**统一认证IOS集成指南1.0**

中国移动统一认证中间件面向客户端应用／插件提供用户认证服务。用户在终端设备上，一次认证即可自由登录所有业务。在插有中国移动SIM卡的终端上，中间件可以利用网络和SIM卡能力提供自动登录服务，登录过程对终端用户来说几乎无感知。

终端应用／插件可通过集成统一认证SDK调用中间件能力，减少开发者的开发难度。

目录

[目录 2](#_Toc482793564)

[统一认证iOS开发指南1.0 4](#_Toc482793565)

[1. 简介 4](#_Toc482793566)

[1.1. 什么是统一认证iOS SDK 4](#_Toc482793567)

[1.2. 本文面向的读者 4](#_Toc482793568)

[1.3. 兼容性 4](#_Toc482793569)

[2. 申请密钥 4](#_Toc482793570)

[2.1. 获取sourceid 5](#_Toc482793571)

[2.2. 进入申请地址获取appid和appkey 5](#_Toc482793572)

[3. 环境配置及发布 5](#_Toc482793573)

[4. Hello 统一认证 7](#_Toc482793574)

[4.1. 统一认证登录流程 7](#_Toc482793575)

[4.2. 统一认证自动登录集成步骤 8](#_Toc482793576)

[4.3. 统一认证手动登录集成步骤 9](#_Toc482793577)

[4.4. 统一认证账号注册集成步骤 10](#_Toc482793578)

[4.5. 统一认证密码重置集成步骤 11](#_Toc482793579)

[4.6. 统一认证密码修改集成步骤 12](#_Toc482793580)

[5. 接口说明 13](#_Toc482793581)

[5.1. SDK接口说明 13](#_Toc482793582)

[5.1.1. 使用IDMPAutoLoginViewController.h中的validateWithAppid方法 13](#_Toc482793583)

[5.1.2. 使用IDMPAutoLoginViewController.h中的getAccessToken方法 14](#_Toc482793584)

[5.1.3. 使用IDMPAutoLoginViewController.h中的getAppPassword方法 15](#_Toc482793585)

[5.1.4. 使用IDMPAutoLoginViewController.h中的getAccessTokenByCondition方法 17](#_Toc482793586)

[5.1.5. 使用IDMPAutoLoginViewController.h中的getAppPasswordByCondition方法 18](#_Toc482793587)

[5.1.6. 使用IDMPTempSmsMode.h中的getSmsCode方法 19](#_Toc482793588)

[5.1.7. 使用IDMPAutoLoginViewController.h中的cleanSSO方法 20](#_Toc482793589)

[5.1.8. 使用IDMPAutoLoginViewController.h中的registerUser方法 21](#_Toc482793590)

[5.1.9. 使用IDMPAutoLoginViewController.h中的resetPassword方法 22](#_Toc482793591)

[5.1.10. 使用IDMPAutoLoginViewController.h中的changePassword方法 23](#_Toc482793592)

[5.1.11. 使用IDMPAutoLoginViewController.h中的checkIsLocalNumber方法 24](#_Toc482793593)

[5.1.12. 使用IDMPAutoLoginViewController.h中的getAuthType方法 24](#_Toc482793594)

[5.1.13. 使用IDMPAutoLoginViewController.h中的currentEdition方法 25](#_Toc482793595)

[5.2. 接口示例 25](#_Toc482793596)

[5.2.1. 使用IDMPAutoLoginViewController中的validateWithAppid方法 25](#_Toc482793597)

[5.2.2. 使用IDMPAutoLoginViewController中的getAccessToken方法 26](#_Toc482793598)

[5.2.3. 使用IDMPAutoLoginViewController中的getAccessPassword方法 27](#_Toc482793599)

[5.2.4. 使用IDMPAutoLoginViewController中的getAccessTokenByCondition方法 27](#_Toc482793600)

[5.2.5. 使用IDMPAutoLoginViewController中的getAccessPasswordByCondition方法 28](#_Toc482793601)

[5.2.6. 使用IDMPTempSmsMode.h中的getSmsCode方法 28](#_Toc482793602)

[5.2.7. 使用IDMPAutoLoginViewController.h中的cleanSSO方法 29](#_Toc482793603)

[5.2.8. 使用IDMPAutoLoginViewController.h中的registerUser方法 29](#_Toc482793604)

[5.2.9. 使用IDMPAutoLoginViewController.h中的resetPassword方法 30](#_Toc482793605)

[5.2.10. 使用IDMPAutoLoginViewController.h中的changePassword方法 30](#_Toc482793606)

[5.2.11. 使用IDMPAutoLoginViewController.h中的checkIsLocalNumber方法 31](#_Toc482793607)

[5.2.12. 使用IDMPAutoLoginViewController.h中的getAuthType方法 31](#_Toc482793608)

[5.2.13. 使用IDMPAutoLoginViewController.h中的currentEdition方法 32](#_Toc482793609)

[5.3. SDK错误码说明 32](#_Toc482793610)

[5.3.1. 返回JSON说明 32](#_Toc482793611)

[5.3.2. 错误码详述 32](#_Toc482793612)

# 统一认证iOS开发指南1.0

# 简介

## 什么是统一认证iOS SDK

统一认证 iOS SDK是一套基于iOS 7.0及以上版本设备的应用程序接口， 您可以通过该接口实现丰富的注册、登录等功能。

自动登录（一键登录）：无需用户输入用户名密码，自动完成本机号码登录。

手工登录：用户使用短信验证码或用户名密码实现登录。

账号管理：实现注册、密码修改、重置等一系列账号管理功能。

## 本文面向的读者

统一认证SDK是提供给具有一定iOS编程经验和了解面向对象概念的读者使用。您在使用中遇到任何问题，都可以通过对应接口人反馈给我们。

## 兼容性

支持iOS7.0 及以上系统。

# 申请密钥

在使用统一认证SDK为您提供的各种账号管理能力之前，您需要获取统一认证iOS版的开发密钥，该密钥与您的身份及创建的应用有关。

## 获取sourceid

您需要发一封邮件给杭研安全产品部郑廷法（zhengtingfa@chinamobile.com）申请sourceid，邮件内容包括：业务名称、业务介绍、业务管理接口人/公司/部门/联系方式、业务开发接口人/公司/部门/联系方式、是否与其他APP有关联、是否需要开通SIP应用、业务规模（预计用户数规模、并发数、日活）、集成计划（计划开始时间、完成时间）。

## 进入申请地址获取appid和appkey

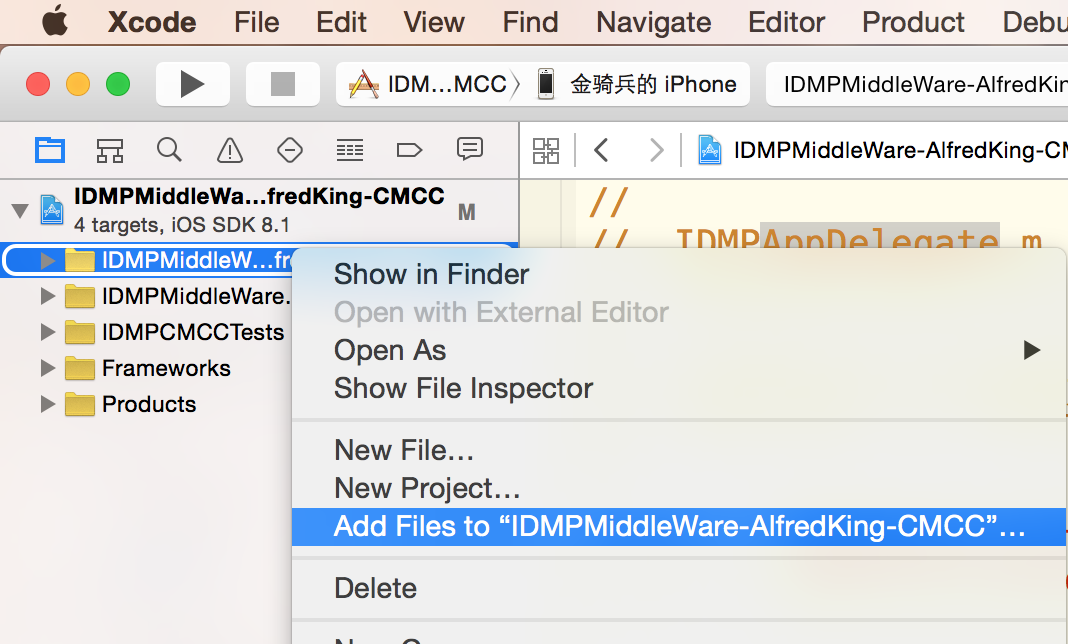


请务必记录好您的APPID和APPKEY，并妥善保管。到此您就可以使用独有的APPID和APPKEY来开始您的开发工作了。

# 环境配置及发布

1. **导入统一认证中间件静态库**

统一认证中间件为libIDMPCMCC.a,在使用静态库的工程之中，使用Add Files to方法将静态库添加到工程之中，同时添加如下头文件，IDMPAutoLoginViewController.h，IDMPTempSmsMode.h。并添加资源文件IDMPCMCC.bundle。如下图所示。

****

1. **添加CoreTelephony.framework**

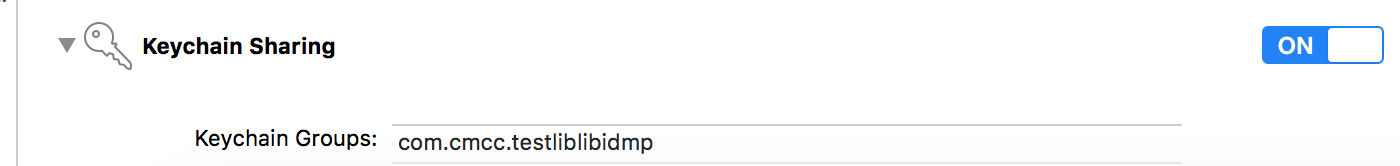
因为要读取SIM卡状态等信息，需要导入CoreTelephony.framework(从IOS7开始为公有框架)。

1. **设置Build Setting**

在你的工程文件中选择Build Setting，在“Other Linker Flags”中加入“-Objc –all\_load”或者“-Objc –force\_load libIDMPCMCC.a路径”。

1. **设置共享**

我们的数据会在keychain中存储，如果你想在同一开发账户下的app登录信息共享，可以开启Capabilities中的Keychain Sharing。

****

# Hello 统一认证

本节内容主要面向新接入统一认证的开发者，介绍快速集成统一认证的基本服务的方法。

## 统一认证登录流程

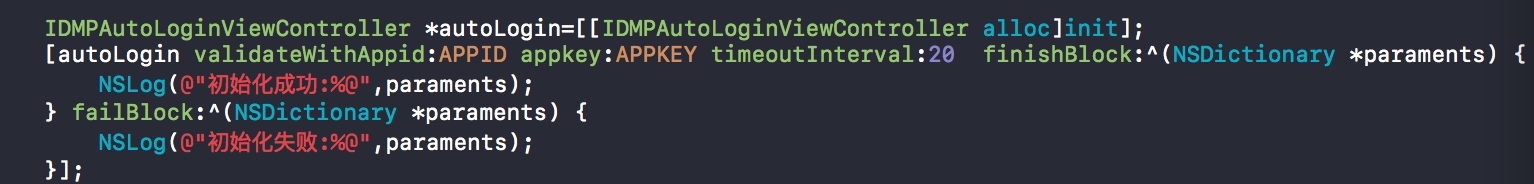


1. 调用getAccessToken接口用于自动登录（基于中移的WAP网关插号码能力及上行短信验证能力，用户可以在无感知，或一键点击的情况下，直接完成本机号码的身份认证并进行登录）。如果该账号Ks存在，则直接签发token返回，如果Ks不存在，sdk会与统一认证平台协商Ks，再签发token返回。
2. 如果getAccessToken接口自动登录失败，则可以调用getAccessTokenByCondtion接口进行手动登录拿到token。
3. 当拿到token后，客户端再携带token进行token验证。验证成功后就登录成功。

## 统一认证自动登录集成步骤

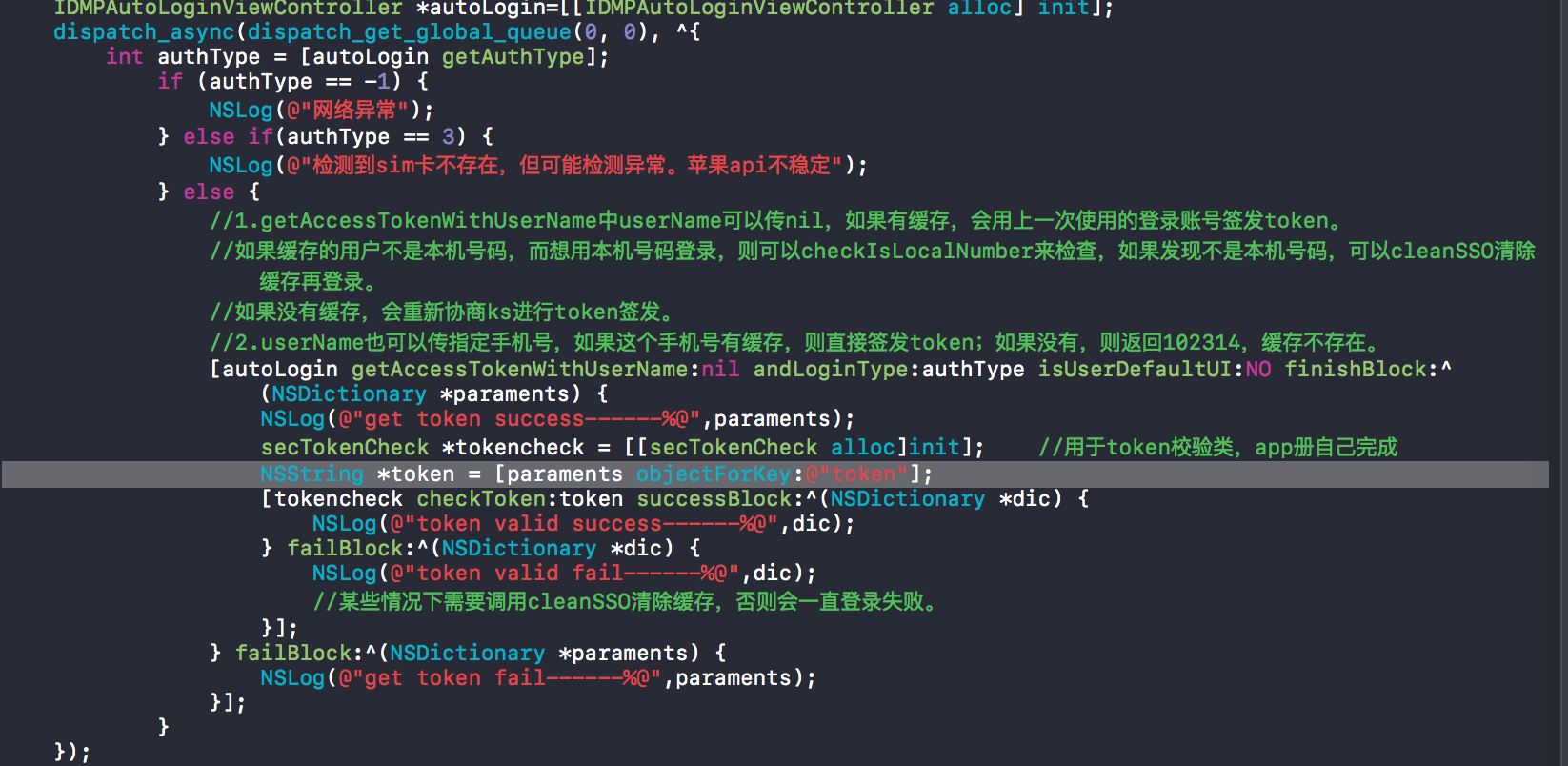
**第一步：**

在appDelegate.m中的didFinish函数中添加初始化IDMP代码。



**第二步：**

在需要用到登录的viewController添加一键登录代码。由于是自动登录，因此getAccessToken接口中userName参数传nil，而loginType可以根据自己的网络状况来传入， 其中SDK提供了getAuthType接口来判断当前的网络状况。当获得token后则可以自己发起token校验请求来验证token，校验成功后完成登录。校验失败时可能需要调用cleanSSO来清除缓存。



## 统一认证手动登录集成步骤

**第一步：**

初始化因为在自动登录中已完成，所以不需要重复做。

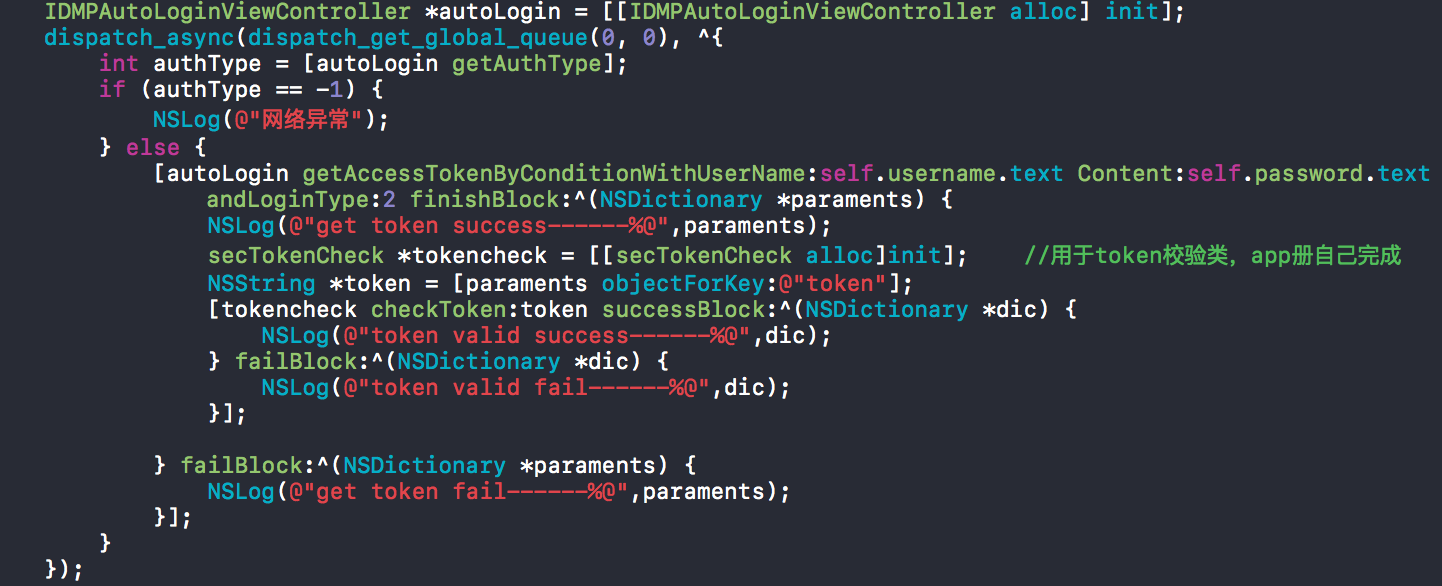
**第二步：**

手动登录分为固定密码登录和短信验证码登录。通过loginType来区分不同的登录方式。

固定密码登录。直接调用getAccessTokenByCondition接口进行手动登录。其中LoginType传1。拿到token后的处理流程与自动登录相同。



短信验证码登录。先调用IDMPTempSmsMode中的getSMSCode接口来获取登录临时短信验证码，其中LoginType传3。然后再调用getAccessTokenByCondition接口进行手动登录。其中LoginType传2。拿到token后的处理流程与自动登录相同。



## 统一认证账号注册集成步骤

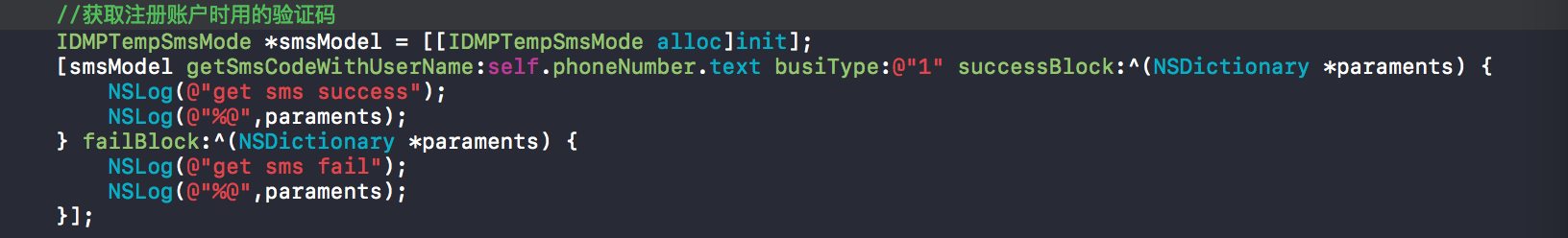


**第一步：**

初始化因为在自动登录中已完成，所以不需要重复做。

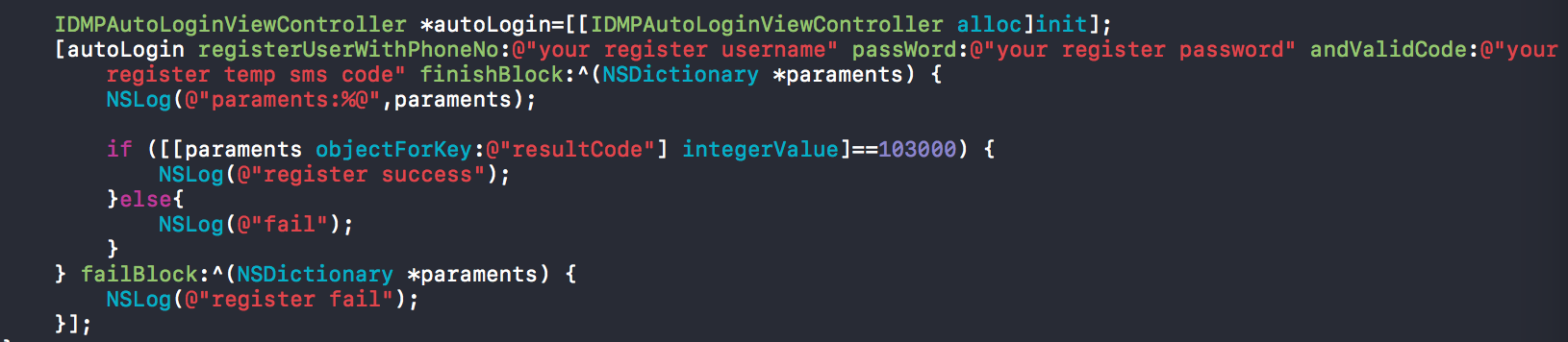
**第二步：**

调用IDMPTempSmsMode中的getSMSCode接口来获取注册临时短信验证码，其中LoginType传1。



**第三步：**

调用IDMPAutoLoginViewController中的registerUser接口来注册用户。



## 统一认证密码重置集成步骤

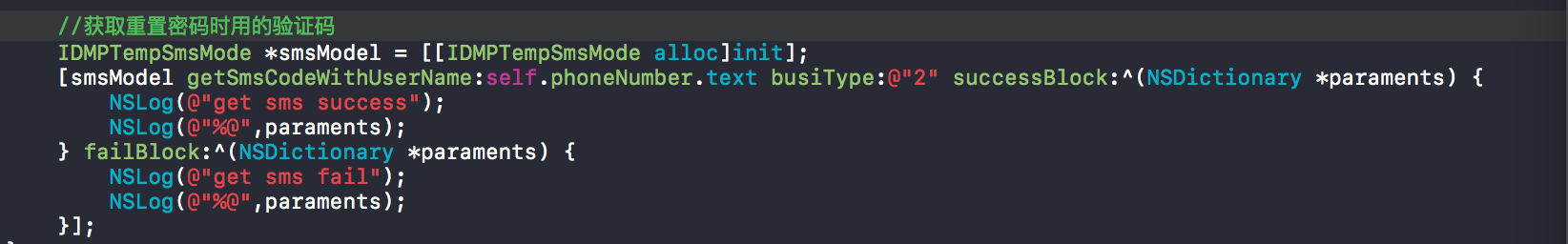


**第一步：**

初始化因为在自动登录中已完成，所以不需要重复做。

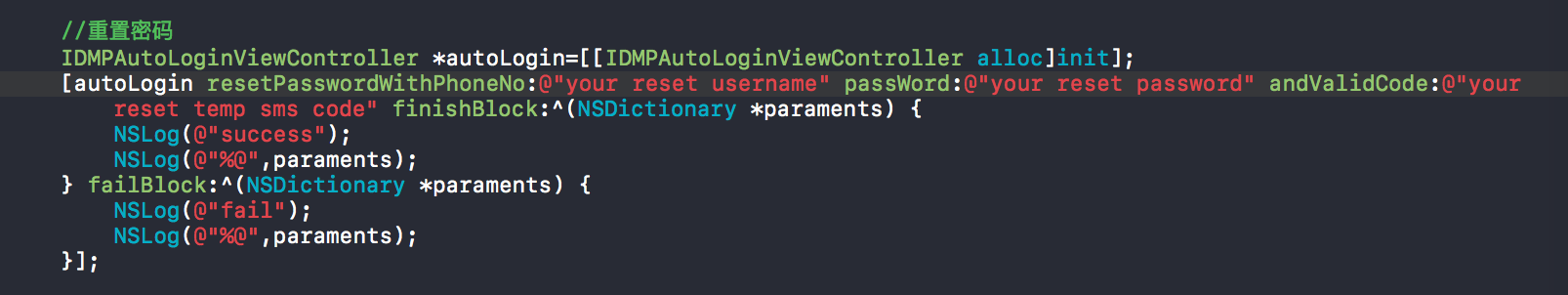
**第二步：**

调用IDMPTempSmsMode中的getSMSCode接口来获取重置密码时的临时短信验证码，其中LoginType传2。



**第三步：**

调用IDMPAutoLoginViewController中的resetPassword接口来重置密码。

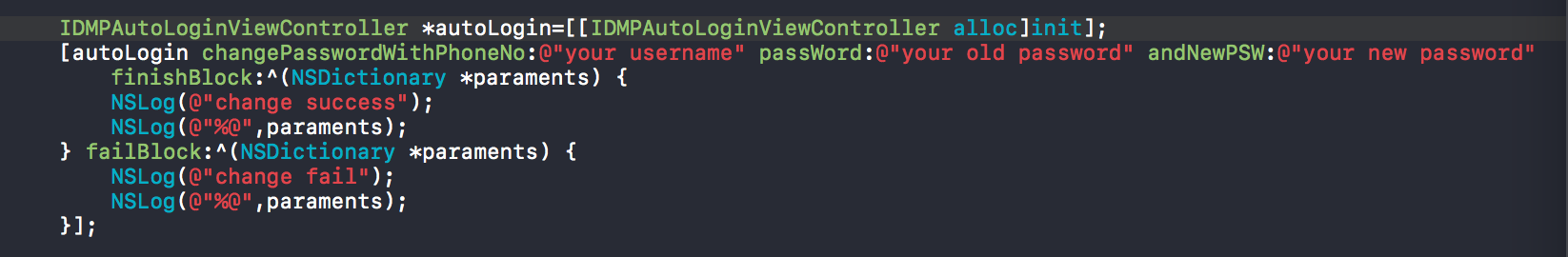


## 统一认证密码修改集成步骤

**第一步：**

初始化因为在自动登录中已完成，所以不需要重复做。

**第二步：**

调用IDMPAutoLoginViewController中的changePassword接口来修改密码。

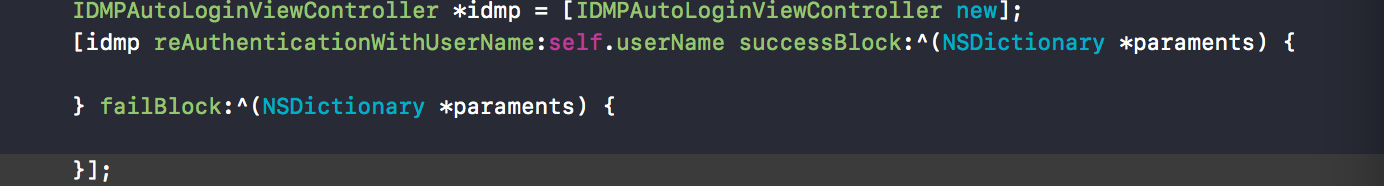
## 统一认证二次鉴权集成步骤

**第一步：**

初始化因为在自动登录中已完成，所以不需要重复做。

**第二步：**

调用IDMPAutoLoginViewController中的reAuthentication接口来二次鉴权。



# 接口说明

## SDK接口说明

### 使用IDMPAutoLoginViewController.h中的validateWithAppid方法

完整函数原型:

- (void)validateWithAppid:(NSString \*)appid appkey:(NSString \*)appkey timeoutInterval:(float)aTime finishBlock:(accessBlock)successBlcok failBlock:(accessBlock)failBlock;

接口用于接口用于初始化appid、appkey、设置。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| appid | NSString | 应用的appid |
| appkey | NSString | 应用密钥 |
| aTime | float | 上行数据短信请求超时时间（0，20） |
| isprint | BOOL | 是否打印log日志到本地文件 |
| successBlcok | accessBlock | 初始化成功之后使用的block方法，由应用开发者自行实现。 |
| successBlcok | accessBlock | 初始化失败之后使用的block方法，由应用开发者自行实现 |

返回值说明

successBlcok的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **含义** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |

### 使用IDMPAutoLoginViewController.h中的getAccessToken方法

完整函数原型:

-(void)getAccessTokenWithUserName:(NSString \*)userName

andLoginType:(NSUInteger) loginType

isUserDefaultUI:(BOOL)isUserDefaultUI finishBlock:(accessBlock) successBlock

failBlock:(accessBlock )failBlock;

接口用于获取统一认证的身份标识；如果为sip应用需要返回应用密码则调用getAppPassword接口来完成。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| userName | NSString | 指定用户登录时的用户名 |
| loginType | NSUInteger | 可以指定登录方式，0表示wifi和蜂窝网络同时开启时使用wap方式，1表示蜂窝网络方式，2表示数据短信方式 |
| isUserDefaultUI | BOOL | 目前只能用NO |
| successBlcok | accessBlock | 登录成功之后使用的block方法，由应用开发者自行实现。 |
| failBlock | accessBlock | 登录失败之后使用的block方法，由应用开发者自行实现 |

返回值说明

successBlcok的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **含义** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |
| token | NSString | 身份标识，字符串形式的token，第三方应用将该凭证经应用平台向IDMP平台请求认证；  第三方应用为非SIP应用时返回（此时不返回password） |
| passid | NSString | 和通行证ID |

### 使用IDMPAutoLoginViewController.h中的getAppPassword方法

接口原型：

- (void)getAppPasswordWithUserName:(NSString \*)username

andLoginType:(NSUInteger)loginType isUserDefaultUI:(BOOL)isUserDefaultUI finishBlock:(accessBlock)successBlock failBlock:(accessBlock)failBlock;

接口用于获取融合通信SIP应用的应用密码（包括身份标示Token）；对于不使用自定义界面的应用，使用该接口获取token。

备注：只有第三方应用APK是SIP应用时，才会返回应用密码，否则只能返回Token。因此对于融合通信APP，需先确定获取token的应用是非SIP应用还是SIP应用，再确定需要调用的接口。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| userName | NSString | 指定用户登录时的用户名 |
| loginType | NSUInteger | 可以指定登录方式，0表示wifi和蜂窝网络同时开启时使用wap方式，1表示蜂窝网络方式，2表示数据短信方式 |
| isUserDefaultUI | BOOL | 目前只能用NO |
| successBlcok | accessBlock | 登录成功之后使用的block方法，由应用开发者自行实现。 |
| failBlock | accessBlock | 登录失败之后使用的block方法，由应用开发者自行实现 |

返回值说明

successBlcok的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **含义** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |
| token | NSString | 身份标识，字符串形式的token，第三方应用将该凭证经应用平台向IDMP平台请求认证 |
| password | NSString | SIP密码 |
| userName | NSString | 所登录的用户名 |
| passid | NSString | 和通行证ID |
| openid | NSString | 伪码 |

### 使用IDMPAutoLoginViewController.h中的getAccessTokenByCondition方法

函数原型为：

- (void) getAccessTokenByConditionWithUserName:(NSString\*)username Content:(NSString\*)content andLoginType:(NSUInteger)loginType finishBlock:(accessBlock)successBlock failBlock:(accessBlock )failBlock;

接口用于获取统一认证的身份标识；

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| userName | NSString | 登录的用户名 |
| content | NSString | 认证方式为1时值为用户密码  认证方式为2时值为短信验证码 |
| loginType | NSUInteger | 选择认证方式：  1表示使用固定密码登录  2表示使用临时密码登录 |
| successBlcok | accessBlock | 登录成功之后执行的block方法，有调用者自行实现 |
| failBlock | accessBlock | 登录失败之后执行的block方法，有调用者自行实现 |

返回值说明

successBlcok的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **含义** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |
| token | NSString | 身份标识，字符串形式的token，第三方应用将该凭证经应用平台向IDMP平台请求认证；  第三方应用为非SIP应用时返回（此时不返回password） |
| passid | NSString | 和通行证ID |

### 使用IDMPAutoLoginViewController.h中的getAppPasswordByCondition方法

函数原型为：

- (void)getAppPasswordByConditionWithUserName:(NSString

\*)userName Content:(NSString \*)content andLoginType:(NSUInteger)loginType finishBlock:(accessBlock)successBlock failBlock:(accessBlock)failBlock;

接口用于获取统一认证的应用密码（同时返回Token身份标示）；对于使用自定义界面的客户端应用，在自动登录失败后，使用该接口进行用户名密码/临时密码登录。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| userName | NSString | 登录的用户名 |
| content | NSString | 认证方式为1时值为用户密码  认证方式为2时值为短信验证码 |
| loginType | NSUInteger | 选择认证方式：  1表示使用固定密码登录  2表示使用临时密码登录 |
| successBlcok | accessBlock | 登录成功之后执行的block方法，有调用者自行实现 |
| failBlock | accessBlock | 登录失败之后执行的block方法，有调用者自行实现 |

返回值说明

successBlcok的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **含义** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |
| token | NSString | 身份标识，字符串形式的token，第三方应用将该凭证经应用平台向IDMP平台请求认证；  第三方应用为非SIP应用时返回（此时不返回password） |
| password | NSString | sip密码 |
| userName | NSString | 所登录的用户名 |
| passid | NSString | 和通行证ID |
| openid | NSString | 伪码 |

### 使用IDMPTempSmsMode.h中的getSmsCode方法

函数原型：

-(void)getSmsCodeWithUserName:(NSNSString\*)username

busiType:(NSString \*)busiType

successBlock:(accessBlock)successBlock

failBlock:(accessBlock)failBlock;

接口用于第三方应用使用自定义登录界面，通过临时密码登录方式登录时，获取临时登录密码使用。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| username | NSString | 用户手机号 |
| busiType | NSString | 业务类型;1为获取注册验证码；2为获取重置密码验证码；3为临时短信登录验证码。 |
| successBlock | accessBlock | 接收临时短信成功时调用，由应用开发者自行实现。 |
| failBlock | accessBlock | 接收临时短信失败时调用，由应用开发者自行实现。 |

返回值说明

successBlcok, failBlock的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |

### 使用IDMPAutoLoginViewController.h中的cleanSSO方法

函数原型：

- (BOOL) cleanSSO;

接口用于第三方应用在Token验证失败时调用此方法清理本地环境使用。此方法会将缓存的所有用户名信息从手机之中清除，或者可以调用-(BOOL) cleanSSOWithUserName:(NSNSString \*)userName用于将缓存的指定用户的信息清除。

返回值说明

YES，清除成功；NO，清除失败。

### 使用IDMPAutoLoginViewController.h中的registerUser方法

接口原型：

-(void)registerUserWithhoneNo:(NSString \*)phoneNo

passWord:(NSString \*)password

andValidCode:(NSString \*)validCode

finishBlock:(accessBlock) successBlock

failBlock:(accessBlock )failBlock;

接口用于第三方应用进行用户注册，本接口仅支持手机号码注册。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| phoneNo | NSString | 手机号码； |
| password | NSString | 注册用户的密码； |
| validCode | NSString | 短信验证码； |
| successBlock | accessBlock | 注册用户成功时调用，由应用开发者自行实现。 |
| failBlock | accessBlock | 注册用户失败时调用，由应用开发者自行实现。 |

返回值说明

successBlcok, failBlock的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |

### 使用IDMPAutoLoginViewController.h中的resetPassword方法

接口原型：

-(void)resetPasswordWithhoneNo:(NSString \*)phoneNo

passWord:(NSString \*)password

andValidCode:(NSString \*)validCode

finishBlock:(accessBlock) successBlock

failBlock:(accessBlock )failBlock;

接口用于第三方应用重置密码。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| phoneNo | String | 手机号码； |
| password | String | 重置后的新密码； |
| validCode | String | 短信验证码； |
| successBlock | accessBlock | 重置密码成功时调用，由应用开发者自行实现。 |
| failBlock | accessBlock | 重置密码失败时调用，由应用开发者自行实现。 |

返回值说明

successBlcok, failBlock的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |

流程图：

### 使用IDMPAutoLoginViewController.h中的changePassword方法

接口原型：

-(void)changePasswordWithhoneNo:(NSString \*)phoneNo

passWord:(NSString \*)password

andNewPSW:(NSString \*)newpassword

finishBlock:(accessBlock) successBlock

failBlock:(accessBlock )failBlock;

接口用于第三方应用密码修改。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| phoneNo | NSString | 手机号码； |
| password | NSString | 旧密码； |
| newPassword | NSString | 新密码 |
| successBlock | accessBlock | 修改密码成功时调用，由应用开发者自行实现。 |
| failBlock | accessBlock | 修改密码失败时调用，由应用开发者自行实现。 |

返回值说明

successBlcok, failBlock的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |

### 使用IDMPAutoLoginViewController.h中的checkIsLocalNumber方法

接口原型：

-(void)checkIsLocalNumberWith:(NSString \*)userName

finishBlock:(accessBlock) successBlock failBlock:(accessBlock )failBlock;

接口用于在移动网络下检验是否登录的是本机号码。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| userName | NSString | 手机号码 |
| successBlock | accessBlock | 检查是否登录本机号码成功的时候调用 |
| failBlock | accessBlock | 检查是否登录本机号码失败的时候调用 |

返回值说明

successBlcok, failBlock的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |

### 使用IDMPAutoLoginViewController.h中的getAuthType方法

接口原型：

- (int)getAuthType;

接口用于获取当前手机的网络状态及sim卡状态

返回值说明：

返回为0，当前的网络为中国移动蜂窝网和wifi同时开启，返回为1，当前的网络为中国移动蜂窝网，返回2、3代表为wifi链接，但2有中国移动SIM卡，3可能有也可能没有，返回-1为错误。

### 使用IDMPAutoLoginViewController.h中的currentEdition方法

接口原型：

- (void)currentEdition;

接口用于打印当前版本号

### 使用IDMPAutoLoginViewController.h中的reAuthentication方法

接口原型：

-(void) reAuthenticationWithUserName:(NSString \*)userName successBlock:(accessBlock)successBlock failBlock:(accessBlock)failBlock;

接口用于二次鉴权。

参数说明

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| userName | NSString | 手机号码 |
| successBlock | accessBlock | 二次鉴权成功时回调，由开发者自行实现 |
| failBlock | accessBlock | 二次鉴权失败时回调，由开发者自行实现 |

返回值说明

successBlcok的参数为NSDictionary类型，里面可能包含的参数含义如下：

|  |  |  |
| --- | --- | --- |
| **字段** | **类型** | **含义** |
| resultCode | NSUInteger | 返回相应的结果码 |
| resultString | NSString | 后续加入 |
| token | NSString | 身份标识，字符串形式的token，第三方应用将该凭证经应用平台向IDMP平台请求认证；  第三方应用为非SIP应用时返回（此时不返回password） |
| passid | NSString | 和通行证ID |

## 接口示例

### 使用IDMPAutoLoginViewController中的validateWithAppid方法

调用initWithAppid方法示例

|  |
| --- |
| #import "IDMPAppDelegate.h"  #import "IDMPAutoLoginViewController.h"  @implementation IDMPAppDelegate  - (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions  {  IDMPAutoLoginViewController \*autoLogin=[[IDMPAutoLoginViewController alloc] validateWithAppid:@" 11001304" Appkey:@" EB95B39FB061A05D" TimeoutInterval:18.6 isLogPrint:YES successBlock:^(NSDictionary \*paraments) {  NSLog(@"成功:%@",paraments);  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"失败:%@",paraments);  }];  return YES;  } |

|  |
| --- |
| #import <UIKit/UIKit.h>  #import "IDMPAutoLoginViewController.h"  @interface IDMPTestViewController : IDMPAutoLoginViewController  @end |

### 使用IDMPAutoLoginViewController中的getAccessToken方法

调用getAccessToken方法示例

|  |
| --- |
| -------- IDMPTestViewController.m---------  #import "IDMPTestViewController.h" #import "IDMPAutoLoginViewController.h"  @implementation IDMPTestViewController  - (void)autologin {  IDMPAutoLoginViewController\* idmp = [IDMPAutoLoginViewController new];  [idmp getAccessTokenWithUserName:nil andLoginType:1 isUserDefaultUI:YES finishBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  }];  } |

### 使用IDMPAutoLoginViewController中的getAccessPassword方法

调用getAccessPassword方法示例

|  |
| --- |
| -------- IDMPTestViewController.m---------  #import "IDMPTestViewController.h" #import " IDMPAutoLoginViewController.h"  @implementation IDMPTestViewController  - (void)autologin {  IDMPAutoLoginViewController\* idmp = [IDMPAutoLoginViewController new];  [idmp getAccessPasswordWithUserName:nil andLoginType:1 isUserDefaultUI:YES finishBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  }];  } |

### 使用IDMPAutoLoginViewController中的getAccessTokenByCondition方法

使用getAccessTokenByCondition示例

|  |
| --- |
| -------- IDMPTestViewController.m---------  #import "IDMPTestViewController.h" #import " IDMPAutoLoginViewController.h"  - (void)tokenCondition  {  IDMPAutoLoginViewController\* idmp = [IDMPAutoLoginViewController new];  [idmp getAccessTokenByConditionWithUserName:@"18867103532" Content:@"4998760759299" andLoginType:1 finishBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  }];  } |

### 使用IDMPAutoLoginViewController中的getAccessPasswordByCondition方法

使用getAccessPasswordByCondition示例

|  |
| --- |
| -------- IDMPTestViewController.m---------  #import "IDMPTestViewController.h" #import " IDMPAutoLoginViewController.h"  - (void)tokenCondition  {  IDMPAutoLoginViewController\* idmp = [IDMPAutoLoginViewController new];  [idmp getAccessPasswordByConditionWithUserName:@"18867103532" Content:@"4998760759299" andLoginType:1 finishBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"paraments:%@",paraments);  }];  } |

### 使用IDMPTempSmsMode.h中的getSmsCode方法

使用getSmsCode方法示例

|  |
| --- |
| #import "IDMPTempSmsMode.h"  - (void)onGetSmsClick {   IDMPTempSmsMode \*tempSms=[[IDMPTempSmsMode alloc] init];  NSString \*userName = \_userName.text;   if(![userName isEqual:@""])   {    [tempSms getSmsCodeWithUserName:(NSString \*)username busiType:@”3” successBlock:    ^{       NSLog(@"获取验证码成功");     }     failBlock:    ^{       NSLog(@"获取验证码失败");    }];    }    else    {      NSLog(@"username error!");    } **}** |

### 使用IDMPAutoLoginViewController.h中的cleanSSO方法

使用cleanSSO示例

|  |
| --- |
| -(void)onclearButtonClick {     [self cleanSSO]; } |

### 使用IDMPAutoLoginViewController.h中的registerUser方法

使用registerUser示例

|  |
| --- |
| - (IBAction)registerAccount:(id)sender {  IDMPAutoLoginViewController \*idmp = [[IDMPAutoLoginViewController alloc]init];  [idmp registerUserWithPhoneNo:self.phoneNumber.text passWord:self.passwordOrOld.text andValidCode:self.valid.text finishBlock:^(NSDictionary \*paraments) {  if ([[paraments objectForKey:@"resultCode"] integerValue]==103000) {  NSLog(@"register success");  }else{  NSLog(@"fail");  }  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"register fail");  }];  } |

### 使用IDMPAutoLoginViewController.h中的resetPassword方法

调用resetPassword示例

|  |
| --- |
| - (IBAction)resetPassword:(id)sender {  IDMPAutoLoginViewController \*idmp = [[IDMPAutoLoginViewController alloc]init];  [idmp resetPasswordWithPhoneNo:self.phoneNumber.text passWord:self.passwordOrNew.text andValidCode:self.valid.text finishBlock:^(NSDictionary \*paraments) {  if ([[paraments objectForKey:@"resultCode"] integerValue]==103000) {  NSLog(@"reset success");  }else{  NSLog(@"fail");  }  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"fail");  }];  } |

### 使用IDMPAutoLoginViewController.h中的changePassword方法

调用changePassword示例

|  |
| --- |
| - (IBAction)changePassword:(id)sender {  IDMPAutoLoginViewController \*idmp= [[IDMPAutoLoginViewController alloc]init];  [idmp changePasswordWithPhoneNo:self.phoneNumber.text passWord:self.passwordOrOld.text andNewPSW:self.passwordOrNew.text finishBlock:^(NSDictionary \*paraments) {  if ([[paraments objectForKey:@"resultCode"] integerValue]==103000) {  NSLog(@"change success");  }else{  NSLog(@"fail");  }  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"change fail");  }];  } |

### 使用IDMPAutoLoginViewController.h中的checkIsLocalNumber方法

调用checkIsLocalNumber示例

|  |
| --- |
| - (void)checkNum {  IDMPAutoLoginViewController \*idmp= [[IDMPAutoLoginViewController alloc]init];  [idmp checkIsLocalNumberWith:@"19978291123" finishBlock:^(NSDictionary \*paraments) {  if ([[paraments objectForKey:@"resultCode"] integerValue]==103000) {  NSLog(@"check success");  }else{  NSLog(@"fail");  }  } failBlock:^(NSDictionary \*paraments) {  NSLog(@"change fail");  }];  } |

### 使用IDMPAutoLoginViewController.h中的getAuthType方法

调用getAuthType示例

|  |
| --- |
| IDMPAutoLoginViewController \*idmp= [[IDMPAutoLoginViewController alloc]init];  int authType = [idmp getAuthType]; |

### 使用IDMPAutoLoginViewController.h中的currentEdition方法

调用getAuthType示例

|  |
| --- |
| IDMPAutoLoginViewController \*idmp= [[IDMPAutoLoginViewController alloc]init];  [idmp currentEdition]; |

### 使用IDMPAutoLoginViewController.h中的reAuthentication方法

调用reAuthentication示例

|  |
| --- |
| IDMPAutoLoginViewController \*idmp= [[IDMPAutoLoginViewController alloc]init];  [idmp reAuthenticationWithUserName:@”your userName” successBlock:^(NSDictionary \*paraments) {    } failBlock:^(NSDictionary \*paraments) {    }]; |

## SDK错误码说明

使用SDK时，SDK会在认证结束后将结果回调给开发者，其中结果为JSONObject对象，其中resultCode为结果响应码，102000/103000代表成功，其他为失败。成功时在根据token字段取出身份标识。失败时根据resultCode定位失败原因。

### 返回JSON说明

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 含义 |
| resultCode | Int | 结果响应码 |
| resultString | String | 失败时的错误描述(后续加入) |

### 错误码详述

|  |  |  |
| --- | --- | --- |
| 分类 | 错误编号 | 返回码描述 |
| 正确 | 102000 | 正确返回 |
| 103000 | 正确返回 |
| 网络类错误 | 10210X |  |
| 102101 | 无网络 |
| 102102 | 网络连接超时（根据Server设置变化） |
| 调用类错误 | 1022XX |  |
| 102201 | 自动登陆失败，用户选择自定义界面时，需要继续处理手动登陆流程，比如弹出登陆界面。 |
| 102202 | APP签名验证不通过 |
| 102203 | 接口入参错误 |
| 102204 | 正在进行GetToken动作，请稍后 |
| 102205 | 当前环境不支持指定的登陆方式 |
| 102206 | 选择用户登陆时，本地不存在指定的用户 |
| 102207 | IDMPSipAppNoSupport |
| 102208 | data sms send failed |
| 102298 | 应用合法性校验失败 |
| 102299 | 其他错误 |
| 用户类错误 | 102301 | 用户取消 |
| 102302 | 缓存用户列表为空 |
| 102303 | 用户名温控 |
| 102304 | 密码为空 |
| 102305 | 验证码获得成功 |
| 102306 | 验证码获得失败 |
| 102307 | 用户名格式错误 |
| 102308 | 验证码为空 |
| 102309 | 验证码格式错误 |
| 102310 | 用户名和验证码格式错误 |
| 102311 | 密码格式错误 |
| 102312 | 用户名和密码格式错误 |
| 102313 | 短信登录失败 |
| 102314 | 指定用户缓存不存在 |
| 102317 | token生成失败 |
| 平台类错误 | 103101 | 错误的请求签名 |
| 103102 | 签名检验失败 |
| 103103 | 用户不存在 |
| 103104 | 用户不支持该种登录方式 |
| 103105 | 密码错误 |
| 103106 | 用户名错误 |
| 103107 | 已存在相同的随机数 |
| 103108 | 短信验证码错误 |
| 103109 | 短信验证码超时 |
| 103110 | 数据短信随机数和http请求随机数不一致（http+sms） |
| 103111 | WAP网关IP不合法 |
| 103112 | 请求错误 reqError |
| 103113 | Token内容错误 |
| 103114 | token验证 KS过期 |
| 103115 | token验证 KS不存在 |
| 103116 | token验证 sqn错误 |
| 103117 | mac异常 macError |
| 103118 | sourceid不存在 |
| 103119 | appid不存在appidNOExist |
| 103120 | clientauth不存在 |
| 103121 | passid不存在 |
| 103122 | btid不存在 |
| 103123 | redisinfo不存在 |
| 103124 | ksnaf校验不一致 |
| 103125 | 手机格式错误 |
| 103126 | 手机号不存在 |
| 103127 | 证书验证，版本过期 |
| 103128 | gba webservice接口调用失败 |
| 103129 | 获取短信验证码的msgtype异常 |
| 103130 | 新密码不能与当前密码相同 |
| 103131 | 密码过于简单 |
| 103132 | 用户注册失败 |
| 103133 | sourceid不合法 |
| 103134 | wap方式手机号为空 |
| 103135 | 昵称非法 |
| 103136 | 邮箱非法 |
| 103137 | 请求其他平台token验证接口异常 |
| 103201 | 数据库中间件服务异常 dmerror |
| 103202 | 平台解密异常 |
| 103203 | 缓存用户不存在 |
| 103204 | 缓存随机数不存在 |
| 103205 | 服务器内部异常 |
| 103206 | 加密机异常 secError |
| 103207 | 发送短信失败 |
| 103208 | 调用第三方接口失败 |
| 103209 | 同步应用密码失败 |
| 103210 | 修改密码失败 |
| 103211 | 其它错误 |
| 103212 | 用户验证失败 |
| 103213 | 旧密码错误 |
| 103265 | 用户已存在 |
| 103266 | 密码格式非法 |
| 103300 | SIP成功 |
| 103301 | sip校验token异常 |
| 103302 | sip 用户已存在 |
| 103303 | sip 用户未开户（获取应用密码） |
| 103304 | sip 用户未开户（注销用户） |
| 103305 | sip 开户用户名错误 |
| 103306 | sip 用户名不能为空（获取应用密码） |
| 103307 | sip 用户名不能为空（注销用户） |
| 103308 | sip 手机号不合法 |
| 103309 | sip opertype为空 |
| 103310 | sip sourceid不存在 |
| 103311 | sip sourceid不合法 |
| 103312 | sip btid不存在 |
| 103313 | sip ks不存在 |
| 103399 | sip 系统错误 |
| 103400 | authorization为空 |
| 103401 | 签名消息为空 |
| 103402 | 无效的authWay |
| 103403 | 默认注册失败 |
| 103404 | 加密失败 |
| 103405 | 保存数据短信手机号为空 |
| 103406 | 保存数据短信短信内容为空 |
| 103407 | 此sourceId，appPackage，si… |
| 103408 | 此sourceId注册已达上限 99次 |
| 103409 | query为空 |
| 103410 | 无效的mac |
| 103411 | query异常 |
| 103412 | 无效的请求 |
| 103413 | 运行时异常 |
| 103414 | 参数校验异常 |
| 103500 | 注册用户处于黑名单无法操作 |
| 103501 | 修改密码用户处于黑名单无法操作 |
| 103502 | 重置密码用户处于黑名单无法操作 |
| 103503 | 登录用户处于黑名单无法操作 |
| 103801 | aoi token 手机号码非法 |
| 103802 | aoi token 手机号码未注册 |
| 103803 | aoi token 请求超时 |
| 103804 | aoi token 用户已确认登录 |
| 103805 | aoi token 用户已取消登录 |
| 103806 | aoi token sourceid不合法 |
| 103807 | aoi 用户不在线 |
| 103808 | aoi 短信已发送，用户还未确认 |
| 103809 | 短信发送失败 |
| 103899 | aoi token 其他错误 |

注：未定义的错误码为系统错误码，不作重复定义。