# ThunderSEC EMM SDK

# 开发文档

版本:**ThunderSec 3.0**

日期:**2016-07-19**

# ThunderSEC EMM SDK 方法列表

| 模块 | 内容 | 方法名 |
| --- | --- | --- |
| **模块 1 状态查询** | root 状态查询 | boolean isDeviceRoot() |
| | Selinux 状态查询 | String checkSELinuxStatus() |
| **模块 2 配置管控** | 应用启动黑名单（配置） | boolean addAPPBlackList() |
| | 应用启动黑名单（获取） | String[] obtainAppBlackList() |
| | 应用启动黑名单（删除） | boolean deleteAppBlackList() |
| | 应用启动白名单（配置） | boolean addAppWhiteList() |
| | 应用启动白名单（获取） | String[] obtainAppWhiteList() |
| | 应用启动白名单（删除） | boolean deleteAppInstallWhiteList() |
| | 移动数据（启用〔true〕/禁用〔false〕） | boolean setMobileData() |
| | 强制关机 | boolean shotDown() |
| **模块 3 策略管控** | 设置 OTG 策略 | boolean setOTGPolicy() |
| | 设置移动数据策略 | boolean setMobileDataPolicy() |
| | 设置摄像头策略 | boolean setCameraPolicy() |
| | 设置 USB 策略 | boolean setUSBPolicy() |
| | 设置麦克风策略 | boolean setVoicePolicy() |
| | 设置蓝牙策略 | boolean setBlueToothPolicy() |
| | 设置 NFC 策略 | boolean setNFCPolicy() |
| | 设置 WIFI 策略 | boolean setWifiPolicy() |
| | 设置截屏策略 | boolean setScreenshotPolicy() |
| | 设置共享策略 | boolean setSharePolicy() |
| | 设置 GPS 策略 | boolean setGPSPolicy() |
| | 设置 APP 安装管控名单 | boolean setAppInstallPolicy() |
| | 设置 App 启动管控名单 | boolean setAppLauncherPolicy() |
| | 设置网络管控名单 | boolean setNetListPolicy() |
| | 设置 SD 卡策略 | boolean SDCardPolicy() |

# ThunderSEC EMM SDK 开发说明

## 模块 1 状态查询（status）

**简介**

状态查询类接口主要实现对设备及 SafeApiTool 的状态和版本进行查询的功能。主要功能包括：设备安全状态查询（root 状态、SElinux 状态）、SafeApiTool 工具版本查询等。

跟随本使用说明一同打包的文件包括，如下：

**接口使用**

root 状态查询：

```
public boolean isDeviceRoot() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用isRootDevice()接口获得设备的root状态
     */
    boolean isRoot = mSettingsManager.isRootDevice();
    /*mSaver
     * 返回值为true时，表示当前设备已经root
     * 返回值为false时，则当前设备未root
     */
    return isRoot;
}
```

Selinux 状态查询：

```
public String checkSELinuxStatus() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
```

```
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用getSeLinuxStatus()接口获得设备的SELinux状态
     */
    String selinuxStatus = mSettingsManager.isRootDevice();
    /*
     * 返回值有三种不同状态：disabled,permissive,enforcing
     */
    return selinuxStatus;
}
```

## 模块 2 配置管控（configuration）

**简介**

配置管控类接口主要用于对设备各系统功能模块进行自定义配置。可以进行添加配置、删除配置、查询配置、更新配置等操作。**（特别的：在该模块中对“应用启动黑白名单”、“应用安装黑白名单”、“网络黑白名单”的配置，需要在模块 3 中进行策略选择后方能生效。）**

**接口使用**

应用启动黑名单（配置）

```
//应用启动黑名单（配置）
public boolean addAPPBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用addAppToBlackList(String packageName, int zId)接口
     * 参数：packageName:要配置为黑名单的应用的包名，如“com.thundersoft.app”
     *       zId:默认为 0;
     */
    boolean isSucess = mSettingsManager.addAppToBlackList("com.thundersoft.app", 0);
    /*
     * 返回值为true时，表示应用启动黑名单配置成功
     * 返回值为false时，表示应用启动黑名单配置失败
     */
    return isSucess;
}
```

应用启动黑名单（获取）

```
//应用启动黑名单（获取）
public String[] obtainAppBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用getAppArrayOnBlackList(int zId)接口获取配置过的所有应用启动黑名单
     * 参数：zId:默认为 0;
     */
    String[] array = mSettingsManager.getAppArrayOnBlackList(0);
    /*
     * array数组中存储的是所有配置过的黑名单
     */
    return array;
}
```

应用启动黑名单（删除）

```
//应用启动黑名单（删除）
public boolean deleteAppBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用removeAppFromBlackList(String packageName, int zId)
     * 参数：packageName:要从应用启动黑名单列表删除的应用包名，如：“com.thundersoft.app”
     *      zId:默认为 0
     */
    boolean isSucess = mSettingsManager.removeAppFromBlackList("com.thundersoft.app", 0);
    /*
     * 返回值为true时，表示删除指定的黑名单成功
     * 返回值为false时，表示删除指定的黑名单失败
     */
    return isSucess;
}
```

应用启动白名单（配置）

```
//应用启动白名单（配置）
public boolean addAppWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“addAppToWhiteList(String packageName, int zId)”接口
     * 参数：packageName:表示要配置为白名单的应用的包名，如“com.ts.app”
     *      zId:默认为 0
     */
    boolean isSucess = mSettingsManager.addAppToWhiteList("com.ts.app", 0);
    /*
     * 返回值为true时，表示配置应用启动白名单成功
     * 返回值为false时，表示配置应用启动白名单失败
     */
    return isSucess;
}
```

应用启动白名单（获取）

```
//应用启动白名单（获取）
public String[] obtainAppWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“getAppArrayOnWhiteList(int zId)”接口获取已配置的应用启动白名单
     * 参数：zId:默认为 0
     */
    String[] whiteListArray = mSettingsManager.getAppArrayOnWhiteList(0);
    /*
     * witeListArray数组存储的是配置过的所有应用启动白名单
     */
    return whiteListArray;
}
```

应用启动白名单（删除）

```
//应用启动白名单（删除）
public boolean deleteAppWhiteList() {
    /*
```

```
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“removeAppFromWhiteList(String packageName, int zId)”接口
     * 参数：packageName:要删除的应用启动白名单的包名，如“com.ts.app”
     *      zId:默认为 0
     */
    boolean isSucess = mSettingsManager.removeAppFromWhiteList("com.ts.app", 0);
    /*
     * 返回值为true时，表示删除指定包名的应用启动白名单成功
     * 返回值为false时，表示删除指定包名的应用启动白名单失败
     */
    return isSucess;
}
```

网络黑名单（配置）

```
//网络黑名单
public boolean addNetBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“addNetworkRuleToBlackList(String rule, int zid)”接口
     * 参数：rule：要加入网络黑名单的域名，如配置“baidu.com”，
     * 则该设备上的所有应用都不能访问“baidu.com”这个域名
     *      zid:默认为 0
     */
    boolean isSucess = mSettingsManager.addNetworkRuleToBlackList("baidu.com", 0);
    /*
     * 返回值为true时，则配置网络黑名单成功
     * 返回值为false时，则配置网络黑名单失败
     */
    return isSucess;
}
```

网络黑名单（获取）

```
//网络黑名单（获取）
public String[] obtainNetBlackList() {
    /*
```

```
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“getNetworkBlackListFromFiles(int zId)”接口
     * 参数：zId:默认为 0
     */
    String[] netBlackList = mSettingsManager.getNetworkBlackListFromFiles(0);
    /*
     * 返回的数组netBlackList存储了当前设备配置的网络黑名单列表
     */
    return netBlackList;
}
```

网络黑名单（删除）

```
//网络黑名单（删除）
public boolean deleteNetBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“removeNetworkRuleFromBlackList(String rule, int zid)”接口
     * 参数：rule：要从网络黑名单配置中删除的域名，如“baidu.com”，
     * 则该设备上的应用可以再次访问“baidu.com”这个域名
     *      zid:默认为 0
     */
    boolean isDeleteSucess = mSettingsManager.removeNetworkRuleFromBlackList("baidu.com", 0);
    /*
     * 返回值为true时，表示删除网络黑名单中指定的域名成功
     * 返回值为false时，表示删除网络黑名单中指定的域名失败
     */
    return isDeleteSucess;
}
```

网络白名单（配置）

```
//网络白名单（配置）
public boolean addNetWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
```

```
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“addNetworkRuleToWhiteList(String rule, int zid)”接口
     * 参数：rule：要加入网络白名单的域名，如配置“baidu.com”，
     * 则该设备上的所有应用都只能访问“baidu.com”这个域名（可以配置多个）
     *      zid:默认为 0
     */
    boolean isAddSucess = mSettingsManager.addNetworkRuleToWhiteList("baidu.com", 0);
    /*
     * 返回值为true时，则配置网络白名单成功
     * 返回值为false时，则配置网络白名单失败
     */
    return isAddSucess;
}
```

网络白名单（获取）

```
//网络白名单（获取）
public String[] obtainNetWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“getNetworkWhiteListFromFiles(int zId)”接口
     * 参数：zId:默认为 0
     */
    String[] netWhiteList = mSettingsManager.getNetworkWhiteListFromFiles(0);
    /*
     * 数组netWhiteList存储的是所有已配置的网络白名单
     */
    return netWhiteList;
}
```

网络白名单（删除）

```
//网络白名单（删除）
public boolean deleteNetWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
```

```
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“removeNetworkRuleFromWhiteList(String rule, int zid)”接口
     * 参数：rule：IP地址，如“baidu.com”
     *      zId:默认为 0
     */
    boolean isDeleteSucess = mSettingsManager.removeNetworkRuleFromWhiteList("baidu.com", 0);
    /*
     * 返回值为true表示删除指定ip地址的网络白名单成功，为false表示删除失败
     */
    return isDeleteSucess;
}
```

**WLAN 配置**

```
//WLAN 配置（配置）〔前提是当前设备已经打开 WIFI〕
public boolean addWlan() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用“addWifiConfig(String wifiJsonStr)”接口
     * 参数：wifiJsonStr:包存着wifi配置的字符串
     * 下面举例说明参数的格式：
     */
    JSONObject json = new JSONObject();
    try {
        json.put("ssid", "AUTO-CONFIG-WIFI");
        json.put("hiddenssid", false);
        json.put("securitytype", "security_psk");
        json.put("password", "1234567890");
        json.put("unspecifiedcert", "unspecifiedcert");
        json.put("eapmethod", "");
        json.put("phase2method", "");
        json.put("eapcaert", "");
        json.put("eapusercert", "");
        json.put("eapidentity", "");
        json.put("eapanonymous", "");
        json.put("ipsetting", "ipstatic");
        json.put("ipaddress", "192.168.16.16");
        json.put("prefixlength", 24);
        json.put("gateway", "192.168.16.1");
        json.put("nds1", "192.168.32.32");
```

```
            json.put("nds2", "202.106.0.20");
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        String wifiConfigJsonStr = json.toString();
        /*
         * 将此处wifiConfigJsonStr作为参数传入接口“addWifiConfig(String wifiJsonStr)”
         */
        boolean isAddSucess = mSettingsManager.addWifiConfig(wifiConfigJsonStr);
        /*
         * 返回值为true表示配置wlan成功，为false表示配置失败（此时请确认是否打开设备WIFI）
         * 设置完之后,可以在Settings > WLAN 中看到这个配置，包括DNS指定。
         */
        return isAddSucess;
    }
```

应用安装黑名单（配置）

```
//应用安装黑名单（配置）
public boolean appInstallBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“addAppInstallToBlackList(String packageName, int zId)”
     * 参数：packageName:要加入应用安装黑名单的应用的包名，可以配置多个
     * 如：“com.ts.app”配置后则此包名对应的app无法在该设备上安装
     *     ：zId:默认为 0
     */
    boolean isAddSucess = mSettingsManager.addAppInstallToBlackList("com.ts.app", 0);
    /*
     * 返回值为true，表示添加指定包名的应用至应用安装黑名单成功，为false，则配置失败
     */
    return isAddSucess;
}
```

应用安装黑名单（获取）

```
//应用安装黑名单（获取）
public String[] obtainAppInstallBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
```

```
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“getAppInstallArrayOnBlackList(int zId)”
     * 参数：zId：默认为 0
     */
    String[] appInstallBlackList = mSettingsManager.getAppInstallArrayOnBlackList(0);
    /*
     * 数组appInstallBlackList中保存的是在该设备上已经配置过的应用安装黑名单列表
     */
    return appInstallBlackList;
}
```

应用安装黑名单（删除）

```
//应用安装黑名单（删除）
public boolean deleteAppInstallBlackList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“removeAppInstallFromBlackList(String packageName, int zId)”
     * 参数：pakcageName：要从黑名单列表删除的应用的包名，如“com.ts.app”
     *      zId:默认为 0
     */
    boolean isDeleteSucess = mSettingsManager.removeAppInstallFromBlackList("com.ts.app", 0);
    /*
     * 返回值为true，表示指定包名的应用安装黑名单删除成功，为false，表示删除失败
     */
    return isDeleteSucess;
}
```

应用安装白名单（配置）

```
//应用安装白名单（配置）
public boolean addAppInstallWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
```

```
     * 然后调用接口“addAppInstallToWhiteList(String packageName, int zId)”
     * 参数：packageName:要加入应用安装白名单的应用的包名，可以配置多个
     * 如：“com.ts.app”配置后则只有此包名对应的app可以在该设备上安装
     *     zId:默认为 0
     */
    boolean isAddSucess = mSettingsManager.addAppInstallToWhiteList("com.ts.app", 0);
    /*
     * 返回值为true，表示配置指定包名的应用至应用安装白名单成功，为false，则配置失败
     */
    return isAddSucess;
}
```

应用安装白名单（获取）

```
//应用安装白名单（获取）
public String[] obtainAppInstallWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“getAppInstallArrayOnWhiteList(int zId)”
     * 参数：zId：默认为 0
     */
    String[] appInstallWhiteList = mSettingsManager.getAppInstallArrayOnWhiteList(0);
    /*
     * 数组appInstallWhiteList中保存的是在该设备上已经配置过的应用安装白名单列表
     */
    return appInstallWhiteList;
}
```

应用安装白名单（删除）

```
//应用安装白名单（删除）
public boolean deleteAppInstallWhiteList() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“removeAppInstallFromWhiteList(String packageName, int zId)”
     * 参数：packageName：要从应用安装白名单列表删除的应用的包名，如“com.ts.app”
     *     zId：默认为 0
     */
```

```
    boolean isDeleteSucess = mSettingsManager.removeAppInstallFromWhiteList("com.ts.app", 0);
    /*
     * 返回值为true，表示指定包名的应用安装白名单删除成功，为false，表示删除失败
     */
    return isDeleteSucess;
}
```

移动数据（启用〔true〕/禁用〔false〕）

```
//移动数（启用〔true〕/禁用〔false〕）
public boolean setMobileData() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setMobileDataEnabled(boolean enabled)”
     * 参数：enabled：为true时表示启用当前设备移动数据，为false时表示禁用当前设备移动数据
     */
    boolean isSetSucess = mSettingsManager.setMobileDataEnabled(true);//启用移动数据
    //boolean isSetSucess = mSettingsManager.setMobileDataEnabled(false);//禁用移动数据
    /*
     * 返回true表示设置启用或禁用当前设备移动数据成功，返回false表示设置失败
     */
    return isSetSucess;
}
```

强制关机

```
//强制关机
public boolean shotDown() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“forceShutdown()”来实现强制关机
     */
    boolean shotDownSucess = mSettingsManager.forceShutdown();
    /*
     * 返回值为true时，表示强制关机成功，为false时，表示强制关机失败
     */
    return shotDownSucess;
}
```

## 模块 3 策略管控/安全加强开关（Safe Enhancement Switch）

**简介**

安全加强开关类接口主要用于对设备各模块功能的开启和关闭，可以实现对设备各模块功能的自定义定制，当然，也可以从远端调用这些接口来加强设备安全性与可控性。

**接口使用**

设置 OTG 策略（启用/禁用/默认）

```
//设置 OTG 策略（启用/禁用/默认）
public boolean setOTGPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setOtgPolicy(int policy)”来设置OTG策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setOTGPolicySucess = mSettingsManager.setOtgPolicy(10);//0 和 9 用户都启用OTG
    //boolean setOTGPolicySucess = mSettingsManager.setOtgPolicy(11);//9 用户启用 0 用户禁用OTG
    //boolean setOTGPolicySucess = mSettingsManager.setOtgPolicy(14);//0 用户启用 9 用户禁用OTG
    //boolean setOTGPolicySucess = mSettingsManager.setOtgPolicy(15);//0 和 9 用户都禁用OTG
    //boolean setOTGPolicySucess = mSettingsManager.setOtgPolicy(0);//0 为恢复默认
     /*
     * 返回值为true表示设置OTG策略成功，为false表示设置OTG策略失败
     */
    return setOTGPolicySucess;
}
```

设置移动数据策略（启用/禁用/默认）

```
//设置移动数据策略（启用/禁用/默认）
public boolean setMobileDataPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
```

```
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setMobileDataPolicy(int policy)”来设置移动数据策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setMobileDataPolicy(10);//0 和 9 用户都启用
    //boolean setPolicySucess = mSettingsManager.setMobileDataPolicy(11);//9 用户启用 0 用户禁用
    //boolean setPolicySucess = mSettingsManager.setMobileDataPolicy(14);//0 用户启用 9 用户禁用
    //boolean setPolicySucess = mSettingsManager.setMobileDataPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setMobileDataPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置移动数据策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置摄像头策略（启用/禁用/默认）

```
//设置摄像头策略（启用/禁用/默认）
public boolean setCameraPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setCameraPolicy(int policy)”来设置摄像头策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setCameraPolicy(10);//0 和 9 都启用摄像头
    //boolean setPolicySucess = mSettingsManager.setCameraPolicy(11);//9 用户启用 0 用户禁用
//boolean setPolicySucess = mSettingsManager.setCameraPolicy(14);//0 用户启用 9 用户禁用
    //boolean setPolicySucess = mSettingsManager.setCameraPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setCameraPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置摄像头策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置 USB 策略（启用/禁用/默认）

```
//设置 USB 策略（启用/禁用/默认）
public boolean setUSBPolicy() {
    /*
```

```
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setUsbPolicy(int policy)”来设置USB策略
     * 参数：policy：10为0和9用户都启用，11为9用户启用0用户禁用，14为0用户启用9用户禁用
       ,15为0和9用户都禁用,0为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setUsbPolicy(10);//0和9都启用USB
    //boolean setPolicySucess = mSettingsManager.setUsbPolicy(11);//9用户启用0用户禁用USB
    //boolean setPolicySucess = mSettingsManager.setUsbPolicy(14);//0用户启用9用户禁用USB
 //boolean setPolicySucess = mSettingsManager.setUsbPolicy(15);//0和9用户都禁用USB
    //boolean setPolicySucess = mSettingsManager.setUsbPolicy(0);//0为恢复默认
    /*
     * 返回值为true表示设置USB策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置麦克风策略（启用/禁用/默认）

```
//设置麦克风策略（启用/禁用/默认）
public boolean setVoicePolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setVoiceRecordPolicy(int policy)”来设置麦克风策略
     * 参数：policy：10为0和9用户都启用，11为9用户启用0用户禁用，14为0用户启用9用户禁用
       ,15为0和9用户都禁用,0为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setVoiceRecordPolicy(10);//0和9都启用麦克风
   //boolean setPolicySucess = mSettingsManager.setVoiceRecordPolicy(11);//9用户启用0用户禁用
   //boolean setPolicySucess = mSettingsManager.setVoiceRecordPolicy(14);//0用户启用9用户禁用
  //boolean setPolicySucess = mSettingsManager.setVoiceRecordPolicy(15);//0和9用户都禁用
  //boolean setPolicySucess = mSettingsManager.setVoiceRecordPolicy(0);//0为恢复默认
    /*
     * 返回值为true表示设置麦克风策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置剪贴板策略（启用/禁用/默认）

```
//设置剪贴板策略（启用/禁用/默认）
public boolean setClipBoardPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setClipboardPolicy(int policy)”来设置剪贴板策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setClipboardPolicy(10);//0 和 9 都启用剪贴板
    //boolean setPolicySucess = mSettingsManager.setClipboardPolicy(11);//9 用户启用 0 用户禁用
剪贴板
    //boolean setPolicySucess = mSettingsManager.setClipboardPolicy(14);//0 用户启用 9 用户禁用
    //boolean setPolicySucess = mSettingsManager.setClipboardPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setClipboardPolicy(0);//0 为恢复默认
/*
     * 返回值为true表示设置剪贴板策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置蓝牙策略（启用/禁用/默认）

```
//设置蓝牙策略（启用/禁用/默认）
public boolean setBlueToothPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setBluetoothPolicy(int policy)”来设置蓝牙策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setBluetoothPolicy(10);//0 和 9 都启用蓝牙
    //boolean setPolicySucess = mSettingsManager.setBluetoothPolicy(11);//9 用户启用 0 用户禁用
蓝牙
    //boolean setPolicySucess = mSettingsManager.setBluetoothPolicy(14);//0 用户启用 9 用户禁用
//boolean setPolicySucess = mSettingsManager.setBluetoothPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setBluetoothPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置蓝牙策略成功，为false表示设置策略失败
```

```
     */
    return setPolicySucess;
}
```

设置 NFC 策略（启用/禁用/默认）

```
//设置 NFC 策略（启用/禁用/默认）
public boolean setNFCPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setNfcPolicy(int policy)”来设置NFC策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认

     */
    boolean setPolicySucess = mSettingsManager.setNfcPolicy(10);//0 和 9 都启用nfc
    //boolean setPolicySucess = mSettingsManager.setNfcPolicy(11);//9 用户启用 0 用户禁用nfc
    //boolean setPolicySucess = mSettingsManager.setNfcPolicy(14);//0 用户启用 9 用户禁用
    //boolean setPolicySucess = mSettingsManager.setNfcPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setNfcPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置nfc策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置 WIFI 策略（启用/禁用/默认）

```
//设置 wifi 策略（启用/禁用/默认）
public boolean setWifiPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setWifiPolicy(int policy)”来设置WIFI策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认

     */
    boolean setPolicySucess = mSettingsManager.setWifiPolicy(10);//0 和 9 都启用wifi
    //boolean setPolicySucess = mSettingsManager.setWifiPolicy(11);//9 用户启用 0 用户禁用wifi
```

```
    //boolean setPolicySucess = mSettingsManager.setWifiPolicy(14);//0 用户启用 9 用户禁用
    //boolean setPolicySucess = mSettingsManager.setWifiPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setWifiPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置wifi策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置截屏策略（启用/禁用/默认）

```
//设置截屏策略（启用/禁用/默认）
public boolean setScreenshotPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setScreenshotPolicy(int policy)”来设置截屏策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setScreenshotPolicy(10);//0 和 9 都启用截屏
    //boolean setPolicySucess = mSettingsManager.setScreenshotPolicy(11);//9 用户启用 0 用户禁用
截屏
    //boolean setPolicySucess = mSettingsManager.setScreenshotPolicy(14);//0 用户启用 9 用户禁用
       //boolean setPolicySucess = mSettingsManager.setScreenshotPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setScreenshotPolicy(0);//0 为恢复默认
     /*
     * 返回值为true表示设置截屏策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置共享策略（启用/禁用/默认）

```
//设置共享策略（启用/禁用/默认）
public boolean setSharePolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setForbiddenSharedPolicy(int policy)”来设置共享策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
```

```
      ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setForbiddenSharedPolicy(10);//0 和 9 都启用截屏分享
    //boolean setPolicySucess = mSettingsManager.setForbiddenSharedPolicy(11);//9 用户启用 0 用户禁用
    //boolean setPolicySucess = mSettingsManager.setForbiddenSharedPolicy(14);//0 用户启用 9 用户禁用
    //boolean setPolicySucess = mSettingsManager.setForbiddenSharedPolicy(15);//0 和 9 用户都禁用
    //boolean setPolicySucess = mSettingsManager.setForbiddenSharedPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置分享策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置 GPS 策略（启用/禁用/默认）

```
//设置 GPS 策略（启用/禁用/默认）
public boolean setGPSPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setGpsPolicy(int policy)”来设置GPS策略
     * 参数：policy：10 为 0 和 9 用户都启用，11 为 9 用户启用 0 用户禁用，14 为 0 用户启用 9 用户禁用
       ,15 为 0 和 9 用户都禁用,0 为恢复默认
     */
    boolean setPolicySucess = mSettingsManager.setGpsPolicy(10);//0 和 9 都启用GPS
    //boolean setPolicySucess = mSettingsManager.setGpsPolicy(11);//9 用户启用 0 用户禁用
    //boolean setPolicySucess = mSettingsManager.setGpsPolicy(14);//0 用户启用 9 用户禁用
   //boolean setPolicySucess = mSettingsManager.setGpsPolicy(15);//0 和 9 用户都禁用
  //boolean setPolicySucess = mSettingsManager.setGpsPolicy(0);//0 为恢复默认
    /*
     * 返回值为true表示设置GPS策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置 APP 安装管控名单（黑名单/白名单/不使用）

```
//设置 App 安装管控名单（黑名单/白名单/不使用）
public boolean setAppInstallPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
```

```
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setAppInstallListPolicy(int policy, int zId)”来设置要是用的名单列表
     * 参数：policy：2 为使用黑名单，1 为使用白名单，0 为不使用配置的App安装管控名单
     *      zId：默认为 0
     */
    boolean setPolicySucess = mSettingsManager.setAppInstallListPolicy(2, 0);//使用App安装管控黑名单
    //boolean setPolicySucess = mSettingsManager.setAppInstallListPolicy(1, 0);//使用App安装管控白名单
    //boolean setPolicySucess = mSettingsManager.setAppInstallListPolicy(0, 0);//不使用App安装管控
    /*
     * 返回值为true表示设置策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置 App 启动管控名单（黑名单/白名单/不使用）

```
//设置 App 启动管控名单（黑名单/白名单/不使用）
public boolean setAppLauncherPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setAppLaunchListPolicy(int policy, int zId)”来设置要使用的App启动管控名单列表
     * 参数：policy：2 为使用黑名单，1 为使用白名单，0 为不使用配置的App启动管控名单
     *      zId：默认为 0
     */
    boolean setPolicySucess = mSettingsManager.setAppLaunchListPolicy(2, 0);//使用App启动管控黑名单
    //boolean setPolicySucess = mSettingsManager.setAppLaunchListPolicy(2, 0);//使用App启动管控白名单
    //boolean setPolicySucess = mSettingsManager.setAppLaunchListPolicy(2, 0);//不使用App启动管控
    /*
     * 返回值为true表示设置策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置网络管控名单（黑名单/白名单/不使用）

```
//设置网络管控名单（黑名单/白名单/不使用）
public boolean setNetListPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setNetworkPolicy(int zId, int policy)”来设置要使用的App启动管控名单列表
     * 参数：policy：2 为使用黑名单，1 为使用白名单，0 为不使用配置的网络管控名单
     *      zId：默认为 0
     */
    boolean setPolicySucess = mSettingsManager.setNetworkPolicy(0, 2);//使用网络管控黑名单
    //boolean setPolicySucess = mSettingsManager.setNetworkPolicy(0, 2);//使用网络管控黑名单
    //boolean setPolicySucess = mSettingsManager.setNetworkPolicy(0, 2);//使用网络管控黑名单
    /*
     * 返回值为true表示设置策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```

设置 SD 卡策略（启用/禁用〔前提条件，当前设备支持 SD 卡〕）

```
//SD 卡管控[前提条件，当前设备支持 SD 卡]
public boolean SDCardPolicy() {
    /*
     * 首先，要获得SecuritySettingsManager对象；
     * 这个对象在Android中是一个系统服务，服务的字符串是：
     * "security_settings_service"
     */
    SecuritySettingsManager mSettingsManager = (SecuritySettingsManager)
            getSystemService("security_settings_service");
    /*
     * 然后调用接口“setSDCardPolicy(int policy)”来设置SD卡策略
     * 参数：policy：10 为启用SD卡，15 为禁用SD卡
     */
    boolean setPolicySucess = mSettingsManager.setSDCardPolicy(10);//启用当前设备SD卡
    //boolean setPolicySucess = mSettingsManager.setSDCardPolicy(15);//禁用当前设备SD卡
    /*
     * 返回值为true表示设置策略成功，为false表示设置策略失败
     */
    return setPolicySucess;
}
```