VPN SDK

Version 2.0.1

2022-01-06

目录

目录 2

综述 3

VPN 生命周期 3

集成 VPN SDK 至项目中 4

复制 so 文件到项目 jniLibs 文件夹 4

在项目 java 文件夹下新建包 net.vpnsdk.vpn 5

初始化 6

设置消息监听器 mHandler 7

启动 VPN 8

停止 VPN 9

认证 9

获取认证信息 10

创建登录界面 11

提交认证信息 11

认证失败处理 15

SSL 优先协议设置 15

SKF SDK 集成步骤 16

复制 so 文件到项目 jniLibs 文件夹 16

关联所需依赖库 17

初始化 17

具体方法解释 17

添加白名单应用 18

其它 18

调试 18

兼容性 18

常见错误码说明 18

注意 & 常见问题 20

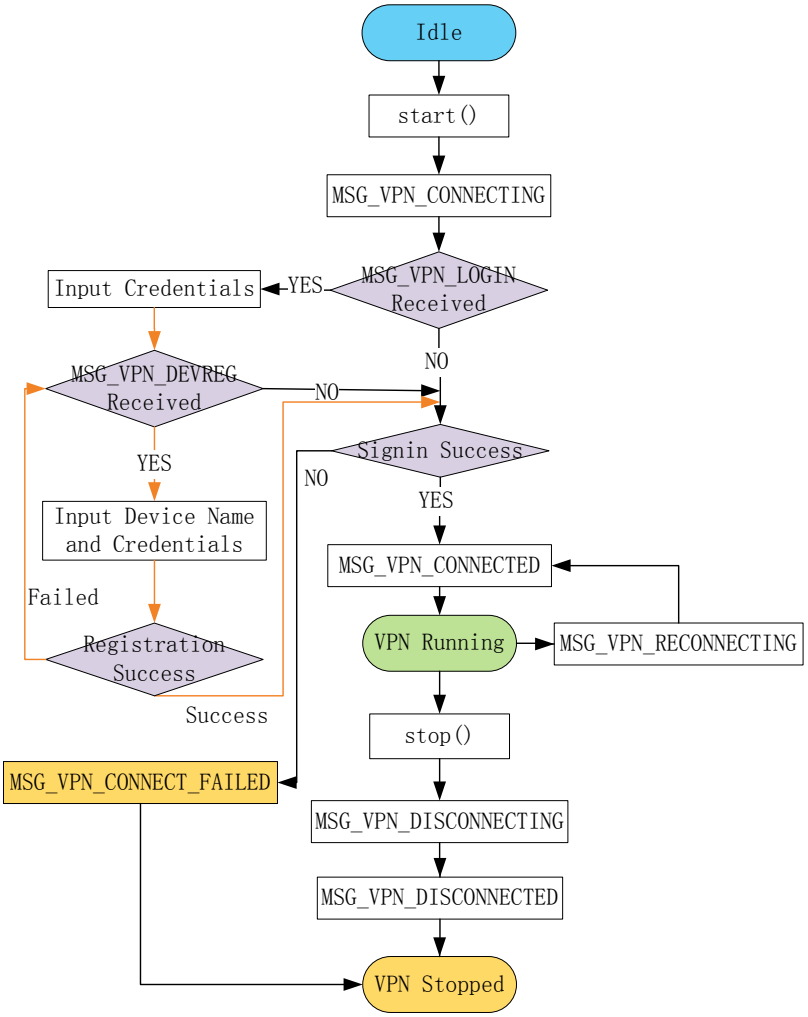
综述

本文介绍面向 Android 平台的 VPN SDK 基本原理和使用方法。
VPN SDK 提供了一套简单易用的 API，帮助开发者在移动平台上建立与 AG 服务器相连接的 VPN 加密隧道。

VPN SDK 支持多种认证方式，包括单用户名密码认证，多因素认证，证书认证，设备 ID 认证，以及由上述各认证方式组合而成的多方法认证。SDK 提供接口支持用户输入各种认证凭证。只有认证通过，才能建立 VPN 隧道。

VPN 生命周期

VPN SDK 的用法主要包括两部分，一是调用函数 startVPN()和 stopVPN()控制 VPN 的启动和停止；二是处理各种 VPN 消息。
VPN 的整体生命周期如下图所示：



从上图可以看出，使用 SDK 建立 VPN 隧道，主要工作是处理 VPN 生命周期中的各种消息。

上图中黑色箭头线所示的路径是单方法认证时 VPN 的运行流程。最初 VPN 处于空闲状态，在调用了 `startVPN(host,port,username,password)` 之后，VPN 进入 Connecting 状态，SDK 将与指定的服务器建立 SSL 连接，并将用户名密码发送至服务器作认证。认证通过后，VPN 隧道将被建立，并以 Connected 消息通知应用程序。因网络原因导致连接中断后，VPN 将自动重连，并发送 Reconnecting 消息，当再次连接成功后，应用程序会再次收到 Connected 消息。

应用程序调用 `stopVPN()` 停止 VPN 连接。应用程序首先收到 Disconnecting 消息，成功登出之后，将收到 Disconnected 消息。

图-1 中橙色路线是多方法认证以及 `DeviceID` 认证时 VPN 的运行流程。多方法认证无法在启动 VPN 时就输入认证凭证，而是在收到 MSG_VPN_LOGIN 消息时，应用程序会收到服务器传来的多个认证方法供用户选择，用户输入认证凭证后，应用程序将认证凭证传入 SDK，再由 SDK 发送至服务端作认证，即可完成登录。

DeviceID 认证在用户首次登录时，需要注册设备，应用程序在收到 MSG_VPN_DEVREG 消息后，将用户输入的设备名称以及注册所需的用户名密码传入 SDK，即可完成注册。

图-1 中 “Input User Info”和“Input Device Name”行为分别用于处理 MSG_VPN_LOGIN 和 MSG_VPN_DEVREG 两个消息。

有关认证的详细内容请参见身份认证一节。

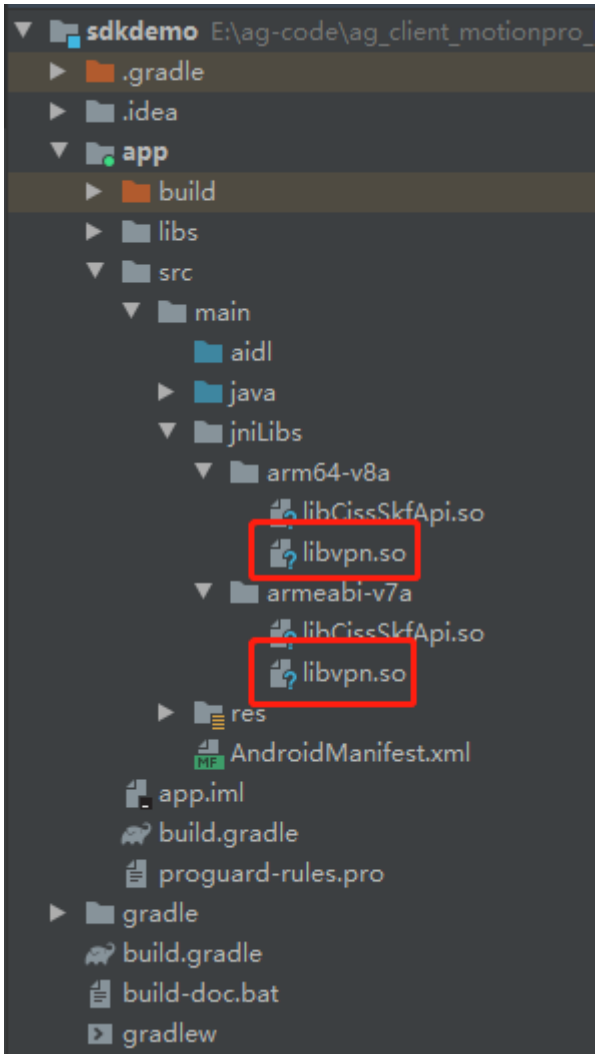
VPN SDK 集成

一、VPN SDK 集成步骤

1.1 集成 VPN SDK 至 项目中

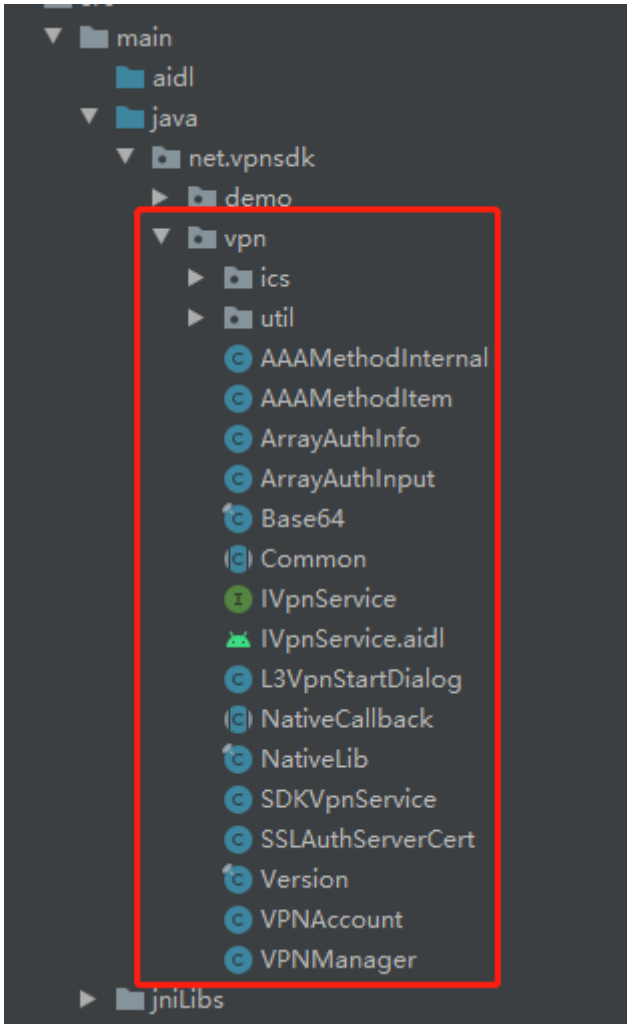
1.1.1 复制 so 文件到项目 jniLibs 文件夹

说明
所需 so 库：libvpn.so，如下图：



1.1.2 在项目 java 文件夹下新建包“net.vpnsdk.vpn”

> **说明**
> 将 demo 中 net.vpnsdk.vpn 包下所有文件 copy 到项目中，目录层级结构如下图：



1.1.3 初始化

> **说明**
> 初始化代码、清单文件、资源文件、依赖项

代码

```
// 代码中初始化：Application.onCreate()
VPNManager.initialize(this);

// 清单文件：AndroidManifest.xml
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<service android:name="net.vpnsdk.vpn.SDKVpnService"
    android:exported="false"
    android:permission="android.permission.BIND_VPN_SERVICE"
    android:stopWithTask="false">
    <intent-filter>
        <action android:name="net.vpnsdk.vpn.VPN_SERVICE" />
```

```
        <action android:name="android.net.VpnService" />
    </intent-filter>
</service>
<activity android:name="net.vpnsdk.vpn.L3VpnStartDialog"
    android:excludeFromRecents="true"
android:theme="@style/CustomNoDisplay"  />

// 资源文件：styles.xml
<style name="CustomNoDisplay">
    <item name="android:windowBackground">@null</item>
    <item name="android:windowContentOverlay">@null</item>
    <item name="android:windowIsTranslucent">true</item>
    <item name="android:windowAnimationStyle">@null</item>
    <item name="android:windowDisablePreview">true</item>
    <item name="android:colorBackgroundCacheHint">@null</item>
    <item name="android:windowNoTitle">true</item>
</style>

// 依赖文件：app/build.gragle
sourceSets {
    main {
        jniLibs.srcDirs = ['src/main/jniLibs']
    }
}
```

1.1.4 设置消息监听器 mHandler

说明

应用程序通过接收 VPN 消息来监听 VPN 生命周期中的关键事件，例如，VPN 连接成功，登录失败，网络原因导致的 VPN 重连等。VPN SDK 提供以下消息。

代码

```
private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case VpnMsg.MSG_VPN_CONNECTING:
            // 正在连接服务器。一般情况只需显示进度条
            break;
        case VpnMsg.MSG_VPN_CONNECTED:
            // 连接成功，隧道已建立。可以使用 VPN 隧道进行安全通信了
            break;
        case VpnMsg.MSG_VPN_DISCONNECTING:
            // 正在登出。一般情况只需显示进度条
```

```
            break;
        case VpnMsg.MSG_VPN_DISCONNECTED:
            // 连接断开，Session 结束。显示登录界面，关闭程序等
            break;
        case VpnMsg.MSG_VPN_CONNECT_FAILED:
            // 登录失败。提示用户失败原因
            break;
        case VpnMsg.MSG_VPN_RECONNECTING:
            // 连接断开，正在进行重连。提示用户或不处理
            break;
        case VpnMsg.MSG_VPN_LOGIN:
            // 需要用户输入登录凭证。获取用户输入的用户名密码
            break;
        case VpnMsg.MSG_VPN_DEVREG:
            // 需要用户输入认证凭证及设备名称。将用户输入的用户名密码及设备名称传给 SDK
            break;
        default:
            super.handleMessage(msg);
        }
    }
};

VPNManager.getInstance().setHandler(mHandler);
```

1.1.5 启动 vpn

说明

关于 host，请参考 MainActivity.java 文件，既可以支持 java 探测多个站点相应时间，又可以支持 c-sdk 自动探测最优站点. 如果 host 中包含@符号则 C-sdk 自动探测每一个站点，并且使用最快的站点连接 VPN；

例如: host: 10.8.7.123:443@10.8.7.124:8443 ,则直接传给 startVPN 即可.这种情况下，startVPN 中的 port 参数可以设置为 443 即可，sdk 第一优先级使用@符合前后的端口；

Host 格式: 10.8.7.123@10.8.7.124 10.8.7.123:8443@10.8.7.124:8444。

函数 startVPN()的部分重载版本包括一个名为 flag 的参数。该参数的值可以是类 Common.VpnFlag 成员变量的组合。

代码

```
Thread t = new Thread(new Runnable() {
    @Override
    public void run() {
        VPNManager.getInstance().startVPN(host, port, username, password);
```

```
    }
});
t.start();
```

注意

VPN 的启动包含网络通信，不能在 UI 线程中执行。一个应用程序中只能存在一个 VPN 隧道。

1.1.6 停止 vpn（须在子线程中）

代码

```
Thread t = new Thread(new Runnable() {
    @Override
    public void run() {
        VPNManager.getInstance().stopVPN();
    }
});
t.start();
```

注意

VPN 的启动包含网络通信，不能在 UI 线程中执行。

1.2 认证

说明：

AG 支持多种认证方式，包括单用户名密码的认证，比如 LocalDB、LDAP、RADIUS 等，以及不需要密码的认证，比如证书认证、DeviceID 认证等，以及由这些认证方式组合而成的多因素认证和多方法认证。

AG 的认证体系中包含两个基本概念，认证服务器(Server)和认证方法(Method)。

认证服务器(Server)：

认证服务器表示对客户端提交的某一项身份凭证作认证的服务器。例如，启用了 LDAP 认证，则 LDAP 服务器会对用户提交的用户名和相应的密码作认证。

认证方法(Method)：

一个方法可包含至少 1 个，最多 3 个认证服务器，认证时用户提交的凭证必须通过所有服务器的认证才能成功。包含多个认证服务器的认证方法被称为多因素认证。一个站点可配置多个方法，用户选择其中一个方法作认证即可。

一个方法中如果包含多个需要提供用户名的认证服务器，则用户名必须相同。但密码是相互独立的。例如，管理员配置了一个认证方法 Method A，包含两个认证服务器 Local DB 和 LDAP，则用户需要输入的认证凭证包括一个用户名“Username”，以及两个分别对应于 Local DB 和 LDAP 的密码“Password1”和“Password2”。

方法与服务器的关系如图-3 所示：

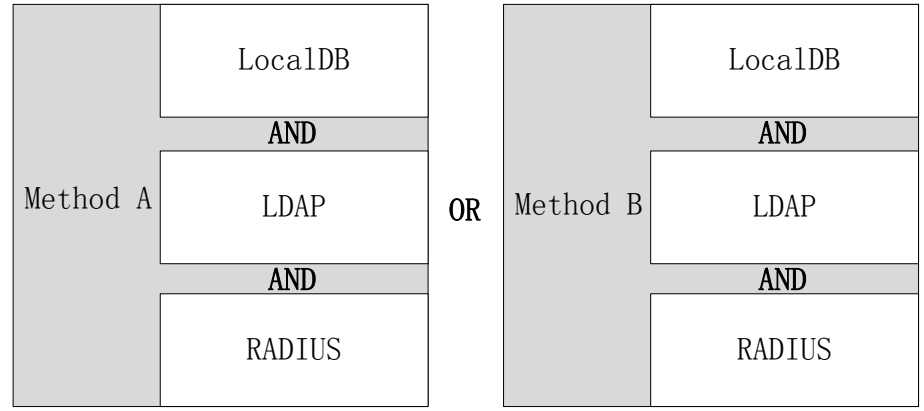


图-3 认证方法与认证服务器

管理员在服务端配置认证方法后，客户端的开发人员可通过 SDK 完成用户认证，步骤如下：

1.2.1 获取认证信息

说明

SDK 通过发送 MSG_VPN_LOGIN 消息提供认证信息

代码

```
private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case VpnMsg.MSG_VPN_LOGIN:
            Methods = (AAAMethod[]) (msg.obj);
            if (0 == mMethods.length) {
                VPNManager.getInstance().cancelLogin();
            }
            …… …… ……
        break;
        default:
            super.handleMessage(msg);
        }
    }
}
```

注意

上述代码中 msg.obj 对象实际上是一个 AAAMethod 对象数组。AAAMethod 类是对认证方法的抽象，该类包含一个 InputField 类的列表，InputField 是对认证服务器的抽象，该类最重要的任务是接收用户输入的身份凭证，比如用户名，设备名，以及和各认证服务器相对应的密码。

在调用 startVPN()启动 VPN 之后，收到消息 MSG_VPN_CONNECTED 之前，可通过调用 VPNManager.getInstance().cancelLogin()取消登录。上例中认证方法的数量是 0，说明管理员对认证的配置有误，至少应有 1 个认证方法，因此取消登录。

1.2.2 创建登录界面

说明

根据上一步得到的认证信息可以创建登录界面，从而让用户输入用户名和密码等身份凭证。调用 AAAMethod.getName()和 AAAMethod.getDescription()可分别得到管理员在服务端配置的认证方法名称和描述。AAAMethod 数组的长度代表认证方法的数量。根据这些信息，我们可以创建如下图所示的界面。

从图-4 中可以看出，服务端配置了 3 个认证方法，分别是“DD_Auth”，“localdb”，“MyMethod”。从下拉框中选择认证方法“MyMethod”之后，界面如图-5 所示。认证方法“MyMethod”包含两个认证服务器，管理员将其分别命名为“mn”和“ldap”。

调用函数 AAAMethod.getInputFieldCount()可以得到该方法所需输入框的数量。调用函数 AAAMethod.getInputField(int index)获取每一个 InputField 对象。InputField.getName()和 InputField.getDescription()分别用于获取认证服务器的名称和描述。如图-5 中所示“mn”和“ldap”即为获取到的认证服务器名称。

通过调用 InputField.getType()得到输入项的类型 InputFieldType，如果类型的枚举值是 USERNAME 表明该 InputField 对象需要输入用户名；如果类型是 PASSWORD 表明该 InputField 对象需要输入密码。根据这些信息可以创建用户选择了某个认证方法之后的界面（如图-5 所示），用户根据提示输入用户名及对应的密码，点击登录按钮即可完成登录。

图-4 显示认证方法

图-5 选择认证方法

1.2.3 提交认证信息

1.2.3.1 提交用户名和密码

说明

首先调用函数 InputField.setInputString(String input)将输入的用户名和密码等信息传入 InputField 对象。以图-5 中的“MyMethod”方法为例，将用户名传入类型是 USERNAME 的 InputField 对象；将用户输入的两个密码分别传入对应的“mn”和 “ldap”的 InputField 对象。示例代码如下。

代码

```
for (int i = 0; i < method.getInputFieldCount(); i++) {
    field = method.getInputField(i);
    if (InputFieldType.DEVICENAME == field.getType()) {
        // set device name
        devName = edtDevName.getText().toString().trim();
        field.setInputString(devName);
    } else if (InputFieldType.USERNAME == field.getType()) {
        // set user name
        username = edtUserName.getText().toString().trim();
        field.setInputString(username);
    } else if (InputFieldType.PASSWORD == field.getType()) {
        // set password, note, there may be more than one passwords,
        // please make sure to set correct password according to
        // the name of InputField. In this example, we keep the same
        // order of EditText list and InputField list, and use an
        // index to make sure the password is correct.
        edtPassword = mPwdEdits.get(passwordIndex);
        password = edtPassword.getText().toString();
        field.setInputString(password);
        passwordIndex++;
    }
}

// 最后，当用户点击登录按钮后，必须调用以下代码
// 其中 method 对象是用户选择的认证方法，且该对象中所有的 InputField 已经被赋值用户输入
VPNManager.getInstance().loginWithMethod(method);
```

注意

处理 MSG_VPN_LOGIN 消息时，loginWithMethod()函数必须被调用，否则 VPN 线程将一直处于阻塞状态。若要取消登录，请调用 cancelLogin()。

1.2.3.2 SMS 认证，提交验证码

说明

SDK 通过回调 onVpnSms()方法提示程序需用户输入验证码，用户输入后将验证码直接返回即可。

```java
/**
 * Function to handle device register event.
 */
@Override
String onVpnSms(String result, String message)
{
    Log.d( tag: "array",  msg: "onVpnSms");
    //第三方app 这里要弹出对话框，让用户输入信息，12345678 就是用户输入的信息  // 格式： f1:"短信口令"
    String sms = "f1:\"123454678\"";
    Log.d( tag: "array",  msg: "onVpnSms: " + sms);
    return sms;
}
```

注意

返回的字符串格式为："f1:12345678"，其中"12345678"为用户输入的验证码。

1.2.3.3 Validcode 认证，提交 Validcode

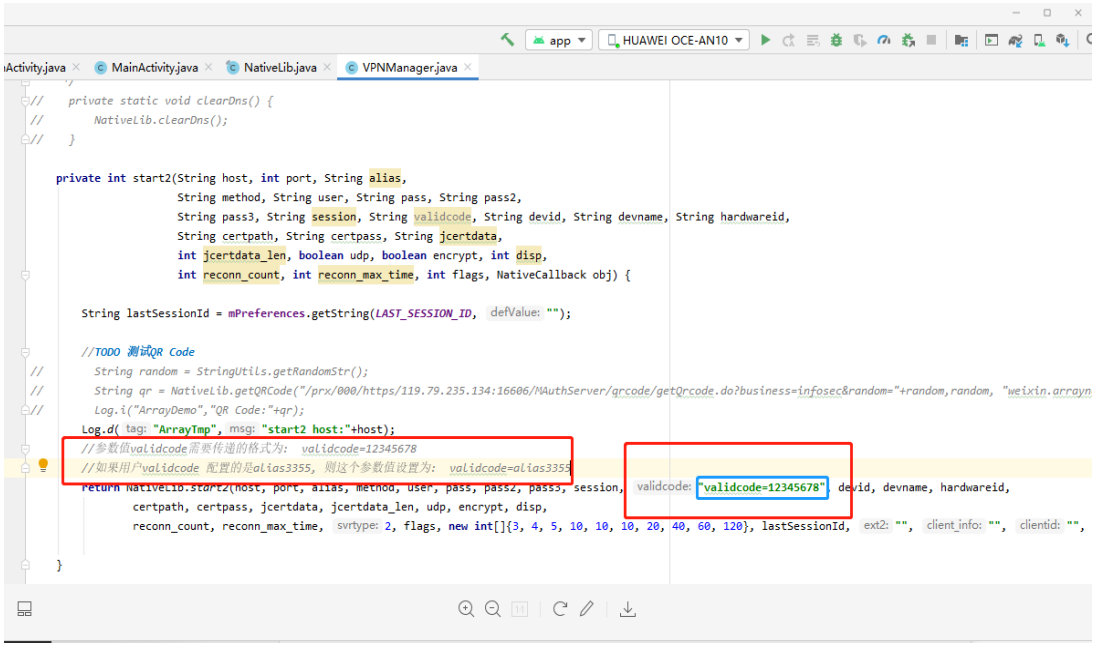
说明

参数值 validcode 需要传递的格式为: validcode=12345678

如果用户 validcode 配置的是 alias3355，则这个参数值设置为: validcode=alias3355

Demo 代码里面找到 VPNManaer.java 里面修改 NativeLib.start2（）即可.

一般 validcode 每个虚拟站点是固定的，不需要终端用户输入，因此这里设置为固定值



1.2.3.4 DeviceID 认证

说明

登录过程中，SDK 会自动收集 DeviceID 并发送至服务器。DeviceID 对每一台设备都是唯一的。当管理员启用了 DeviceID 认证后，客户端首次登录时，需要注册设备。应用程序将收到 MSG_VPN_DEVREG 消息。对该消息的处理和 MSG_VPN_LOGIN 大体一致，区别在于

需要输入设备名称。如果 InputField 的类型是 DEVICENAME，则该 InputField 对象用于接收用户输入的设备名称，代码如上例所示。设备注册的示例界面如图-6。

图-6 设备注册界面

代码

```
// 用户输入完毕，点击注册按钮后，必须执行下述代码。
VPNManager.getInstance().registerWithMethod(method);
```

注意

处理 MSG_VPN_DEVREG 消息时，retisterWithMethod()函数必须被调用，否则 VPN 线程将一直处于阻塞状态。若要取消登录，请调用 cancelLogin()。

1.2.3.5 证书认证

说明

管理员启动证书认证之后，客户端可以通过以下 2 种方式提供证书

其中 certPath 是证书文件的本地路径，注意证书路径需要时绝对路径，比如“/data/data/net.vpnsdk.demo/files/test.p12”，SDK 仅支持.p12 和.pfx 格式的证书。certPassword 是证书文件的密码。

如果证书认证失败，消息 MSG_VPN_CONNECT_FAILED 会被发送。通过以下代码可以得到错误码，msg.getData().getInt(VpnMsg.MSG_VPN_ERROR_CODE)，错误码的具体内容可参见 Common.VpnError。调用 getErrorInfo()获取错误描述。

代码

```
// 方式一
startVPN(String host, int port, String username,
          String password, String certPath, String certPassword);

// 方式二
VPNAccount auth = new VPNAccount();
auth.setUsername(username);
```

```
auth.setHost(server);
auth.setPort(port);
auth.setPassword(password);
auth.setCertPath(certPath);
auth.setCertPass(certPassword);
startVPN(VPNAccount auth);
```

1.2.4 认证失败处理

说明

如果在调用 startVPN()时没有提供用户名密码，或提交的用户名密码错误，将导致 MSG_VPN_LOGIN 消息被发送。代码如下。

代码中 msg.getData().getInt(VpnMsg.MSG_VPN_ERROR_CODE) 将返回错误码，错误码在 Common.VpnError 中定义。调用 getErrorInfo()可得到错误描述。

用户名密码认证失败时，大部分应用的处理方式是让用户再次输入用户名密码，因此认证失败时 SDK 通过发送 MSG_VPN_LOGIN 消息通知应用程序。

代码

```
private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case VpnMsg.MSG_VPN_LOGIN:
            int error = msg.getData().getInt(VpnMsg.MSG_VPN_ERROR_CODE);
            String[] msg = VPNManager.getInstance().getErrorInfo(error);
            Log.e(Tag, "information " + msg[0] + "\n" +
                       "suggestion " + msg[1]);
            …… ……
        break;
        default:
            super.handleMessage(msg);
        }
    }
}
```

1.3 SSL 优先协议设置

说明

调用 setSslProtocol(int sslProtocol)函数设置优先协议，默认为 TLSv1.2；

代码

```
// 协议对应的常量值定义在 VPNManager 中。
```

```
public final int SSL_PROTOCOL_INIT = 0;
public final int SSL_PROTOCOL_TLS10 = 1;
public final int SSL_PROTOCOL_TLS12 = 2;
public final int SSL_PROTOCOL_SSLV3 = 3;
public final int SSL_PROTOCOL_SM2 = 4;

NativeLib.setSslProtocol(int sslProtocol);
```

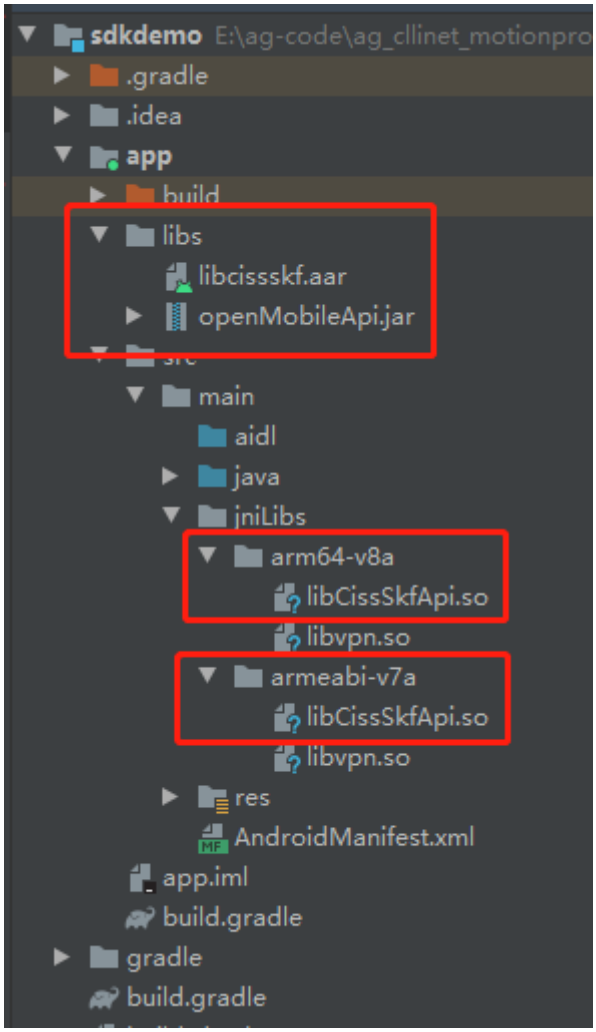
二、SKF SDK 集成步骤（不需要 SKF 认证略过此步骤）

2.1 复制 so 文件到项目 jniLibs 文件夹，如下图：

说明

所需 so 库文件及 jar 包：

1. libCissSkfApi.so
2. libcissskf.aar
3. openMobileApi.jar



2.2 关联所需依赖库，如下图

代码

```
implementation 'org.bouncycastle:bcprov-jdk15on:1.62'
implementation files('libs/libcissskf.aar')
compileOnly files('libs/openMobileApi.jar')
```

2.3 初始化

说明

初始化代码、清单文件

代码

```
// 代码中初始化：Application.onCreate()
SDKConfig.setStartSkf(true); //是否开启 skf 认证
boolean scanBleNow = false;//是否进行蓝牙扫描，否表示不进行蓝牙扫描
CISSSKFInit.init(this, scanBleNow); //初始化之后可以设置日志显示与否，默认不显示
OMAAPDUSenderForSKF.getInstance().setLogAble(true);

// 配置清单文件
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.NFC" />
<uses-permission android:name="android.permission.SMARTCARD" />
<uses-permission android:name="org.simalliance.openmobileapi.SMARTCARD" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-library
    android:name="org.simalliance.openmobileapi"
android:required="false" />
```

2.4 具体方法解释

说明

相关函数均声明在 VPNManager 中，调用前需导包：net.vpnsdk.vpn.VPNManager

代码

```
// 枚举设备列表，返回设备列表字符串，以“,”分割
String simEnumDev();

// 枚举 Application 列表，返回 Application 列表字符串，以“,”分割
```

```
String simEnumApp(String devName);

// 枚举 Container 列表，返回 Container 列表字符串，以“,”分割
String simEnumContainer(String appName);

// 校验 pin 码，返回值 0 表示 pin 码正确
int simCheckPinKey(String pinKey)

// 设置 sim 信息
simSetInfo(String selectedDevName, String selectedAppName, String
            selectedContainerName, String pinKey)

// Pin 码是否已经输入 返回值 0 表示 pin 码已经正确输入
isPinKey()
```

三、添加白名单应用

说明

支持添加白名单应用列表。如果列表中包含一个或多个应用，则只有列表中的应用才会使用 VPN。（不在列表中的）所有其它应用都将使用系统网络。如果列表为空，则所有应用都将使用 VPN。默认列表为空。

代码

```
String[] arr = new String[]{"白名单应用包名"};
VPNManager.getInstance().addAllowedApplication(arr);

//调用示例
//添加一个白名单应用
String[] arr = new String[]{"com.android.chrom"};
//添加两个或多个白名单应用
String[] arr = new String[]{"com.android.chrom", "com.android.aiqiyi"};
VPNManager.getInstance().addAllowedApplication(arr);
```

四、其它

3.1 调试

3.1.1 使用 ADB 命令查看 SDK 的 Logcat 日志

```
adb logcat
```

3.1.2 设置日志级别，默认为 INFO 级

```
mVpnManager.setLogLevel(Common.LogLevel.LOG_INFO, 0);
mVpnManager.setLogLevel(Common.LogLevel.LOG_DEBUG, 0);
```

3.2 兼容性

SDK 版本	Supported Android Platform version	Supported CPU Architecture
2.0.1	>= Android 4.4	ARM v-7

3.3 常见错误码说明

错误码定义在 net.vpnsdk.vpn.Common.VpnError。

ERR_SUCCESS	成功
ERR_FAILURE	失败
ERR_SESS_INVALID	***VPN*** 会话无效，需要重新登陆
ERR_SESS_TIMEOUT	***VPN*** 会话超时，需要重新登陆
ERR_SOCK_CONN	建立 ***socket*** 连接失败，常见原因是网络不通
ERR_SERVER_CONFIG	服务器配置错误
ERR_WRONG_USER_PASS	用户名或密码错误
ERR_CONFIG_L3VPN	配置***VPN***隧道失败，常见原因是没有建立***VPN***的权限，详情请参考*https://developer.android.google.cn/reference/android/net/VpnService*，这时需要重新调用VPNManager.*getInstance*().bindVPNService("net.vpnsdk.vpn.VPN_SERVICE")(VPNManager内部调用android.net.Vp

	nService.prepare())来获取建立VPN的权限
ERR_GET_L3VPN_CONFIG	获取 ***VPN*** 配置失败，常见原因是服务器配置错误
ERR_INTERRUPTED	用户主动停止 ***VPN***
ERR_TUN_DOWN	网卡被停止，常见原因是用户通过 ***android*** 的通知栏断开 ***VPN***。这时需要重新调用*VPNManager.getInstance().bindVPNService("net.vpnsdk.vpn.VPN_SERVICE")(VPNManager*内部调用*android.net.VpnService.*prepare())来获取建立VPN的权限
ERR_CERT_NO	没有提供证书
ERR_CERT_INVALID_SIGNTURE	证书签名不合法
ERR_CERT_UNTRUSTED	证书不受信任
ERR_CERT_EXPIRED	证书已经过期
ERR_CERT_INVALID	证书不合法
ERR_CERT_REVOKED	证书被吊销

3.4 注意 & 常见问题

3.4.1 混淆

```
// 如果需要混淆代码，为了保证 sdk 的正常使用，需要在混淆文件中加上下面混淆规则：
-keep class net.vpnsdk.vpn.** { *; }
```

3.4.2 代码初始化

SDK 初始化代码应在启动 vpn 的 `Activity.onCreate()` 中执行。

SDK 初始化代码：

```
VPNManager.getInstance().bindVPNService("net.vpnsdk.vpn.VPN_SERVICE");
```

具体可参考 sdkdemo。

本文档对 **SDK** 集成步骤做了详细说明，具体实例代码可参考 **sdkdemo**。