

实人认证CLRPSDK_iOS集成文档105

SDK说明

适用版本

本文匹配的 SDK 版本：V1.0.5及以上版本。

使用 Xcode 9.4.1 及以上版本,支持iOS8.0及以上版本

资源文件

CLRPSDK_iOS_v1.X.X压缩文件内目录详情：

CLRPSDKDemo：实人认证Demo代码

framework：是存放sdk的framework和资源文件bundle的

framework --> CLRPSDK.framework：实人认证的framework

framework --> CLRPSDKBundle.bundle：实人认证的资源文件

framework --> IDLFaceSDK.framework：活体检测模块的framework

framework --> com.baidu.idl.face.model.bundle：活体检测模块的资源文件

framework --> com.baidu.idl.face.faceSDK.bundle：活体检测模块的资源文件

framework --> AipBase.framework：身份证OCR模块的framework

framework --> AipOcrSdk.framework：身份证OCR模块的framework

framework --> IdcardQuality.framework：身份证OCR模块的framework

创建应用

- 在 [创蓝万数平台的闪验产品](#) 上创建应用，通过审核后将会得到 `appKey` 和 `appID`












集成实人认证SDK基本步骤：

1. 导入SDK
2. 初始化
3. 调起实人认证并获取认证结果

一. 导入SDK

1: 导入SDK,配置Xcode:

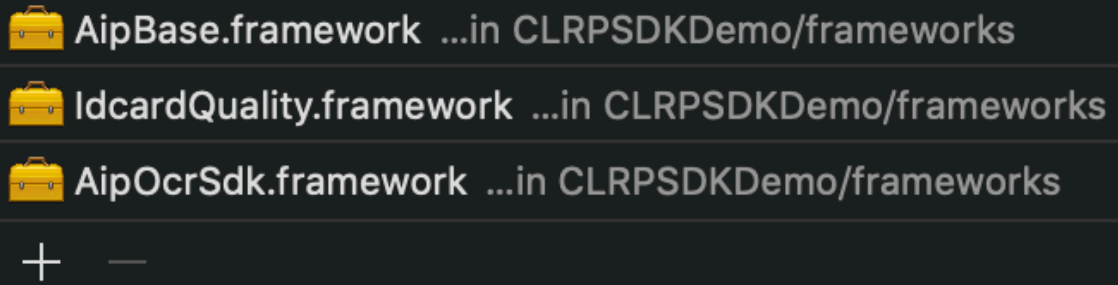
1. 导入framework: 将实人认证SDK压缩包中framework文件夹下所有资源添加到工程中 (注意勾选 `copyItems if need`)
2. Xcode配置:
 - 关闭 `BitCode` : 在 `xcode->BuildSetting->EnableBitcode` 设置为 **NO**
 - 添加 `OtherLinkerFlags` 项 **-ObjC**:
 - 在 `xcode->BuildSetting->Other Linker Flags` 添加 **-ObjC**
 - 添加 `libc++.tdb` :
 - 在 `xcode->General->Linked Frameworks and Libraries` 中点击 + , 搜索并选择添加 **libc++.tdb**
 - 添加 `AVFoundation.framework` :
 - 在 `xcode->General->Linked Frameworks and Libraries` 中点击 + , 搜索并选择添加 **AVFoundation.framework**

▼ Linked Frameworks and Libraries		
Name		Status
 CLRPSDK.framework		Required ⌵
 AipBase.framework		Required ⌵
 AVFoundation.framework		Required ⌵
 AipBase.framework		Required ⌵
 IDLFaceSDK.framework		Required ⌵
 IdcardQuality.framework		Required ⌵
 libc++.tdb		Required ⌵
 libPods-CLRPSDKDemo.a		Required ⌵
 IdcardQuality.framework		Required ⌵
 AipOcrSdk.framework		Required ⌵
 AipOcrSdk.framework		Required ⌵
+ -		

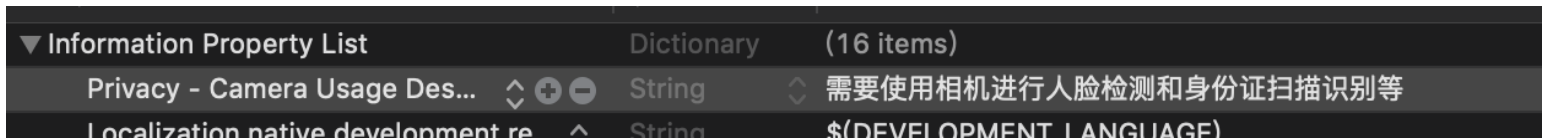
- 添加`Embedded Binary`:

- 在`xcode->General->Embedded Binary`中点击 ****+**** , 将`AipBase.framework`、`AipOcrSdk.framework`、`IdcardQuality.framework`三个框架添加进去

▼ Embedded Binaries



- 添加 Info.plist 使用相机权限：
 - 在项目的Info.plist中添加使用相机权限的申请和描述，如果元工程已经添加，则可以跳过。
Privacy- Camera Usage Description : 需要使用相机进行人脸检测和身份证扫描识别等（根据AppStore审核规则，请不要直接写"使用相机"等类似描述，应填写使用相机的具体用途）



1. Swift工程需要额外添加 `-force_load` :

- 在 `xcode->BuildSetting->Other Linker Flags` 添加 `-force_load`
- 在 `-force_load` 下方添加 `CLRPSDK.framework/CLRPSDK` 所在路径，具体操作可以将 `CLRPSDK.framework` 拖入空栏，在尾部拼接静态库名称 `CLRPSDK`，将前缀绝对地址改成相对地址 `$(SRCROOT)`，最终为 `"$(SRCROOT)/.../CLRPSDK.framework/CLRPSDK"` 形式

2: 导入授权文件:

- 联系客服获取 授权证书文件 ,此文件与 BundleID 绑定，用于提供sdk通讯授权，获取后拖入 sdk的.framework文件同目录下

注: 授权证书有两个，文件名为 `idl-license.face-ios` 和 `aip.license` ,文件名不可修改，文件和 `BundleID` 绑定，修改工程 `BundleID` 后需要重新获取 授权证书文件

二. 初始化

接口名称

```
/**初始化*/
```

```
+ (void)initWithAppId:(NSString *)appId appKey:(NSString *)appKey licenseID:(NSString *)licenseID complete:(CLRPComplete)complete;
```

• 接口参数详细说明

- `appId` : **必填** appId
- `appKey` : **必填** appKey
- `licenseID` : **必填** 授权标识, 与授权证书文件绑定, 可联系客服获取
- `complete` : **选填** 预初始化回调block, 可以在此回调block中接收初始化情况, 成功后会返回 `ticketID`, 也可以不关心初始化结果

接口作用

1. **初始化SDK** :传入用户的appId、appKey, 初始化SDK
2. **获取ticketID** :初始化成功后, SDK回调将返回 `ticketID`, 可以通过 `CLRPCompleteResult.clModel` 属性直接获取, 也可以通过 `CLRPCompleteResult.data` 中取出 `ticket` key对应的值
 - `ticketID` 说明: 执行实人认证流程, 需先生成 `tikerID`, 方便客户查询交易流水记录, 如不需要查询, 则可以忽略 `ticketID` 相关部分。SDK在调起实人认证时会检查 `ticketID`, 如果 `ticketID` 获取失败将直接返回错误, 不会调起实人认证页面。
 - 由于每个流程SDK都会创建新的 `TicketID`, 如果一次启动中多次执行流程或认证流程中断重新开始新的流程会生成多个 `TicketID`, 建议使用通知方式接收 `TicketID`, 通知名称 **NSNOTIF_NAME_CL_DIDGETNEWTICKETID**
 - 通知方式接收TicketID示例代码:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    //初始化  
    [CLRPSDKManager initWithAppId:CLRP_APPID appKey:CLRP_APPKEY licenseID:@"yourLicenseID" complete:nil];  
    //查看TicketID  
    [[NSNotificationCenter defaultCenter] addObserverForName:NSNOTIF_NAME_CL_DIDGETNEWTICKETID object:nil queue:[NSOperationQueue mainQueue] usingBlock:^(NSNotification * _Nonnull note) {  
        NSLog(@"ticket:%@", note.object);  
    }];  
}
```

使用场景

- 建议在app启动时调用
- 必须在实人认证前至少调用一次
- 只需调用一次

请求示例代码

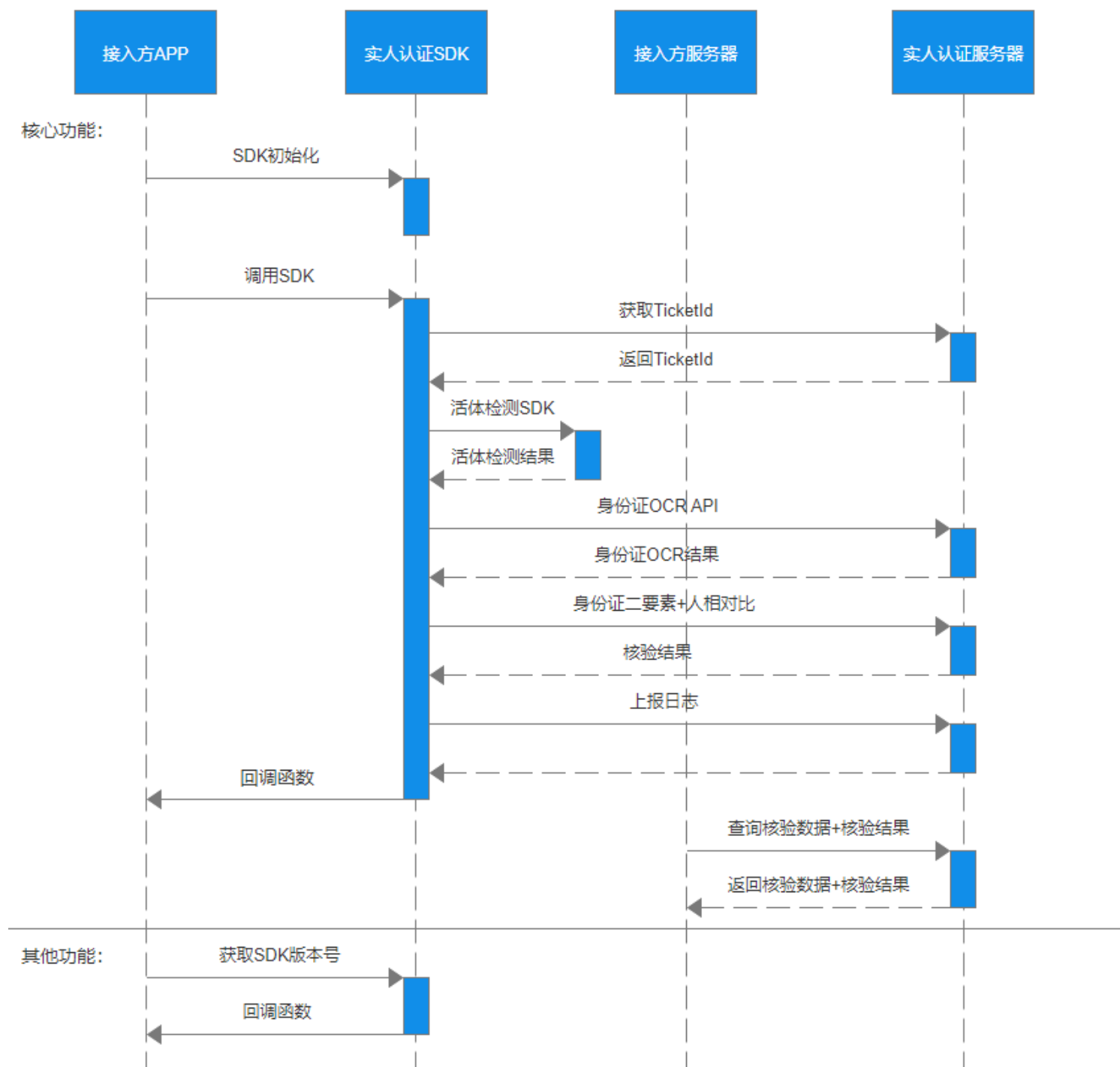
- ObjC:

1. 导入SDK头文件 `#import <CLRPSDK/CLRPSDK.h>`
2. 在AppDelegate中的 `didFinishLaunchingWithOptions` 方法中添加预初始化代码

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    ...  
    //初始化  
    [CLRPSDKManager initWithAppId:<#yourAppID#> appKey:<#yourAppKEY#> licenseID:<#yourLicenseID#> complete:^(CLRPCompleteResult *completeResult) {  
        if (completeResult.error) {  
            NSLog(@"%@",completeResult.error);  
        }else{  
            NSLog(@"initData:%@",completeResult.data);  
            NSLog(@"ticket:%@",completeResult.clModel);  
        }  
    }  
}];  
...
```

三. 调起实人认证

实人认证逻辑说明



接口说明

1. 调起实人认证前的设置接口（可选实现）

- 接口名称

```
+ (void)configure:(CLRPConfigure *)configure;
```

- 接口参数详细说明
 - configure: **必填** 实人认证配置对象

使用场景

- 调起实人认证前配置相关参数
- 需在调用调起实人认证前之前调用
- 如不调用，将使用默认设置：
 - 默认使用传入的vc,以modal方式弹出认证页面
 - 默认从活体检测动作枚举中随机抽取动作和数目检测
 - 默认单个接口超时30s

CLRPConfigure 属性说明:

```
//pushNav
UINavigationController * navigationController;//设置后，使用此nav以push方式打开实人认证页面，完成认证后会从此nav的viewControllers中移除所有实人认证相关页面，如不设置，将使用modal方式

/*****活体检测配置*****/
BOOL isByOrder;//是否按设置的动作顺序执行，默认NO,顺序为随机
NSInteger numOfLiveness;//活体动作数目（liveActionArray为nil时起作用，从所有动作中随机抽取此设置的动作数目）
NSMutableArray *liveActionArray;//活体动作列表数组，eg:@[(CLRPFaceLivenessActionTypeLiveEye),@(CLRPFaceLivenessActionTypeLiveMouth),@(CLRPFaceLivenessActionTypeLiveYaw),@(CLRPFaceLivenessActionTypeLivePitchDown)]

/**单个接口超时设置,单位秒，默认30s
eg CLRPConfigure.requestTimeOut = [NSNumber numberWithFloat:10.0];
*/
NSNumber * requestTimeOut;
```

请求示例代码

1. 导入SDK头文件 `#import <CLRPSDK/CLRPSDK.h>`

2. 在需要使用实人认证的地方调用实人认证设置接口

```
//开发者调用实人认证的方法
- (IBAction)start:(id)sender {
    //实人认证设置
    CLRPConfigure * config = [CLRPConfigure new];
    config.navigationController = self.navigationController;//使用PUSH方式则传入
    当前navigationController, 不传则使用Modal方式
    config.requestTimeout = [NSNumber numberWithInt:25]; //单个请求超时设置
    config.liveActionArray = @[@(CLRPFaceLivenessActionTypeLiveEye),@(CLRPFace
    LivenessActionTypeLiveMouth),@(CLRPFaceLivenessActionTypeLiveYaw),@(CLRPFaceLivene
    ssActionTypeLivePitchDown)].mutableCopy;
    config.isByOrder = YES; //是否按liveActionArray的设置顺序执行动作，默认为NO(随机
    顺序)

    config.numOfLiveness = 1; //设置了liveActionArray后此属性无效
    config.nameOpenEdit = YES; //OCR身份证识别后 姓名可编辑
    config.addressOpenEdit = YES; //OCR身份证识别后 地址可编辑
    config.idCardOpenEdit = YES; //OCR身份证识别后 身份证号码可编辑
    [CLRPSDKManager configure:config];

    [SVProgressHUD setContainerView:self.view];
    [SVProgressHUD show];
    //调用实人认证API，将拉起实人认证页面
    [CLRPSDKManager verify:self complete:^(CLRPCompleteResult *completeResult)
    {
        ...
    }
}
```

2. 调起实人认证接口

- 接口名称

```
+ (void)verify:(UIViewController *)vc complete:(CLRPComplete)complete;
```

- 接口参数详细说明

- vc: **必填** 调起实人认证的ViewController控制器
- complete: **必填** 回调block，用于接收实人认证的结果

使用场景

- 用户进行实人认证操作时，调用实人认证方法，如果初始化获取TicketID成功，SDK将会拉起实人认证页面，流程完成后，SDK将返回实人认证结果
- 可以在多处调用
- 需在调用初始化方法之后调用

请求示例代码

• ObjC:

1. 导入SDK头文件 `#import <CLRPSDK/CLRPSDK.h>`
2. 在需要使用实人认证的地方调用实人认证接口

```
//开发者调用实人认证的方法
- (IBAction)start:(id)sender {
    ...

    //此处建议添加loading
    // 以下情况会出现短暂延迟
    //1: 当初始化获取ticket失败后，调实人认证会尝试再次获取，此时若网络环境差会出现短暂延迟
    //2: 当实人认证流程结束后，会重新生成ticket供下次使用，若生成ticket失败，调实人认证
    会尝试再次获取，此时若网络环境差会出现短暂延迟
    [SVProgressHUD setContainerView:self.view];
    [SVProgressHUD show];

    //调用实人认证API，将拉起实人认证页面
    [CLRPSDKManager verify:self complete:^(CLRPCompleteResult *completeResult)
    {
        dispatch_async(dispatch_get_main_queue(), ^{

            [SVProgressHUD dismiss];

            //如需跳转，此处为开发者自己的跳转页
            VerifyResultViewController * vc = [[VerifyResultViewController all
            oc]init];

            //身份证信息从completeResult.clModel属性获取，为CLRP0crIDCardModel类，
            即使失败，如果身份证OCR步骤成功依然可以获取到身份证信息
            CLRP0crIDCardModel * iDCardModel = completeResult.clModel;
            if (iDCardModel) {
                NSString * name = iDCardModel.name;
                NSString * address = iDCardModel.address;
                NSString * birth = iDCardModel.birth;
                NSString * cardNum = iDCardModel.cardNum;
```

```
NSString * sex = iDCardModel.sex;
NSString * nation = iDCardModel.nation;
NSString * issuingAuthority = iDCardModel.issuingAuthority;
NSString * issuingDate = iDCardModel.issuingDate;
NSString * expiryDate = iDCardModel.expiryDate;
```

```
NSString * riskType = iDCardModel.riskType;
```

```
NSLog(@"\n姓名: %@\n身份证号: %@\n性别: %@\n出生日期: %@\n名族: %@\n地址: %@\n签发机关: %@\n有效期限: %@~%@ \n身份证风险提示类型: %@", name, cardNum, sex, birth, nation, address, issuingAuthority, issuingDate, expiryDate, riskType); //认证分数从completeResult.clValue属性获取, 建议分值大于80分为成功
```

```
if (completeResult.error == nil && completeResult.clValue >= 80) {
    vc.isVerifySuccess = YES;
```

```
//如需获取实人认证过程中的照片, 可通过以下方式
```

```
NSDictionary * livenessImages = completeResult.data;
```

```
//身份证正面
```

```
UIImage * CLRPIDCard_Front = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPIDCard_Front"]];
```

```
//身份证反面
```

```
UIImage * CLRPIDCard_Back = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPIDCard_Back"]];
```

```
//活体检测最优照片
```

```
UIImage * livenessBestimage = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLiveness"]];
```

```
//活体检测动作照片 (如设置了活体动作, 将只会取到相应动作的照片, 如未设置, 将取出活体中检测中随机检测的动作的照片) :
```

```
UIImage * livenessLiveEye = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLiveEye"]];
```

```
UIImage * livenessLiveMouth = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLiveMouth"]];
```

```
UIImage * livenessLiveYawRight = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLiveYawRight"]];
```

```
UIImage * livenessLiveYawLeft = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLiveYawLeft"]];
```

```
UIImage * livenessLivePitchUp = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLivePitchUp"]];
```

```
UIImage * livenessLivePitchDown = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLivePitchDown"]];
```

```
UIImage * livenessLiveYaw = [UIImage imageWithContentsOfFile:livenessImages[@"CLRPFaceLivenessActionTypeLiveYaw"]];
```

```
        NSLog(@"%@",completeResult);
        NSLog(@"%@",completeResult.data);

    } else if (completeResult.code == 1005) {
        NSLog(@"身份证正面认证失败");
        return;
    } else if (completeResult.code == 1006) {
        NSLog(@"身份证反面认证失败");
        return;
    } else if (completeResult.code == 1007) {
        NSLog(@"人证核验失败");
    } else if (completeResult.code == 1008) {
        NSLog(@"身份证二要素验证失败");
    } else if (completeResult.code == 1104) {
        NSLog(@"身份证图像信息识别回调");
        return ;
    } else if (completeResult.code == 1105) {
        NSLog(@"身份证正面认证成功");
        return;
    } else if (completeResult.code == 1106) {
        NSLog(@"身份证反面认证成功");
        return;
    } else if (completeResult.code == 1107) {
        NSLog(@"人证核验成功");
        return;
    } else if (completeResult.code == 1108) {
        NSLog(@"身份证二要素校验成功");
        return;
    } else {

        if (completeResult.code == 1010) {
            //用户点击返回
            return ;
        }

        vc.isVerifySuccess = NO;
        vc.verifyMessage = completeResult.message;
    }

    //清除缓存（清除后将不能取到活体检测照片等信息，建议在最后调用）
    [CLRPSDKManager cleanTempCache];

    [self.navigationController pushViewController:vc animated:YES];
```

```
    });  
  }];  
}
```

-verify:complete:正确返回示例

`completeResult->_error` `error` 为空则为成功

`completeResult->_clValue`：认证分数(`int`)，`0~100`，建议`80`分以上可认为成功

`completeResult->_clModel`：身份证识别文本信息，`CLRPOcrIDCardModel*` 类型实例，即使认证失败，如果身份证OCR步骤成功依然可以获取到身份证信息

`completeResult->_data`： 实人认证过程中的照片保存路径(包括身份证正反面，活体检测最优照片，活体检测动作照片)

- 照片取值对应key:

- 身份证正面：`CLRPIDCard_Front`

- 身份证反面：`CLRPIDCard_Back`

- 活体检测最优照片：`CLRPFaceLiveness`

- 活体检测动作照片（如设置了活体动作，将只会取到相应动作的照片，如未设置，将取出活体中检测中随机检测的的动作的照片）：

- 眨眼：`CLRPFaceLivenessActionTypeLiveEye`

- 张嘴：`CLRPFaceLivenessActionTypeLiveMouth`

- 向右转头：`CLRPFaceLivenessActionTypeLiveYawRight`

- 向左转头：`CLRPFaceLivenessActionTypeLiveYawLeft`

- 抬头：`CLRPFaceLivenessActionTypeLivePitchUp`

- 低头：`CLRPFaceLivenessActionTypeLivePitchDown`

- 左右摇头：`CLRPFaceLivenessActionTypeLiveYaw`

-verify:complete:报错处理

* `completeResult->_error` `error` 不为空则为报错

- 1010，用户点击返回，此时已经返回到弹出实人认证前的用户页面，建议直接跳过

- 1101，网络测异常，请检查网络

- 1102，请检查实人认证时的环境光线等，确保摄像头中视频流清晰；请确保摄像头正对身份证，身份证边框完整，字体清晰，亮度均匀，无反光

- 1103，其他错误具体查看错误描述

- 1104，身份证正反面拍摄后，信息识别出来后的回调，将识别出的信息返回给用户（此处并不是整个实人认证流程结束，只作为识别）

- 1105，身份证二要素不匹配

- 1106，身份证反面认证成功

- 1107，人证合验成功

如何上传到AppStore和支持模拟器运行的说明

本SDK包中有个别framework使用动态库（AipBase.framework、AipOcrSdk.framework、IdcardQuality.framework。为了支持模拟器正常运行，其中AipBase.framework、AipOcrSdk.framework有包含模拟器架构和进包涵真机架构两个包），使用动态库有诸多优势，但若在动态库包含多个架构，在上传AppStore前需要删除模拟器架构

支持AppStore但不支持模拟器方式：

sdk的framework文件夹中默认使用的AipBase.framework、AipOcrSdk.framework是不支持模拟器但支持AppStore的版本，可以直接使用

支持模拟器但不支持AppStore方式：

用仅包含真机架构的AipBase.framework、AipOcrSdk.framework替换

最终打包上传Appstore时必需使用已经移除模拟器架构的AipBase.framework和AipOcrSdk.framework

SDK 返回码对照表

返回码	返回码描述
1000	成功
信息回调返回码	返回码描述
1005	身份证正面认证失败
1006	身份证反面认证失败
1007	人证合验失败
1008	身份证二要素验证失败
1010	用户点击返回
1101	网络测异常
1102	业务接口报错

1103	其他错误
1104	身份证信息识别回调
1105	身份证正面认证成功
1106	身份证反面认证成功
1107	人证合验成功
1108	身份证二要素校验成功
已变更返回码	原返回码描述
1105	身份证信息不匹配(已变更为 身份证正面认证失败)

常见问题

- 身份证扫描页面提示无权限：
 - 请确保手机网络正常，返回后重新进入页面
 - 请确保授 `权证书文件` 和工程对应的BundleID一致

- 编译报

```
ld: 38 duplicate symbols for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

- 解决方案： `OtherLinkerFlags` 项去掉 `-all_load` 添加 `-force_load` 并添加 framework 路径，参考 [导入SDK,配置Xcode->setup3](#)

- 运行报

```
Terminating app due to uncaught exception 'NSInvalidArgumentException',
reason: '-[xxxx xxx]: unrecognized selector sent to instance
```

错误

- 解决方案：将此错误对应的.framework添加进 `-force_load` ： `OtherLinkerFlags` 项 `-force_load` 并添加framework路径，参考 [导入SDK,配置Xcode->setup3](#)

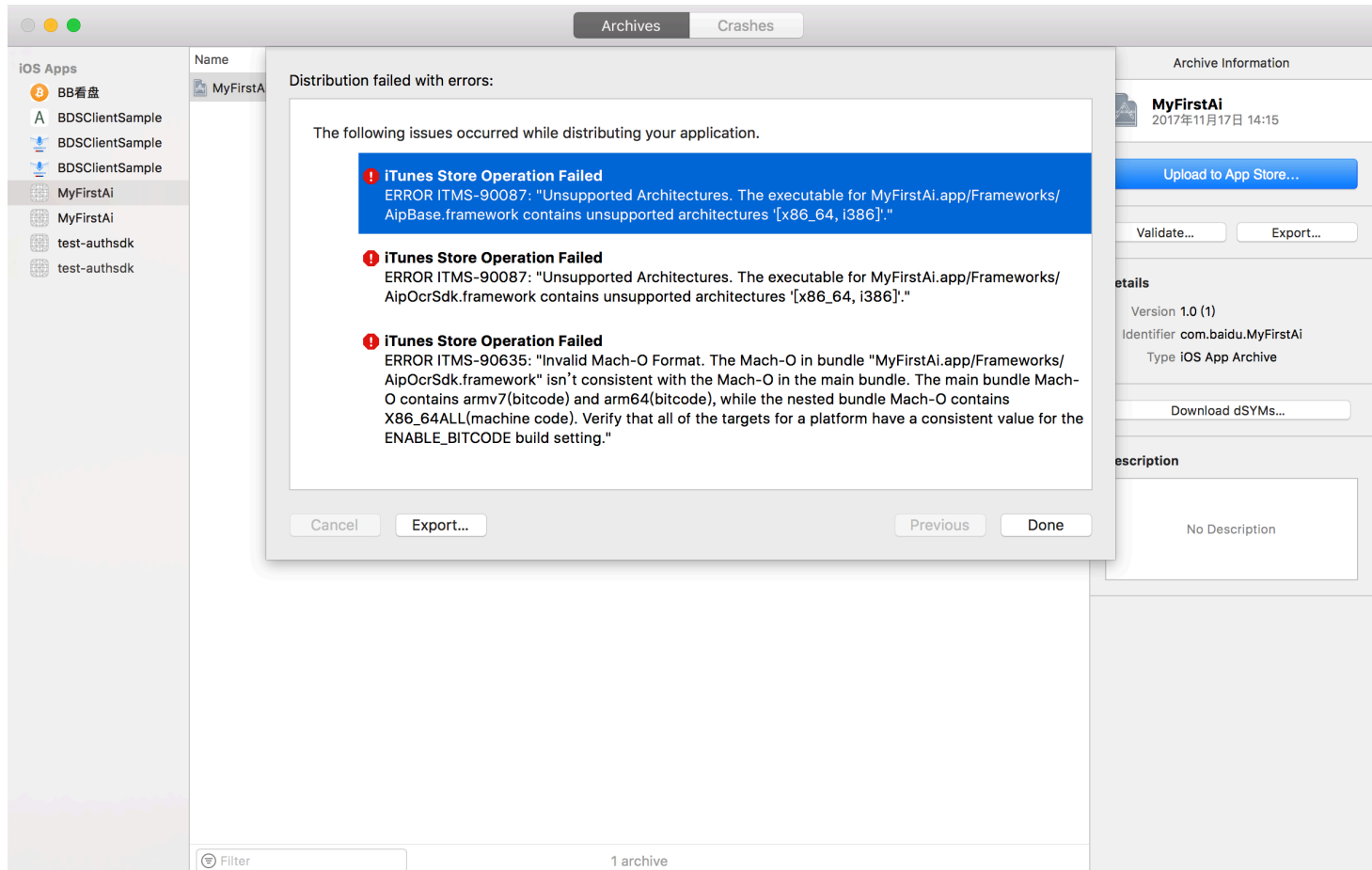
- 报错 App identifier unmatched. 错误的BundleId或PackageName，如何解决？

- 请确保授 `权证书文件` 和工程对应的BundleID一致
 - 请联系客服重新获取

- 直接运行无问题，但 `Archive/IPA/Upload AppStore` 时报错"Unsupported Architecture.

Your executable contains unsupported architecture '[x86_64, i386]...'

ERROR ITMS-90087: "Unsupported Architectures. The executable for MyFirstAi.app/Frameworks/AipBase.framework contains unsupported architectures '[x86_64, i386]'."



- 为了方便开发者调试，AipBase.framework和AipOcrSdk.framework合并了模拟器和真机架构，上线前，使用lipo工具移除相关架构即可，参考：
- 开发者可以使用一下命令手动移除，SDK包也会打包一个已经移除了的这两个framework，可以直接替换

```
cd lib
# 使用lipo -info 可以查看包含的架构
lipo -info AipBase.framework/AipBase # Architectures in the fat file: AipBase
are: i386 x86_64 armv7 armv7s arm64
# 移除x86_64, i386
lipo -remove x86_64 AipBase.framework/AipBase -o AipBase.framework/AipBase
lipo -remove i386 AipBase.framework/AipBase -o AipBase.framework/AipBase
lipo -remove x86_64 AipOcrSdk.framework/AipOcrSdk -o AipOcrSdk.framework/AipOcrSdk
```

```
lipo -remove i386 Aip0crSdk.framework/Aip0crSdk -o Aip0crSdk.framework/Aip0crSdk
# 再次查看
lipo -info AipBase.framework/AipBase # Architectures in the fat file: AipBase
are: armv7 armv7s arm64
lipo -info Aip0crSdk.framework/Aip0crSdk # Architectures in the fat file: AipBase
are: armv7 arm64
```

```
➤ lib lipo -info AipBase.framework/AipBase
Architectures in the fat file: AipBase.framework/AipBase are: i386 x86_64 armv7 arm64
➤ lib lipo -remove x86_64 AipBase.framework/AipBase -o AipBase.framework/AipBase
➤ lib lipo -remove i386 AipBase.framework/AipBase -o AipBase.framework/AipBase
➤ lib lipo -remove x86_64 Aip0crSdk.framework/Aip0crSdk -o Aip0crSdk.framework/Aip0crSdk
➤ lib lipo -remove i386 Aip0crSdk.framework/Aip0crSdk -o Aip0crSdk.framework/Aip0crSdk
```