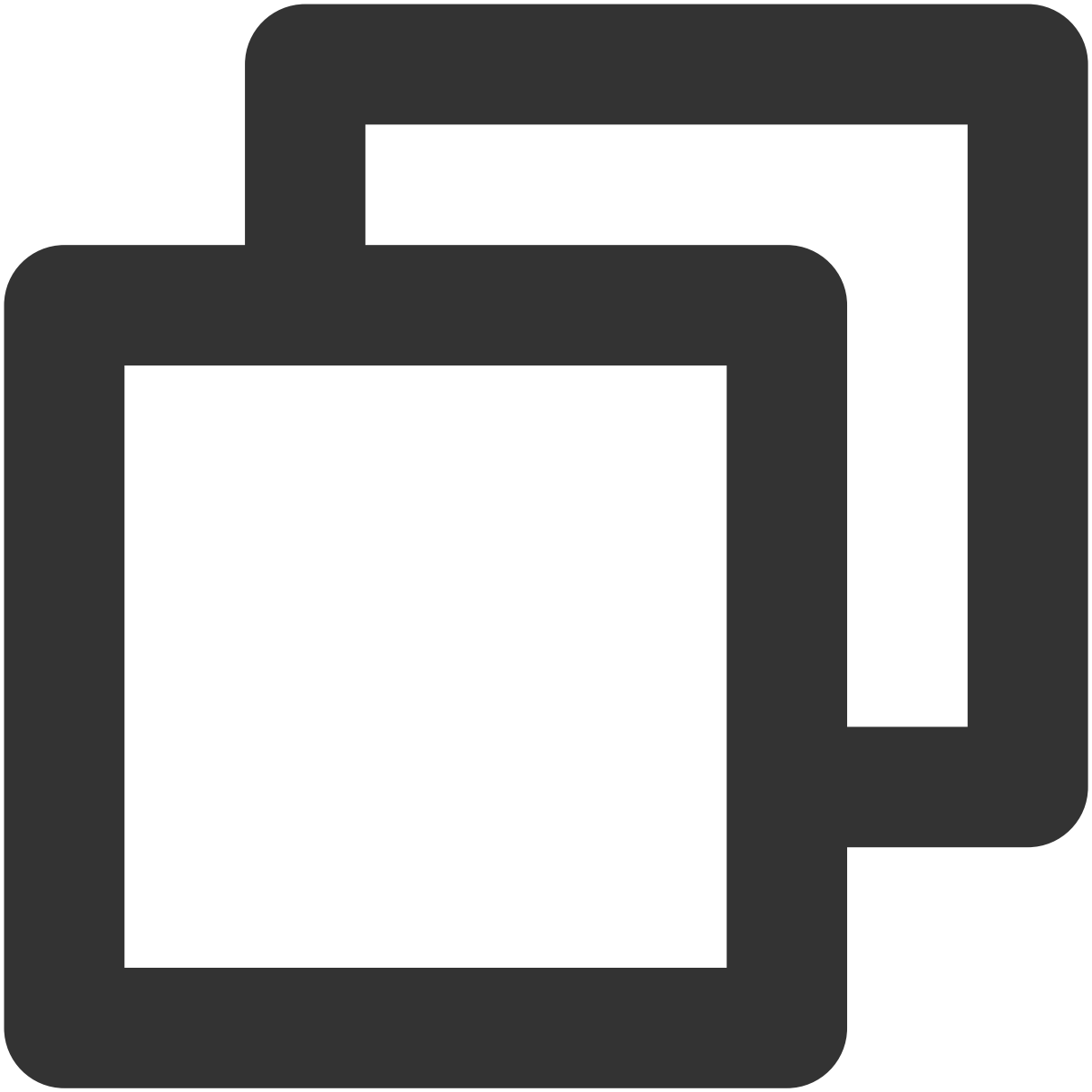
Link Binary With Libraries导入相关Framework

2. SDK依赖的库如下



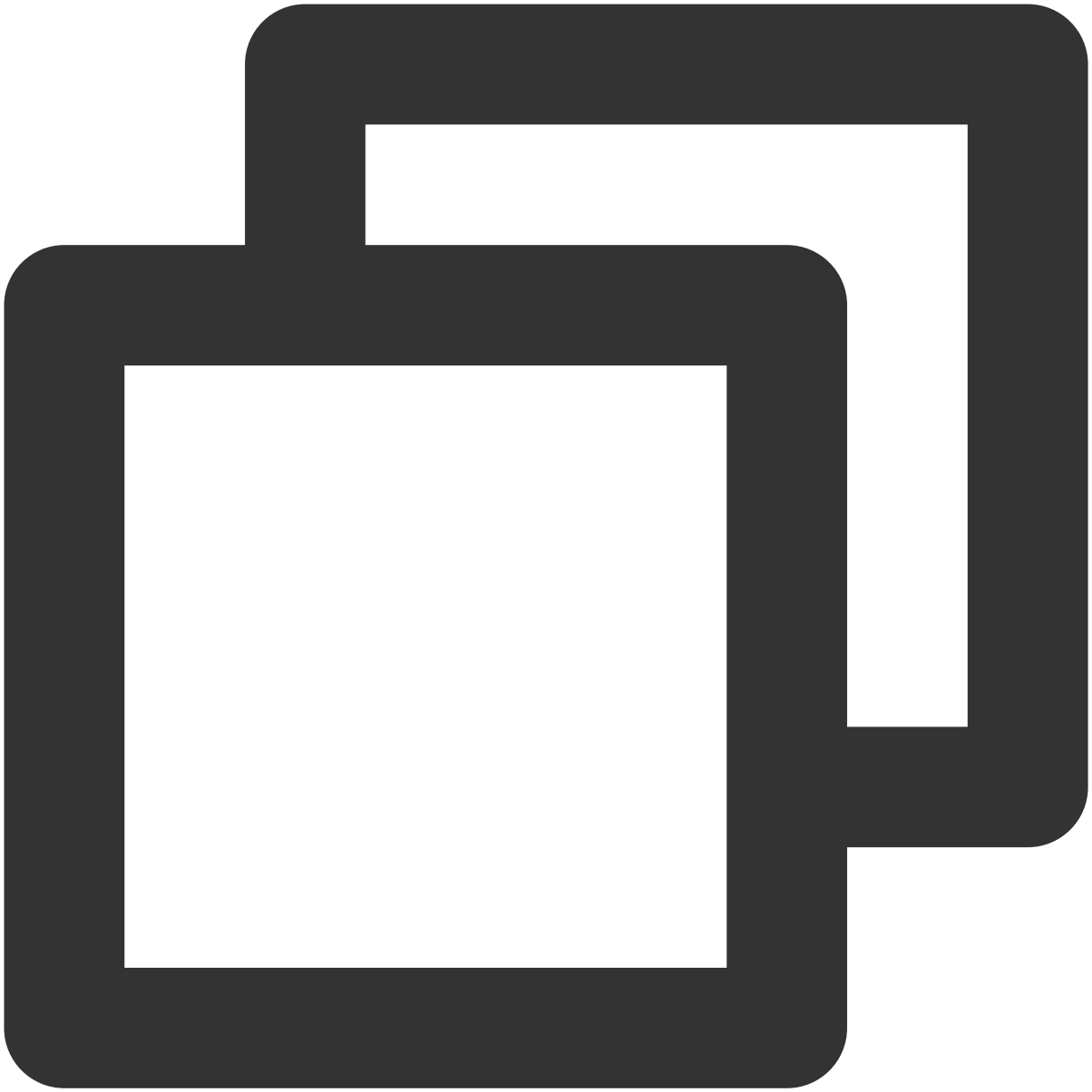
```
├──HuiYanSDK.framework
├──YtSDKKitSilentLiveness.framework
├──YtSDKKitReflectLiveness.framework
├──YtSDKKitActionLiveness.framework
├──YtSDKKitFramework.framework
├──tnnliveness.framework
├──YTFaceAlignmentTinyLiveness.framework
├──YTFaceTrackerLiveness.framework
├──YTFaceDetectorLiveness.framework
├──YTPoseDetector.framework
├──YTCommonLiveness.framework
└──YTFaceLiveReflect.framework
```

3. Link Binary With Libraries导入系统Framework



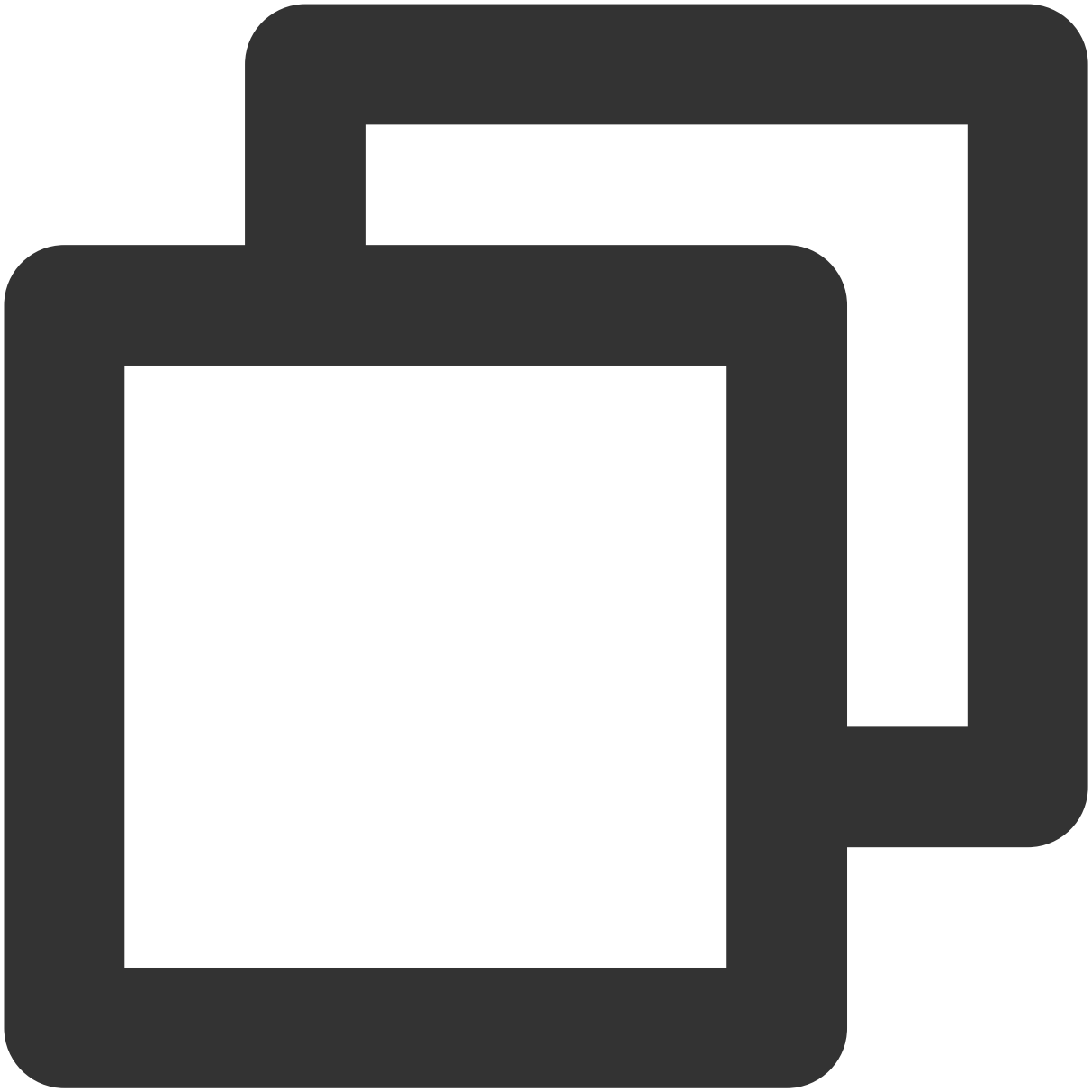
```
├── AVFoundation.framework
├── libc++.tbd
└── Accelerate.framework
```

4. Copy Bundle Resources中导入模型



```
└── face-tracker-v001.bundle
```

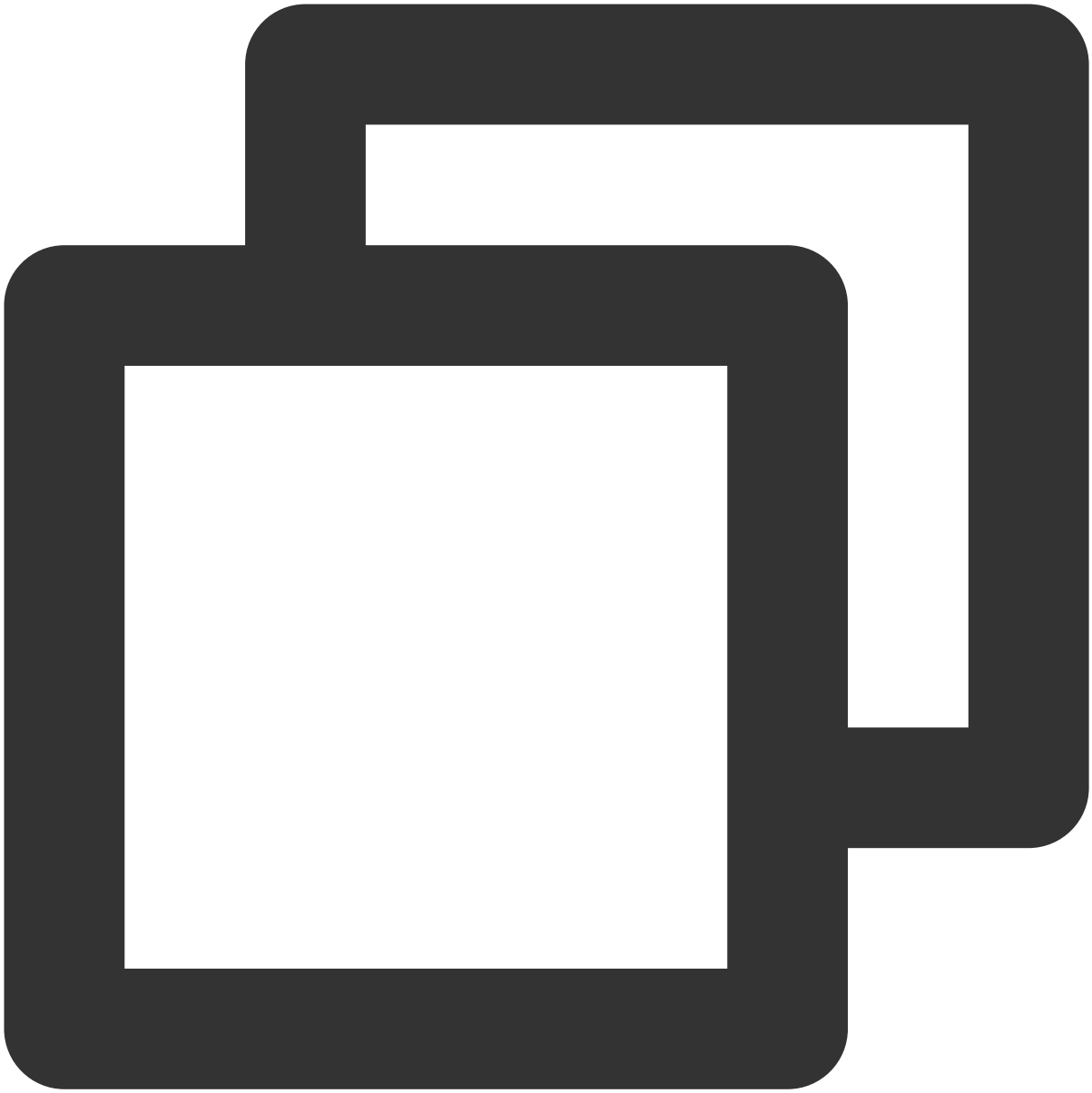
5. Copy Bundle Resources导入资源文件



```
└── HuiYanSDKUI.bundle
```

使用Pod方式接入

1. 将CloudHuiYanSDK_FW文件夹复制到集成项目Podfile同级目录下
2. 在Podfile设置



```
target 'HuiYanAuthDemo' do
  use_frameworks!
  pod 'CloudHuiYanSDK_FW', :path => './CloudHuiYanSDK_FW'
end
```

3. pod install 更新

说明：

文件层级和具体的设置可以参考 demo。

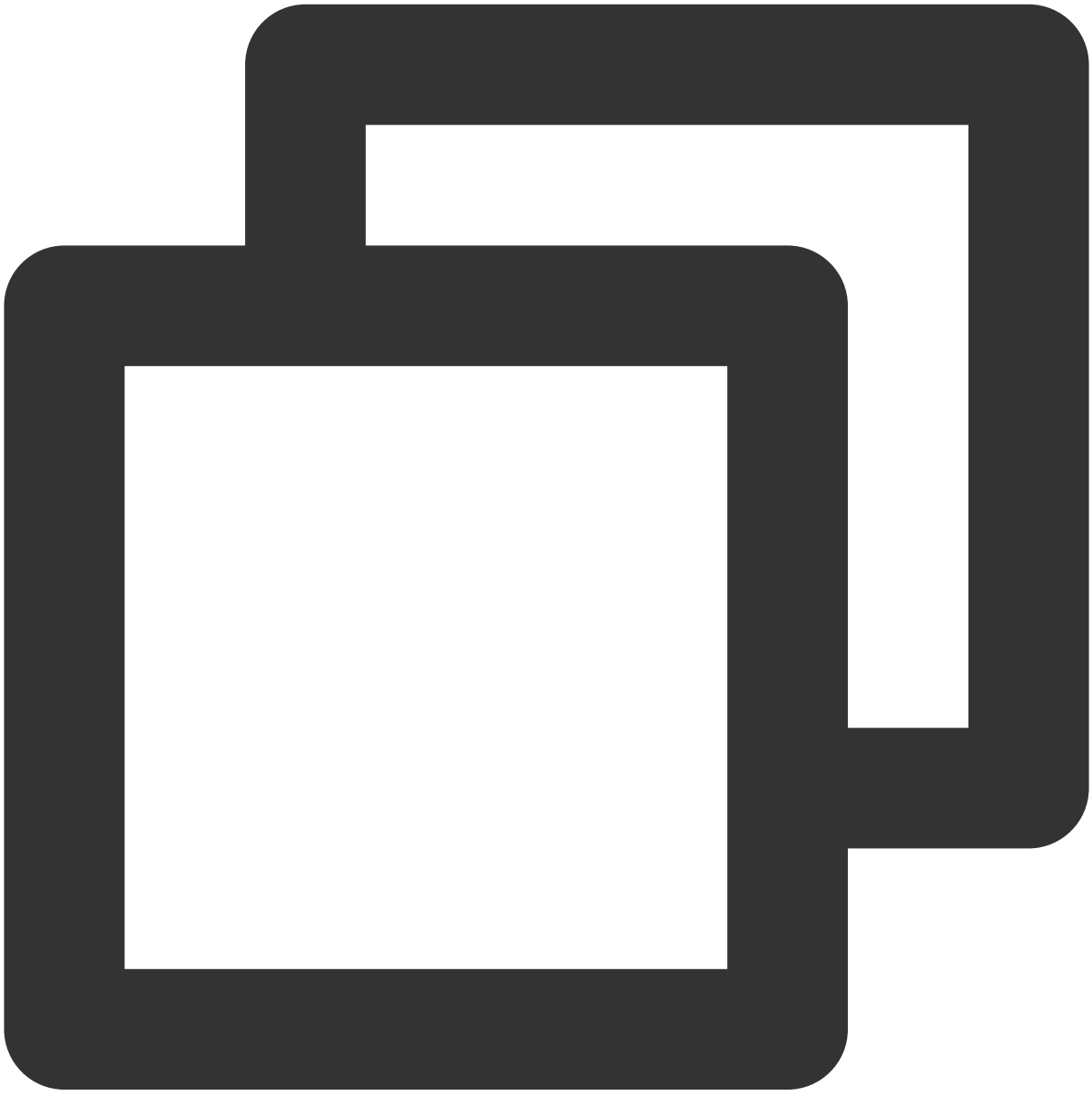
iOS demo

Build Phases设置

1. Other Linker Flags 新增 **-ObjC**
2. 接入ViewController.m 设置后缀为.mm(swift 工程添加系统库libc++.tbd)

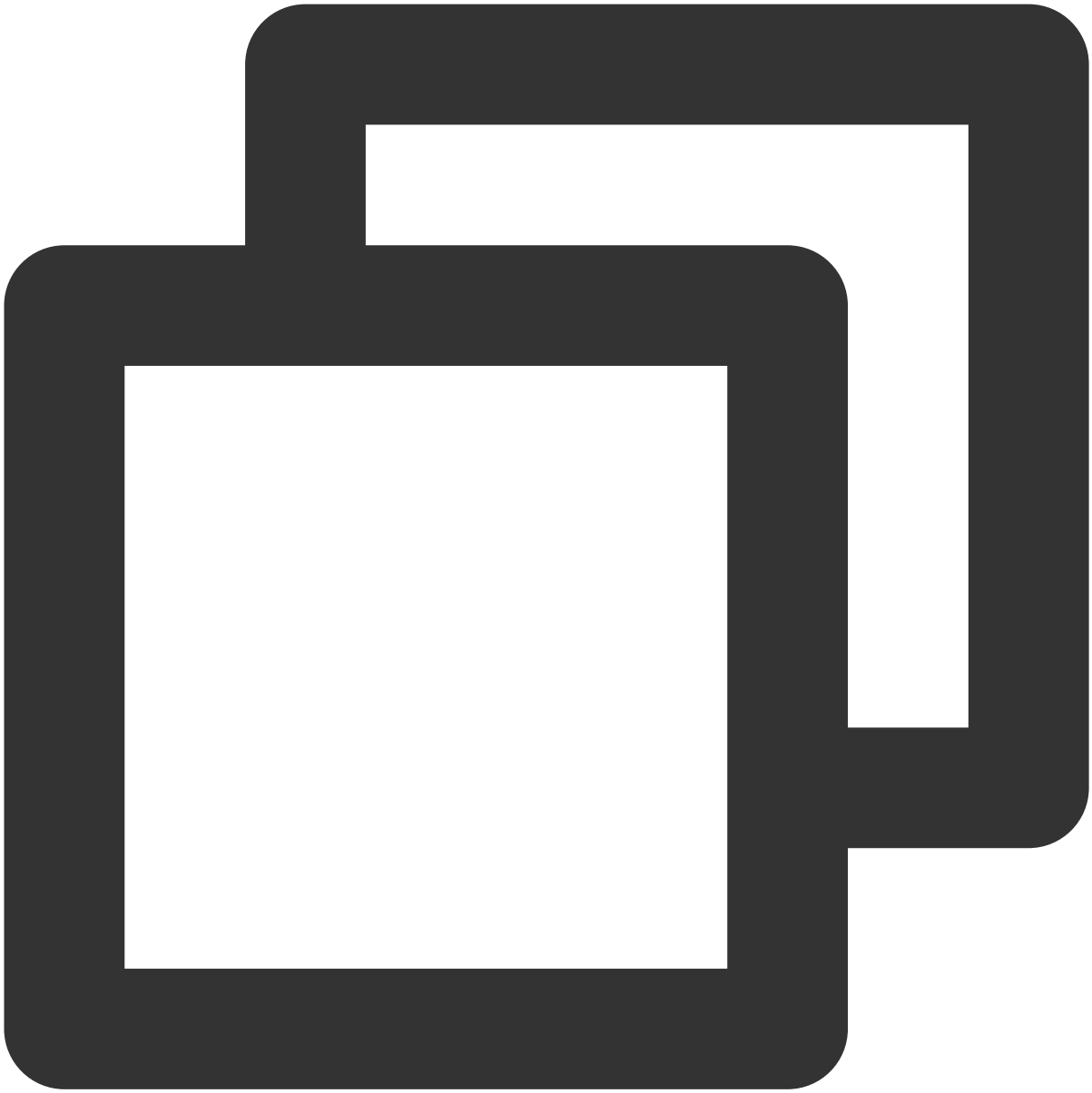
权限设置

SDK需要手机网络及 摄像头使用权限，请添加对应的权限声明。在主项目info.plist 配置中添加下面key-value值



```
<key>Privacy - Camera Usage Description</key>
<string>人脸核身需要开启您的摄像头权限，用于人脸识别</string>
```

3. 开启活体检测接口



```
#import <HuiYanSDK/HuiYanOsApi.h>
#import <HuiYanSDK/HuiYanOSKit.h>

  //获取token
    NSString *faceToken = self.tokenTextField.text;
    // 配置SDK
    HuiYanOsConfig *config = [[HuiYanOsConfig alloc] init];
    //设置lic
    config.authLicense = [[NSBundle mainBundle] pathForResource:@"xxx.lic" ofType:@
```

```
    //准备阶段超时配置
    config.prepareTimeoutMs = 20000;
    //检测阶段超时配置
    config.authTimeOutMs = 20000;
    config.isDeleteVideoCache = YES;
    config.languageType = EN;
    //    config.userLanguageFileName = @"ko";
    //    config.userLanguageBundleName = @"UseLanguageBundle";
    config.iShowTipsPage = YES;
    config.isGetBestImg = YES;

    [[HuiYanOSKit sharedInstance] startHuiYaneKYC:faceToken withConfig:config
                                  witSuccCallback:^(HuiYanOsAuthResult * _Nonnull a
        NSString *bestImg = authResult.bestImage;
        NSString *token = authResult.faceToken;

    } withFailCallback:^(int errCode, NSString * _Nonnull errMsg, id  _Nullable res
        NSString *showMsg = [NSString stringWithFormat:@"err:%d:%@",errCode,errMsg]
        NSLog(@"err:%@",showMsg);
    }];
```

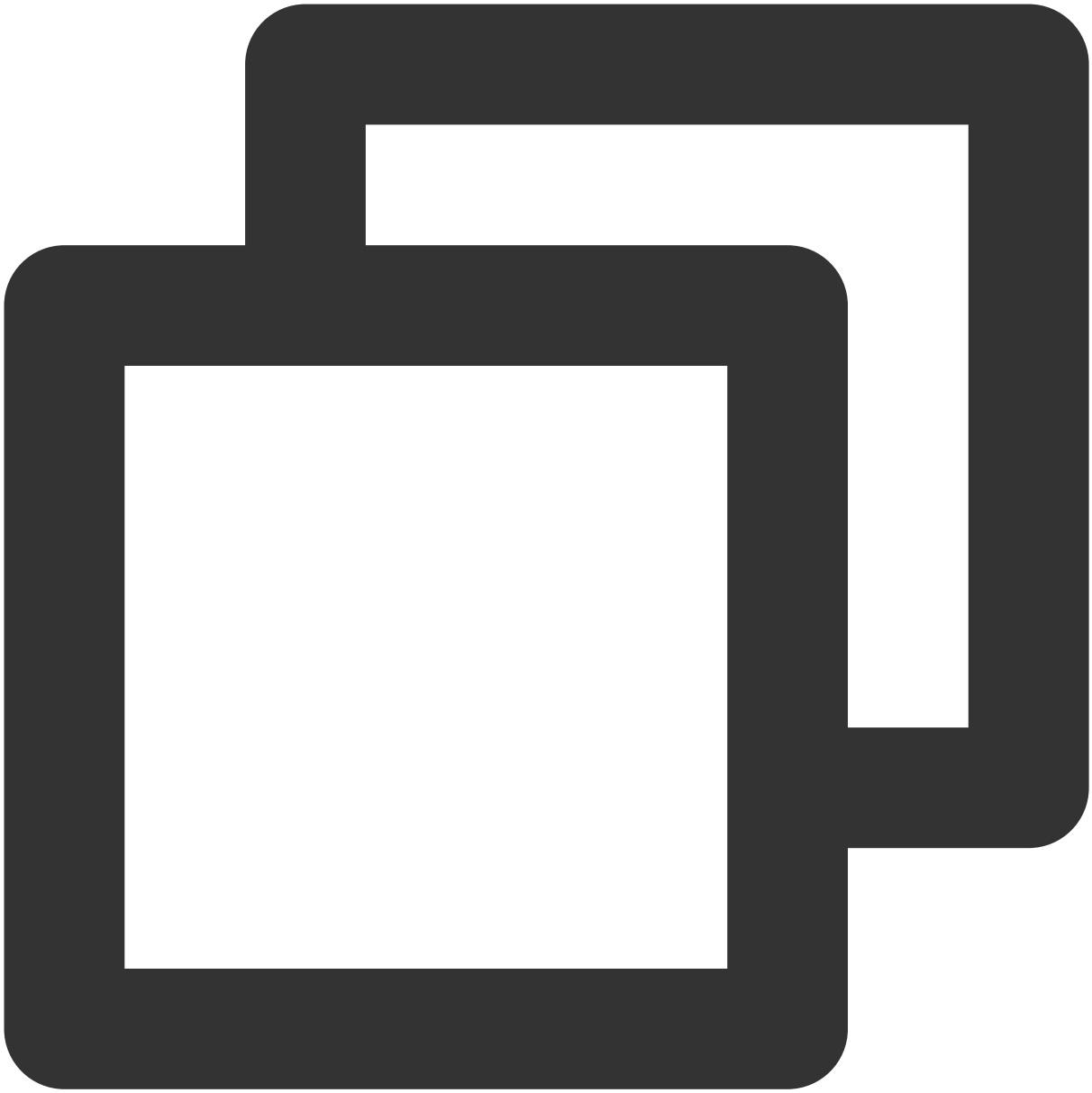
HuiYanOsAuthResult为活体成功的返回结果, 最终的核身结果可以通过token，访问GetFaceIdResultIntl获取。

注意：

当前的"xxx.lic"文件是需要您主动申请的，暂时您可以联系客服人员进行license申请

4. SDK 资源释放

在您APP退出使用的时候，可以调用SDK资源释放接口



```
// 退出时做资源释放
- (void)dealloc {
    [HuiYanOsApi release];
}
//[iOS demo](https://github.com/TencentCloud/huiyan-faceid-demo/tree/main/faceid-iO
[Android demo](https://github.com/TencentCloud/huiyan-faceid-demo/tree/main/id-veri
```

iOS demoAndroid demo

SDK接口API概述
Android 接口概述

最近更新时间：2023-05-09 14:35:04

API的详细说明

慧眼SDK主要涉及如下几个类，它们分别是API的接口类HuiYanOsApi， 参数配置类HuiYanOsConfig，结果回调类HuiYanOsAuthCallBack和HuiYanAuthEventCallBack。

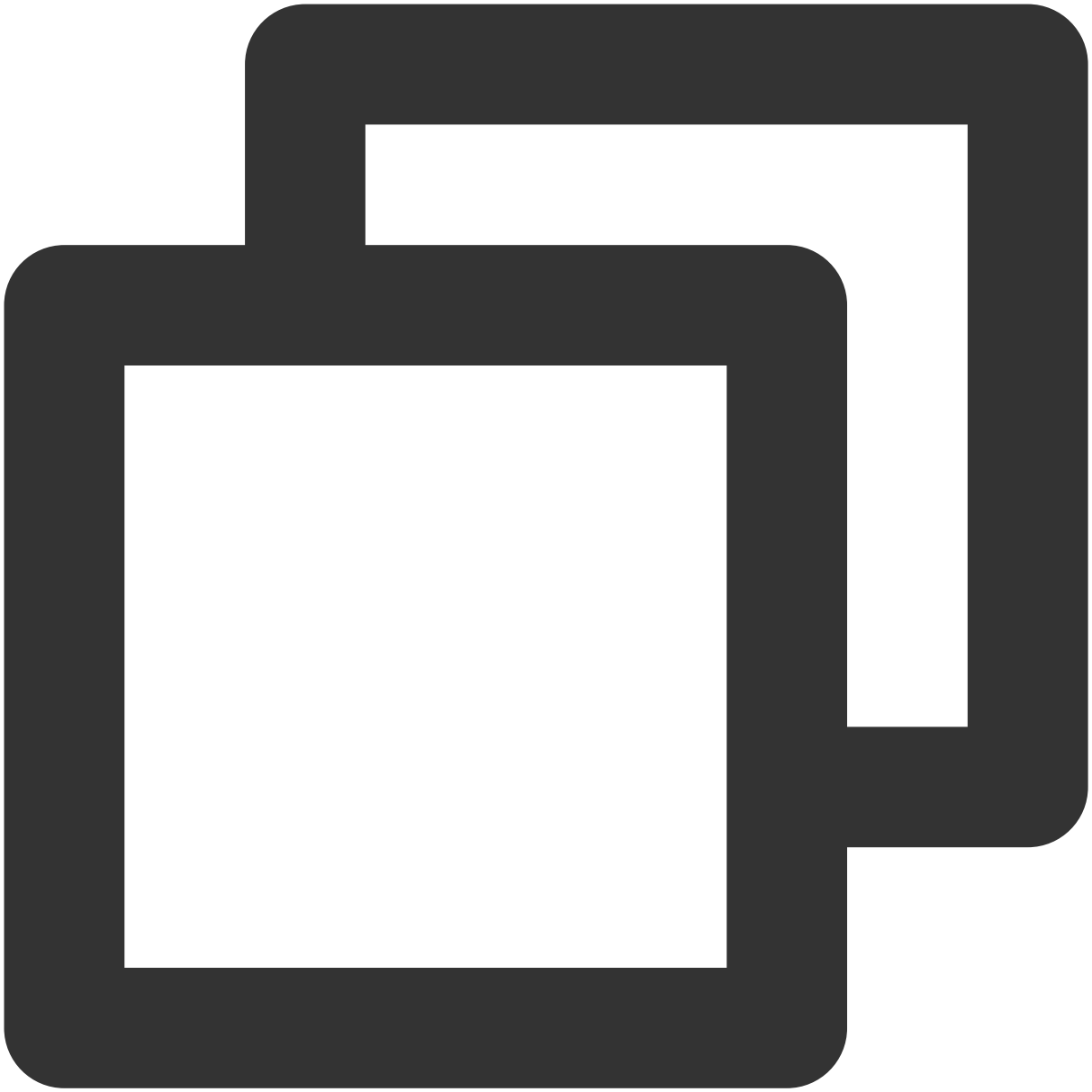
HuiYanOsApi

API	功能描述
init()	初始化接口
release()	资源释放接口
setAuthEventCallBack()	设置核身过程中的核身关键动作的回调
startHuiYanAuth()	海外核身接口，只需调用此接口即可完成整体核身流程。

init()

功能介绍：

慧眼SDK的初始化接口。



```
public static void init(Context context)
```

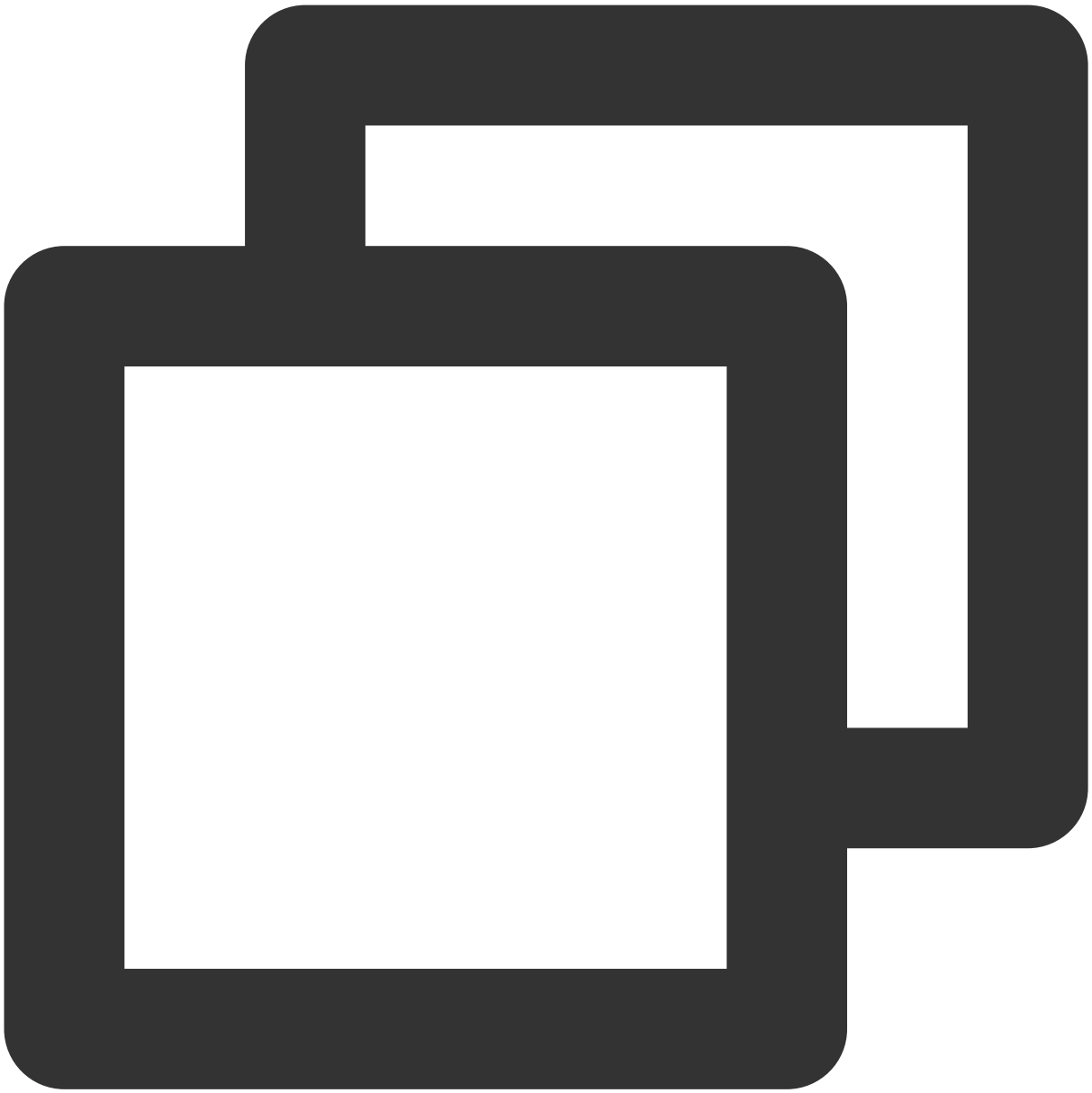
传入参数:

参数类型	参数名称	参数含义
Context	context	App的上下文信息

release()

功能介绍：

慧眼SDK资源释放的接口。

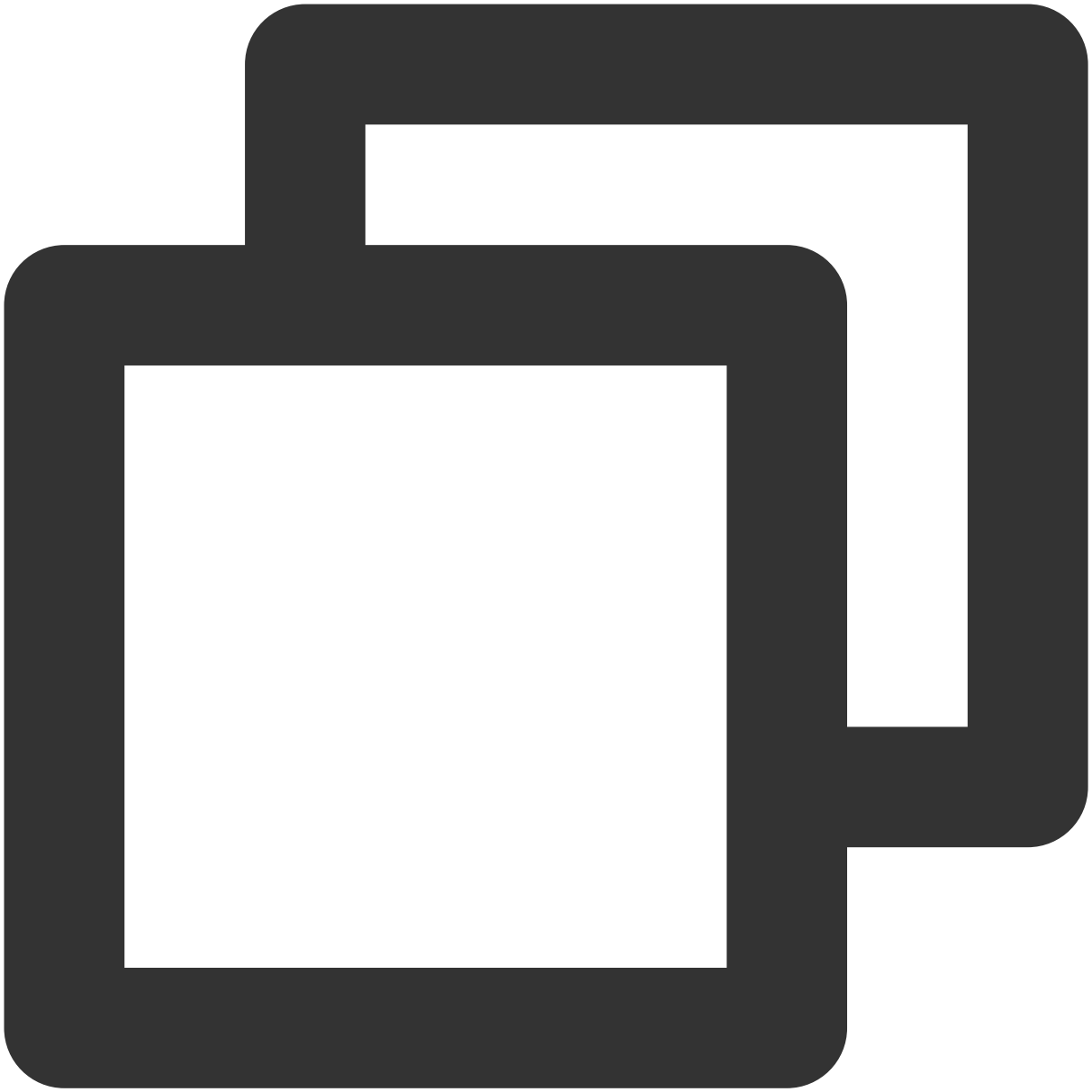


```
public static void release()
```

setAuthEventCallBack()

功能介绍：

用来注册核身过程中的核身关键动作的回调。



```
public static void setAuthEventCallBack(HuiYanAuthEventCallBack authEventCallBack)
```

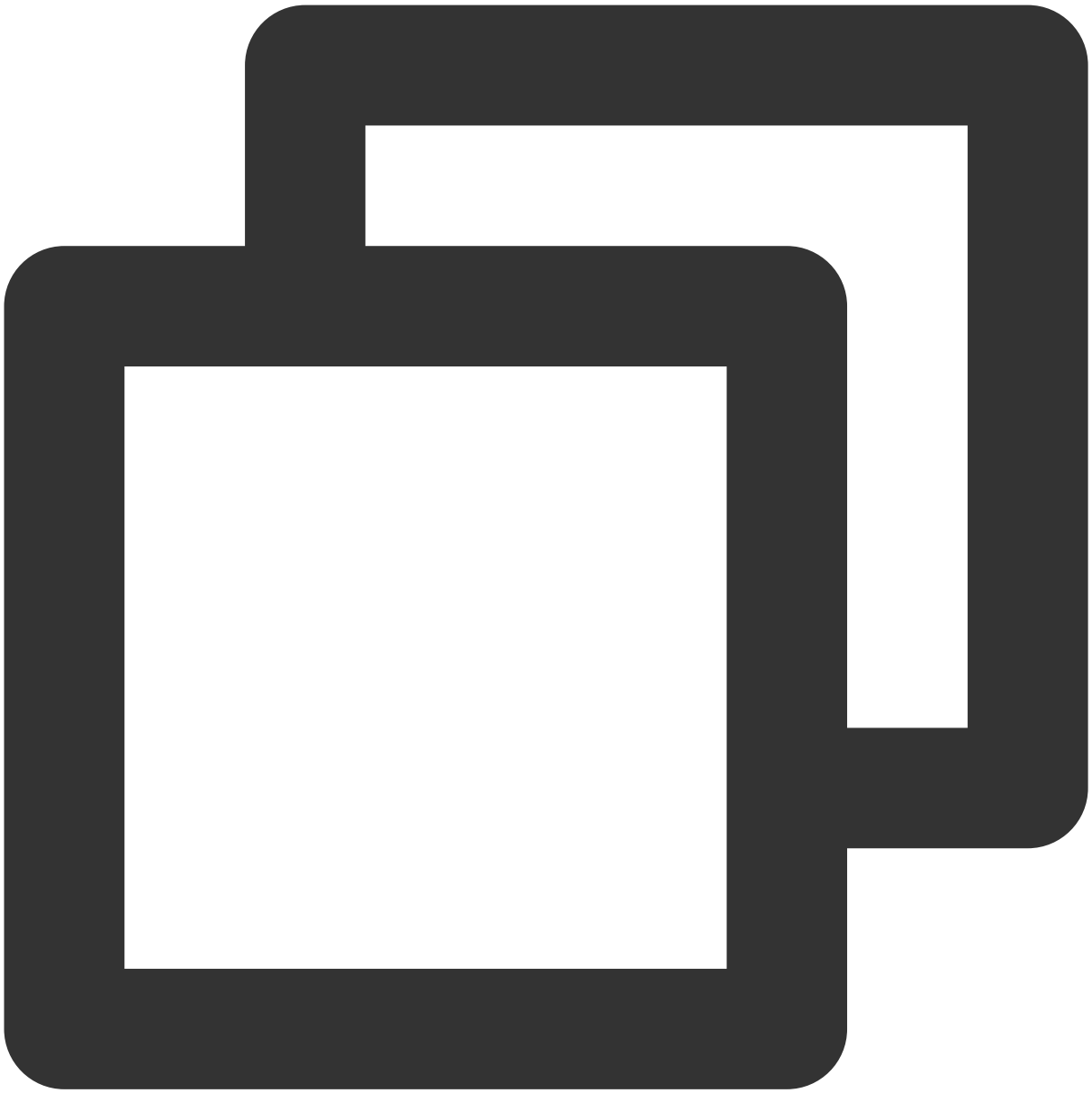
传入参数：

参数类型	参数名称	参数含义
HuiYanAuthEventCallBack	huiYanAuthEventCallBack	核身关键动作的回调

startHuiYanAuth()

功能介绍：

海外核身接口，只需调用此接口即可完成整体核身流程。



```
public static void startHuiYanAuth(final String startToken, final HuiYanOsConfig st
```

传入参数：

参数类型	参数名称	参数含义
String	startToken	从服务器兑换来启动核身使用的业务Token
HuiYanOsConfig	startConfig	配置的参数

HuiYanOsAuthCallBack	authCallBack	活体结果的回调

HuiYanOsConfig

HuiYanOsConfig是在启动慧眼SDK时的配置实体类，主要包含了以下属性。

类型	名称	含义	默认值
PageColorStyle	pageColorStyle	此次人脸核身检测的配色	PageColorStyle.Light
String	authLicense	客户申请的用户核审授权的Licens文件名	空
long	authTimeOutMs	设置活体检测的超时时间	10000毫秒（10秒）
boolean	isDeleteVideoCache	是否删除核身视频的本地缓存	true
boolean	isShowGuidePage	是否打开核身的引导页	true
boolean	isNeedBestImage	是否需要返回最佳帧	false
LanguageStyle	languageStyle	语言类型	跟随系统语言
String	languageCode	语言码（见附录），配合languageStyle一起使用	空
String[]	backUpIPs	备用IP列表	空
String	backUpHost	备用域名	空
AuthUiConfig	authUiConfig	UI配置项	空

PageColorStyle

默认核身界面默认配色的枚举类，当前主要包括了两种配色，白色系与黑暗色系。

PageColorStyle 类型	含义
PageColorStyle.Light	亮色调配色
PageColorStyle.Dark	暗色调配色

LanguageStyle

LanguageStyle 类型	含义
LanguageStyle.AUTO	跟随系统
LanguageStyle.ENGLISH	英语

LanguageStyle.SIMPLIFIED_CHINESE	简体中文
LanguageStyle.TRADITIONAL_CHINESE	繁体中文
LanguageStyle.CUSTOMIZE_LANGUAGE	自定义语言

AuthUiConfig

核身页面自定义UI的参数配置

类型	名称	含义	默认值
VideoSize	videoSize	核身过程中的分辨率	480P
boolean	isShowCountdown	是否显示倒计时的控件	true
boolean	isShowErrorDialog	是否显示错误的dialog	true
int	authLayoutResId	核身导出客户自定义的布局resID，不调整使用-1	-1
int	feedBackErrorColor	异常反馈Tips的颜色（0xFFFFFFFF的类型）不调整使用-1	-1
int	feedBackTxtColor	正常反馈Tips颜色（0xFFFFFFFF的类型）不调整使用-1	-1
int	authCircleErrorColor	动作错背景圆形框的颜色（0xFFFFFFFF的类型）不调整使用-1	-1
int	authCircleCorrectColor	动作正确时背景圆形框的颜色（0xFFFFFFFF的类型）不调整使用-1	-1
int	authLayoutBgColor	核身界面背景的颜色（0xFFFFFFFF的类型）不调整使用-1	-1

VideoSize

慧眼核身支持的分辨率枚举

VideoSize类型	含义
VideoSize.SIZE_480P	480P
VideoSize.SIZE_720P	720P

HuiYanOsAuthResult

核身流程的成功回调对应的结果类型。

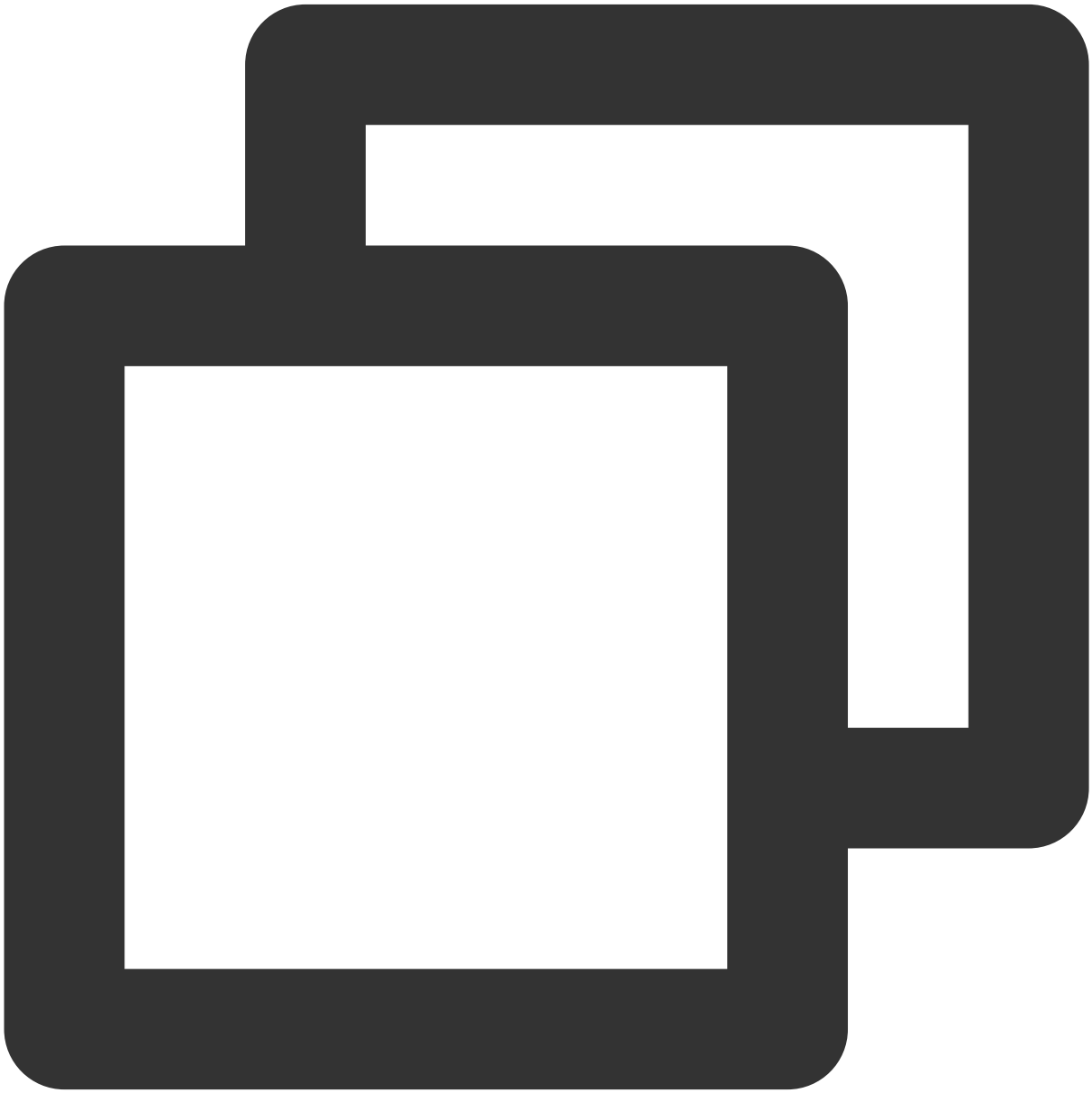
类型	名称	含义	默认值

String	token	此次活体流程中使用的token	空
String	bestImage	活体最佳帧图片的Base64数据	空

最后需要通过这个token，去腾讯云API的后台接口GetFaceIdResultIntl拉取最终活体是否成功的数据。

HuiYanOsAuthCallBack

核身流程的回调接口

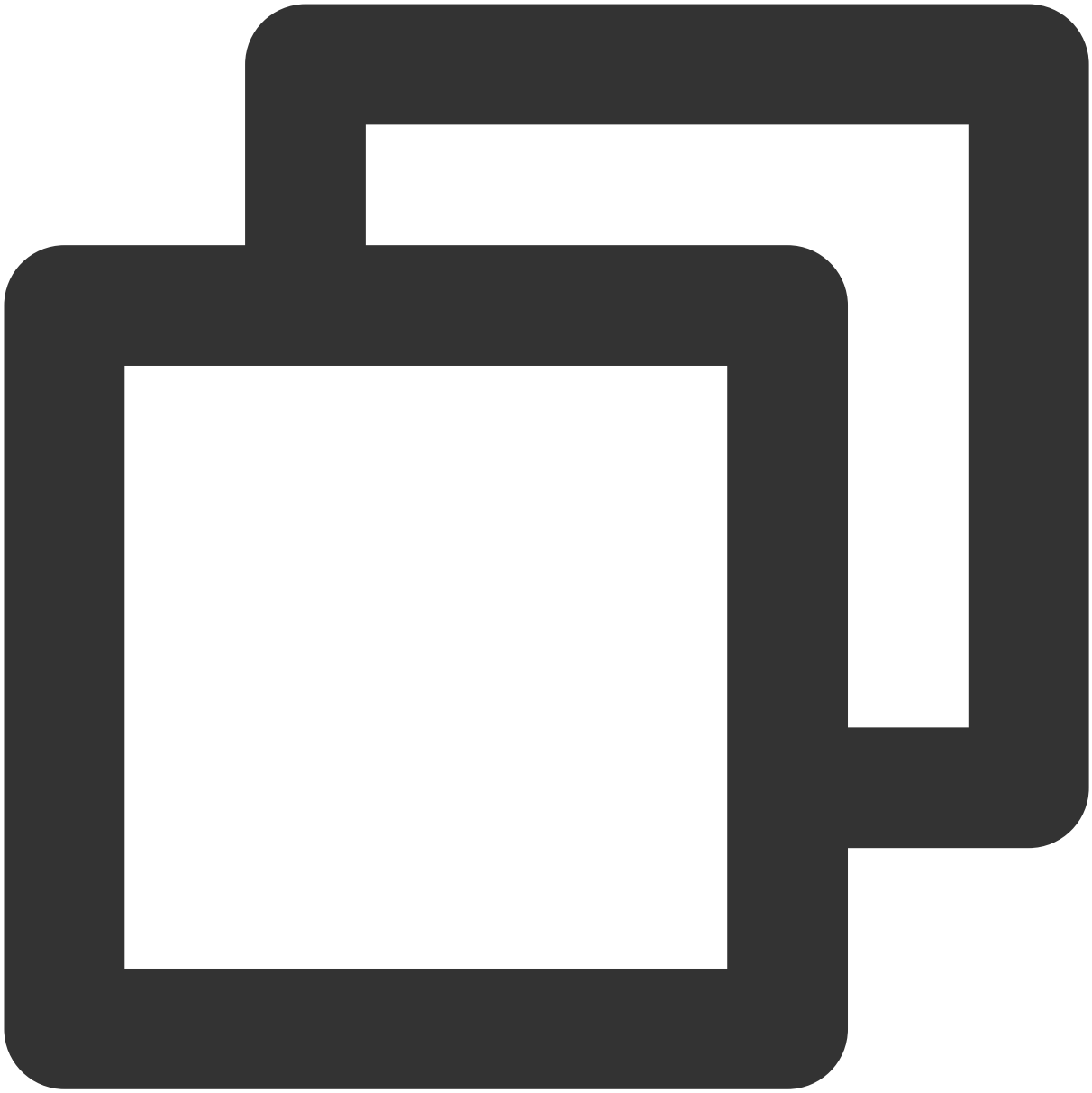


```
/**
```

```
 * 海外的结果回调
 *
 * @author jerrydong
 * @since 2022/6/10
 */
public interface HuiYanOsAuthCallBack {

    /**
     * 活体成功回调
     *
     * @param authResult 结果
     */
    void onSuccess(HuiYanOsAuthResult authResult);

    /**
     * 活体失败
     *
     * @param errorCode 错误码
     * @param errorMsg 错误信息
     * @param token 本次核身使用的
     */
    void onFail(int errorCode, String errorMsg, String token);
}
```

HuiYanAuthEventCallBack

用来监听核身过程中的关键事件的回调，以及如果需要使用自定义布局的UI绑定可以事件绑定(可以参考自定义能力的文档)。



```
/**
 * 慧眼SDK核身的事件回调
 */
public interface HuiYanAuthEventCallBack {

    /**
     * 核身时tips发生改变的事件通知回调
     *
     * @param tipsEvent tips关键事件
     */
    void onAuthTipsEvent(HuiYanAuthTipsEvent tipsEvent);
```

```
    /**
     * 核身的Event事件
     *
     * @param authEvent authEvent
     */
    void onAuthEvent(HuiYanAuthEvent authEvent);

    /**
     * 当认证的主View被创建的回调
     *
     * @param authView
     */
    void onMainViewCreate(View authView);

    /**
     * 界面被回收的回调
     */
    void onMainViewDestroy();

}
```

错误码

这里是SDK在失败回调中的错误码，目前慧眼海外版SDK包含的错误码与其含义如下：

错误码	错误码值	错误码含义
HY_NETWORK_ERROR	210	网络请求出现异常
HY_LOCAL_REF_FAILED_ERROR	211	本地初始化SDK时，检测失败，常见异常不存在license文件或者license过期
HY_USER_CANCEL_ERROR	212	用户主动取消核身流程
HY_INNER_ERROR_CODE	213	SDK内部产生的异常，终止了核身流程
HY_DO_NOT_CHANGE_ERROR	214	在核身过程中切换应用发生终止流程的异常
HY_CAMERA_PERMISSION_ERROR	215	获取摄像头过程中发生异常
HY_INIT_SDK_ERROR	216	未调用init()方法，直接调用了
HY_VERIFY_LOCAL_ERROR	217	本地人脸检测失败
HY_PERMISSION_CHECK_ERROR	218	本地SDK所需要的权限不足

HY_APP_STOP_ERROR	219	集成者主动终止核身流程，startAuthByLightData的reflectSequence为null时
HY_CHECK_LIVE_DATA_ERROR	220	传入的光线序列参数校验失败
HY_INITIALIZATION_PARAMETER_EXCEPTION	221	在未获取设备配置的前提下，直接调用了设置光线序列参数的方法时，会出现的异常
HY_VERIFY_LOCAL_TIME_OUT	222	本地核身动作检测超时
HY_PREPARE_TIME_OUT	223	准备过程超时（启动摄像头到第一次检测到人脸的时间超时）
HY_CHECK_PERMISSION_ERROR	224	SDK内部申请摄像头权限失败

iOS 接口概述

最近更新时间：2023-05-10 17:51:35

API 的详细说明

接入慧眼SDK主要涉及如下几个类，它们分别是API的接口类HuiYanOSKit， 参数配置类HuiYanOsConfig，结果回调类HuiYanOKitSuccCallback以及HuiYanOKitFailCallback。

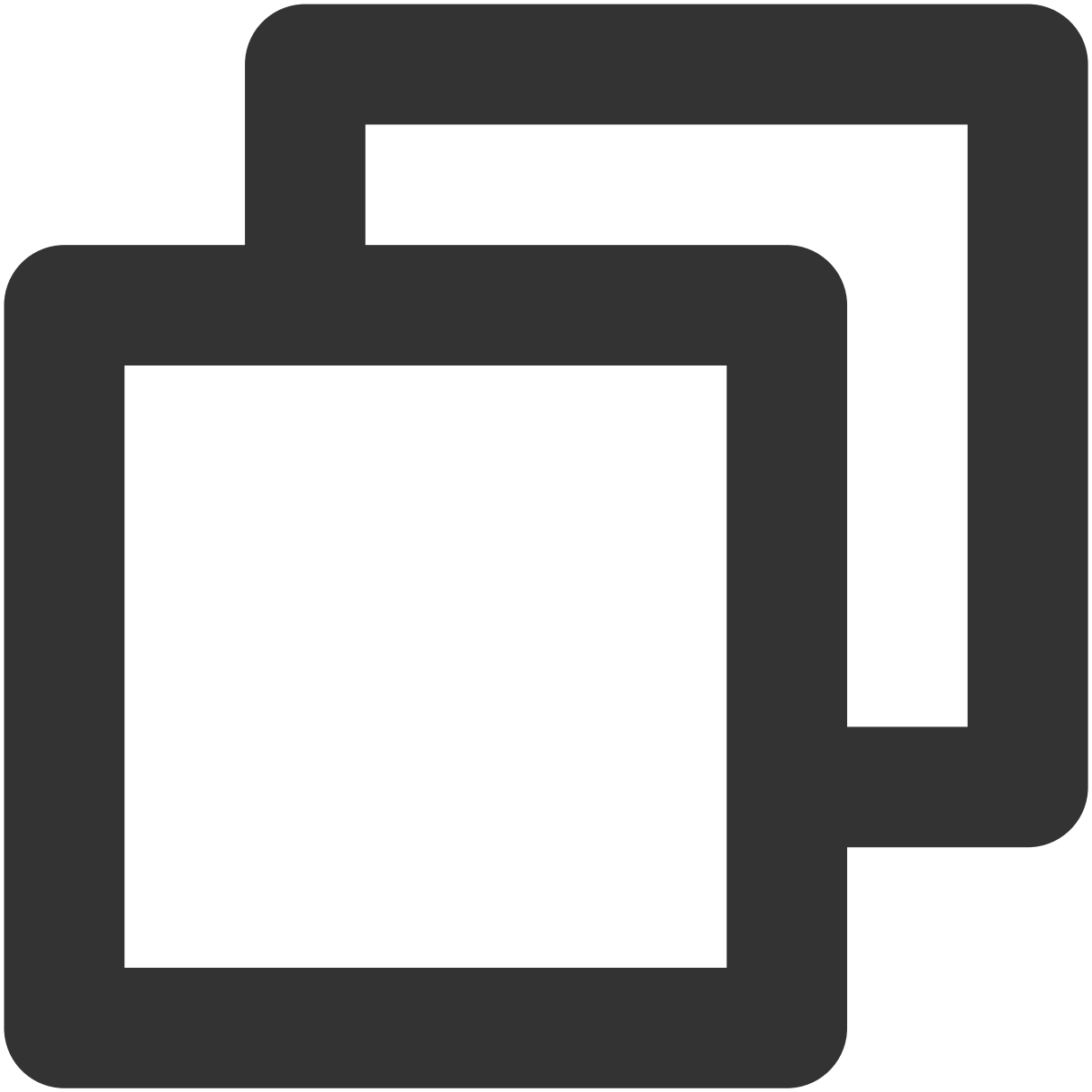
HuiYanOSKit

API	功能描述
startHuiYaneKYC()	进入慧眼SDK活体检测流程
release()	释放SDK资源的接口

release()

功能介绍：

慧眼SDK资源释放的接口。

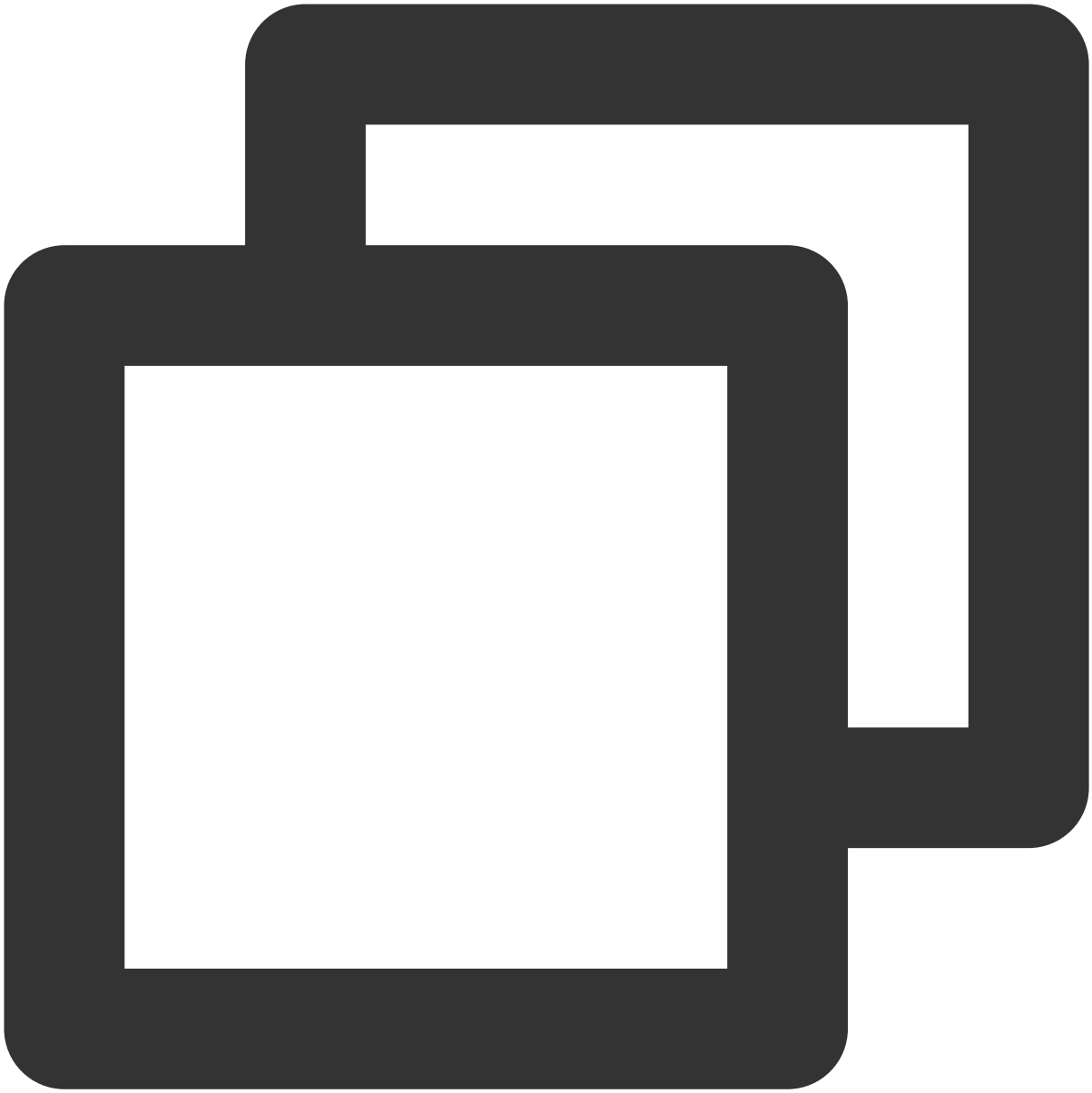


```
+ (void)release;
```

startHuiYaneKYC

功能介绍：

开启活体检测SDK，并配置SDK，回调返回结果或错误



```
/// 开启慧眼活体检测流程
/// @param faceToken token
/// @param kitConfig SDK配置
/// @param succCallback 活体检测成功回调
/// @param failCallback 活体检测失败回调
- (void)startHuiYaneKYC:(NSString *)faceToken withConfig:(HuiYanOsConfig *)kitConfi
        witSuccCallback:(HuiYanOKitSuccCallback)succCallback
       withFailCallback:(HuiYanOKitFailCallback)failCallback;
```

传入参数：

数类型	参数名称	参数含义
NSSting	faceToken	活体流程单号
HuiYanOsConfig	kitConfig	SDK配置类
HuiYanOKitSuccCallback	succCallback	活体检测成功回调
HuiYanOKitFailCallback	failCallback	活体检测失败回调

HuiYanOsAuthResult

数类型	参数名称	参数含义
NSSting	faceToken	活体流程token
NSSting	bestImage	开启最佳桢返回时，成功后返回的最佳桢base64图片

最后需要通过这个token，去腾讯云API的后台接口**GetFaceIdResultIntl**拉取最终活体是否成功的数据。

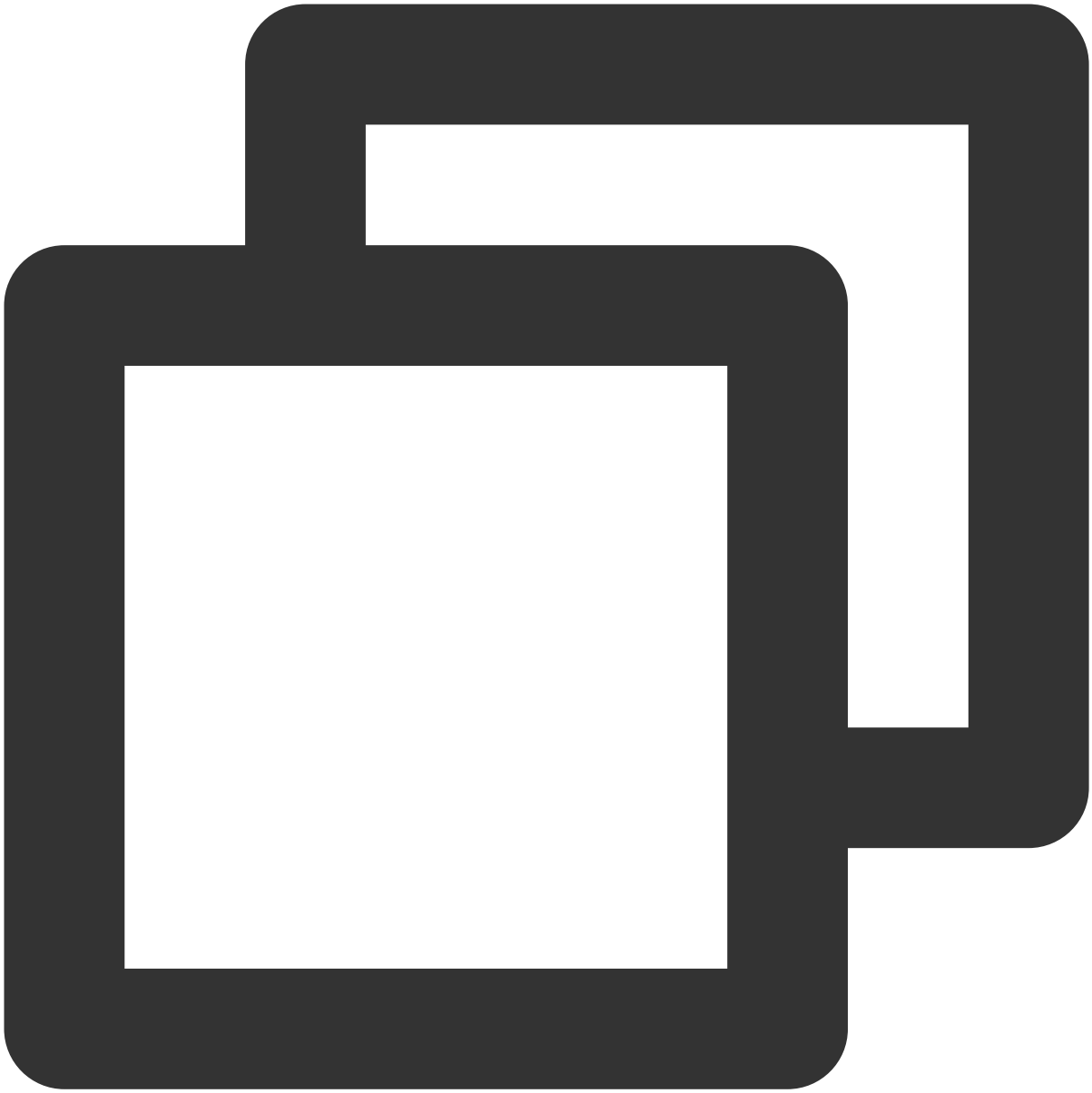
HuiYanOsConfig

HuiYanOsConfig是在启动慧眼SDK时的配置实体类，主要包含了以下属性。

类型	名称	含义	默认值
NSString	authLicense	客户申请的用户核审授权的Licens文件名	空
long	authTimeOutMs	设置活体检测的超时时间	10000毫秒（10秒）
long	prepareTimeoutMs	设置准备阶段检测的超时时间	0
HYShowTimeOutMode	showTimeOutMode	设置显示倒计时阶段	HYShowTimeOutMode_PRE
BOOL	isDeleteVideoCache	是否删除核身视频的本地缓存	YES
BOOL	iShowTipsPage	是否显示引导页	NO
NSString	userUIBundleName	自定义的UI的bundle文件名 比如UserUIBundle.bundle则设置为"UserUIBundle";	nil

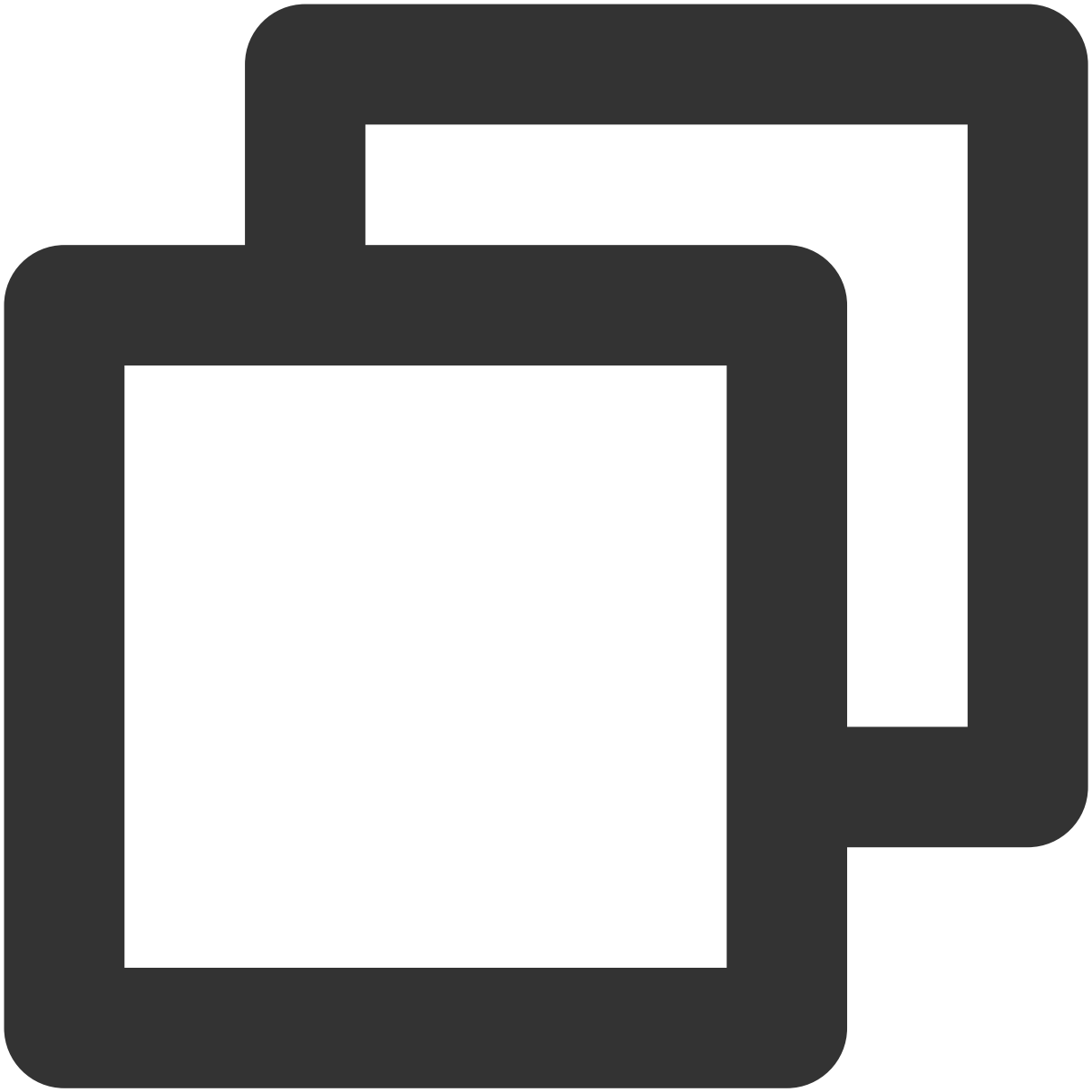
NSString	userLanguageFileName	自定义的languageBundle 名称 比如 UseLanguage.bundle 则设置 为"UseLanguage";	nil
NSString	userLanguageBundleName	自定义本地国际化文件名 比如 en.lproj 则设置为"en";	nil
LanguageType	languageType	SDK内部文字语言设置	DEFAULT
BOOL	isGetBestImg	是否获取最佳帧图片	NO
NSString	setLanguageFileName	默认 HuiYanSDKUI.bundle 内新增的语言文件目录名称 优先级最高	nil

HuiYanOKitSuccCallback



```
/**
 * 活体检测成功回调
 *
 * @param authResult   活体验证结果
 * @param reserved 预留位
 */
typedef void (^HuiYanOKitSuccCallback)(HuiYanOsAuthResult * _Nonnull authResult, id
```

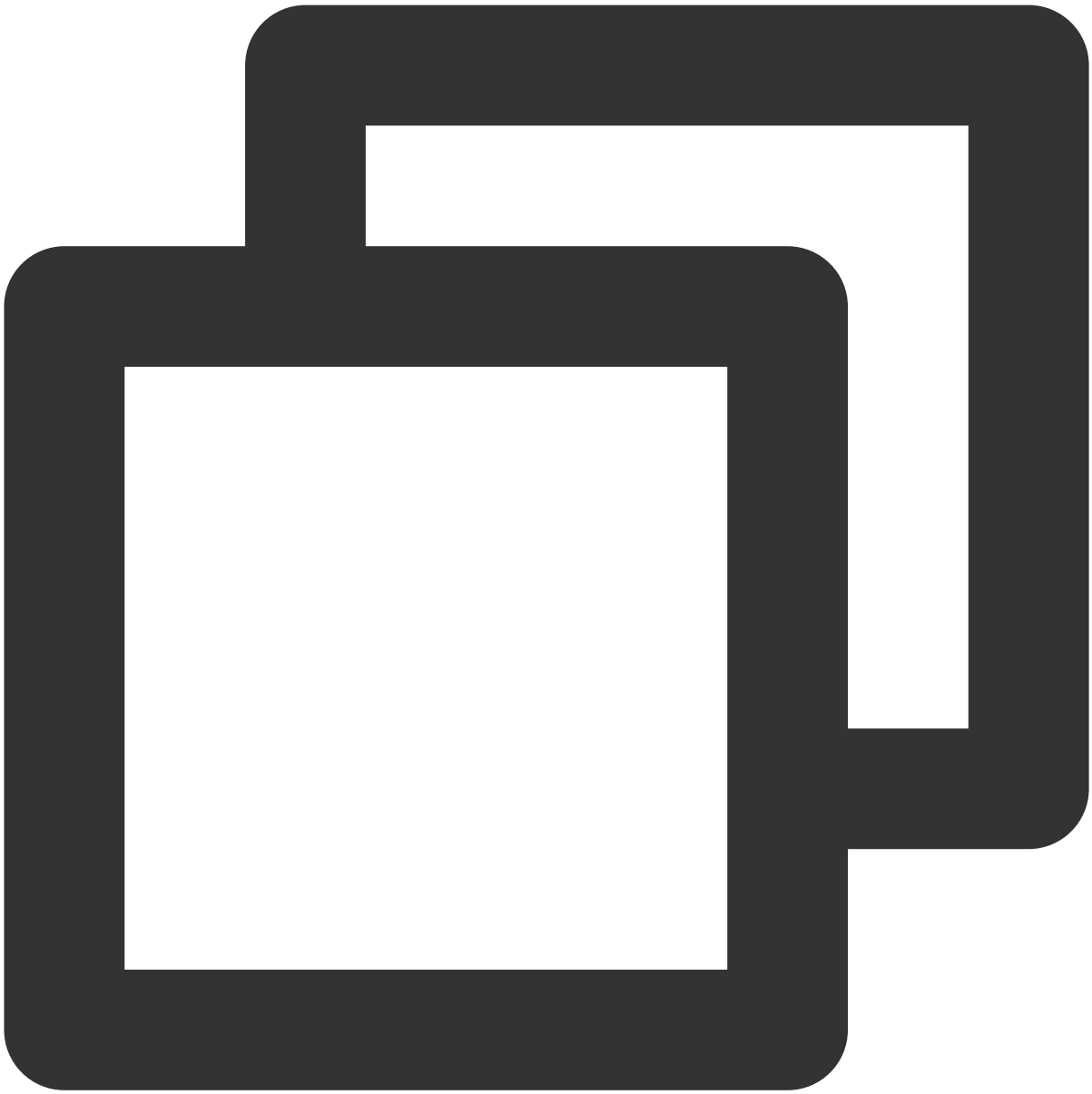
HuiYanOKitFailCallback



```
/**
 * 活体检测失败回调
 *
 * @param errCode 错误码
 * @param errMsg 错误信息
 * @param reserved 预留位
 */
typedef void (^HuiYanOKitFailCallback)(int errCode, NSString * _Nonnull errMsg ,id
```

LanguageType

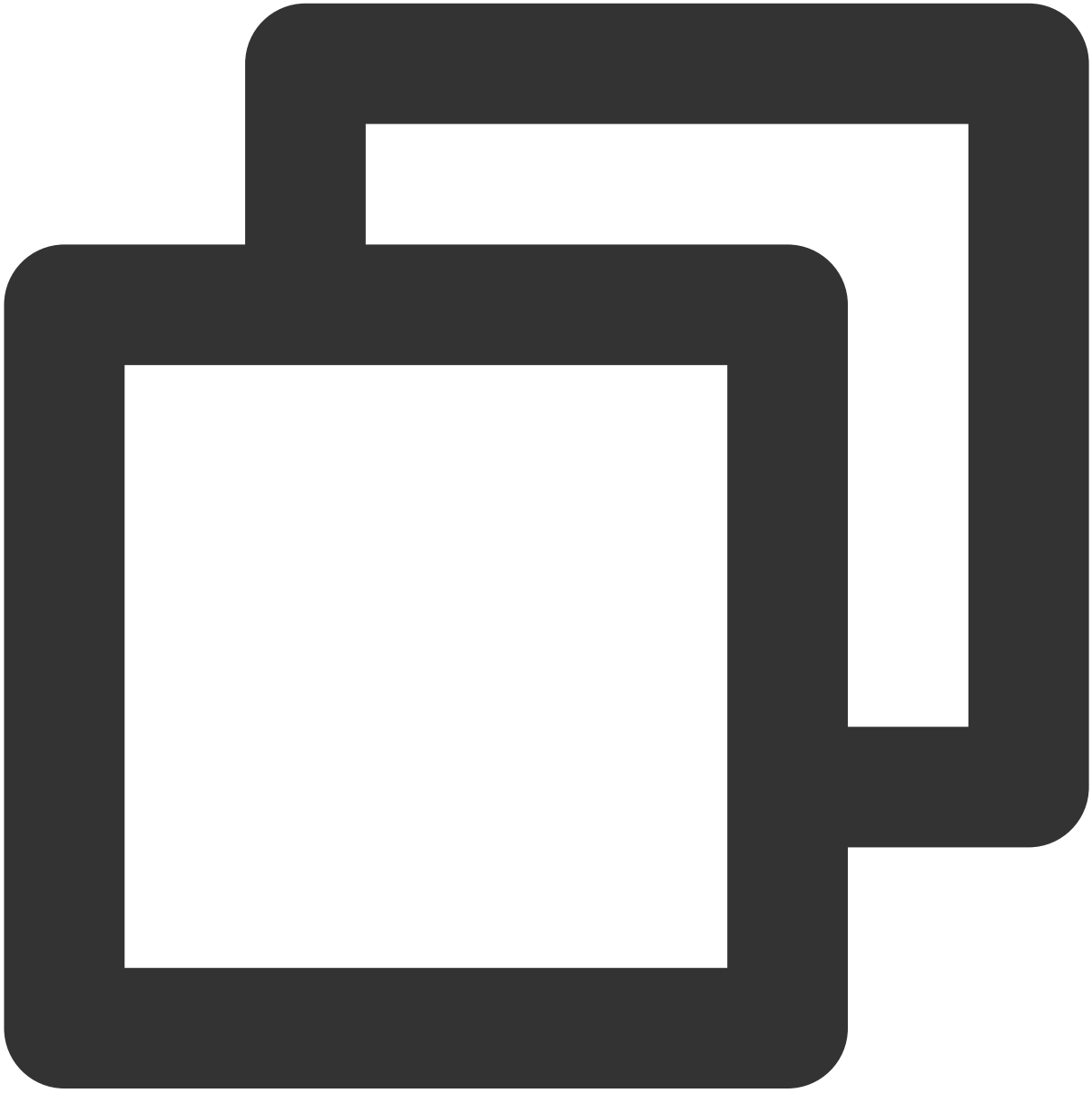
SDK内包含的国际化String



```
typedef enum : NSUInteger {
    DEFAULT = 0,//跟随系统设置
    ZH_HANS,//中文简体
    ZH_HANT,//中文繁体
    ZH_HK,//中文繁体香港
    ZH_TW,//中文繁体台湾
    EN,//英文
    MS,//马来西亚语
    RU,//俄语
```

```
    JA,//日语
    CUSTOMIZE_LANGUAGE, //定制语言
} LanguageType;
```

HYShowTimeOutMode



```
typedef NS_OPTIONS(int, HYShowTimeOutMode) {
    HYShowTimeOutMode_TIMEOUT_HIDDEN = 1 << 0,// 隐藏所有倒计时
    HYShowTimeOutMode_PREPARE = 1 << 1,// 准备阶段倒计时
    HYShowTimeOutMode_ACTION = 1 << 3,// 动作阶段倒计时
};
```

SDK终端错误码(iOS)

错误码	错误码值	错误码含义
HY_SUCCESS	0	成功
HY_INITIALIZATION_PARAMETER_EXCEPTION	210	初始化参数异常
HY_BUNDLE_CONFIGURATION_EXCEPTION	211	bundle配置异常
HY_YTSDK_CONFIGURATION_EXCEPTION	212	优图配置异常
HY_PLEASE_CALL_FIRST_INIT_API	213	需要先进行初始化接口
HY_SDK_AUTH_FAILED	214	SDK 授权失败
HY_USER_VOLUNTARILY_CANCELED	215	用户手动取消
HY_YTSDK_LOCAL_AUTH_FAILED	216	SDK 人脸本地检测失败
HY_CAMERA_OPEN_FAIL	217	相机开启失败
HY_DONOT_SWITCH_APPS	218	请勿在核身过程中切换应用
HY_CAMEREA_PERMISSION_EXCEPTION	219	摄像头权限异常
HY_SDK_VEDIO_CUT_EXCEPTION	220	视频裁剪失败
HY_LIGHT_DATA_FORMAT_EXCEPTION	221	光线数据格式错误

SDK自定义能力说明
Android 自定义能力

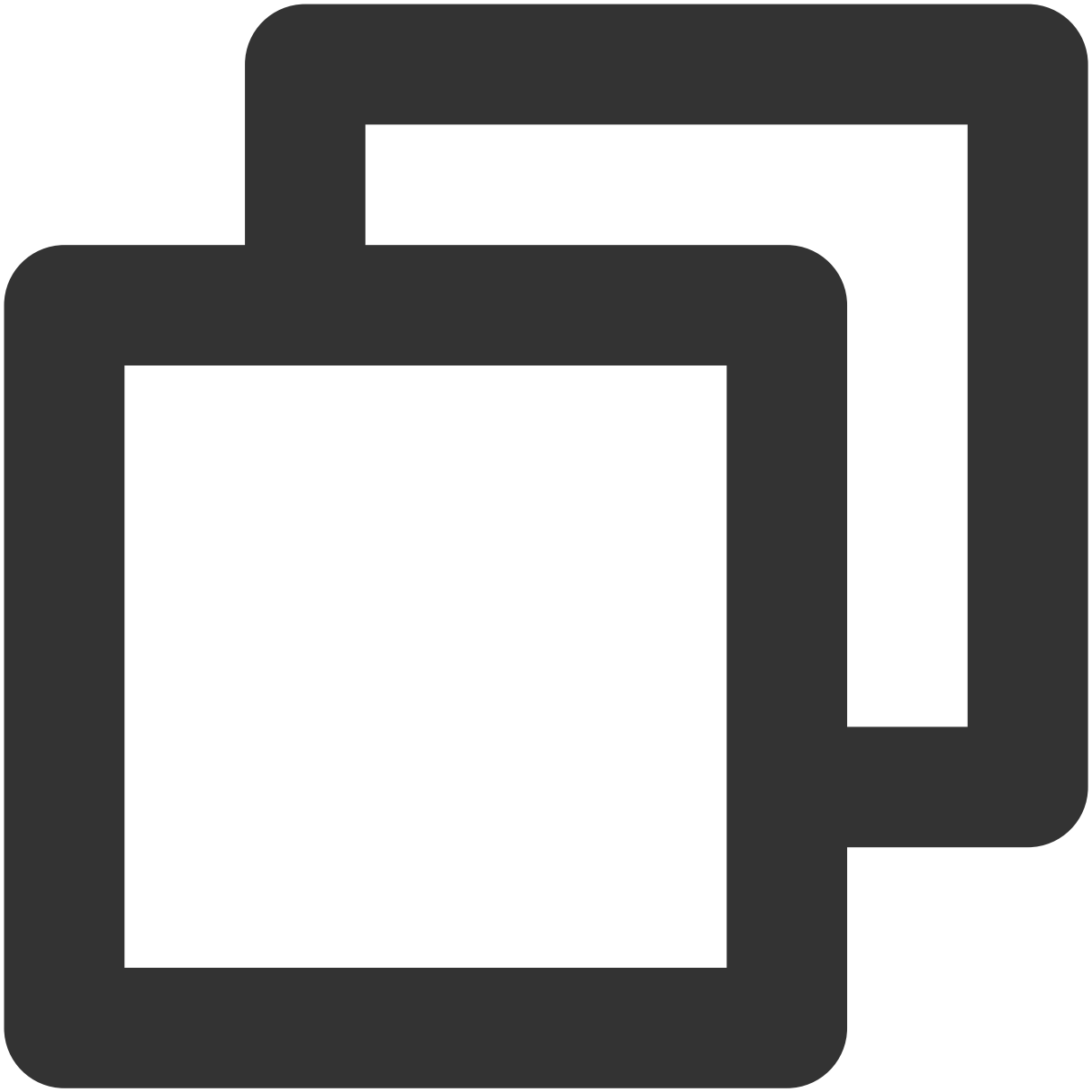
最近更新时间：2023-05-09 11:25:12

本文主要介绍海外版慧眼 SDK 当前所具备的自定义的能力

一、自定义 UI

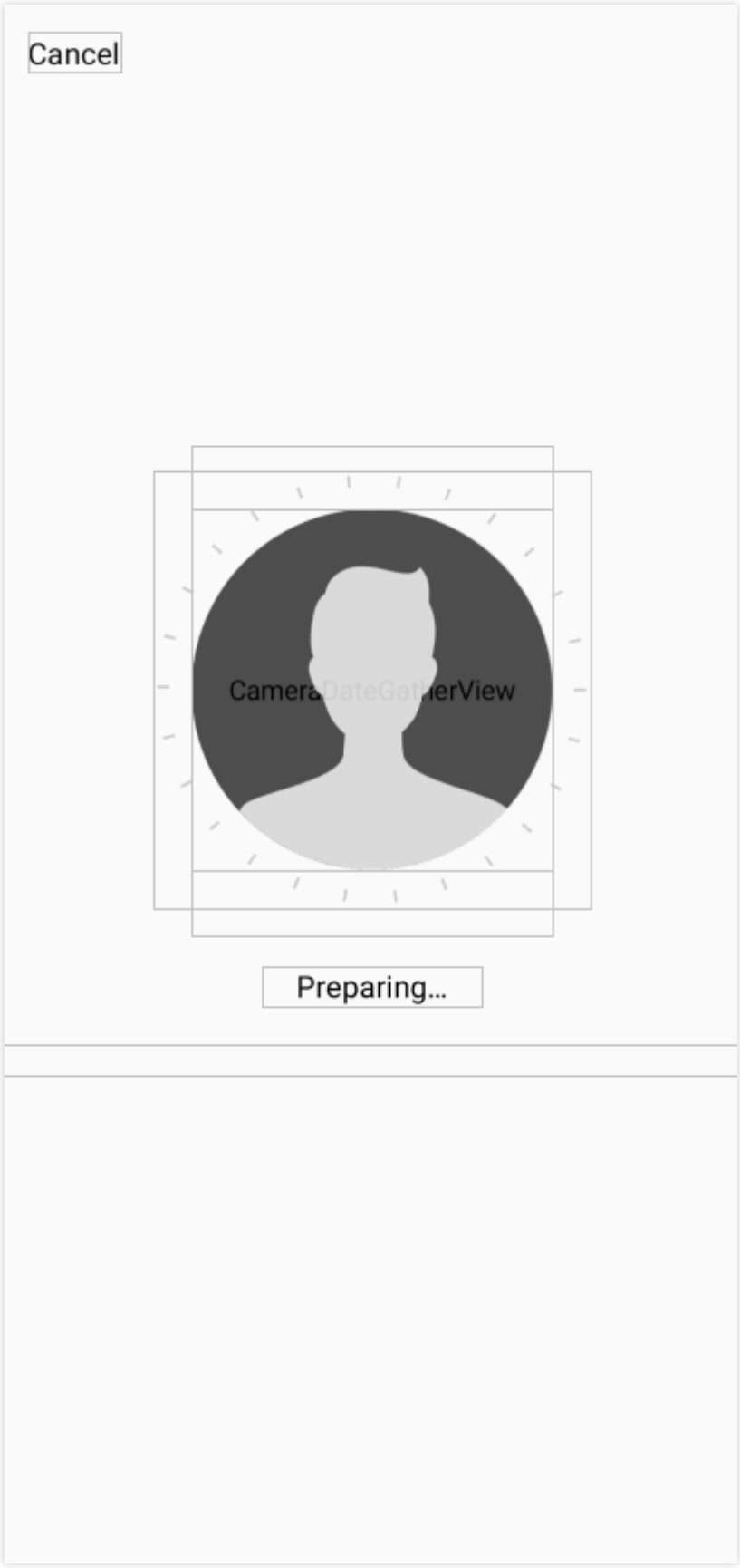
自定义布局

慧眼SDK支持自定义UI，主要是通过**AuthUiConfig**(可以参考接口概述文档)配置参数实现，通过传入**Layout resId**的方式可自定义UI布局。
使用方式大致如下：



```
AuthUiConfig authConfig = new AuthUiConfig();
authUiConfig.setAuthLayoutResId(R.layout.demo_huiyan_fragment_authing);
authUiConfig.setAuthCircleCorrectColor(resources.getColor(R.color.demo_blue));
huiYanOsConfig.setAuthUiConfig(authUiConfig);
```

下图所示属于默认布局：

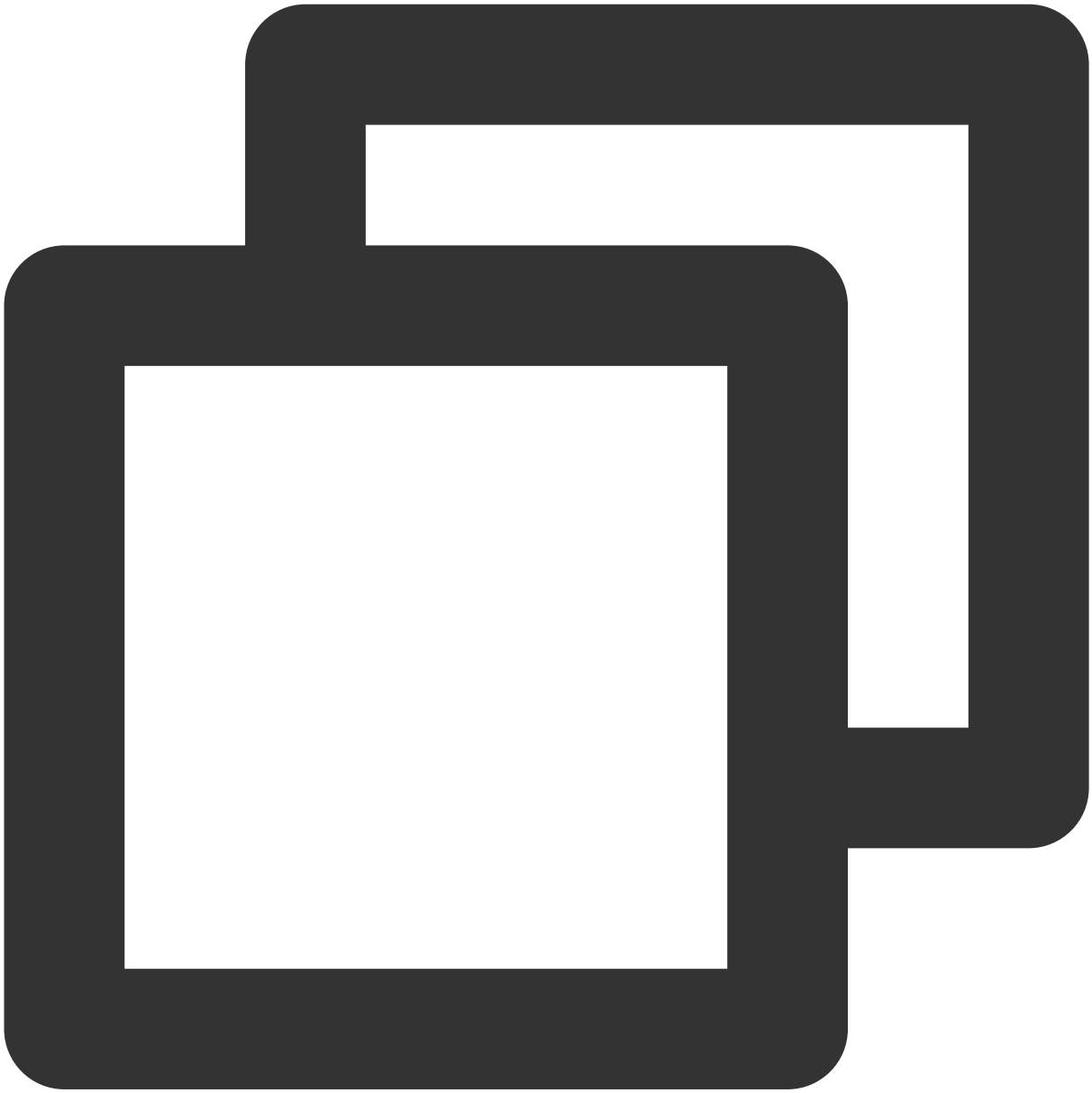


默认布局如上图所示，图中的所以控件的位置均可以通过修改布局Layout.xml进行修改。具体的可以参考demo中的**demo_huiyan_fragment_authing.xml**文件。

注意：

demo_huiyan_fragment_authing.xml 中的**View类型**以及对应的 **android:id** 由于参与的界面事件绑定，**请不要修改**！。

这里提供默认的布局如下



```
<?xml version="1.0" encoding="utf-8"?>
<com.tencent.could.huiyansdk.view.HuiYanReflectLayout xmlns:android="http://schemas
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/txy_auth_layout_bg"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
<!-- 取消按钮 -->
<TextView
    android:id="@+id/txy_cancel_txt_btn"
    android:text="@string/txy_cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:textColor="@color/txy_black"
    android:textSize="16sp"
    android:layout_marginTop="@dimen/txy_title_margin_top"
    android:layout_marginStart="@dimen/txy_protocol_margin_size"
    />

<!-- 倒计时的显示控件 -->
<TextView
    android:id="@+id/txy_count_down_txt_view"
    android:text="@string/txy_count_down_txt"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:textSize="16sp"
    android:textColor="@color/txy_black"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/txy_title_margin_top"
    android:layout_marginRight="@dimen/txy_protocol_margin_size"
    android:visibility="gone"
    />

<!-- 摄像头预览界面框 720P的话，代码中height会自定乘以1.3 -->
<com.tencent.could.huiyansdk.view.CameraDateGatherView
    android:id="@+id/txy_camera_gather_view"
    android:layout_width="@dimen/txy_auth_head_size"
    android:layout_height="258dp"
    android:background="@android:color/transparent"
    android:layout_marginBottom="@dimen/txy_auth_view_move"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<!-- 通用背景，待竖线的圆环 -->
<com.tencent.could.huiyansdk.view.CommonAuthBackView
    android:id="@+id/txy_auth_common_background_views"
    android:layout_width="230dp"
    android:layout_height="230dp"
```

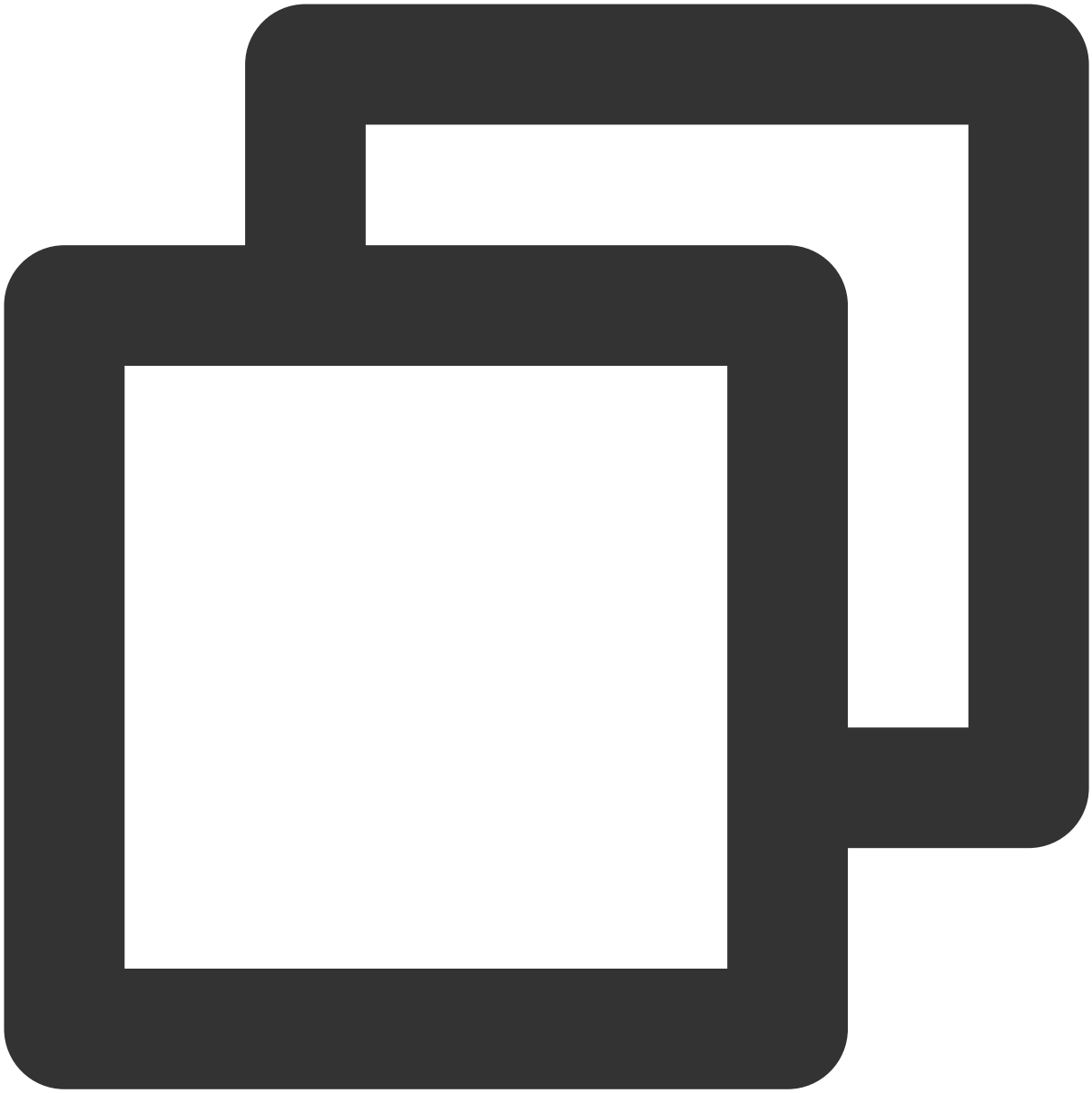
```
        android:layout_marginBottom="@dimen/txy_auth_view_move"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

    <!-- 展示的头像图片 -->
    <ImageView
        android:id="@+id/txy_camera_prepare_img"
        app:srcCompat="@drawable/txy_prepare_face_head_white"
        android:layout_width="@dimen/txy_auth_head_size"
        android:layout_height="@dimen/txy_auth_head_size"
        android:layout_marginBottom="@dimen/txy_auth_view_move"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <!-- 前台的动画View -->
    <com.tencent.could.huiyansdk.view.LoadingFrontAnimatorView
        android:id="@+id/txy_auth_loading_front_animator_view"
        android:visibility="gone"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginBottom="@dimen/txy_auth_view_move"
        android:layout_width="@dimen/txy_auth_head_size"
        android:layout_height="@dimen/txy_auth_head_size"
        />

    <!--  提示信息的展示界面  -->
    <TextView
        android:id="@+id/txy_auth_feed_back_txt"
        app:layout_constraintTop_toBottomOf="@id/txy_auth_common_background_views"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:textColor="@color/txy_black"
        android:layout_marginTop="@dimen/txy_protocol_line_space"
        android:text="@string/txy_face_preparing3"
        android:textSize="18sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <!--  额外的警告信息的提示控件  -->
    <TextView
```

```
        android:id="@+id/txy_auth_feed_back_extra_tip_txt"
        android:textSize="14sp"
        android:textColor="@color/txy_black"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="20dp"
        app:layout_constraintTop_toBottomOf="@id/txy_auth_feed_back_txt"
        />

    <!--  核身的内容提示 -->
    <TextView
        android:id="@+id/txy_auth_tips_txt"
        android:textSize="14sp"
        android:textColor="@color/txy_black"
        android:paddingHorizontal="25dp"
        android:layout_marginHorizontal="25dp"
        android:layout_marginBottom="35dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</com.tencent.could.huiyansdk.view.HuiYanReflectLayout>
```

自定义布局的事件绑定

上一节介绍了如何进行自定义UI布局，本节主要介绍如何对自定义UI布局里新增的控件进行事件绑定，以满足您在使用阶段的各项需求。

当核身界面被创建时，HuiYanAuthEventCallBack的 onMainViewCreate(View authView)被调用到。当核身界面即将销毁的时候，会回调onMainViewDestroy()方法，您可以在对应的生命周期中自定义处理逻辑。



```
// 这里设置慧眼关键事件的回调监听
HuiYanOsApi.setAuthEventCallBack(new HuiYanAuthEventCallBack() {
    @Override
    public void onAuthTipsEvent(HuiYanAuthTipsEvent tipsEvent) {
        Log.e(TAG, "current is : " + tipsEvent);
    }

    @Override
    public void onMainViewCreate(View authView) {
        if (authView == null) {
            return;
```

```
        }
        // 获取自定义添加的控件，并且注册自定义的事件
        Button findBtn = authView.findViewById(R.id.add_view_offer_button);
        if (findBtn != null) {
            findBtn.setOnClickListener(view -> {
                Log.e(TAG, "click test button!");
            });
        }
    }

    @Override
    public void onMainViewDestroy() {
        Log.e(TAG, "onMainViewDestroy");
    }
});
```

根据onMainViewCreate的回调与onMainViewDestroy的回调方式，您可以在完成任意对应您新增UI控件的事件绑定，以实现对应效果。

二、自定义提示与增加语言

自定义提示

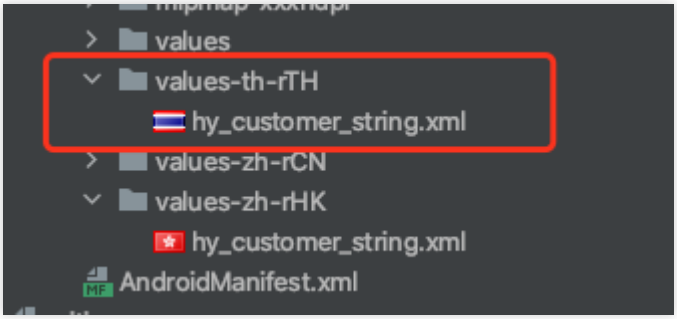
如果您需要修改提示语，或者新增其他的语言文件的话，可以按照下面的方式取实现。慧眼SDK会提供一个翻译文件**hy_customer_string.xml** 里面包括了所有慧眼SDK对外可以进行修改的配置文字文件。

1. 打开在主module（集成慧眼SDK的module）工程。
2. 将**hy_customer_string.xml**添加到对应的语言文件夹下。
3. 修改其中需要自定义的文字内容即可。
4. 打包以后，您修改的内容会自动覆盖原有的内容。

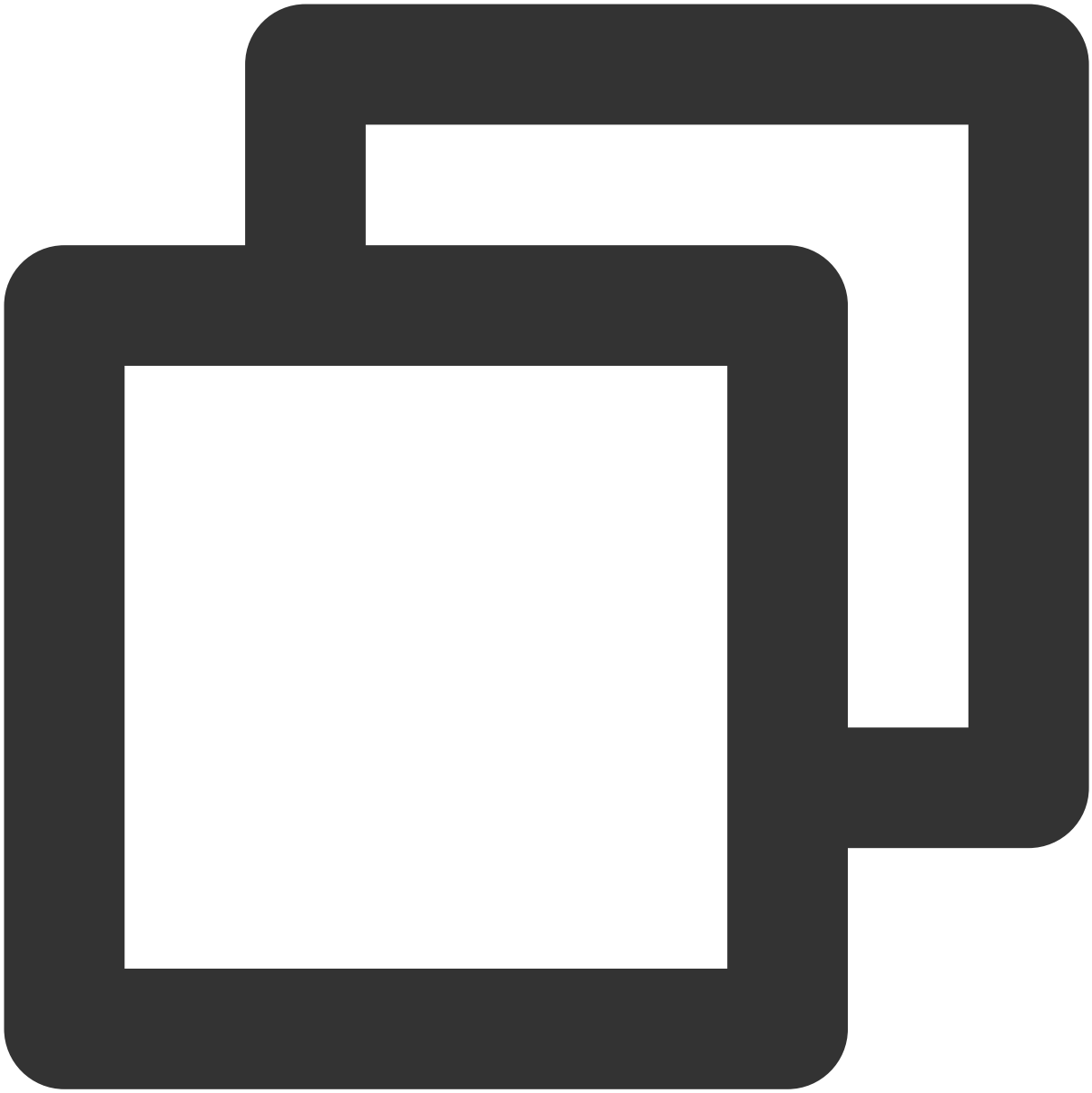
新增语言

新增语言的话文件，则需要执行以下两部就可以实现：

1. 在主module（集成慧眼SDK的module）工程里新增对应语言文件夹。
2. 将**hy_customer_string.xml**拷贝到改语言文件夹下，并修改对应的value内容。



3. 在代码里指定对应的语言码即可（以泰文为例）。



```
huiYanOsConfig.setLanguageStyle(LanguageStyle.CUSTOMIZE_LANGUAGE);
huiYanOsConfig.setLanguageCode("th-TH");
```

附录 Android 语言码

这里提供一下 Android 的部分语言码，以供参考。

语言码	对应使用的国家或地区
af-ZA	**公用荷兰语 - 南非**

sq-AL	阿尔巴尼亚 -阿尔巴尼亚
ar-DZ	阿拉伯语 -阿尔及利亚
ar-BH	阿拉伯语 -巴林
ar-EG	阿拉伯语 -埃及
ar-IQ	阿拉伯语 -伊拉克
ar-JO	阿拉伯语 -约旦
ar-KW	阿拉伯语 -科威特
ar-LB	阿拉伯语 -黎巴嫩
ar-LY	阿拉伯语 -利比亚
ar-MA	阿拉伯语 -摩洛哥
ar-OM	阿拉伯语 -阿曼
ar-QA	阿拉伯语 -卡塔尔
eu-ES	巴斯克 -巴斯克
be-BY	Belarusian-白俄罗斯
bg-BG	保加利亚 -保加利亚
ca-ES	嘉泰罗尼亚 -嘉泰罗尼亚
zh-HK	华语 - 中国香港
zh-MO	华语 - 中国澳门
zh-CN	华语 -中国
zh-SG	华语 -新加坡
zh-TW	华语 - 中国台湾
zh-CHS	华语 (简体化)
zh-CHT	华语 (传统的)
hr-HR	克罗埃西亚 -克罗埃西亚
cs-CZ	捷克 - 捷克

da-DK	丹麦文 -丹麦
div-MV	Dhivehi-马尔代夫
nl-BE	荷兰 -比利时
nl-NL	荷兰 - 荷兰
en-AU	英国 -澳洲
en-CA	英国 -加拿大
en-ZA	英国 - 南非
en-PH	英国 -菲律宾共和国
en-NZ	英国 - 新西兰
en-GB	英国 - 英国
en-US	英国 - 美国
fa-IR	波斯语 -伊朗王国
fi-FI	芬兰语 -芬兰
fr-FR	法国 -法国
fr-BE	法国 -比利时
fr-MC	法国 -摩纳哥
fr-CH	法国 -瑞士
gl-ES	加利西亚 -加利西亚
ka-GE	格鲁吉亚州 -格鲁吉亚州
de-DE	德国 -德国
de-LU	德国 -卢森堡
de-CH	德国 -瑞士
el-GR	希腊 -希腊
gu-IN	Gujarati-印度
he-IL	希伯来 -以色列

hi-IN	北印度语 -印度
hu-HU	匈牙利的 -匈牙利
is-IS	冰岛的 -冰岛
it-IT	意大利 -意大利
ja-JP	日本 -日本
kk-KZ	Kazakh-哈萨克
kn-IN	卡纳达语 -印度
ko-KR	韩国 -韩国
lv-LV	拉脱维亚的 -拉脱维亚
lt-LT	立陶宛 -立陶宛
ms-BN	马来 -汶莱
ms-MY	马来 -马来西亚
mr-IN	马拉地语 -印度
mn-MN	蒙古 -蒙古
nn-NO	挪威 (Nynorsk)- 挪威
pl-PL	波兰 -波兰
pt-BR	葡萄牙 -巴西
pt-PT	葡萄牙 -葡萄牙
ro-RO	罗马尼亚语 -罗马尼亚
sa-IN	梵文 -印度
ru-RU	俄国 -俄国
sk-SK	斯洛伐克 -斯洛伐克
es-AR	西班牙 -阿根廷
es-ES	西班牙 -西班牙
sv-SE	瑞典 -瑞典

th-TH	泰国 -泰国
tr-TR	土耳其语 -土耳其
uk-UA	乌克兰 -乌克兰
ur-PK	Urdu-巴基斯坦
vi-VN	越南 -越南

iOS 自定义能力

最近更新时间：2023-05-09 11:22:11

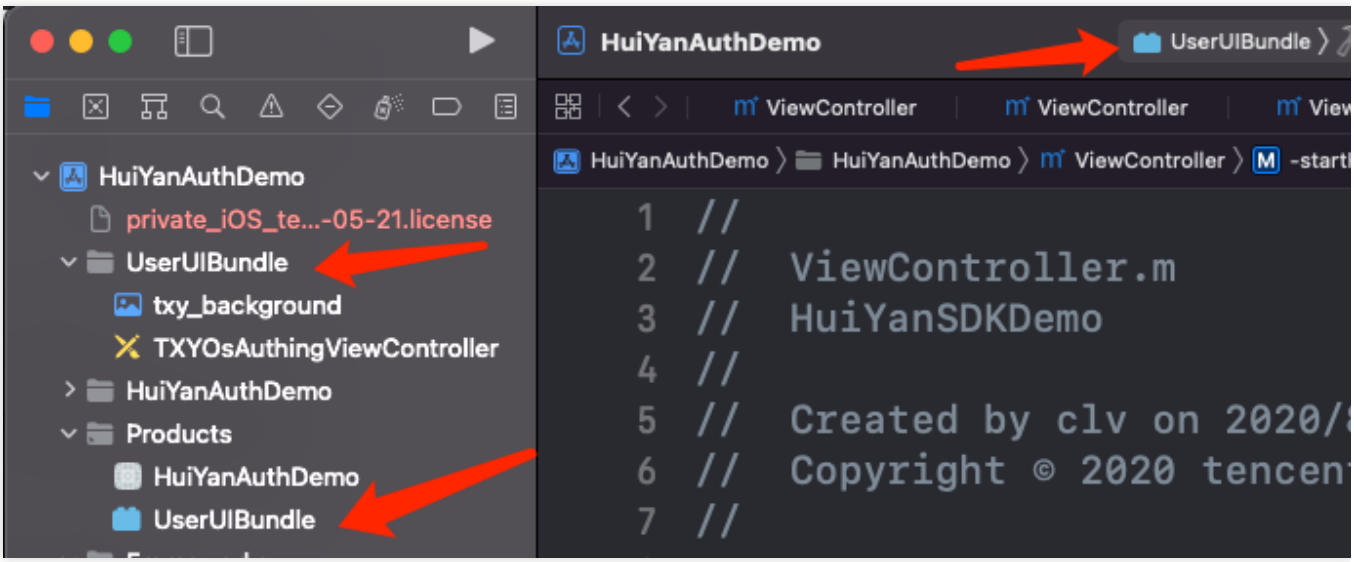
本文主要介绍海外版慧眼 SDK 当前所具备的自定义的能力

一、自定义 UI

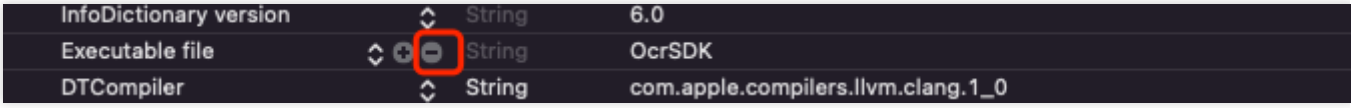
本文主要介绍如何使用自定义UI，定制慧眼识别页面以及图标。

构建步骤

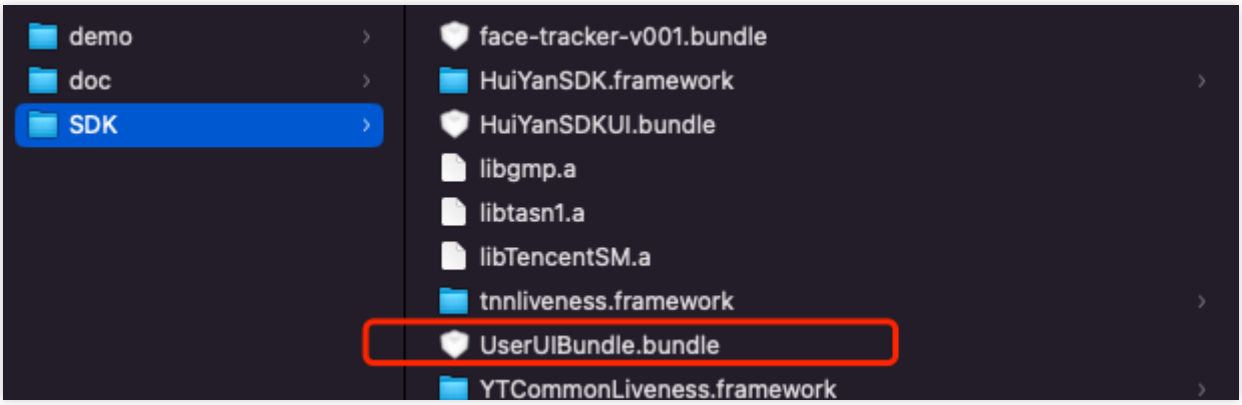
1. SDK压缩包解压后，进入demo目录，打开HuiYanODemo项目。
2. 切换构建secheme到UserUIBundle，command+B可以构建出产物。



3. 构建出bundle后右键查看包内容，打开info.plist，删掉“Executable file”字段，否则会影响AppStore上架，导致上传IPA报错。也可以将info.plist文件删掉。

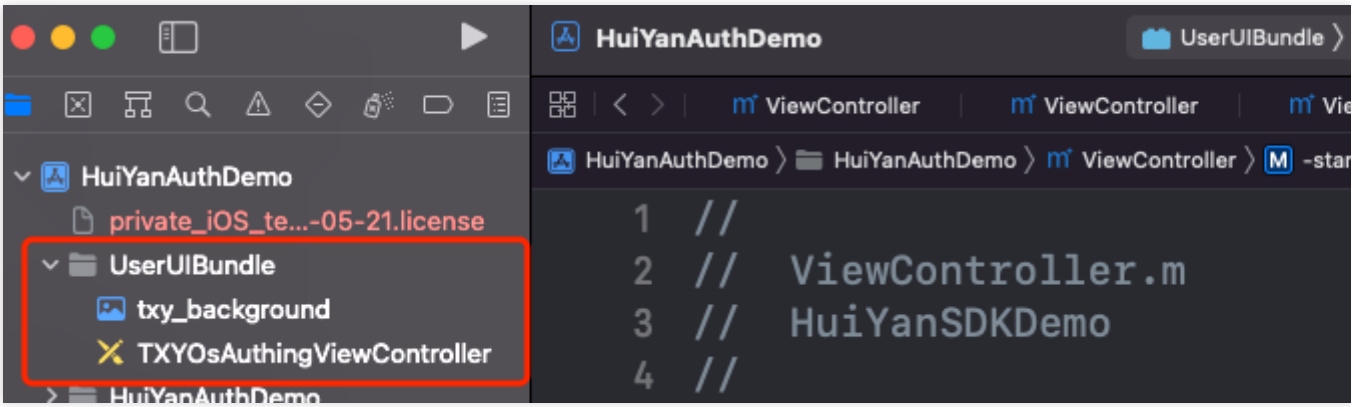


4. 将构建产物直接替换掉根目录SDK目录下的UserUIBundle.bundle，将该目录下所有lib和bundle添加到自己的项目中（注：若没有修改bundle内容，可不进行替换，直接使用该目录下的bundle即可）。

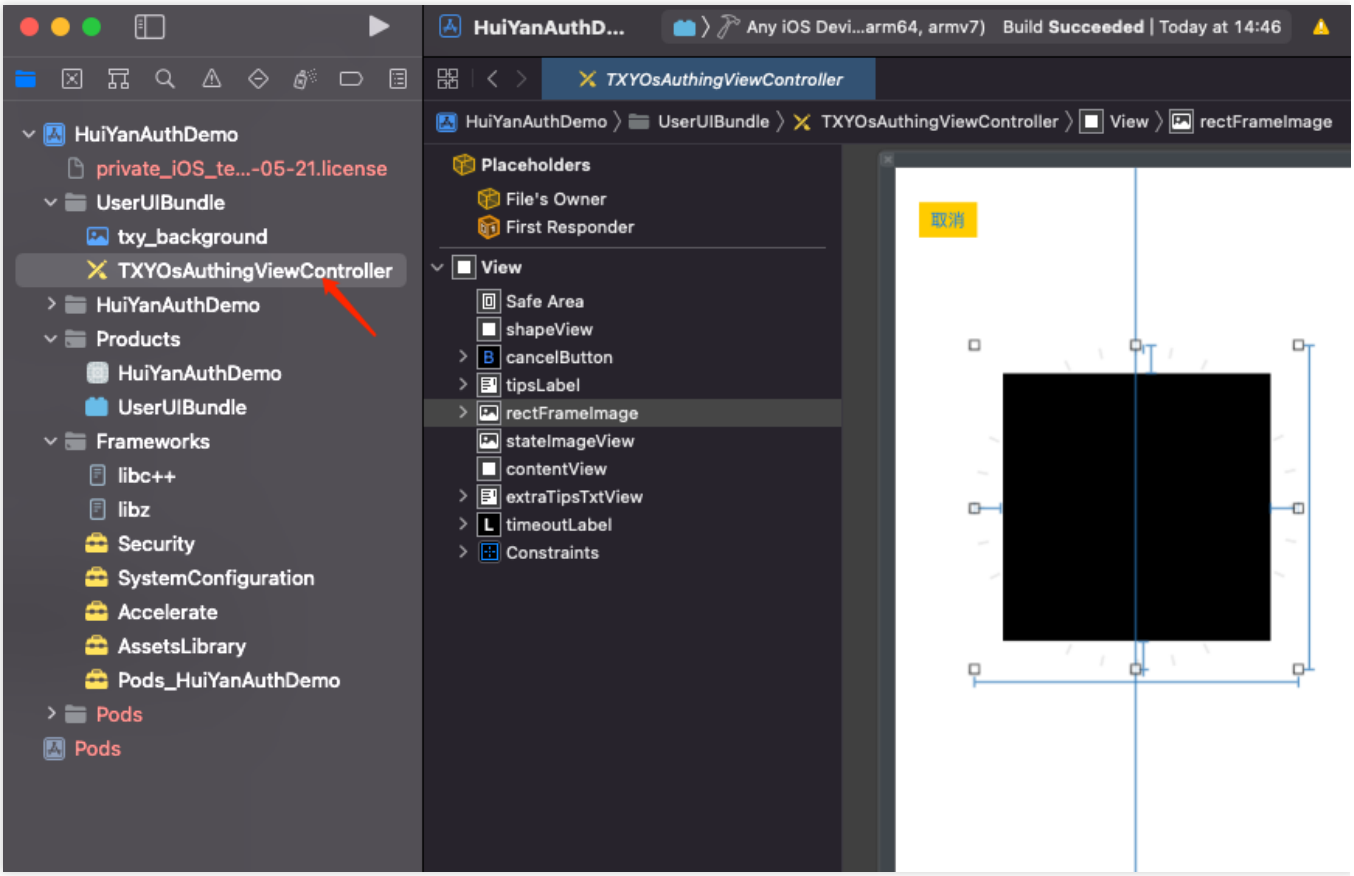


自定义内容

1. 图标Icon。可以将Icon替换到以下列表，命名要保持原有命名。



2. xib内布局调整。如TXYOsAuthingViewController 识别页面，可以修改xib内组件的布局，增加静态组件，但是不允许删除组件。该页面逻辑由SDK内部.m文件完成，仅支持修改布局和增加无逻辑组件（如增加背景图）。



3. 通过设置userUIBundleName字段设置到SDK中

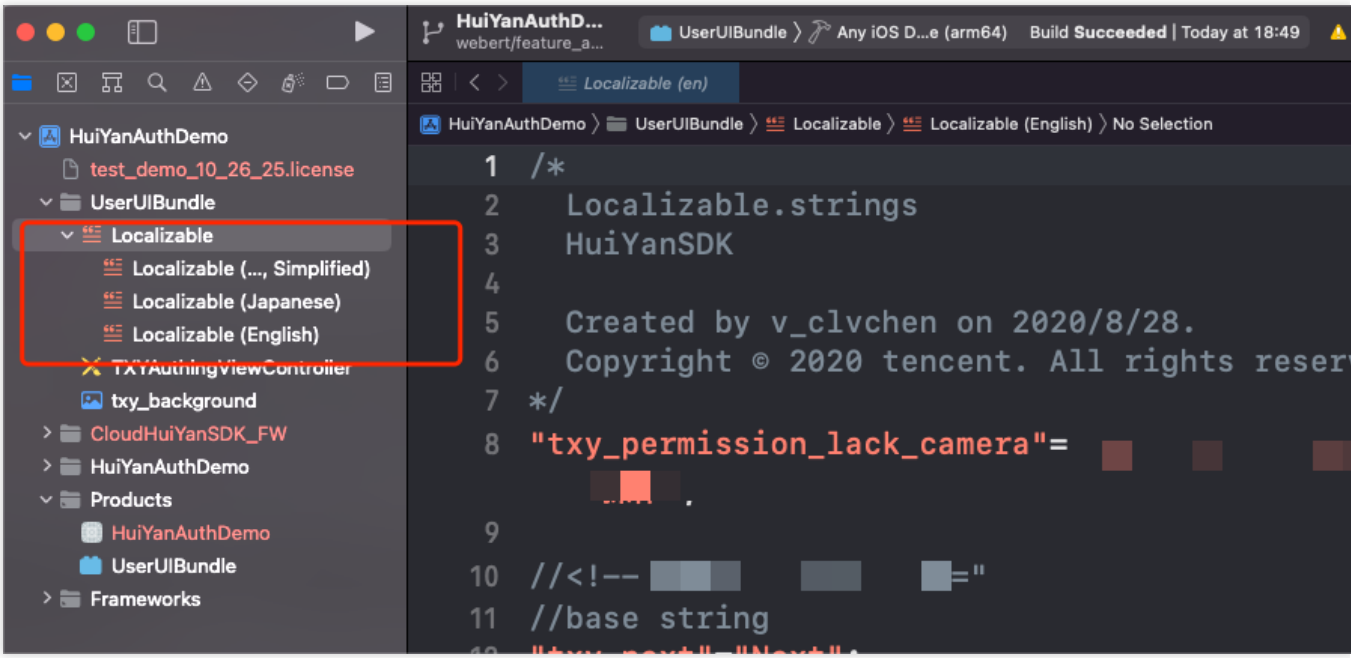
这个设置了会使用自定义打包的UI bundle 文件，会加载里面这个名称的TXYOsAuthingViewController布局文件

若是未找到TXYOsAuthingViewController 名称的布局文件，将会加载默认布局。

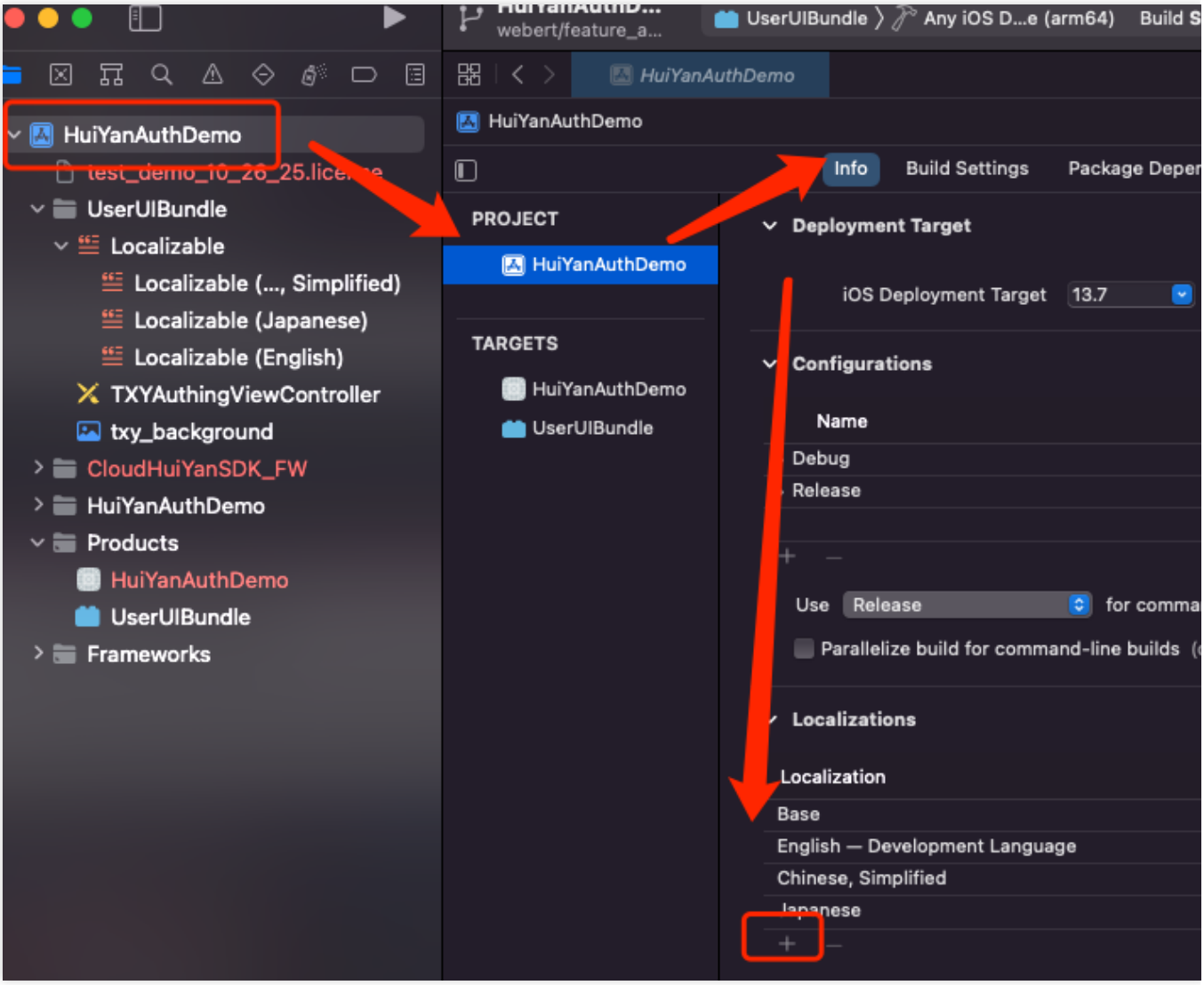
二、自定义语言

添加自定义语言

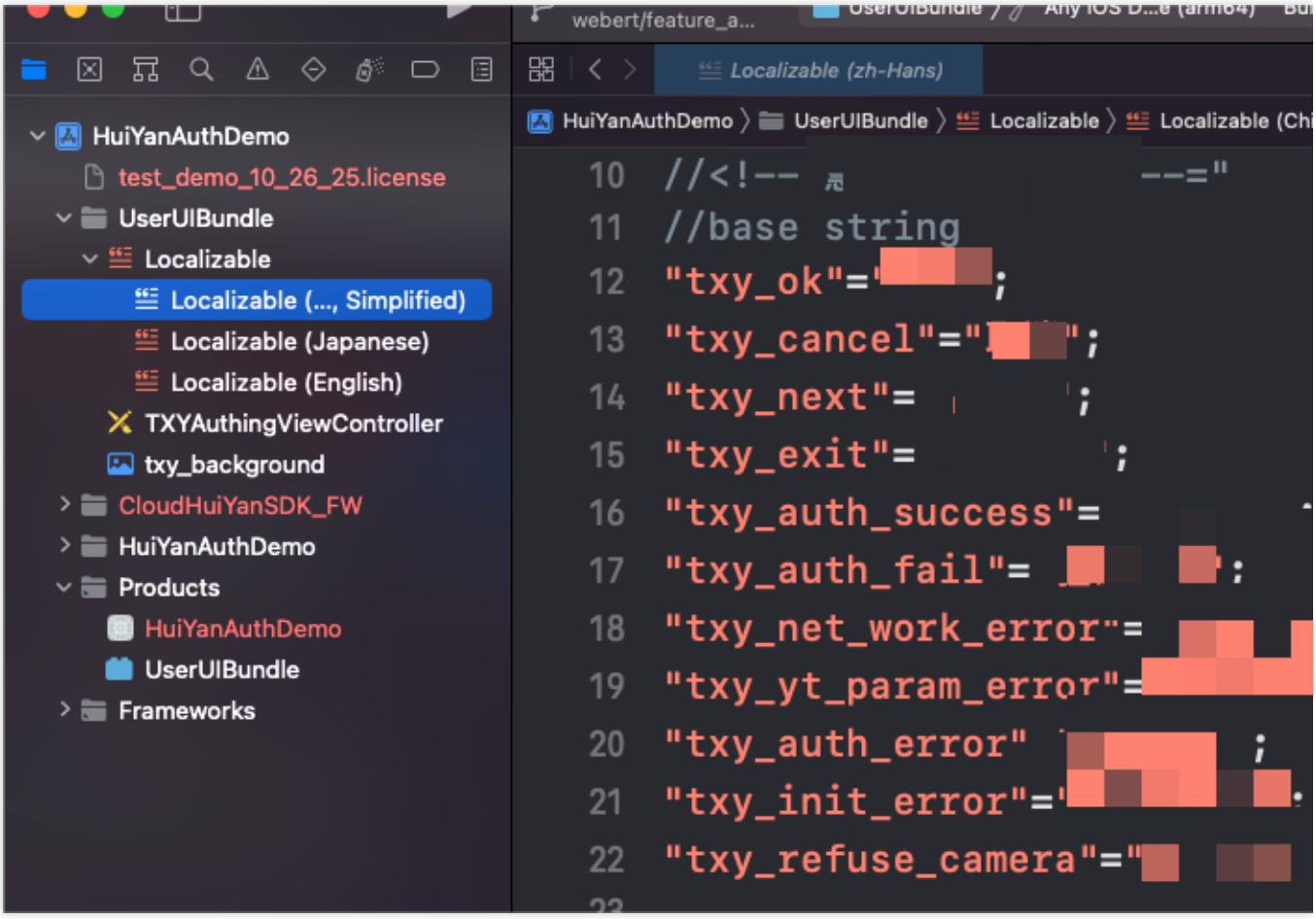
1. demo中UserUIBundle文件夹中包含Localizable。下图中右侧可设置支持的语言类型，对应左侧会出现子文件，在子文件中对已有的key字符串做多语言映射。



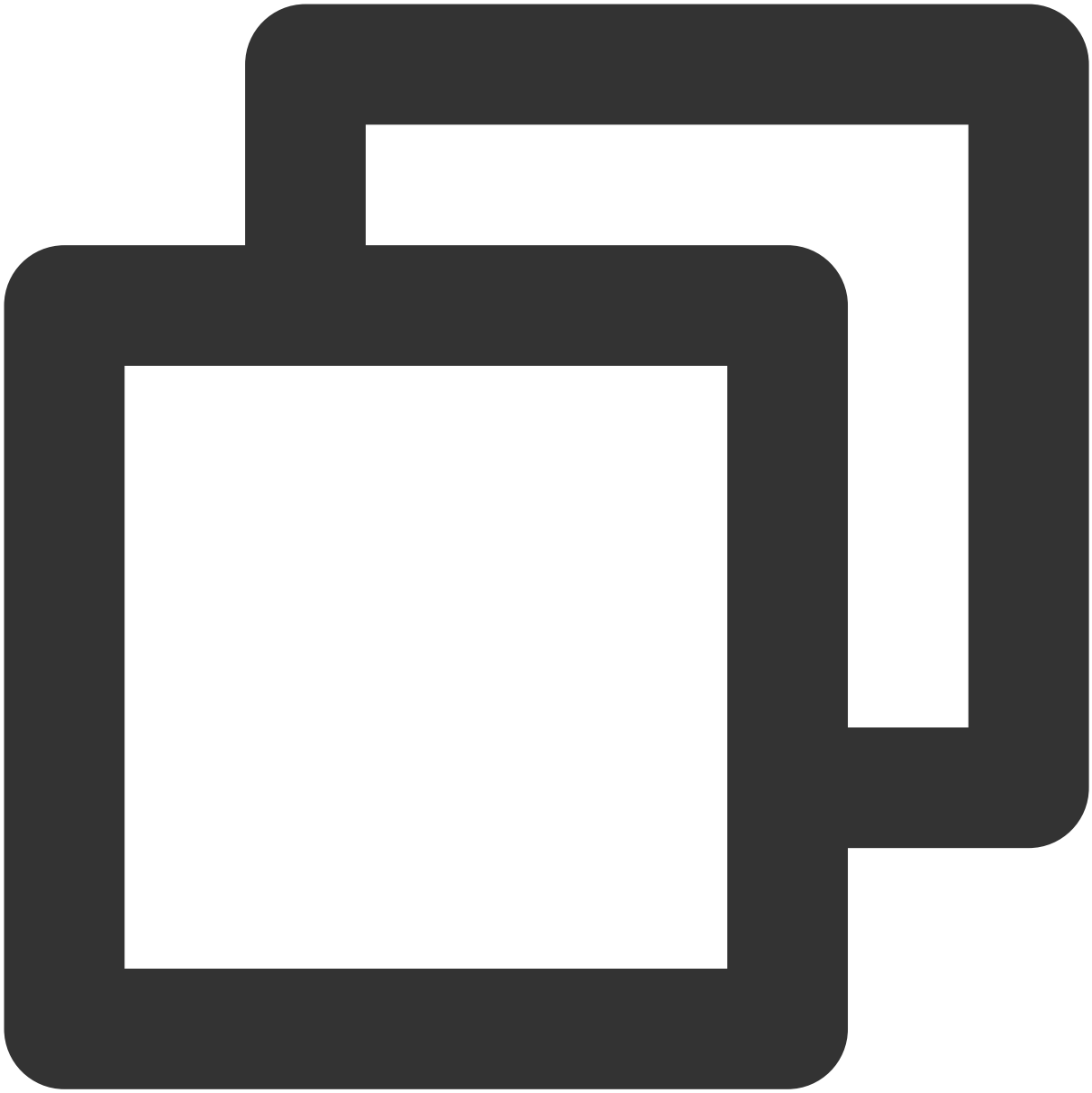
2. 若右侧没有目标语言可先对工程设置里添加对应语言，之后重复步骤1即可



3. 对目标文件进行翻译映射。下图示例为简体中文的映射，若添加其他语言左侧key保持不变，右侧为译文即可。

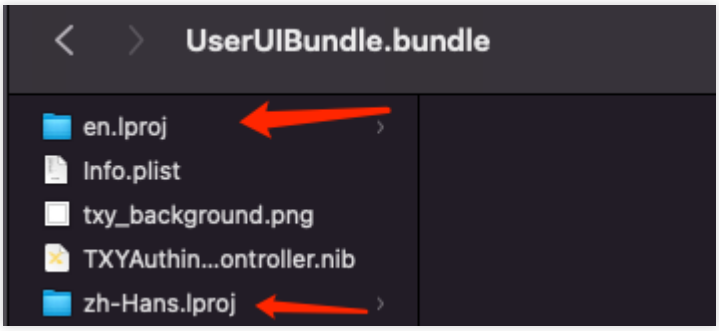

HuiYanAuthDemo
test_demo_10_26_25.license
UserUIBundle
Localizable
Localizable (..., Simplified)
Localizable (Japanese)
Localizable (English)
TXYAuthingViewController
txy_background
CloudHuiYanSDK_FW
HuiYanAuthDemo
Products
HuiYanAuthDemo
UserUIBundle
Frameworks
Localizable (zh-Hans)
HuiYanAuthDemo 〉 UserUIBundle 〉 Localizable 〉 Localizable (Chi
10 //<!--
11 //base string
12 "txy_ok"="
13 "txy_cancel"="
14 "txy_next"=
15 "txy_exit"=
16 "txy_auth_success"=
17 "txy_auth_fail"=
18 "txy_net_work_error"=
19 "txy_yt_param_error"=
20 "txy_auth_error"
21 "txy_init_error"="
22 "txy_refuse_camera"="

4. SDK中设置为：



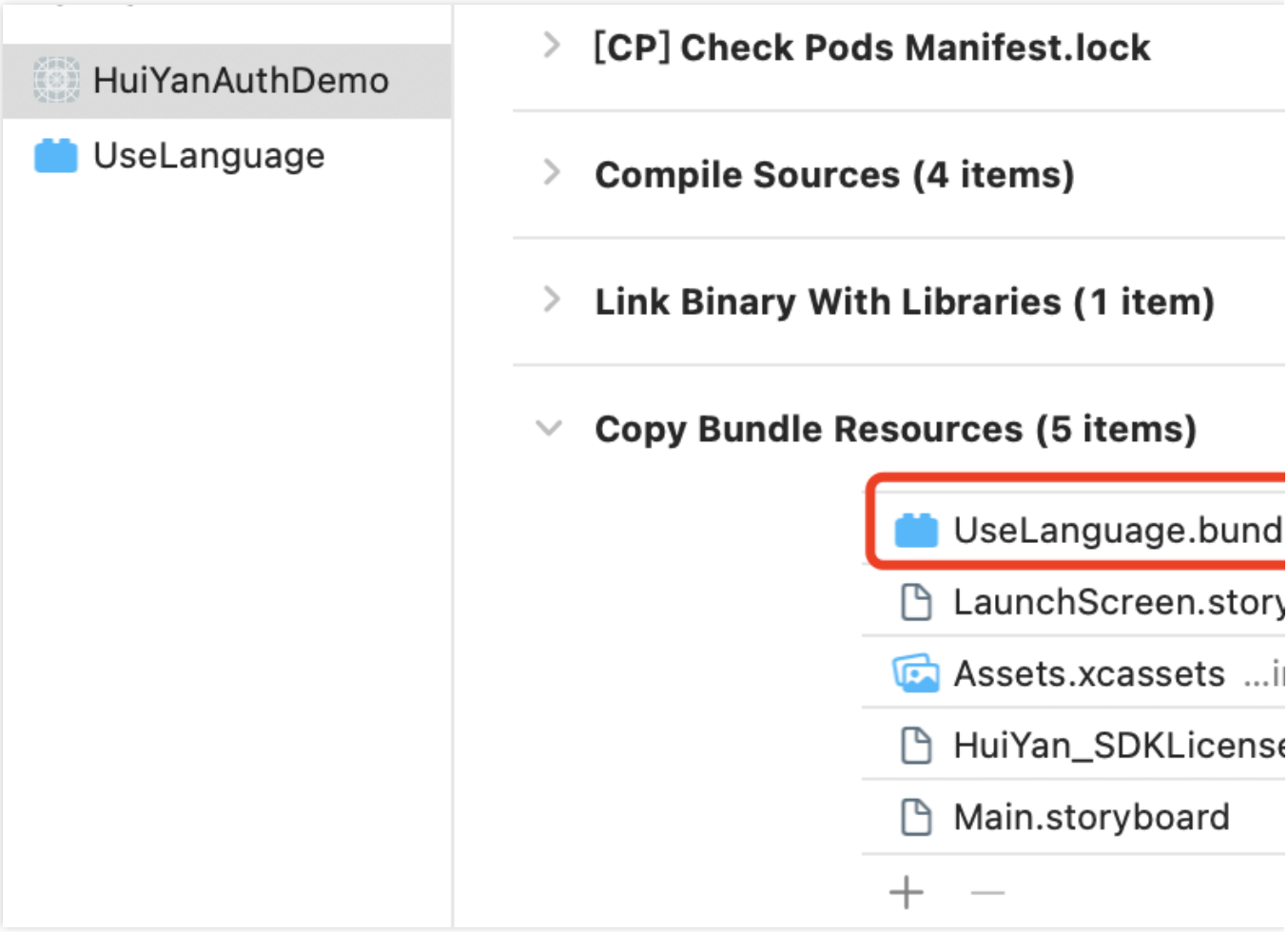
```
customerConfig.userLanguageBundleName = @"UserUIBundle";
customerConfig.userLanguageFileName = @"en.lproj";
```

userLanguageFileName可以查看编译出的bundle文件中对应的文件名

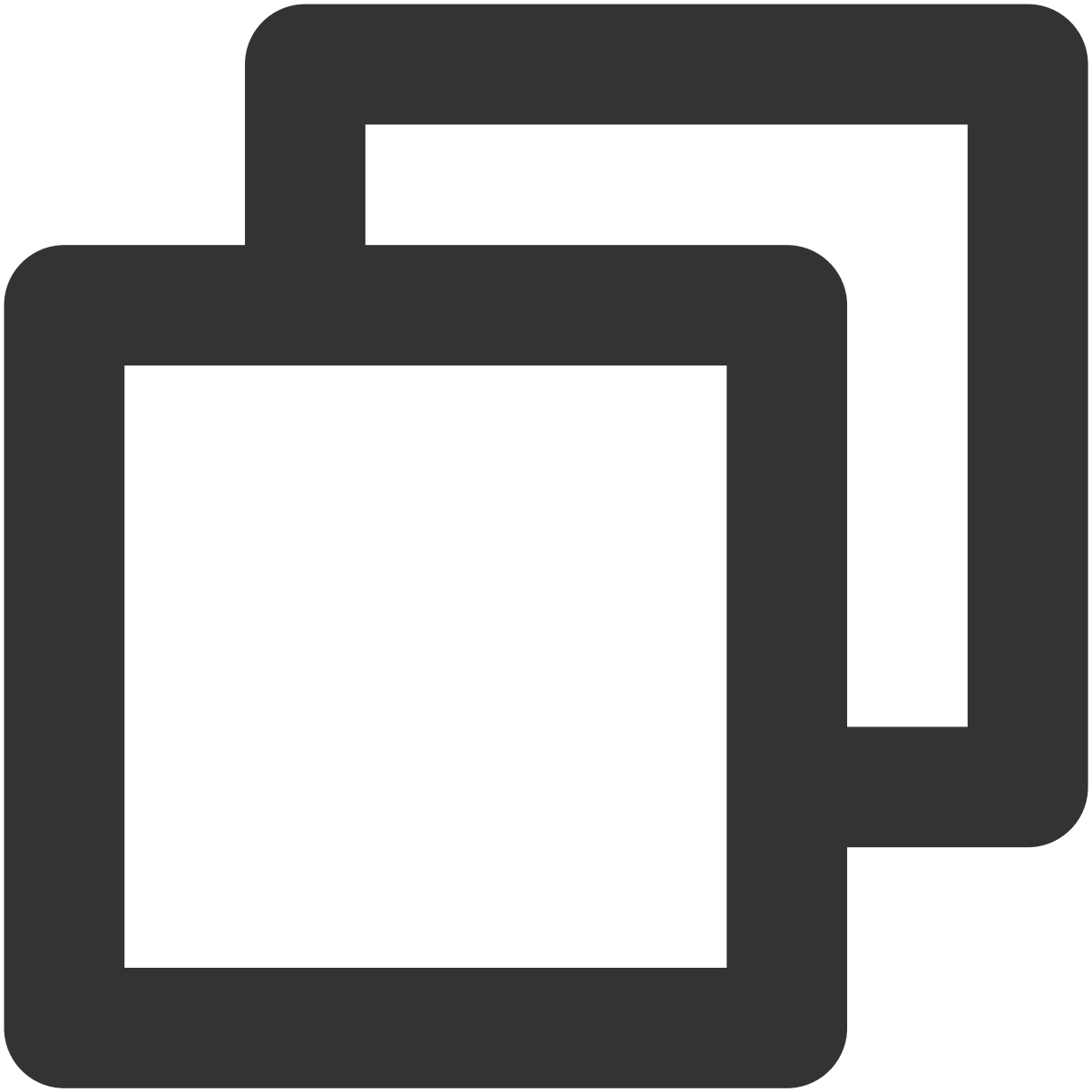


设置自定义SDK语言

1. 将自定义UseLanguage.bundle添加至项目中(Copy Bundle Resources)



2. 配置设置如下：



```
HuiYanOsConfig *config = [[HuiYanOsConfig alloc] init];
config.languageType = CUSTOMIZE_LANGUAGE;
config.userLanguageFileName = @"ko";//例：设置 ko.lproj
config.userLanguageBundleName = @"UseLanguage";//自定义打包bundle名称 例： UseLanguage
```

若config.languageType = DEFAULT;则会从自定义Bundle找该地区的语言文件，若是未找到则默认为en。

维护方式

将Demo工程作为自定义UI的工程，通过修改Demo工程里的bundle源文件，然后构建bundle接入自己项目。缺点需要自行维护Demo工程。

常见问题与修复指引

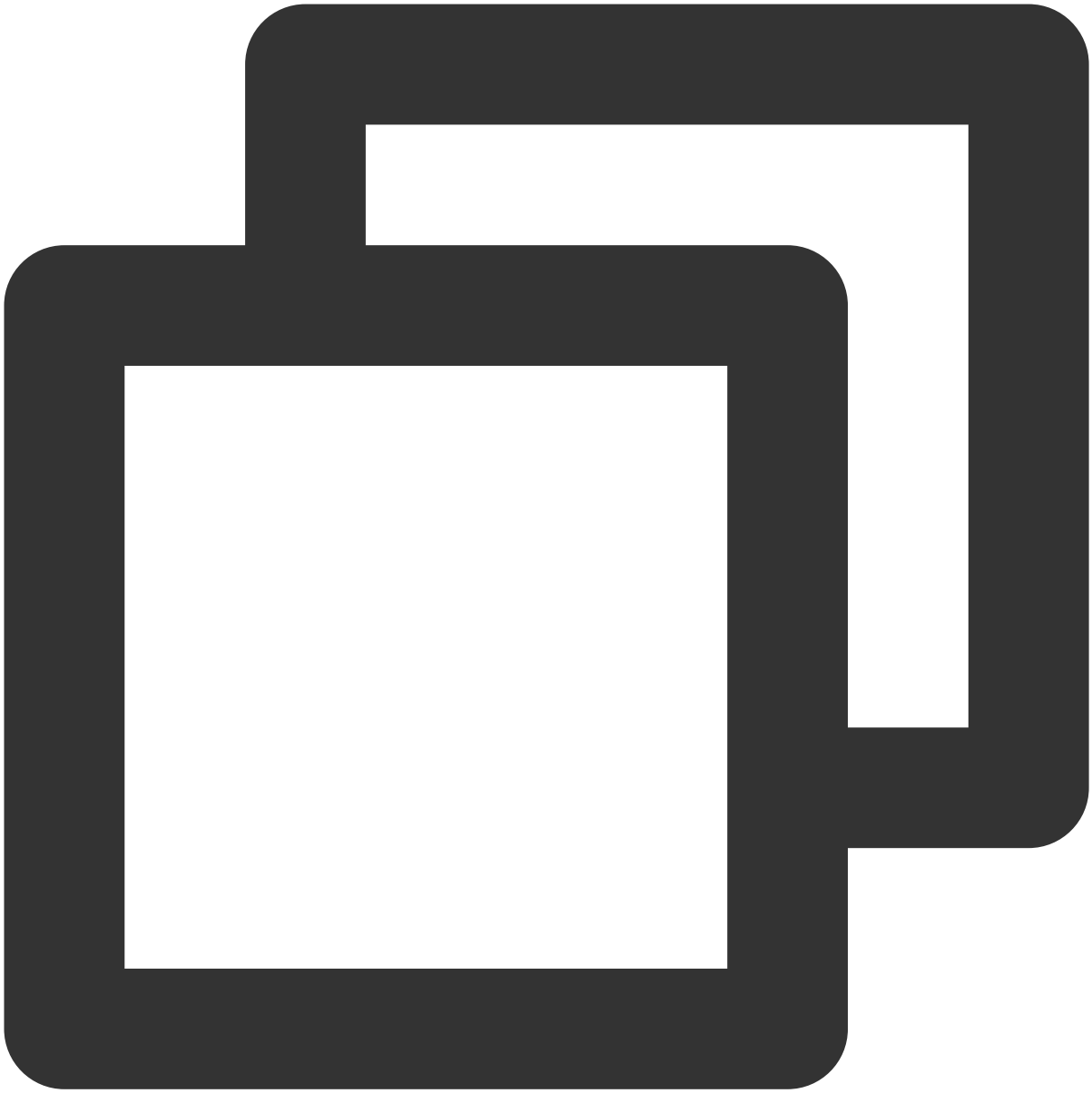
最近更新时间：2023-05-09 11:29:34

本文主要介绍活体人脸比对SDK的场景问题，并且给出相应的修复指引

客户端相关

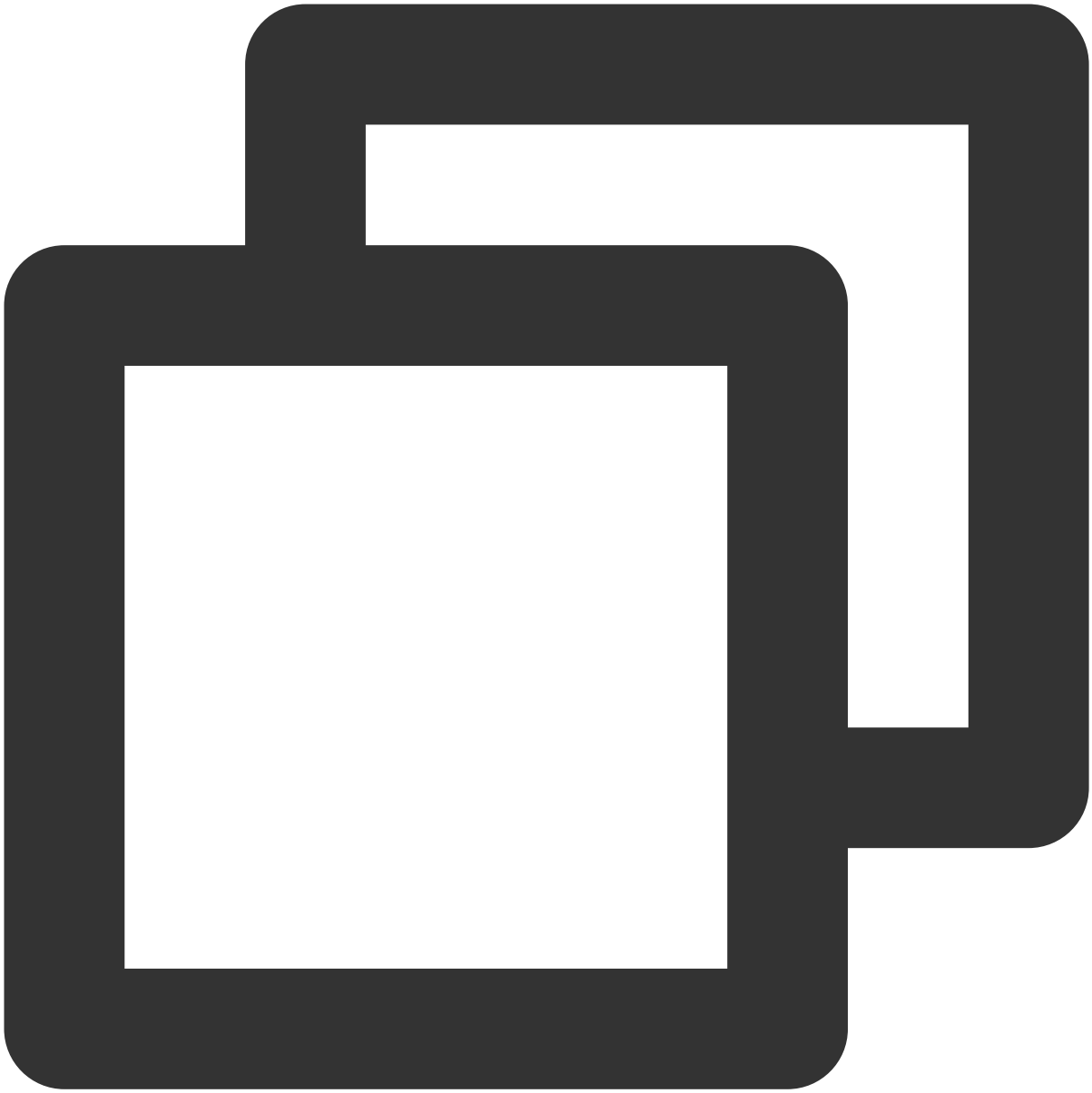
Android 端常见问题

1. 集成慧眼后出现**Invoke-customs are only supported starting with Android O (--min-api 26)**<错误？
需要在build.gradle中添加如下配置：



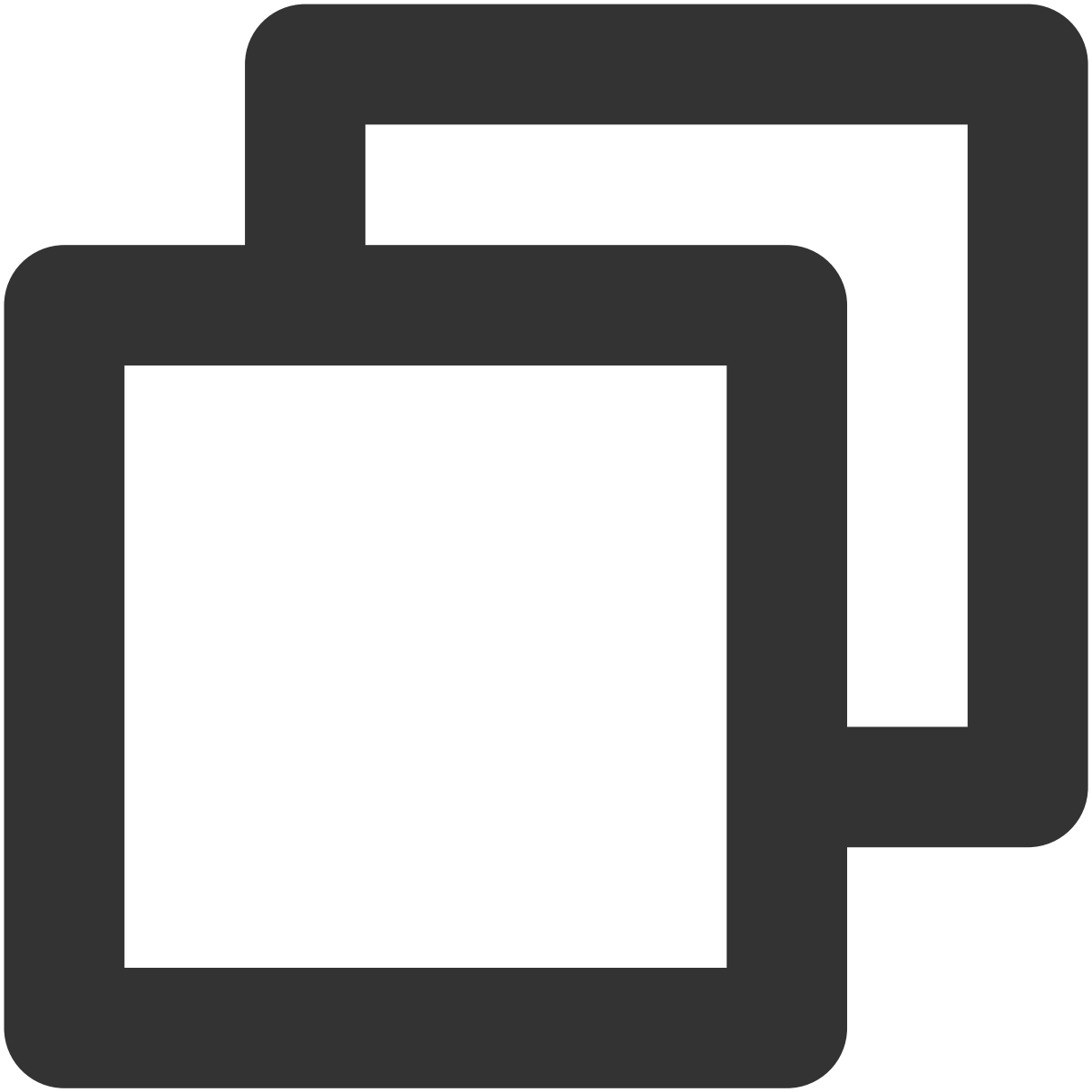
```
// java版本支持1.8
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

2. 如果集成方使用了AndResGuard的混淆工具，可以添加混淆配置：



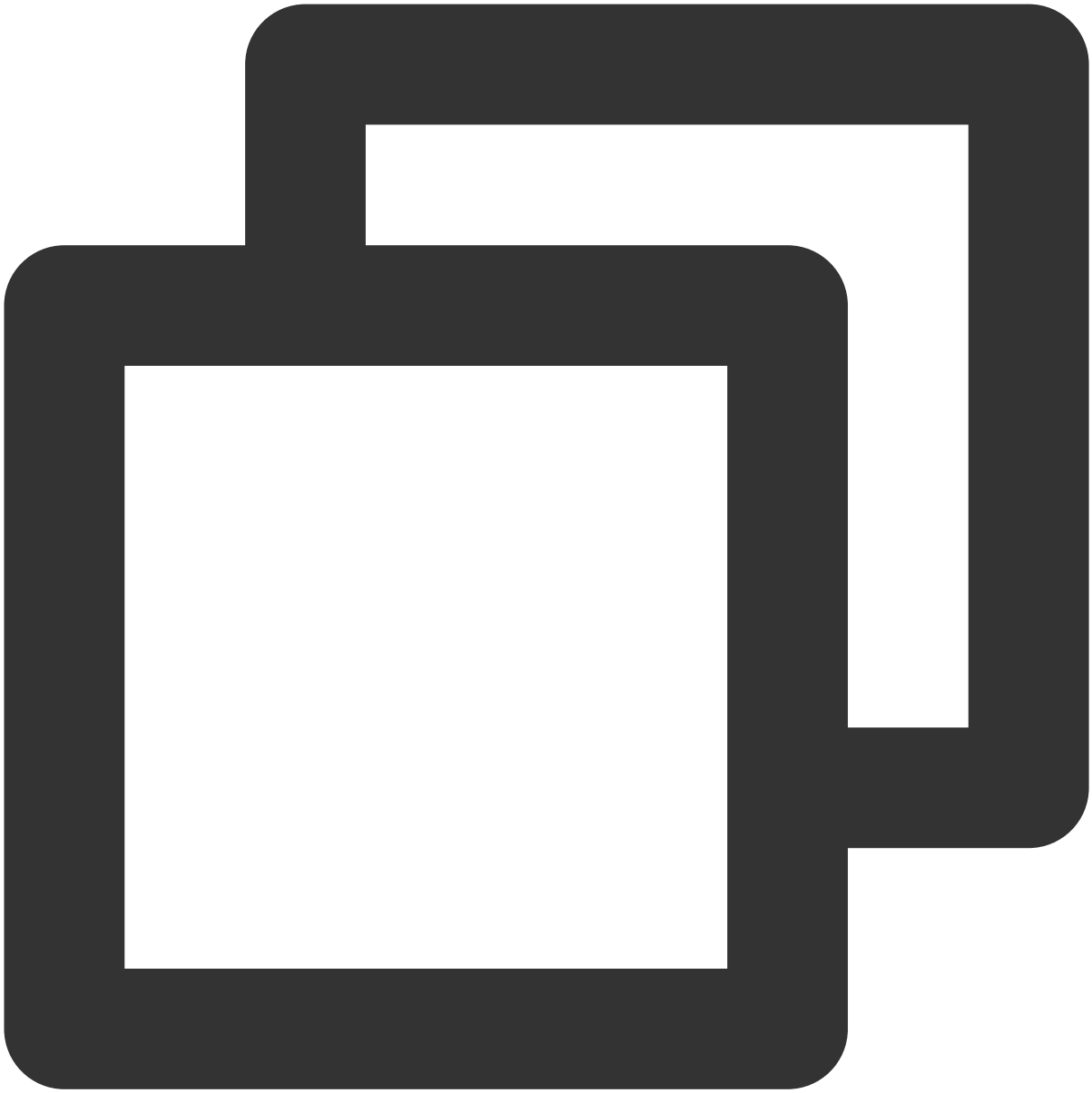
```
// for HuiYanSDK
"R.string.ocr_*",
"R.string.rst_*",
"R.string.net_*",
"R.string.msg_*",
"R.string.fl_*",
```

3. Android X针对一些低版本的设备报错 android.content.res.Resources$NotFoundException:from xml type xml resource ID #0x7f0800c3 这一类的错误，可以考虑添加矢量图依赖：



```
// 低版本的矢量图
implementation 'androidx.vectordrawable:vectordrawable:1.1.0'
```

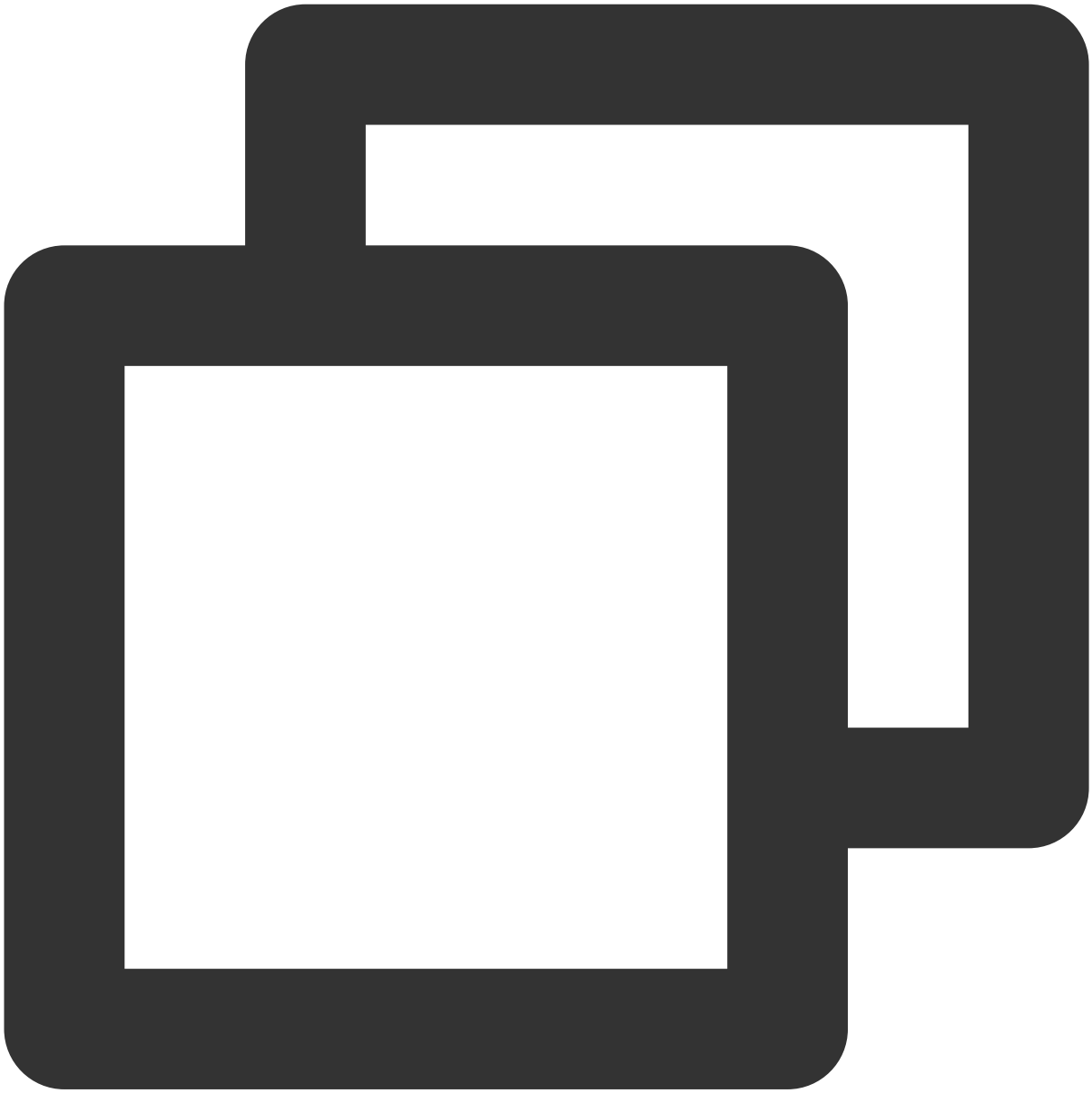
4. Android Support 如果在低版本设备上（6.0及以下）出现如下报错：



```
android.content.res.Resources$NotFoundException: File res/drawable/$txy_face_id_log
        at android.content.res.Resources.loadColorStateListForCookie(Resources.jav
        at android.content.res.Resources.loadColorStateList(Resources.java:2749)
        at android.content.res.TypedArray.getColor(TypedArray.java:441)
        at android.content.res.XResources$XTypedArray.getColor(XResources.java:128
        at android.support.v4.content.res.TypedArrayUtils.getNamedColor(TypedArray
        at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.update
        at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.inflat
        at android.support.graphics.drawable.VectorDrawableCompat.inflateInternal(
        at android.support.graphics.drawable.VectorDrawableCompat.inflate(VectorDr
        at android.support.graphics.drawable.VectorDrawableCompat.createFromXmlInn
```

```
        at android.support.v7.widget.AppCompatDrawableManager$VdcInflateDelegate.c
```

需要更新support依赖到最新版本28.0.0:



```
implementation 'com.android.support:appcompat-v7:28.0.0'
// 兼容低版本矢量图依赖的组件库
implementation 'com.android.support:support-vector-drawable:28.0.0'
implementation 'com.android.support:animated-vector-drawable:28.0.0'
```

iOS 端常见问题

1. 进入SDK，出现crash 打印日志 "**reason: '*** -[__NSDictionaryM setObject:forKey:]: key cannot be nil'"** ,处理方法是在build Settings - Other Linker Flags 添加 **-ObjC**

2. 编译时报：

Undefined symbol: _vImageConvert_Planar16FtoPlanarF

Undefined symbol: _vImageConvert_PlanarFtoPlanar16F

处理方法：添加系统库Accelerate.framework

3. "**face-tracker-v001 bundle path is nil**" 或者 **HuiYanSDKUI bundle path is nil** 添加两个bundle 资源文件至 Copy Bundle Resources

Identity Verification(App SDK) 接入指引

接入流程说明

最近更新时间：2024-08-01 16:43:36

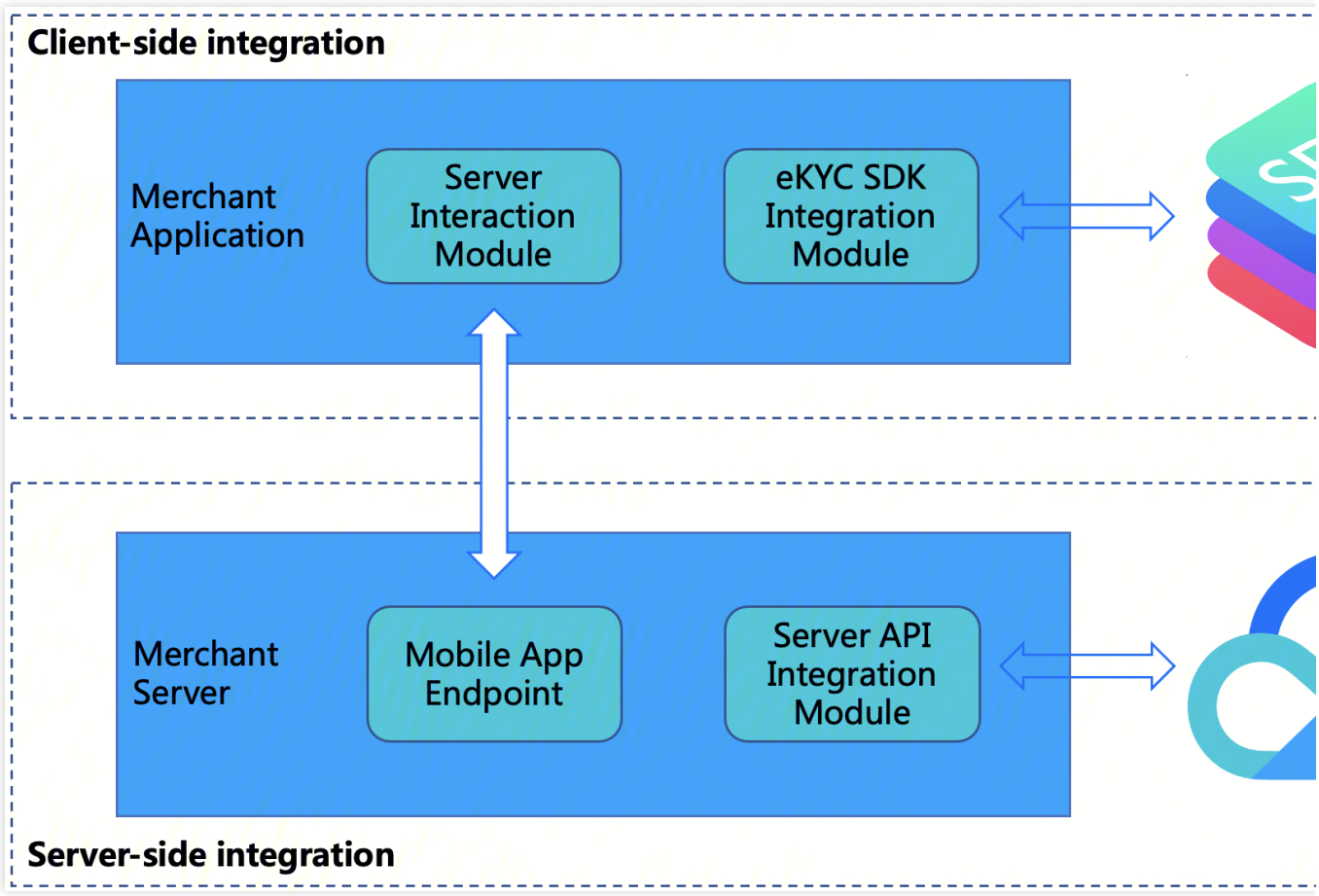
本文介绍Identity Verification(App SDK)整体接入的流程。

接入准备

注册腾讯云企业账号，请见注册指引
完成企业实名认证，请见企业实名指引
登录 eKYC 控制台开通服务
联系我们获取最新的 SDK 及 License

整体架构图

下图为腾讯云eKYC产品的Identity Verification(App SDK)集成的架构图：



SDK集成包括两部分：

A.客户端集成： 将SDK集成到客户终端业务App中。

B.服务器端集成： 在您的（商家）服务器中公开您的（商家）应用程序的端点，以便商家应用程序可以与商家服务器交互，然后访问 Server API 以获取串联活体比对流程的SdkToken以及通过SdkToken拉取最终的核身结果。

整体交互流程

集成方只需要传入Token并启动对应的Identity Verification(App SDK)，便可以实现包含证件识别+活体检测+人脸比对全流程的用户身份认证，待终端用户完成认证后，集成方可通过API接口获取完整的认证结果。

获取Token的API接口：ApplySdkVerificationToken

拉取身份认证结果的API接口: GetSdkVerificationResult

下图展示了SDK、客户端以及服务器端的整体交互逻辑，图中负责模块解析：

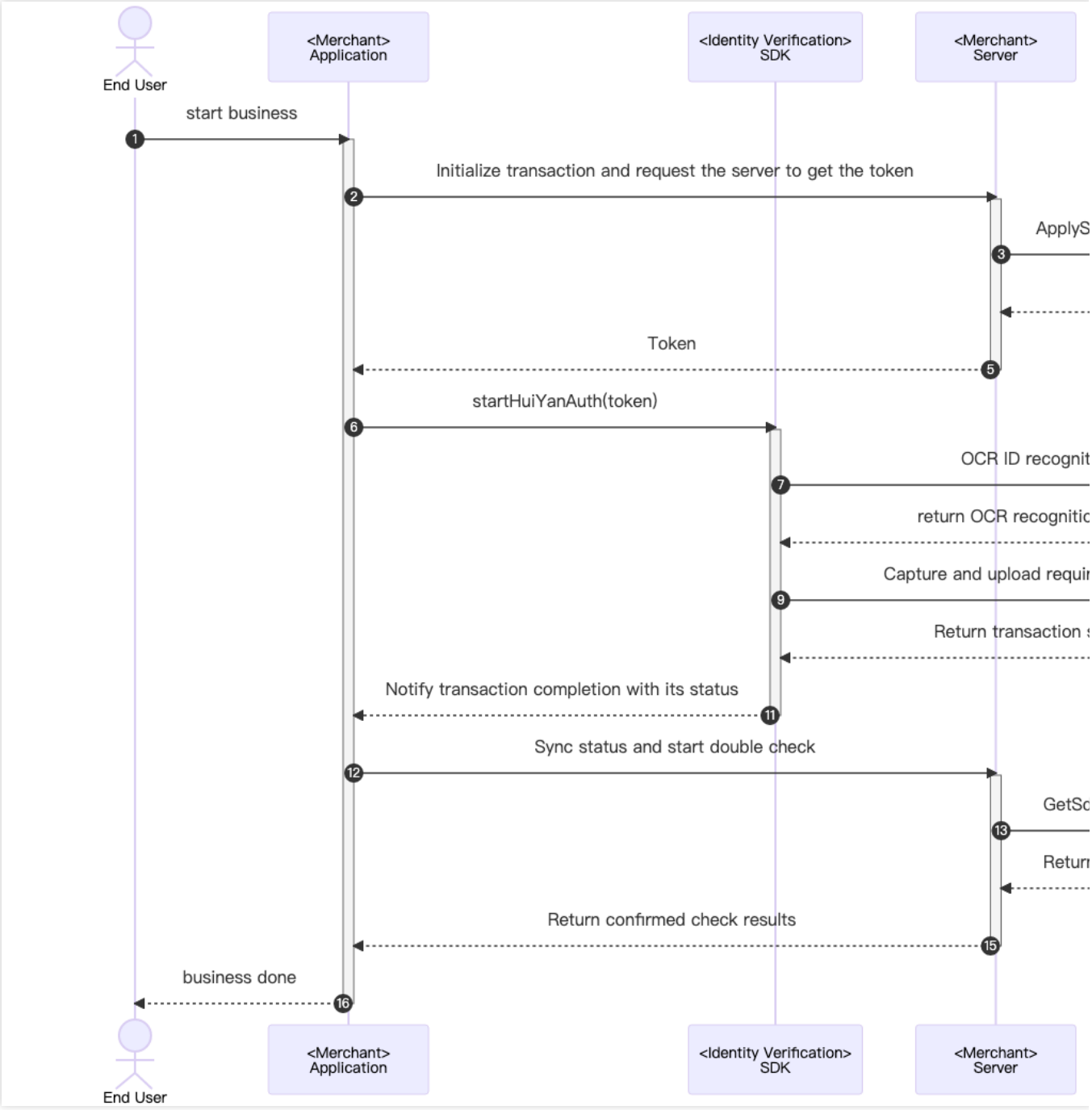
End User：终端用户

Identity Verification(App SDK)：在准备阶段获取到的Identity Verification(App SDK)

Merchant Application：使用并集成Identity Verification SDK的客户终端业务App

Merchant Server：客户的服务器的程序

Identity Verification Server：腾讯云Identity Verification后台服务API接口



具体的推荐交互流程详细交互如下：

1. 用户触发终端 Merchant Application 准备调用核身业务场景。
2. Merchant Application 发送请求到 Merchant Server，告知启动一次核身业务需要活体业务Token。
3. Merchant Server 传入相关参数调用云API接口ApplySdkVerificationToken。
4. Identity Verfication Server 接收到ApplySdkVerificationToken调用后，下发此次业务的token给 Merchant Server。
5. Merchant Server可以将获取到的业务Token，下发给客户的 Merchant Application。

6. Merchant Application 调用Identity Verfication SDK的启动接口**startHuiYanAuth**传入token与配置信息，开始核身验证。
7. Identity Verfication SDK 启动OCR，将证件照片上传至Identity Verfication Server识别用户证件信息。
8. Identity Verfication Server将证件识别结果返回给Identity Verfication SDK。
9. Identity Verfication SDK 捕捉并上传所需的用户数据，包括活体数据等，上传至 Identity Verfication Server 。
10. Identity Verfication Server 在完成核身检测（包括活体与比对流程）以后，会返回结果给 Identity Verfication SDK 。
11. Identity Verfication SDK 主动触发回调给 Merchant Application 通知核验完成以及核验状态。
12. Merchant Application 接收到回调以后，可以发送请求通知 Merchant Server去主动获取本次核身的结果，进行确认检查。
13. Merchant Server主动调用 Identity Verfication Server 的接口GetSdkVerificationResult传入相关参数以及本次业务的Token，去获取本次核身的结果。
14. Identity Verfication Server 接收到GetSdkVerificationResult调用后，会返回本次核身的结果到 Merchant Server。
15. Merchant Server接收到本次核身的结果后，可以下发需要的信息到 Merchant Application。
16. Merchant Application 展示最后的结果，呈现在UI界面上，告知用户认证的结果。

接入流程

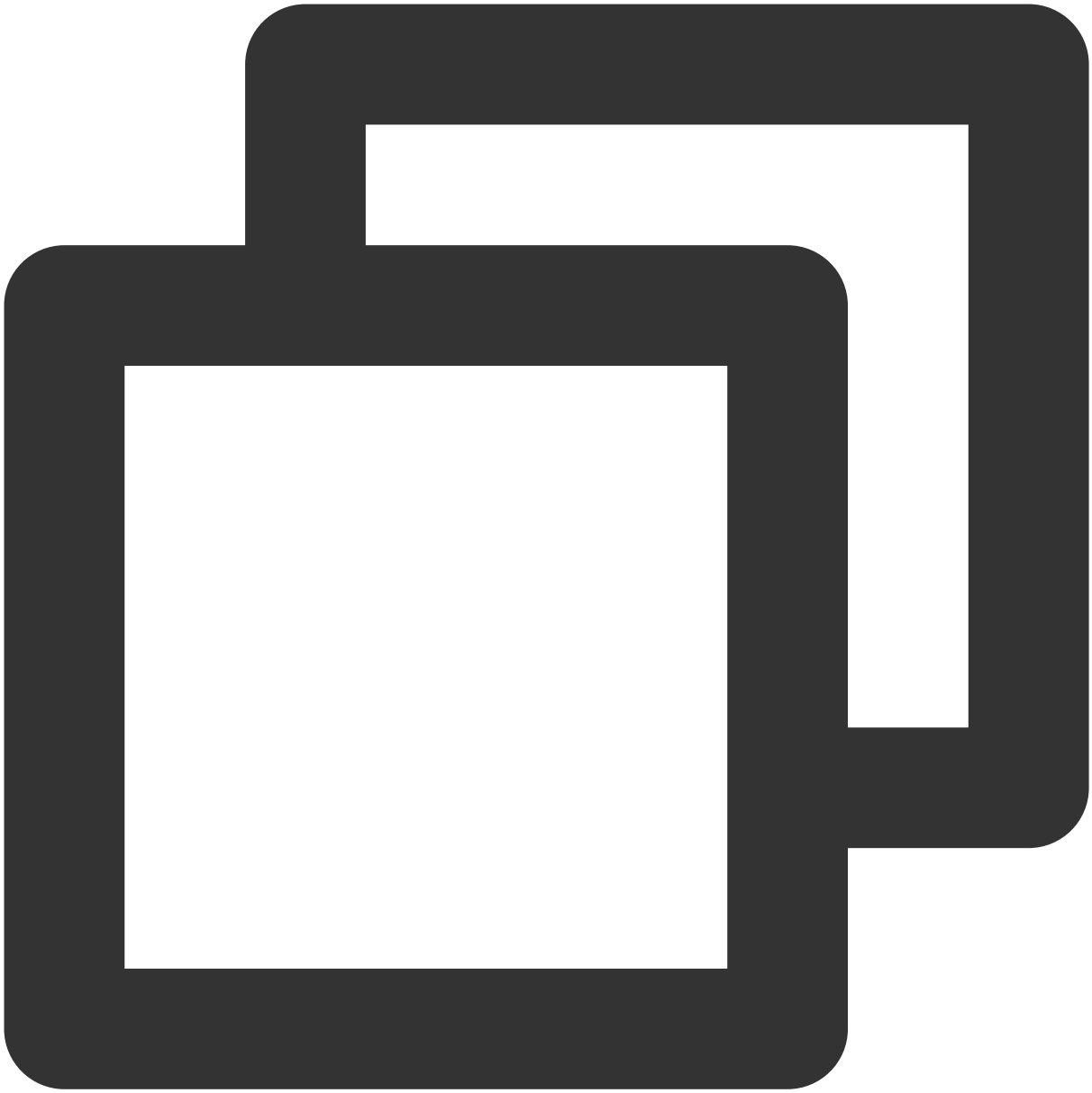
服务器端集成

集成准备

在服务端集成之前，你需要按照获取API秘钥指引中的说明进行操作，开通腾讯云慧眼服务，并且拿到了腾讯云api访问秘钥SecretId和SecretKey。除此之外你还需要按照连接腾讯云API接口中的操作流程，引入你所熟悉开发语言的SDK包到你服务端模块中，以确保可以成功调用腾讯云API以及正确处理API请求和响应。

开始集成

为了确保你的（商户）客户端应用程序能够跟你的（商户）服务端正常交互，商户服务端需要调用慧眼提供的API接口 **ApplySdkVerificationToken** 获取SDKToken串联身份认证全流程以及调用 **GetSdkVerificationResult** 接口获取身份认证结果，商户服务端需要提供相应的端点给商户客户端调用，下面的示例代码使用 Golang语言作为案例，展示如何在服务端调用腾讯云API接口并拿到正确的响应结果。
注意： 该示例中仅仅演示商户服务端与腾讯云API服务交互所需要的处理逻辑，如果有需要的话你需要自己实现你的业务逻辑，比如：
当你通过 **ApplySdkVerificationToken** 接口获取SDKToken之后，可以将客户端应用程序需要的其他响应同SDKToken一并返回给客户端。
当你通过 **GetSdkVerificationResult** 接口获取身份认证结果之后，可以将返回的最佳帧照片保存起来，以便后续业务逻辑使用。



```
var FaceIdClient *faceid.Client

func init() {
        // 初始化客户端配置对下，您可以指定超时时间和其他配置项
        prof := profile.NewClientProfile()
        prof.HttpProfile.ReqTimeout = 60
        // TODO 需要替换成您调用账号的 SecretId 和 SecretKey
        credential := cloud.NewCredential("SecretId", "SecretKey")
        var err error
        // 初始化调用慧眼人脸核身服务的客户端
        FaceIdClient, err = faceid.NewClient(credential, "ap-singapore", prof)
```

```
        if nil != err {
                log.Fatal("FaceIdClient init error: ", err)
        }
}

// ApplySdkVerificationToken 获取人脸核身Token
func ApplySdkVerificationToken(w http.ResponseWriter, r *http.Request) {
        log.Println("get face id token")
        // 步骤1: 解析请求参数
        _ = r.ParseForm()
        var IdCardType = r.FormValue("IdCardType")
        var NeedVerifyIdCard = false

        // 步骤2: 初始化请求对象，并给必要的参数赋值
        request := faceid.NewApplySdkVerificationTokenRequest()
        request.IdCardType = &IdCardType
        request.NeedVerifyIdCard = &NeedVerifyIdCard
        // 步骤3: 通过FaceIdClient调用人脸核身服务
        response, err := FaceIdClient.ApplySdkVerificationToken(request)

        // 步骤4: 处理腾讯云API的响应，并构造返回对象
        if nil != err {
                log.Println("error: ", err)
                _, _ = w.Write([]byte("error"))
                return
        }
        SdkToken := response.Response.SdkToken
        apiResp := struct {
                SdkToken *string
        }{SdkToken: SdkToken}
        b, _ := json.Marshal(apiResp)

    // ... 其他业务处理代码

    //步骤5: 返回服务响应
        _, _ = w.Write(b)
}

// GetFaceIdResult 获取人脸核身核验结果
func GetSdkVerificationResult(w http.ResponseWriter, r *http.Request) {
    // 步骤1: ... 解析请求参数
        _ = r.ParseForm()
        SdkToken := r.FormValue("SdkToken")
    // 步骤2: 初始化请求对象，并给必要的参数赋值
        request := faceid.NewGetSdkVerificationResultRequest()
        request.SdkToken = &SdkToken
    // 步骤3: 通过FaceIdClient调用人脸核身服务
```

```
        response, err := FaceIdClient.GetSdkVerificationResult(request)

          // Step 4: 处理腾讯云API的响应，并构造返回对象
        if nil != err {
                _, _ = w.Write([]byte("error"))
                return
        }
        result := response.Response.Result
        apiResp := struct {
                Result *string
        }{Result: result}
        b, _ := json.Marshal(apiResp)

    // ... 其他业务处理代码

    //步骤5: 返回服务响应
        _, _ = w.Write(b)
}

func main() {
        // 注册http接口路径
        http.HandleFunc("/api/v1/get-token", ApplySdkVerificationToken)
        http.HandleFunc("/api/v1/get-result", GetSdkVerificationResult)
        // 监听端口
        err := http.ListenAndServe(":8080", nil)
        if nil != err {
                log.Fatal("ListenAndServe error: ", err)
        }
```

说明：完整的代码示例详见faceid-server-demo。

接口测试

当你完成了集成之后，可以通过postman或者curl命令来测试当前的集成是否正确，通过访问http://ip:port/api/v1/get-token 接口查看是否正常返回 SdkToken , 访问http://ip:port/api/v1/get-result 接口查看 Result 字段的响应是否为0以判断服务端集成是否成功，具体的响应结果详见API接口部分的介绍。

Android 端集成

依赖环境

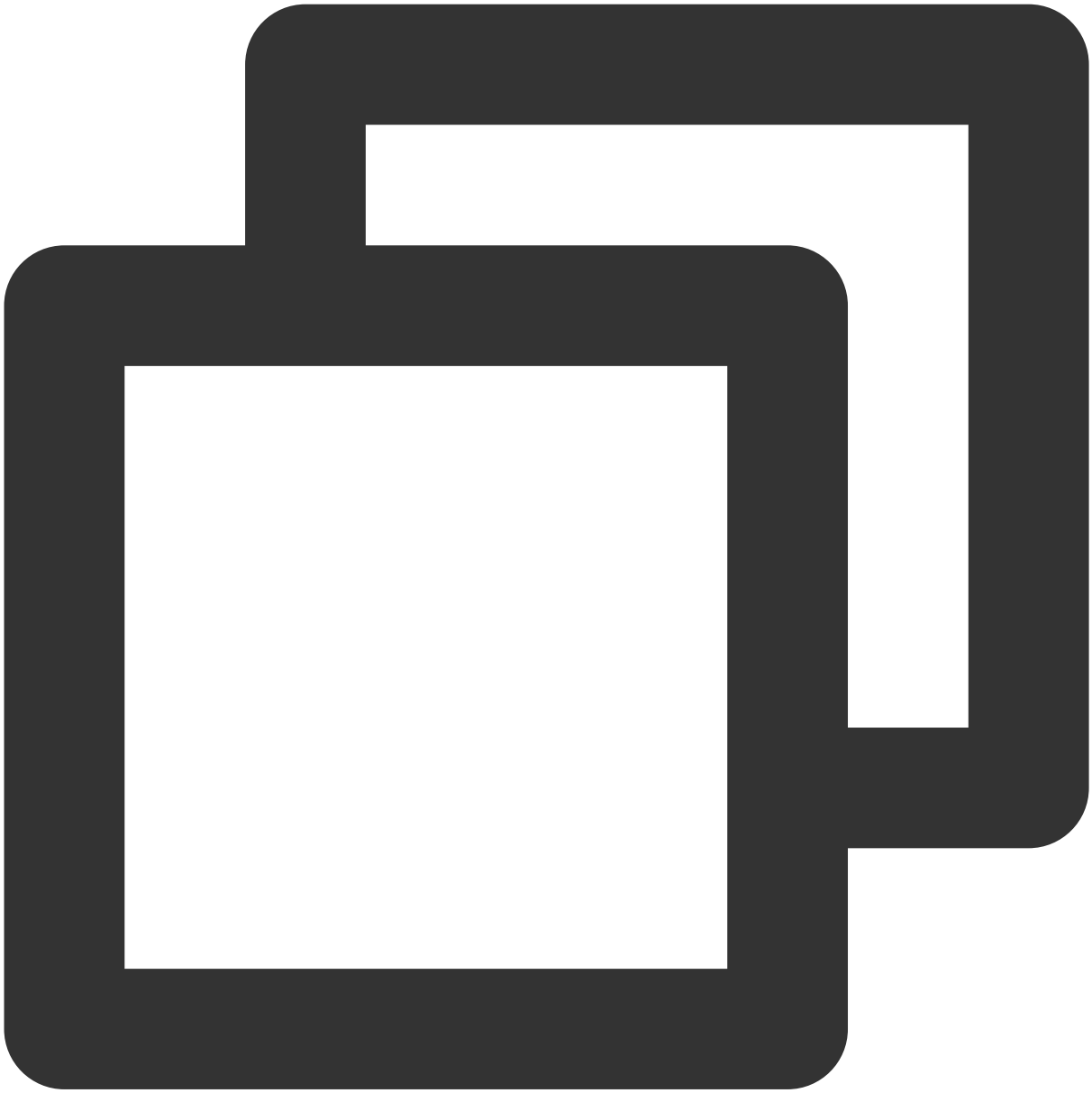
当前Android端SDK适用于API 19 (Android 4.4) 及以上版本。

SDK 接入步骤

1. 将**ekyc_android_1.0.x.x_release.aar、huiyansdk_android_1.0.x.x_release.aar、OcrSDK-private-v1.0.x.x-release.aar、OcrSDK-common-model-v1.0.x.x-release.aar、tencent-ai-sdk-youtu-base-v1.0.x.x-**

release.aar、tencent-ai-sdk-common-v1.0.x.x-release.aar 、tencent-ai-sdk-aicamera-v1.0.x.x-release.aar(具体版本号以官网下载为准)添加到您工程的libs目录下。

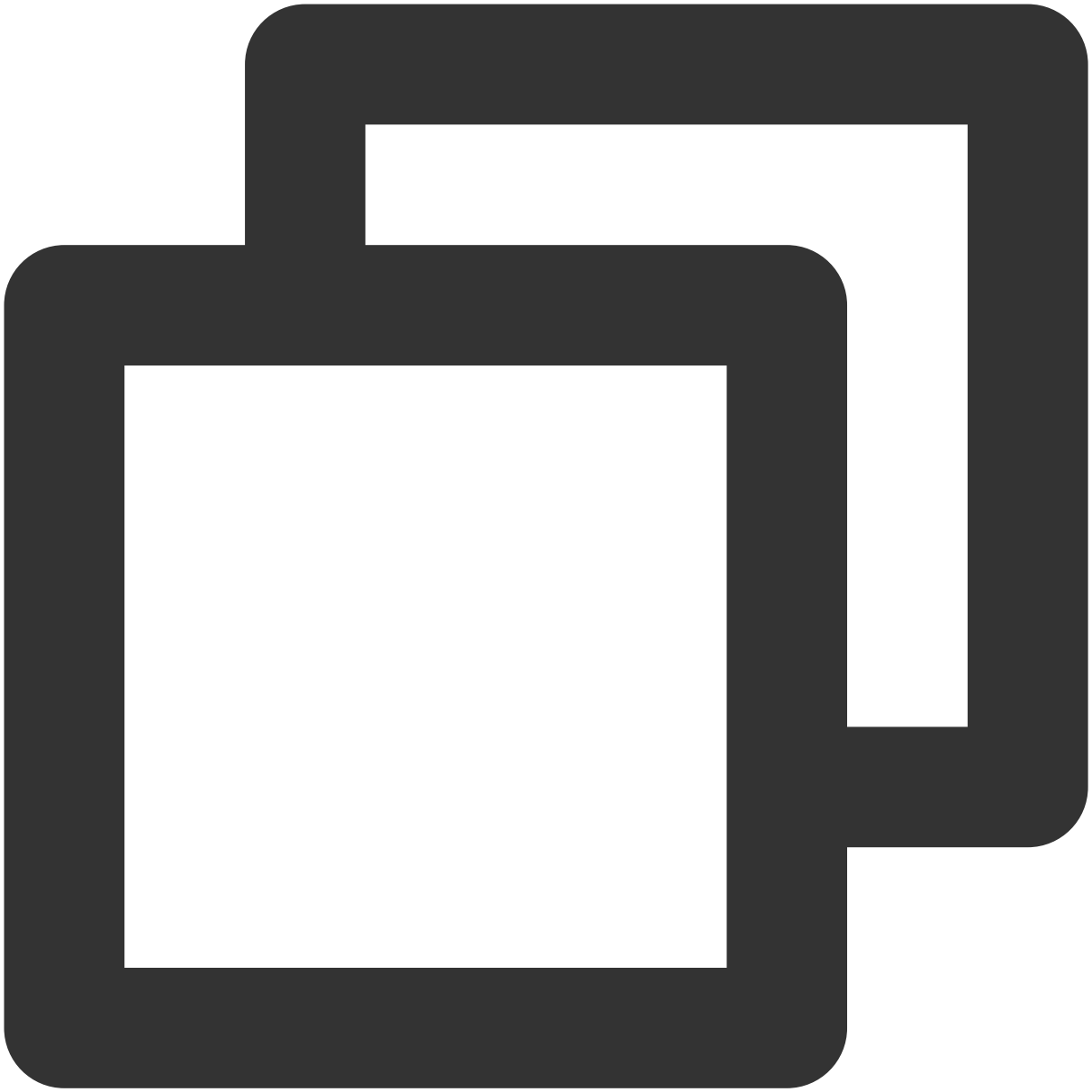
2. 在您工程的**build.gradle**中进行如下配置：



```
// 设置ndk so架构过滤(以armeabi-v7a为例)
ndk {
    abiFilters 'armeabi-v7a'
}

dependencies {
    // Identity verification SDK版本库
```

```
    implementation files("libs/ekyc_android_1.0.x.x_release.aar")
    // 核身组件库
    implementation files("libs/huiyansdk_android_1.0.x.x_release.aar")
    // Ocr组件库
    implementation files("libs/OcrSDK-common-model-v1.x.x-release.aar")
    implementation files("libs/OcrSDK-private-v2.x.x-release.aar")
    // 通用能力组件库
    implementation files("libs/tencent-ai-sdk-youtu-base-1.0.x.x-release.aar")
    implementation files("libs/tencent-ai-sdk-common-1.x.x-release.aar")
    implementation files("libs/tencent-ai-sdk-aicamera-1.x.x-release.aar")
    // 引入gson三方库
    implementation 'com.google.code.gson:gson:2.8.5'
}
```

3. 同时需要在AndroidManifest.xml文件中进行必要的权限声明



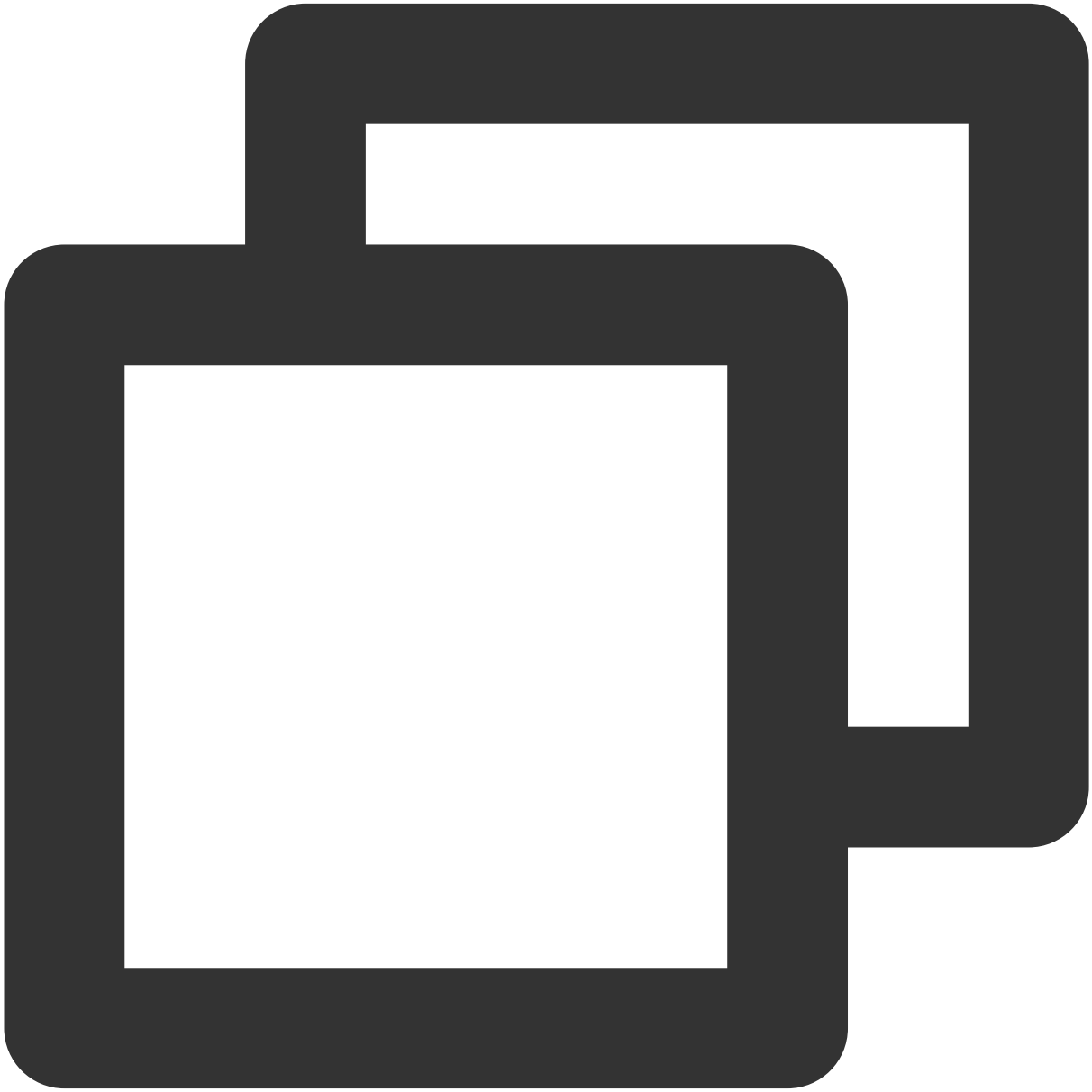
```
<!-- 摄像头权限 -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />
<uses-feature android:name="android.hardware.camera.autofocus" />

<!-- SDK需要的权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<!-- SDK可选权限 -->
```

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

对于需要兼容Android 6.0以上的用户，以上权限除了需要在AndroidManifest.xml文件中声明权以外，还需使用代码动态申请权限。

初始化接口

在您APP初始化的时候调用，推荐在Application中调用，主要是进行一些SDK的初始化操作

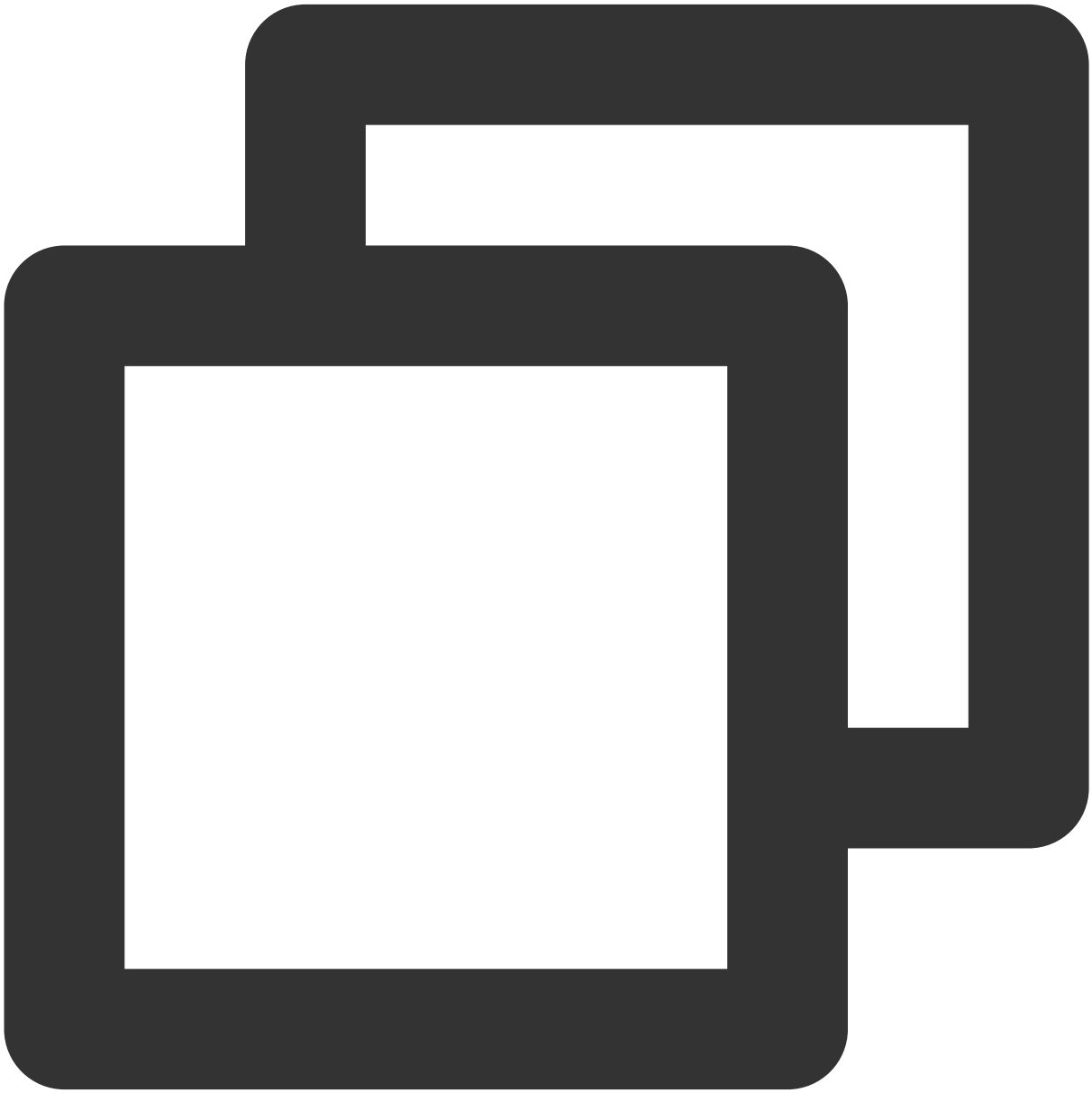


```
@Override
public void onCreate() {
```

```
        super.onCreate();
        EkycHySdk.init(this);
}
```

启动身份认证（Identity verification）检测流程

当您需要启动身份认证检测流程的时候，只需要调用EkycHySdk.startEkycCheck()函数，同时传入此次身份认证检测需要的sdkToken、配置信息以及监听结果的回调即可。

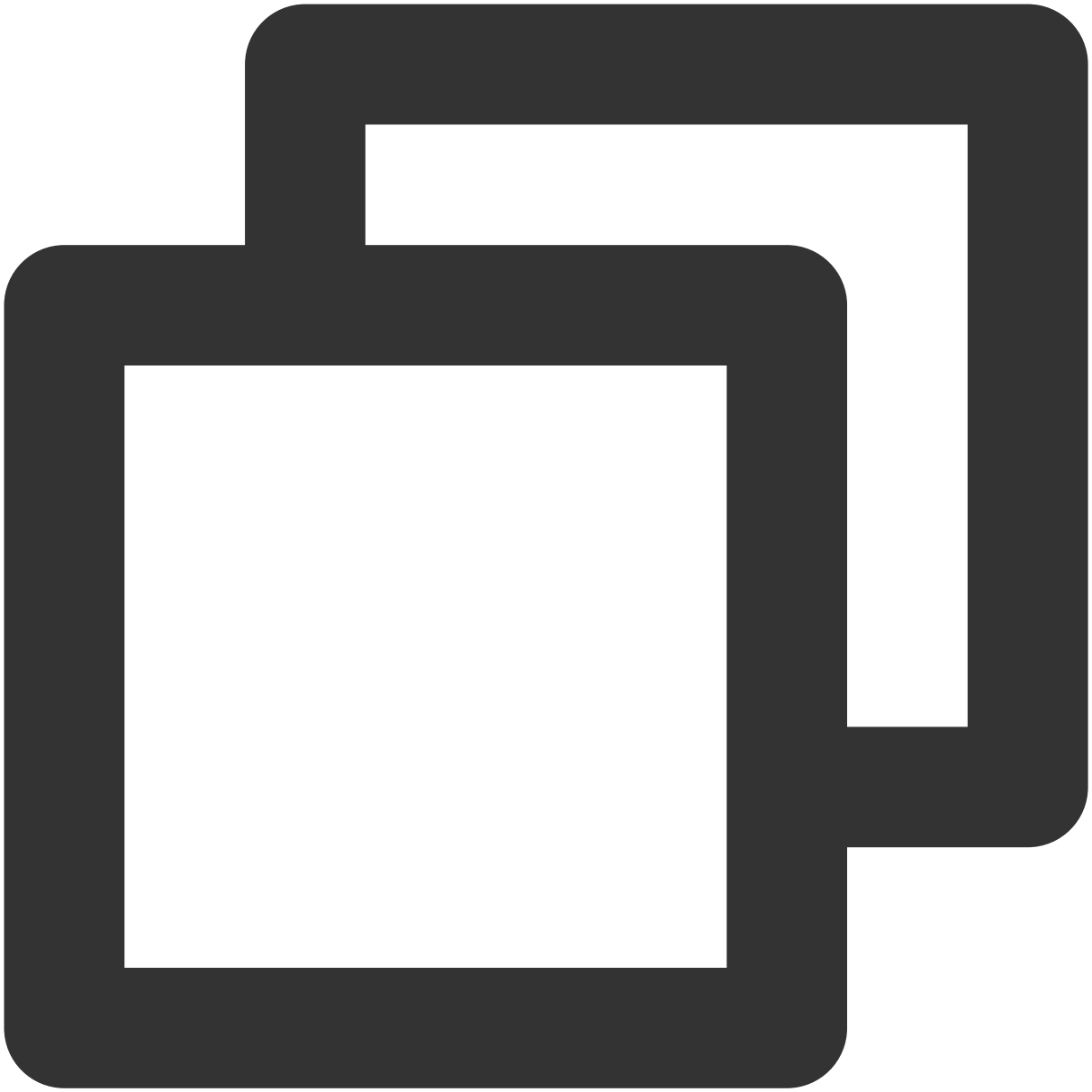


```
// 设置启动的配置
EkycHyConfig ekycHyConfig = new EkycHyConfig();
```

```
// 设置license的名称
ekycHyConfig.setLicenseName("ekycLicense.license");
ekycHyConfig.setOcrType(OcrRegionType.HK);
// 自定义UI配置
OcrUiConfig config = new OcrUiConfig();
ekycHyConfig.setOcrUiConfig(config);
// 具体的启动核验逻辑
// sdkToken 是从服务器端获取的当次流程的唯一凭证
EkycHySdk.startEkycCheck(sdkToken, ekycHyConfig, new EkycHyCallBack() {
  @Override
  public void onSuccess(EkycHyResult result) {
    Log.e(TAG, "result: " + result.toString());
    showToast("核身成功: " + result.toString());
  }

  @Override
  public void onFail(int errorCode, String errorMsg, String ekycToken) {
    Log.e(TAG, "code: " + errorCode + " msg: " + errorMsg + " token: " + ekycToken)
    showToast("核身失败: " + "code: " + errorCode + " msg: " + errorMsg + " token: "
  }
});
```

sdkToken 为从服务器兑换的本次身份认证流程的唯一凭证。

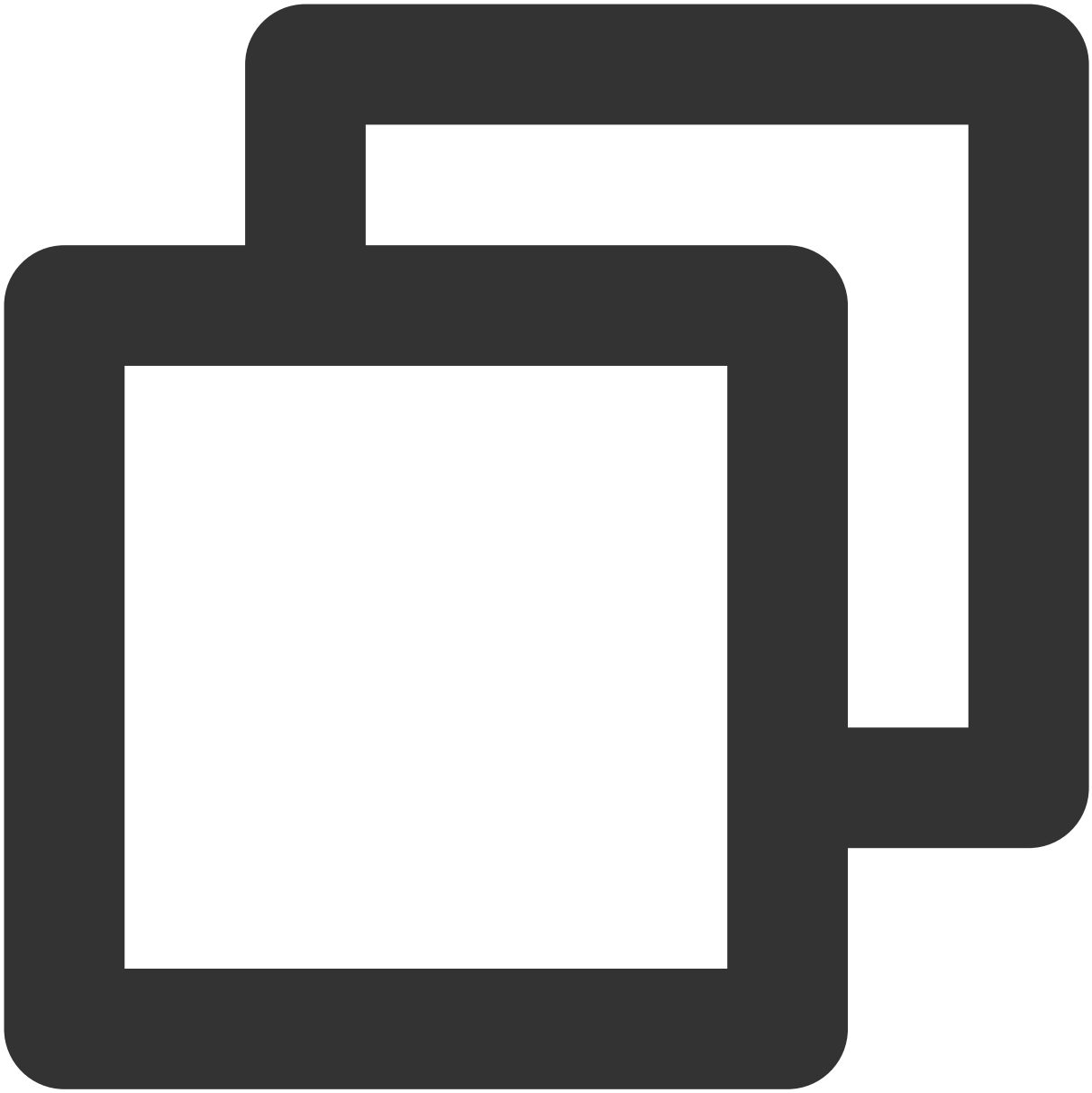
注意： **"ekycLicense.license"**文件是需要联系商务或者客服人员进行license申请。将申请完成后的license文件放到assets文件下。



```
└── src
    └── main
        ├── assets
        │   └── ekycLicense.license
```

SDK 资源释放

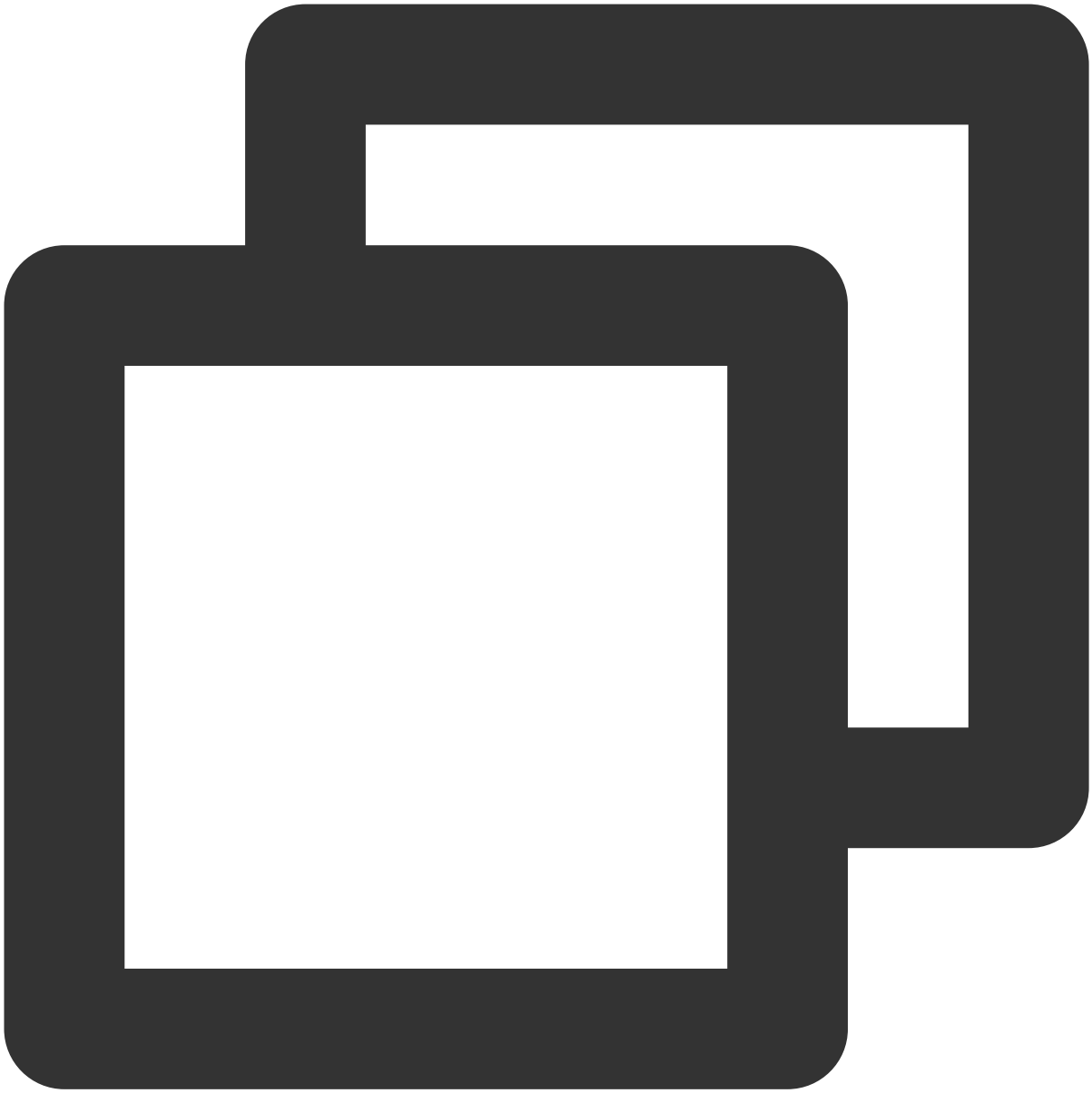
在您APP退出使用的时候，可以调用SDK资源释放接口



```
@Override
protected void onDestroy() {
  EkycHySdk.release();
  super.onDestroy();
}
```

混淆规则配置

如果您的应用开启了混淆功能，为确保SDK的正常使用，请把以下部分添加到您的混淆文件中。



```
# 需要混淆的对象
-keep class com.google.gson.** {*;}
-keep class com.tencent.could.** {*;}
-keep class com.tencent.youtu.** {*;}
-keep class com.tencent.cloud.ocr.** {*;}
-keep class com.tencent.cloud.ekyc.** {*;}
```

说明：Android完整的代码示例详见Android demo。

iOS 端集成

依赖环境

1. 开发环境 Xcode 12.0 及以上版本，推荐使用最新的版本。
2. SDK 适用于iOS11.0及以上版本。

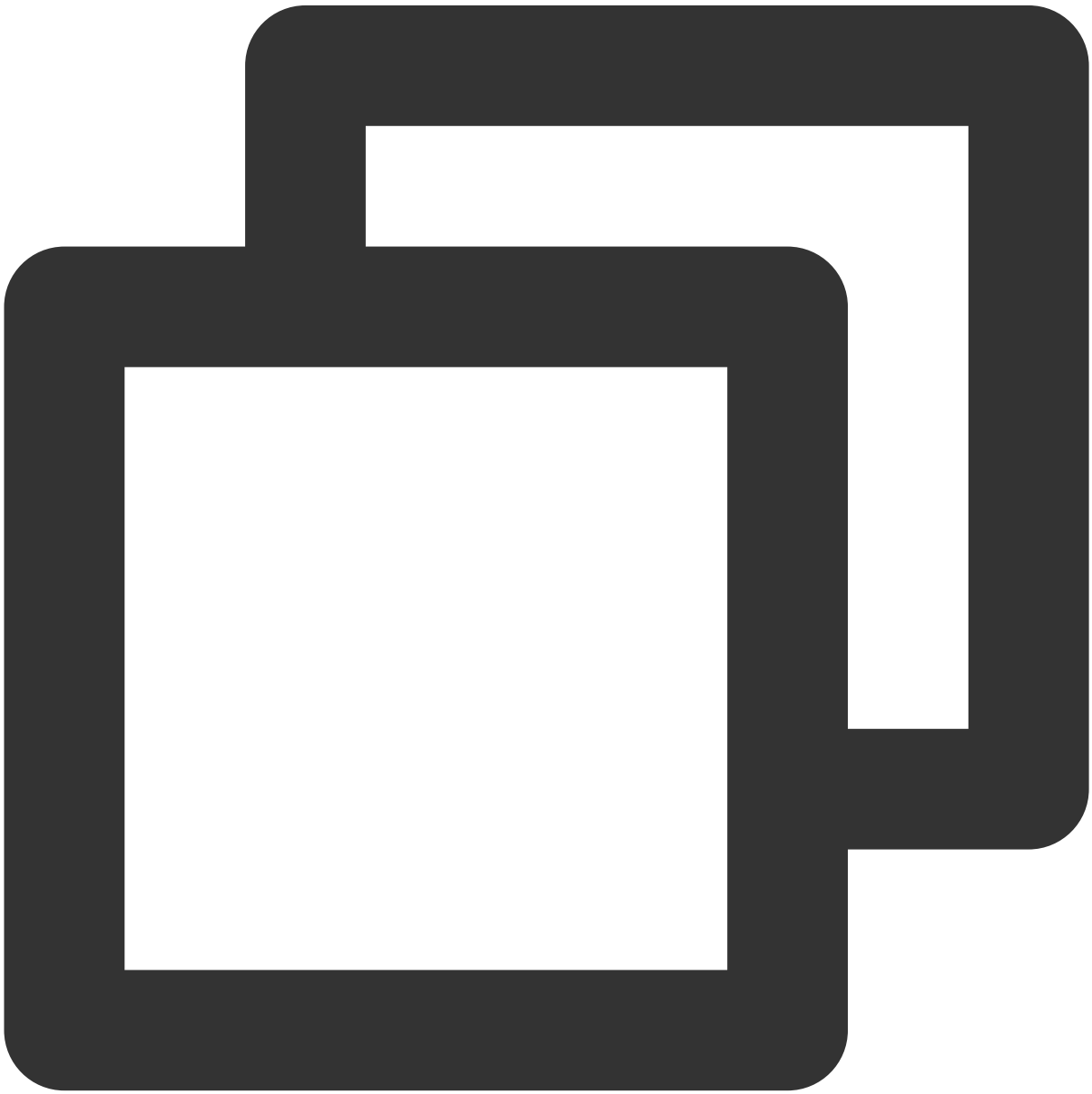
SDK 接入步骤

手动接入方式

1. 导入相关库及文件

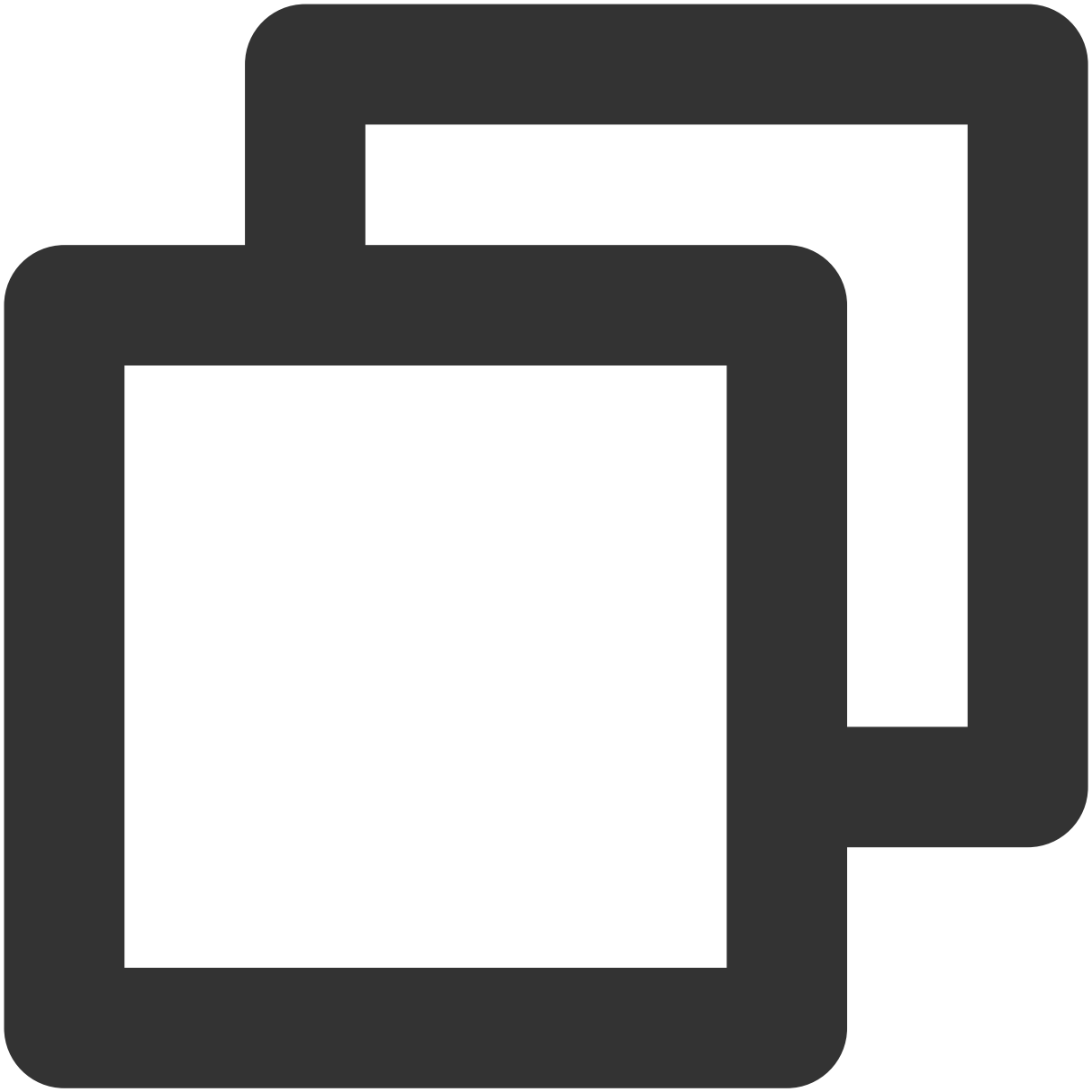
Link Binary With Libraries导入相关Framework

2. SDK依赖的库如下



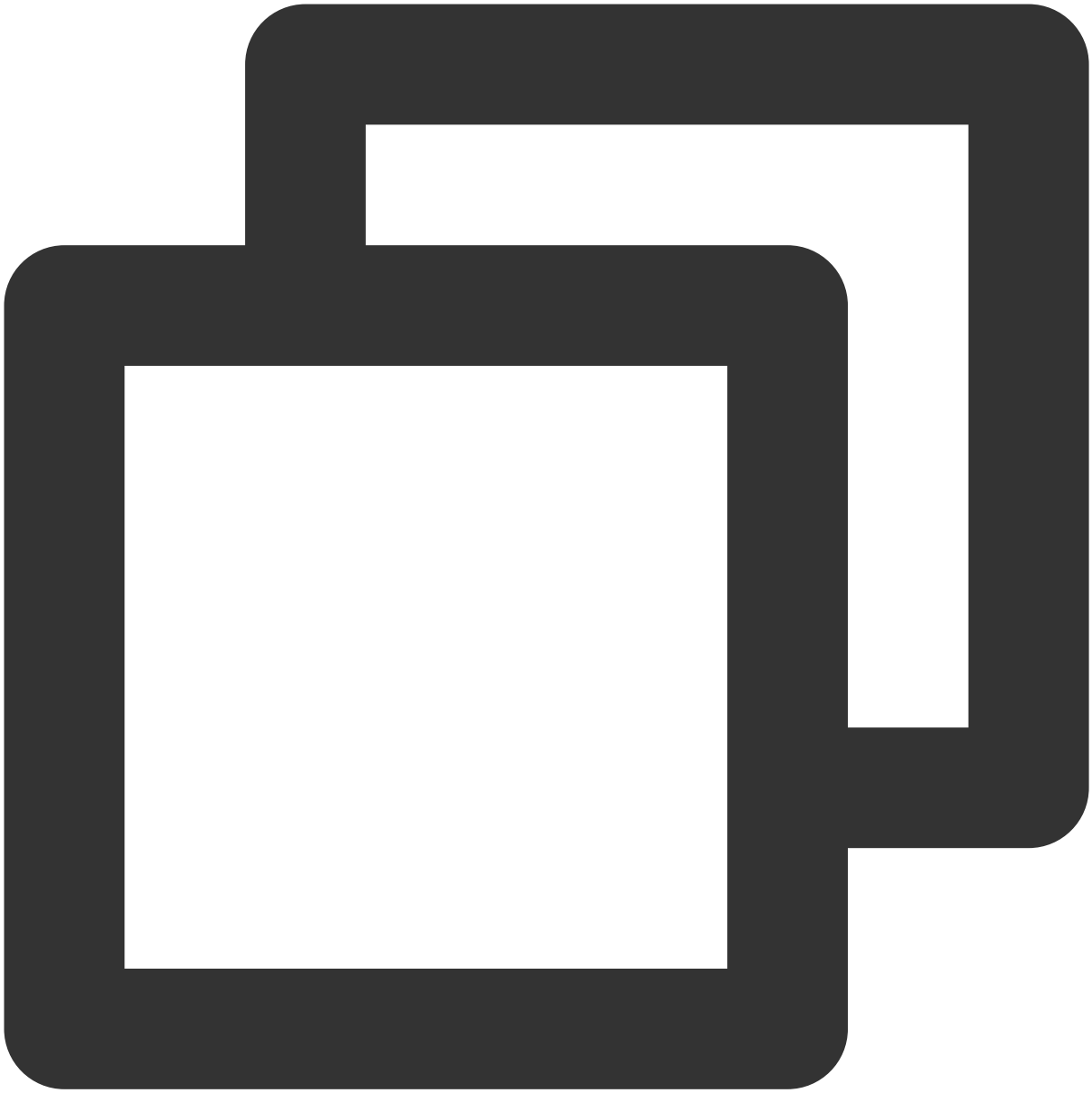
```
├── HuiYanEKYCVerification.framework
├── tnn.framework
├── tnnliveness.framework
├── YTCommonLiveness.framework
├── YTFaceAlignmentTinyLiveness.framework
├── YTFaceDetectorLiveness.framework
├── YTFaceLiveReflect.framework
├── YTFaceTrackerLiveness.framework
├── YTPoseDetector.framework
├── YtSDKKitActionLiveness.framework
├── YtSDKKitFramework.framework
├── YtSDKKitOcrVideoIdent.framework
├── YtSDKKitReflectLiveness.framework
├── YtSDKKitSilentLiveness.framework
├── HKOCRSDK.framework
├── tiny_opencv2.framework
├── HuiYanOverseasSDK.framework
├── IdVerification.framework
├── OcrSDKKit.framework
├── TXYCommonDevice.framework
├── TXYCommonNetworking.framework
├── TXYCommonUtils.framework
├── YTCv.framework
├── YTImageRefiner.framework
├── YtSDKKitFrameworkTool.framework
└── YTSm.framework
```

Link Binary With Libraries导入系统Framework



```
├── AVFoundation.framework
├── libc++.tbd
├── Accelerate.framework
└── CoreML.framework
```

Copy Bundle Resources导入资源文件



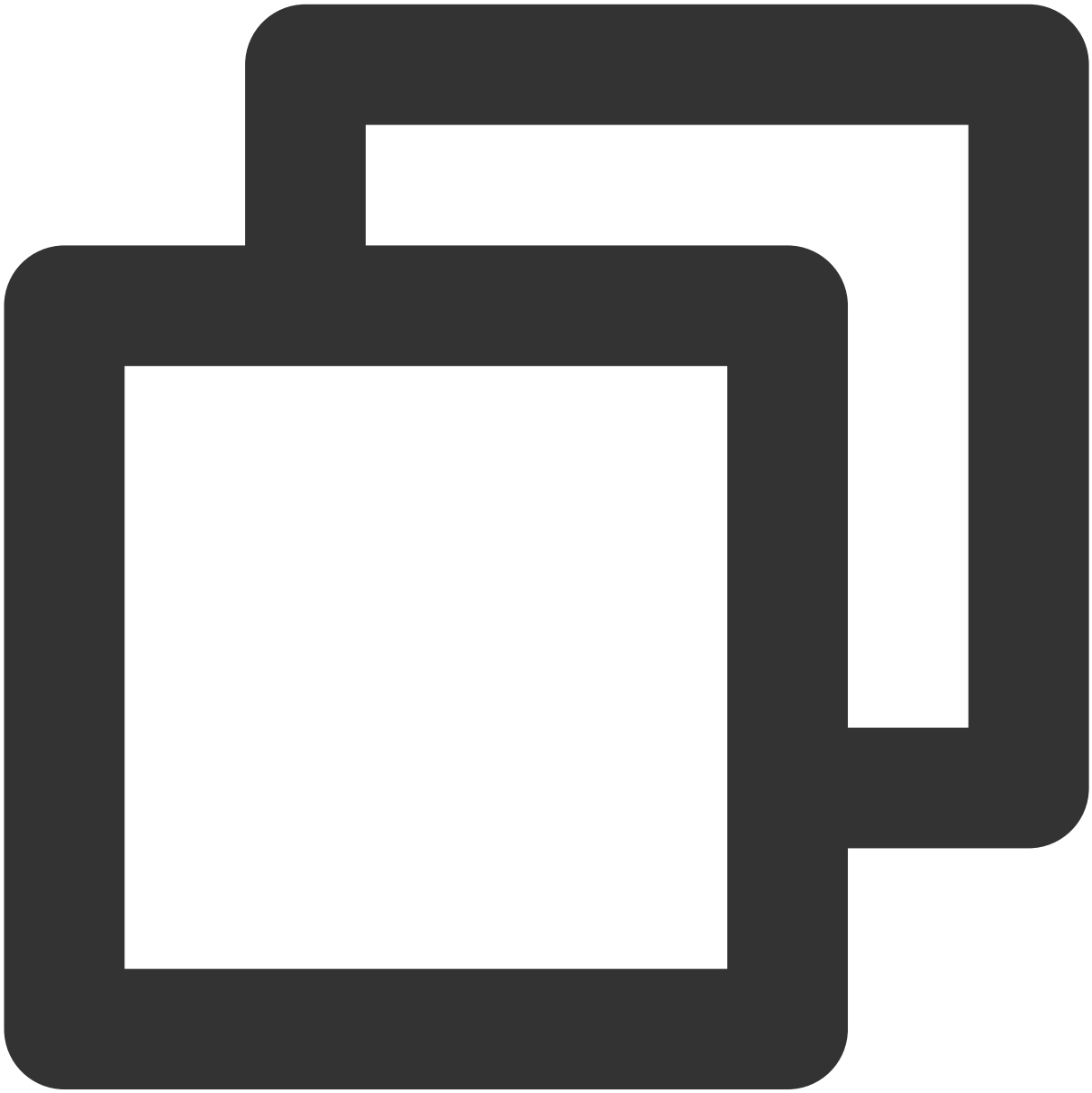
```
├── face-tracker-v003.bundle
├── huiyan_verification.bundle
├── HuiYanSDKUI.bundle
├── idverificationres.bundle
├── ocr-v001.bundle
├── OcrSDK.bundle
└── ytsdkviidres.bundle
```

Build Phases 设置

1. Other Linker Flags 新增 -ObjC。
2. 接入ViewController.m 设置后缀为.mm(swift 工程添加系统库libc++.tbd)。

权限设置

SDK需要手机网络及 摄像头使用权限，请添加对应的权限声明。在主项目info.plist 配置中添加下面key-value值。

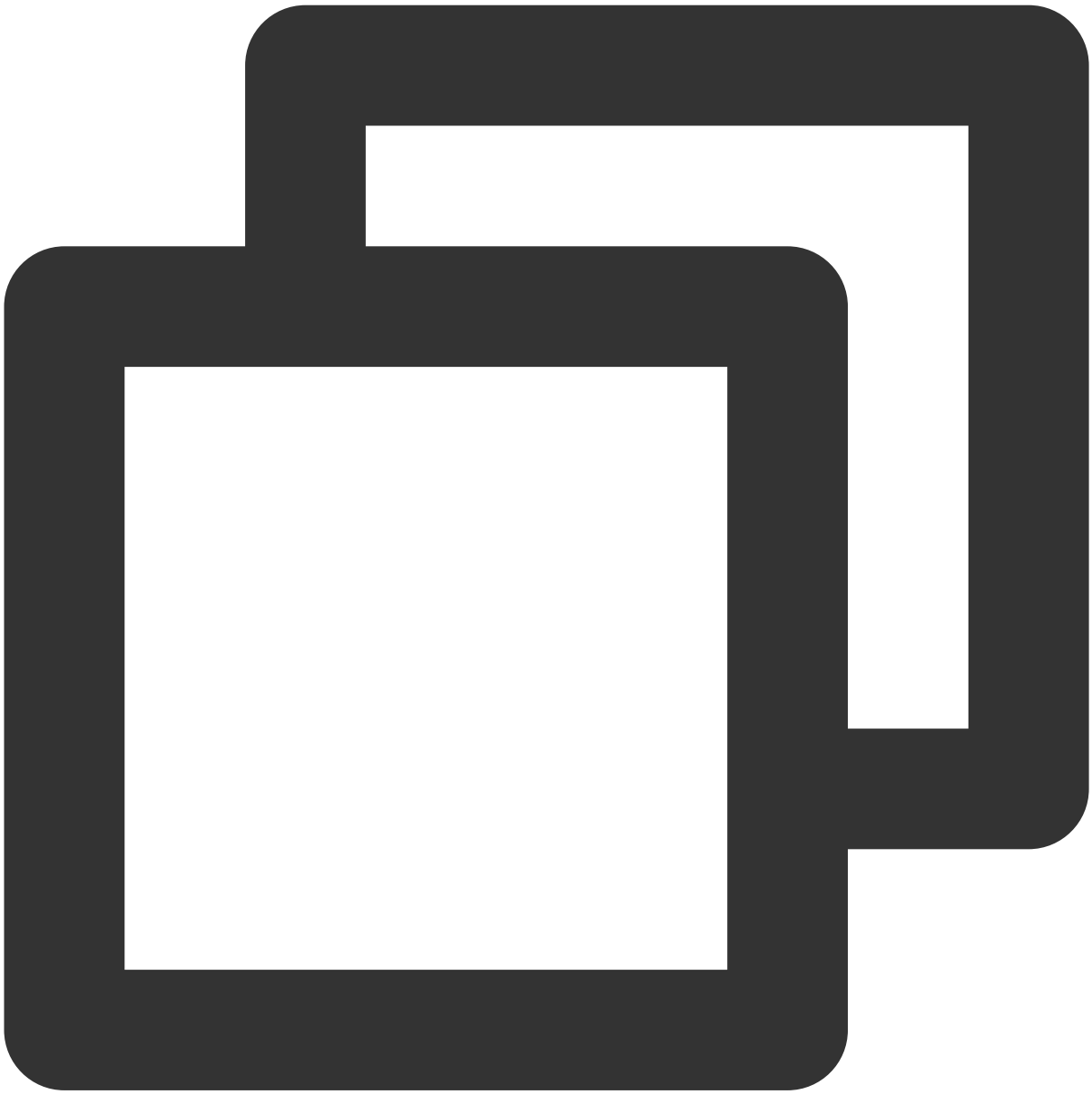


```
<key>Privacy - Camera Usage Description</key>
        <string>需要访问您的相机权限</string>
<key>Privacy - Photo Library Usage Description</key>
        <string>需要访问您的相册权限</string>
```

启动身份认证检测

1. 初始化

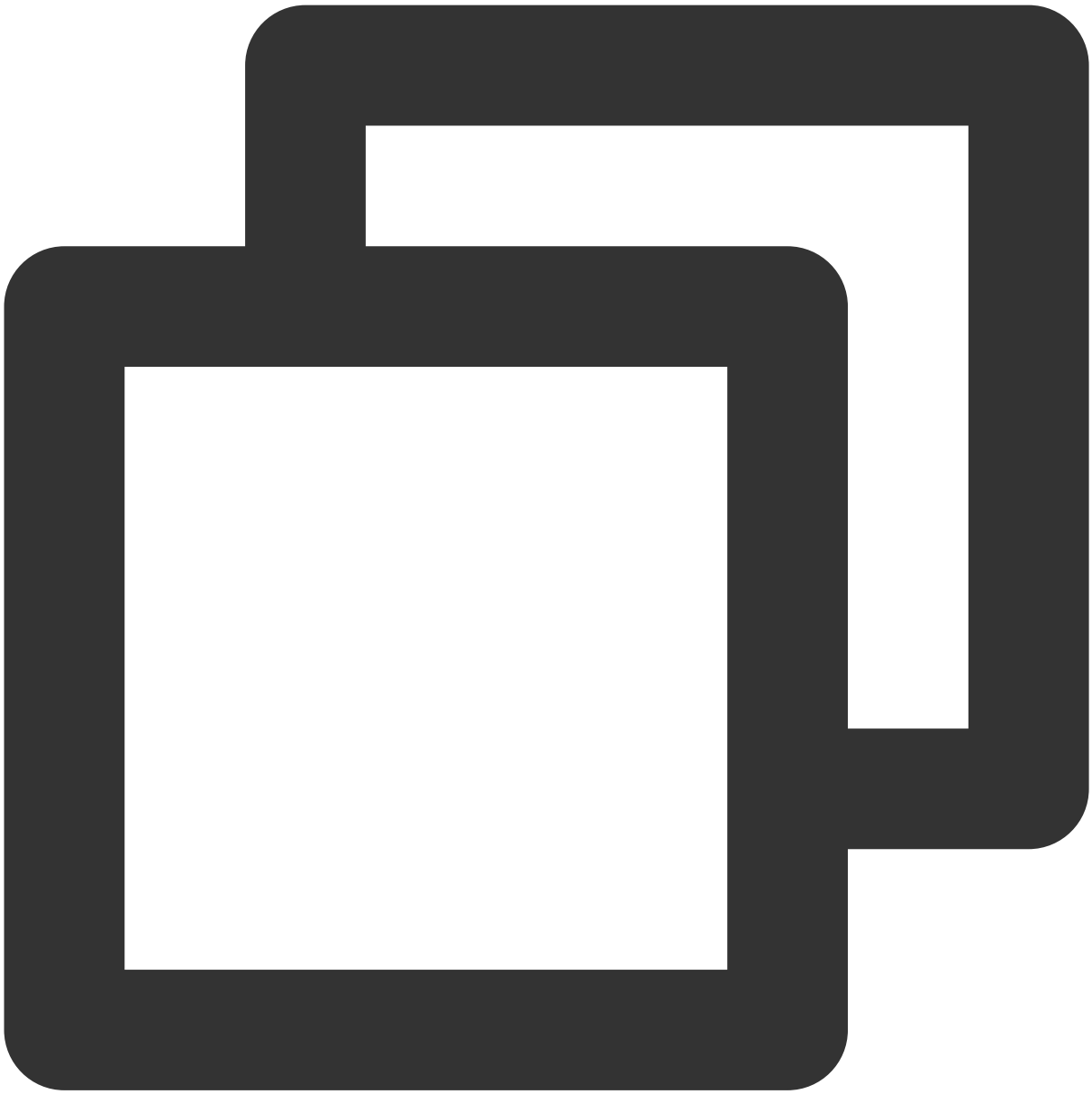
在您APP初始化的时候调用，主要是进行一些SDK的初始化操作



```
#import <HuiYanEKYCVerification/VerificationKit.h>
- (void)viewDidLoad {
    [[VerificationKit sharedInstance] initWithViewController:self];
}
```

2. 启动身份认证检测流程

当您需要启动身份认证eKYC检测流程的时候，只需要调用startVerifiWithConfig方法，配置ekycToken，和一些自定义配置。



```
VerificationConfig *config = [[VerificationConfig alloc] init];
config.licPath = [[NSBundle mainBundle] pathForResource:@"" ofType:nil];
config.languageType = HY_EKYC_EN;
config.verAutoTimeOut = 30000;//鉴伪超时时间设置
config.hyFaceTimeOut = 15000;//人脸单动作超时设置
config.ekycToken = @"";
[[VerificationKit sharedInstance] startVerifiWithConfig:config withSuccCallback:^(i
        NSLog(@"ErrCode:%d msg:%@",errorCode,resultInfo);
```

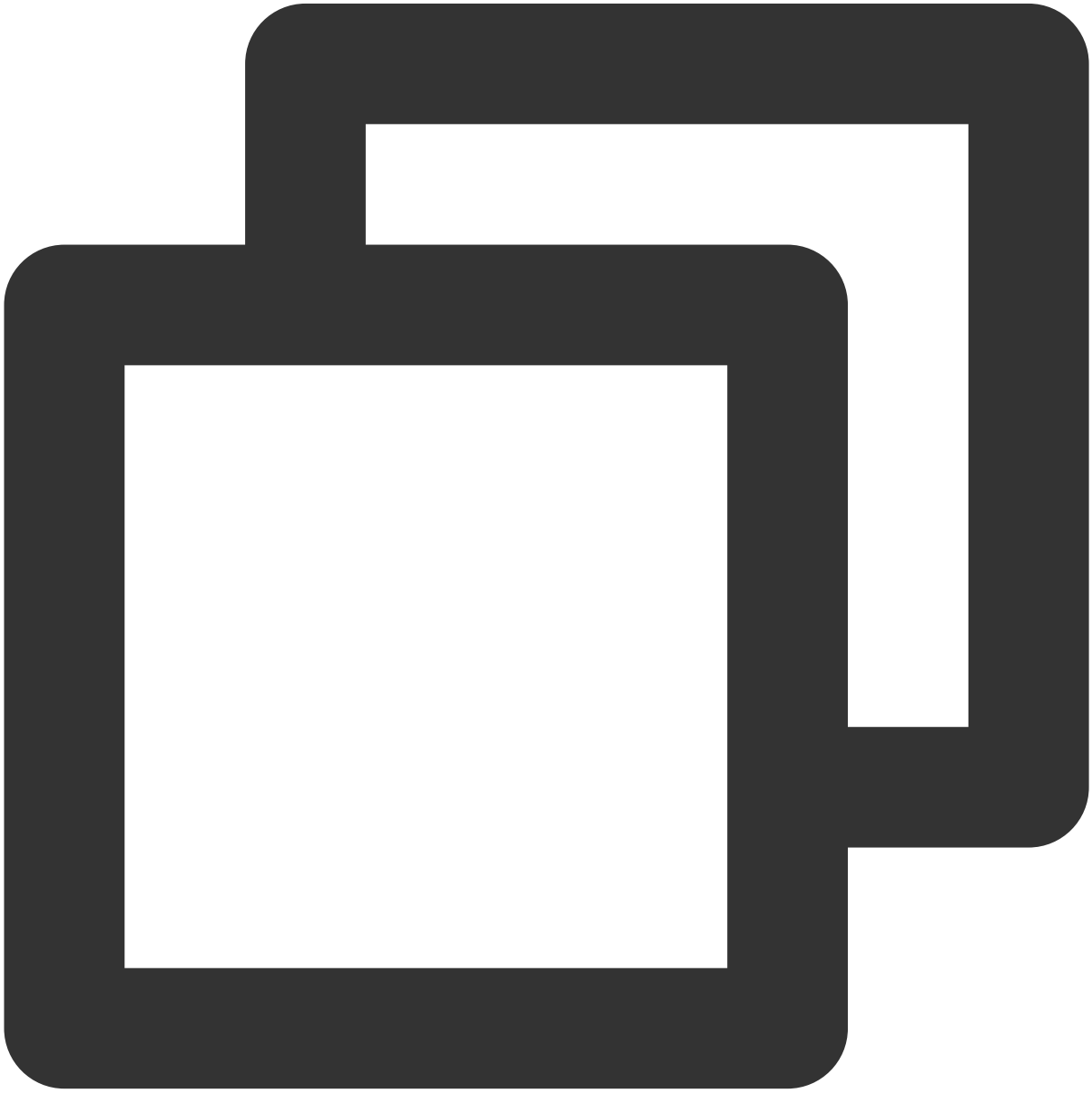
```
} withFialCallback:^(int errorCode, NSString * _Nonnull errorMsg, id  _Nullable res
        NSLog(@"ErrCode:%d msg:%@ extra:%@",errorCode,errorMsg,reserved);
}];
```

ekycToken为从服务器兑换的本次身份认证流程的唯一凭证。

注意：**"eKYC_license.lic"**文件是需要联系商务或者客服人员进行license申请。将申请完成后的license文件放到Copy Bundle Resources下。

SDK 资源释放

在您调用完 SDK 不在使用时，可以调用 SDK 资源释放接口：



```
- (void)dealloc {
    [VerificationKit clearInstance];
}
```

说明：iOS 完整的代码示例详见iOS demo。

SDK 接口 API 概述

Identity Verification Android 接口概述文档

最近更新时间：2023-12-28 16:06:07

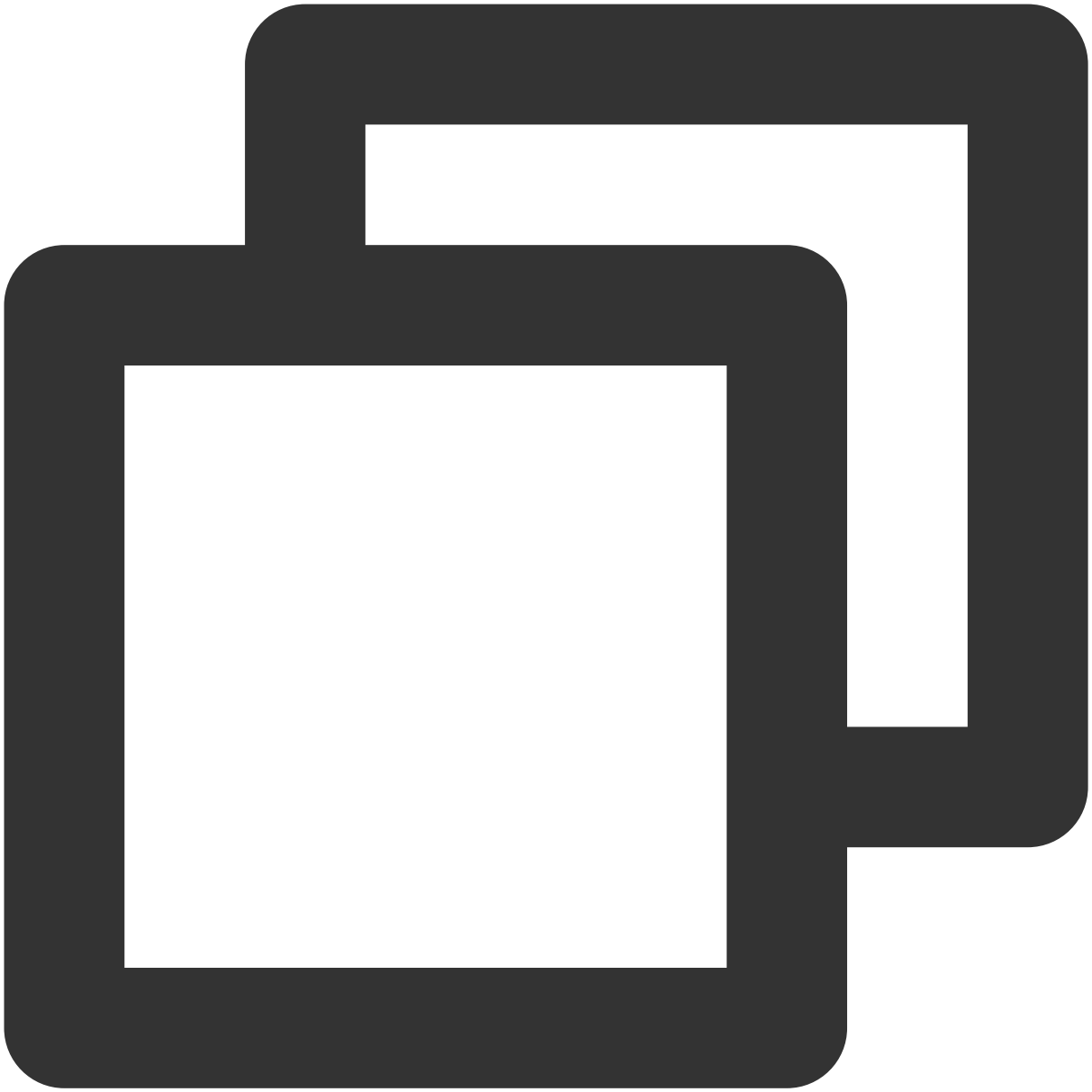
Android端Identity Verification(App SDK)主要涉及的类主要包含EkycHySdk、EkycHyConfig、EkycHyCallBack这几个主要类型下面对其支持的API做出详细说明。

EkycHySdk

EkycHySdk为Identity Verification(App SDK)的对外接口类，主要逻辑也都是调用此类完成。

API	功能描述
init()	初始化接口
release()	资源释放接口
startEkycCheck()	启动身份认证Identity Verification检测流程

init()



```
public static void init(Context context)
```

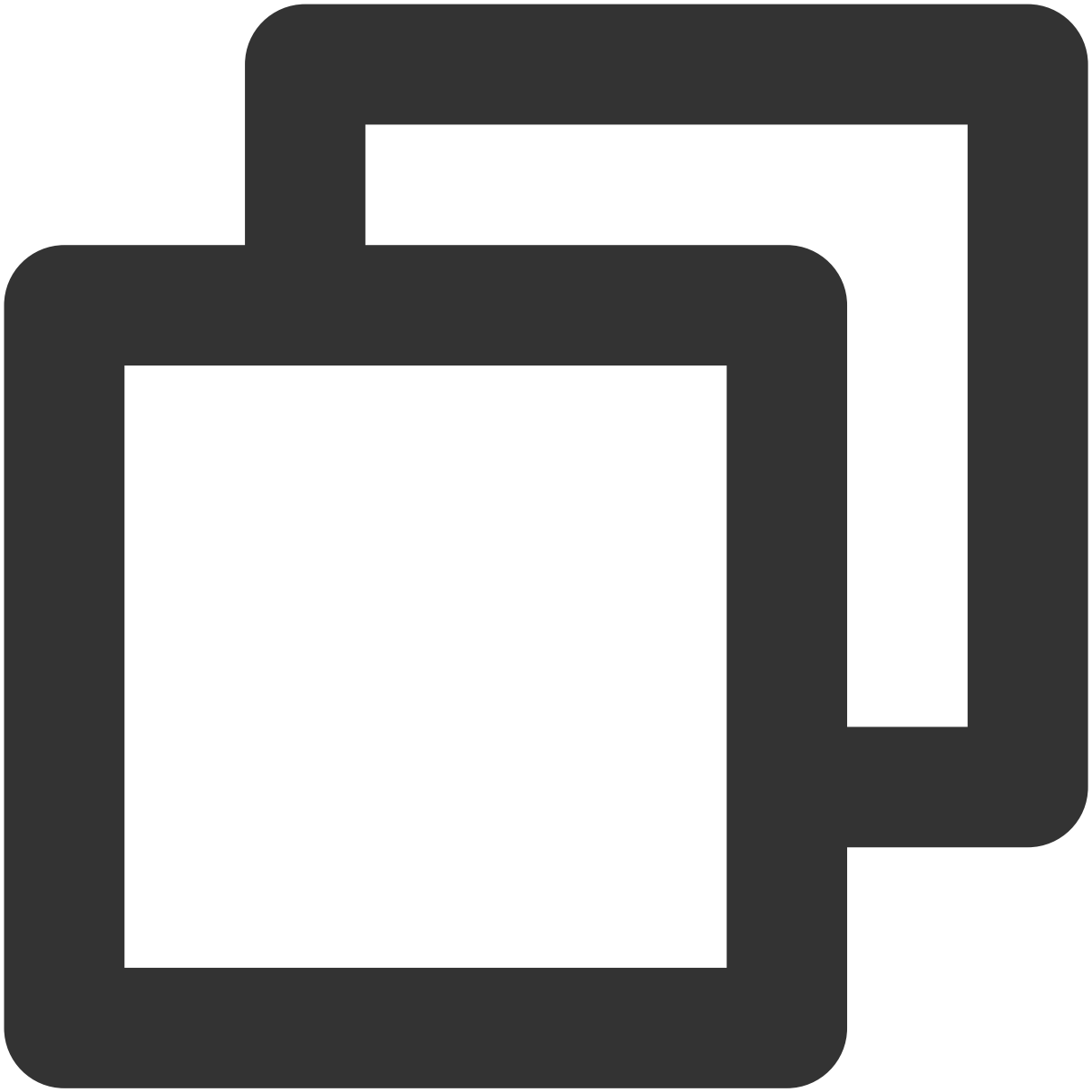
功能介绍：

Identity Verification(App SDK)的初始化接口。

传入参数:

参数类型	参数名称	参数含义
Context	context	App的上下文信息

release()

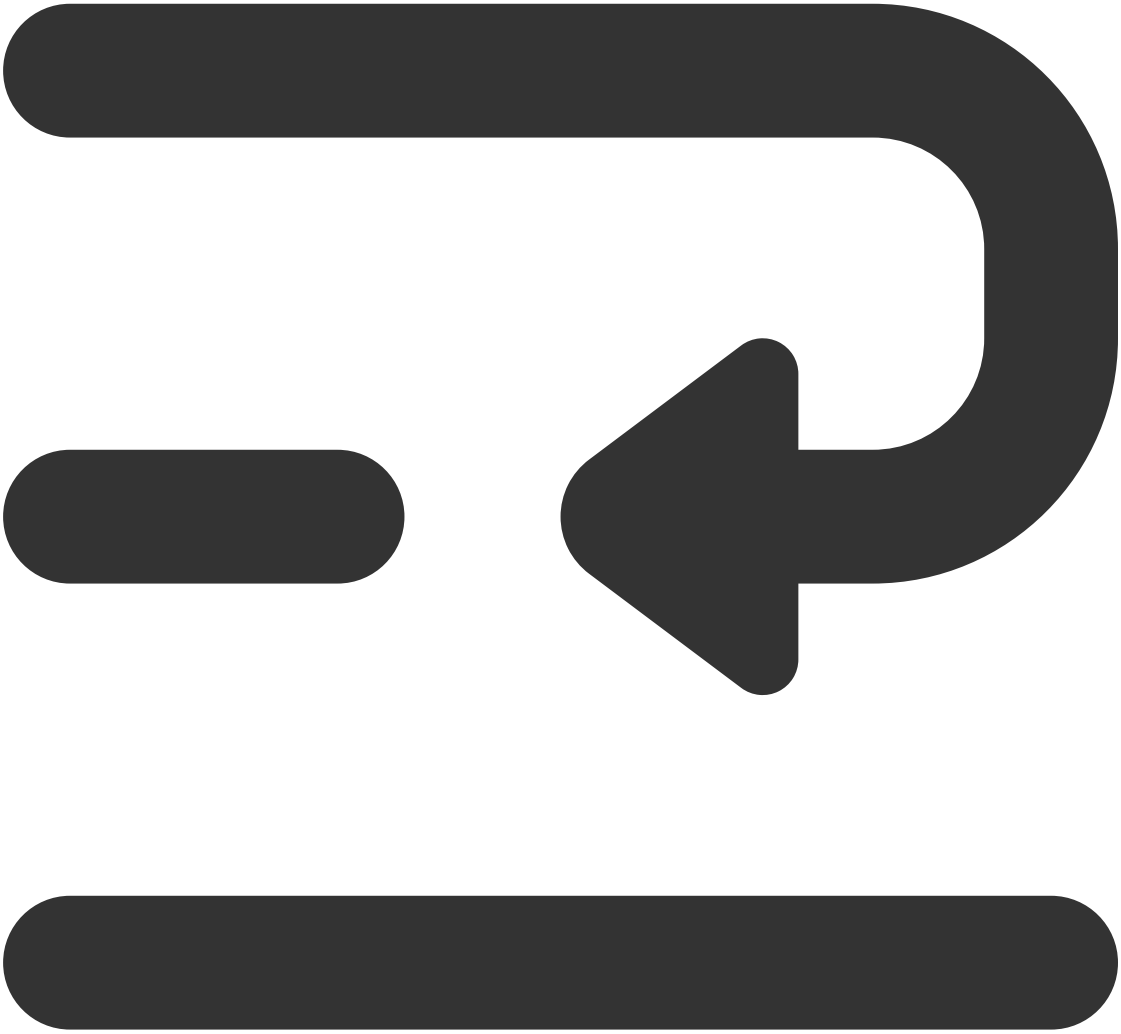


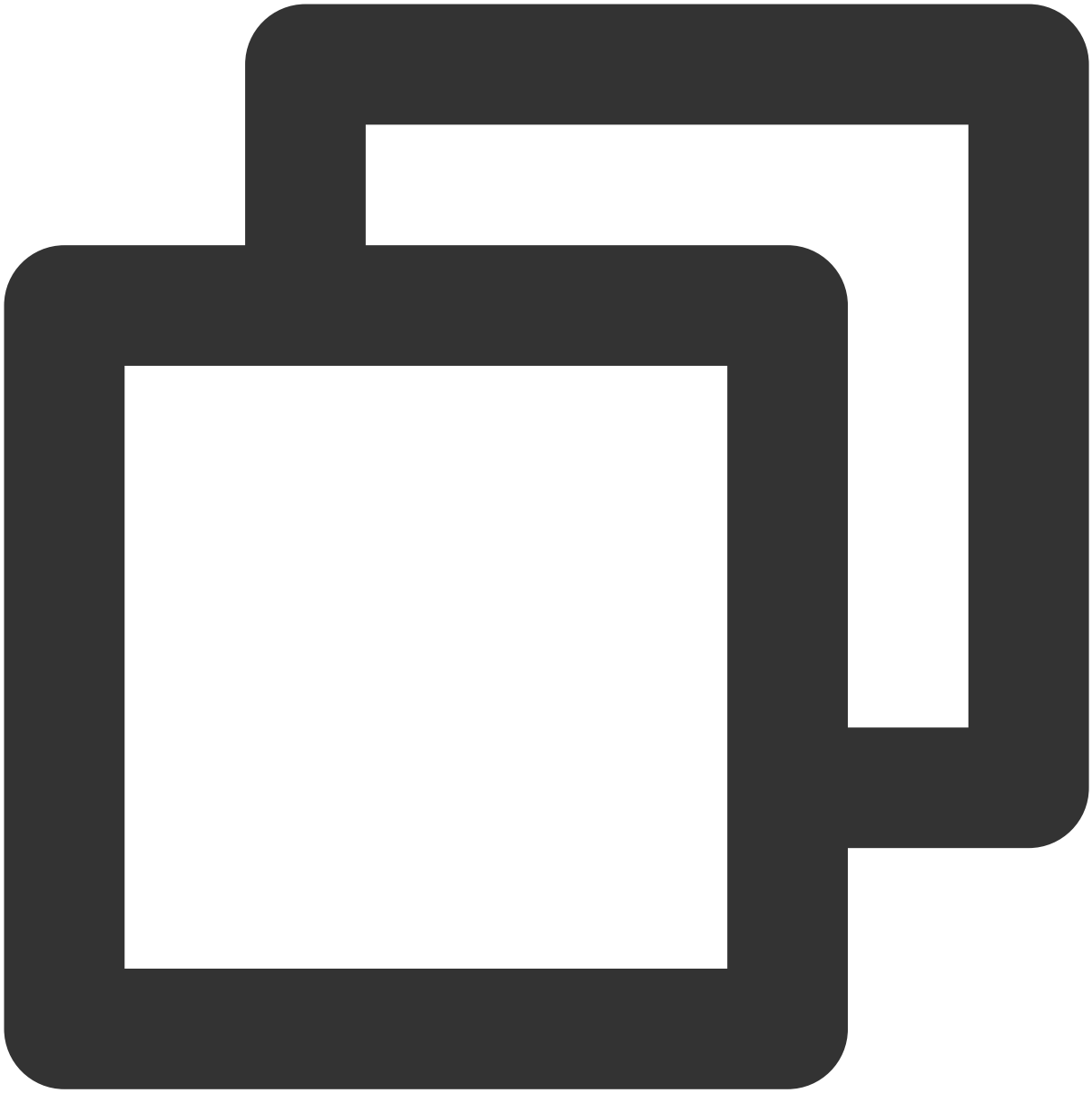
```
public static void release()
```

功能介绍：

Identity Verification(App SDK)资源释放的接口。

startEkycCheck()





```
public static void startEkycCheck(final String ekycToken, EkycHyConfig ekycHyConfig
            EkycHyCallBack ekycHyCallBack)
```

功能介绍：

启动身份认证Identity Verification检测流程的流程函数。

传入参数：

参数类型	参数名称	参数含义
String	ekycToken	从服务器端获取的Token值，作为此次流程唯一业务凭证

EkycHyConfig	ekycHyConfig	Identity Verification本次流程启动的配置信息
EkycHyCallBack	ekycHyCallBack	用来接收认证结果回调的接口

EkycHyConfig

EkycHyConfig是在启动Identity Verification(App SDK)时的配置实体类，主要包含了以下属性。

类型	名称	含义	默认值
OcrUiConfig	ocrUiConfig	ocr界面的一些自定义配置	null
String	licenseName	客户申请的用户授权Licens文件名	空
int	ocrAutoTimeout	OCR_DETECT_AUTO_MANUAL模式下，自动捕获的超时时间（毫秒单位，最少设置5秒，内部上限30秒）	20000毫秒（20秒）
LanguageStyle	languageStyle	本次流程的语言风格	LanguageStyle.AUTO
OcrModeType	ocrModeType	Ocr识别模式, 手动拍摄模式、自动识别模式、自动+手动识别模式	OcrModeType.OCR_DETECT_MA
OcrRegionType	ocrType	卡片类型	null

OcrRegionType

检测证件类型

枚举名	说明
HK	香港证件
ML	马来西亚证件
PhilippinesDrivingLicense	菲律宾-驾驶执照
PhilippinesVoteID	菲律宾-VoteID
PhilippinesTinID	菲律宾-TinID
PhilippinesSSSID	菲律宾-SSSID
PhilippinesUMID	菲律宾-UMID
IndonesiaIDCard	印尼证件

MLIDPassport	护照

LanguageStyle

Identity Verification默认界面的多语言配置信息。

LanguageStyle类型	含义
LanguageStyle.AUTO	跟随系统语言版本
LanguageStyle.ENGLISH	英语
LanguageStyle.SIMPLIFIED_CHINESE	简体中文
LanguageStyle.TRADITIONAL_CHINESE	繁体中文

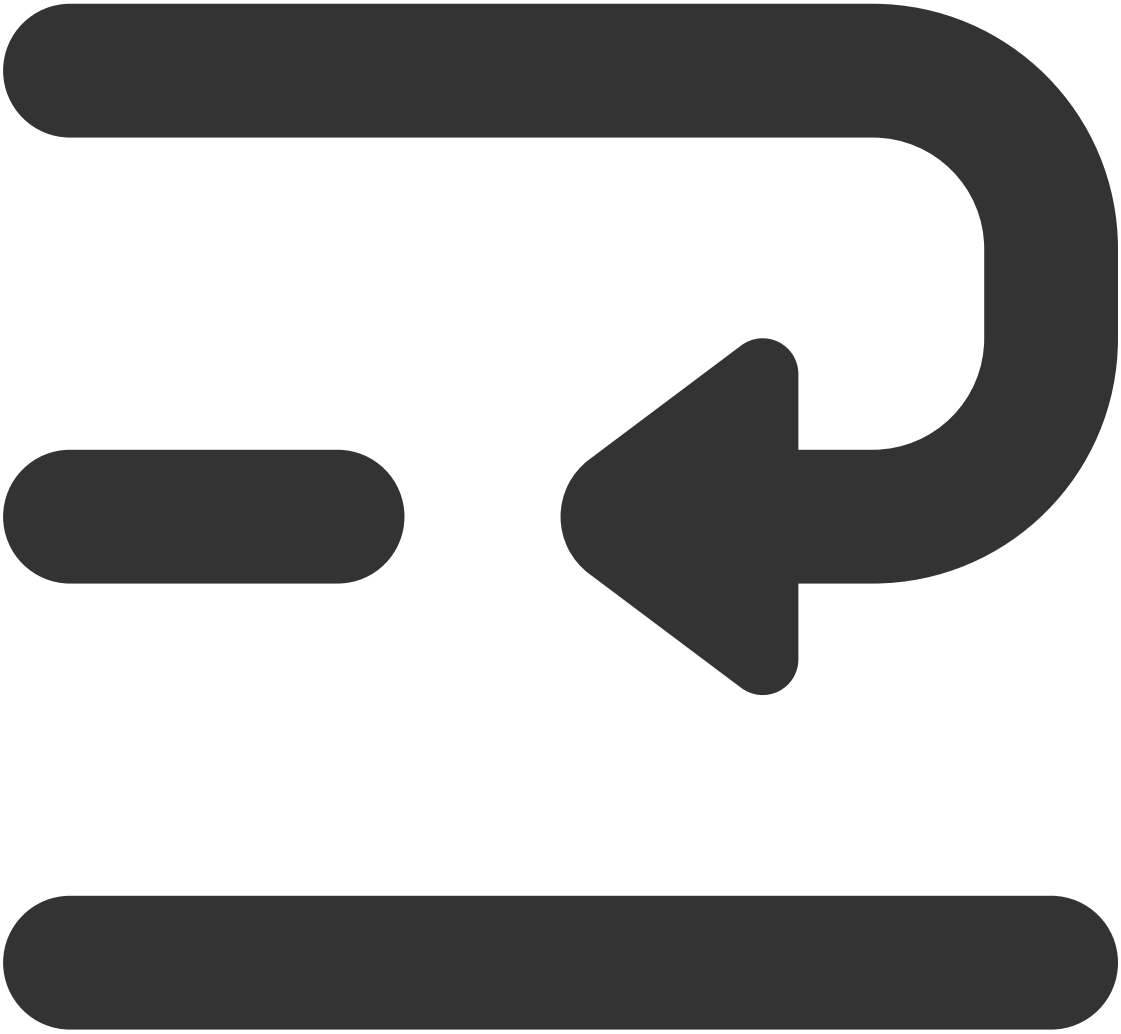
OcrModeType

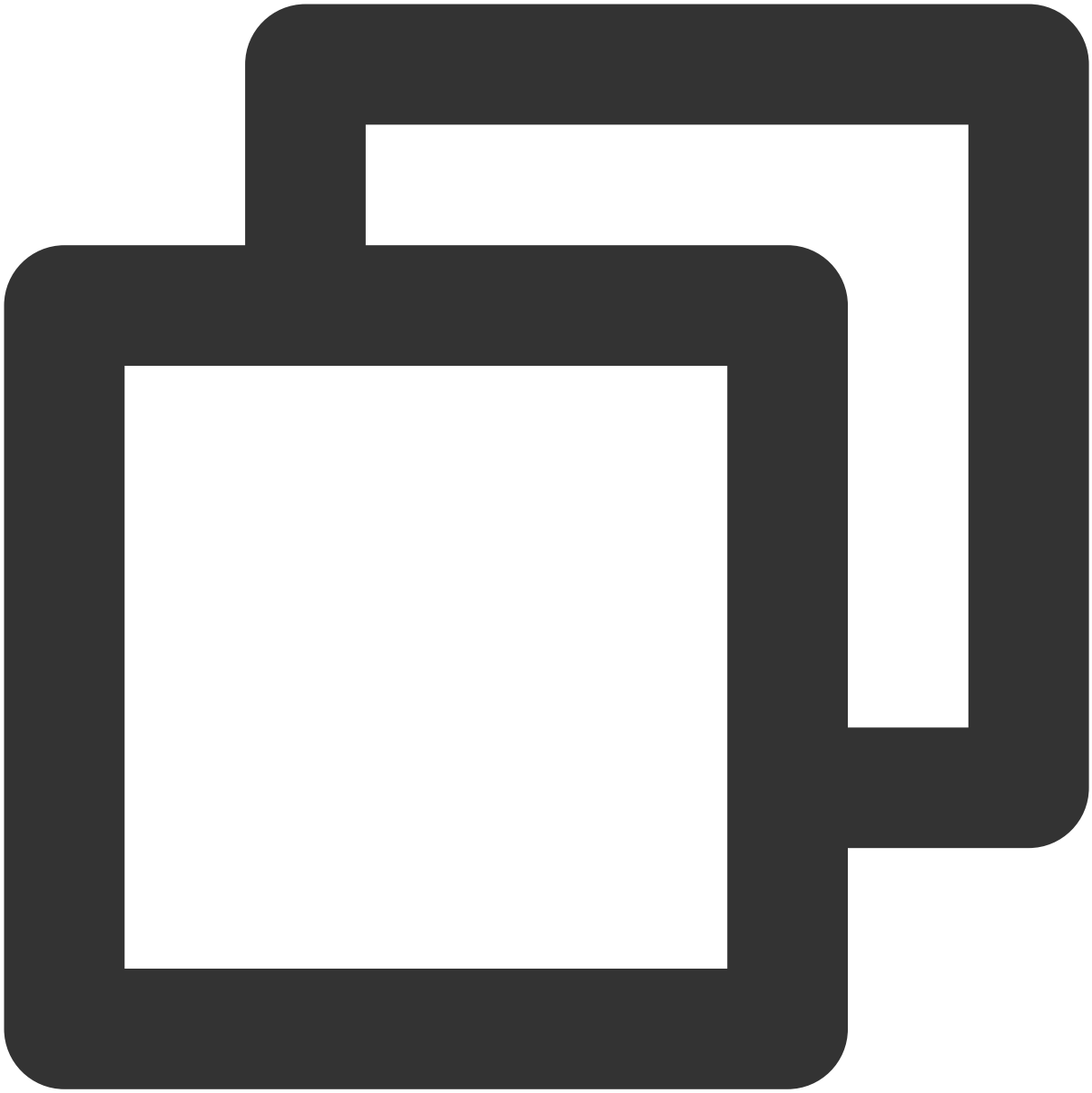
Ocr识别模式

OcrModeType类型	含义
OCR_DETECT_MANUAL	纯手动拍摄模式
OCR_DETECT_AUTO	纯自动拍摄模式
OCR_DETECT_AUTO_MANUAL	自动+手动识别模式

EkycHyCallBack

用于接收Identity Verification流程结果的监听类。





```
/**
 * Identity verification的结果回调类
 */
public interface EkycHyCallBack {

    /**
     * 识别成功的结果信息
     *
     * @param result 结果信息
     */
    void onSuccess(EkycHyResult result);
```

```
    /**
     * Identity verification流程失败的内容
     *
     * @param errorCode 错误码
     * @param errorMsg 错误信息
     * @param ekycToken 当次流程的token
     */
    void onFail(int errorCode, String errorMsg, String ekycToken);
}
```

其中 **EkycHyResult** 是成功返回的结果对象。

EkycHyResult

EkycHyResult是Identity Verification(App SDK)流程成功后的结果对象。

类型	名称	含义	默认值
String	ekycToken	当次Identity Verification流程的token，此token可以在服务器拉取身份认证过程关键数据	空

错误码与含义

错误码	对应含义
12000	用户主动取消
12001	网络请求失败
12002	OCR识别异常导致的错误
12003	本地人脸检测失败引起的异常
12004	失效的token
12005	本地卡证识别失败
12006	Identity Verification(App SDK)初始化流程失败
12007	启动参数校验失败
12008	人脸核身结果返回失败
12009	人脸核身过程本地失败

Identity Verification iOS 接口概述文档

最近更新时间：2023-08-10 15:03:03

API 概述

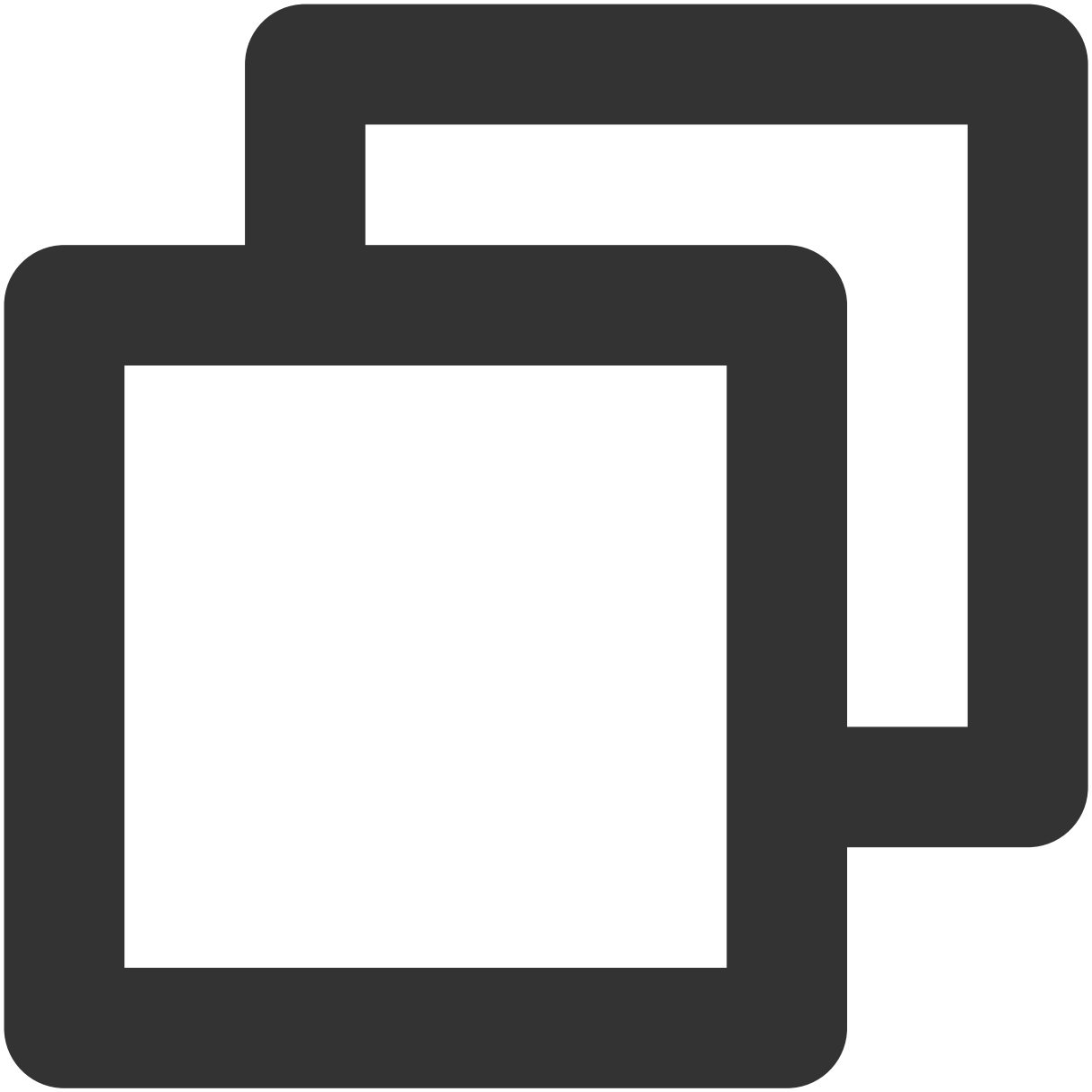
iOS端Identity Verification(App SDK)主要涉及的类主要包含VerificationKit、VerificationConfig、VerifiCommDef这几个主要类型下面对其支持的API做出详细说明。

1.VerificationKit

VerificationKit为Identity Verification(App SDK)的对外接口类，主要逻辑也都是调用此类完成。

API	功能描述
initWithViewController	初始化接口
clearInstance	资源释放接口
startVerifiWithConfig	启动Identity Verification身份证核验的流程

1.1. initWithViewController



```
// 初始化方法-
- (void)initWithViewController:(UIViewController *)viewController;
```

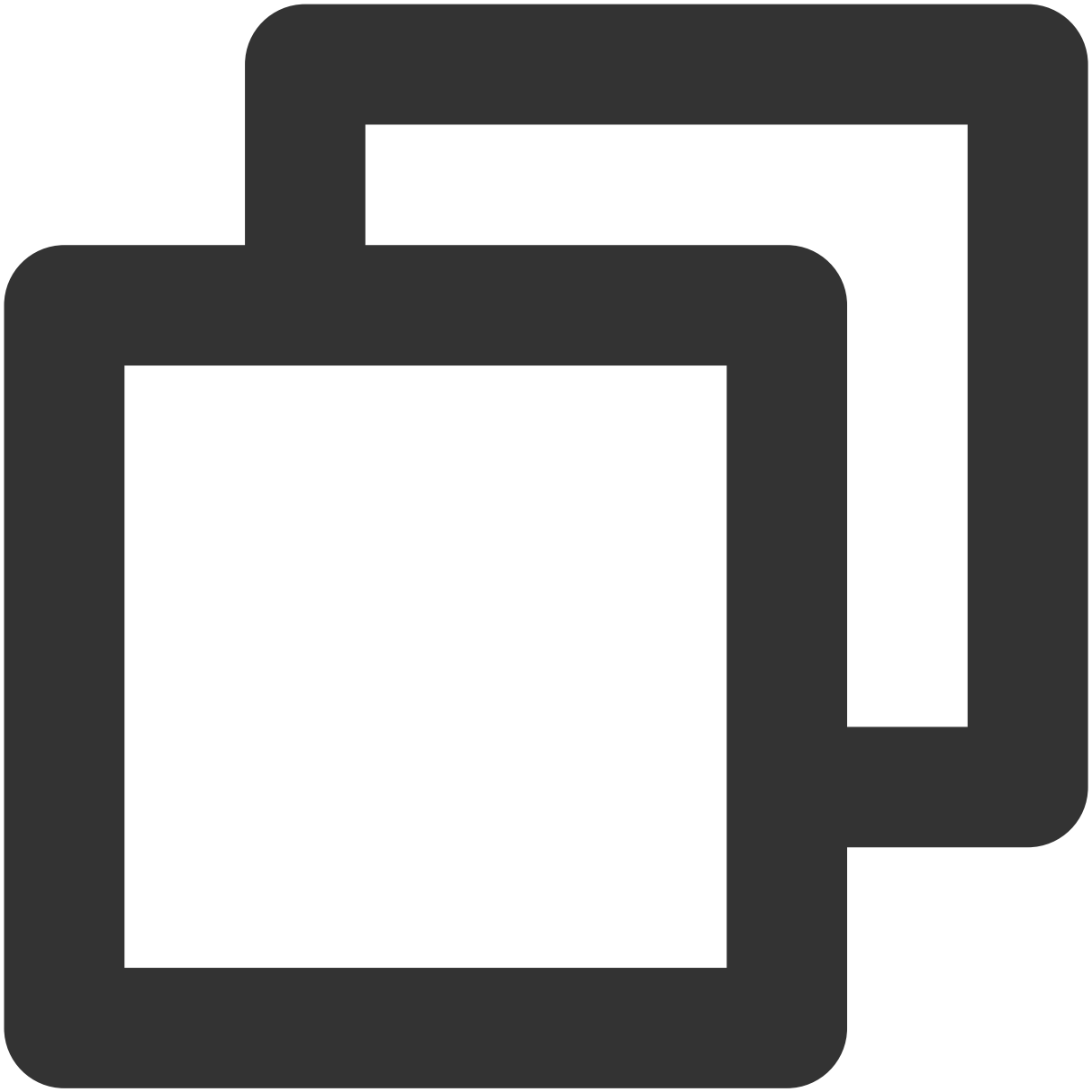
功能介绍：

IIdentity Verification(App SDK)的初始化接口。

传入参数:

参数类型	参数名称	参数含义
UIViewController	viewController	当前调用SDK页面VC对象

1.2 clearInstance

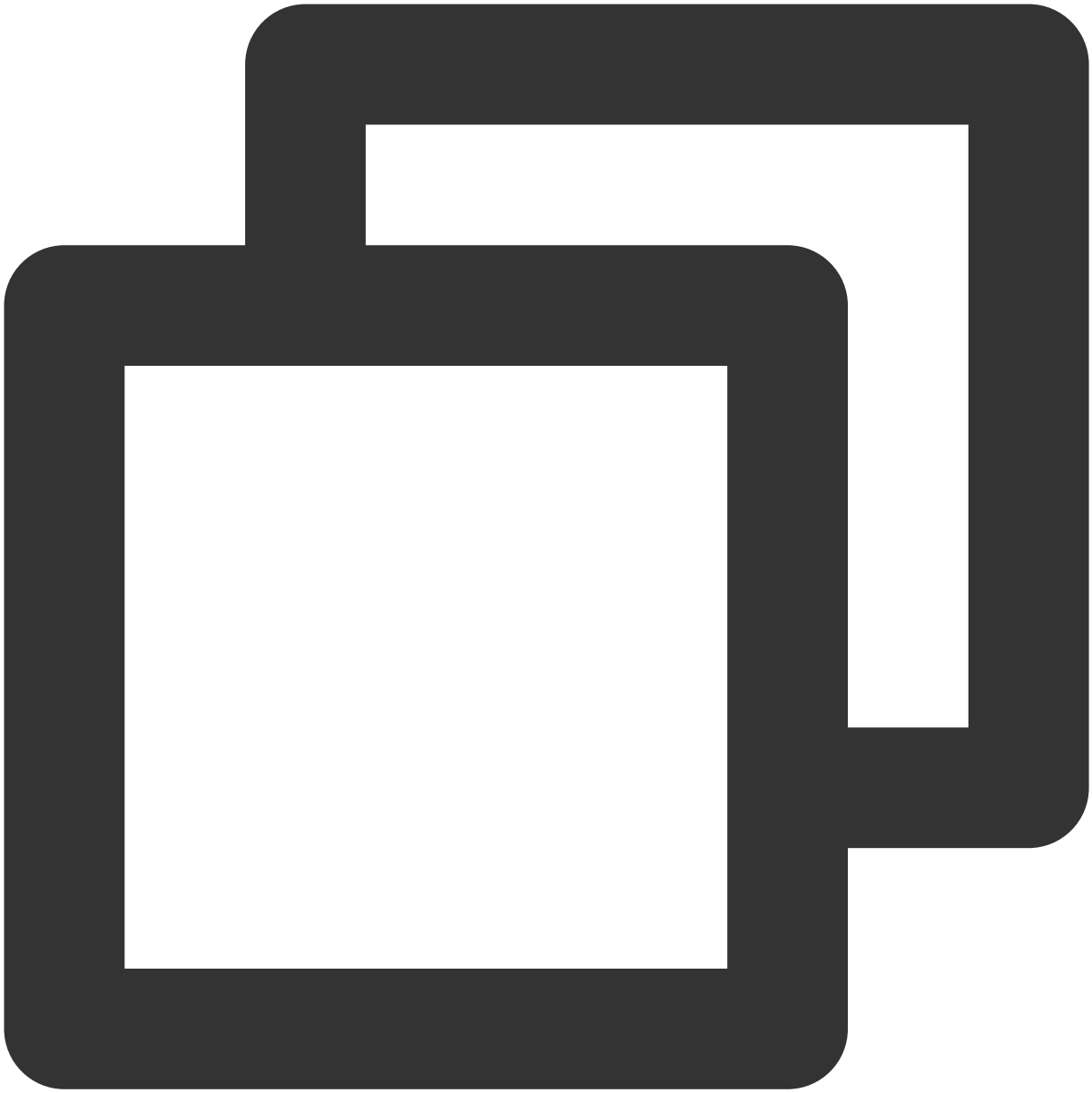


```
/// 清理SDK资源
+ (void)clearInstance;
```

功能介绍：

SDK资源释放的接口。

1.3 startVerifiWithConfig:



```
/// 开启验证
- (void)startVerifiWithConfig:(VerificationConfig *)verifiConfig
            withSuccCallback:(TXYVerifiKitProcessSucceedBlock)succCallback
             withFialCallback:(TXYVerifiKitProcessFailedBlock)failCallback;
```

功能介绍：

启动身份证核验的流程方法。

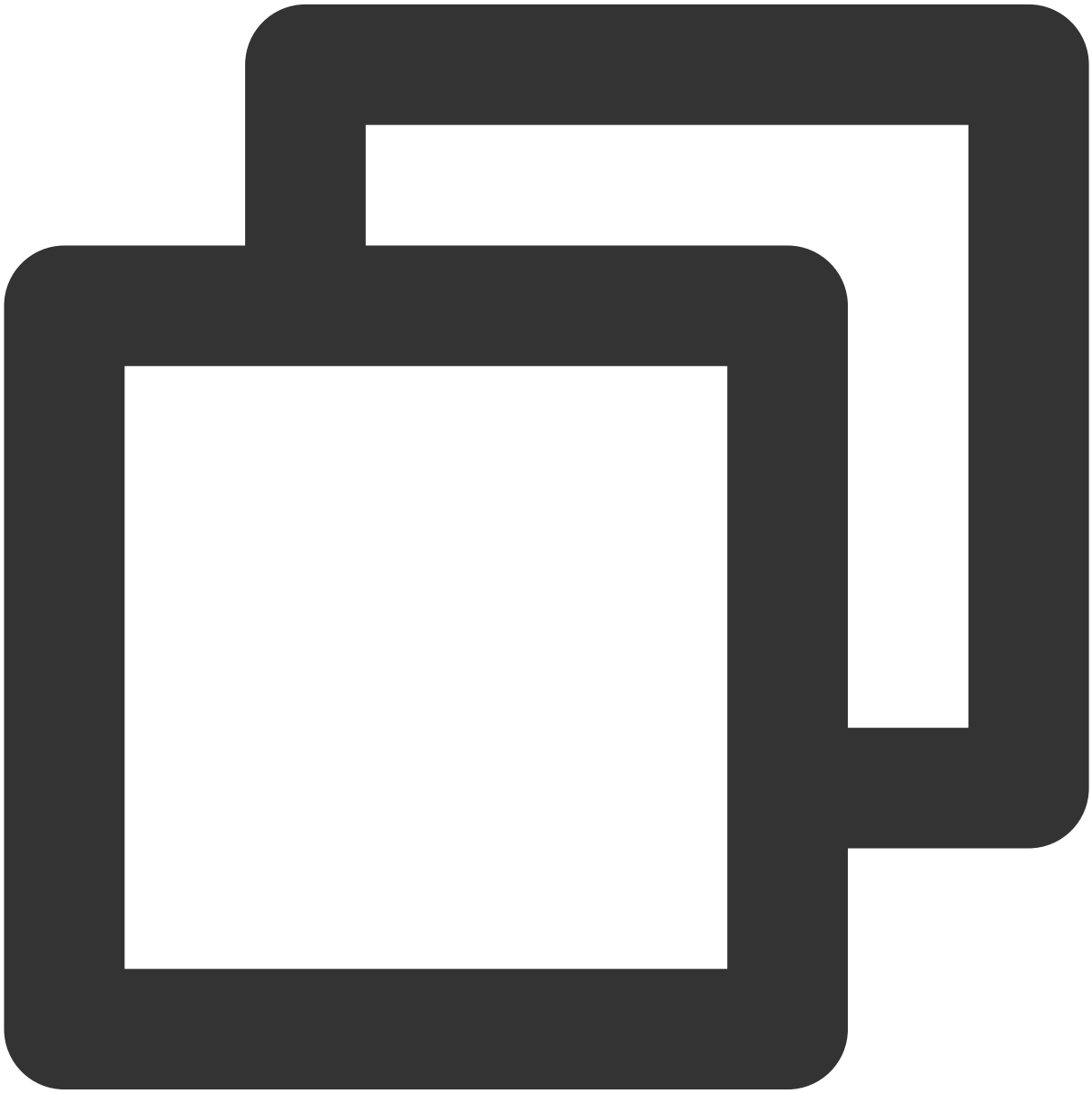
传入参数：

参数类型	参数名称	参数含义

VerificationConfig	verifiConfig	本次流程启动的配置信息
TXYVerifiKitProcessSucceedBlock	succCallback	SDK检测成功回调
TXYVerifiKitProcessFailedBlock	failCallback	SDK检测失败回调

1.4 TXYVerifiKitProcessSucceedBlock

SDK检测成功回调。

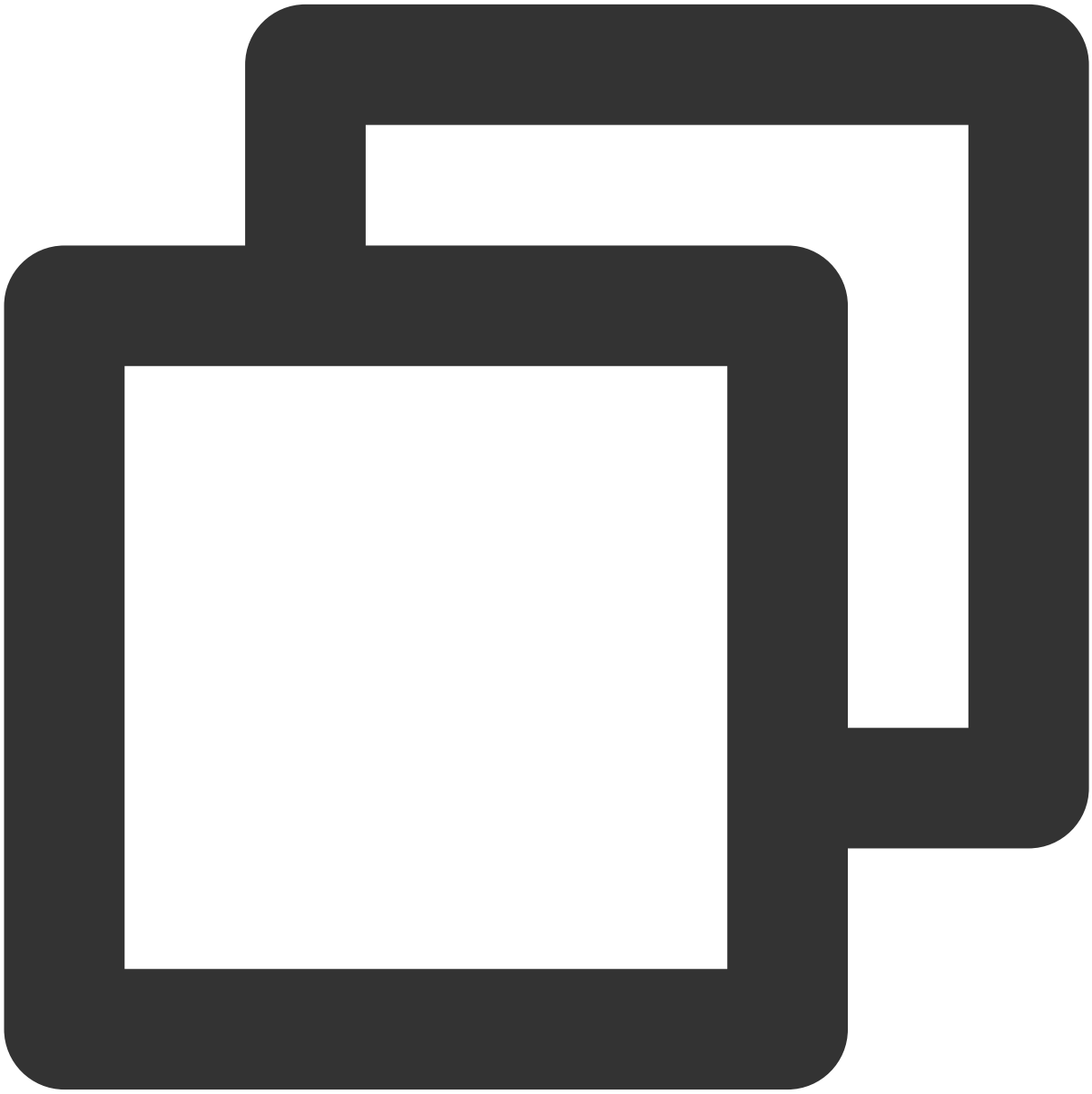


```
/// SDKKIt处理成功回调接口
/// @param errorCode 错误码
```

```
/// @param resultInfo 回调返回的信息
/// @param reserved 预留位
typedef void (^TXYVerifiKitProcessSucceedBlock)(int errorCode,id _Nonnull resultInf
```

1.5 TXYVerifiKitProcessFailedBlock

SDK检测失败回调。



```
/// SDKKIt处理失败回调接口
/// @param errorCode 错误码
/// @param errorMsg 错误信息
```

```
/// @param reserved 预留位
typedef void (^TXYVerifiKitProcessFailedBlock)(int errorCode, NSString *_Nonnull er
```

2.VerificationConfig

VerificationConfig是在启动SDK时的配置实体类，主要包含了以下属性。

类型	名称	含义	默认值
NSString	ekycToken	从服务器端获取的Token值，作为此次核身唯一业务凭证	空
NSString	licPath	客户申请的用户授权Licens文件路径	空
long	hyFaceTimeOut	人脸核身单一动作的超时时间	10000毫秒（10秒）
BOOL	isHiddenAlbum	是否隐藏OCR相册按钮	NO
BOOL	isHiddenFlash	是否隐藏OCR手电筒按钮	NO
HYEkycLanguageType	languageType	本次流程的语言风格	DEFAULT (0)
OCRRegionType	ocrRegionType	卡证检测类型	0
NSString	userUIBundleName	用户自定义UI资源包名	Nil
NSString	userLanguageBundleName	用户自定义多语言资源包名	nil
NSString	userLanguageFileName	指定自定义多语言使用文件名	nil

3.VerifiCommDef

3.1 HYEkycLanguageType

SDK默认界面的多语言配置信息。

类型	含义
HY_EKYC_DEFAULT = 0	跟随系统语言版本
HY_EKYC_ZH_HANS	简体中文
HY_EKYC_ZH_HANT	繁体中文
HY_EKYC_EN	英语
HY_EKYC_CUSTOMIZE_LANGUAGE	自定义语言，使用设置的自定义语言bundle

	（userLanguageBundleName）

3.2 OCRRegionType

卡证检测类型。

枚举名	说明
OCR_TYPE_DEFULT = 0	默认空
OCR_TYPE_HK	香港证件
OCR_TYPE_ML	马来西亚证件
OCR_TYPE_PV_ID	菲律宾-驾驶执照证件
OCR_TYPE_PDL	菲律宾-VoteID证件
OCR_TYPE_INDONESIA	印尼证件
OCR_TYPE_SINGAPORE	新加坡证件
OCR_TYPE_PH_TINID	菲律宾-TINID
OCR_TYPE_PH_SSSID	菲律宾-SSSID
OCR_TYPE_PH_UMID	菲律宾-UMID
OCR_TYPE_MLID_PASSPORT	护照证件

4.错误码与含义

错误码	错误码值	错误码含义
HY_SUCCESS	0	成功
HY_VERIFI_FAIL	-1	检测失败
HY_VERIFI_OCR_FAIL	-2	卡证识别失败
HY_SDK_INNER_ERR	-4	慧眼内部错误
HY_INITIALIZATION_PARAMETER_EXCEPTION	310	初始化参数异常
HY_BUNDLE_CONFIGURATION_EXCEPTION	311	bundle配置异常
HY_YTSDK_CONFIGURATION_EXCEPTION	312	优图配置异常
HY_PLEASE_CALL_FIRST_INIT_API	313	先调用初始化接口

HY_SDK_AUTH_FAILED	314	SDK 授权失败
HY_USER_VOLUNTARILY_CANCELED	315	用户手动取消
HY_YTSDK_LOCAL_AUTH_FAILED	316	SDK 人脸本地检测失败
HY_CAMERA_OPEN_FAIL	317	相机开启失败
HY_DONOT_SWITCH_APPS	318	请勿在核身过程中切换应用
HY_CAMEREA_PERMISSION_EXCEPTION	319	摄像头权限异常
HY_SDK_VEDIO_CUT_EXCEPTION	320	视频裁剪失败
HY_LIGHT_DATA_FORMAT_EXCEPTION	321	光线数据格式错误
HY_GET_REMOTE_DATA_EXCEPTION	322	获取远程数据错误

常见问题与修复指引

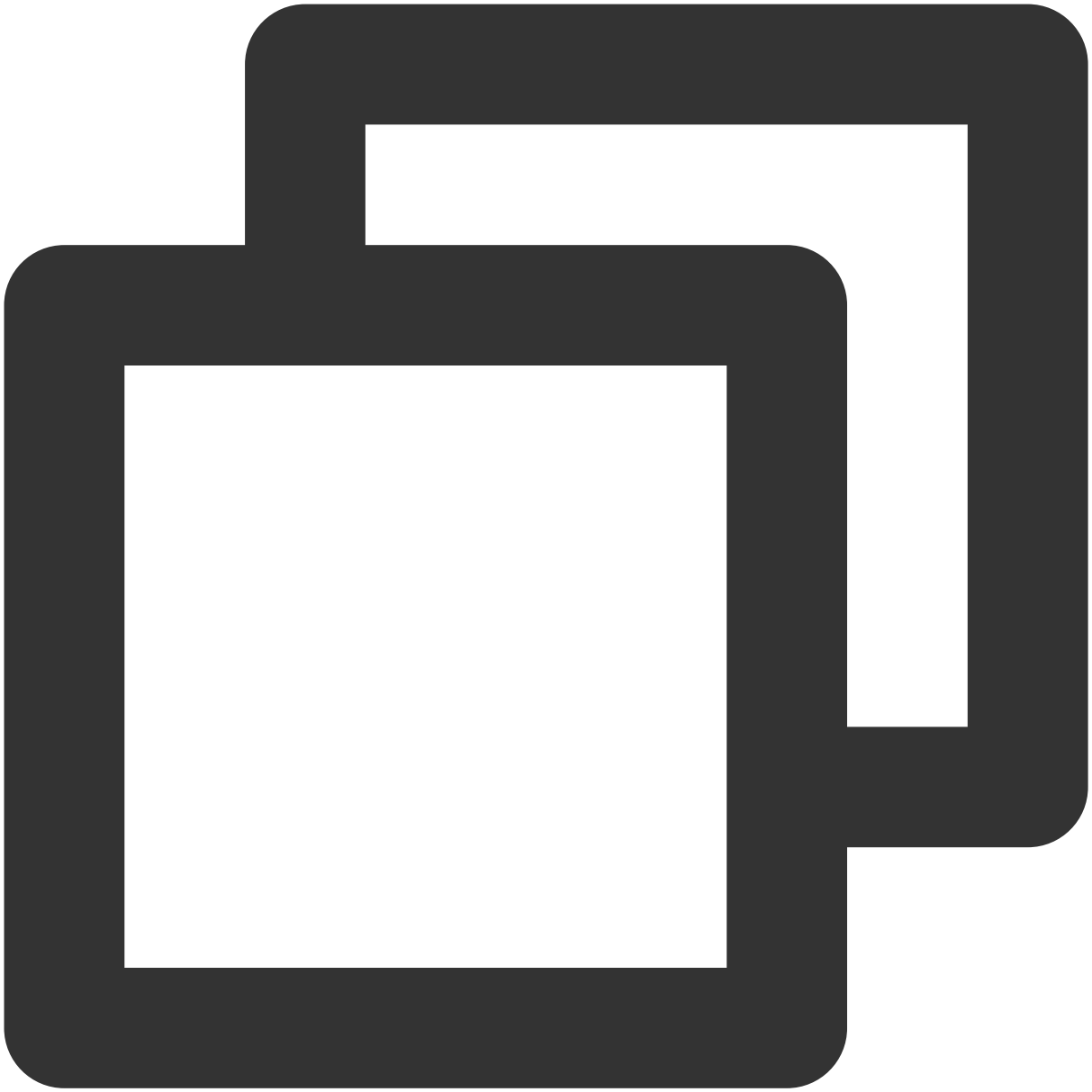
最近更新时间：2023-06-12 15:33:00

本文主要介绍Identity Verification(App SDK)的场景问题，并且给出相应的修复指引

客户端相关

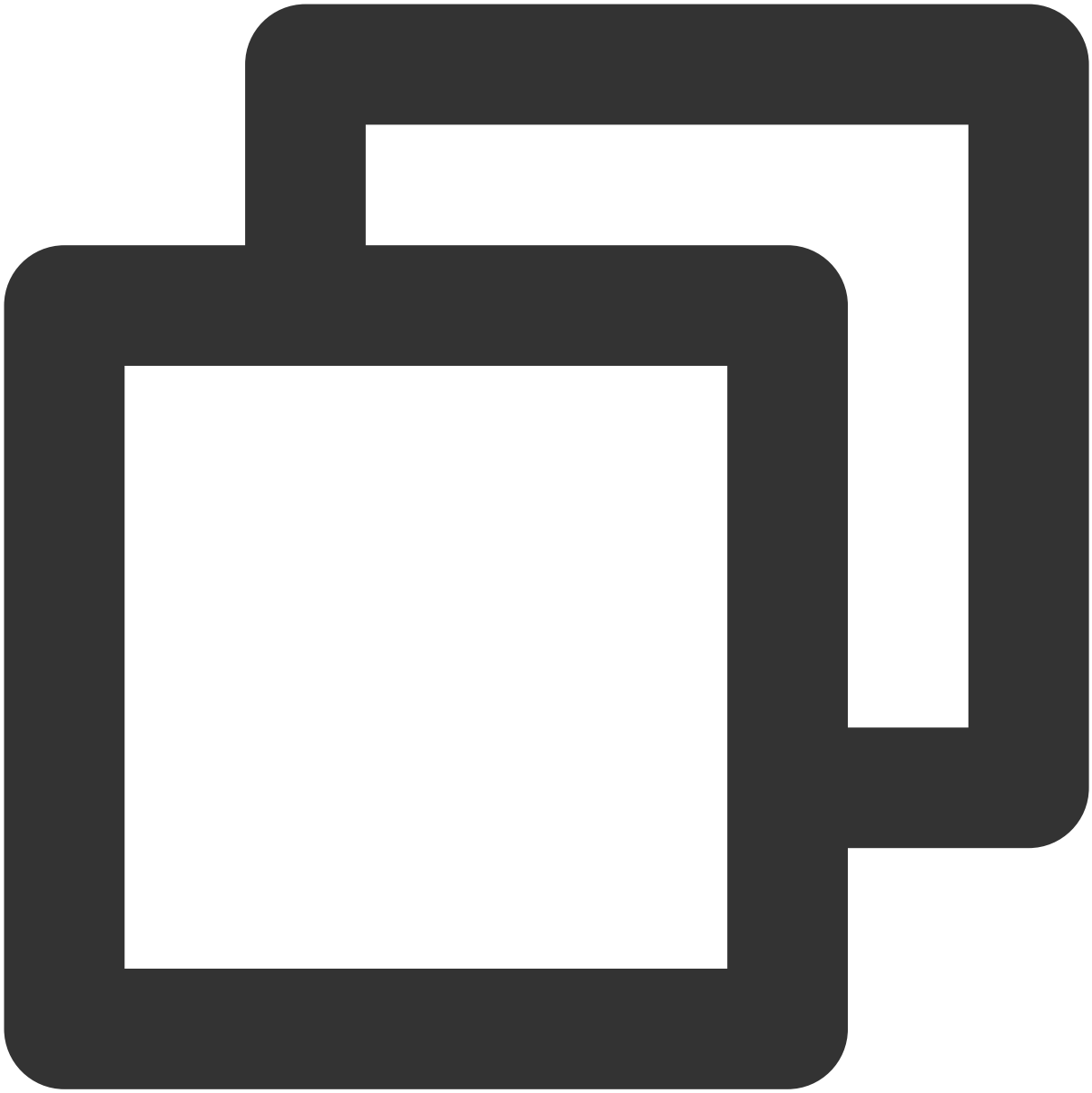
Android 端常见问题

1. 集成慧眼后出现Invoke-customs are only supported starting with Android O (--min-api 26)错误？需要在build.gradle中添加如下配置：



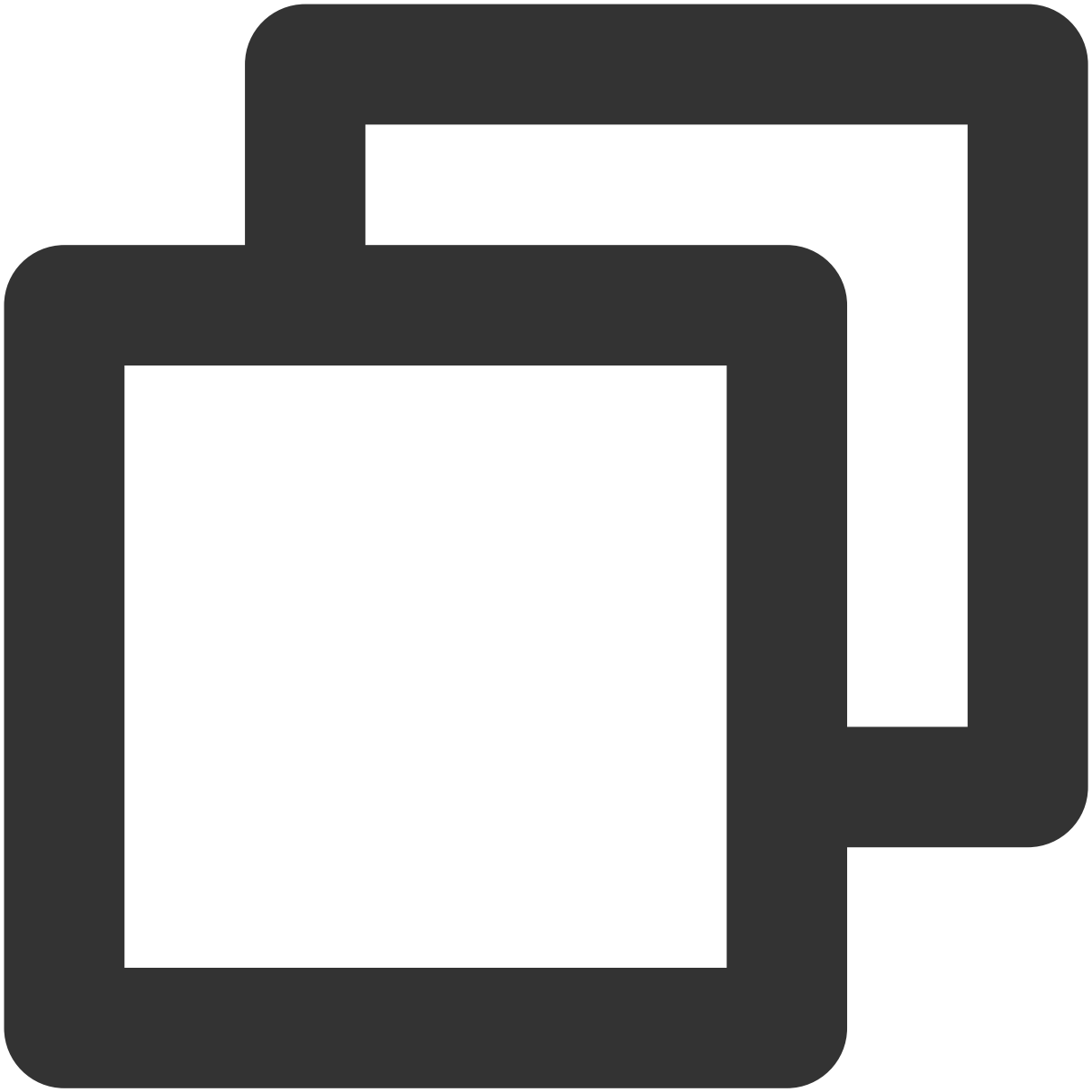
```
// java版本支持1.8
    compileOptions {
           sourceCompatibility JavaVersion.VERSION_1_8
           targetCompatibility JavaVersion.VERSION_1_8
     }
```

2. 如果集成方使用了AndResGuard的混淆工具，可以添加混淆配置：



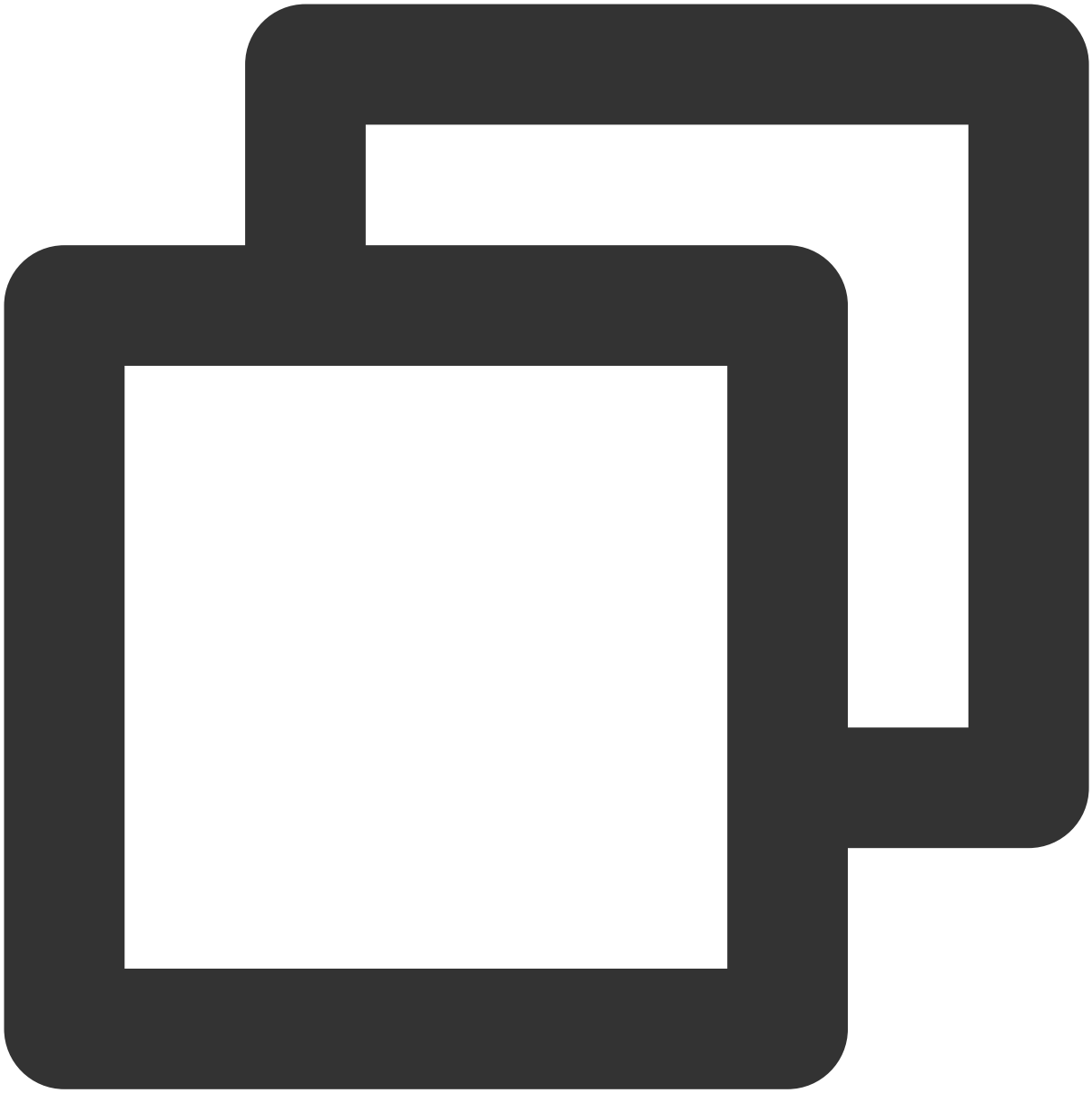
```
// for HuiYanSDK
"R.string.ocr_*",
"R.string.rst_*",
"R.string.net_*",
"R.string.msg_*",
"R.string.fl_*",
```

3. Android X针对一些低版本的设备报错 android.content.res.Resources$NotFoundException:from xml type xml resource ID #0x7f0800c3 这一类的错误，可以考虑添加矢量图依赖：



```
// 低版本的矢量图
 implementation
 'androidx.vectordrawable:vectordrawable:1.1.0'
```

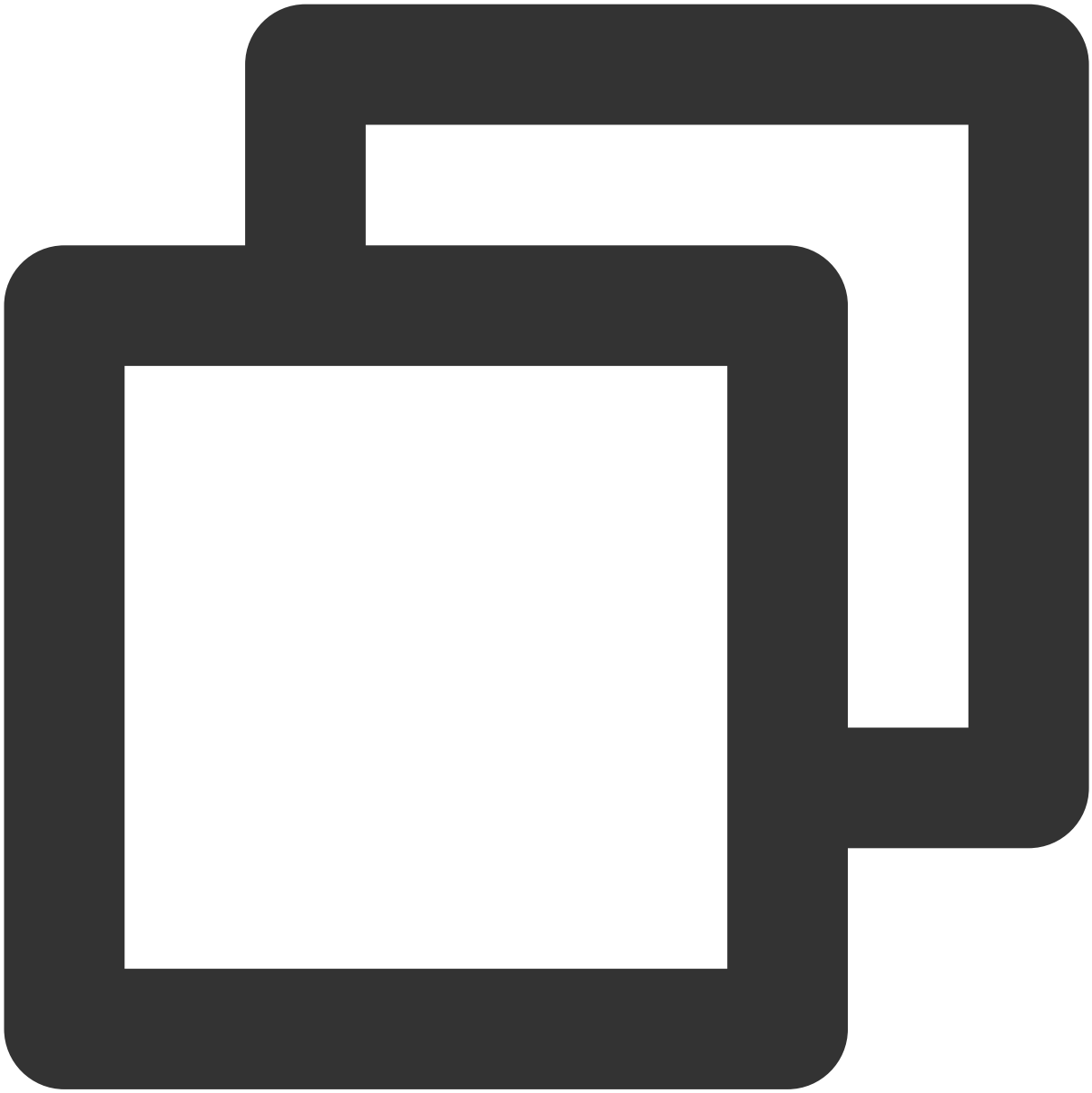
4. Android Support 如果在低版本设备上（6.0及以下）出现如下报错：



```
android.content.res.Resources$NotFoundException: File
res/drawable/$txy_face_id_logo__0.xml from color state list resource ID #0x7f070001
        at android.content.res.Resources.loadColorStateListForCookie(Resources.jav
        at android.content.res.Resources.loadColorStateList(Resources.java:2749)
        at android.content.res.TypedArray.getColor(TypedArray.java:441)
        at android.content.res.XResources$XTypedArray.getColor(XResources.java:128
        at android.support.v4.content.res.TypedArrayUtils.getNamedColor(TypedArray
        at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.update
        at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.inflat
        at android.support.graphics.drawable.VectorDrawableCompat.inflateInternal(
        at android.support.graphics.drawable.VectorDrawableCompat.inflate(VectorDr
```

```
        at android.support.graphics.drawable.VectorDrawableCompat.createFromXmlInn
        at android.support.v7.widget.AppCompatDrawableManager$VdcInflateDelegate.c
```

需要更新support依赖到最新版本28.0.0：



```
implementation 'com.android.support:appcompat-v7:28.0.0'
// 兼容低版本矢量图依赖的组件库
implementation 'com.android.support:support-vector-drawable:28.0.0'
implementation 'com.android.support:animated-vector-drawable:28.0.0'
```

iOS 端常见问题

1. 进入SDK，出现crash 打印日志 "reason: '* -[__NSDictionaryM setObject:forKey:]: key cannot be nil'"，**处理方法是在build Settings - Other Linker Flags 添加 -**ObjC 。

2. 编译时报：

Undefined symbol: _vImageConvert_Planar16FtoPlanarF

Undefined symbol: _vImageConvert_PlanarFtoPlanar16F

处理方法：添加系统库Accelerate.framework

3. **"face-tracker-v001 bundle path is nil"** 或者 **HuiYanSDKUI bundle path is nil** 添加两个bundle 资源文件至Copy Bundle Resources。

Identity verification（移动端 H5）接入指引

接入流程说明

最近更新时间：2024-01-25 10:41:35

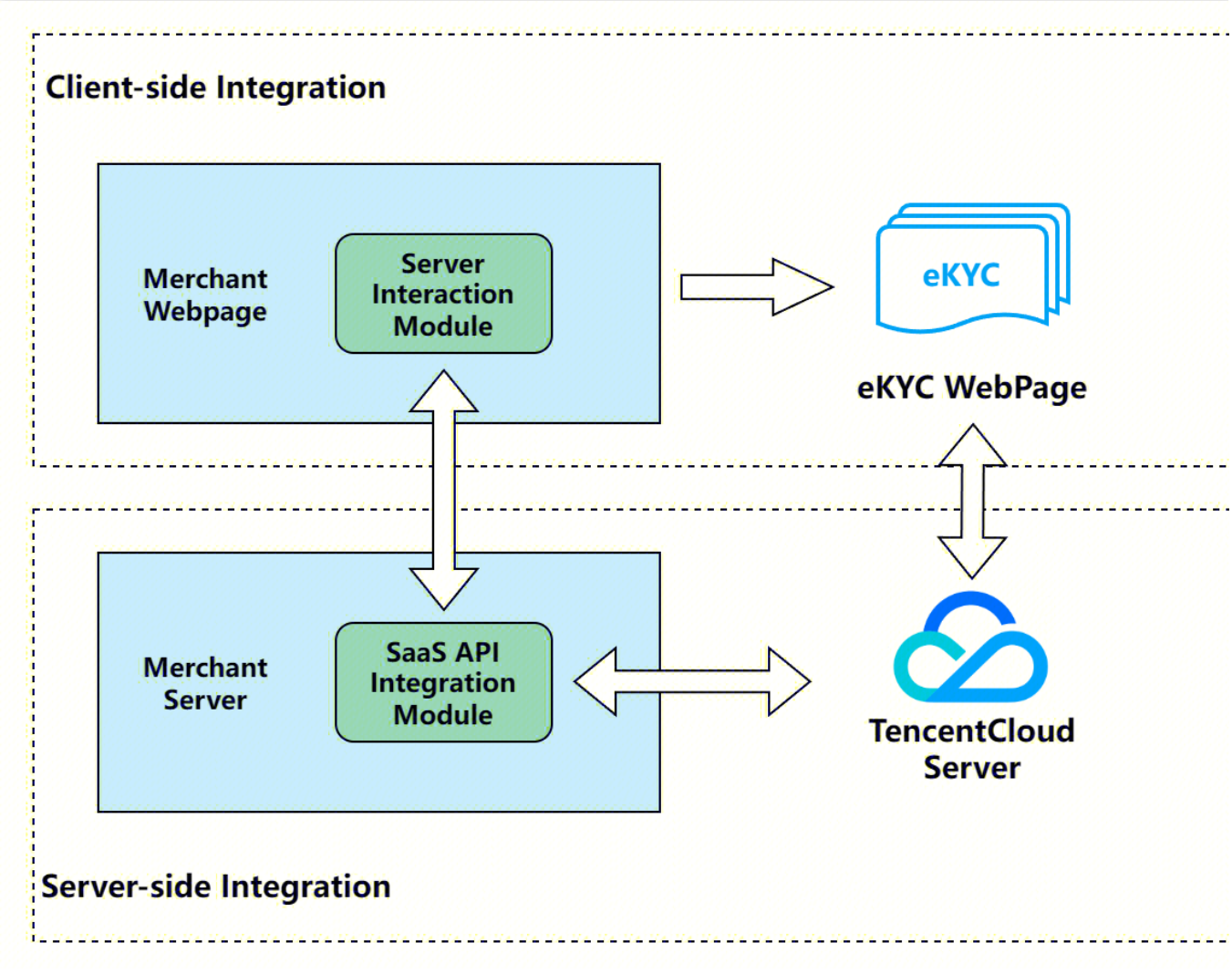
本文介绍 eKYC 活体比对（移动端 H5）整体接入的流程。

接入准备

注册腾讯云企业账号，请参见 注册指引。
完成企业实名认证，请参见 企业实名指引。
登录 eKYC 控制台 开通服务。

整体架构图

下图为 eKYC 活体比对（移动端 H5）的架构图：



整体交互流程

下图展示了客户和腾讯云 eKYC 的整体交互逻辑，图中角色说明：

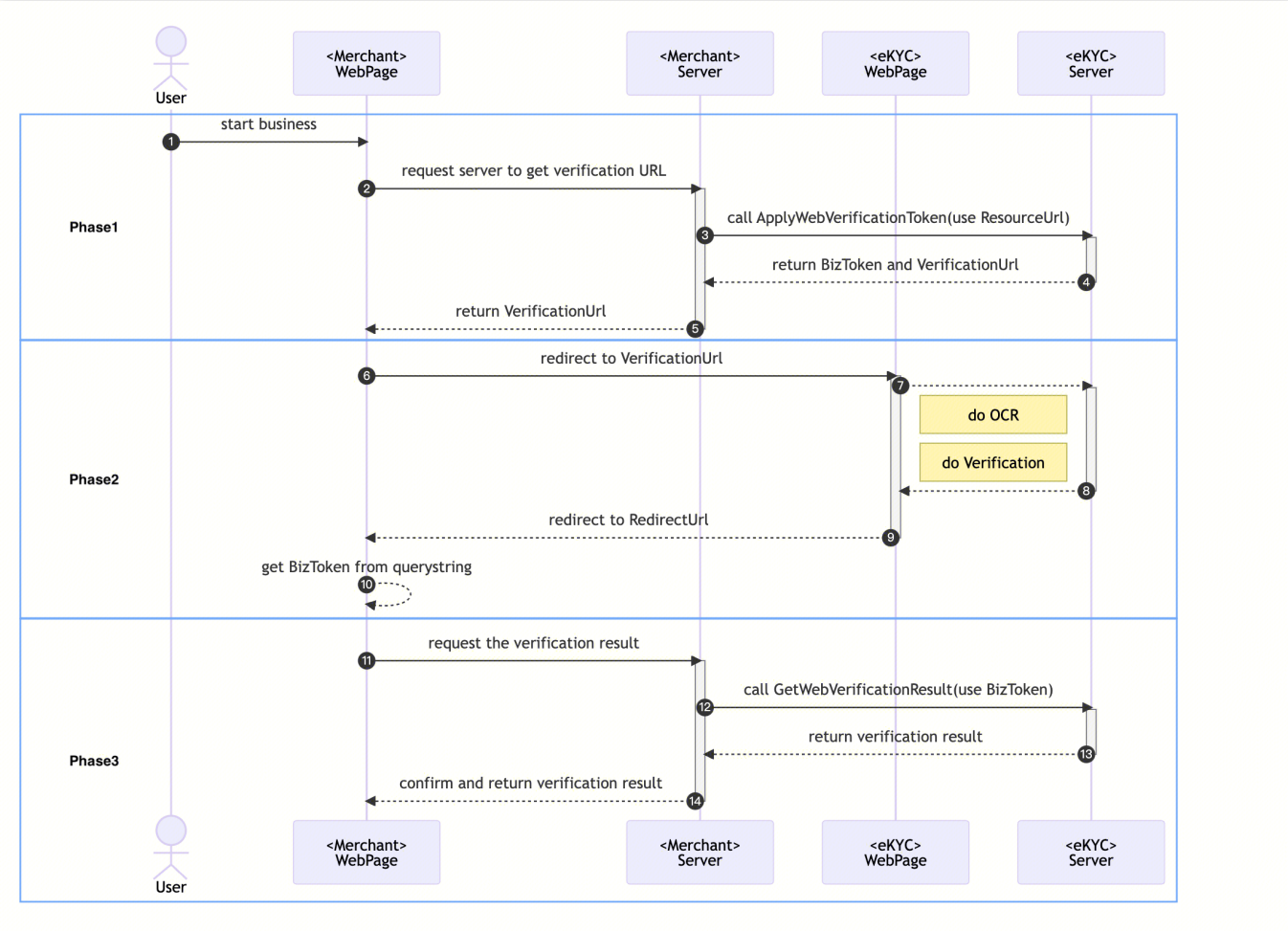
User：移动端 H5 用户

Merchant WebPage：客户前端页面

Merchant Server：客户后端服务

eKYC WebPage：eKYC 前端页面

eKYC Server：eKYC 后端服务



具体的推荐交互流程详细交互如下：

阶段一

1. 用户端业务触发核验流程。
2. Merchant WebPage 发送请求到 Merchant Server，告知 Merchant Server 需要发起一次核验流程。
3. Merchant Server 传入相关参数调用 ApplyWebVerificationBizTokenIntl 接口，参考服务端集成第一步。
4. eKYC Server 接收到请求后，将该次核验流程的 BizToken 和 VerificationURL 返回给 Merchant Server。
5. Merchant Server 将获取到的 BizToken 进行保存，并将 VerificationURL 下发给 Merchant WebPage。

阶段二

1. Merchant WebPage 跳转到 VerificationURL，打开 eKYC WebPage，参考前端集成第一步。
2. 用户在 eKYC WebPage 上进行 OCR 证件识别和核验流程。
3. 核验完成后，eKYC Server 将核验结果下发给 eKYC WebPage，Merchant WebPage 展示结果页面。
4. 用户点击下一步后，eKYC WebPage 会跳转回 RedirectURL 地址，并在 URL 上拼接 token 参数。
5. Merchant WebPage 通过地址获取到当前核验流程的 token 参数，参考前端集成第二步。

阶段三

1. Merchant WebPage 发送请求到 Merchant Server，告知 Merchant Server 获取核验的结果信息。
2. Merchant Server 传入相关参数调用 GetWebVerificationResultIntl 接口，参考服务端集成第二步。
3. eKYC Server 接收到请求后，将该次核验流程的详细信息返回给 Merchant Server。
4. Merchant Server 将结果信息返回给 Merchant WebPage，Merchant WebPage 根据结果进行后续业务流程。

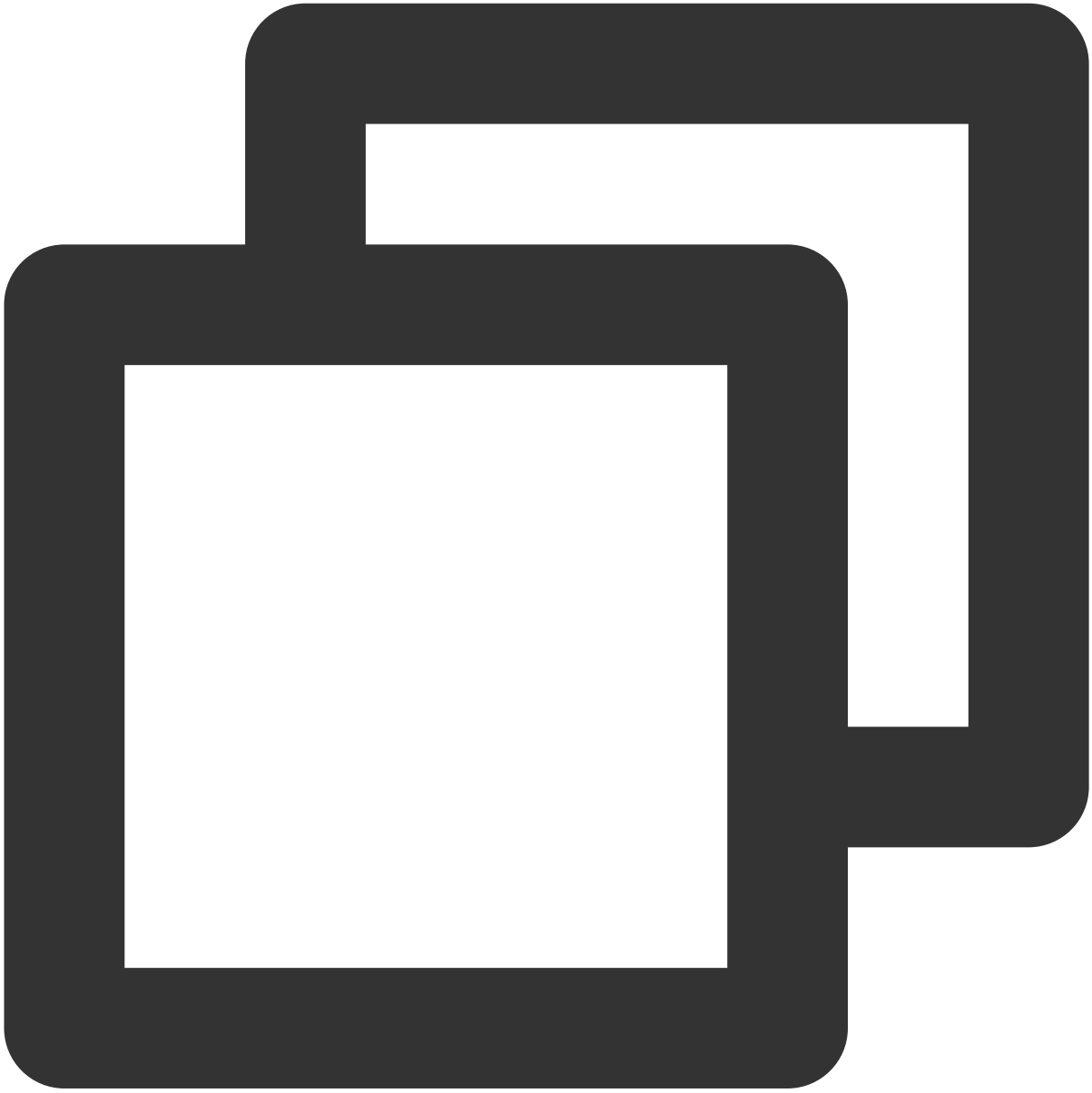
服务端集成

1. 调用申请 Web 核验令牌生成核验地址（对应阶段一）

调用 ApplyWebVerificationBizTokenIntl 接口，获取 BizToken 和核验地址 VerificationURL。对应时序图的第3点。
RedirectURL：核验结束后重定向的 Web 回跳地址，包含协议头，主机名和路径，
如：`https://www.tencentcloud.com/products/faceid`。核验流程完成后，回跳地址会拼接当次流程的 BizToken，以`https://www.tencentcloud.com/products/faceid?token={BizToken}`的格式进行跳转。
Extra：业务透传参数，最大长度为1000个字符，会在 GetWebVerificationResultIntl 接口中返回，如无需要可以不传。
Config：自定义核验页面的相关配置。
AutoSkip：验证成功时，是否跳过结果展示页面，自动跳转到 RedirectURL，默认为 false。
CheckMode：检测模式，此参数为必填。参数取值如下：1：OCR + 活体检测 + 人脸比对；
IDCardType: 支持识别的卡证类型，此参数为必填。当前支持的卡证如下所示：
HKIDCard：中国香港身份证
MLIDCard：马来西亚身份证
IndonesiaIDCard：印度尼西亚身份证
PhilippinesVoteID：菲律宾选民卡
PhilippinesDrivingLicense：菲律宾驾驶证
PhilippinesTinID：菲律宾 TinID
PhilippinesSSSID：菲律宾 SSSID
PhilippinesUMID：菲律宾 UMID
InternationalIDPassport：港澳台地区以及境外护照
IndonesiaDrivingLicense：印度尼西亚驾驶证
ThailandIDCard：泰国身份证
ThailandDrivingLicense：泰国驾驶证
MLDrivingLicense：马来西亚驾驶证
SingaporeIDCard：新加坡身份证
SingaporeDrivingLicense：新加坡驾驶证
JapanIDCard：日本身份证
JapanDrivingLicense：日本驾驶证
PhilippinesIDCard：菲律宾身份证通用

DisableCheckOcrWarnings：是否关闭证件告警，默认为 false（开启告警检测功能）。当开启时，身份认证流程将会根据证件的告警情况进行拦截。若您需要使用证件鉴伪功能，请 联系我们。

接口调用代码示例



```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.faceid.v20180301.FaceidClient;
import com.tencentcloudapi.faceid.v20180301.models.*;;
```

```
public class ApplyWebVerificationBizTokenIntl
{
    public static void main(String [] args) {
        try{
            // 实例化一个认证对象，入参需要传入腾讯云账户secretId，secretKey,此处还需注意密钥
            // 密钥可前往https://console.tencentcloud.com/cam/capi网站进行获取
            Credential cred = new Credential(
                    "TENCENTCLOUD_SECRET_ID",
                    "TENCENTCLOUD_SECRET_KEY"
            );
            // 实例化一个http选项，可选的，没有特殊需求可以跳过
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("faceid.ap-guangzhou.tencentcloudapi.woa.com");
            // 实例化一个client选项，可选的，没有特殊需求可以跳过
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            // 实例化要请求产品的client对象,clientProfile是可选的
            FaceidClient client = new FaceidClient(cred, "ap-singapore", clientProf
            // 实例化一个请求对象,每个接口都会对应一个request对象
            ApplyWebVerificationBizTokenIntlRequest req = new ApplyWebVerificationB
            req.setRedirectURL("https://www.tencentcloud.com/products/faceid");
            WebVerificationConfigIntl webVerificationConfigIntl = new WebVerificati
            webVerificationConfigIntl.setCheckMode(1L);
            webVerificationConfigIntl.setIDCardType("HKIDCard");
            req.setConfig(webVerificationConfigIntl);
            // 返回的resp是一个ApplyWebVerificationBizTokenIntlResponse的实例，与请求对象
            ApplyWebVerificationBizTokenIntlResponse resp = client.ApplyWebVerifica
            // 输出json格式的字符串回包
            System.out.println(ApplyWebVerificationBizTokenIntlResponse.toJsonStrin
            String bizToken = resp.getBizToken();
            String verificationURL = resp.getVerificationURL();
            System.out.printf("BizToken: %s, VerificationURL: %s", bizToken, verifi
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

2. 确认当次核验流程的结果（对应阶段三）

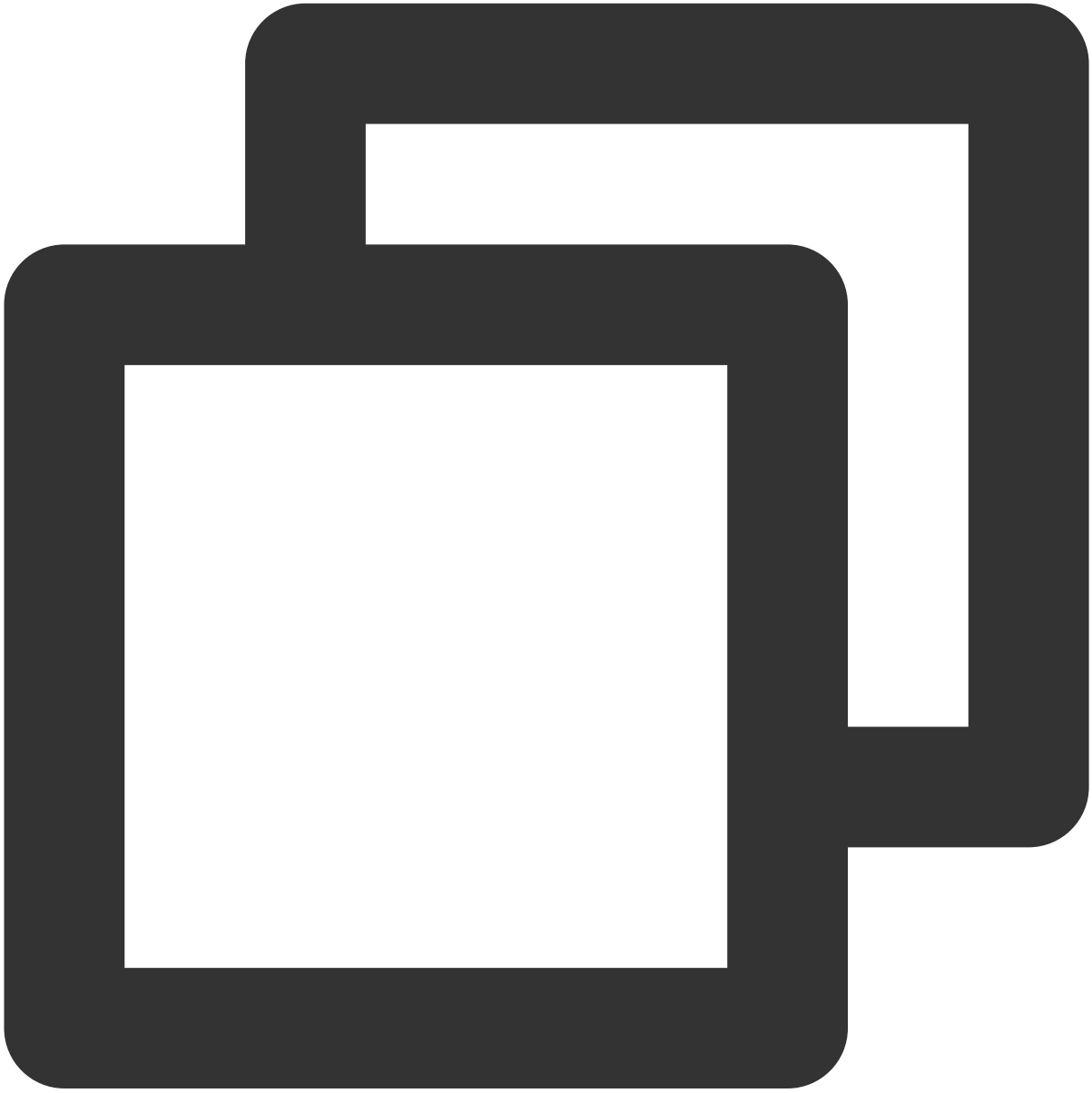
核验流程结束后，商户前端通知商户服务端获取本次核验的结果，商户服务端调用 GetWebVerificationResultIntl 接口，将最终的结果返回给前端页面。对应时序图的第12点。

本次核验的最终结果应以该接口返回的信息为准，当响应的 ErrorCode 字段为0时，视为本次核验流程通过，其他情况下视为未通过。详细的错误码列表请参考Liveness Detection and Face Comparison (Mobile HTML5) Error

Codes。

BizToken：由 ApplyWebVerificationBizTokenIntl 接口生成的 BizToken，当次核验流程的唯一标识。

接口调用代码示例



```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.faceid.v20180301.FaceidClient;
import com.tencentcloudapi.faceid.v20180301.models.*;
```

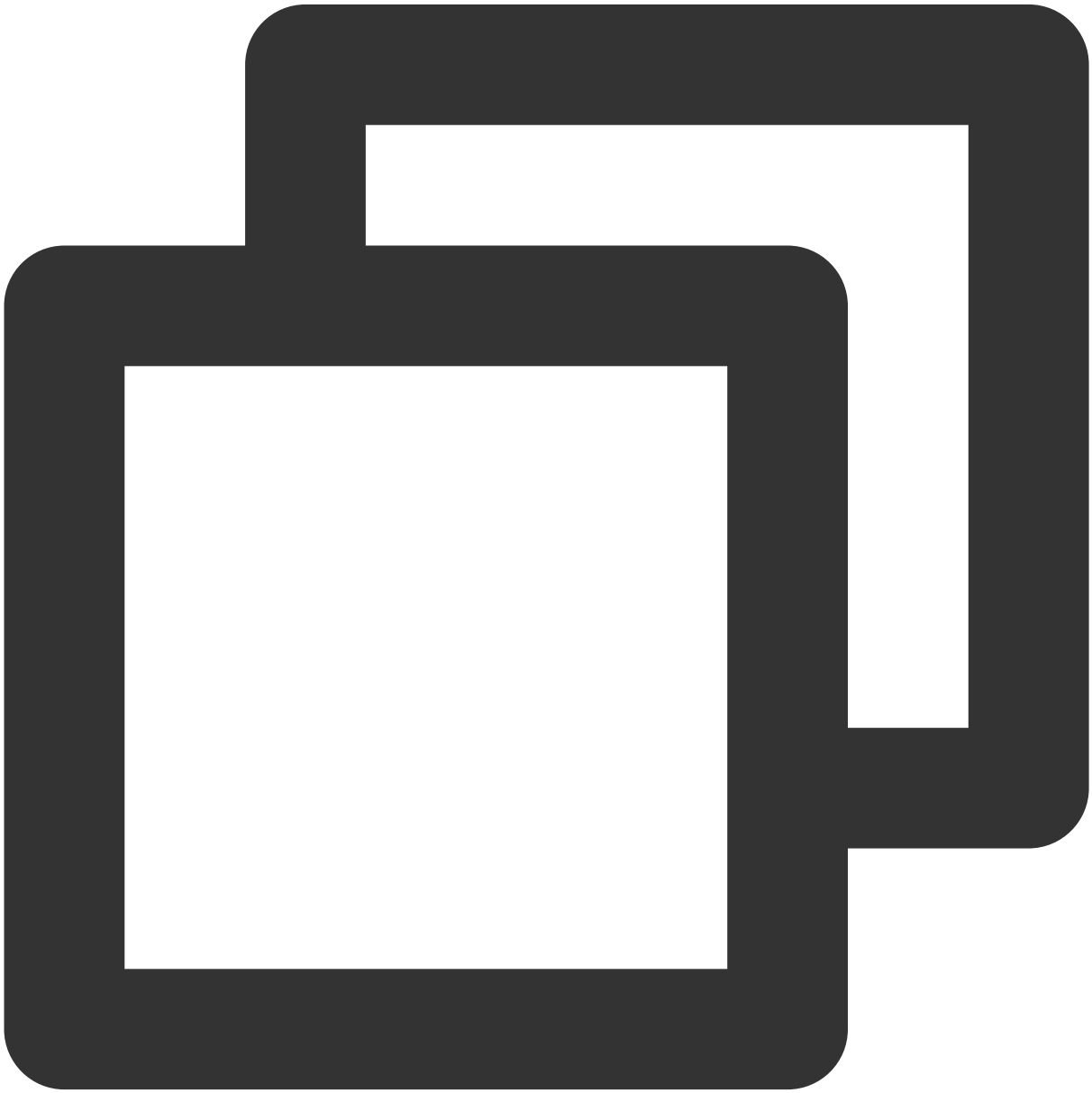
```
import java.util.Arrays;

public class GetWebVerificationResultIntl {
    public static void main(String [] args) {
        try{
            // 实例化一个认证对象，入参需要传入腾讯云账户secretId，secretKey,此处还需注意密钥
            // 密钥可前往https://console.tencentcloud.com/cam/capi网站进行获取
            Credential cred = new Credential(
                    "TENCENTCLOUD_SECRET_ID",
                    "TENCENTCLOUD_SECRET_KEY"
            );
            // 实例化一个http选项，可选的，没有特殊需求可以跳过
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("faceid.ap-guangzhou.tencentcloudapi.woa.com");
            // 实例化一个client选项，可选的，没有特殊需求可以跳过
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            // 实例化要请求产品的client对象,clientProfile是可选的
            FaceidClient client = new FaceidClient(cred, "ap-singapore", clientProf
            // 实例化一个请求对象,每个接口都会对应一个request对象
            GetWebVerificationResultIntlRequest req = new GetWebVerificationResultI
            req.setBizToken("xxx"); //输入ApplyWebVerificationBizTokenIntl阶段返回的B
            // 返回的resp是一个GetWebVerificationResultIntlResponse的实例，与请求对象对应
            GetWebVerificationResultIntlResponse resp = client.GetWebVerificationRe
            // 输出json格式的字符串回包
            System.out.println(GetWebVerificationResultIntlResponse.toJsonString(re
            Long errorCode = resp.getErrorCode();
            String errorMsg = resp.getErrorMsg();
            //详细 OCRResult 字段，请参考OCRResult相关文档https://www.tencentcloud.com/
            OCRResult[] ocrResult = resp.getOCRResult();
            if (errorCode == 0) {
                // 核验通过
                System.out.println("Success");
                System.out.printf("OCRResult:%s", Arrays.toString(ocrResult));
            }else {
                // 核验不通过
                System.out.printf("Fail: %s\\n", errorMsg);
            }
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

前端集成

1. 获取 VerificationURL 并跳转，发起核验流程（对应阶段二）

客户前端页面获取服务端请求到的 VerificationURL，并跳转到该地址进入核验流程，用户按照提示完成活体比对流程。对应时序图的第6点。

代码示例



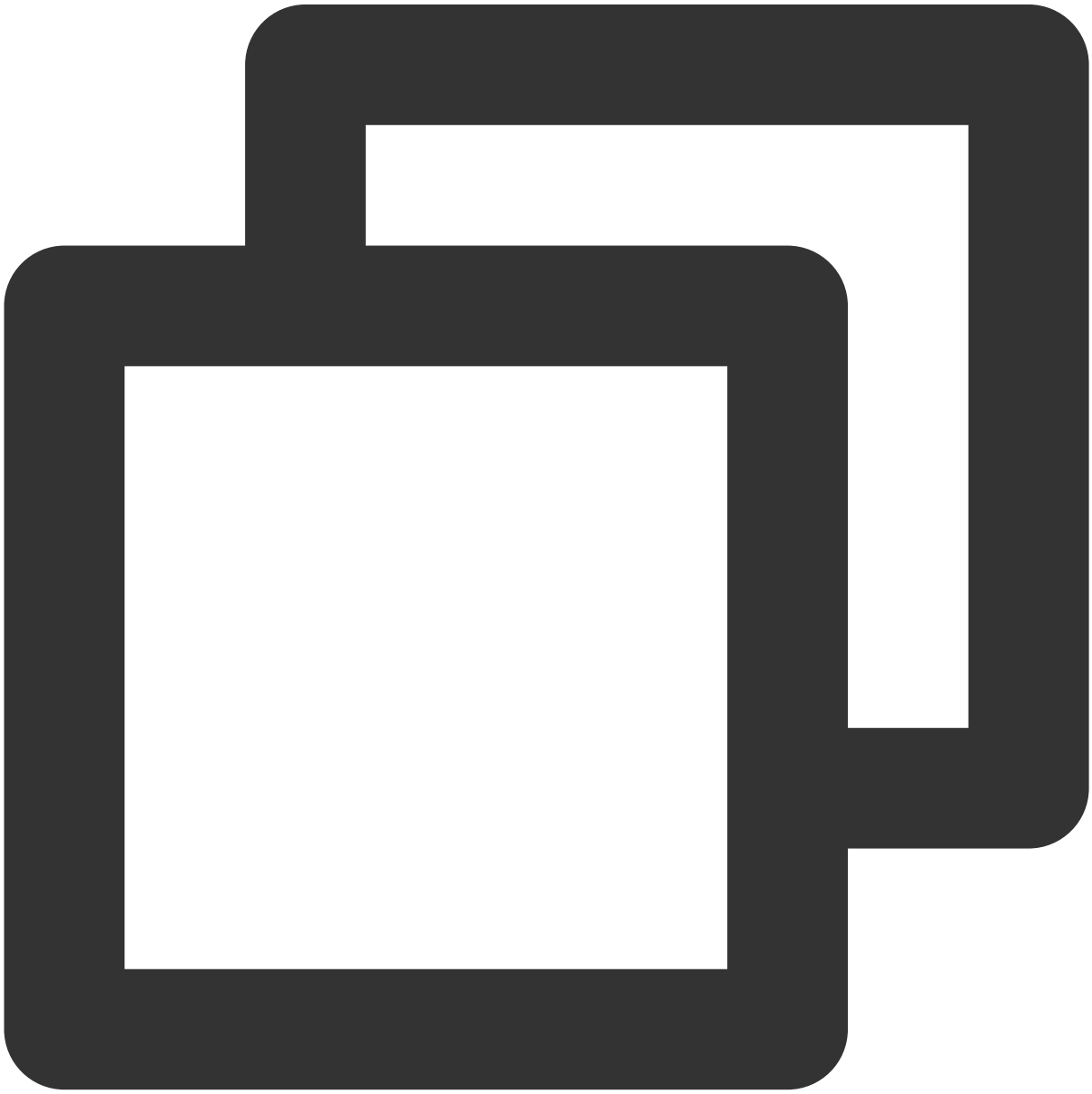
```
// 从服务端获取VerificationURL
```

```
const VerificationURL = 'https://sg.faceid.qq.com/reflect/?token=*****';
// 前端页面跳转
window.location.href = VerificationURL;
```

2. 从回调地址获取 BizToken，向后端请求核验结果（对应阶段二）

在核验完成后，页面会跳转到 RedirectURL 上。RedirectURL 会拼接当次流程的 BizToken 参数，通过解析 RedirectURL 获取 BizToken 参数，用于拉取本次活体比对的结果信息。对应时序图的第12点。

代码示例



```
// 获取RedirectURL
const RedirectURL = "https://*****?token={BizToken}";

// 解析获得RedirectURL的BizToken参数，用于拉取本次活体比对的结果信息
const bizToken = getURLParameter(RedirectURL, "token");
if (bizToken) {
  // 使用bizToken用于拉取本次活体比对的结果信息
}

/**
 * 解析url的参数
/* @params url 查询url
/* @params variable 查询参数
*/
function getURLParameter(url, variable) {
  const query = url.split('?')[1] || '';
  const vars = query.split('&');
  for (let i = 0; i < vars.length; i++) {
    const pair = vars[i].split('=');
    if (pair[0] == variable) {
      return pair[1];
    }
  }
  return (false);
}
```

活体人脸比对（纯 API）接入指引
接入流程说明

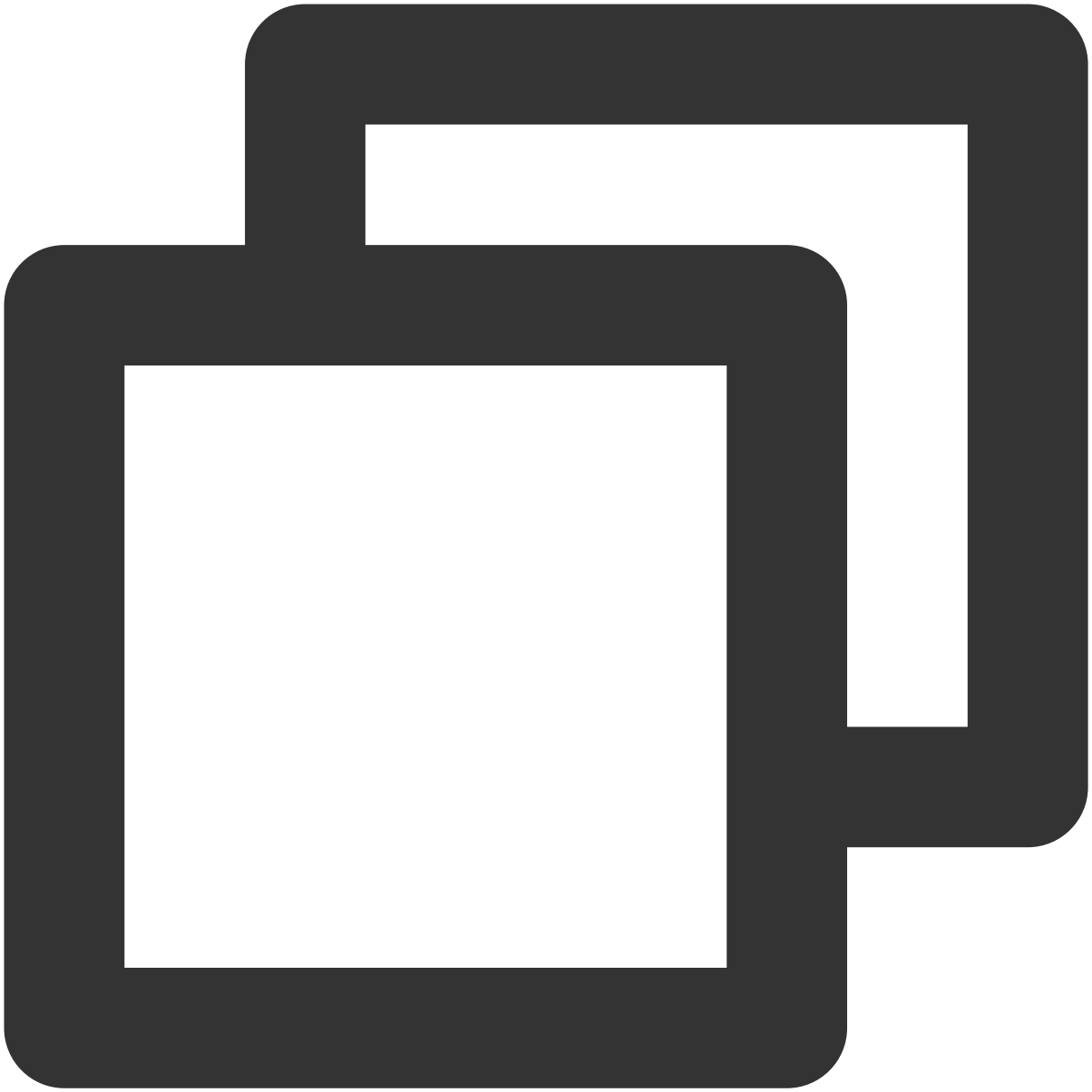
最近更新时间：2023-11-29 15:59:56

本文介绍活体人脸比对纯 API 模式整体接入流程，该 API 与语言无关，您可以使用任何您熟悉的语言进行集成。但是在使用 API 集成活体人脸比对服务之前，您最好能够阅读理解 连接腾讯云 API 接口，它可以告诉你如何使用腾讯云 SDK 以简化 API 接入的流程。在这篇文章中，我们仅使用 Java API SDK 演示如何集成我们的 API。

接入准备

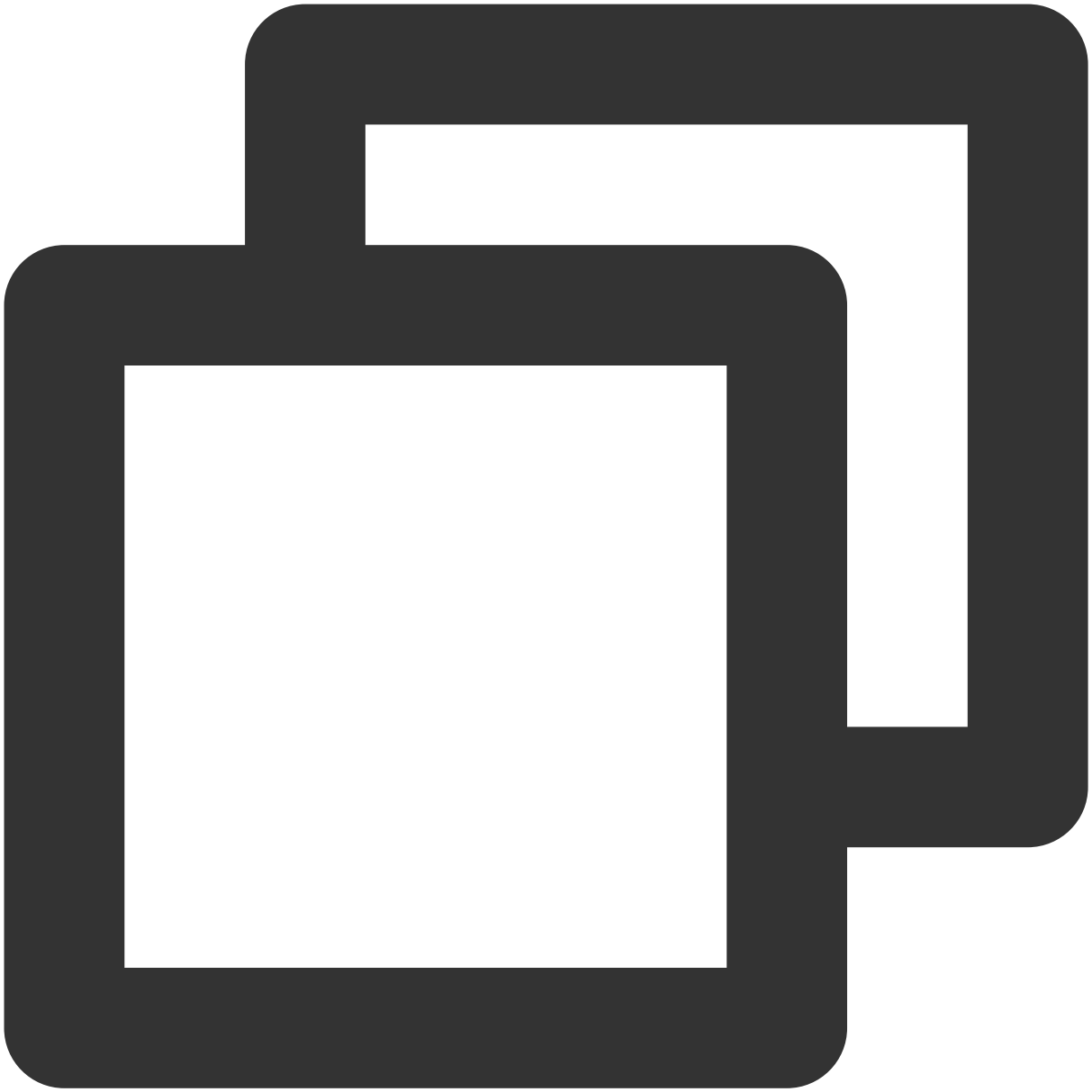
注册腾讯云企业账号，请参见 注册指引。
完成企业实名认证，请参见 企业实名指引。
登录慧眼控制台 开通服务。
获取 API 访问 秘钥。

引入腾讯云 API SDK



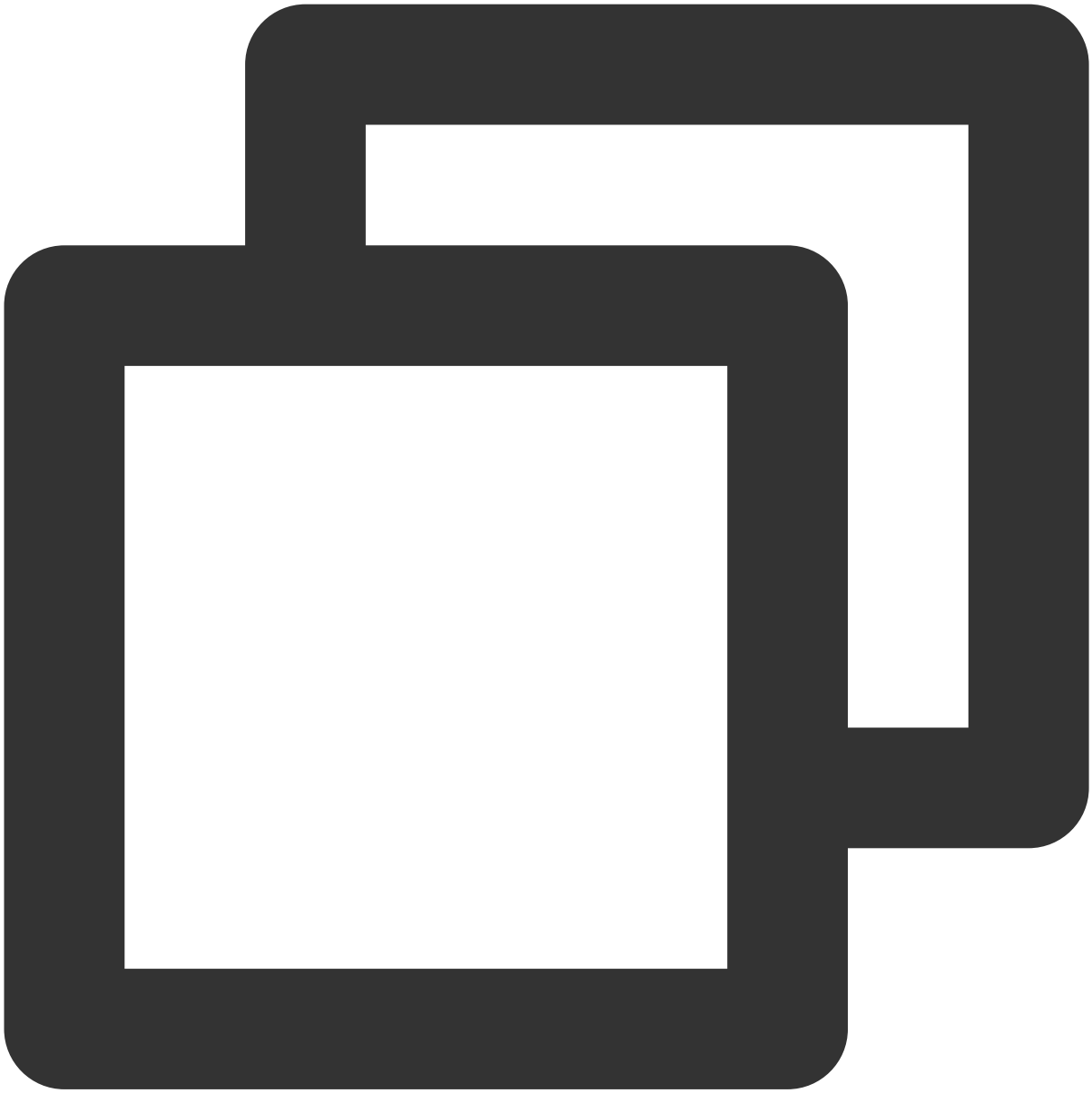
```
<dependency>
        <groupId>com.tencentcloudapi</groupId>
        <artifactId>tencentcloud-sdk-java-intl-en</artifactId>
        <version>3.0.798</version>
</dependency>
```

API SDK 客户端初始化以及配置



```
// Instantiate an authentication object. The Tencent Cloud account `secretId` and `
Credential cred = new Credential("secretId", "secretKey");

// Instantiate the client object of the requested product
ClientProfile clientProfile = new ClientProfile();
clientProfile.setSignMethod(ClientProfile.SIGN_TC3_256);

FaceidClient client = new FaceidClient(cred, "ap-singapore", clientProfile);
```

用 CompareFaceLiveness API



```
// Step 1: Instantiate the request object and provide necessary parameters
CompareFaceLivenessRequest request = new CompareFaceLivenessRequest();
request.setLivenessType("SILENT");
request.setImageBase64(getBase64(cmd.getOptionValue(IMAGE_PATH)));
request.setVideoBase64(getBase64(cmd.getOptionValue(VIDEO_PATH)));

// Step 2: Call the Tencent Cloud API through FaceIdClient
CompareFaceLivenessResponse response = client.CompareFaceLiveness(request);
```

```
// Step 3: Process the Tencent Cloud API response and construct the return object
System.out.println(CompareFaceLivenessResponse.toJsonString(response));
```

使用示例

可以在 GitHub 仓库查看完整的示例源码：

CompareFaceLiveness

API 文档

可以在腾讯云官网查看完整的 API 接口文档：

CompareFaceLiveness

其他指引
三分钟跑通接口

最近更新时间：2024-07-29 11:43:48

操作场景

该文档指导您在购买腾讯云人脸核身服务后，通过 API 3.0 Explorer 在线接口调试页面调用腾讯云人脸核身接口，并将该接口对应语言的 SDK 集成到项目。您只需要完成以下步骤，即可快速接入腾讯云人脸核身接口。

前提条件

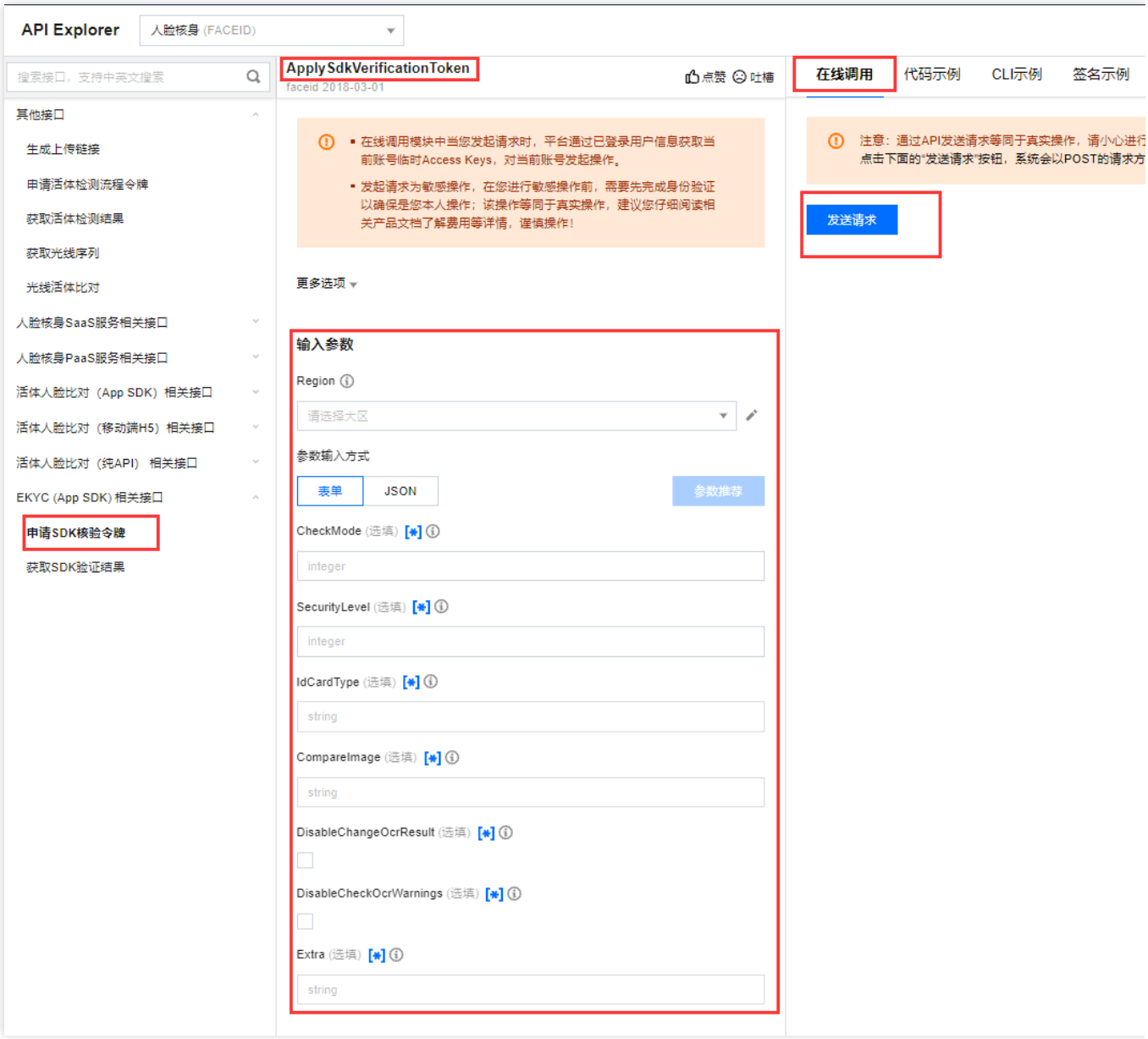
进入腾讯云人脸核身 API 3.0 Explorer 在线接口调试页面，按照以下操作步骤调用接口。

操作步骤

以下步骤以调用 申请 SDK 核验令牌 接口为示例

步骤一：

在 API 3.0 Explorer 在线接口调试页面 左侧导航栏，选择申请SDK核验令牌的接口，填入必填参数，发起调用，得到响应结果 。



Region 参数：参考接口文档为必填参数，域名中的地域信息，该参数决定访问的接入点，例如 `ap-hongkong` 就是要访问中国香港这个接入点 。每个接口支持的 Region 可能会有所不同，具体可参考您当前调用的接口文档中的 `list of regions` ，如本篇示例 申请 SDK 核验令牌 接口的可支持 Region 需参考 list of regions。

2. Input Parameters

The following request parameter list only provides API request parameters and some common parameters. For the complete common parameter list, see Common Request Parameters.

Parameter Name	Required	Type	Description
Action	Yes	String	Common Params. The value used for this API: ApplySdkVerificationToken.
Version	Yes	String	Common Params. The value used for this API: 2018-03-01.
Region	Yes	String	Common Params. For more information, please see the list of regions supported by the product. This API only supports: ap-hongkong, ap-singapore.
NeedVerifyIdCard	Yes	Boolean	Whether ID card authentication is required. If not, only document OCR will be performed. Currently, authentication is available only when the value of IdCardType is HK.

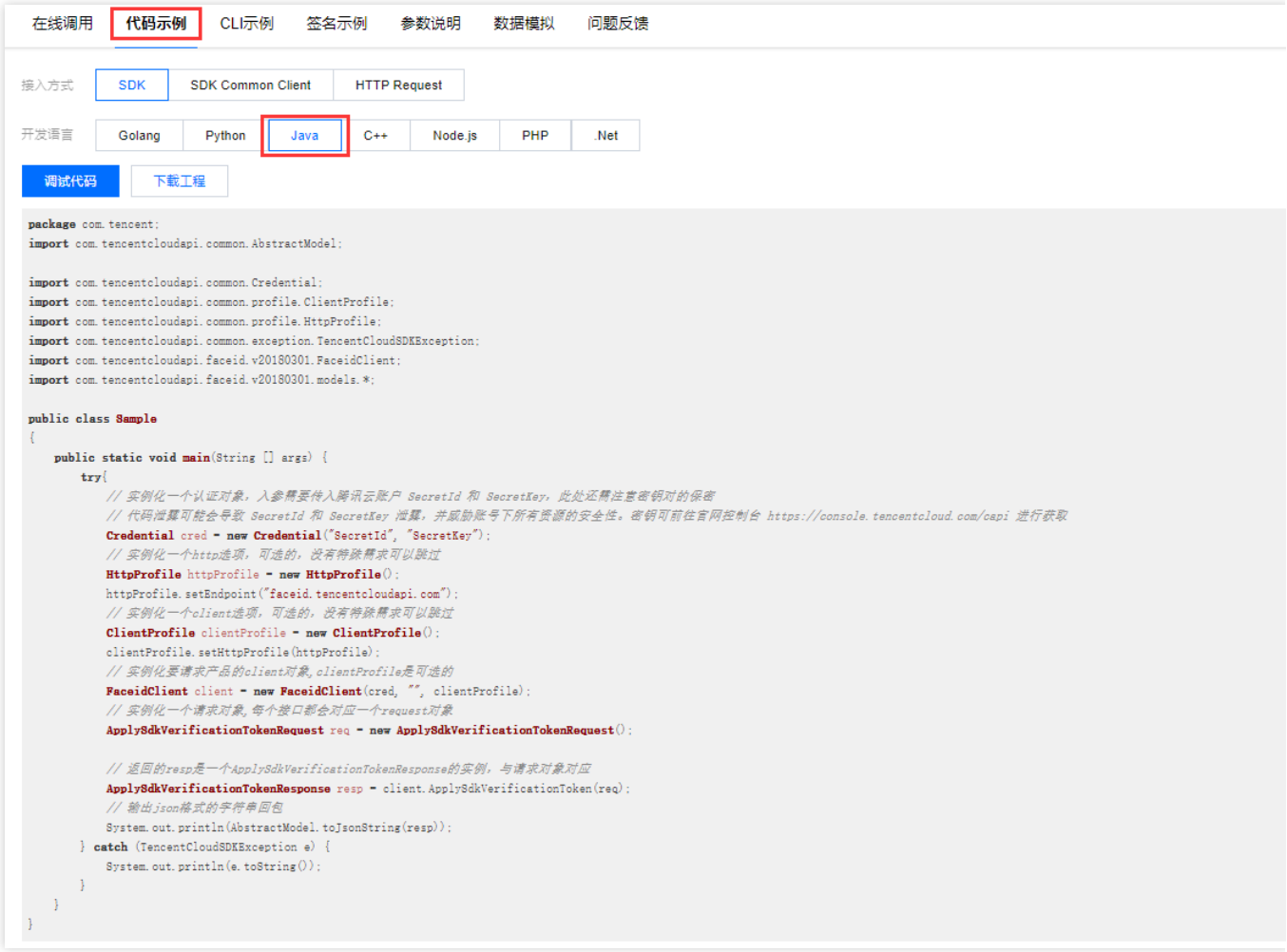
NeedVerifyIdCard 参数：参考接口文档为必填参数，表示是否需要身份证鉴伪，如果不需要，则仅做证件OCR。当前仅IdCardType为HK支持鉴伪 。

步骤二：

选择对应的后台语言，生成代码，集成 SDK 到项目中，这里以 Java 为例：

1. 集成代码到项目中，需要导入 SDK 依赖包（可参考以下截图右上角 JAVA SDK 使用说明）、获取密钥信息并填充到代码的 `SecretId` 和 `SecretKey` 位置，才能调用成功 。

2. 生成代码中的部分字段信息和填写内容是关联的。如需调整传入参数，要在左侧修改参数值后重新生成代码 。

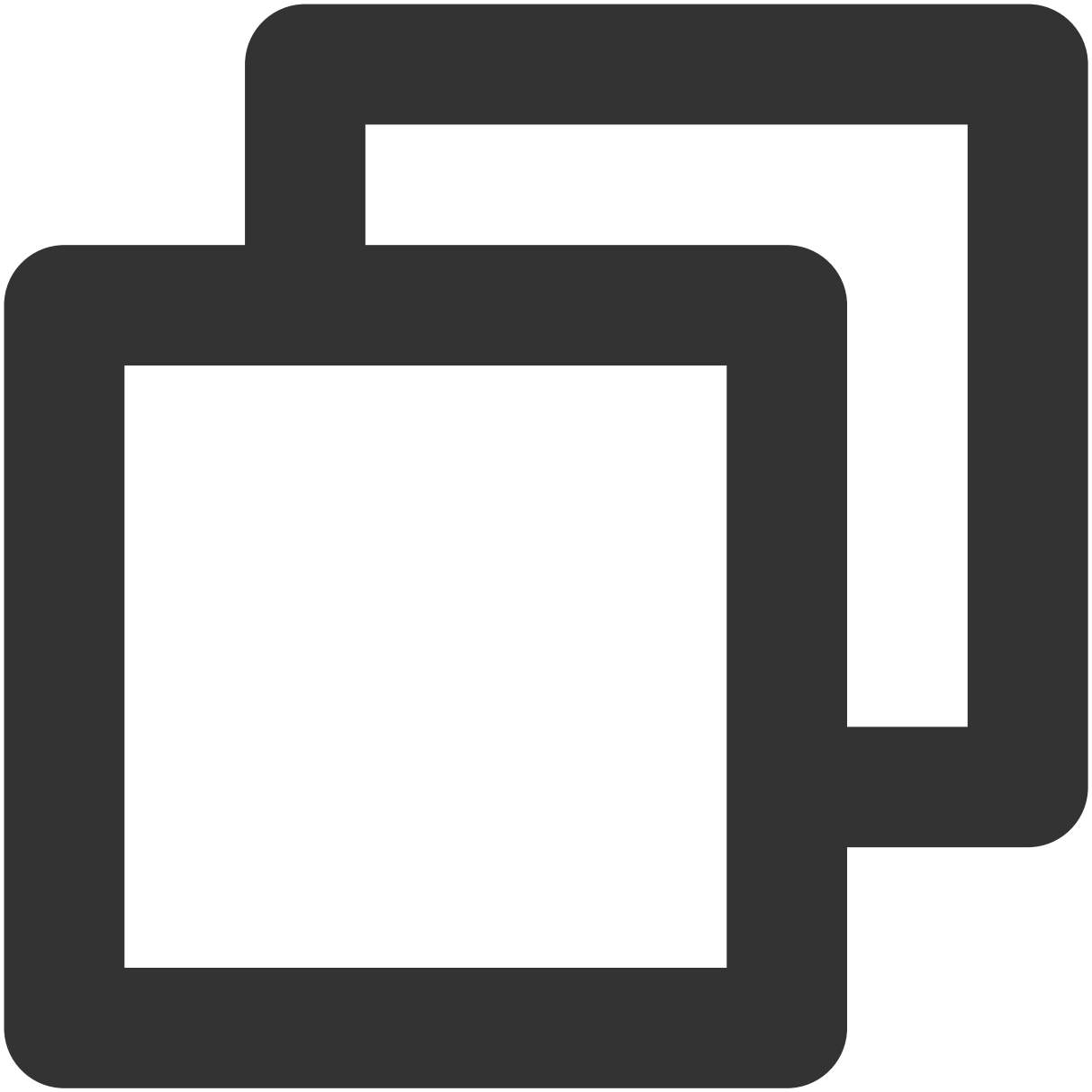


注意事项

SecretId/SecretKey 生成地址： `https://console.tencentcloud.com/cam/capi` 。

当入参中需要传图片/视频 Base64时，需要去掉相关前缀 `` 和换行符 `\\n` 。

如果请求结果提示如下， 需要手动设置签名类型：



```
[TencentCloudSDKException]message:AuthFailure.SignatureFailure-The provided credent
could not be validated because of exceeding request size limit, please use new sign
method `TC3-HMAC-SHA256`. requestId:719970d4-5814-4dd9-9757-a3f11ecc9b20
```

设置签名类型：

```
 `clientProfile.setSignMethod("TC3-HMAC-SHA256"); // 指定签名算法（默认为 HmacSHA256）`
```

连接腾讯云 API 接口

最近更新时间：2023-06-06 16:22:17

腾讯云 API 3.0集成支持多种编程语言的 SDK，让您调用 API 更简单，云 API 支持就近地域接入，本产品就近地域接入域名为 faceid.tencentcloudapi.com ，也支持指定地域域名访问，例如新加坡地域的域名为 faceid.ap-singapore.tencentcloudapi.com。
推荐使用就近地域接入域名。根据调用接口时客户端所在位置，会自动解析到**最近的**某个具体地域的服务器。例如在新加坡发起请求，会自动解析到新加坡的服务器，效果和指定 faceid.ap-singaporef.tencentcloudapi.com 是一致的。
注意：
1. **对时延敏感的业务，建议指定带地域的域名。**
2. **域名是 API 的接入点，并不代表产品或者接口实际提供服务的地域。产品支持的地域列表请在调用方式/公共参数文档中查阅，接口支持的地域请在接口文档输入参数中查阅**
你可以选择你所熟悉的语言进行编码，这部分将使用golang语言作为示例演示如何将腾讯云SDK继承到自己的服务端模块中。
Tencent Cloud SDK 3.0 for Python
Tencent Cloud SDK 3.0 for Java
Tencent Cloud SDK 3.0 for PHP
Tencent Cloud SDK 3.0 for Go
Tencent Cloud SDK 3.0 for NodeJS
Tencent Cloud SDK 3.0 for .NET
Tencent Cloud SDK 3.0 for C++

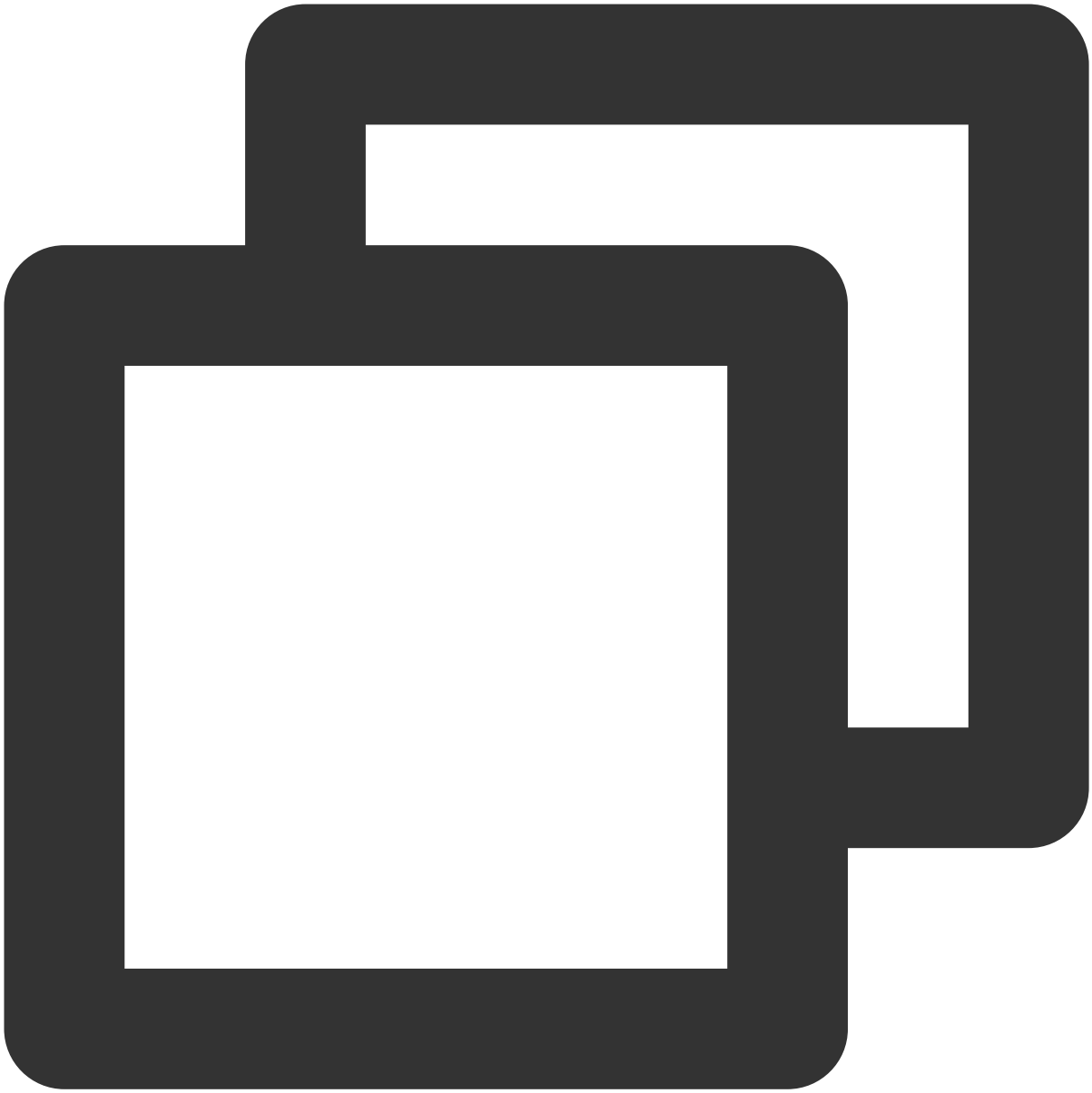
环境依赖

1. Go1.9或者更高版本，并且必须正确设置必要的环境变量，比如GOPATH
2. 使用前请阅读流程指引开通人脸核身产品
3. 按照获取秘钥指引操作获取SecretId和SecretKey

安装

通过 go get 安装（推荐）

推荐使用语言自带的工具安装SDK。



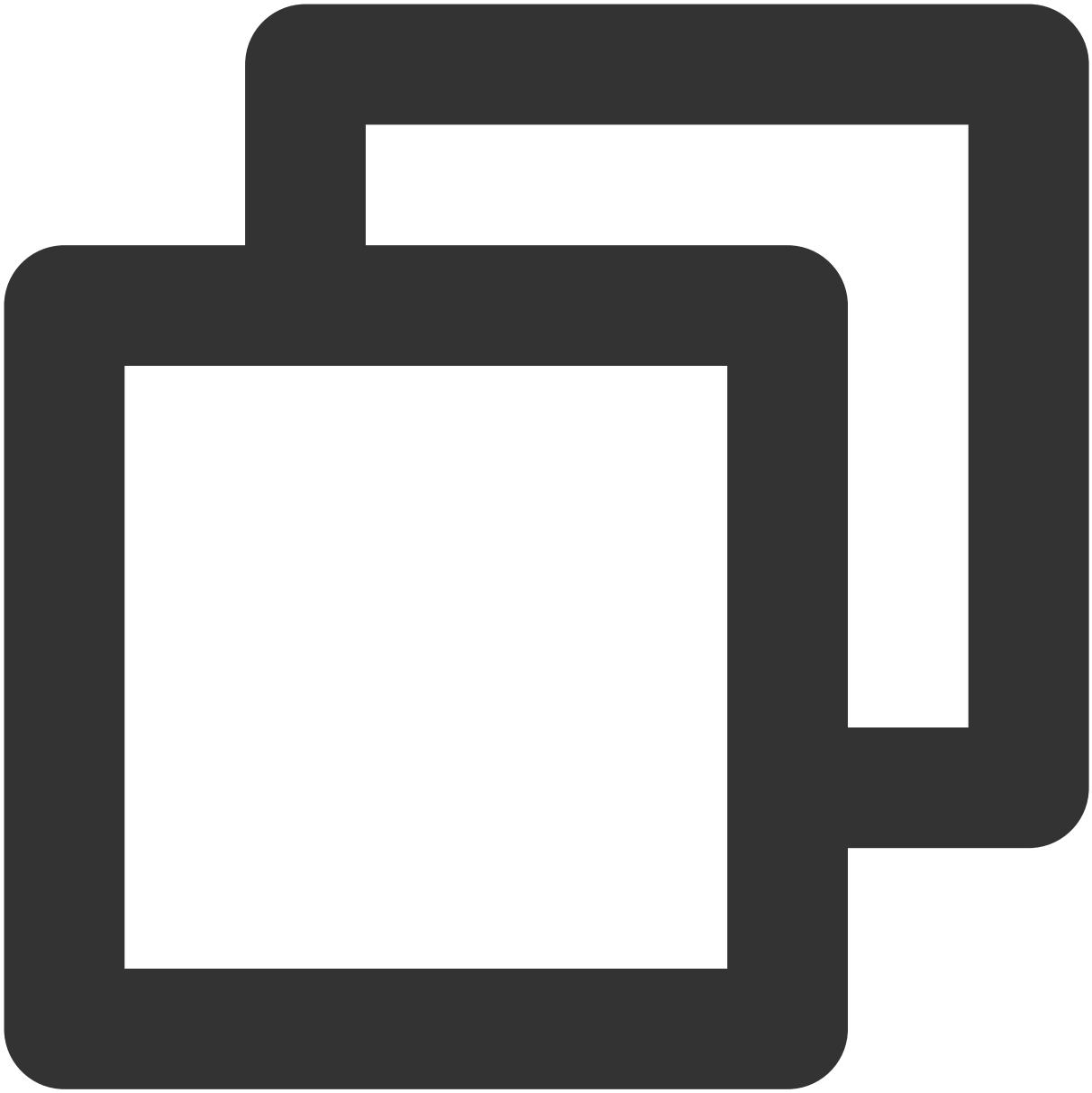
```
go get -u github.com/tencentcloud/tencentcloud-sdk-go-intl-en
```

通过源码安装

到Github代码托管页面下载最新代码，解压安装到$GOPATH/src/github.com/tencentcloud目录下。

集成到服务端

腾讯云SDK安装完成之后，可以通过import指令将SDK集成服务端，示例代码如下：



```
package main

import (
    "fmt"
    cloud "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/profile
    faceid "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/faceid/
)

func main() {
    // Instantiate a client configuration object. You can specify the timeout period
```

```
    prof := profile.NewClientProfile()
    prof.HttpProfile.ReqTimeout = 60
    // TODO replace the SecretId and SecretKey string with the API SecretId and Secr
    credential := cloud.NewCredential("SecretId", "SecretKey")
    // Instantiate the client object of the requested faceid
    FaceIdClient, _ := faceid.NewClient(credential, "ap-singapore", prof)

    // Instantiate the request object and provide necessary parameters
    request := faceid.NewGetFaceIdTokenIntlRequest()
    var SecureLevel = "4"
    request.SecureLevel = &SecureLevel
    // Call the Tencent Cloud API through FaceIdClient
    response, _ := FaceIdClient.GetFaceIdTokenIntl(request)
    // Process the Tencent Cloud API response
    fmt.Println("response: ", *response.Response.SdkToken)
}
```

错误码

最近更新时间：2023-07-03 17:18:38

本文描述腾讯云eKYC产品的所有服务错误码与对应计费关系（是否计费列中“N”代表“否”，“Y”代表“是”）：

公共错误码（所有服务共用）

错误码	描述	是否计费
InvalidParameter	Invalid parameter.	N
InvalidParameterValue.BizTokenIllegal	BizToken illegal.	N
InvalidParameterValue.BizTokenExpired	BizToken expired.	N
UnauthorizedOperation.Nonactivated	The service has not been activated.	N
UnauthorizedOperation.Activating	The service is activating	N
UnauthorizedOperation.ActivateError	The service has not been activated.	N
UnauthorizedOperation.Arrears	The account is in arrears.	N
OperationDenied	Operation denied	N
UnauthorizedOperation.NonAuthorize	Identity verification has not been completed for the account.	N
UnauthorizedOperation	Unauthorized operation.	N
FailedOperation.ImageSizeTooLarge	Image size too large	N
InternalError	Internal error	N

以下是eKYC子产品的错误码列表信息：

活体人脸比对（移动端H5）错误码

错误码	描述	是否计费

0	Success	Y
1	Invalid parameter	N
2	Incorrect parameter value	N
3	Bad request	N
4	Internal service error	N
8	Internal service error	N
9	Internal service error	N
10	Internal service error	N
11	Internal service error	N
14	Verification has been completed	N
15	Token expired	N
16	Retry limit has been reached	N
202	Failed to get video	N
1001	Failed to call the liveness engine	N
1002	Suspected spoofed recording.	N
1003	Real person detection failed	N
1004	Face detection failed	N
1005	Liveness detection failed	N
1101	No sound is detected	N
1102	The face is not fully exposed	N
1103	Speech recognition failed	N
1104	The video format is incorrect	N
1105	Failed to pull the video. Please try again	N
1106	The volume of the video is too low试	N
1107	The video is empty or its size is inappropriate. The recording duration should be about 6 seconds	N

1108	The video definition is too low	N
1109	The lip movement range is too small	N
1201	The lighting is too dim	N
1202	The lighting is too strong	N
1203	The face is too close to the screen	N
1204	The face is too far right from the screen	N
1205	The face is too far from the screen	N
1206	The face is too far left from the screen	N
1207	No motions of eye closing are detected	N
1208	The first motion is not detected	N
1209	No motions of mouth opening are detected	N
1210	Failed to detect a full face	N
1301	Real person detection failed	N
1302	Real person detection did not reach the passing standard	N
1303	The video is too short. Please capture a video longer than 2 seconds	N
2001	Error calling the comparison engine	N
2004	The image passed in is too large or too small	N
2010	Multiple faces are detected	N
2011	Real person comparison failed	N
2012	Failed to detect a full face	N
2013	No faces are detected	N
2014	The resolution of the image passed in is too low. Please upload a new one	N
2015	Comparison failed	N
2016	The comparison similarity did not reach the passing standard	Y
1401	face reflect liveness detect failed	N

活体人脸比对（App SDK）错误码

错误码	描述	是否计费
0	Succeeded	Y
1001	System error	N
1004	Liveness detection and face comparison failed	N
2004	The image passed in is too large or too small	N
2012	Several faces were detected	N
2013	No face was detected, or the face detected was incomplete	N
2014	The image resolution is too low or the quality does not meet the requirements	N
2015	Face comparison failed	N
2016	The similarity did not reach the standard passing threshold	Y
-999	The verification process wasn't finished	N

活体人脸比对（纯API）错误码

错误码	描述	是否计费
Success	Success	Y
FailedOperation.CompareLowSimilarity	The comparison similarity did not reach the passing standard.	Y
FailedOperation.ActionCloseEye	No motions of eye closing are detected.	N
FailedOperation.ActionFaceClose	The face is too close to the screen.	N
FailedOperation.ActionFaceFar	The face is too far from the screen.	N
FailedOperation.ActionFaceLeft	The face is too far left from the screen.	N
FailedOperation.ActionFaceRight	The face is too far right from the screen.	N
FailedOperation.ActionFirstAction	No movement is detected.	N
FailedOperation.ActionLightDark	The lighting is too dim.	N

FailedOperation.ActionLightStrong	The lighting is too strong.	N
FailedOperation.ActionNodetectFace	Failed to detect a full face.	N
FailedOperation.ActionOpenMouth	No motions of mouth opening are detected.	N
FailedOperation.CompareFail	Comparison failed.	N
FailedOperation.CompareSystemError	Error calling the comparison engine API.	N
FailedOperation.DownLoadError	File download failed.	N
FailedOperation.DownLoadTimeoutError	File download timed out.	N
FailedOperation.LifePhotoDetectFaces	Multiple faces are detected.	N
FailedOperation.LifePhotoDetectFake	Real person comparison failed.	N
FailedOperation.LifePhotoDetectNoFaces	Failed to detect a full face.	N
FailedOperation.LifePhotoPoorQuality	The resolution of the image passed in is too low. Please upload a new one.	N
FailedOperation.LifePhotoSizeError	The image passed in is too large or too small.	N
FailedOperation.LipFaceIncomplete	The face is not fully exposed.	N
FailedOperation.LipMoveSmall	The lip movement range is too small.	N
FailedOperation.LipNetFailed	Failed to pull the video. Please try again.	N
FailedOperation.LipSizeError	The video is empty or its size is inappropriate. The recording duration should be about 6 seconds.	N
FailedOperation.LipVideoInvalid	The video format is incorrect.	N
FailedOperation.LipVideoQuaility	The video definition is too low.	N
FailedOperation.LipVoiceDetect	No sound is detected.	N
FailedOperation.LipVoiceLow	The volume of the video is too low.	N
FailedOperation.LipVoiceRecognize	Speech recognition failed.	N
FailedOperation.LivessBestFrameError	Face detection failed. Unable to extract the photo for comparison.	N
FailedOperation.LivessDetectFail	Liveness detection failed.	N

FailedOperation.LivessDetectFake	Suspected spoofed recording.	N
FailedOperation.LivessSystemError	Error calling the liveness engine API.	N
FailedOperation.LivessUnknownError	Video-based real person detection failed.	N
FailedOperation.SilentDetectFail	Real person detection failed.	N
FailedOperation.SilentEyeLiveFail	Eye detection failed.	N
FailedOperation.SilentFaceDetectFail	No face is detected in the video.	N
FailedOperation.SilentFaceQualityFail	Low face quality.	N
FailedOperation.SilentFaceWithMaskFail	A face mask is detected.	N
FailedOperation.SilentMouthLiveFail	Mouth detection failed.	N
FailedOperation.SilentMultiFaceFail	Multiple faces are detected in the video.	N
FailedOperation.SilentPictureLiveFail	The video might be spoofed.	N
FailedOperation.SilentThreshold	Real person detection did not reach the passing standard.	N
FailedOperation.SilentTooShort	The video is too short. Please capture a video longer than 2 seconds.	N
FailedOperation.UnKnown	Unknown internal error.	N
InvalidParameter	Invalid parameter.	N
InvalidParameterValue	Incorrect parameter value.	N
UnauthorizedOperation	Unauthorized operation.	N
UnauthorizedOperation.Arrears	The account is in arrears.	N
UnauthorizedOperation.NonAuthorize	Identity verification has not been completed for the account.	N
UnauthorizedOperation.Nonactivated	The service has not been activated.	N
UnsupportedOperation	Unsupported operation.	N

Identity Verification（App SDK）错误码

错误码	描述	是否计费
0	Success	Y
1001	Failed to call the liveness engine	Y
1004	Face detection failed	Y
2004	The image passed in is too large or too small	Y
2012	Failed to detect a full face	Y
2013	No faces are detected	Y
2014	The resolution of the image passed in is too low. Please upload a new one	Y
2015	Comparison failed	Y
2016	The comparison similarity did not reach the passing standard	Y