目录

[目录 1](#_Toc1164)

[1. TLS SDK集成 2](#_Toc24002)

[1.1获取TLS SDK 2](#_Toc7940)

[1.2 Android Studio集成示例 2](#_Toc12543)

[1.3 Eclipse集成示例 7](#_Toc1143)

[2. 功能开发 12](#_Toc3529)

[2.1 密码注册（手机帐号） 13](#_Toc18133)

[2.2 密码登录 17](#_Toc15389)

[2.3 密码重置 21](#_Toc23725)

[2.4 无密码注册 25](#_Toc5296)

[2.5 短信登录 29](#_Toc3778)

[2.6 刷新UserSig 33](#_Toc11438)

[2.7密码注册（字符串账号） 35](#_Toc5937)

[2.8 匿名登录(或称访客登录) 37](#_Toc4287)

[2.9 第三方账号与自有账号绑定 38](#_Toc9989)

[2.10 获取本地保存的SSO票据 42](#_Toc4309)

## 1. TLS SDK集成

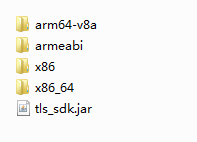
本节主要介绍如何使用Anroid Studio/Eclipse创建一个应用，并集成TLS SDK。这里假设用户已经了解如何集成IMSDK，如有疑问，可参考[IMSDK相关文档](#_1._TLS_SDK集成)。

关于android 开发环境的搭建，请先参考：

<http://developer.android.com/sdk/installing/installing-adt.html>

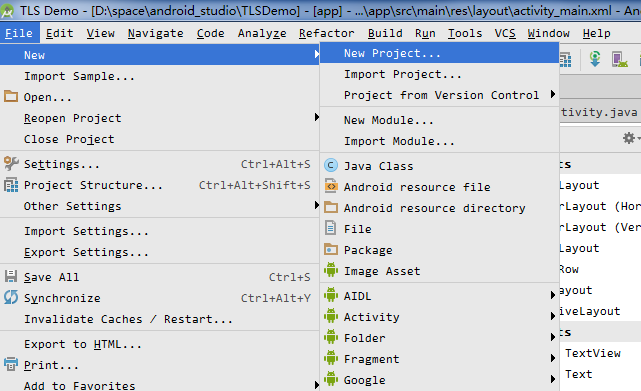
### 1.1获取TLS SDK

从[官网](http://url.cn/5TbilhD)下载TLS SDK，包括tls\_sdk.jar 和 libwtcrypto.so(so 文件可自行选择支持平台)。

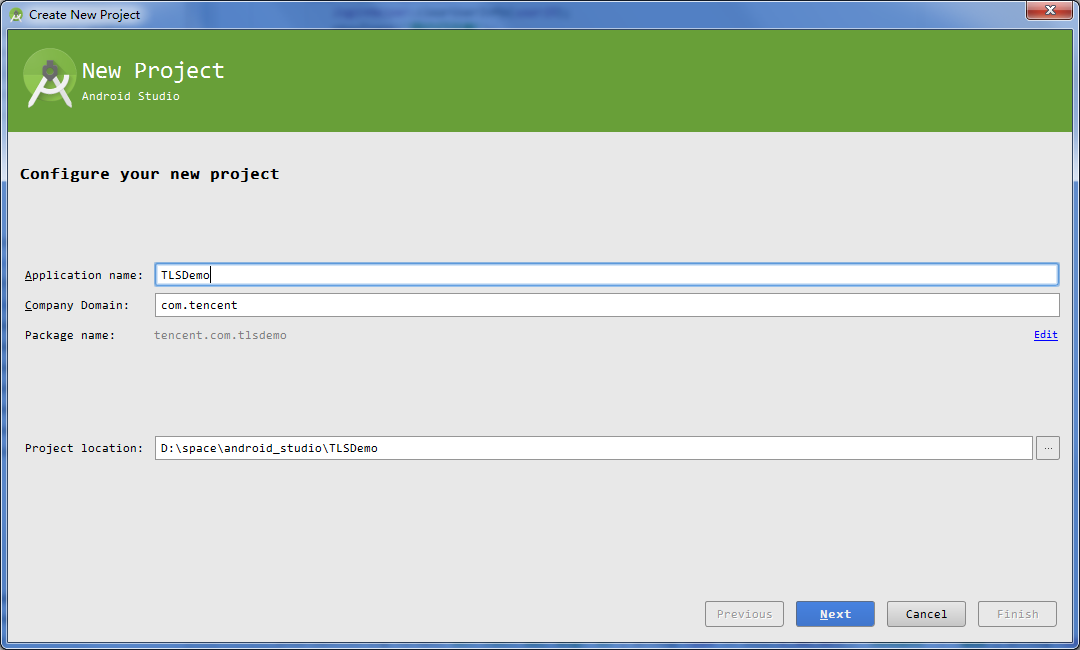


### 1.2 Android Studio集成示例

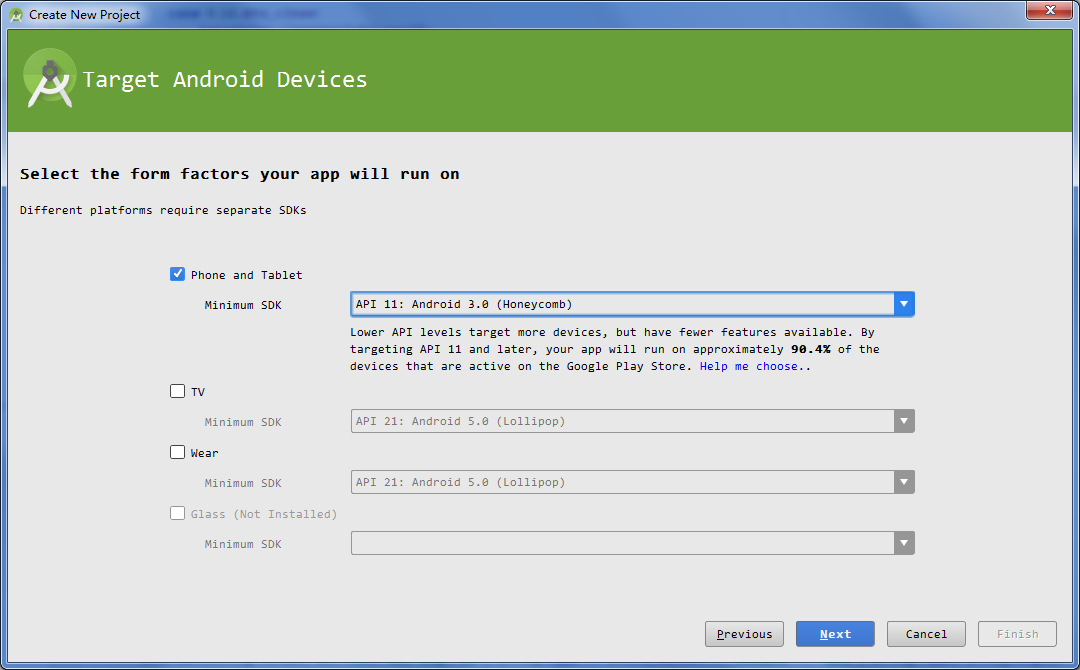
打开Android Studio, 新建工程：



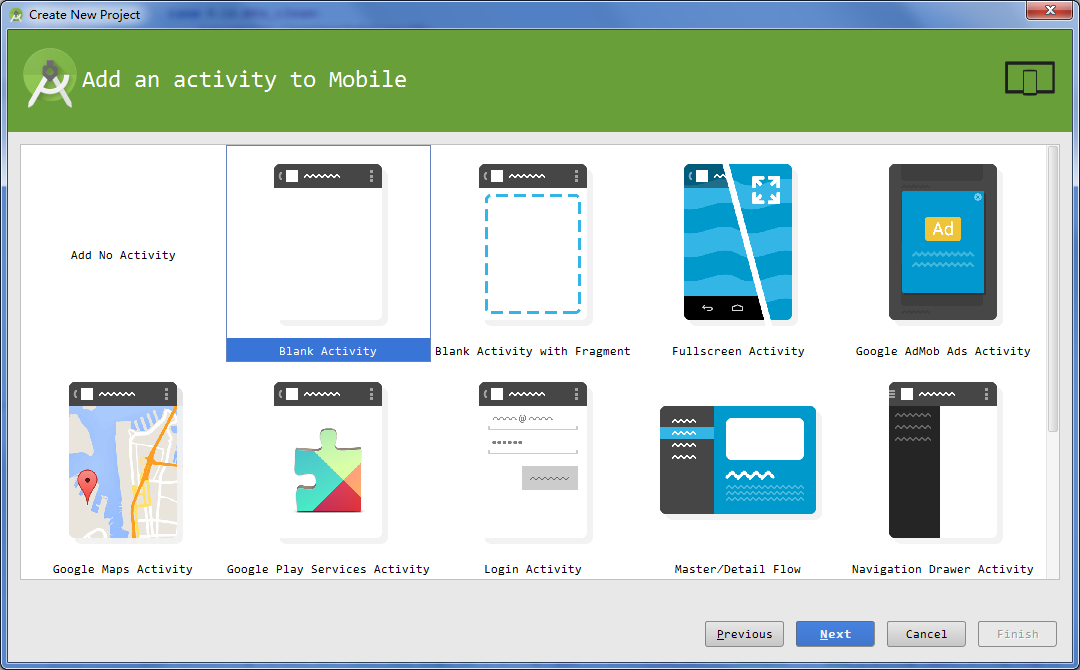
工程命名为TLSDemo，并选择合适的工程目录：



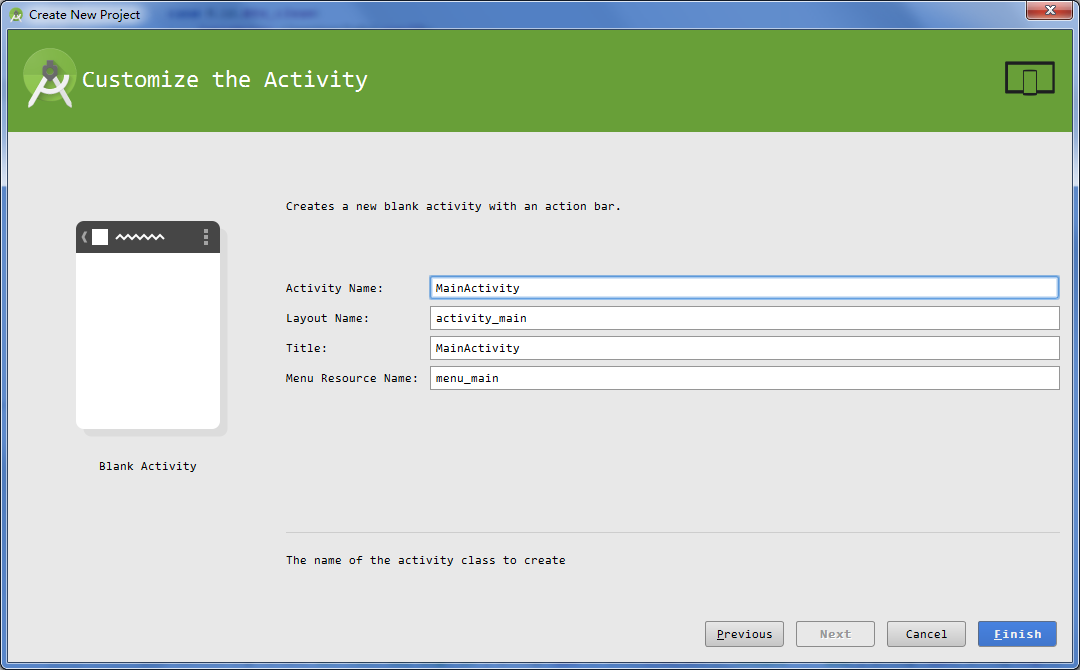
选择Android SDK API版本(TLS SDK 并无特别要求，这里使用了默认选项)：



选择一个空的界面：

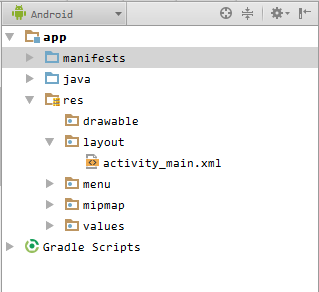


并为之命名(这里仍然选择默认，可自行修改)：



完成。

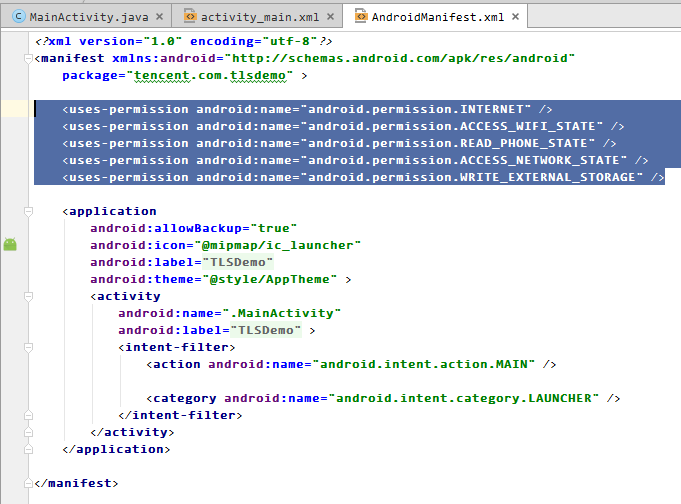
打开mainfest文件：



在<application/>标签之前贴入以下权限：

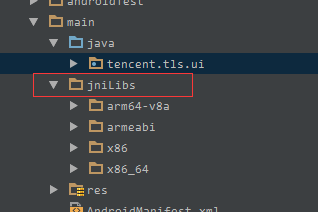
<**uses-permission android:name="android.permission.INTERNET"** />  
<**uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"** />  
<**uses-permission android:name="android.permission.READ\_PHONE\_STATE"** />  
<**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />  
<**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />

最终效果如图：

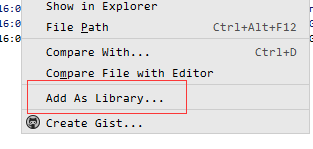
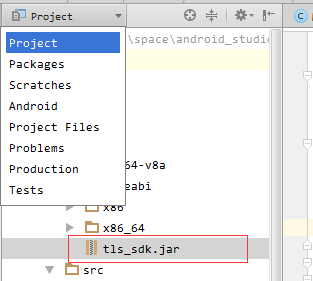


把1.1.1中下载到的相关库文件直接复制到app/libs目录，通过查看模块的build.gradle文件可以知道libs目录下的jar包在编译的时候会自动被包含的。so 文件目录请放到 jniLibs 目录下。

dependencies {  
 compile fileTree(dir: **'libs'**, include: [**'\*.jar'**])  
 compile **'com.android.support:appcompat-v7:21.0.3'**  
}

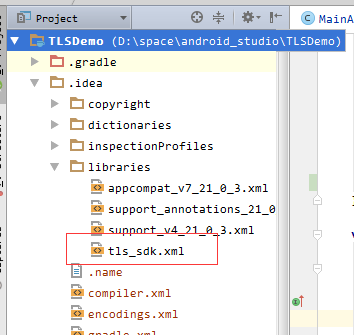


如果发现tls\_sdk 没有被成功集成，可切换到Project模式，在tls\_sdk.jar文件上点击右键，手动添加为库：



在成功添加tls\_sdk.jar之后，我们还可以为之添加文档。

在工程目录下找到配置文件tls\_sdk.xml:

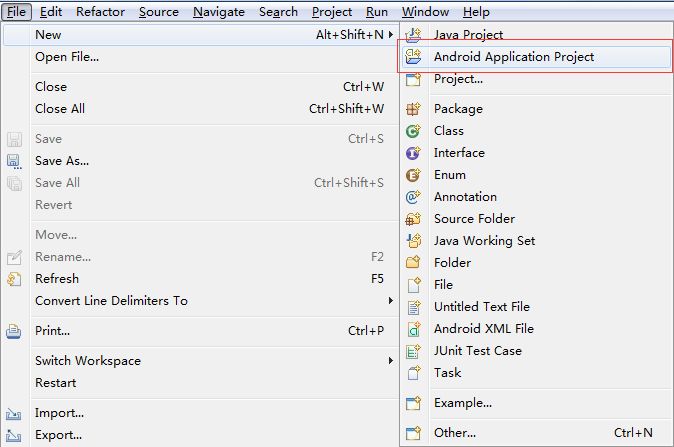


在JAVADOC节点下做如下配置(url即tls sdk 对应的接口文档目录)：

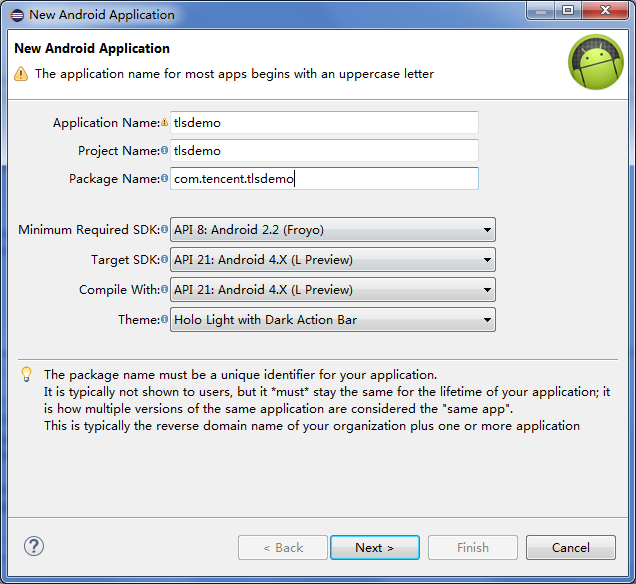


### 1.3 Eclipse集成示例

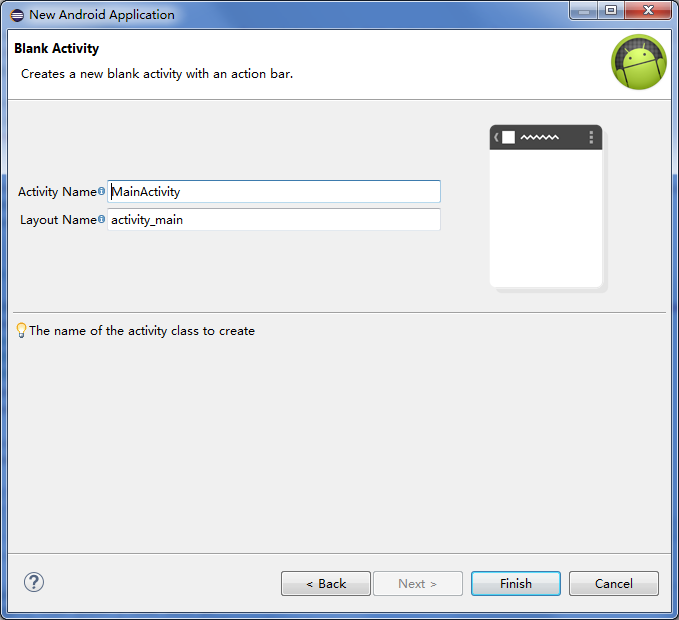
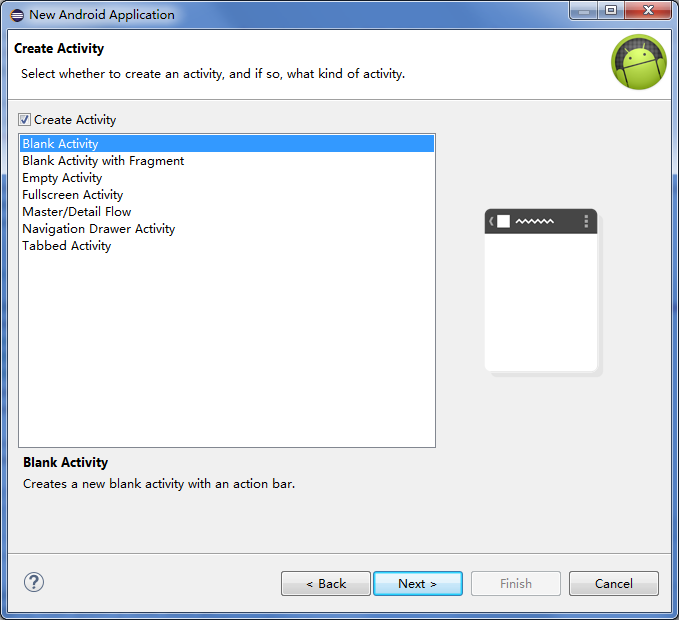
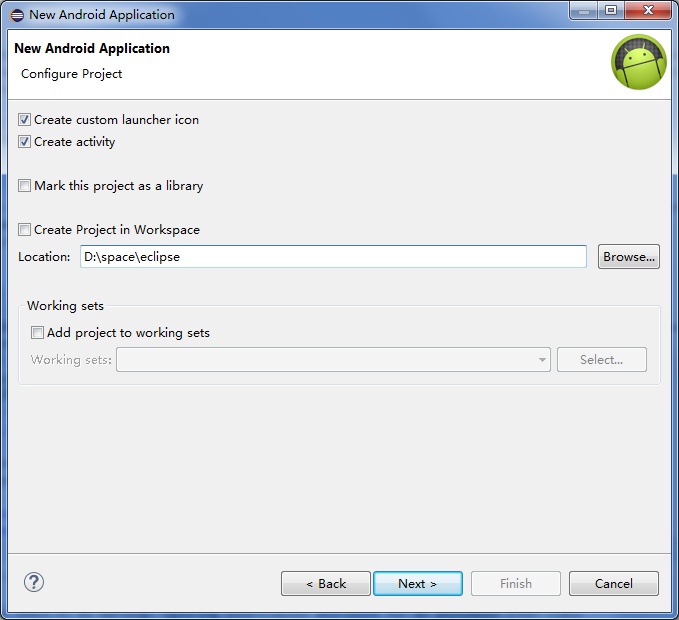
打开eclipse, 新建Android Application Project:



填写应用名称，API 可自行选择，TLS SDK 无要求：



点击下一步，根据需要设置工程目录：



在ApplicationManifest.xml 中声明必需的权限：

<uses-permission android:name=*"android.permission.INTERNET"* />

<uses-permission android:name=*"android.permission.ACCESS\_WIFI\_STATE"* />

<uses-permission android:name=*"android.permission.READ\_PHONE\_STATE"* />

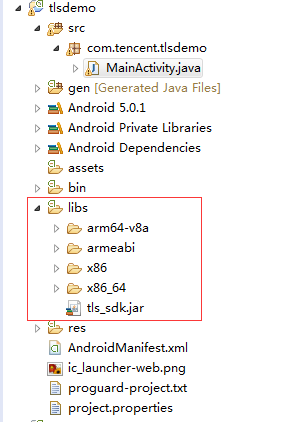
<uses-permission android:name=*"android.permission.ACCESS\_NETWORK\_STATE"* />

<uses-permission android:name=*"android.permission.WRITE\_EXTERNAL\_STORAGE"* />

最终效果如图：



把一开始下载到的相关库文件复制到工程目录的libs目录之下，就可以开始使用TLS SDK进行开发了：



如果没有复制文件之后没有预期效果，可以试下在项目名称上右键进行刷新，从而看到最新目录结构。

## 2. 功能开发

接下来按照不同场景，调用相应接口实现注册和登录功能。我们将以TLS SDK相关接口的使用为主，并假设开发者已经得到sdkAppid等帐号属性。

在接下来的流程中，假设已完成以下初始化：

int *sdkAppid* = 开发者申请的SDK Appid;

// 务必检查IMSDK已做以下初始化

QALSDKManager.getInstance().setEnv(0);

QALSDKManager.*getInstance*().init(getApplicationContext(), sdkAppid, 0);

// 初始化TLSSDK

TLSHelper tlsHelper = TLSHelper.getInstance().init(getApplicationContext(), sdkAppid);

### 2.1 密码注册（手机帐号）



#### 初始化设置

变量设置:

PwdRegListener *pwdRegListener*;

#### 步骤一 请求验证手机号码

引导用户输入手机号码之后，调用以下接口请求下发验证短信：

// 手机号码请输入正确的真实的手机号码，格式是 国家码-手机号码，比如 86-186xxx

tlsHelper.TLSPwdRegAskCode("86-186xxxx", *pwdRegListener*);

该接口可能的回调是**OnPwdRegAskCodeSuccess**, OnPwdRegFail以及OnPwdRegTimeout。具体实现请参考[监听函数实现](#_3.1.5_监听函数实现) 相关说明。

成功调用该接口之后，如果用户长时间没有收到短信，在reaskDuration设定的时间之后可以请求重新下发短信验证码。这种情况应该调用TLSPwdRegReaskCode，调用如下：

tlsHelper.TLSPwdRegReaskCode(*pwdRegListener*);

该接口可能的回调是**OnPwdRegReaskCodeSuccess**, OnPwdRegFail以及OnPwdRegTimeout。具体实现请参考[监听函数实现](#_3.1.5_监听函数实现) 相关说明。

该函数可能多次调用，处理情况同首次请求下发短信。

#### 步骤二 验证短信验证码

在相应的界面引导用户输入收到的短信验证码之后，调用以下接口：

tlsHelper.TLSPwdRegVerifyCode("用户输入的验证码", *pwdRegListener*);

该接口可能的回调是**OnPwdRegVerifyCodeSuccess**, OnPwdRegFail以及OnPwdRegTimeout。具体实现请参考[监听函数实现](#_3.1.5_监听函数实现) 相关说明。

验证通过之后，到达最后一步，用户设置密码并确认注册。请看下一步：

#### 步骤三 提交密码完成注册

引导用户输入密码之后，调用以下接口确认注册：

tlsHelper. TLSPwdRegCommit ("用户输入的密码", *pwdRegListener*);

该接口可能的回调是**OnPwdRegCommitSuccess**, OnPwdRegFail以及OnPwdRegTimeout。具体实现请参考[监听函数实现](#_3.1.5_监听函数实现) 相关说明。

至此用户的手机号码已经成功完成注册，接下来可以使用注册时设置的密码进行[登录](#_2.2_密码登录)了。

#### 监听函数实现

pwdRegListener = new TLSPwdRegListener() {  
 @Override  
 public void OnPwdRegAskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 请求下发短信成功，可以跳转到输入验证码进行校验的界面，同时可以开始倒计时, (reaskDuration 秒内不可以重发短信，如果在expireDuration 秒之后仍然没有进行短信验证，则应该回到上一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用PwdRegVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnPwdRegReaskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 重新请求下发短信成功，可以跳转到输入验证码进行校验的界面，并开始倒计时，(reaskDuration 秒内不可以再次请求重发，在expireDuration 秒之后仍然没有进行短信验证，则应该回到第一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用PwdRegVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnPwdRegVerifyCodeSuccess() {  
 /\* 短信验证成功，接下来可以引导用户输入密码，然后调用PwdRegCommit 进行注册的最后一步\*/  
 }  
  
 @Override  
 public void OnPwdRegCommitSuccess(TLSUserInfo userInfo) {  
 /\* 最终注册成功，接下来可以引导用户进行密码登录了，登录流程请查看相应章节\*/  
 }  
  
 @Override  
 public void OnPwdRegFail(TLSErrInfo tlsErrInfo) {  
 /\* 密码注册过程中任意一步都可以到达这里，可以根据tlsErrInfo 中ErrCode, Title, Msg 给用户弹提示语，引导相关操作\*/  
 }  
  
 @Override  
 public void OnPwdRegTimeout(TLSErrInfo tlsErrInfo) {  
 /\* 密码注册过程中任意一步都可以到达这里，顾名思义，网络超时，可能是用户网络环境不稳定，一般让用户重试即可\*/  
 }  
};

### 2.2 密码登录



#### 初始化设置

变量设置:

PwdLoginListener *pwdLoginListener*;

#### 步骤一 验证用户密码

注意： TLS SDK会负责已登录用户登录凭证的保存，所以并非每次都需要用户输入用户名和密码登录的。这里可以认为SDK简单实现了保存”密码”的功能，只不过并没有保存明文密码那么简单。可通过调用 *tlsHelper*.needLogin("用户帐号") 判断是否需要输入密码登录，如果不需要输入密码，可直接进行[刷新UserSig](#_3.6_刷新UserSig)操作，否则，开始一般的验证密码登录流程，如下：

引导用户输入用户名和密码，调用以下接口登录：

// 这里的"用户帐号", 如果是手机号码，格式为“86-186xxx”。  
 *tlsHelper*.TLSPwdLogin("用户帐号", "用户密码".getBytes(), *pwdLoginListener*);

该接口可能的回调是**OnPwdLoginSuccess**, **OnPwdLoginNeedImgcode**, OnPwdLoginFail以及OnPwdLoginTimeout。具体实现请参考[监听函数实现](#_3.2.4_监听函数实现) 相关说明。

如果OnPwdLoginNeedImgcode 被回调，意味着用户本次登录存在安全风险，需要验证图片验证码完成之后才能继续登录。验证码相关接口的使用，请看下一步。

#### 步骤二 验证图片验证码

引导用户输入图片验证码，然后可以调用如下接口进行验证：

*tlsHelper*.TLSPwdLoginVerifyImgcode("验证码", *pwdLoginListener*);

该接口可能的回调是**OnPwdLoginSuccess**， **OnPwdLoginNeedImgcode**，OnPwdLoginFail以及 OnPwdLoginTimeout。具体实现请参考[监听函数实现](#_3.2.4_监听函数实现) 相关说明。

图片验证码如果验证不通过仍然会回调 **OnPwdLoginNeedImgcode**，验证通过而且密码正确，那么会回调 OnPwdLoginSuccess，验证通过却密码错误，那么回调的是 OnPwdLoginFail。

当然，有时如果用户觉得看到的图片验证码不够清晰，可以使用如下接口请求刷新图片验证码：

*tlsHelper*.TLSPwdLoginReaskImgcode(*pwdLoginListener*);

该接口可能的回调是**OnPwdLoginReaskImgcodeSuccess**，OnPwdLoginFail以及 OnPwdLoginTimeout。具体实现请参考[监听函数实现](#_3.2.4_监听函数实现) 相关说明。

该接口可多次调用。

#### 监听函数实现

pwdLoginListener = new TLSPwdLoginListener() {  
 @Override  
 public void OnPwdLoginSuccess(TLSUserInfo userInfo) {  
 /\* 登录成功了，在这里可以获取用户票据\*/  
 String usersig = *tlsHelper*.getUserSig(userInfo.identifier);  
 }  
  
 @Override  
 public void OnPwdLoginReaskImgcodeSuccess(byte[] picData) {  
 /\* 请求刷新图片验证码成功，此时需要用picData 更新验证码图片，原先的验证码已经失效\*/  
 }  
  
 @Override  
 public void OnPwdLoginNeedImgcode(byte[] picData, TLSErrInfo errInfo) {  
 /\* 用户需要进行图片验证码的验证，需要把验证码图片展示给用户，并引导用户输入；如果验证码输入错误，仍然会到达此回调并更新图片验证码\*/  
 }  
  
 @Override  
 public void OnPwdLoginFail(TLSErrInfo errInfo) {  
 /\* 登录失败，比如说密码错误，用户帐号不存在等，通过errInfo.ErrCode, errInfo.Title, errInfo.Msg等可以得到更具体的错误信息\*/  
 }  
 @Override  
 public void OnPwdLoginTimeout(TLSErrInfo errInfo) {  
 /\* 密码登录过程中任意一步都可以到达这里，顾名思义，网络超时，可能是用户网络环境不稳定，一般让用户重试即可\*/  
 }  
};

### 2.3 密码重置



#### 初始化设置

变量设置:

PwdResetListener *pwdResetListener*;

#### 步骤一 请求验证手机号码

引导用户输入手机号码之后，调用以下接口请求下发验证短信：

// 手机号码请输入正确的真实的手机号码，格式是 国家码-手机号码，比如 86-186xxx

*tlsHelper*.TLSPwdResetAskCode("86-186xxxx", *pwdResetListener*);

该接口可能的回调是**OnPwdResetAskCodeSuccess**, OnPwdResetFail以及OnPwdResetTimeout。具体实现请参考[监听函数实现](#_3.3.5_监听函数实现) 相关说明。

成功调用该接口之后，如果用户长时间没有收到短信，在reaskDuration设定的时间之后可以请求重新下发短信验证码。这种情况应该调用TLSPwdResetReaskCode，调用如下：

*tlsHelper*.TLSPwdResetReaskCode(*pwdResetListener*);

该接口可能的回调是**OnPwdResetReaskCodeSuccess**, OnPwdResetFail以及OnPwdResetTimeout。具体实现请参考[监听函数实现](#_3.3.5_监听函数实现) 相关说明。

该函数可能多次调用，处理情况同首次请求下发短信。

#### 步骤二 验证短信验证码

在相应的界面引导用户输入收到的短信验证码之后，调用以下接口：

*tlsHelper*.TLSPwdResetVerifyCode("用户输入的验证码", *pwdResetListener*);

该接口可能的回调是**OnPwdResetVerifyCodeSuccess**, OnPwdResetFail以及OnPwdResetTimeout。具体实现请参考[监听函数实现](#_3.3.5_监听函数实现) 相关说明。

验证通过之后，到达最后一步，用户设置密码并确认使用新密码。请看下一节。

#### 步骤三 确认使用新密码

引导用户输入密码之后，调用以下接口确认使用新密码：

*tlsHelper*.TLSPwdResetCommit ("用户输入的密码", *pwdResetListener*);

该接口可能的回调是**OnPwdResetCommitSuccess**, OnPwdResetFail以及OnPwdResetTimeout。具体实现请参考[监听函数实现](#_3.3.5_监听函数实现) 相关说明。

至此用户的手机帐号已经成功完成密码的重置，接下来可以使用新密码进行[登录](#_3.2_密码登录)了。

#### 监听函数实现

pwdResetListener = new TLSPwdResetListener() {  
 @Override  
 public void OnPwdResetAskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 请求下发短信成功，可以跳转到输入验证码进行校验的界面，同时可以开始倒计时, (reaskDuration 秒内不可以重发短信，如果在expireDuration 秒之后仍然没有进行短信验证，则应该回到上一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用PwdResetVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnPwdResetReaskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 重新请求下发短信成功，可以跳转到输入验证码进行校验的界面，并开始倒计时，(reaskDuration 秒内不可以再次请求重发，在expireDuration 秒之后仍然没有进行短信验证，则应该回到第一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用PwdResetVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnPwdResetVerifyCodeSuccess() {  
 /\* 短信验证成功，接下来可以引导用户输入密码，然后调用PwdResetCommit 完成重置密码流程\*/  
 }  
  
 @Override  
 public void OnPwdResetCommitSuccess(TLSUserInfo userInfo) {  
 /\* 重置密码成功，接下来可以引导用户进行新密码登录了，登录流程请查看相应章节\*/  
 }  
  
 @Override  
 public void OnPwdResetFail(TLSErrInfo tlsErrInfo) {  
 /\* 重置密码过程中任意一步都可以到达这里，可以根据tlsErrInfo 中ErrCode, Title, Msg 给用户弹提示语，引导相关操作\*/  
 }  
  
 @Override  
 public void OnPwdResetTimeout(TLSErrInfo tlsErrInfo) {  
 /\* 重置密码过程中任意一步都可以到达这里，顾名思义，网络超时，可能是用户网络环境不稳定，一般让用户重试即可\*/  
 }  
};

### 2.4 无密码注册

无密码注册，顾名思义，与前面密码注册的区别在于，在完成验证操作之后，直接确认注册即可，不需要用户设置密码。那么，用户没有密码如何登录？SDK提供了短信登录的方式，用户登录的时候，只需要通过手机短信的验证就可以了，详情参考下一节 [2.5短信登录](#_2.5_短信登录)。



#### 初始化设置

变量设置:

SmsRegListener *smsRegListener*;

#### 步骤一 请求验证手机号码

引导用户输入手机号码之后，调用以下接口请求下发验证短信：

// 手机号码请输入正确的真实的手机号码，格式是 国家码-手机号码，比如 86-186xxx

tlsHelper.TLSSmsRegAskCode("86-186xxxx", *smsRegListener*);

该接口可能的回调是**OnSmsRegAskCodeSuccess**, OnSmsRegFail以及OnSmsRegTimeout。具体实现请参考[监听函数实现](#_3.4.5_监听函数实现) 相关说明。

成功调用该接口之后，如果用户长时间没有收到短信，在reaskDuration设定的时间之后可以请求重新下发短信验证码。这种情况应该调用TLSSmsRegReaskCode，调用如下：

*tlsHelper*.TLSSmsRegReaskCode(*smsRegListener*);

该接口可能的回调是OnSmsRegReaskCodeSuccess, OnSmsRegFail以及OnSmsRegTimeout。具体实现请参考[监听函数实现](#_3.4.5_监听函数实现) 相关说明。

该函数可能多次调用，处理情况同首次请求下发短信。

#### 步骤二 验证短信验证码

在相应的界面引导用户输入收到的短信验证码之后，调用以下接口：

*tlsHelper*.TLSSmsRegVerifyCode("用户输入的验证码", *smsRegListener*);

该接口可能的回调是**OnSmsRegVerifyCodeSuccess**, OnSmsRegFail以及OnSmsRegTimeout。具体实现请参考[监听函数实现](#_3.4.5_监听函数实现) 相关说明。

验证通过之后，到达最后一步，不需要用户设置密码，直接确认注册即可。请看下一小节。

#### 步骤三 完成注册

直接调用以下接口确认注册：

*tlsHelper*.TLSSmsRegCommit(*smsRegListener*);

该接口可能的回调是**OnSmsRegCommitSuccess**, OnSmsRegFail以及OnSmsRegTimeout。具体实现请参考[监听函数实现](#_监听函数实现) 相关说明。

至此用户的手机号码已经成功完成注册，接下来可以直接进行[短信登录](#_3.5_无密码登录_1)了。

#### 监听函数实现

*smsRegListener*= new TLSSmsRegListener() {  
 @Override  
 public void OnSmsRegAskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 请求下发短信成功，可以跳转到输入验证码进行校验的界面，同时可以开始倒计时, (reaskDuration 秒内不可以重发短信，如果在expireDuration秒之后仍然没有进行短信验证，则应该回到上一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用SmsRegVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnSmsRegReaskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 重新请求下发短信成功，可以跳转到输入验证码进行校验的界面，并开始倒计时，(reaskDuration 秒内不可以再次请求重发，在expireDuration 秒之后仍然没有进行短信验证，则应该回到第一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用SmsRegVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnSmsRegVerifyCodeSuccess() {  
 /\* 短信验证成功，接下来只需要用户确认操作，然后调用SmsRegCommit 完成注册流程\*/  
 }  
  
 @Override  
 public void OnSmsRegCommitSuccess(TLSUserInfo userInfo) {  
 /\* 最终注册成功，接下来可以引导用户进行短信登录了，登录流程请查看相应章节\*/  
 }  
  
 @Override  
 public void OnSmsRegFail(TLSErrInfo tlsErrInfo) {  
 /\* 无密码注册过程中任意一步都可以到达这里，可以根据tlsErrInfo 中ErrCode, Title, Msg 给用户弹提示语，引导相关操作\*/  
 }  
  
 @Override  
 public void OnSmsRegTimeout(TLSErrInfo tlsErrInfo) {  
 /\* 无密码注册过程中任意一步都可以到达这里，顾名思义，网络超时，可能是用户网络环境不稳定，一般让用户重试即可\*/  
 }  
};

### 2.5 短信登录

用户通过无密码注册获取到的帐号，可使用短信登录方式。登录过程中不需要用户输入密码，只需要通过手机短信验证即可。



#### 初始化设置

变量设置:

SmsLoginListener *smsLoginListener*;

#### 步骤一 请求验证手机号码

引导用户输入手机号码，调用以下接口请求下发短信验证码：

// 手机号码请输入正确的真实的手机号码，格式是 国家码-手机号码，比如 86-186xxx

*tlsHelper*.TLSSmsLoginAskCode("86-186xxxx", *smsLoginListener*);

该接口可能的回调是**OnSmsLoginAskCodeSuccess**, OnSmsLoginFail以及OnSmsLoginTimeout。具体实现请参考[监听函数实现](#_3.5.5_监听函数实现) 相关说明。

成功调用该接口之后，如果用户长时间没有收到短信，在reaskDuration设定的时间之后可以请求重新下发短信验证码。这种情况应该调用TLSSmsLoginReaskCode，调用如下：

*tlsHelper*.TLSSmsLoginReaskCode(*smsLoginListener*);

该接口可能的回调是**OnSmsLoginReaskCodeSuccess**, OnSmsLoginFail以及OnSmsLoginTimeout。具体实现请参考[监听函数实现](#_3.5.5_监听函数实现) 相关说明。

该函数可能多次调用，处理情况同首次请求下发短信。

#### 步骤二 验证短信验证码

在相应的界面引导用户输入收到的短信验证码之后，调用以下接口：

*tlsHelper*.TLSSmsLoginVerifyCode("用户输入的验证码", *smsLoginListener*);

该接口可能的回调是**OnSmsLoginVerifyCodeSuccess**, OnSmsLoginFail以及OnSmsLoginTimeout。具体实现请参考[监听函数实现](#_3.5.5_监听函数实现) 相关说明。

验证通过之后，到达最后一步，调用登录接口，才能完成登录流程。请看下一节。

#### 步骤三 完成登录

使用当前流程验证通过的手机号码，调用如下接口，完成短信登录：

// 手机号码请输入正确的真实的手机号码，格式是 国家码-手机号码，比如 86-186xxx

*tlsHelper*.TLSSmsLogin ("86-186xxx", *smsLoginListener*);

该接口可能的回调是**OnSmsLoginSuccess**, OnSmsLoginFail以及OnSmsLoginTimeout。具体实现请参考[监听函数实现](#_3.5.5_监听函数实现) 相关说明。

#### 监听函数实现

*smsLoginListener* = new TLSSmsLoginListener() {  
 @Override  
 public void OnSmsLoginAskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 请求下发短信成功，可以跳转到输入验证码进行校验的界面，同时可以开始倒计时, (reaskDuration 秒内不可以重发短信，如果在expireDuration 秒之后仍然没有进行短信验证，则应该回到上一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用SmsLoginVerifyCode进行验证。\*/  
 }  
  
 @Override  
 public void OnSmsLoginReaskCodeSuccess(int reaskDuration, int expireDuration) {  
 /\* 重新请求下发短信成功，可以跳转到输入验证码进行校验的界面，并开始倒计时，(reaskDuration 秒内不可以再次请求重发，在expireDuration 秒之后仍然没有进行短信验证，则应该回到第一步，重新开始流程)；在用户输入收到的短信验证码之后，可以调用SmsLoginVerifyCode 进行验证。\*/  
 }  
  
 @Override  
 public void OnSmsLoginVerifyCodeSuccess() {  
 /\* 这时候仅仅是通过了短信验证，还需要调用登录接口TLSSmsLogin 完成登录 \*/  
 }  
  
 @Override  
 public void OnSmsLoginSuccess(TLSUserInfo userInfo) {  
 /\* 登录成功了，在这里可以获取用户票据\*/  
 String usersig = *tlsHelper*.getUserSig(userInfo.identifier);  
 }  
  
 @Override  
 public void OnSmsLoginFail(TLSErrInfo errInfo) {  
 /\* 短信登录过程中任意一步都可以到达这里，可以根据tlsErrInfo 中ErrCode, Title, Msg 给用户弹提示语，引导相关操作\*/  
 }  
  
 @Override  
 public void OnSmsLoginTimeout(TLSErrInfo errInfo) {  
 /\* 短信登录过程中任意一步都可以到达这里，顾名思义，网络超时，可能是用户网络环境不稳定，一般让用户重试即可\*/  
 }  
};

### 2.6 刷新UserSig

作为一种优化体验，TLS SDK 不需要用户每次都要完成登录流程(密码登录或者短信登录)，通过调用 tlsHelper.needLogin("用户名") ，如果返回结果为 false, 即可以直接调用换票接口，拿到最新票据。否则，再根据用户选择完成完整的登录流程。



#### 初始化设置

变量设置:

TLSRefreshUserSigListener *refreshUserSigListener*;

#### 步骤一 获取已登录用户帐号

当用户已经登录的情况下才能刷新UserSig， 所以要先获得已登录用户的帐号：

// 获取最近的一个已登录用户

TLSUserInfo userInfo = *tlsHelper*.getLastUserInfo();  
  
 // 获取所有已登录用户  
 List<TLSUserInfo> userInfoList = *tlsHelper*.getAllUserInfo();

然后判断某个选择用户是否已登录：

boolean hasLogin = userInfo != null && !*tlsHelper*.needLogin(userInfo.identifier);

#### 步骤二 刷新UserSig

调用刷新票据接口：

if (hasLogin)  
 *tlsHelper*.TLSRefreshUserSig(*userID*, refreshUserSigListener);

TLSRefreshUserSig 可能的回调函数是**OnRefreshUserSigSuccess**, OnRefreshUserSigFail以及OnRefreshUserSigTimeout。具体实现请参考[监听函数实现](#_3.6.3_监听函数实现) 相关说明。

这里务必要做好对换票失败的处理，比如用户可能在别处修改了密码，那么就算本地保存的登录凭证就是无效的了，所以在回调函数OnRefreshUserSigFail中应该引导用户重新登录了。

#### 监听函数实现

*refreshUserSigListener* = new TLSRefreshUserSigListener() {  
 @Override  
 public void OnRefreshUserSigSuccess(TLSUserInfo userInfo) {  
 /\* 成功换票成功了，在这里可以获取用户票据\*/  
 String usersig = *tlsHelper*.getUserSig(userInfo.identifier);  
 }  
  
 @Override  
 public void OnRefreshUserSigFail(TLSErrInfo errInfo) {  
 /\* 换票失败了，需要引导用户重新登录。  
特别是errInfo.ErrCode == TLSErrInfo. LOGIN\_WRONG\_PWD的情况下，很可能是用户在别处修改了密码，此时TLS SDK会自行删除本地票据的。\*/  
 }  
  
 @Override  
 public void OnRefreshUserSigTimeout(TLSErrInfo errInfo) {  
 /\* 网络超时，可能是用户网络环境不稳定，一般让用户重试即可。\*/  
 }  
 };

### 2.7密码注册（字符串账号）

#### 初始化设置

变量设置:

TLSStrAccRegListener *strAccRegListener*;

#### 提交用户名和密码

// 引导用户输入合法的用户名和密码

int result = tlsHelper.TLSStrAccReg(*userID*, "password", *strAccRegListener*);

if (result == TLSErrInfo.*INPUT\_INVALID*)  
 displayInfo("引导用户输入合法的用户名和密码");

该接口可能的回调是**OnStrAccRegSuccess**, OnStrAccRegFail以及OnStrAccRegTimeout。具体实现请参考[监听函数实现](#_监听函数实现_1) 相关说明。

#### 监听函数实现

*strAccRegListener* = new TLSStrAccRegListener {  
 @Override  
 public void OnStrAccRegSuccess(TLSUserInfo userInfo) {  
 /\* 成功注册了一个字符串帐号， 可以引导用户使用刚注册的用户名和密码登录 \*/  
 }  
  
 @Override  
 public void OnStrAccRegFail(TLSErrInfo errInfo) {  
 /\* 注册失败，请提示用户失败原因 \*/  
 }  
  
 @Override  
 public void OnStrAccRegTimeout(TLSErrInfo errInfo) {  
 /\* 网络超时，可能是用户网络环境不稳定，一般让用户重试即可。\*/  
 }  
}

### 2.8 匿名登录(或称访客登录)

#### 判断访客状态

// 调用该接口可以判断当前访客登录状态，返回null说明暂时无访客登录，否则返回访客标识

String *guest = tlsHelper*.getGuestIdentifier()

#### 匿名登录

// 这里直接调用登录接口即可，该接口内部实现了自动注册，这也是tlsHelper需要初始化的原因。

// 注意：每次调用该接口都会重新注册访客账号，并在本地删除旧的访客账号，请谨慎

*tlsHelper*.TLSGuestLogin(new TLSGuestLoginListener() {  
 @Override  
 public void OnGuestLoginSuccess(TLSUserInfo userInfo) {  
 displayInfo("访客登录成功 " + userInfo.identifier);  
 }  
  
 @Override  
 public void OnGuestLoginFail(TLSErrInfo errInfo) {  
 notOK(errInfo);  
 }  
  
 @Override  
 public void OnGuestLoginTimeout(TLSErrInfo errInfo) {  
 notOK(errInfo);  
 }  
});

#### 刷新UserSig

与普通账号相同，请参见 [刷新UserSig](#_2.6_刷新UserSig).

### 2.9 第三方账号与自有账号绑定

以下API说明均与第三方账号（比如QQ、微信、新浪微博账号等）相关，故在使用之前需要首先初始化第三方账号的信息：

* 第三方账号类型open\_accounttype： 1：QQ，2：微信，3：新浪微博
* 第三方账号appid：需在第三方账号开放平台申请
* 第三方账号openid：第三方账号登录成功后会返回（微信例外）
* 第三方账号access\_token：第三账号登录成功后会返回（微信例外）

由于微信授权登录成功后只会返回code，之后需要使用code、appid、app\_secret到微信服务器换取，使用者在官网配置页面配置好appid和app\_secret。TLSSDK提供了如下接口用于使用微信授权登录成功后返回的code换取openid和access\_token。

#### Code换取openid和access\_token（仅用于微信）

tlsHelper.TLSOpenAccessToken(open\_accounttype, code, new TLSOpenAccessTokenListener() {  
 @Override  
 public void OnOpenAccessTokenSuccess(AccessTokenInfo accessTokenInfo) {

/\*\*

\* code换取openid和access\_token成功，可以通过accessTokenInfo.openid

\* 和accessTokenInfo.access\_token分别获取openid和access\_token

\* openid = accessTokenInfo.openid;  
 \* access\_token = accessTokenInfo.access\_token;

\*/  
 }  
  
 @Override  
 public void OnOpenAccessTokenFail(TLSErrInfo tlsErrInfo) {

/\*\*

\* code换取openid和access\_token失败

\*/  
 }  
  
 @Override  
 public void OnOpenAccessTokenTimeout(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* code换取openid和access\_token超时

\*/  
  
 }  
});

以下均假设用户已经在官网配置页面已经配置好第三方账号的appid和app\_secret等信息且用户第三方登录成功后获取了openid和access\_token。

#### 初始化

*/\*\*  
 \** ***@function*** *在使用与第三账号有关的功能时需要首先首先调用此接口进行  
 \* 初始化操作，后绪与第三方账号相关的信息均从此OpenAccountInfo对象  
 \* 中获取；如需切换第三方账号只需再次调用此接口即可。  
 \** ***@param*** *open\_accounttype 第三方账号类型，QQ: 1，微信：2  
 \** ***@param*** *appid 第三方账号appid  
 \** ***@param*** *openid 第三方账号openid  
 \** ***@param*** *access\_token 第三方账号access\_token  
 \* \*/*tlsHelper.setOpenAccountInfo(open\_accounttype, appid, openid, access\_token);

或者

tlsHelper.setOpenAccountInfo(new OpenAccountInfo(open\_accounttype, appid, openid,

Access\_token));

注意每次使用不同的第三方账号时均需要重新调用此接口进行第三方账号信息初始化，可以通过如下接口获取你最近设置的第三方账号信息

tlsHelper.getOpenAccountInfo();

#### 第三方账号登录

tlsHelper.TLSOpenAccountLogin(new TLSExchangeTicketListener() {  
 @Override  
 public void OnExchangeTicketSuccess(TLSUserInfo tlsUserInfo) {  
 /\*\*

\* 登录成功

\*/  
 }  
  
 @Override  
 public void OnExchangeTicketFail(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* 登录失败

\*/  
 }  
  
 @Override  
 public void OnExchangeTicketTimeout(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* 登录超时

\*/  
 }  
});

#### 查询第三方账号的绑定状态

tlsHelper.TLSOpenAccountQuery(new TLSOpenAccountQueryListener() {  
 @Override  
 public void onUnlogined(OpenAccountInfo openAccountInfo) {  
 /\*\*

\* 第三方账号从未登录过，此时可以与未注册的自有账号绑定（参见带第三方账号信息注册），

\* 也可与已注册的自有账号绑定（参见[第三方账号与已注册的自有账号绑定](#_第三方账号与已注册的自有账号绑定)）

\*/  
 }  
  
 @Override  
 public void onLoginedButUnbinded(OpenAccountInfo openAccountInfo) {  
 /\*\*

\* 第三方账号已登录过但尚未绑定自有账号，此时只可与未注册的自有账号绑定，

\* 不可与已注册的自有账号绑定

\*/  
 }  
  
 @Override  
 public void onLoginedAndBinded(OpenAccountInfo openAccountInfo) {  
 /\*\*

\* 第三方账号已登录过且已绑定自有账号

\*/  
 }  
  
 @Override  
 public void onFailure(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* 查询失败

\*/  
 }  
  
 @Override  
 public void onTimeOut(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* 查询超时

\*/  
 }  
});

#### 第三方账号与已注册的自有账号绑定

*/\*\*  
 \* 绑定第三方账号和自有账号  
 \** ***@param*** *identifier 自有账号的标识符  
 \** ***@param*** *usersig 自有账号票据，可以通过getUserSig()接口获取  
 \** ***@return*** *0 表示调用成功  
 \* \*/*

tlsHelper.TLSOpenAccountBind(identifer, usersig, new TLSOpenAccountBindListener() {  
 @Override  
 public void onBindSuccess(OpenAccountInfo openAccountInfo) {  
 /\*\*

\* 绑定成功

\*/  
 }  
  
 @Override  
 public void onBindFailure(OpenAccountInfo openAccountInfo) {  
 /\*\*

\* 绑定失败

\*/  
 }  
  
 @Override  
 public void onFailure(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* 发生错误

\*/  
 }  
  
 @Override  
 public void onTimeOut(TLSErrInfo tlsErrInfo) {  
 /\*\*

\* 发生超时

\*/  
 }  
});

#### 带第三方账号信息注册

以下带有后缀”WithOA”的注册接口与不带此后缀的注册接口调用方式完全一致，除了在调用这些接口之前需要进行[初始化](#_初始化)第三方账号信息，上文说过此初始化只需要在使用与第三方账号相关接口时调用一次即可。因此下面仅仅列出这些接口，其调用方法参考相应的不带”WithOA”的注册接口，比如TLSSmsRegAskCodeWithOA的调用方法可以参考TLSSmsRegAskCode。

// 带第三方账号信息的手机号短信注册

tlsHelper.TLSSmsRegAskCodeWithOA(String mobile, TLSSmsRegListener listener);

// 带第三方账号信息的手机号密码注册

tlsHelper.TLSPwdRegAskCodeWithOA(String mobile, TLSPwdRegListener listener);

// 带第三方账号信息的字符串账号密码注册

tlsHelper.TLSStrAccRegWithOA(String account, String password,

TLSStrAccRegListener listener);

### 2.10 获取本地保存的SSO票据

用户登录成功后（包括手机号短信登录、手机号密码登录、字符串账号登录、匿名账号登录、第三方账号登录），均可以通过**getSSOTicket()**接口获取包括A2/A2Key，D2/D2Key，内部标识符和TinyID信息，这些信息通过键值对返回，调用参数为用户标识符。

*/\*\*  
 \* 获取本地保存的SSO票据  
 \** ***@param*** *identifier 用户帐号, 第三方开放平台则直接传入 openid 等标识  
 \** ***@return*** *{"A2": xxx, "A2Key": xxx, "D2": xxx, "D2Key": xxx, "identifier": @tls#xxx, "tinyID": xxxx}  
 \*/*public Map<String, Object> getSSOTicket(String identifier);