卓策板卡开发说明文档

注意：zckjAPI.jar 包 2020 年 12 月以前的固件仅支持部分接口，2020 年 12 月以后的固件才会支持比较全面。

日期	版本	说明
2020-03-30	1	第一版
2020-03-31	2	优化部分方法，新增内容
2020-05-09	3	新增串口说明及附录，调整部分内容

2020-08-07	4	修改部分错误
2020-12-08	5	整合卓策 API、增添修改 GPIO、串口列表
2021-07-09	6	添加了设置以太网静态 IP 的接口、添加了 ZC-3568，ZC-972 型号的 GPIO 口控制

说明：若有前置‘*’号的，代表该操作需要系统级权限，需在 manifest 中加入 android:sharedUserId="android.uid.system”至<manifest />结点中，并为应用使用系统签名。

若有前置‘#’号的，代表该方法需要调用 zckjAPI.jar，必须先执行下文中“获取 zckjAPI 实例”

目录

为应用使用系统签名文件签名

应用签名

当您需要系统权限时，除了在 manifest 中应用 **android:sharedUserId="android.uid.system"** 以外，还需要为编译好的应用使用我们提供对应的板卡签名文件签名。

签名需要系统中具有 JAVA 运行环境，并有如下两种方法可以实现：

1. 使用 signapk.jar 工具，于命令行下输入如下：
$ java -jar signapk.jar platform.x509.pem platform.pk8 app.apk app_signed.apk
2. 使用 apksigner 工具，于命令行下输入如下：
$ apksigner sign --key platform.pk8 --cert platform.x509.pem --out app_signed.apk app.apk

命令执行完毕后，app_signed.apk 会被生成，即是已签名的应用。

提示：signapk.jar 与签名文件见附带目录**【签名文件及方法】**。

提示：有时候在 Linux 个别发行版或 MacOS 下 signapk.jar 会出现找不到库的问题，此时可以使用包管理器安装 apksigner，正常情况下会将依赖自动配置好，该工具在 SDK build tools

中也可找到。
提示：若签名失败，可尝试两种方法
1：替换安卓 6.0 系统的 signapk.jar。
2：platform.pk8 platform.x509.pem 的位置尝试互换。

验证签名

要验证系统签名只需要将声明过系统权限(android.uid.system)的应用安装至系统中，若**成功安装则为签名成功**。

若声明过系统权限而没有对应的系统签名安装会**失败**。

添加 zckjAPI-2.0.jar 包到工程

下列带#号的是本次整合到 zckjAPI-2.0.jar 里的方法，需要调用里面的接口，必须先添加 zckjAPI-2.0.jar 包到工程里，记得 sync project with gradle files。
方法：将 Androidstudio 中的项目切换为 project，再把 zckj.jar 包复制到 app/libs 文件夹下即可。
更详细可参考 https://blog.csdn.net/yushuangping/article/details/81873630

获取 zckjAPI 实例（导入包）

想要调用 zckjAPI 里的接口，必需先导入 com.zckj.zcapi 包，获取实例并设置 context。以下带“#”接口都需要先获取实例后才能操作。
关于 zckjAPI-2.0.jar 里的操作，在【卓策 api DEMO】里都有演示。
注意：该 zckjAPI-2.0.jar 不支持 2020 年 12 月以前的固件，需要使用该功能， 2020 年 12 月以后的固件才会更新到该功能，具体要咨询下售后。

```
import com.zckj.zcapi;
zcapi zcApi=new zcapi();
zcApi.getContext(getApplicationContext());
```

系统控制

#关机

调用接口 public void shutDown()
zcApi.shutDown();

#重启

调用接口 public void reboot()
zcApi.reboot();

#恢复出厂设置

注意！！！恢复出厂设置时不要插着 U 盘，否则可能格式化 U 盘！！！！

调用接口 public void factoryReset()
zcApi.factoryReset();

#OTA 升级

该方法需有对应的 OTA 包，目前 328 及 339 于 Android 7.1 才可使用，可由应用自行分发 OTA 包至设备后启动更新过程。

调用接口 public void updateOta() 注意：328 的 OTA 包必须置于/sdcard/下，且文件名限定为“zc328_update.zip” 注意：339 的 OTA 包必须置于/sdcard/下，且文件名限定为“zc339_update.zip” 注意：不论是否升级，系统重启后，OTA 包均会被删除
zcApi.updateOta();

提示：在欲写/sdcard/前，可能需要先动态申请 permission.READ_EXTERNAL_STORAGE 权限。

#定时开关机

本次更新中，定时开关机已经整合到 zckjAPI.jar 中，调用接口即可。

调用接口 public void setPowetOnOffTime(enable,onTime,offTime)
 	
//参数：enable，取消或使能定时开关机，如果设置为 false，那么已设定的定时开机与//关机时间都会清除掉。
//onTime、offTime:开机与关机时间 int 数组，包含 年 月 日 时 分，数据格式与顺序//必须正确，并且设定的开机与关机时间必须超过当前时间，否则，会清除掉已设定的开//机时间或关机时间，导致不能达到预想的要求。

```
int []onTime={year,month,day,hour,minute};
int []offTime={year,month,day,hour,minute};

zcApi.setPowetOnOffTime(true,onTime,offTime);
```

#设置系统时间

调用接口 public void setSystemTime(int[] time)
//int[]time：int 型时间数组，包括 年 月 日 时 分，顺序格式不可错。

```
zcApi.setSystemTime();
```

系统信息

#获取板卡型号

调用接口 public String getBuildModel()

```
zcApi.getBuildModel();
```

#获取板卡 SN 号

调用接口 public String getBuildSerial()

zcApi.getBuildSerial();

#获取板卡以太网 MAC 地址

调用接口 public String getEthMacAddress()
zcApi.getEthMacAddress();

#获取板卡 WLAN MAC 地址（前提是打开 WIFI）

调用接口 public String getWifiMacAddress()
zcApi.getWifiMacAddress();

获取板卡以太网 IP 地址

1.利用 NetworkInterface 读取，需声明权限： android.permission.INTERNET

NetworkInterface ethInterface = NetworkInterface.getByName("eth0");
for(InterfaceAddress address: ethInterface.getInterfaceAddresses()){
if(address.getAddress() instanceof Inet4Address){
sb.append(address.getAddress().getHostAddress() + ", ");
}
}

获取板卡 WLAN IP 地址

1.利用 NetworkInterface 读取，需声明权限： android.permission.INTERNET
NetworkInterface wlanInterface = NetworkInterface.getByName("wlan0");

```
                for(InterfaceAddress address: wlanInterface.getInterfaceAddresses()){
                    if( address.getAddress() instanceof Inet4Address ){
                        sb.append(address.getAddress().getHostAddress() + ", ");
                    }
                }
```

2.利用 WifiManager 读取，需声明权限：android.permission.ACCESS_WIFI_STATE

```
        WifiManager wifiMan = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        WifiInfo wifiInfo = wifiMan.getConnectionInfo();
        String res = TransIPToString(wifiInfo.getIpAddress()); //该方法返回的是 int 值，
需自行处理
```

背光控制

#开启、关闭背光

注意，背光控制只影响 BLON 脚是否使能，完全不影响系统状态，若您使用 HDMI 则有可能无法观察到效果。

调用接口 public void setLcdOnOff(boolean enable)

```
//enable 设置背光开启或关闭
//true:打开背光
//false:关闭背光
```

```
        zcApi.setLcdOnOff(enable);
```

#开启、关闭 HDMI

本功能目前仅支持 328、339 系列产品

调用接口 public void setLcdOnOff(boolean enable，int lcdOrHdmi)

```
//enable: 打开或关闭 hdmi，true:打开，false:关闭
//lcdOrHdmi:为 0 时只操控 lcd 的背光，为 1 时同时操控 HDMI 跟 lcd 背光，为-1 时仅
操控 HDMI。
```

```
        zcApi.setLcdOnOff(true,-1);
        //此例为打开 HDMI。
```


控制系统亮度

该方法通过标准方法写系统设置实现，且在 Android 6.0 以上需申请权限。

需声明权限： **<uses-permission android:name="android.permission.WRITE_SETTINGS"** **tools:ignore="ProtectedPermissions"/>**
Android6.0+ 需先申请权限： **Uri packageName = Uri.parse("package:"+getPackageName());** **Intent intent = new** **Intent(Settings.ACTION_MANAGE_WRITE_SETTINGS, packageName);** **startActivity(intent);**
Uri uri = Settings.System.getUriFor(Settings.System.SCREEN_BRIGHTNESS); Settings.System.putInt(getContentResolver(), Settings.System.SCREEN_BRIGHTNESS, level); getContentResolver().notifyChange(uri, null); //通知系统亮度发生了变化

状态栏控制

#隐藏导航栏、状态栏

调用接口 public void setStatusBar(boolean enable)
//enable 设置导航栏、状态栏显示或隐藏
//true:显示
//false:隐藏

zcApi.setStatusBar(enable);

#允许、禁止滑动呼出状态栏

调用接口 public void setGestureStatusBar(boolean enable) //enable 允许或禁止滑动呼出状态栏

//true:允许 //false:禁止
zcApi.setGestureStatusBar(enable);

音量控制

音量增加

通过安卓标准方法即可，也可以使用 setStreamVolumn 方法，但要先确认系统的 MaximumVolumn。

STREAM_MUSIC (媒体) STREAM_ALARM (闹钟) STREAM_NOTIFICATION (通知)
mAudioManager = (AudioManager) getSystemService(AUDIO_SERVICE); mAudioManager.adjustStreamVolume(AudioManager.STREAM_MUSIC, AudioManager.ADJUST_RAISE,0);

音量减少

通过安卓标准方法即可，也可以使用 setStreamVolumn 方法，但要先确认系统的 MaximumVolumn。

STREAM_MUSIC (媒体) STREAM_ALARM (闹钟) STREAM_NOTIFICATION (通知)
mAudioManager = (AudioManager) getSystemService(AUDIO_SERVICE); mAudioManager.adjustStreamVolume(AudioManager.STREAM_MUSIC, AudioManager.ADJUST_LOWER,0);

静音

通过安卓标准方法即可，也可以使用 setStreamVolumn 方法。

STREAM_MUSIC (媒体) STREAM_ALARM (闹钟) STREAM_NOTIFICATION (通知) **该方法需 SDK Level ≥ 23，即 Android 6.0 及以后，在较低版本可以直接 setStreamVolumn 到 0 即可，但需要自己保存最后音量状态**
mAudioManager = (AudioManager) getSystemService(AUDIO_SERVICE);

mAudioManager.adjustStreamVolume(AudioManager.STREAM_MUSIC, AudioManager.ADJUST_TOGGLE_MUTE,0);

GPIO 控制

GPIO 控制已被封装到 zckjAPI.jar 中，直接调用即可，具体查看 ZCKJAPI.apk 的 demo，主要在 gpioAdapter.java 部分。

#设置 IO 引脚模式

调用接口 public void setMulSelGpio(char group, int num, int value)
//group 为组名 num 为引脚号 value 为模式
//value 为 1 为输出，为 0 为输入
//如 PB18 则 group 为'B', num 为 18

zcApi.setMulSelGpio('B', 18, 1); //PB18 设置为输出状态

#读取 IO 引脚状态

读取前必须先设置引脚状态为输入

调用接口 public int readGpio(char group, int num)
//group 为组名 num 为引脚号
//如 PB18 则 group 为'B', num 为 18
//函数返回 int 值，1 代表高电平，0 代表低电平。

zcApi.readGpio('B', 18); //读取 PB18 引脚状态

#写入 IO 引脚状态

写入前必须先设置引脚状态为输出

调用接口 public void writeGpio(char group, int num, int value)
//group 为组名 num 为引脚号 value 为值
//GPIO 值只有 1 或 0 两种状态
//如 PB18 则 group 为'B', num 为 18

zcApi.writeGpio('B', 18, 1); //设置 PB18 引脚高电平

串口通信（仅供参考）

使用前需要将封装好的 SerialPort 类 SerialPort.java 放入包 com.wits.serialport 下。并引入库文件 libserial_port.so 至应用项目中(libs/armeabi)。

串口在初始化后以流方式读写，不再使用时请关闭对应串口设备。

串口工作于 N81 模式。

提示：SerialPort.java 与 libserial_port.so 可从**【定时开关机操作 DEMO2.0】**项目中取得，鉴于定时开关机是利用串口与 MCU 通信实现，因此串口的具体用法也可参照该项目。

注意：**【定时开关机操作 DEMO2.0】**只是一个演示，需要定时开关机功能建议使用 zckjAPI.jar 里包装好的方法，上例有演示，也可以采用此 DEMO 来做定时开关机的方法

提示：系统串口号对应板卡插座见“**附录 B・串口号与板卡插座速查表**”

初始化串口设备

public **SerialPort**(File device, int baudrate, int flags) throws SecurityException,IOException
//构造函数，注意处理异常
//device 文件：描述串口设备号，如 File(“/dev/ttyS1”)
//baudrate 波特率：不超过 115200 的整数
//flag 附加参数：串口均会已 RW 方式打开，因此可忽视该参数，给 0 即可。

SerialPort device = new SerialPort(new File(“/dev/ttyS1”), 9600, 0); //初始化 ttyS1 设备，波特率为 9600

获取输出流及输出

Public OutputStream **getOutputStream**(); //取得输出流后按普通方式写入即可。
OutputStream deviceOS = device.getOutputStream(); deviceOS.write(bytesArray); //串口发送 bytesArray

获取输入流及输入

public InputStream **getInputStream**(); //取得输入流后按普通方式读取即可。
InputStream deviceIS = device.getInputStream();

```
byte[] buffer = new byte[32];
int readCnt = deviceIS.read(buffer); //获取串口输入到 buffer，readCnt 为读的字节数
```

关闭串口设备

串口设备初始化后，一旦不再使用应当及时关闭释放。

```
SerialPort device = new SerialPort(new File("/dev/ttyS1"), 9600, 0);
//blah blah blah 干了很多事情
device.close(); //关闭串口设备
device = null;
```

其他

#看门狗

调用接口 public void watchDogEnable(boolean dog)

//boolean dog:true 时使能看门狗，false 关闭看门狗。

//如果设置了 true 且调用了这个接口（喂狗）后，两分钟内没有调用接口再次使能（喂//狗）则主板重启，如果设置为 false 那么看门狗关闭。

```
zcApi.watchDogEnable(watchDogEnable);
```

#执行 SU 命令

调用接口 public void execShellCmd(String cmd)

//String cmd:需要执行的指令，下例是使用超级权限在/sdcard 下创建一个文件

```
String cmd=" touch /sdcard/execShellCmd";
zcApi.execShellCmd(cmd);
```

#截屏

调用接口 public void screenshot(String path, String name)
 	
//path:截屏图片保存的路径
//name:截屏图片保存的名字

String path=”/sdcard”; String fileName=”screenshot.png”; zcApi. screenshot (path,fileName);

#静默安装应用

调用接口 public void InstallApk(String path, boolean mode)
//String path:需要安装的 apk 的路径
//boolean mode:true 是使用超级权限静默安装，需要 sdk 版本大于 17，安卓 4.2 以后.
//false 是使用普通方式安装。
//下例是使用超级权限静默安装位于/sdcard/下面的 hello.apk。

String path="/sdcard/hello.apk"; zcApi.InstallApk(path,true);

#设置以太网静态 IP

调用接口 public void SetStaticIP(String ip, String gateway, String netMask, String dns1, String dns2)
//String IP:需要设置的静态 IP
//String gateway：需要设置的网关
//String netMask：需要设置的子网掩码。
//String dns1：需要设置的 dns1。
//String dns2：需要设置的 dns2。
将参数按照严格的格式要求填入，调用接口即可完成设置。

zcApi.SetStaticIP(ipAddr,gateway,netMask,dns1,dns2);

附录 A GPIO 与板卡引脚速查表

请注意，**引脚组均为英文字符，并非数字**，部分板卡 gpio 引脚相同。
PCB 丝印号的"/ "是或者的意思。设备号的"/ "是文件路径的意思

板卡	PCB 丝印号	内部端口号	引脚组	引脚号
ZC-20A	IO1	PI11	I	11
	IO2	PI10	I	10
	IO3	PB18	B	18
	IO4	PB19	B	19
ZC-40A	IO1/PB14	PB14	B	14
	IO2/PB15	PB15	B	15
	IO3/PB16	PB16	B	16
	IO4/PB17	PB17	B	17
ZC-40M	GPIO1	PB19	B	19
	GPIO2	PB18	B	18
	GPIO3	PH0	H	0
	GPIO4	PH1	H	1
ZC-64A	PE0	PE0	E	0
	PE1	PE1	E	1
	PE2	PE2	E	2
	PE3	PE3	E	3
	PE12	PE12	E	12
	PE13	PE13	E	13
	PE14	PE14	E	14
	PE15	PE15	E	15
ZC-83A	IO3/PE14	PE14	E	14
	IO2/PE15	PE15	E	15
	IO1/PE16	PE16	E	16
ZC-83E	PE14	PE14	E	14
	PE15	PE15	E	15
	PE16	PE16	E	16
	PH4	PH4	H	4
ZC-328 ZC-328D ZC-328S	IO1	IO1	0	1
	IO2	IO2	0	2
	IO3	IO3	0	3
	IO4	IO4	0	4
ZC-328E	4	PO4	0	4
	3	PO3	0	3
	2	PO14	0	14
	1	PO15	0	15

	IO2	P02	0	2
	IO1	P01	0	1
	继电器 1	P012	0	12
	继电器 2	P013	0	13
ZC-339 ZC-339E	IO4/G4-D2	P01	0	1
	IO3/G4-D3	P02	0	2
	IO2/G4-D4	P03	0	3
	IO1/G4-D5	P04	0	4
ZC-312	IO1	P01	0	1
ZC-3568	IO1	IO1	0	1
	IO2	IO2	0	2
	IO3	IO3	0	3
	IO4	IO4	0	4
ZC-972	IO1	IO1	0	1
	IO2	IO2	0	2
	IO3	IO3	0	3
	IO4	IO4	0	4

附录 B 串口号与板卡插座速查表

▲标注的为默认被蓝牙占用的串口，默认情况下无法使用

■标注的为默认没有贴的插座，默认情况下无法使用

●标注的串口被 debug 口占用，无法使用

板卡	PCB 丝印号	设备号
ZC-20A	ttyS4/UART4	/dev/ttyS4
	ttyS7/UART7	/dev/ttyS7
ZC-40A	ttyS4/UART4	/dev/ttyS4
	ttyS5/UART5	/dev/ttyS5
	ttyS7/UART7	/dev/ttyS7
ZC-40M	ttyS4/UART4	/dev/ttyS4
	ttyS5/UART5	/dev/ttyS5
ZC-64A	● ttyS0/UART0	/dev/ttyS0
	ttyS1/UART1	/dev/ttyS1
	ttyS2/UART2	/dev/ttyS2
ZC-83A	ttyS3/UART3	/dev/ttyS3
	ttyS4/UART4	/dev/ttyS4
ZC-83E	ttyS1/UART1	/dev/ttyS1
	ttyS3/UART3	/dev/ttyS3
	ttyS4/UART4	/dev/ttyS4
ZC-328 ZC-328D	▲ ttyS0/UART0	/dev/ttyS0
	ttyS4/UART4	/dev/ttyS4
	ttyS1/UART1	/dev/ttyS1
ZC-328S	TTYZC0	/dev/ttyZC0
	TTYZC1	/dev/ttyZC1
	TTYZC2	/dev/ttyZC2
	TTYZC3	/dev/ttyZC3
	TTYS1	/dev/ttyS1
	TTYS4	/dev/ttyS4
ZC-328E	ttyS1/UART1	/dev/ttyS1
	ttyS4/UART4	/dev/ttyS4
	■ ttyZC1/UART10	/dev/ttyZC1
	■ ttyZC0/UART11	/dev/ttyZC0
	■ ttyZC3/UART12	/dev/ttyZC3
	■ ttyZC2/UART13	/dev/ttyZC2

ZC-339	ttyZC0/UART11	/dev/ttyZC0
	ttyZC2/UART12	/dev/ttyZC2
	ttyZC1/UART14	/dev/ttyZC1
	TTYZC2	/dev/ttyZC3
ZC-339E	TTYZC0	/dev/ttyZC0
	TTYZC1	/dev/ttyZC1
	TTYZC2	/dev/ttyZC2
	TTYZC3	/dev/ttyZC3
ZC-312	ttyS1	/dev/ttyS1
ZC-972	ttyS2	/dev/ttyS2
	ttyS4	/dev/ttyS4
ZC-3568	ttyS1	/dev/ttyS1
	ttyS2	/dev/ttyS2
	ttyS3	/dev/ttyS3
	ttyS7	/dev/ttyS7