顶象无感验证SDK ReactNative for iOS 使用说明

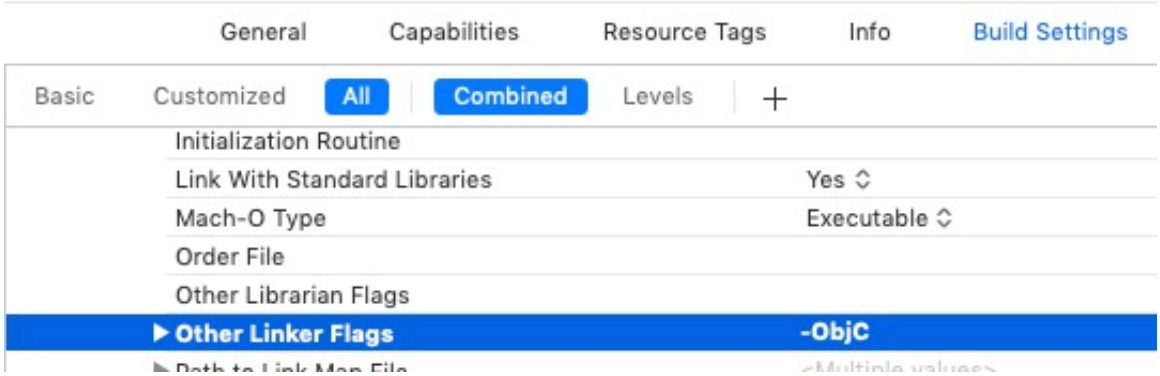
- 访问顶象官网，注册账号后登录控制台，访问“无感验证”模块，申请开通后系统会分配一个唯一的AppId、AppSecret。
- 兼容性：iOS8.0以上

一、集成SDK

- 导入依赖，集成无感验证，需要同时集成风控SDK（如项目本身需集成顶象风控SDK则直接参照风控SDK集成说明，跳过此步），无感验证版本1.1以上须对应集成4.2或以上版本的风控SDK，下载顶象风控 dx-risk SDK ，解压得以下几个文件
- dx-risk-iOS-x.x.x-xxxxxxx目录 DXRisk sdk
  - DXRisk.framework 不带idfa获取逻辑的Dynamic Library Framework
  - DXRiskWithIDFA.framework 带idfa获取逻辑的Dynamic Library Framework
  - DXRiskStatic.framework 不带idfa获取逻辑的Static Library Framework
  - DXRiskStaticWithIDFA.framework 带idfa获取逻辑的Static Library Framework

  若在项目中添加`DXRisk.framework`或者`DXRiskWithIDFA.framework`其中之一，选择`Target` -> `General`，在`Frameworks,Libraries,and Embedded Content`中，将`DXRisk.framework`或者`DXRiskWithIDFA.framework` 对应的 Embed 切换到Embed & Sign。如下图：

  若在项目中添加`DXRiskStatic.framework`或者`DXRiskStaticWithIDFA.framework`其中之一，需要在Build Settings -> Other Linker Flags 设置 -ObjC 如下图：



- 下载顶象无感验证 dx-captcha SDK ，解压得到 `DXCaptchaSDK` 文件夹，内含以下几个文件

- DXCaptchaSDK CaptchaFramework
  - `DingxiangCaptcha.bundle` 无感验证资源包
  - `DingxiangCaptchaSDK.framework` Dynamic Library Framework
  - `DingxiangCaptchaSDKStatic.framework` Static Library Framework
  - `DXCaptchaDemo` 集成demo

  将 `DingxiangCaptchaSDK.framework` 或者 `DingxiangCaptchaSDKStatic.framework` 文件夹拖入工程根目录，若添加`DingxiangCaptchaSDK.framework` 则在项目中添加`DingxiangCaptchaSDK.framework`，选择`Target` -> `General`，在`Embedded Binaries`里点击加号，添加`DingxiangCaptchaSDK.framework`，并添加`DingxiangCaptcha.bundle` 相应的资源至项目中
- 在 `Build Phases` -> `Link Binary With Libraries` 里点击加号，添加 `libc++.tbd` 、`libresolv.tbd` 和 `libz.tbd` 库
- 在项目中添加Linking配置，选择`Target` -> `Build Settings`，在`Other Linker Flags`里添加`-ObjC`配置

- 配置打包脚本

  以下的操作仅限导入DingxiangCaptchaSDK.framework动态库

此步骤主要是解决上传Store架构不符合的问题，如项目中已配置过Carthage或有其他相关的打包Framework调整脚本，可略过此步自行调整 选择`Target` -> `Build Phases`，点击`+`按钮，选择`New Run Script Phase`，添加如下脚本：

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

EXTRACTED_ARCHS=()

for ARCH in $ARCHS
do
echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o
"$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

echo "Merging extracted architectures: ${ARCHS}"
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
```

```
rm "${EXTRACTED_ARCHS[@]}"

echo "Replacing original executable with thinned version"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```

二、初始化

假设在 `ViewController` 中添加无感验证，首先引入头文件

```
#import "DXCaptchaDelegate.h"
#import "DXCaptchaView.h"
#import <React/RCTBundleURLProvider.h>
#import <React/RCTRootView.h>
#import <Foundation/Foundation.h>
#import <React/RCTBridgeModule.h>
#import <React/RCTEventDispatcher.h>
#import <React/RCTEventEmitter.h>
```

然后实现 `DXCaptchaDelegate` 协议中的 `captchaView:didReceiveEvent:arg:` 方法，以接收验证结果回调

```
@interface ViewController () <DXCaptchaDelegate,
RCTBridgeDelegate,RCTBridgeModule>

@end

@implementation ViewController

- (void) captchaView:(DXCaptchaView *)view didReceiveEvent:
(DXCaptchaEventType)eventType arg:(NSDictionary *)dict {
    switch(eventType) {
        case DXCaptchaEventSuccess: { // 验证成功的回调
            NSString *token = dict[@"token"]; // 成功时会传入token参数
            [[[UIAlertView alloc]initWithTitle:@"Verify Success" message:token
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil] show];

            //以下是回调RN 接口，需要自己定义
             [[[MallManageAPI alloc]init]
CollectionProductSuccessForRNAction:str];
            break;
        }
        case DXCaptchaEventFail: // 验证失败的回调
            [[[UIAlertView alloc]initWithTitle:@"Verify Failed"
message:@"DXCaptcha Verify Failed!" delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil] show];
```

```
                        //以下是回调RN 接口，需要自己定义
             [[[MallManageAPI alloc]init]
CollectionProductSuccessForRNAction:@"error"];
             break;
         default:
             break;
    }
}

// RN 导出符号
RCT_EXPORT_MODULE();

// RN -> Native 接口调用
RCT_EXPORT_METHOD(RNTransferIOSWithCallBack:(RCTResponseSenderBlock)callback)
{
//该方法定义弹出 无感验证框
  [self login:callback];
  }

@end
```

无感验证框弹出如下代码：

```
objective-c
- (void) login :(RCTResponseSenderBlock)callback{
  if ([NSThread isMainThread])
  {
    CGRect frame = CGRectMake(self.view.center.x - 150, self.view.center.y -
100, 300, 200);
    _captchaView = [[DXCaptchaView alloc]
initWithAppId:@"xxxxxxxxxxxxxxxxxxxxxxxx" delegate:self frame:frame];
    [[UIApplication sharedApplication].keyWindow addSubview:
     _captchaView];
  }
  else
  {
    dispatch_sync(dispatch_get_main_queue(), ^{
      //Update UI in UI thread here
      CGRect frame = CGRectMake(self.view.center.x - 150, self.view.center.y -
100, 300, 200);
      _captchaView = [[DXCaptchaView alloc]
initWithAppId:@"xxxxxxxxxxxxxxxxxxxxxxxx" delegate:self frame:frame];
      [[UIApplication sharedApplication].keyWindow addSubview:_captchaView];
    });
  }
}
```

创建无感验证组件，需要传入 `appId`， `appSecret`， `delegate`(验证回调的代理)， `frame`(位置及尺寸)

```
objective-c
CGRect frame = CGRectMake(self.view.center.x - 150, self.view.center.y - 100,
300, 200);
DXCaptchaView *captchaView = [[DXCaptchaView alloc] initWithAppId:@"your
appId" appSecret:@"your appSecret" delegate:self frame:frame];
[self.view addSubview:captchaView];
```

如果除 `appId` `appSecret` 以外还需要配置其他的初始化参数，可调用 `initWithConfig:delegate:frame` 接口，如

```
NSDictionary *config = @{@"appId": @"your appId",@"appSecret": @"your
appSecret", @"language": @"en"};
DXCaptchaView *captchaView = [[DXCaptchaView alloc] initWithConfig:config
delegate:self frame:frame];
```

以下是 上述Native 回调 RN的例子

```
MallManageAPI.h

#import <Foundation/Foundation.h>
#import <React/RCTEventEmitter.h>
#import <React/RCTBridgeModule.h>
#import <React/RCTRootView.h>

@interface MallManageAPI  : RCTEventEmitter <RCTBridgeModule>
- (void)CollectionProductSuccessForRNAction:(NSString*) data;
@end

MallManageAPI.m

#import "MallManageAPI.h"

@implementation MallManageAPI

RCT_EXPORT_MODULE();

-(NSArray<NSString *> *) supportedEvents {
//此处 EventReminder_DX 需要定义上传RN 层的管道
  return @[@"EventReminder_DX"];
}


+(id)allocWithZone:(NSZone *)zone {
  static MallManageAPI *sharedInstance = nil;
  static dispatch_once_t onceToken;
```

```
  dispatch_once(&onceToken, ^{
    sharedInstance = [super allocWithZone:zone];
  });
  return sharedInstance;
}

- (instancetype)init {
  if (self = [super init]) {
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(CollectionProductSuccessForRNAction:)
name:@"CollectionProductSuccessForRN" object:nil];
  }
  return self;
}


- (void)CollectionProductSuccessForRNAction:(NSString*) data{
     [self sendEventWithName:@"EventReminder_DX" body:data];
}

- (void)dealloc {
  [[NSNotificationCenter defaultCenter] removeObserver:self
name:@"CollectionProductSuccessForRN" object:nil];
}

- (dispatch_queue_t)methodQueue {
  return dispatch_get_main_queue();
}

@end
```

在RN js层需要定义如App.js 中需要定义引入

```
import {NativeModules,NativeEventEmitter} from 'react-native'
```

以及传入 Native 层事件的管道

```
  const ViewController = NativeModules.ViewController; // ViewController为自定义
接受类
     ViewController.RNTransferIOSWithCallBack((data) => {
    });
```

添加监听事件

```
   componentWillMount() {
```

```
          var NativeModulesByIOS = NativeModules.MallManageAPI; // 此处是作为接受
native 管道方法处理
          var emitter = new NativeEventEmitter(NativeModulesByIOS);
          subscription = emitter.addListener(
                    'EventReminder_CollectionProductSuccessForRN',
                     (body) => {
                      console.log('通知信息:'+body);
                      //以下是接受进行处理 Token
                     Alert.alert("tips",body,
                           [{text: 'ask me later'},{text: '取消', onPress:
this.userCanceled},{text: '确定', onPress: this.userConfirmed}
                           ],
                       {cancelable: true}
                     );

                     }
              );
      }


    componentWillUnmount(){
      subscription.remove();
    }
```

参数说明

初始化有若干参数，详情可查阅 前端接入

其中在 `iOS SDK` 中

- `style` 固定为 `embed` ，若指定为其他值会被忽略
- `width` 参数会被忽略，实际值会根据 `frame` 进行自适应调整，建议尺寸为`300x200`
- `success` 和 `fail` 回调参数会被忽略，请在 `delegate` 中处理回调
- `constID_js` 、 `constIDServer` 和 `constID_options` 配置不生效
- 增加 `userId` 参数，用于传递给 `ConstID` 模块作用户标识

事件说明

回调的事件类型，请查阅 事件 及 `DXCaptchaDelegate.h` 头文件

注:

- `iOS SDK` 中不触发 `show/hide` 系列的事件
- 如SDK无效且运行过程中出现`NOT FOUND DXRISK-SDK`日志，请检查是否遗漏顶象风控SDK的集成