

- title: iOS 开发文档
type: iOS 开发文档
order: 0
- 概述及资源
 - 1、环境需求
 - 2、相关开发资料
- 准备工作
 - 1、创建应用
 - 2、快速体验 Demo
 - 3、开发环境搭建
 - 3.1、CocoaPods 集成
 - 3.2、手动集成
 - 3.3、工程配置
- OneLogin (一键登录)
 - 1、进阶模式
 - 1.1、调用逻辑
 - 1.2、初始化
 - 1.3、进入授权页面
 - 1.4、手动关闭授权页面
 - 1.5、重新开始预取号
 - 1.6、判断预取号结果是否有效
 - 1.7、设置请求超时时间
 - 1.8、改变隐私条款勾选框状态
 - 1.9、开始取号
 - 2、常规模式
- 授权页面 UI 修改
 - 1、设计规范
 - 2、页面控件位置大小设置
 - 2.1、设置控件的 OLRect 属性
 - 2.2、控件进行自动布局
 - 3、页面控件属性设置
 - 4、横竖屏设置
 - 5、弹窗模式设置
 - 6、授权页面中添加自定义控件
 - 7、自定义授权页面 loading
 - 8、设置服务条款
 - 9、设置授权页面弹出模式

- 10、设置状态栏
 - 11、授权页在服务条款未选中时的弹窗
 - 12、自定义授权弹窗
 - 其他接口说明
 - 1、设置日志开关
 - 2、获取当前网络信息和运营商信息
 - 3、设置超时时长
 - 4、服务条款复选框是否勾选
 - 5、获取SDK版本号
 - 6、获取当前授权页面
 - 7、回调接口
 - 7.1、加载授权页面进度条回调
 - 7.2、停止授权页面进度条回调
 - 7.3、授权页面视图生命周期回调
 - 7.4、监听授权按钮点击的回调
 - 7.5、点击授权页面勾选框的回调
 - 7.6、点击授权页面弹窗背景的回调
 - 7.7、授权页面旋转时的回调
 - 7.8、接管授权按钮点击事件的回调
 - 7.9、点击切换账号按钮的回调
 - 7.10、点击运营商隐私协议的回调
 - OnePass(本机号码认证)
 - 1、初始化
 - 2、本机号码认证
 - 3、配置是否缓存手机号到本地
 - 4、获取缓存在本地的手机号
 - 5、获取缓存在本地的手机号列表
 - 插件资源
 - Flutter 插件
 - 其他插件
-

title: iOS 开发文档
type: iOS 开发文档
order: 0

概述及资源

本文是 OneLogin iOS SDK 的部署文档，用于指导 OneLogin iOS SDK 的集成，读者需具有一定 iOS 编程知识基础。

1、环境需求

条目	资源
开发目标	iOS 11+
开发环境	Xcode 14+
系统依赖	<code>libc++.1.tbd</code> 、 <code>libz.1.2.8.tbd</code> ，iOS12 以下系统依赖 <code>Network.framework</code>
SDK 三方依赖	<code>EAccountApiSDK.xcframework</code> 、 <code>TYRZUISDK.xcframework</code> 、 <code>OAuth.xcframework</code>
包增量	1M
网络制式	移动 2G/3G/4G/5G，联通 3G/4G/5G，电信 4G/5G（2G/3G 网络下时延相对较高，成功率相对较低）
网络环境	打开蜂窝数据流量并且给予应用蜂窝数据权限

2、相关开发资料

条目	资源
产品结构流程	交互流程 ， 通讯流程
常见问题	常见问题
XCFramework SDK 资源包	点击下载

准备工作

1、创建应用

登录后台创建应用获取 APPID 和 Key，具体步骤可参照[账号创建](#)。

2、快速体验 Demo

iOS 压缩包附带的 OneLoginExample 文件夹中是 的示例工程，使用 Xcode 打开示例工程，修改 BundleID 和 AppId 为创建应用时绑定的 BundleID 和 AppId。

若体验 OneLogin（一键登录），可完成以下配置步骤进行本地测试：

1. 修改 bundleId
2. OC 版本，请修改 BaseViewController.h 文件中的 GTOneLoginAppId，Swift 版本，请修改 BaseViewController.swift 文件中的 GTOneLoginAppId
3. OC 版本，请修改 BaseViewController.h 文件中的 GTOneLoginResultURL，Swift 版本，请修改 BaseViewController.swift 文件中的 GTOneLoginResultURL
4. 参照服务端接入文档完成服务端接口的对接(该接口即为步骤 3 中的GTOneLoginResultURL)，**注意客户端 AppID 需与服务端保持一致**

ObjC:

```
// OneLogin
#define GTOneLoginAppId @"b41a959b5cac4dd1277183e074630945"
#define GTOneLoginResultURL @"http://onepass.geetest.com/onelogin/result"
```

Swift:

```
// OneLogin
let GTOneLoginAppId = "b41a959b5cac4dd1277183e074630945"
let GTOneLoginResultURL = "http://onepass.geetest.com/onelogin/result"
```

若体验 OnePass（本机号码认证），，可完成以下配置步骤进行本地测试：

1. 修改 bundleId
2. OC 版本，请修改 BaseViewController.h 文件中的 GTOnePassAppId，Swift 版本，请修改 BaseViewController.swift 文件中的 GTOnePassAppId
3. OC 版本，请修改 BaseViewController.h 文件中的 GTOnePassVerifyURL，Swift 版本，请修改 BaseViewController.swift 文件中的 GTOnePassVerifyURL
4. 参照服务端接入文档完成服务端接口的对接(该接口即为步骤 3 中的GTOnePassVerifyURL)，**注意客户端 AppID 需与服务端保持一致**

ObjC:

```
// OnePass
#define GTOnePassAppId @"3996159873d7ccc36f25803b88dda97a"
#define GTOnePassVerifyURL @"http://onepass.geetest.com/v2.0/result"
```

Swift:

```
// OnePass
let GTOnePassAppId = "3996159873d7ccc36f25803b88dda97a"
let GTOnePassVerifyURL = "http://onepass.geetest.com/v2.0/result"
```

注：如未完成服务端搭建，则只能体验 APP 端功能，不能获取真实手机号。

3、开发环境搭建

3.1、CocoaPods 集成

XCFramework 集成

XCFramework SDK 需要 iOS 11+, Xcode 14+。

执行 `pod repo update` 更新。

Podfile 里面添加以下代码：

```
# 以下两种版本选择方式示例

# 集成最新版 SDK：
pod 'OneLoginSDK-iOS-xcframework'

# 集成指定版本 SDK，具体版本号可先执行 pod search OneLoginSDK-iOS，根据返回的版本信息自行决定:pod 'OneLoginSDK-iOS-xcframework', '~> 2.8.0'
```

保存并执行 `pod install` 即可，若未执行 `pod repo update`，请执行 `pod install --repo-update`。

3.2、手动集成

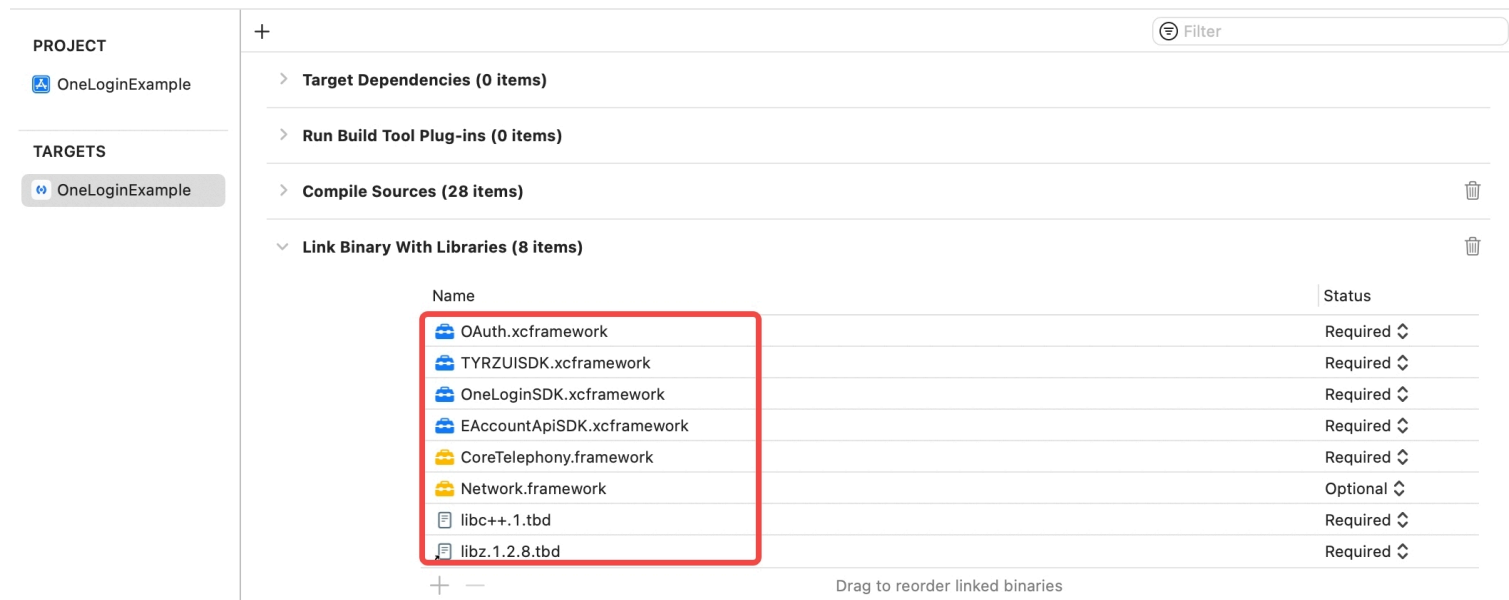
1. 将下载获取的

`OneLoginSDK.xcframework`、`EAccountApiSDK.xcframework`、`TYRZUISDK.xcframework`、`OAuth.xcframework` 以及 `OneLoginResource.bundle` 共 5 个文件添加到工程中，确保 `Copy items if needed` 已被勾选。

2. 在拖入

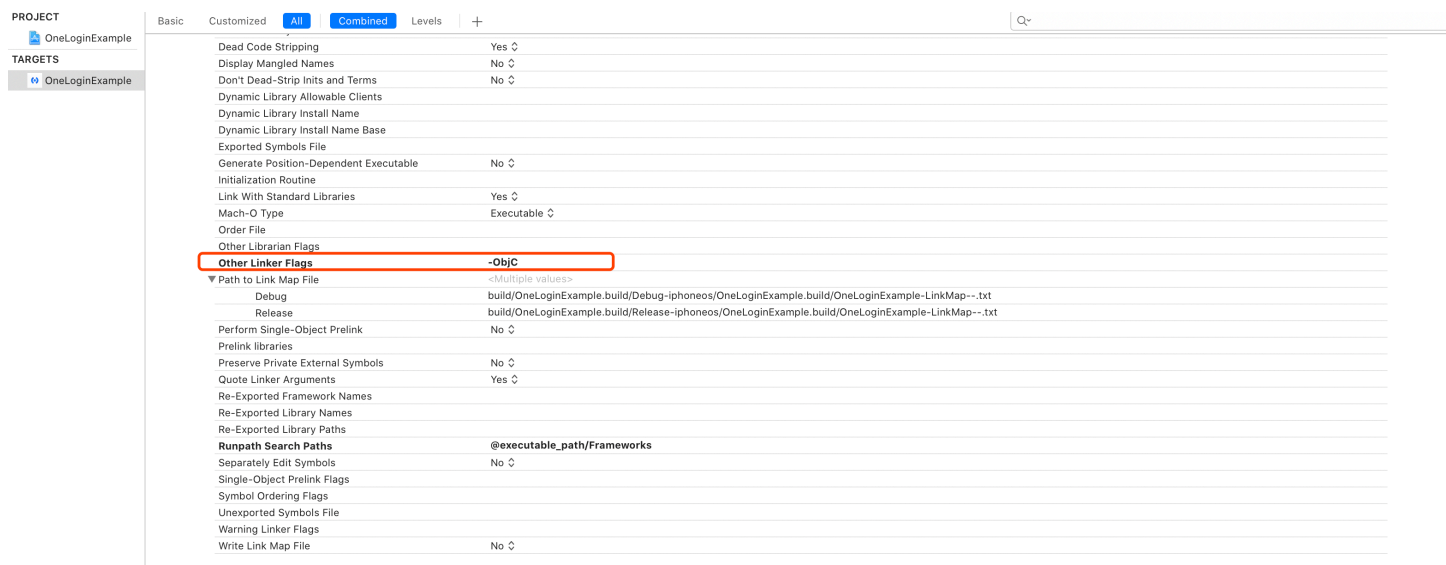
`OneLoginSDK.xcframework`、`EAccountApiSDK.xcframework`、`TYRZUISDK.xcframework`、`OAuth.xcframework` 到工程后，请检查所有的 `.xcframework` 是否被添加到 `PROJECT -> Build Phases -> Linked Frameworks and Libraries`，以确保正常编译。

3. 此外，需要添加系统的 `libc++.1.tbd`、`libz.1.2.8.tbd` 库进行依赖，如需支持 iOS 12 以下系统，还需要添加 `Network.framework`，并将 `status` 设置为 `Optional`。



3.3、工程配置

1. 针对静态库中的 `Category`，需要在对应 target 的 `Build Settings -> Other Linker Flags` 添加 `-ObjC` 编译选项。



苹果在 WWDC23 上发布了应用程序（包括 SDK）的新隐私政策，并有 [Get started with privacy manifests - WWDC23 - Videos - Apple Developer](#) 的单独专题。7 月 27 日苹果发布了此则新闻，新闻称：2023 年秋季开始，如果新上传的应用程序中使用了没有提供隐私清单的相关 API，那么你会收到一封邮件通知，而从 2024 年春季开始，隐私清单会变成一个强制要求。涉及到的 API 以及使用原因可参考：[Describing use of required reason API | Apple Developer Documentation](#)，如果使用原因未在列表中列出也可直接提交具体使用理由。

如何创建新隐私清单说明，可参考 [Privacy manifest files | Apple Developer Documentation](#)

一键登录 iOS SDK 使用了其中部分函数用于产品配置及日志相关，涉及

`NSPrivacyAccessedAPICategoryUserDefaults`、`NSPrivacyAccessedAPICategoryFileTimestamp`，通过交

互信息确认同意相关用户协议并授权使用相关的产品，涉及 `NSPrivacyCollectedDataTypeProductInteraction` 特此说明。

OneLogin (一键登录)

进阶模式和常规模式是一键登录的两种调用逻辑，如无特别需求建议直接使用进阶模式，两模式不可混合调用。

1、进阶模式

预取号逻辑封装在 SDK 内部，开发者只需控制授权页拉起时机。SDK 内部处理预取号的逻辑，包括预取号超期后的重新预取号，以及弱网状态下的重试等。使用这种方式时，请使用 `OneLoginPro` 类中的方法

1.1、调用逻辑

- `registerWithAppID`: 初始化 SDK 并配置 `APPID` (应用启动或进入登录页的前一个页面是调用该方法时机，该方法在整个 APP 生命周期内只需调用一次)
- `requestTokenWithViewController:viewModel:completion:` 拉起授权页面(调用该方法前可以调用 `isPreGetTokenResultValidate` 判断预取号是否成功)
- `dismissAuthViewController:completion:` 关闭授权页面

具体可参考 [Demo 中的示例代码](#)，[点击下载1](#)。

2、初始化

方法原型

```
/**
 * 向SDK注册AppID
 *
 * @discussion `AppID`通过后台注册获得，从 后台获取该AppID，AppID需与bundleID配套 @
 *
 * param appID 产品ID
 */
+ (void)registerWithAppID:(NSString *)appID;
```

参数描述

参数	是否必填	类型	说明
appID	是	NSString	appID

接口作用

传入 appID，并开始预取号，**注意客户端 AppID 需与服务端保持一致**

用场景

- 保证在拉起授权页面前至少调用一次，建议在应用启动时或进入登录页面的前一个页面调用该方法
- 只需调用一次，多次调用不会多次初始化，与一次调用效果一致

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 在 UIViewController 的 `viewDidLoad` 方法中添加初始化代码

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // 设置AppId, AppID通过后台注册获得, 从 后台获取该AppID, AppID需与bundleID配套[OneLoginPro registerWithAppID:GTOneLoginAppId];
}
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 在 UIViewController 的 `viewDidLoad` 方法中添加初始化代码

```
override func viewDidLoad() {
    super.viewDidLoad()
    // 设置AppId, AppID通过后台注册获得, 从 后台获取该AppID, AppID需与bundleID配套
    OneLoginPro.register(withAppID: GTOneLoginAppId)
}
```

1.3、进入授权页面

方法原型

```
/**
  进行用户认证授权，获取网关 token 。
```

@discussion 调用限制说明

为避免授权页面多次弹出，在调用该方法后，授权页面弹出，再次调用该方法时，该方法会直接跳出，不执行授权逻辑。

@discussion 需要用户在弹出的页面上同意服务条款后，才会进行免密认证。

@param viewController present 认证页面控制器的 vc
@param viewModel 自定义授权页面的视图模型
@param completion 结果处理回调

@seealso OLAuthViewModel

```
*/  
+ (void)requestTokenWithViewController:(nullable UIViewController *)viewController  
    viewModel:(nullable OLAuthViewModel *)viewModel  
    completion:(void(^)(NSDictionary * _Nullable  
result))completion;
```

参数描述

参数	是否必填	类型	说明
viewController	是	UIViewController	进入授权页面的上一级页面
viewModel	否	OLAuthViewModel	授权页面UI配置参数
completion	是	void(^)(NSDictionary * _Nullable result)	获取token回调

- 注意：
- 1、在成功进入授权页面时，不会立即收到 completion 回调，在授权页面点击一键登录、切换账号或者返回按钮时，才会收到 completion 回调；
 - 2、若不能成功进入授权页面，则会立即收到 completion 回调

回调的字段如下：

```
获取成功：  
{  
  "model" : "iPhone9,1",  
  "authcode" : "0000",  
  "operatorType" : "CU",  
  "release" : "13.5.1",  
  "processID" : "967ceb230b3fd4d74ebcb470c5830c",  
  "appID" : "b41a959b5cac4dd1277183e074630945",  
  "pre_token_time" : "1012",  
  "token" :  
  "CU__0__b41a959b5cac4dd1277183e074630945__2.3.4__1__f632d01ab7c64efda96580c3274de971__NOTC  
  UCC",  
  "number" : "186***6173",  
  "preGetTokenSuccesstedTime" : 1604890895.807291,  
  "errorCode" : "0",  
  "msg" : "获取accessToken成功",  
  "status" : 200,
```

```
"expire_time" : 580
}

获取失败:
{
    "status" : 500,
    "operatorType" : "CU",
    "appID" : "b41a959b5cac4dd1277183e074630945",
    "model" : "iPhone9,1",
    "release" : "13.5.1",
    "msg" : "Can't access cellular.",
    "errorCode" : "-20202"
}
```

字段	说明
status	状态码，200 表示成功，500 表示失败
token	换取手机号需要的 token
processID	流水号
authcode	authcode
applD	applD
operatorType	运营商
number	脱敏手机号
errorCode	失败时的具体错误码
msg	失败时表示失败原因

接口作用

拉起授权页面，用户在授权页面点击一键登录，即可获取 token，拿该 token 即可换取对应的手机号

使用场景

- 用户进行一键登录操作时调用该方法，如果初始化成功SDK 将会拉起授权页面
- 可以在多处调用
- 需在调用初始化方法之后调用

示例代码

ObjC:

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

2. 在需要使用一键登录的地方调用 一键登录接口

```
- (IBAction)normalLoginAction:(UIButton *)sender {
    OLAAuthViewModel *viewModel = [OLAAuthViewModel new];
    [OneLoginPro requestTokenWithViewController:self viewModel:viewModel
     completion:^(NSDictionary * _Nullable result) {
        NSLog(@"OneLoginPro requestTokenWithViewController result: %@", result);
    }];
}
```

Swift:

- 1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
- 2. 在需要使用一键登录的地方调用 一键登录接口

```
@IBAction func normalLoginAction(_ sender: UIButton) {
    let viewModel = OLAAuthViewModel()
    // 进入授权页面
    OneLoginPro.requestToken(with: self, viewModel: viewModel) { [weak self] result in
    }
}
```

1.4、手动关闭授权页面

方法原型

```
/**
 * @abstract 关闭当前的授权页面
 *
 * @param animated 是否需要动画
 * @param completion 关闭页面后的回调
 *
 * @discussion
 * 请不要使用其他方式关闭授权页面，否则可能导致 OneLogin 无法再次调起
 */
+ (void)dismissAuthViewController:(BOOL)animated completion:(void (^ __nullable)
(void))completion;
```

参数描述

参数	是否必填	类型	说明
animated	是	BOOL	关闭授权页面时是否需要动画
completion	否	void(^)(void)	关闭授权页面后的回调

接口作用

当开发者设置点击一键登录或者自定义控件不自动销毁授权页时，将需要自行调用此方法主动销毁授权页，建议在置换手机号成功后销毁，请不要使用其他方式关闭授权页面

关闭授权页面时机

- 在授权页面点击切换账号按钮时
- 在授权页面点击一键登录按钮收到 requestToken 结果回调之后

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 在需要关闭授权页面的地方调用 关闭授权页面接口

```
- (void)dismissAuthVC {  
    [OneLoginPro dismissAuthViewController:nil];  
}
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 在需要关闭授权页面的地方调用 关闭授权页面接口

```
@objc func dismissAuthVC(_ sender: UIButton) {  
    OneLoginPro.dismissAuthViewController {  
  
    }  
}
```

1.5、重新开始预取号

方法原型

```
/**  
 * @abstract 重新预取号  
 *  
 * @discussion 在通过requestTokenWithCompletion方法成功登录之后，为保证用户在退出登录之后，能快速拉起授权页面，请在用户退出登录时，调用此方法  
 */  
+ (void)renewPreGetToken;
```

接口作用

在 SDK 初始化之后，调用 `requestTokenWithCompletion` 获取 token 之前，SDK 内部会一直维护预取号的结果，但是在调用 `requestTokenWithCompletion` 获取 token 之后，就不再维护预取号结果了，调用该接口，就会重新开始预取号，并维持预取号结果有效，以保证在用户退出登录之后再次登录时能快速进入授权页面

调用时机

- 建议在用户退出登录时调用该接口

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 重新预取号

```
[OneLoginPro renewPreGetToken];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 重新预取号

```
OneLoginPro.renewPreGetToken()
```

1.6、判断预取号结果是否有效

方法原型

```
/**
 * @abstract 判断预取号结果是否有效
 *
 * @return YES：预取号结果有效，可直接拉起授权页面 NO：预取号结果无效，需加载进度条，等待预取号完成之后拉起授权页面
 */
+ (BOOL)isPreGetTokenResultValidate;
```

接口作用

判断预取号结果是否有效

调用时机

- 调用 `requestTokenWithCompletion` 时，判断是否需要加载进度条

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 判断预取号结果是否有效

```
// 在SDK内部预取号未成功时，建议加载进度条
if (![OneLoginPro isPreGetTokenResultValidate]) {
    [GTPProgressHUD showLoadingHUDWithMessage:nil];
}
[OneLoginPro requestTokenWithViewController:self viewModel:viewModel
completion:^(NSDictionary * _Nullable result) {
    NSLog(@"OneLoginPro requestTokenWithViewController result: %@", result);
}];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 判断预取号结果是否有效

```
// 进入授权页面
if !OneLoginPro.isPreGetTokenResultValidate() {
    GTPProgressHUD.showLoadingHUD(withMessage: nil)
}

OneLoginPro.requestToken(with: self, viewModel: viewModel) { [weak self] result in
    if let strongSelf = self {
        strongSelf.finishRequsetingToken(result: result!)
    }
    sender.isEnabled = true
}
```

1.7、设置请求超时时间

方法原型

```
/**
    分别设置预取号和取号请求超时时长。默认时长8s。

    @param preGetTokenTimeout 预取号超时时长
    @param requestTokenTimeout 取号超时时长
    */
```

```
+ (void)setRequestTimeout:(NSTimeInterval)preGetTokenTimeout requestTokenTimeout:
(NSTimeInterval)requestTokenTimeout;
```

接口作用

分开设置预取号和取号请求超时时间

调用时机

- 调用 `registerWithAppID` 前先设置请求超时时间

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 分别设置预取号和取号请求超时时长

```
[OneLoginPro setRequestTimeout:10 requestTokenTimeout:10];
[OneLoginPro registerWithAppID:@"您的appID"];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 分别设置预取号和取号请求超时时长

```
// 进入授权页面
OneLoginPro.setRequestTimeout(10, requestTokenTimeout: 10)
OneLoginPro.register(withAppID: "您的appID")
```

1.8、改变隐私条款勾选框状态

方法原型

```
/**
/**
@abstract 改变服务条款左边复选框勾选状态
@param isChecked 是否勾选
*/
+ (void)setProtocolCheckState:(BOOL)isChecked;
```

接口作用

改变隐私条款勾选框状态

调用时机

- 可使用 `hintBlock` 自定义未勾选隐私条款勾选框点击授权按钮的操作，然后调用此接口勾选条款

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 改变隐私条款勾选框状态

```
[OneLoginPro setProtocolCheckState:YES];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 改变隐私条款勾选框状态

```
OneLoginPro.setProtocolCheckState(true)
```

1.9、开始取号

方法原型

```
/**
 * @abstract 开始取号
 */
+ (void)startRequestToken;
```

接口作用

开始取号

调用时机

- 使用 `customAuthActionBlock` 返回 YES 自定义授权页面登录按钮点击事件，点击授权按钮后先接入其他验证方式，在此加入接入方的逻辑，验证通过后调用 `startRequestToken` 方法继续执行取号操作

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 开始取号


```
[OneLoginPro startRequestToken];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 开始取号

```
OneLoginPro.startRequestToken()
```

2、常规模式

预取号和授权页拉起时机均由开发者自主控制。使用此调用逻辑，开发者需自主处理预取号的时机和超期重试的逻辑等。使用这种方式时，请使用 `OneLogin` 类中的方法。

若无特别需求，建议直接使用进阶模式，常规模式集成步骤请参考[常规模式集成文档](#)

授权页面 UI 修改

1、设计规范

IOS样式规范说明

授权页背景图片

backgroundImage 设置授权页背景图片

状态栏

statusBarStyle 设置状态栏样式

号码栏

phoneNumColor 设置手机号码字体颜色
phoneNumFont 设置手机号码字体属性
phoneNumRect 设置号码栏位置
attrPhone 设置号码富文本

切换账号

switchButtonText 设置切换账号文字
switchButtonFont 设置切换账号字体属性
switchButtonColor 设置切换账号字体颜色
switchButtonBackgroundColor 设置切换账号背景颜色
switchButtonHidden 设置隐藏“切换账号”
switchButtonRect 设置切换账号位置及大小

Slogan

sloganTextColor 设置slogan文字颜色
sloganTextFont 设置slogan字体属性
sloganRect 设置slogan位置及大小

自定义控件

customUIHandler 添加自定义控件，可在导航栏底部到屏幕底部之间添加自定义控件

导航栏

naviTitle 设置导航栏标题文字及属性
naviBgColor 设置导航栏背景颜色
naviBackImage 设置导航栏返回按钮图标
backButtonRect 设置导航栏返回按钮位置及大小
backButtonHidden 设置导航栏返回按钮是否隐藏
naviRightControl 设置导航栏右侧自定义控件
naviHidden 设置隐藏导航栏

页面Logo

appLogo 设置Logo图片
logoHidden 设置隐藏Logo
logoCornerRadius 设置圆角
logoRect 设置Logo位置及大小

登录按钮

authButtonTitle 设置登录按钮文字及属性
authButtonImages 设置授权登录按钮三种状态的图片数组
authButtonCornerRadius 设置圆角
authButtonRect 设置登录按钮位置及大小

隐私栏

additionalPrivacyTerms 设置额外的开发者隐私条款名称、URL和顺序，最多2个
privacyTermsAttributes 设置隐私条款文字属性
termTextColor 设置隐私条款基础文字的颜色
uncheckedImage 设置复选框未选中时的图片
checkedImage 设置复选框选中时的图片
checkBoxRect 设置复选框位置及大小
defaultCheckBoxState 设置隐私条款check框默认状态
termsRect 设置隐私条款位置及大小
defaultCheckBoxState 设置勾选框默认是否勾选

注意：

- 1、开发者不得通过任何技术手段，破解授权页，或将授权页面的隐私栏、品牌露出内容隐藏、覆盖；
- 2、登录按钮文字描述必须包含“登录”或“注册”等文字，不得诱导用户授权
- 3、自定义控件不允许覆盖SDK默认的UI；
- 4、对于接入极验SDK并上线的应用，我方和运营商会对接线的授权页面做审查，如果有出现未按要求弹出或设计授权页面的，将关闭应用的认证取号服务。

2、页面控件位置大小设置

支持两种方式设置授权页面控件的位置和大小：

- 设置控件的 OLRect 属性，通过设置控件距离屏幕顶部和屏幕左边的偏移量来控制控件的位置，通过设置控件的宽高来控制控件的大小
- 在回调中拿到授权页面所有控件，通过 masnory 或其他方式进行自动布局

2.1、设置控件的 OLRect 属性

```
/**
 * @abstract 1、若授权页面只支持竖屏，只设置竖屏方向偏移；
 *           2、若授权页面只支持横屏，只设置横屏方向偏移；
 *           3、若授权页面支持旋转自动切换横竖屏，则同时设置竖屏方向和横屏方向偏移
 *           4、弹窗模式，同以上1、2、3
 *           5、size默认都可以不用设置，会根据字体大小自适应
 *           6、x轴方向偏移量有两个值可以设置，portraitCenterXOffset为控件的x轴中点到弹窗x轴中点的距离，portraitLeftXOffset为控件的左边缘到屏幕左边缘的距离，两者选其一即可
 */
typedef struct OLRect {
```

```
/**
 竖屏时
 导航栏隐藏时，为控件顶部到状态栏的距离；导航栏显示时，为控件顶部到导航栏底部的距离
 弹窗时
 为控件顶部到弹窗顶部的距离
 */
```

```
CGFloat portraitTopYOffset;
```

```
/**
 竖屏时
 控件的x轴中点到屏幕x轴中点的距离，默认为0
 弹窗时
 控件的x轴中点到弹窗x轴中点的距离，默认为0
 */
```

```
CGFloat portraitCenterXOffset;
```

```
/**
 竖屏时
 控件的左边缘到屏幕左边缘的距离，默认为0
 弹窗时
 控件的左边缘到屏幕左边缘的距离，默认为0
```

portraitLeftXOffset与portraitCenterXOffset设置一个即可，portraitLeftXOffset优先级大于portraitCenterXOffset,

```
设置此属性时，portraitCenterXOffset属性失效
 */
```

```
CGFloat portraitLeftXOffset;
```

```
/**
 横屏时
 导航栏隐藏时，为控件顶部到屏幕顶部的距离；导航栏显示时，为控件顶部到导航栏底部的距离
 弹窗时
 为控件顶部到弹窗顶部的距离
 */
```

```
CGFloat landscapeTopYOffset;
```

```
/**
 横屏时
 控件的x轴中点到屏幕x轴中点的距离，默认为0
 弹窗时
 控件的x轴中点到弹窗x轴中点的距离，默认为0
 */
```

```
CGFloat landscapeCenterXOffset;
```

```
/**
 横屏时
 控件的左边缘到屏幕左边缘的距离，默认为0
 弹窗时
 控件的左边缘到屏幕左边缘的距离，默认为0
```

landscapeLeftXOffset与landscapeCenterXOffset设置一个即可，landscapeLeftXOffset优先级大于

```
landscapeCenterXOffset,
    设置此属性时，landscapeCenterXOffset属性失效
    */
    CGFloat landscapeLeftXOffset;

    /**
     控件大小，只有宽度、高度同时大于0，设置的size才会生效，否则为控件默认的size
     */
    CGSize size;
} OLRect;

/**
 返回按钮位置及大小，返回按钮最大size为CGSizeMake(40, 40)。
  */
@property (nonatomic, assign) OLRect backButtonRect;

/**
  Logo 位置及大小。
  */
@property (nonatomic, assign) OLRect logoRect;

/**
  号码预览 位置及大小
  */
@property (nonatomic, assign) OLRect phoneNumRect;

/**
  授权页切换账号按钮 位置及大小。
  */
@property (nonatomic, assign) OLRect switchButtonRect;

/**
  授权按钮 位置及大小。
  */
@property (nonatomic, assign) OLRect authButtonRect;

/**
  Slogan 位置及大小。
  */
@property (nonatomic, assign) OLRect sloganRect;

/**
  授权页面上条款勾选框大小及位置。
  */
@property (nonatomic, assign) OLRect checkBoxRect;

/**
  隐私条款 位置及大小，隐私条款，宽需大于50，高需大于20，才会生效。
  */
@property (nonatomic, assign) OLRect termsRect;
```

```
/**
 弹窗 位置及大小。弹窗模式时，x轴偏移只支持portraitLeftXOffset和landscapeLeftXOffset。
 */
@property (nonatomic, assign) OLRect popupRect;
```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置控件的 OLRect 属性

```
OLRect backButtonRect = {0, 0, 0, 0, 0, 0, {0, 0}}; // 返回按钮偏移、大小设置，偏移量和大小设置值需大于0，否则取默认值，默认可不设置
viewModel.backButtonRect = backButtonRect;

OLRect logoRect = {0, 0, 0, 20, 0, 0, {0, 0}}; // logo偏移、大小设置，偏移量和大小设置值需大于0，否则取默认值，默认可不设置，logo大小默认为图片大小
viewModel.logoRect = logoRect;

OLRect phoneNumRect = {0, 0, 0, 0, 0, 0, {0, 0}}; // 手机号偏移设置
viewModel.phoneNumRect = phoneNumRect;

OLRect switchButtonRect = {0, 0, 0, 0, 0, 0, {0, 0}}; // 切换按钮偏移、大小设置，偏移量和大小设置值需大于0，否则取默认值，默认可不设置
viewModel.switchButtonRect = switchButtonRect;

OLRect authButtonRect = {0, 0, 0, 0, 0, 0, {300, 40}}; // 授权按钮偏移、大小设置，偏移量和大小设置值需大于0，否则取默认值，默认可不设置
viewModel.authButtonRect = authButtonRect;

OLRect sloganRect = {0, 0, 0, 0, 0, 0, {0, 0}}; // slogan偏移、大小设置，偏移量和大小设置值需大于0，否则取默认值，默认可不设置
viewModel.sloganRect = sloganRect;

OLRect checkBoxRect = {0, 0, 0, 0, 0, 0, {12, 12}};
viewModel.checkBoxRect = checkBoxRect; // 复选框尺寸，默认为12*12

OLRect termsRect = {0, 0, 0, 0, 0, 0, {0, 0}}; // 服务条款偏移、大小设置，偏移量和大小设置值需大于0，否则取默认值，默认可不设置
viewModel.termsRect = termsRect;
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置控件的 OLRect 属性

```

let backButtonRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // 返回按钮偏移、大小设置, 偏移
量和大小设置值需大于0, 否则取默认值, 默认可不设置
viewModel.backButtonRect = backButtonRect

let logoRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 20, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // logo偏移、大小设置, 偏移量
和大小设置值需大于0, 否则取默认值, 默认可不设置, logo大小默认为图片大小
viewModel.logoRect = logoRect

let phoneNumRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // 手机号偏移设置
viewModel.phoneNumRect = phoneNumRect

let switchButtonRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // 切换按钮偏移、大小设置, 偏
移量和大小设置值需大于0, 否则取默认值, 默认可不设置
viewModel.switchButtonRect = switchButtonRect

let authButtonRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 300, height: 40)) // 授权按钮偏移、大小设置,
偏移量和大小设置值需大于0, 否则取默认值, 默认可不设置
viewModel.authButtonRect = authButtonRect

let sloganRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // slogan偏移、大小设置, 偏
移量和大小设置值需大于0, 否则取默认值, 默认可不设置
viewModel.sloganRect = sloganRect

let checkBoxRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 12, height: 12)) // 复选框尺寸, 默认为12*12
viewModel.checkBoxRect = checkBoxRect

let termsRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // 服务条款偏移、大小设置, 偏
移量和大小设置值需大于0, 否则取默认值, 默认可不设置
viewModel.termsRect = termsRect

```

2.2、控件进行自动布局

```

/**
 * @abstract 授权页面视图控件自动布局回调，可在该回调中，对控件通过 masonry 或者其他方式进行自动布局，若
需要自定义视图，请直接在该回调中添加，实现该回调后，授权页面所有视图的约束都会被删除，您需要重新设置所有视图
的约束
 *
 * authView 为 authContentView 的父视图
 * authContentView 为 authBackgroundImageView、authNavigationView、authLogoView、
authPhoneView、authSwitchButton、authLoginButton、authSloganView、authAgreementView、
authClosePopupButton 的父视图
 * authNavigationView 为 authNavigationContainerView 的父视图
 * authNavigationContainerView 为 authBackButton 和 authNavigationTitleView 的父视图
 * authAgreementView 为 authCheckbox 和 authProtocolView 的父视图
 *
 */
typedef void(^OLAuthVCAutoLayoutBlock)(UIView *authView, UIView *authContentView, UIView
*authBackgroundImageView, UIView *authNavigationView, UIView *authNavigationContainerView,
UIView *authBackButton, UIView *authNavigationTitleView, UIView *authLogoView, UIView
*authPhoneView, UIView *authSwitchButton, UIView *authLoginButton, UIView *authSloganView,
UIView *authAgreementView, UIView *authCheckbox, UIView *authProtocolView, UIView
*authClosePopupButton);

/**
 * 授权页面视图控件自动布局回调
 */
@property (nullable, nonatomic, copy) OLAuthVCAutoLayoutBlock autolayoutBlock;

```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 通过回调对控件进行自动布局

```

viewModel.autolayoutBlock = ^(UIView *authView, UIView *authContentView, UIView
*authBackgroundImageView, UIView *authNavigationView, UIView
*authNavigationContainerView, UIView *authBackButton, UIView *authNavigationTitleView,
UIView *authLogoView, UIView *authPhoneView, UIView *authSwitchButton, UIView
*authLoginButton, UIView *authSloganView, UIView *authAgreementView, UIView
*authCheckbox, UIView *authProtocolView, UIView *authClosePopupButton) {
    // content
    [authContentView mas_makeConstraints:^(MASConstraintMaker *make) {
        make.edges.equalTo(authView);
    }];

    // background
    [authBackgroundImageView mas_makeConstraints:^(MASConstraintMaker *make) {
        make.edges.equalTo(authContentView);
    }];

```

```

// navigation
[authNavigationView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.left.top.right.equalTo(authContentView);
    make.height.mas_equalTo(64);
}];

[authNavigationContainerView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.edges.equalTo(authNavigationView);
}];

[authBackButton mas_makeConstraints:^(MASConstraintMaker *make) {
    make.left.equalTo(authNavigationContainerView).offset(20);
    make.centerY.equalTo(authNavigationContainerView).offset(10);
    make.size.mas_equalTo(CGSizeMake(20, 20));
}];

[authNavigationView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.centerX.equalTo(authNavigationContainerView);
    make.centerY.equalTo(authNavigationContainerView).offset(10);
    make.size.mas_equalTo(CGSizeMake(100, 40));
}];

UIButton *rightBarButton = [UIButton buttonWithTypeCustom];
[rightBarButton setTitle:@"完成" forState:UIControlStateNormal];
[rightBarButton addTarget:self action:@selector(doneAction:)
forControlEvents:UIControlEventTouchUpInside];
[authNavigationContainerView addSubview:rightBarButton];
[rightBarButton mas_makeConstraints:^(MASConstraintMaker *make) {
    make.right.equalTo(authNavigationContainerView).offset(-10);
    make.centerY.equalTo(authNavigationContainerView).offset(10);
    make.size.mas_equalTo(CGSizeMake(60, 40));
}];

// logo
[authLogoView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.centerX.equalTo(authContentView);
    make.top.equalTo(authNavigationView.mas_bottom).offset(100);
    make.size.mas_equalTo(CGSizeMake(107, 22));
}];

// phone
[authPhoneView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.centerX.equalTo(authContentView);
    make.top.equalTo(authLogoView.mas_bottom).offset(20);
    make.size.mas_equalTo(CGSizeMake(200, 40));
}];

// switchbutton
[authSwitchButton mas_makeConstraints:^(MASConstraintMaker *make) {
    make.centerX.equalTo(authContentView);

```



```

        make.top.equalTo(authPhoneView.mas_bottom).offset(20);
        make.size.mas_equalTo(CGSizeMake(200, 20));
    }];

// loginbutton
[authLoginButton mas_makeConstraints:^(MASConstraintMaker *make) {
    make.centerX.equalTo(authContentView);
    make.top.equalTo(authSwitchButton.mas_bottom).offset(30);
    make.size.mas_equalTo(CGSizeMake(260, 40));
}];

// slogan
[authSloganView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.centerX.equalTo(authContentView);
    make.top.equalTo(authLoginButton.mas_bottom).offset(20);
    make.size.mas_equalTo(CGSizeMake(260, 20));
}];

// agreementview
[authAgreementView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.left.equalTo(authContentView).offset(20);
    make.right.equalTo(authContentView).offset(-20);
    make.top.equalTo(authSloganView.mas_bottom).offset(50);
    make.height.mas_equalTo(80);
}];

[authCheckbox mas_makeConstraints:^(MASConstraintMaker *make) {
    make.left.equalTo(authAgreementView).offset(10);
    make.top.equalTo(authAgreementView).offset(10);
}];

[authProtocolView mas_makeConstraints:^(MASConstraintMaker *make) {
    make.left.equalTo(authCheckbox.mas_right).offset(5);
    make.right.equalTo(authAgreementView).offset(-10);
    make.height.equalTo(authAgreementView);
}];

// 自定义视图
UIButton *customBtn = [[UIButton alloc] initWithFrame:CGRectMake(0, 0, 200, 40)];
[customBtn setTitle:@"我是自定义UI" forState:UIControlStateNormal];
customBtn.backgroundColor = [UIColor lightGrayColor];
customBtn.layer.cornerRadius = 2.0;
[customBtn addTarget:self action:@selector(dismissAuthVC)
forControlEvents:UIControlEventTouchUpInside];
[authContentView addSubview:customBtn];
[customBtn mas_makeConstraints:^(MASConstraintMaker *make) {
    make.left.equalTo(authContentView).offset(20);
    make.right.equalTo(authContentView).offset(-20);
    make.height.mas_equalTo(40);
    make.top.equalTo(authAgreementView.mas_bottom).offset(30);
}];

```

```
};
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 通过回调对控件进行自动布局

```
viewModel.autolayoutBlock = { [weak self] (authView: UIView, authContentView: UIView,
authBackgroundImageView: UIView, authNavigationView: UIView,
authNavigationContainerView: UIView, authBackButton: UIView, authNavigationView:
UIView, authLogoView: UIView, authPhoneView: UIView, authSwitchButton: UIView,
authLoginButton: UIView, authSloganView: UIView, authAgreementView: UIView,
authCheckbox: UIView, authProtocolView: UIView, authClosePopupButton: UIView) in
    // content
    authContentView.mas_makeConstraints { (make: MASConstraintMaker?) in
        make?.edges.equalTo()(authView)
    }

    // background
    authBackgroundImageView.mas_makeConstraints { (make: MASConstraintMaker?) in
        make?.edges.equalTo()(authContentView)
    }

    // navigation
    authNavigationView.mas_makeConstraints { (make: MASConstraintMaker?) in
        make?.left.top().right().equalTo()(authContentView)
        make?.height.mas_equalTo()(64)
    }

    authNavigationContainerView.mas_makeConstraints { (make: MASConstraintMaker?) in
        make?.edges.equalTo()(authNavigationView)
    }

    authBackButton.mas_makeConstraints { (make: MASConstraintMaker?) in
        make?.left.equalTo()(authNavigationContainerView)?.offset()(20)
        make?.centerY.equalTo()(authNavigationContainerView)?.offset()(10)
        make?.size.mas_equalTo()(CGSize.init(width: 20, height: 20))
    }

    authNavigationView.mas_makeConstraints { (make: MASConstraintMaker?) in
        make?.centerX.equalTo()(authNavigationContainerView)
        make?.centerY.equalTo()(authNavigationContainerView)?.offset()(10)
        make?.size.mas_equalTo()(CGSize.init(width: 100, height: 40))
    }

    // 导航栏右侧控制视图
    let strongSelf = self
    let rightBarButton = UIButton.init(type: UIButton.ButtonType.custom)
    rightBarButton.setTitle("完成", for: UIControl.State.normal)
```

```

rightBarButton.addTarget(strongSelf, action: #selector(strongSelf?.doneAction),
for: UIControl.Event.touchUpInside)
authNavigationContainerView.addSubview(rightBarButton)
rightBarButton.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.right.equalTo()(authNavigationContainerView)?.offset()(-10)
    make?.centerY.equalTo()(authNavigationContainerView)?.offset()(10)
    make?.size.mas_equalTo()(CGSize.init(width: 60, height: 40))
}

// logo
authLogoView.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.centerX.equalTo()(authContentView)
    make?.top.equalTo()(authNavigationView.mas_bottom)?.offset()(100)
    make?.size.mas_equalTo()(CGSize.init(width: 107, height: 22))
}

// phone
authPhoneView.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.centerX.equalTo()(authContentView)
    make?.top.equalTo()(authLogoView.mas_bottom)?.offset()(20)
    make?.size.mas_equalTo()(CGSize.init(width: 200, height: 40))
}

// switchbutton
authSwitchButton.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.centerX.equalTo()(authContentView)
    make?.top.equalTo()(authPhoneView.mas_bottom)?.offset()(20)
    make?.size.mas_equalTo()(CGSize.init(width: 200, height: 20))
}

// loginbutton
authLoginButton.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.centerX.equalTo()(authContentView)
    make?.top.equalTo()(authSwitchButton.mas_bottom)?.offset()(30)
    make?.size.mas_equalTo()(CGSize.init(width: 260, height: 40))
}

// slogan
authSloganView.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.centerX.equalTo()(authContentView)
    make?.top.equalTo()(authLoginButton.mas_bottom)?.offset()(20)
    make?.size.mas_equalTo()(CGSize.init(width: 260, height: 20))
}

// agreementview
authAgreementView.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.left.equalTo()(authContentView)?.offset()(20)
    make?.right.equalTo()(authContentView)?.offset()(-20)
    make?.top.equalTo()(authSloganView.mas_bottom)?.offset()(50)
    make?.height.mas_equalTo()(80)
}

```

```

authCheckbox.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.left.equalTo()(authAgreementView)?.offset()(10)
    make?.top.equalTo()(authAgreementView)?.offset()(10)
}

authProtocolView.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.left.equalTo()(authCheckbox.mas_right)?.offset()(5)
    make?.right.equalTo()(authAgreementView)?.offset()(-10)
    make?.height.equalTo()(authAgreementView);
}

// 自定义视图
let customBtn = UIButton.init(frame: CGRect.init(x: 0, y: 0, width: 200, height:
40))
customBtn.setTitle("我是自定义UI", for: UIControl.State.normal)
customBtn.backgroundColor = UIColor.red
customBtn.layer.cornerRadius = 2.0
customBtn.addTarget(strongSelf, action: #selector(strongSelf?.dismissAuthVC), for:
UIControl.Event.touchUpInside)
authContentView.addSubview(customBtn)
customBtn.mas_makeConstraints { (make: MASConstraintMaker?) in
    make?.left.equalTo()(authContentView)?.offset()(20)
    make?.right.equalTo()(authContentView)?.offset()(-20)
    make?.height.mas_equalTo()(40)
    make?.top.equalTo()(authAgreementView.mas_bottom)?.offset()(30)
}
}

```

3、页面控件属性设置

#pragma mark - Status Bar/状态栏

```

/**
 状态栏样式。 默认 `UIStatusBarStyleDefault`。
 */
@property (nonatomic, assign) UIStatusBarStyle statusBarStyle;

```

#pragma mark - Navigation/导航

```

/**
 授权页导航的标题。默认为空字符串。
 */
@property (nullable, nonatomic, strong) NSAttributedString *naviTitle;

/**
 授权页导航的背景颜色。默认白色。
 */

```

```
@property (nullable, nonatomic, strong) UIColor *naviBgColor;

/**
  授权页导航左边的返回按钮的图片。默认黑色系统样式返回图片。
 */
@property (nullable, nonatomic, strong) UIImage *naviBackImage;

/**
  授权页导航右边的自定义控件。
 */
@property (nullable, nonatomic, strong) UIView *naviRightControl;

/**
  导航栏隐藏。默认不隐藏。
 */
@property (nonatomic, assign) BOOL naviHidden;

/**
  返回按钮隐藏。默认不隐藏。
 */
@property (nonatomic, assign) BOOL backButtonHidden;

#pragma mark - Logo/图标

/**
  授权页面上展示的图标。默认为 "OneLogin" 图标。
 */
@property (nullable, nonatomic, strong) UIImage *appLogo;

/**
  Logo 图片隐藏。默认不隐藏。
 */
@property (nonatomic, assign) BOOL logoHidden;

/**
  logo圆角，默认为0。
 */
@property (nonatomic, assign) CGFloat logoCornerRadius;

#pragma mark - Phone Number Preview/手机号预览

/**
  号码预览文字的颜色。默认黑色。
 */
@property (nullable, nonatomic, strong) UIColor *phoneNumColor;

/**
  号码预览文字的字体。默认粗体，24pt。
 */
@property (nullable, nonatomic, strong) UIFont *phoneNumFont;
```

#pragma mark - Switch Button/切换按钮

```
/**
  授权页切换账号按钮的文案。默认为“切换账号”。
  */
@property (nullable, nonatomic, copy) NSString *switchButtonText;

/**
  授权页切换账号按钮的颜色。默认蓝色。
  */
@property (nullable, nonatomic, strong) UIColor *switchButtonColor;

/**
  授权页切换账号按钮背景颜色。默认为 nil。
  */
@property (nullable, nonatomic, strong) UIColor *switchButtonBackgroundColor;

/**
  授权页切换账号的字体。默认字体，15pt。
  */
@property (nullable, nonatomic, strong) UIFont *switchButtonFont;

/**
  隐藏切换账号按钮。默认不隐藏。
  */
@property (nonatomic, assign) BOOL switchButtonHidden;
```

#pragma mark - Authorization Button/认证按钮

```
/**
  授权页认证按钮的背景图片，@[正常状态的背景图片，不可用状态的背景图片，高亮状态的背景图片]。默认正常状态为蓝色纯色，不可用状态的背景图片时为灰色，高亮状态为灰蓝色。
  */
@property (nullable, nonatomic, strong) NSArray<UIImage *> *authButtonImages;

/**
  授权按钮文案。默认白色的"一键登录"。
  */
@property (nullable, nonatomic, strong) NSAttributedString *authButtonTitle;

/**
  授权按钮圆角，默认为0。
  */
@property (nonatomic, assign) CGFloat authButtonCornerRadius;
```

#pragma mark - Slogan/口号标语

```
/**
  Slogan 文字颜色。默认灰色。
  */
@property (nonatomic, strong) UIColor *sloganTextColor;
```

```

/**
    Slogan字体。默认字体，12pt。
    */
@property (nonatomic, strong) UIFont *sloganTextFont;

#pragma mark - CheckBox & Privacy Terms/隐私条款勾选框及隐私条款

/**
    授权页面上条款勾选框初始状态。默认 YES。
    */
@property (nonatomic, assign) BOOL defaultCheckBoxState;

/**
    授权页面上勾选框勾选的图标。默认为蓝色图标。推荐尺寸为12x12。
    */
@property (nullable, nonatomic, strong) UIImage *checkedImage;

/**
    授权页面上勾选框未勾选的图标。默认为白色图标。推荐尺寸为12x12。
    */
@property (nullable, nonatomic, strong) UIImage *uncheckedImage;

/**
    隐私条款文字属性。默认基础文字灰色，条款蓝色高亮，12pt。
    */
@property (nullable, nonatomic, strong) NSDictionary<NSAttributedStringKey, id>
    *privacyTermsAttributes;

/**
    额外的条款。默认为空。
    */
@property (nullable, nonatomic, strong) NSArray<OLPrivacyTermItem *>
    *additionalPrivacyTerms;

/**
    服务条款普通文字的颜色。默认灰色。
    */
@property (nullable, nonatomic, strong) UIColor *termTextColor;

/**
    除隐私条款外的其他文案，数组大小必须为4，元素依次为：条款前的文案、条款一和条款二连接符、条款二和条款三连接符，条款后的文案。
    默认为@[@"登录即同意", @"和", @"、", @"并使用本机号码登录"]
    */
@property (nullable, nonatomic, copy) NSArray<NSString *> *auxiliaryPrivacyWords;

/**
    * 点击授权页面隐私协议前勾选框的回调
    */
@property (nullable, nonatomic, copy) OLClickCheckboxBlock clickCheckboxBlock;

```

```

/**
 * 服务条款文案对齐方式，默认为NSTextAlignmentLeft
 */
@property (nonatomic, assign) NSTextAlignment termsAlignment;

#pragma mark - Background Image/授权页面背景图片

/**
 授权页背景颜色。默认白色。
 */
@property (nullable, nonatomic, strong) UIColor *backgroundColor;

/**
 授权页面背景图片
 */

@property (nullable, nonatomic, strong) UIImage *backgroundImage;

/**
 横屏模式授权页面背景图片
 */
@property (nullable, nonatomic, strong) UIImage *landscapeBackgroundImage;

#pragma mark - Background Gif/授权页面背景 gif

/**
 授权页面背景Gif路径，与背景图片、视频，三者只有一个有效，视频优先级最高，背景图其次，gif 最后
 */
@property (nullable, nonatomic, strong) NSString *backgroundGifPath;

#pragma mark - Background Video/授权页面背景视频

/**
 授权页面背景视频路径，可支持本地和网络视频，与背景图片、gif，三者只有一个有效，视频优先级最高，背景图其次，gif 最后
 */
@property (nullable, nonatomic, strong) NSString *backgroundVideoPath;

#pragma mark - 授权页文案多语言配置

/**
 * 设置授权页（含服务条款页面，但不包含运营商服务条款的具体内容）语言，仅支持设置中文简体、中文繁体、英文，
默认为中文简体（语言类型仅代表UI展示，一键登录功能仅支持中国移动、中国联通、中国电信的SIM卡）
 */
@property (nonatomic, assign) OLLanguageType languageType;

```

示例代码

ObjC:

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

2. 设置控件属性

```
// -----状态栏设置 -----
viewModel.statusBarStyle = UIStatusBarStyleLightContent;

// ----- 授权页面背景设置 -----
viewModel.backgroundColor = UIColor.lightGrayColor;
// viewModel.backgroundVideoPath = [[NSBundle mainBundle] pathForResource:@"test.mp4"
// ofType:nil]; //本地视频
// viewModel.backgroundGifPath = [[NSBundle mainBundle] pathForResource:@"test.gif"
// ofType:nil]; // gif

// ----- 导航栏设置 -----
viewModel.naviTitle = [[NSAttributedString alloc] initWithString:@"一键登录"

attributes:@{NSForegroundColorAttributeName : UIColor.whiteColor,
                                                     NSFontAttributeName
: [UIFont boldSystemFontOfSize:18]
                                                     }]; // 导航栏标题

viewModel.naviBgColor = UIColor.greenColor; // 导航栏背景色
viewModel.naviBackImage = [UIImage imageNamed:@"back"]; // 导航栏返回按钮
viewModel.backButtonHidden = NO; // 是否隐藏返回按钮，默认不隐藏

// ----- logo设置 -----
viewModel.appLogo = [UIImage imageNamed:@"网关取号_logo"]; // 自定义logo图片
viewModel.logoHidden = NO; // 是否隐藏logo，默认不隐藏
viewModel.logoCornerRadius = 0; // logo圆角，默认为0

// ----- 手机号设置 -----
viewModel.phoneNumColor = UIColor.redColor; // 颜色
viewModel.phoneNumFont = [UIFont boldSystemFontOfSize:25]; // 字体

// ----- 切换账号设置 -----
viewModel.switchButtonColor = UIColor.brownColor; // 切换按钮颜色
viewModel.switchButtonFont = [UIFont systemFontOfSize:15]; // 切换按钮字体
viewModel.switchButtonText = @"自定义切换按钮文案"; // 切换按钮文案
viewModel.switchButtonHidden = NO; // 是否隐藏切换按钮，默认不隐藏

// ----- 授权登录按钮设置 -----
viewModel.authButtonImages = @[
    [UIImage imageNamed:@"bg_logo_launch"],
    [UIImage imageNamed:@"bg_logo_launch"],
    [UIImage imageNamed:@"bg_logo_launch"]
]; // 授权按钮背景图片
viewModel.authButtonTitle = [[NSAttributedString alloc] initWithString:@"授权登录"

attributes:@{NSForegroundColorAttributeName : UIColor.whiteColor,
```

```
NSFontAttributeName : [UIFont boldSystemFontOfSize:18]
```

```
}}]; // 登录按
```

钮文案

```
viewModel.authButtonRect = authButtonRect;
viewModel.authButtonCornerRadius = 0; // 授权按钮圆角，默认为0
viewModel.clickAuthButtonBlock = ^(void) { // 点击授权页面登录按钮回调
    NSLog(@"clickAuthButtonBlock");
};

// ----- slogan设置 -----
viewModel.sloganTextColor = UIColor.cyanColor; // slogan颜色
viewModel.sloganTextFont = [UIFont systemFontOfSize:14]; // slogan字体

// ----- 服务条款设置 -----
viewModel.defaultCheckBoxState = YES; // 是否默认选择同意服务条款，默认同意
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置控件属性

```
// ----- 状态栏设置 -----
viewModel.statusBarStyle = .lightContent

// ----- 授权页面背景图片设置 -----
viewModel.backgroundColor = .lightGray
// viewModel.backgroundVideoPath = Bundle.main.path(forResource: "test.mp4", ofType:
// "")! //本地视频
// viewModel.backgroundGifPath = Bundle.main.path(forResource: "test.gif", ofType:
// "")! // gif

// ----- 导航栏设置 -----
viewModel.naviTitle = NSAttributedString.init(string: "一键登录", attributes:
[NSAttributedString.Key.foregroundColor : UIColor.white, NSAttributedString.Key.font :
UIFont.boldSystemFont(ofSize: 18)]) // 导航栏标题
viewModel.naviBgColor = .green // 导航栏背景色
viewModel.naviBackImage = UIImage.init(named: "back") // 导航栏返回按钮
viewModel.backButtonHidden = false // 是否隐藏返回按钮，默认不隐藏

// ----- logo设置 -----
viewModel.appLogo = UIImage.init(named: "网关取号_logo")
viewModel.logoHidden = false // 是否隐藏logo，默认不隐藏
viewModel.logoCornerRadius = 0

// ----- 手机号设置 -----
viewModel.phoneNumColor = UIColor.red // 颜色
viewModel.phoneNumFont = UIFont.boldSystemFont(ofSize: 25) // 字体
```

```
// ----- 切换账号设置 -----
viewModel.switchButtonColor = UIColor.brown // 切换按钮颜色
viewModel.switchButtonFont = UIFont.systemFont(ofSize: 15) // 切换按钮字体
viewModel.switchButtonText = "自定义切换按钮文案" // 切换按钮文案
viewModel.switchButtonHidden = false // 是否隐藏切换按钮，默认不隐藏

// ----- 授权登录按钮设置 -----
viewModel.authButtonImages = [(UIImage.init(named: "bg_logo_launch") ??
UIImage.init()), (UIImage.init(named: "bg_logo_launch") ?? UIImage.init()),
(UIImage.init(named: "bg_logo_launch") ?? UIImage.init())] // 授权按钮背景图片
viewModel.authButtonTitle = NSAttributedString.init(string: "授权登录", attributes:
[NSAttributedString.Key.foregroundColor : UIColor.white, NSAttributedString.Key.font :
UIFont.boldSystemFont(ofSize: 18)]) // 登录按钮文案

viewModel.authButtonCornerRadius = 0 // 授权按钮圆角，默认为0
viewModel.clickAuthButtonBlock = { // 点击授权页面登录按钮回调
    print("clickAuthButtonBlock")
}

// ----- slogan设置 -----
viewModel.sloganTextColor = UIColor.cyan // slogan颜色
viewModel.sloganTextFont = UIFont.systemFont(ofSize: 14) // slogan字体

// ----- 服务条款设置 -----
viewModel.defaultCheckBoxState = true
```

4、横竖屏设置

```
/**
 * 授权页面支持的横竖屏方向
 */
@property (nonatomic, assign) UIInterfaceOrientationMask supportedInterfaceOrientations;
```

通过设置 `supportedInterfaceOrientations` 属性，即可控制授权页面的横竖屏，授权页面可以支持横竖屏自由组合，若设置授权页面同时支持竖屏和横屏，授权页面会跟随设备旋转自动进行横竖屏切换

示例代码

ObjC:

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

2. 设置 `supportedInterfaceOrientations` 属性

```
// ----- 授权页面支持的横竖屏设置 -----
viewModel.supportedInterfaceOrientations = UIInterfaceOrientationMaskAllButUpsideDown;
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置 `supportedInterfaceOrientations` 属性

```
// ----- 授权页面支持的横竖屏设置 -----  
viewModel.supportedInterfaceOrientations = UIInterfaceOrientationMask.allButUpsideDown
```

5、弹窗模式设置

```
#pragma mark - Popup  
  
/**  
 * 是否为弹窗模式  
 */  
@property (nonatomic, assign) BOOL isPopup;  
  
/**  
 弹窗 位置及大小。弹窗模式时，x轴偏移只支持portraitLeftXOffset和landscapeLeftXOffset。  
 */  
@property (nonatomic, assign) CGRect popupRect;  
  
/**  
 弹窗圆角，默认为6。  
 */  
@property (nonatomic, assign) CGFloat popupCornerRadius;  
  
/**  
 当只需要设置弹窗的部分圆角时，通过popupCornerRadius设置圆角大小，通过popupRectCorners设置需要设置圆角的位置。  
 popupRectCorners数组元素不超过四个，超过四个时，只取前四个。比如，要设置左上和右上为圆角，则传值：  
 @[(UIRectCornerTopLeft), @(UIRectCornerTopRight)]  
 */  
@property (nonatomic, strong) NSArray<NSNumber *> *popupRectCorners;  
  
/**  
 * 弹窗动画类型，当popupAnimationStyle为OLAuthPopupAnimationStyleCustom时，动画为用户自定义，用户需要传一个CATransition对象来设置动画  
 */  
@property (nonatomic, assign) OLAuthPopupAnimationStyle popupAnimationStyle;  
  
/**  
 * 弹窗自定义动画  
 */  
@property (nonatomic, strong) CAAnimation *popupTransitionAnimation;
```

```

/**
 弹窗关闭按钮图片，弹窗关闭按钮的尺寸跟图片尺寸保持一致。
 弹窗关闭按钮位于弹窗右上角，目前只支持设置其距顶部偏移和距右边偏移。
 */
@property (nullable, nonatomic, strong) UIImage *closePopupImage;

/**
 弹窗关闭按钮距弹窗顶部偏移。
 */
@property (nonatomic, strong) NSNumber *closePopupTopOffset;

/**
 弹窗关闭按钮距弹窗右边偏移。
 */
@property (nonatomic, strong) NSNumber *closePopupRightOffset;

/**
 是否需要通过点击弹窗的背景区域以关闭授权页面。
 */
@property (nonatomic, assign) BOOL canClosePopupFromTapGesture;

/**
 * 点击授权页面弹窗背景的回调
 */
@property (nonatomic, copy) OLTapAuthBackgroundBlock tapAuthBackgroundBlock;

```

通过设置 isPopup 属性为 YES，即可以弹窗模式弹出授权页面，可自定义弹窗的大小、弹窗弹出动画形式及弹窗圆角，弹窗中各控件大小、位置、属性的设置同上

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置弹窗

```

OLAuthViewModel *viewModel = [OLAuthViewModel new];
viewModel.isPopup = YES;
viewModel.switchButtonHidden = YES;

// -----点击弹窗背景收起弹窗 -----
viewModel.canClosePopupFromTapGesture = YES;

// ----- 弹窗设置 -----

// 自定义弹窗动画
viewModel.popupAnimationStyle = OLAuthPopupAnimationStyleCoverVertical; // 弹窗动画风格，
支持CoverVertical、StyleFlipHorizontal、CrossDissolve和自定义模式，默认为CoverVertical
CATransition *animation = [CATransition animation];

```

```

animation.duration = 1;
animation.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
animation.type = @"rippleEffect";
animation.subtype = kCATransitionFromLeft;
viewModel.popupTransitionAnimation = animation; // 只有在popupAnimationStyle为
OLAuthPopupAnimationStyleCustom时生效

// 弹窗位置、大小设置, 弹窗默认大小为300*340, 居于屏幕中间, 假如要弹窗居于底部, 可做如下设置
CGRect popupRect = {[self ol_screenHeight] - 340, 0, 0, 0, 0, 0, {[self
ol_screenWidth], 340}}; // 弹窗偏移、大小设置
viewModel.popupRect = popupRect;
viewModel.popupCornerRadius = 8; // 弹窗圆角, 默认为6
viewModel.popupRectCorners = @[@(UIRectCornerTopLeft), @(UIRectCornerTopRight)]; // 设
置部分圆角
viewModel.closePopupImage = [UIImage imageNamed:@"back"]; // 关闭按钮
viewModel.closePopupTopOffset = @(3); // 关闭按钮距弹窗顶部偏移
viewModel.closePopupRightOffset = @(-8); // 关闭按钮距弹窗右边偏移

viewModel.tapAuthBackgroundBlock = ^{
    NSLog(@"tapAuthBackgroundBlock");
};

__weak typeof(self) wself = self;
// 在SDK内部预取号未成功时, 建议加载进度条
if (![OneLoginPro isPreGetTokenResultValidate]) {

}
// -----授权页面生命周期回调 -----
viewModel.viewLifeCycleBlock = ^(NSString *viewLifeCycle, BOOL animated) {
    NSLog(@"viewLifeCycle: %@, animated: %@", viewLifeCycle, animated ? @"YES" :
@"NO");
    if ([viewLifeCycle isEqualToString:@"viewDidDisappear:"]) {

    } else if ([viewLifeCycle isEqualToString:@"viewDidLoad"]) {
        // 授权页面出现时, 关掉进度条
    }
};

```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置弹窗

```

let viewModel = OLAuthViewModel()
// 设置为弹窗模式
viewModel.isPopup = true
// 隐藏切换账号按钮
viewModel.switchButtonHidden = true

```

```

// 点击弹窗之外的空白处关闭弹窗
viewModel.canClosePopupFromTapGesture = true

viewModel.tapAuthBackgroundBlock = {
    print("tapAuthBackgroundBlock")
}

viewModel.supportedInterfaceOrientations = UIInterfaceOrientationMask.allButUpsideDown

// ----- 弹窗设置 -----
viewModel.popupAnimationStyle = OLAAuthPopupAnimationStyle.coverVertical

let animation = CATransition.init()
animation.duration = 1
animation.timingFunction = CAMediaTimingFunction.init(name:
CAMediaTimingFunctionName.easeInEaseOut)
animation.type = CATransitionType.init(rawValue: "rippleEffect")
animation.subtype = CATransitionSubtype.fromLeft
viewModel.popupTransitionAnimation = animation

let popupRect = OLRect(portraitTopYOffset: self.ol_screenHeight() - 340,
portraitCenterXOffset: 0, portraitLeftXOffset: 0, landscapeTopYOffset: 0,
landscapeCenterXOffset: 0, landscapeLeftXOffset: 0, size: CGSize(width:
self.ol_screenWidth(), height: 340))
viewModel.popupRect = popupRect
viewModel.popupCornerRadius = 5
viewModel.closePopupTopOffset = NSNumber.init(value: 3) // 关闭按钮距弹窗顶部偏移
viewModel.closePopupRightOffset = NSNumber.init(value: -8)
viewModel.popupRectCorners = [NSNumber.init(value: UIRectCorner.topLeft.rawValue),
NSNumber.init(value: UIRectCorner.topRight.rawValue)]

// -----授权页面生命周期回调 -----
viewModel.viewLifeCycleBlock = { (viewLifeCycle : String, animated : Bool) in
    print("viewLifeCycle: %@, animated: %@", viewLifeCycle, animated ? "true" :
"false")
    if viewLifeCycle == "viewDidDisappear:" {
        sender.isEnabled = true
    } else if viewLifeCycle == "viewDidLoad" {
        // 授权页面出现时, 关掉进度条
        GTPProgressHUD.hideAllHUD()
    }
}

// 进入授权页面

if !OneLoginPro.isPreGetTokenResultValidate() {
}

OneLoginPro.requestToken(with: self, viewModel: viewModel) { [weak self] result in
    if let strongSelf = self {

```

```
}  
}
```

6、授权页面中添加自定义控件

```
/**  
 * @abstract 授权登录页面自定义视图，customAreaView为授权页面的view，如，可将三方登录添加到授权登录页  
面  
 */  
typedef void(^OLCustomUIHandler)(UIView *customAreaView);  
  
/**  
 自定义区域视图的处理block  
  
@discussion  
提供的视图容器使用NSLayoutConstraint与相关的视图进行布局约束。  
如果导航栏没有隐藏，顶部与导航栏底部对齐，左边与屏幕左边对齐，右边与屏幕右边对齐，底部与屏幕底部对齐。  
如果导航栏隐藏，顶部与状态栏底部对齐，左边与屏幕左边对齐，右边与屏幕右边对齐，底部与屏幕底部对齐。  
 */  
@property (nullable, nonatomic, copy) OLCustomUIHandler customUIHandler;
```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 添加自定义控件

```
// ----- 自定义UI设置，如，可以在授权页面添加三方登录入口 -----  
UIButton *customBtn = [[UIButton alloc] initWithFrame:CGRectMake(0, 0, 200, 40)];  
[customBtn setTitle:@"我是自定义UI" forState:UIControlStateNormal];  
customBtn.backgroundColor = [UIColor redColor];  
customBtn.layer.cornerRadius = 2.0;  
[customBtn addTarget:self action:@selector(dismissAuthVC)  
forControlEvents:UIControlEventTouchUpInside];  
__block CGFloat customAreaWidth = 0;  
__block CGFloat customAreaHeight = 0;  
viewModel.customUIHandler = ^(UIView * _Nonnull customAreaView) {  
    [customAreaView addSubview:customBtn];  
    customBtn.center = CGPointMake(customAreaView.bounds.size.width/2,  
customAreaView.bounds.size.height/2 + 150);  
    customAreaWidth = customAreaView.bounds.size.width >  
customAreaView.bounds.size.height ? customAreaView.bounds.size.height :  
customAreaView.bounds.size.width;  
    customAreaHeight = customAreaView.bounds.size.width <  
customAreaView.bounds.size.height ? customAreaView.bounds.size.height :
```



```
customAreaView.bounds.size.width;
};
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 添加自定义控件

```
// ----- 自定义UI设置, 如, 可以在授权页面添加三方登录入口 -----
let customBtn = UIButton.init(frame: CGRect.init(x: 0, y: 0, width: 200, height: 40))
customBtn.setTitle("我是自定义UI", for: UIControl.State.normal)
customBtn.backgroundColor = UIColor.red
customBtn.layer.cornerRadius = 2.0
customBtn.addTarget(self, action: #selector(dismissAuthVC), for:
UIControl.Event.touchUpInside)

var customAreaWidth = 0.0
var customAreaHeight = 0.0

viewModel.customUIHandler = { (customAreaView: UIView) in
    customAreaView.addSubview(customBtn)
    customBtn.center = CGPoint.init(x: customAreaView.bounds.size.width/2, y:
customAreaView.bounds.size.height/2 + 150)
    customAreaWidth = Double(customAreaView.bounds.size.width >
customAreaView.bounds.size.height ? customAreaView.bounds.size.height :
customAreaView.bounds.size.width)
    customAreaHeight = Double(customAreaView.bounds.size.width <
customAreaView.bounds.size.height ? customAreaView.bounds.size.height :
customAreaView.bounds.size.width)
}
```

7、自定义授权页面 loading

```
/**
 * 授权页自定义Loading, 会在点击登录按钮之后触发
 * containerView为loading的全屏蒙版view
 * 请自行在containerView添加自定义loading
 * 设置block后, 默认loading将无效
 */
typedef void(^OLLoadingViewBlock)(UIView *containerView);

/**
 * 停止授权页自定义Loading, 会在调用[OneLogin stopLoading]时触发
 * containerView为loading的全屏蒙版view
 */
typedef void(^OLStopLoadingViewBlock)(UIView *containerView);
```

```

/**
 * 授权页面，自定义加载进度条，点击登录按钮之后的回调
 */
@property (nonatomic, copy, nullable) OLLoadingViewBlock loadingViewBlock;

/**
 * 授权页面，停止自定义加载进度条，调用[OneLogin stopLoading]之后的回调
 */
@property (nonatomic, copy, nullable) OLStopLoadingViewBlock stopLoadingViewBlock;

```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 添加自定义 loading

```

// ----- 授权页面自动旋转时的回调，在该回调中调整自定义视图的frame，若授权页面不支持自动旋
// 转，或者没有添加自定义视图，可不用实现该block -----
viewModel.authVCTransitionBlock = ^(CGSize size,
id<UIViewControllerTransitionCoordinator> _Nonnull coordinator, UIView * _Nonnull
customAreaView) {
    if (size.width > size.height) { // 横屏
        customBtn.center = CGPointMake(customAreaView.height/2, customAreaView.width/2 - 15);
    } else { // 竖屏
        customBtn.center = CGPointMake(customAreaView.width/2, customAreaView.height/2 + 150);
    }
};

// ----- 授权页面点击登录按钮之后的loading设置 -----
viewModel.loadingViewBlock = ^(UIView * _Nonnull containerView) {
    if ([OneLogin isProtocolCheckboxChecked]) {
        UIActivityIndicatorView *indicatorView = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhiteLarge];
        [containerView addSubview:indicatorView];
        indicatorView.center = CGPointMake(containerView.bounds.size.width/2,
containerView.bounds.size.height/2);
        [indicatorView startAnimating];
    }
};

```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 添加自定义 loading

```
// ----- 授权页面点击登录按钮之后的loading设置 -----
viewModel.loadingViewBlock = { (containerView: UIView) in
    if OneLogin.isProtocolCheckboxChecked() {
        let indicatorView = UIActivityIndicatorView.init(style:
UIActivityIndicatorView.Style.whiteLarge)
        containerView.addSubview(indicatorView)
        indicatorView.center = CGPoint.init(x: containerView.bounds.size.width/2, y:
containerView.bounds.size.height/2)
        indicatorView.startAnimating()
    }
}

viewModel.stopLoadingViewBlock = { (containerView: UIView) in
    for subview in containerView.subviews {
        if subview is UIActivityIndicatorView {
            (subview as! UIActivityIndicatorView).stopAnimating()
            subview.removeFromSuperview()
            break
        }
    }
}
```

8、设置服务条款

#pragma mark - CheckBox & Privacy Terms/隐私条款勾选框及隐私条款

```
/**
  授权页面上条款勾选框初始状态。默认 YES。
  */
@property (nonatomic, assign) BOOL defaultCheckBoxState;

/**
  授权页面上勾选框勾选的图标。默认为蓝色图标。推荐尺寸为12x12。
  */
@property (nullable, nonatomic, strong) UIImage *checkedImage;

/**
  授权页面上勾选框未勾选的图标。默认为白色图标。推荐尺寸为12x12。
  */
@property (nullable, nonatomic, strong) UIImage *uncheckedImage;

/**
  授权页面上条款勾选框大小。
  */
@property (nonatomic, assign) CGSize checkBoxSize __attribute__((deprecated("use
checkBoxRect instead.")));
```

```
/**
  授权页面上条款勾选框大小及位置。
 */
@property (nonatomic, assign) CGRect checkBoxRect;

/**
  隐私条款文字属性。默认基础文字灰色，条款蓝色高亮，12pt。
 */
@property (nullable, nonatomic, strong) NSDictionary<NSAttributedStringKey, id>
*privacyTermsAttributes;

/**
  额外的条款。默认为空。
 */
@property (nullable, nonatomic, strong) NSArray<OLPrivacyTermItem *>
*additionalPrivacyTerms;

/**
  服务条款普通文字的颜色。默认灰色。
 */
@property (nullable, nonatomic, strong) UIColor *termTextColor;

/**
  隐私条款 位置及大小，隐私条款，宽需大于50，高需大于20，才会生效。
 */
@property (nonatomic, assign) CGRect termsRect;

/**
  除隐私条款外的其他文案，数组大小必须为4，元素依次为：条款前的文案、条款一和条款二连接符、条款二和条款三连接符、条款后的文案。
  默认为@[@"登录即同意", @"和", @"、", @"并使用本机号码登录"]
 */
@property (nullable, nonatomic, copy) NSArray<NSString *> *auxiliaryPrivacyWords;

/**
  * 点击授权页面隐私协议前勾选框的回调
 */
@property (nullable, nonatomic, copy) OLClickCheckboxBlock clickCheckboxBlock;

/**
  * 服务条款文案对齐方式，默认为NSTextAlignmentLeft
 */
@property (nonatomic, assign) NSTextAlignment termsAlignment;

/**
  * 点击授权页面运营商隐私协议的回调
 */
@property (nullable, nonatomic, copy) OLViewPrivacyTermItemBlock carrierTermItemBlock;

/**
  * 是否在运营商协议名称上加书名号《》
 */
```

```

*/
@property (nonatomic, assign) BOOL hasQuotationMarkOnCarrierProtocol;

/**
 * 未勾选勾选框时，是否禁止一键登录按钮的点击
 */
@property (nonatomic, assign) BOOL disableAuthButtonWhenUnchecked;

/**
 * 未勾选授权页面隐私协议前勾选框时，点击授权页面登录按钮时勾选框与协议的抖动样式，默认不抖动
 */
@property (nonatomic, assign) OLNotCheckProtocolShakeStyle shakeStyle;

```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置服务条款

```

// 隐私条款文字属性
NSMutableParagraphStyle *paragraphStyle = [[NSMutableParagraphStyle alloc] init];
paragraphStyle.lineSpacing = 1.33;
paragraphStyle.alignment = NSTextAlignmentLeft;
paragraphStyle.paragraphSpacing = 0.0;
paragraphStyle.lineBreakMode = NSLineBreakByWordWrapping;
paragraphStyle.firstLineHeadIndent = 0.0;
viewModel.privacyTermsAttributes = @{
    NSForegroundColorAttributeName :
UIColor.orangeColor,
    NSParagraphStyleAttributeName : paragraphStyle,
    NSFontAttributeName : [UIFont
systemFontOfSize:12]
};

// 额外自定义服务条款，注意index属性，默认的index为0，SDK会根据index对多条服务条款升序排列，假如想
设置服务条款顺序为 自定义服务条款1 默认服务条款 自定义服务条款2，则，只需将自定义服务条款1的index设
为-1，自定义服务条款2的index设为1即可
OLPrivacyTermItem *item1 = [[OLPrivacyTermItem alloc] initWithTitle:@"自定义服务条款1"
    linkURL:[NSURL
URLWithString:@"https://docs.geetest.com/onelogin/overview/start"]
    index:0

block:^(OLPrivacyTermItem * _Nonnull termItem, UIViewController *controller) {

    NSLog(@"termItem.termLink: %@, controller: %@", termItem.termLink, controller);
    // 自定义操作，可进入自
    定义服务条款页面

}];

OLPrivacyTermItem *item2 = [[OLPrivacyTermItem alloc] initWithTitle:@"自定义服务条款2"

```

```

linkURL:[NSURL
URLWithString:@"https://docs.geetest.com/"]

index:0];

// 加载本地的html
NSURL *URL = [[NSBundle mainBundle] URLForResource:@"index.html" withExtension:nil];
NSURLRequest *URLRequest = [NSURLRequest requestWithURL:URL];
OLPrivacyTermItem *item3 = [[OLPrivacyTermItem alloc] initWithTitle:@"自定义服务条款3"
                                                                    URLRequest:URLRequest
                                                                    index:0
                                                                    block:nil];

viewModel.additionalPrivacyTerms = @[item1, item2, item3];
OLRect termsRect = {0, 0, 0, 0, 0, 0, {0, 0}}; // 服务条款偏移、大小设置，偏移量和大小设置值
需大于0，否则取默认值，默认可不设置
viewModel.termsRect = termsRect;
viewModel.auxiliaryPrivacyWords = @[@"条款前文案", @"&", @"&", @"条款后的文案"]; // 条款
之外的文案，默认可不设置

viewModel.clickCheckboxBlock = ^(BOOL isChecked) { // 点击隐私条款前勾选框回调
    NSLog(@"clickCheckboxBlock isChecked: %@", isChecked ? @"YES" : @"NO");
};

viewModel.termsAlignment = NSTextAlignmentCenter;
viewModel.shakeStyle = OLNotCheckProtocolShakeStyleHorizontal; // 设置服务条款水平抖动，默认
不抖动

```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置服务条款

```

let paragraphStyle = NSMutableParagraphStyle.init()
paragraphStyle.lineSpacing = 1.33
paragraphStyle.alignment = NSTextAlignment.left
paragraphStyle.paragraphSpacing = 0.0
paragraphStyle.lineBreakMode = NSLineBreakMode.byWordWrapping
paragraphStyle.firstLineHeadIndent = 0.0
viewModel.privacyTermsAttributes = [NSAttributedString.Key.foregroundColor :
UIColor.orange, NSAttributedString.Key.font : UIFont.systemFont(ofSize: 12),
NSAttributedString.Key.paragraphStyle : paragraphStyle]

let item1 = OLPrivacyTermItem.init(title: "自定义服务条款1", linkURL: URL.init(string:
"https://docs.geetest.com/onelogin/overview/start")!, index: 0) { (termItem:
OLPrivacyTermItem, controller: UIViewController) in
    print("termItem.termLink: %@, controller: %@", termItem.termLink, controller)
}
let item2 = OLPrivacyTermItem.init(title: "自定义服务条款2", linkURL: URL.init(string:
"https://docs.geetest.com/")!, index: 0)
// 加载本地的html
let url = Bundle.main.url(forResource: "index.html", withExtension: nil, subdirectory:

```

```

nil)
let item3 = OLPrivacyTermItem.init(title: "自定义服务条款2", linkURL: url!, index: 0)
viewModel.additionalPrivacyTerms = [item1, item2, item3]
let termsRect = OLRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: 0, height: 0)) // 服务条款偏移、大小设置, 偏移量和大小设置值需大于0, 否则取默认值, 默认可不设置
viewModel.termsRect = termsRect
viewModel.auxiliaryPrivacyWords = ["条款前文案", "&", "&", "条款后的文案"] // 条款之外的文案, 默认可不设置

viewModel.clickCheckboxBlock = { (isChecked) in // 点击隐私条款前勾选框回调
    print("clickCheckboxBlock isChecked: %@", isChecked ? "true" : "false")
}

viewModel.termsAlignment = NSTextAlignment.center
viewModel.shakeStyle = .horizontal

```

9、设置授权页面弹出模式

授权页面弹出模式与 iOS UIViewController 的弹出模式保持一致, 支持两种方式弹出授权页面, 默认以模式方式弹出授权页面

```

/**
 * @abstract 进入授权页面的方式, 默认为 modal 方式, 即 present 到授权页面, 从授权页面进入服务条款页面的方式与此保持一致
 *
 * @discussion push 模式时, 不支持弹窗模式, 进入授权页面时, 会隐藏导航栏, 退出授权页面时, 会显示导航栏, 如果您进入授权页面的当前页面导航栏为隐藏状态, 在授权页面消失时, 请注意将导航栏隐藏
 */
typedef NSInteger (NSInteger, OLPullAuthVCStyle) {
    OLPullAuthVCStyleModal,
    OLPullAuthVCStylePush
};

#pragma mark - OLPullAuthVCStyle

/**
 * @abstract 进入授权页面的方式, 默认为 modal 方式, 即 present 到授权页面, 从授权页面进入服务条款页面的方式与此保持一致
 */
@property (nonatomic, assign) OLPullAuthVCStyle pullAuthVCStyle;

```

示例代码

ObjC:

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

2. 设置 pullAuthVCStyle

```
viewModel.pullAuthVCStyle = OLPullAuthVCStylePush; // 默认为 modal
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置 pullAuthVCStyle

```
viewModel.pullAuthVCStyle = OLPullAuthVCStyle.push // 默认为 modal
```

10、设置状态栏

授权页面状态栏已默认适配黑夜模式，若您使用 SDK 默认的授权页面，可以无需设置状态栏，若您需要改变授权页面的背景，请您根据实际情况设置状态栏

```
/**
 * 状态栏样式。 默认 `UIStatusBarStyleDefault`。
 */
@property (nonatomic, assign) UIStatusBarStyle statusBarStyle;
```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置状态栏

```
olAuthViewModel.statusBarStyle = UIStatusBarStyleDefault;
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置状态栏

```
viewModel.statusBarStyle = .lightContent
```

11、授权页在服务条款未选中时的弹窗

#pragma mark - Auth Dialog/授权弹窗

```
/**
 * 未勾选同意协议时是否弹出授权弹窗
 */
@property (nonatomic, assign) BOOL willAuthDialogDisplay;

/**
 * 点击授权弹窗外是否关闭授权弹窗
 */
@property (nonatomic, assign) BOOL canCloseAuthDialogFromTapGesture;

/**
 * 授权弹窗宽、高、起始点位置
 */
@property (nonatomic, assign) CGRect authDialogRect;

/**
 * 授权弹窗是否显示在屏幕下方
 */
@property (nonatomic, assign) BOOL isAuthDialogBottom;

/**
 * 授权弹窗背景颜色
 */
@property (nullable, nonatomic, strong) UIColor *authDialogBgColor;

/**
 * 授权弹窗标题文字
 */
@property (nullable, nonatomic, strong) NSString *authDialogTitleText;

/**
 * 授权弹窗标题颜色
 */
@property (nullable, nonatomic, strong) UIColor *authDialogTitleColor;

/**
 * 授权弹窗字体样式及大小
 */
@property (nullable, nonatomic, strong) UIFont *authDialogTitleFont;

/**
 * 授权弹窗富文本字体样式及大小
 */
@property (nullable, nonatomic, strong) UIFont *authDialogContentFont;

/**
 * 授权弹窗不同意按钮文字
 */
```

```
@property (nullable, nonatomic, strong) NSString *authDialogDisagreeBtnText;

/**
 * 授权弹窗不同意按钮样式及大小
 */
@property (nullable, nonatomic, strong) UIFont *authDialogDisagreeBtnFont;

/**
 * 授权弹窗不同意按钮文字颜色
 */
@property (nullable, nonatomic, strong) UIColor *authDialogDisagreeBtnColor;

/**
 * 授权弹窗不同意按钮的背景图片，@[正常状态的背景图片，高亮状态的背景图片]。默认正常状态为灰色，高亮状态为深灰色。
 */
@property (nullable, nonatomic, strong) NSArray<UIImage *> *authDialogDisagreeBtnImages;

/**
 * 授权弹窗同意按钮文字
 */
@property (nullable, nonatomic, strong) NSString *authDialogAgreeBtnText;

/**
 * 授权弹窗同意按钮样式及大小
 */
@property (nullable, nonatomic, strong) UIFont *authDialogAgreeBtnFont;

/**
 * 授权弹窗同意按钮文字颜色
 */
@property (nullable, nonatomic, strong) UIColor *authDialogAgreeBtnColor;

/**
 * 授权弹窗同意按钮的背景图片，@[正常状态的背景图片，高亮状态的背景图片]。默认正常状态为蓝色纯色，高亮状态为灰蓝色。
 */
@property (nullable, nonatomic, strong) NSArray<UIImage *> *authDialogAgreeBtnImages;

/**
 * 授权弹窗圆角，默认为10。
 */
@property (nonatomic, assign) CGFloat authDialogCornerRadius;

/**
 * 当只需要设置授权弹窗的部分圆角时，通过 authDialogCornerRadius 设置圆角大小，通过 authDialogRectCorners 设置需要设置圆角的位置。
 * authDialogRectCorners 数组元素不超过四个，超过四个时，只取前四个。比如，要设置左上和右上为圆角，则传值：@[(UIRectCornerTopLeft), (UIRectCornerTopRight)]
 */
@property (nonatomic, strong) NSArray<NSNumber *> *authDialogRectCorners;
```

```

/**
 * 授权弹窗动画类型，当authDialogAnimationStyle为OLAuthDialogAnimationStyleCustom时，动画为用户
 * 自定义，用户需要传一个CATransition对象来设置动画
 */
@property (nonatomic, assign) OLAuthDialogAnimationStyle authDialogAnimationStyle;

/**
 * 授权弹窗自定义动画
 */
@property (nonatomic, strong) CAAAnimation *authDialogTransitionAnimation;

/**
 * 点击授权弹窗背景关闭授权弹窗时的回调
 */
@property (nullable, nonatomic, copy) OLTapAuthDialogBackgroundBlock
tapAuthDialogBackgroundBlock;

/**
 * 点击授权弹窗不同意按钮时的回调
 */
@property (nullable, nonatomic, copy) OLClickAuthDialogDisagreeBtnBlock
clickAuthDialogDisagreeBtnBlock;

```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置授权弹窗

```

// 是否显示授权弹窗，不显示的话在未授权时会弹出请同意服务条款提示
viewModel.willAuthDialogDisplay = YES;
// 弹窗是否显示在底部，为 NO 时弹窗会在屏幕中间显示
viewModel.isAuthDialogBottom = YES;
// 弹窗尺寸、圆角大小及位置
OLRect authDialogRect = {0, 0, 0, 0, 0, 0, {[UIScreen mainScreen].bounds.size.width,
340}};
viewModel.authDialogRect = authDialogRect;
viewModel.authDialogCornerRadius = 8;
viewModel.authDialogRectCorners = @[@(UIRectCornerTopLeft), @(UIRectCornerTopRight)];

// 弹窗动画
viewModel.authDialogAnimationStyle = OLAuthDialogAnimationStyleCoverVertical;

// 弹窗标题、内容设置
viewModel.authDialogTitleFont = [UIFont systemFontOfSize:20];
viewModel.authDialogTitleText = @"请同意下述服务条款";
viewModel.authDialogTitleColor = [UIColor blackColor];

```

```

viewModel.authDialogContentFont = [UIFont systemFontOfSize:14];

// 同意按钮设置
viewModel.authDialogAgreeBtnFont = [UIFont systemFontOfSize:16];
viewModel.authDialogAgreeBtnText = @"同意并继续";
viewModel.authDialogAgreeBtnColor = [UIColor whiteColor];

// 不同意按钮设置
viewModel.authDialogDisagreeBtnFont = [UIFont systemFontOfSize:16];
viewModel.authDialogDisagreeBtnText = @"不同意并返回";
viewModel.authDialogDisagreeBtnColor = [UIColor blackColor];

// 是否可以点击弹窗外区域关闭弹窗，默认为 YES
viewModel.canCloseAuthDialogFromTapGesture = NO;
// 点击弹窗外区域关闭弹窗的回调
viewModel.tapAuthDialogBackgroundBlock = ^{
    NSLog(@"tapAuthDialogBackgroundBlock");
};
// 点击不同意按钮关闭弹窗的回调
viewModel.clickAuthDialogDisagreeBtnBlock = ^{
    NSLog(@"clickAuthDialogDisagreeBtnBlock");
};

```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 设置授权弹窗

```

// 是否显示授权弹窗，不显示的话在未授权时会弹出请同意服务条款提示
viewModel.willAuthDialogDisplay = true
// 弹窗是否显示在底部，为 NO 时弹窗会在屏幕中间显示
viewModel.isAuthDialogBottom = true
// 弹窗尺寸、圆角大小及位置
viewModel.authDialogRect = CGRect(portraitTopYOffset: 0, portraitCenterXOffset: 0,
portraitLeftXOffset: 0, landscapeTopYOffset: 0, landscapeCenterXOffset: 0,
landscapeLeftXOffset: 0, size: CGSize(width: UIScreen.main.bounds.size.width, height:
34))
viewModel.authDialogCornerRadius = 8
viewModel.authDialogRectCorners = [NSNumber.init(value:
UIRectCorner.topLeft.rawValue), NSNumber.init(value: UIRectCorner.topRight.rawValue)]

// 弹窗动画
viewModel.authDialogAnimationStyle = .coverVertical

// 弹窗标题、内容设置
viewModel.authDialogTitleFont = UIFont.systemFont(ofSize: 20)
viewModel.authDialogTitleText = "请同意下述服务条款"
viewModel.authDialogTitleColor = UIColor.black
viewModel.authDialogContentFont = UIFont.systemFont(ofSize: 14)

```

```

// 同意按钮设置
viewModel.authDialogAgreeBtnFont = UIFont.systemFont(ofSize: 16)
viewModel.authDialogAgreeBtnText = "同意并继续"
viewModel.authDialogAgreeBtnColor = UIColor.white

// 不同意按钮设置
viewModel.authDialogAgreeBtnFont = UIFont.systemFont(ofSize: 16)
viewModel.authDialogAgreeBtnText = "不同意并返回"
viewModel.authDialogAgreeBtnColor = UIColor.black

// 是否可以点击弹窗外区域关闭弹窗，默认为 true
viewModel.canCloseAuthDialogFromTapGesture = false;
viewModel.tapAuthDialogBackgroundBlock = {
    print("tapAuthDialogBackgroundBlock")
}
// 点击不同意按钮关闭弹窗的回调
viewModel.clickAuthDialogDisagreeBtnBlock = {
    print("clickAuthDialogDisagreeBtnBlock")
}

```

12、自定义授权弹窗

```

/**
 * 未勾选隐私条款时点击授权页面登录按钮的回调，可用于自定义授权弹窗。当返回 YES 时，可以在 block 中添加自定义操作
 */
@property (nullable, nonatomic, copy) OLCustomDisabledAuthActionBlock
customDisabledAuthActionBlock;

```

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 自定义授权弹窗

```

viewModel.customDisabledAuthActionBlock = ^BOOL {
    UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"您必须同意条款才能继续登录" message:nil preferredStyle:UIAlertControllerStyleAlert];

    UIAlertAction *agreeAction = [UIAlertAction actionWithTitle:@"同意"
style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action) {
        // 同意后开始取号
        [OneLoginPro startRequestToken];
    }];
    UIAlertAction *disagreeAction = [UIAlertAction actionWithTitle:@"不同意"

```

```

style:UIAlertActionStyleCancel handler:^(UIAlertAction * _Nonnull action) {
    NSLog(@"用户不同意服务条款");
}];

[alertController addAction:agreeAction];
[alertController addAction:disagreeAction];

dispatch_async(dispatch_get_main_queue(), ^{
    [[OneLoginPro currentAuthViewController] presentViewController:alertController
    animated:YES completion:nil];
});
return YES;
};

```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 自定义授权弹窗

```

viewModel.customDisabledAuthActionBlock = {
    let alertvc = UIAlertController(title: "您必须同意条款才能继续登录", message: nil,
    preferredStyle: .alert)

    let agreeAction = UIAlertAction(title: "同意", style: .default)
    let disagreeAction = UIAlertAction(title: "不同意", style: .default) { (action) in
        print("用户不同意服务条款")
    }
    alertvc.addAction(agreeAction)
    alertvc.addAction(disagreeAction)

    OneLoginPro.currentAuthViewController()?.present(alertvc, animated: true)
    return true
}

```

其他接口说明

1、设置日志开关

设置日志开关，建议开发调试时打开日志，上线时关闭日志

```

/**
 * @abstract 设置是否允许打印日志
 *
 * @param enabled YES, 允许打印日志 NO, 禁止打印日志
 */

```

```
+ (void)setLogEnabled:(BOOL)enabled;
```

2、获取当前网络信息和运营商信息

获取当前网络信息和运营商信息，该接口可判断当前是否开启了 Wifi 和 蜂窝移动网络，并能准确判断当前蜂窝移动网络是 4G、3G 还是 2G，还能判断当前对应的运营商

```
/**
 获取当前 OneLogin 可用的网络信息

  @discussion
  当使用的是非移动、联通、电信三大运营商，则返回nil。
  OneLogin 仅在大陆支持移动、联通、电信三大运营商。

  @seealso
  OLNetworkInfo 中有属性的详细描述
 */
+ (nullable OLNetworkInfo *)currentNetworkInfo;
```

3、设置超时时长

设置请求超时时长

```
/**
 设置请求超时时长。默认时长8s。

  @param timeout 超时时长
 */
+ (void)setRequestTimeout:(NSTimeInterval)timeout;
```

4、服务条款复选框是否勾选

判断服务条款左边复选框是否勾选

```
/**
 * @abstract 服务条款左边复选框是否勾选
 */
+ (BOOL)isProtocolCheckboxChecked;
```

5、获取SDK版本号

获取 SDK 版本号

```
/**
 * 获取SDK版本号
 *
 * @return SDK当前的版本号
 */
+ (NSString *)sdkVersion;
```

6、获取当前授权页面

获取当前授权页面对应的 Controller

```
/**
 * @abstract 获取当前授权页面对应的ViewController
 *
 * @return 当前授权页面对应的ViewController
 */
+ (UIViewController * _Nullable)currentAuthViewController;
```

7、回调接口

7.1、加载授权页面进度条回调

```
/**
 * 授权页自定义Loading，会在点击登录按钮之后触发
 * containerView为loading的全屏蒙版view
 * 请自行在containerView添加自定义loading
 * 设置block后，默认loading将无效
 */
typedef void(^OLLoadingViewBlock)(UIView *containerView);
```

7.2、停止授权页面进度条回调

```
/**
 * 停止授权页自定义Loading，会在调用[OneLogin stopLoading]时触发
 * containerView为loading的全屏蒙版view
 */
typedef void(^OLStopLoadingViewBlock)(UIView *containerView);
```

7.3、授权页面视图生命周期回调


```
/**
 * 授权页面视图生命周期回调
 * @param viewLifeCycle 值为viewDidLoad、viewWillAppear、viewWillDisappear、viewDidAppear、viewDidDisappear
 * @param animated 是否有动画
 */
typedef void(^OLAuthViewLifecycleBlock)(NSString *viewLifeCycle, BOOL animated);
```

7.4、监听授权按钮点击的回调

```
/**
 * 点击授权页面授权按钮的回调，用于监听授权页面登录按钮的点击
 */
typedef void(^OLClickAuthButtonBlock)(void);
```

7.5、点击授权页面勾选框的回调

```
/**
 * 点击授权页面隐私协议前勾选框的回调
 */
typedef void(^OLClickCheckboxBlock)(BOOL isChecked);
```

7.6、点击授权页面弹窗背景的回调

```
/**
 * 点击授权页面弹窗背景的回调
 */
typedef void(^OLTapAuthBackgroundBlock)(void);
```

7.7、授权页面旋转时的回调

```
/**
 * @abstract 授权页面旋转时的回调，可在该回调中修改自定义视图的frame，以适应新的布局
 */
typedef void(^OLAuthVCTransitionBlock)(CGSize size,
id<UIViewControllerTransitionCoordinator> coordinator, UIView *customAreaView);
```

7.8、接管授权按钮点击事件的回调

```
/**
 * 是否自定义授权页面登录按钮点击事件，用于完全接管授权页面点击事件，当返回 YES 时，可以在 block 中添加自
```

定义操作

```
*/  
typedef BOOL(^OLCustomAuthActionBlock)(void);
```

7.9、点击切换账号按钮的回调

```
/**  
 * 点击授权页面切换账号按钮的回调  
 */  
typedef void(^OLClickSwitchButtonBlock)(void);
```

7.10、点击运营商隐私协议的回调

```
typedef void(^OLViewPrivacyTermItemBlock)(OLPrivacyTermItem *termItem, UIViewController  
*controller);  
  
/**  
 * 点击授权页面运营商隐私协议的回调  
 */  
@property (nullable, nonatomic, copy) OLViewPrivacyTermItemBlock carrierTermItemBlock;
```

OnePass(本机号码认证)

注：若只需集成本机号码认证功能，可删除掉

account_login_sdk_noui_core.framework、EAccountApiSDK.framework、OneLoginResource.bundle

1、初始化

方法原型

```
/**  
 Initializes and returns a newly allocated GOPManager object.  
  
 @discussion Register customID from `geetest.com`, and configure your verifyUrl  
 API base on Server SDK. Check Docs on `docs.geetest.com`. If OnePass  
 fail, GOPManager will request SMS URL that you set.  
 @param customID custom ID, nonull  
 @param timeout timeout interval  
 @return A initialized GOPManager object.  
 */  
- (instancetype)initWithCustomID:(NSString * _Nonnull)customID timeout:
```

```
(NSTimeInterval)timeout;
```

参数描述

参数	是否必填	类型	说明
initWithCustomID	是	NSString	appId
timeout	是	NSTimeInterval	请求超时

接口作用

传入 appId，设置请求超时时长使

用场景

- 保证在进行本机号码认证之前至少调用一次

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 在进行本机号码认证之前，初始化，此处使用懒加载方式

```
- (GOPManager *)gopManager {
    if (!_gopManager) {
        _gopManager = [[GOPManager alloc] initWithCustomID:GTOnePassAppId
        timeout:10.f];
        _gopManager.delegate = self;
    }
    return _gopManager;
}
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 在进行本机号码认证之前，初始化，此处使用懒加载方式

```
private lazy var gopManager: GOPManager = {
    let manager = GOPManager.init(customID: GTOnePassAppId, timeout: 10.0)
    manager.delegate = self as GOPManagerDelegate
    return manager
}()
```

2、本机号码认证

方法原型

```
/**
 * Verify phone number through OnePass.
 * See a sample result from `https://github.com/GeeTeam/gop-ios-sdk/blob/master/SDK/gop-ios-dev-doc.md#verifyphonenumcompletionfailure`
 *
 * @param phoneNumber phone number
 */
- (void)verifyPhoneNumber:(NSString * _Nullable)phoneNumber;
```

参数描述

参数	是否必填	类型	说明
phoneNumber	是	NSString	需进行认证的手机号码

接口作用

获取认证传入的手机号码是否为本机号码、是否与本机 SIM 卡一致（对于双卡手机，则认证是否为当前流量对应的本机号码）需要的 token，具体认证需调用服务端接口，传入该接口获取到的 token 进行验证

使用场景

- 初始化之后调用
- 输入手机号后进行认证

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 进行本机号码认证

```
- (void)startOnePass {
    NSString *phoneNumber = self.phoneNumberTF.text;
    phoneNumber = [phoneNumber stringByReplacingOccurrencesOfString:@" "
    withString:@""];
    if ([self checkPhoneNumFormat:phoneNumber]) {
        [GTPProgressHUD showLoadingHUDWithMessage:nil];
        [self.gopManager verifyPhoneNumber:phoneNumber];
    } else {
        self.phoneNumberTF.text = nil;
        [self.phoneNumberTF gtm_shake:9 withDelta:2.5 speed:0.1 completion:nil];
    }
}
```

```

        [GTPProgressHUD showToastWithMessage:@"不合法的手机号"];
    }
}

```

3. 校验 token (accesscode)

```

- (void)gtOnePass:(GOPManager *)manager didReceiveDataToVerify:(NSDictionary *)data {
    NSLog(@"gt onepass received data: %@", data);
    [self stopLoading];
    // TO-DO
    // 提交包含 token 的 data 数据对本机号码校验的结果进行校验
    NSMutableDictionary *mdict = [data mutableCopy];
    mdict[@"id_2_sign"] = YourGTOnePassAppId;
    NSURL *url = [NSURL URLWithString:YourGTOnePassVerifyURL]; // 业务服务器校验接口
    NSMutableURLRequest *req = [NSMutableURLRequest requestWithURL:url];
    req.HTTPMethod = @"POST";
    req.HTTPBody = [NSJSONSerialization dataWithJSONObject:mdict options:0 error:nil];
    NSURLSessionDataTask *task = [NSURLSession.sharedSession dataTaskWithRequest:req
    completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response,
    NSError * _Nullable error) {
        // TO-DO
        // 处理业务服务接口返回的结果
    }];
    [task resume];
}

```

返回数据示例：

```

...
{
    "process_id" : "ec59c065a0a5eea63c58d9ffce194f36", // 流水号
    "accesscode" :
    "CT__0__9396159873d7ccc36f25803b88dda97a__2.5.1.1__nmae890b3719b44aa08a48cb3429227633", //
    accesscode
    "operatorType" : "CT", // 运营商类型 (CM移动/CU联通/CT电信)
    "expiredTime" : 3600, // 校验过期时间, 单位秒
    "phone" : "18012345678" // 待认证的手机号码
}
...

```

4. 处理错误

```

- (void)gtOnePass:(GOPManager *)manager errorHandler:(GOError *)error {
    // TO-DO
    // 处理认证过程中发生的错误

}

```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`

2. 进行本机号码认证

```
func startOnepass() {
    var phoneNumber = self.phoneNumberTF.text
    phoneNumber = phoneNumber?.replacingOccurrences(of: " ", with: "")
    if self.checkPhoneNumFormat(phoneNumber) {
        GTPProgressHUD.showLoadingHUD(withMessage: nil)
        self.gopManager.verifyPhoneNumber(phoneNumber)
    } else {
        self.phoneNumberTF.text = nil
        self.phoneNumberTF.gtm_shake(9, withDelta: 2.0, speed: 0.1, completion: nil)
        GTPProgressHUD.showToast(withMessage: "不合法的手机号")
    }
}
```

3. 校验 token (accesscode)

```
func gtOnePass(_ manager: GOPManager, didReceiveDataToVerify data: [AnyHashable : Any]) {
    print("gt onepass received data: \(data)")
    stopLoading()
    // TO-DO
    // 提交包含 token 的 data 数据对本机号码校验的结果进行校验
    var params = data
    params["id_2_sign"] = YourGTOnePassAppId
    do {
        let data = try JSONSerialization.data(withJSONObject: params, options:
JSONSerialization.WritingOptions.fragmentsAllowed)
        let url = URL.init(string: YourGTOnePassVerifyURL)
        var mRequest = URLRequest.init(url: url!, cachePolicy:
NSURLRequest.CachePolicy.useProtocolCachePolicy, timeoutInterval: 10.0)
        mRequest.httpMethod = "POST"
        mRequest.httpBody = data
        let dataTask = URLSession.shared.dataTask(with: mRequest) { [weak self] (data:
Data?, response: URLResponse?, error: Error?) in
            // TO-DO
            // 处理业务服务接口返回的结果
        }
        dataTask.resume()
    } catch {
    }
}
```

返回数据示例：

```
...
{
  "process_id" : "ec59c065a0a5eea63c58d9ffce194f36", // 流水号
  "accesscode" :
  "CT__0__9396159873d7ccc36f25803b88dda97a__2.5.1.1__nmae890b3719b44aa08a48cb3429227633", //
  accesscode
  "operatorType" : "CT", // 运营商类型（CM移动/CU联通/CT电信）
  "expiredTime" : 3600, // 校验过期时间，单位秒
  "phone" : "18012345678" // 待认证的手机号码
}
...
```

4. 处理错误

```
func gtOnePass(_ manager: GOPManager, errorHandler error: GOPError) {
    // TO-DO
    // 处理认证过程中发生的错误

}
```

错误码请参考 [OnePass iOS 错误清单](#)

3、配置是否缓存手机号到本地

方法原型

```
/**
 * @abstract 设置是否允许缓存手机号，若允许缓存，则将校验过的手机号加密之后缓存到沙盒，缓存中仅存最近一次
 * 校验过的手机号，默认允许缓存
 *
 * @param enabled YES，允许缓存 NO，禁止并清空缓存
 */
+ (void)setCachePhoneEnabled:(BOOL)enabled;
```

参数描述

参数	是否必填	类型	说明
enabled	否	Bool	是否缓存手机号，默认值为 YES

接口作用

初始化 `GOPManager` 之前调用该接口配置是否开启缓存手机号功能。配置开启后，每次调用

`verifyPhoneNumber:` 方法时，会加密缓存当前传入的手机号到本地，用于减少用户登录页重复输入手机号流程，提升登录效率；配置关闭时，则会清除本地缓存的手机号，并且调用 `verifyPhoneNumber:` 方法时，不会缓存手机号。

使用场景

- 初始化之前调用

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 配置是否开启缓存手机号功能

```
[GOPManager setCachePhoneEnabled:YES];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 配置是否开启缓存手机号功能

```
GOPManager.setCachePhoneEnabled(true)
```

注意：手机号缓存功能默认开启，配置开启后，手机号仅加密缓存到本地，每次 `verifyPhoneNumber:` 后都会覆盖更新，不会搜集与上传。

4、获取缓存在本地的手机号

方法原型

```
/**
 * @abstract 获取最近一次缓存的手机号
 *
 * @return 手机号
 */
+ (NSString * _Nullable)getCachedPhone;
```

参数描述

返回参数	是否必填	类型	说明
手机号	否	NSString	缓存在本地的手机号

接口作用

开启缓存手机号功能后，每次进入登录页面，可调用该方法获取最近一次调用 `verifyPhoneNumber:` 时缓存的手机号。未开启手机号缓存、首次调用、内部解密失败或其他异常时，返回结果为 `nil`。

使用场景

- 进入登录页面时调用，获取手机号并填充在手机号输入框

示例代码

ObjC:

- 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
- 获取缓存在本地的手机号

```
// 获取缓存在本地的手机号
NSString *phone = [GOPManager getCachedPhone];
if (phone.length > 0) {
    self.phoneNumberTF.text = phone;
}
```

Swift:

- 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
- 获取缓存在本地的手机号

```
// 获取缓存在本地的手机号
if let phone = GOPManager.getCachedPhone(), phone.count > 0 {
    self.phoneNumberTF.text = phone
}
```

5、获取缓存在本地的手机号列表

方法原型

```
/**
 * @abstract 获取缓存的手机号列表
 *
 * @param completionHandler 手机号列表
 */
+ (void)getCachedPhonesWithCompletionHandler:(void(^)(NSMutableArray<NSString *> *phones))completionHandler;
```

参数描述

返回参数	是否必填	类型	说明
手机号列表	否	NSMutableArray<NSString *>	缓存在本地的手机号列表

接口作用

开启缓存手机号功能后，每次进入登录页面，可调用该方法获取缓存的手机号列表。

使用场景

- 输入框输入号码时，弹出关联手机号列表

示例代码

ObjC:

1. 导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 获取缓存在本地的手机号

```
// 获取缓存手机号列表
[GOPManager getCachedPhonesWithCompletionHandler:^(NSMutableArray<NSString *> *
_Nonnull phones) {
    if (phones.count > 0) {
        dispatch_async(dispatch_get_main_queue(), ^{
            if ((!self.phoneTextField.text || 0 == self.phoneTextField.text.length) &&
phones[0].length > 0) {
                self.phoneTextField.text = phones[0];
                [self textFieldEditingChanged:self.phoneTextField];
            }
        });
    }
}];
```

Swift:

1. 创建混编桥接头文件并导入 SDK 头文件 `#import <OneLoginSDK/OneLoginSDK.h>`
2. 获取缓存在本地的手机号

```
// 获取缓存手机号列表
GOPManager.getCachedPhones { (phones) in
    if phones.count > 0 {
        DispatchQueue.main.async {
            if let phone = phones.firstObject as? String {
                self.phoneNumberTF.text = phone
            }
        }
    }
}
```

```
}  
  }  
}  
}
```

插件资源

客户端集成插件（除了原生的 iOS SDK 之外，OneLogin 提供主流的开发工具集成插件）

Flutter 插件

官方已提供 Flutter 官方插件：

在工程 `pubspec.yaml` 中 `dependencies` 块中添加下列配置

Github 集成

```
dependencies:  
  gt_onelogin_flutter_plugin:  
    git:  
      url: https://github.com/GeeTeam/gt_onelogin_flutter_plugin.git  
      ref: master
```

或

pub 集成

```
dependencies:  
  gt_onelogin_flutter_plugin: ^0.0.1
```

[Github:gt_onelogin_flutter_plugin](#)

其他插件

React Native: <https://github.com/GeeTeam/gt-onelogin-rn-example>

Uniapp: <https://github.com/GeeTeam/OneLoginUniappPlugin>

Unity: <https://github.com/GeeTeam/gt-onelogin-unity-example>

更详细示例代码见相应 Demo

更详细的接口说明见 SDK 头文件