

主板 API 文档

版 本	V1.7.2
日 期	2023-05-22

敬告：本文档版权归内容原创公司所有，并保留一切权力。文档内容如有修改更新，请联系提供方获取最新版本，恕不另行通知。

修改记录

1.0.0	2017-05-12	本文档第一个版本。
1.1.0	2017-06-15	增加了串口关闭接口描述。
1.2.0	2017-07-04	增加了 Android Studio 开发环境下如何导入库的说明。
1.3.0	2017-08-18	增加 HDMIIN 编程接口; 增加 APK 签名说明。
1.3.1	2017-12-26	APK 签名地址更新 http://120.78.220.29:18080/
1.3.2	2018-03-05	修正\$3.1 小节显示密度部分文字笔误。
1.4.0	2018-04-09	屏幕旋转接口增加调用权限说明。
1.5.0	2018-07-30	增加 Android Studio 3.0 集成参数说明。
1.6.0	2018-10-30	增加 setSystemTime 的权限说明。
1.6.1	2018-11-12	更新签名服务器 IP 地址。
1.6.2	2019-02-18	增加安卓 7.1 以上版本签名网址。
1.6.3	2020-09-28	增加 AS 混淆配置文件说明。
1.7.0	2021-12-15	V1.2.0.20211022 版本支持完全无需 ROOT 执行、修复安卓 7.1 以上静态 IP 接口 DNS 串格式问题; 增加定时开关机广播接口和应用启动管理广播接口说明。
1.7.1	2022-03-18	订正每天模式定时开关机示例代码文字错误
1.7.2	2023-05-22	定时开关机和应用启动管理广播接口增加安卓 8.0 以上系统调用说明!

目 录

1	编程说明	7
1.1	ECLIPSE IDE 集成	7
1.2	ANDROID STUDIO 集成	8
1.3	API 调用方法	11
1.4	APK 系统签名	11
2	硬件管理	13
2.1	无线和网络	13
2.1.1	读取以太网状态	13
2.1.2	打开/关闭以太网	13
2.1.3	设置以太网静态地址	13
2.1.4	设置以太网 DHCP 模式	14
2.1.5	读取以太网设置	14
2.1.6	读取以太网 MAC 地址	14
2.1.7	读取当前网络类型	15
2.1.8	WiFi 网络操作	15
2.1.9	读取移动网络设置	15
2.2	存储空间	15
2.2.1	读取内部存储路径	15
2.2.2	读取外部 SD 卡路径	16

2.2.3	读取外部 U 盘路径.....	16
2.2.4	读取存储器容量.....	16
2.3	运行内存.....	17
2.3.1	读取系统内存大小.....	17
2.4	背光开关.....	17
2.4.1	打开/关闭 LCD 背光.....	17
2.5	系统时间.....	17
2.5.1	设置安卓系统时钟.....	17
2.6	GPIO.....	18
2.6.1	使能 IO 口	18
2.6.2	设置 IO 方向-输入	18
2.6.3	设置 IO 方向-输出	18
2.6.4	读取 IO 高低状态.....	19
2.6.5	设置 IO 高低状态.....	19
2.7	串口.....	19
2.7.1	打开串口.....	20
2.7.2	串口读数据.....	20
2.7.3	串口写数据.....	20
2.7.4	关闭串口.....	21
2.8	硬件看门狗.....	21
2.8.1	打开看门狗.....	21
2.8.2	看门狗喂狗.....	21

2.8.3	关闭看门狗.....	21
2.9	开关机.....	22
2.9.1	硬件关机.....	22
2.9.2	软件重启.....	22
2.9.3	硬件重启.....	22
2.9.4	定时开机.....	23
2.10	HDMIIN.....	23
2.10.1	读取 HDMI 输入状态	23
2.10.2	读取 HDMI 输入分辨率	23
3	安卓管理	25
3.1	显示管理.....	25
3.1.1	读取分辨率.....	25
3.1.2	屏幕截屏.....	25
3.1.3	屏幕旋转.....	26
3.1.4	读取显示密度.....	26
3.1.5	设置显示密度.....	26
3.2	导航条.....	26
3.2.1	显示导航条.....	27
3.2.2	隐藏导航条.....	27
3.2.3	打开/关闭滑动.....	27
3.2.4	导航条自动隐藏.....	27
3.3	状态条.....	28

3.4	应用管理	28
3.4.1	读取应用列表	28
3.4.2	应用静默安装	28
3.4.3	应用自动安装	28
3.4.4	应用静默卸载	29
3.4.5	应用自动卸载	29
3.4.6	应用自动启动	29
3.5	系统信息	29
3.5.1	读取安卓版本号	29
3.5.2	读取安卓序列号	30
3.5.3	读取系统版本号	30
3.5.4	读取内核版本	30
3.5.5	读取 API 版本	30
3.5.6	读取 WiFi 硬件地址	30
3.5.7	读取以太网硬件地址	31
3.6	系统升级	31
3.6.1	系统 OTA 升级	31
3.7	SHELL 命令执行	31
3.7.1	Shell 命令执行	31
4	定时开关机广播接口	34
5	应用启动管理广播接口	36

1 编程说明

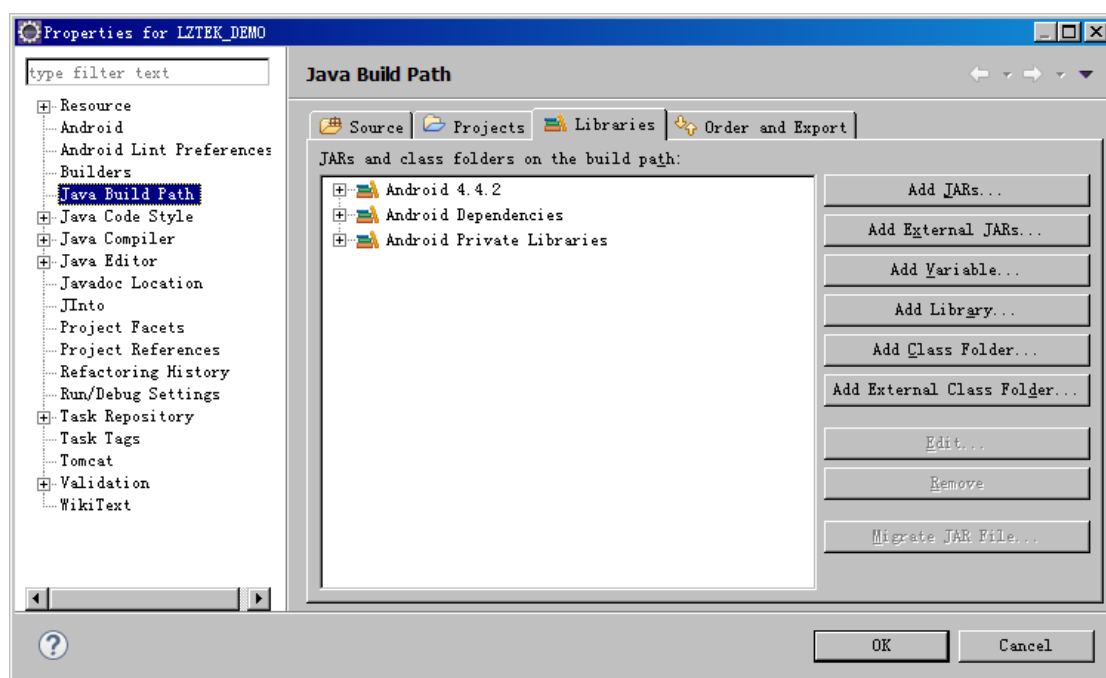
请将相应的库文件集成到工程中并按照 API 说明进行调用即可。注意：部分 API 需要用到 root 权限，请咨询主板厂商获取安卓系统 root 方法。具体哪些 API 需要 root 权限请查询相应 API 说明。

提示：本文档 API 接口必须在我司 20170518 以上安卓版本上才能使用！

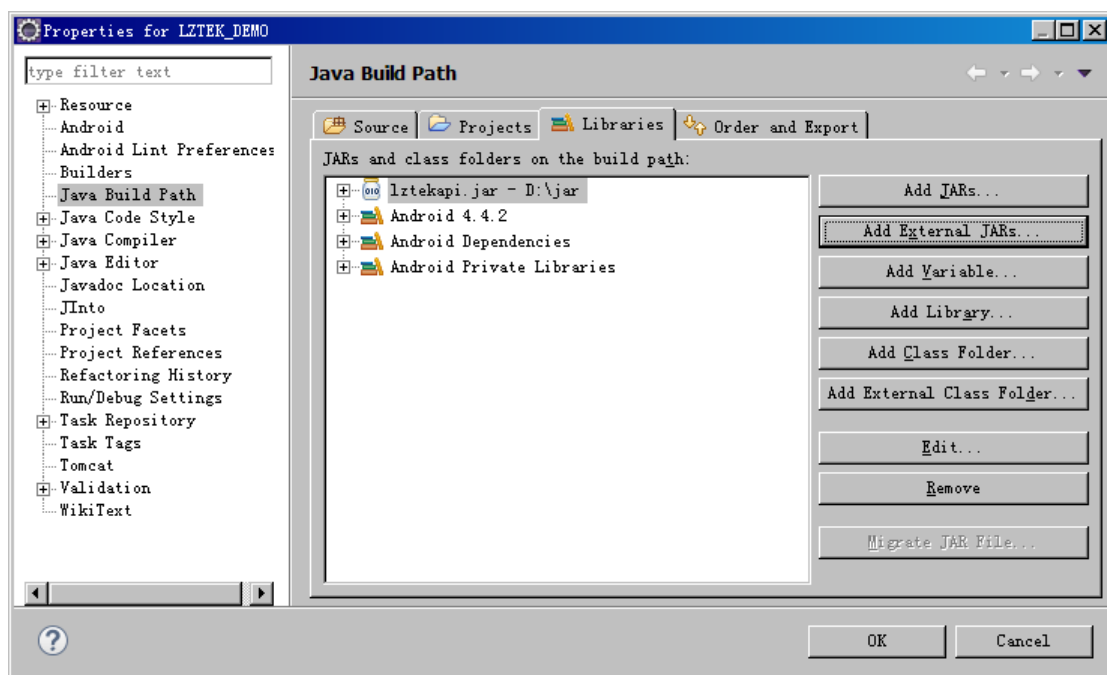
1.1 Eclipse IDE 集成

→ 如何在 Eclipse 中使用 API 的方法如下：

- 1) 打开 Eclipse IDE，选中项目，点击鼠标右键，选中 Properties。
- 2) 在弹出对话框中选中“Java Build Path”，选择右侧“Libraries”选项卡。



- 3) 在 Libraries 中点击右侧“Add External Jars”按键，在弹出对话框中选中添加“sdkapi.jar”。

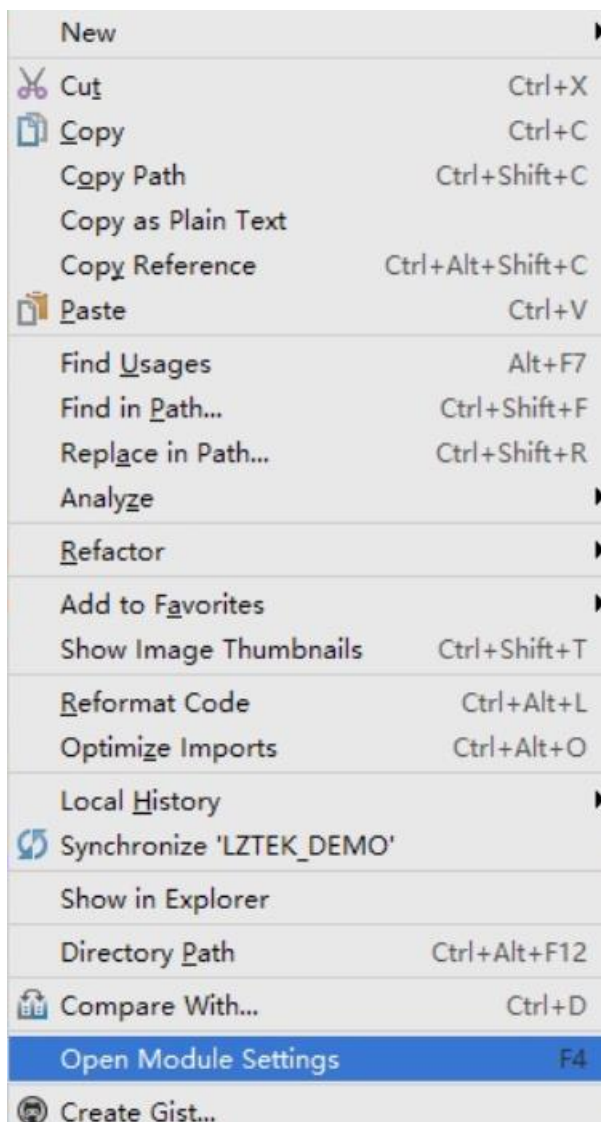


- 4) 选择右侧“Libraries”选项卡中点击右侧“Add External Jars”按键，在弹出对话框中选择相应目录添加“sdkapi.jar”。

1.2 Android Studio 集成

➔ 如何在 Android Studio 中使用 API 的方法如下：

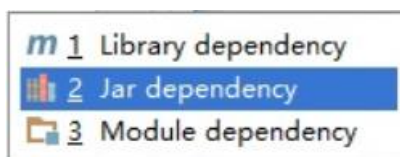
- 1) 将 sdkapi.jar 放至项目模块的第三方库的文件夹如 app\libs 中。
- 2) 在项目视图中右键单击项目名称弹出功能菜单并点击 “Open Module Settings”。



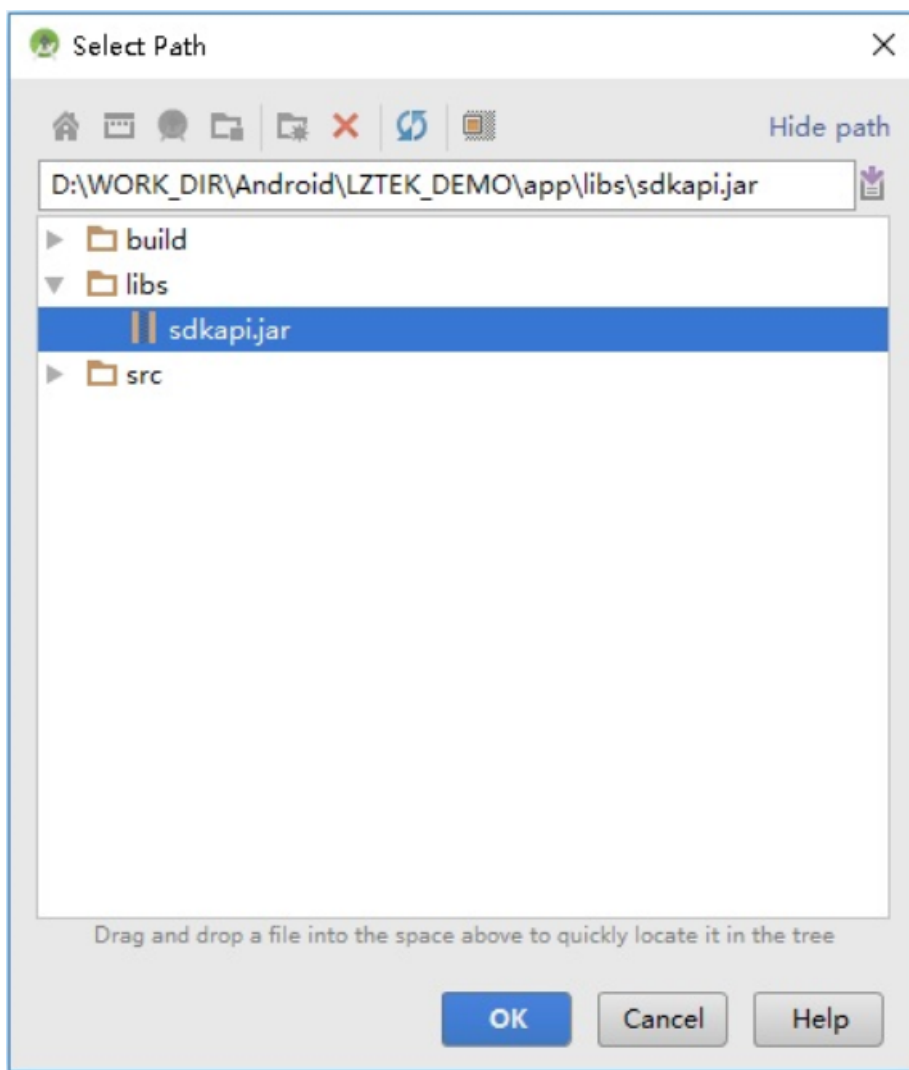
3) 选择 “Dependencies” 选项卡。



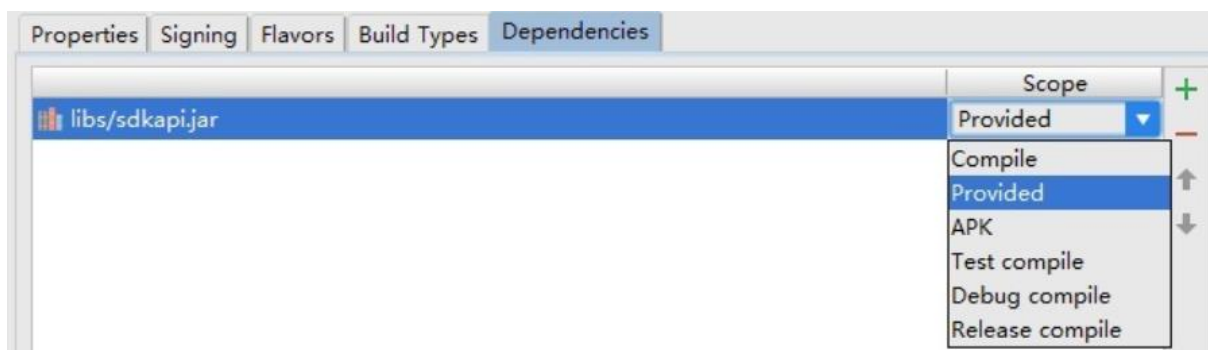
4) 点击右侧加号，弹出菜单中选择 “2 Jar Dependency”



- 5) 文件选择对话框中选中 sdkapi.jar，点击“OK”按钮返回。



- 6) 在列表框选中 sdkapi.jar 行，点击右侧“Scope”列下拉框，下拉框中选择“Provided”项，并点击下方“OK”按钮确认。

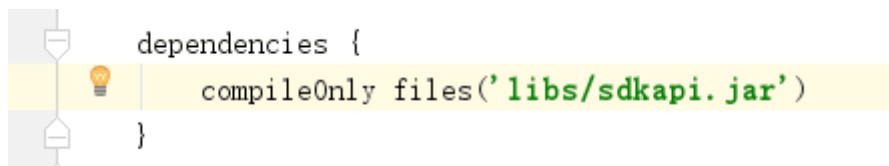


说明 1: 对于 Android Studio 3.0 之前的版本, 可编辑项目模块 build.gradle 文件, 加入依赖配置段内容如 provided files('libs/sdkapi.jar')。



```
dependencies {  
    provided files('libs/sdkapi.jar')  
}
```

说明 2: 对于 Android Studio 3.0 以上的版本, 可编辑项目模块 build.gradle 文件, 加入依赖配置段内容如 compileOnly files('libs/sdkapi.jar')。



```
dependencies {  
    compileOnly files('libs/sdkapi.jar')  
}
```

如果需要混淆, 则配置文件 proguard-rules.pro 中请加入:

```
-dontwarn com.lztek.toolkit.**  
-keep class com.lztek.toolkit.** { *; }
```

1.3 API 调用方法

➔ 在工程中添加 sdkapi.jar, 其中所有的 API 均可通过 Lztek 对象调用, 调用的方法举例如下:

```
Lztek lztek = Lztek.create(context);  
boolean enable = lztek.getEthEnable();
```

1.4 APK 系统签名

如果 APK 需要进行系统签名, 请访问我司签名云服务器 <http://47.107.162.209:18080> (7.1 以下)

或 <http://47.107.162.209:19090> (7.1 和以上) 上传 APK 进行签名和下载。



2 硬件管理

本章描述主板硬件相关资源的软件编程接口，包括无线和网络、存储空间、运行内存、背光开关、系统时间、GPIO、串口、硬件看门狗、关机/重启、定时开关机等。

2.1 无线和网络

功能概述：设置以太网 IP 参数、读取以太网 IP 参数、打开/关闭以太网、读取当前网络连接。

注意：使用以太网设置功能时，清单文件需增加如下权限申明：

```
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
```

2.1.1 读取以太网状态

➔ boolean getEthEnable()

参数名/返回值	类型	描述	示例
返回值	boolean	true - 以太网打开 false - 以太网关闭	

2.1.2 打开/关闭以太网

➔ void setEthEnable(boolean enable)

参数名/返回值	类型	描述	示例
enable	boolean	true - 打开以太网 false - 关闭以太网	

2.1.3 设置以太网静态地址

➔ void setEthIpAddress(String ip, String mask, String gateway, String dns)

描述：设置以太网静态地址。如果之前以太网被设置为 DHCP 模式，此调用将关闭 DHCP 模式。

参数名/返回值	类型	描述	示例
ip	String	IP地址	192.168.1.200
mask	String	子网掩码	255.255.255.0
gateway	String	网关地址	192.168.1.1
dns	String	域名解析服务器地址	8.8.8.8

2.1.4 设置以太网 DHCP 模式

➔ void setEthDhcpMode()

描述：将以太网设置为自动读取 IP 地址模式；如果要关闭 DHCP 模式则请直接调用设置以太网静态地址 API (setEthIpAddress)。

2.1.5 读取以太网设置

➔ AddrInfo getEthAddrInfo()

描述：返回网络地址信息类对象，其定义为 public class AddrInfo {}, 包含 DHCP 模式、IP 地址、子网掩码、网关地址等信息。

参数名/返回值	类型	描述	示例
返回值	AddrInfo	AddrInfo类对象	

2.1.6 读取以太网 MAC 地址

➔ String getEthMac()

描述：读取以太网硬件地址串，这个地址默认是根据 CPUID 或主板序列号生成的；如果需要设定特殊的 MAC 地址则请联系原厂技术支持接口。

参数名/返回值	类型	描述	示例
返回值	String	以太网MAC地址	00:01:02:03:04:05

2.1.7 读取当前网络类型

请使用安卓系统 `ConnectivityManager` 对象的 `getActiveNetworkInfo()`方法获得 `NetworkInfo` 对象，使用此对象的 `getType()`进行判断。

```
ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = cm.getActiveNetworkInfo();
int netType = networkInfo.getType();
if (ConnectivityManager.TYPE_ETHERNET == netType) {
    /* 网络类型：以太网 */
} else if (ConnectivityManager.TYPE_WIFI == netType) {
    /* 网络类型：WiFi */
} else if (ConnectivityManager.TYPE_MOBILE == netType) {
    /* 网络类型：移动网络 */
}
```

2.1.8 WiFi 网络操作

请直接使用安卓系统标准 API 接口。

2.1.9 读取移动网络设置

➔ `AddrInfo getMobileAddrInfo()`

描述：返回网络地址信息类对象，其定义为 `public class AddrInfo { }`，暂时只支持读取 IP 地址、子网掩码。

参数名/返回值	类型	描述	示例
返回值	AddrInfo	AddrInfo类对象	

2.2 存储空间

功能概述：读取内部存储路径、读取外部 SD 卡路径、读取外部 U 盘路径、读取内部存储空间、读取外部 SD 卡空间、读取 USB 存储空间。

2.2.1 读取内部存储路径

➔ `String getInternalStoragePath ()`

描述：读取主板内置存储器路径如/mnt/embsd 或/mnt/internal_sd。建议使用安卓系统标准方法 android.os.Environment.getExternalStorageDirectory()读取以提高程序的兼容性。

参数名/返回值	类型	描述	示例
返回值	String	主板内置存储器路径	/mnt/internal_sd

2.2.2 读取外部 SD 卡路径

➔ String getStorageCardPath()

描述：读取主板外部 SD 卡存储器路径如/mnt/extsd 或/mnt/external_sd。

参数名/返回值	类型	描述	示例
返回值	String	主板外部SD卡路径	/mnt/external_sd

2.2.3 读取外部 U 盘路径

➔ String getUsbStoragePath()

描述：读取主板 U 盘存储器路径如/mnt/udisk 或/mnt/usb_storage。

参数名/返回值	类型	描述	示例
返回值	String	主板外部U盘存储路径	/mnt/usb_storage

2.2.4 读取存储器容量

推荐使用系统 API 来读取存储器的容量大小，可通过 android.os.StatFs 对象读取。比如读取外部 SD 卡的容量可用如下代码实现：

```
android.os.StatFs statfs = new android.os.StatFs("/mnt/extsd");
long blocSize = statfs.getBlockSizeLong();
long availableSize = statfs.getAvailableBlocksLong()*blocSize;
long totalSize = statfs.getBlockCountLong()*blocSize;
```


2.3 运行内存

功能概述：读取系统运行内存大小。

2.3.1 读取系统内存大小

➔ `long getSystemMemory()`

描述：读取主板运行内存容量大小，单位为字节数，比如 1GB 则返回值等于 1024*1024*1024。

参数名/返回值	类型	描述	示例
返回值	long	内存大小，单位Byte	1073741824

2.4 背光开关

功能概述：打开/关闭 LCD 背光。

2.4.1 打开/关闭 LCD 背光

➔ `void setLcdBackLight(boolean on)`

描述：打开或关闭 LCD 液晶屏的背光电源并关闭显示输出信号；注意此接口并不能控制 HDMI

外接显示器/电视机的背光。

参数名/返回值	类型	描述	示例
on	boolean	true - 打开背光 false - 关闭背光	

2.5 系统时间

功能概述：设置安卓系统时钟。

2.5.1 设置安卓系统时钟

➔ `void setSystemTime(long milliseconds1970)`

描述：设置安卓系统的实时时钟时间，参数为从 1970 年开始到设置点的总秒数。请在应用中申明权限并进行系统签名<uses-permission android:name="android.permission.SET_TIME" />。

参数名/返回值	类型	描述	示例
milliseconds1970	long	从1970到设置点的秒数	

2.6 GPIO

功能概述：设置 IO 输入/输出、读取 IO 高低状态、设置 IO 高低状态。

2.6.1 使能 IO 口

➔ boolean gpioEnable(int port)

描述：将某个 IO 口设置为使能状态，IO 只有使能后才能进行输入或者输出设置。具体的 IO 编号需要根据硬件设计来确定，请查询相应的主板手册。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
port	int	GPIO对应的端口号	180
返回值	boolean	true - 使能IO成功 false - 使能IO失败	

2.6.2 设置 IO 方向-输入

➔ void setGpioInputMode(int port)

描述：将某个 IO 口设置为输入状态；注意设置前需要先将 IO 使能。具体的 IO 编号需要根据硬件设计来确定，请查询相应的主板手册。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
port	int	GPIO对应的端口号	180

2.6.3 设置 IO 方向-输出

➔ void setGpioOutputMode(int port)

描述：将某个 IO 口设置为输出状态；注意设置前需要先将 IO 使能。具体的 IO 编号需要根据硬件设计来确定，请查询相应的主板手册。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
port	int	GPIO对应的端口号	180

2.6.4 读取 IO 高低状态

```
➔ int getGpioValue(int port)
```

描述：读取某个 IO 口的高低状态，返回 0 代表低电平，返回 1 代表高电平；注意读取前需要先将 IO 使能，如果未设置 IO 的方向则此操作会自动将 IO 设置为输入。具体的 IO 编号需要根据硬件设计来确定，请查询相应的主板手册。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
port	int	GPIO对应的端口号	180
返回值	int	0 - 低电平；1 - 高电平	

2.6.5 设置 IO 高低状态

```
➔ void setGpioValue(int port, int value)
```

描述：设置某个 IO 口的高低状态，value=0 代表输出低电平，value=1 代表输出高电平；注意设置前需要先将 IO 使能并设置为输出状态。具体的 IO 编号需要根据硬件设计来确定，请查询相应的主板手册。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
port	int	GPIO对应的端口号	180
value	int	0 - 低电平；1 - 高电平	

2.7 串口

功能概述：打开串口、关闭串口、串口读数据、串口写数据。

2.7.1 打开串口

➔ `SerialPort openSerialPort(String path, int baudrate, int dataBit, int parity, int stopBits, int dataFlow)`

描述：打开指定串口设备并返回一个 `SerialPort` 对象，该对象的定义请参考 jar 包；如果打开失败则返回 `null`。目前该接口只实现了 `path` 和 `baudrate` 设置，其他参数都是默认值，建议直接使用下面的简化接口。

参数名/返回值	类型	描述	示例
<code>path</code>	<code>String</code>	串口端口设备路径	<code>/dev/ttyS1</code>
<code>baudrate</code>	<code>int</code>	串口通信波特率	<code>9600, 115200</code>
<code>dataBit</code>	<code>int</code>	数据位-只支持8bit模式	<code>8</code>
<code>parity</code>	<code>int</code>	奇偶校验-只支持无校验模式	<code>0</code>
<code>stopBits</code>	<code>int</code>	停止位-只支持1bit模式	<code>1</code>
<code>dataFlow</code>	<code>int</code>	数据流控-只支持无流控模式	<code>0</code>
返回值	<code>SerialPort</code>	自定义 <code>SerialPort</code> 类对象	

➔ `SerialPort openSerialPort(String path, int baudrate)`

描述：打开指定串口设备并返回一个 `SerialPort` 对象，该对象的定义请参考 jar 包；如果打开失败则返回 `null`。该接口只实现了 `path` 和 `baudrate` 设置，其他参数均为系统默认值（数据位=8、奇偶校验=无、停止位=8、数据流控=无）。

2.7.2 串口读数据

通过 `SerialPort` 对象的 `getInputStream` 方法获得一个 `InputStream` 接口，使用此接口的系统 API 进行数据读取操作。

2.7.3 串口写数据

通过 `SerialPort` 对象的 `getOutputStream` 方法获得一个 `OutnputStream` 接口，使用此接口的系统 API 进行数据输出操作。

2.7.4 关闭串口

➔ `void close(SerialPort port)`

描述: 关闭指定的串口对象。注意: 串口关闭时会自动关闭打开的流接口!

2.8 硬件看门狗

功能概述: 打开硬件看门狗、关闭硬件看门狗、硬件看门狗喂狗。

2.8.1 打开看门狗

➔ `boolean watchDogEnable()`

描述: 打开硬件看门狗。硬件看门狗打开后需要定时喂狗, 一旦软件停止喂狗则看门狗定时器

超时后系统就自动重启了。**注意: 该 API 需要 root 权限!**

参数名/返回值	类型	描述	示例
返回值	boolean	true - 打开硬件看门狗成功 false - 打开硬件看门狗失败	

2.8.2 看门狗喂狗

➔ `boolean watchDogFeed()`

描述: 硬件看门狗定时喂狗接口。此接口返回值请忽略, 除非是没打开看门狗设备就进行喂狗

操作则会返回 false。**注意: 硬件看门狗超时时间通常为 10~20 秒钟, 应用软件中请至少每隔 5 秒钟**

进行一次喂狗操作。该 API 需要 root 权限!

参数名/返回值	类型	描述	示例
返回值	boolean	此返回值请忽略	

2.8.3 关闭看门狗

➔ `boolean watchDogDisable()`

描述：关闭硬件看门狗，此接口返回值请忽略。硬件看门狗关闭后则不用在定期喂狗了。**注意：**

该 API 需要 root 权限！

参数名/返回值	类型	描述	示例
返回值	boolean	此返回值请忽略	

2.9 开关机

功能概述：硬件关机、硬件重启、定时重启、定时开机。

2.9.1 硬件关机

➔ void hardShutdown()

描述：基于开关机电路的硬件关机操作。关机后只能通过遥控器、重新插拔电源、开机按键等方式开机。**注意：该 API 需要 root 权限！**

2.9.2 软件重启

➔ void softReboot()

描述：通过系统 reboot 命令实现的系统热复位重启。注意软件重启只是让 CPU 重新从初始指令开始启动，而并不能确保所有硬件设备和接口的完全复位。**注意：该 API 需要 root 权限！**

2.9.3 硬件重启

➔ void hardReboot ()

描述：通过硬件开关机电路实现的系统关机重启。此接口可以确保系统除了 12V 输入和待机 5V 之外的电源断电，是一种比较可靠的掉电重启机制。因为此命令需要触发一个关机断电动作，因而并不能实时重启，从断电到重新上电之间的时间间隔为几十秒钟（这个时间最大不会超过 60 秒）。

注意：该 API 需要 root 权限！

2.9.4 定时开机

➔ void alarmPoweron(int onSeconds)

描述：此接口调用后会立即关机并在指定的时间秒数后自动开机。参数秒数最小为 60 秒即 1 分钟以后自动开机。比如现在是 18:00，希望主板现在关机并于第二天 08:00 开机，则参数应该设置为 50400。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
onSeconds	int	从当前关机点到下次开机点的秒数	

2.10 HDMIIN

功能概述：读取 HDMI 输入状态、读取 HDMI 输入分辨率。注意：HDMIIN 功能需硬件主板支持。

2.10.1 读取 HDMI 输入状态

➔ int getHdmiinStatus()

描述：获取 HDMI 输入信号的状态，返回 1 代表有 HDMI 输入信号，返回 0 代表无 HDMI 输入信号。

参数名/返回值	类型	描述	示例
返回值	int	1 - HDMIIN有信号 0 - HDMIIN无信号	

注意：此 API 需 1.1.0.20170818 版本 SDK 才能支持；且此 API 仅用于未打开 HDMIIN 摄像头预览时调用，一旦检测到信号进入 HDMIIN 预览处理流程，则只能通过查询安卓系统属性 `getprop sys.hdmiin.status` 来判断 HDMIIN 输入信号状态（返回 1 代表有信号，返回 0 代表无信号）。

2.10.2 读取 HDMI 输入分辨率

➔ int getHdmiinResolution()

描述: 获取 HDMI 输入信号的分辨率, 返回 0 代表未知, 返回 1 代表 1080P 即 1920x1080, 返回 2 代表 720P 即 1280x720。注意: 目前只支持 1080P 和 720P 两种输入分辨率! 且需 1.1.0.20170818 版本才能支持!

参数名/返回值	类型	描述	示例
返回值	int	0 - HDMIIN未知 1 - HDMIIN 1080P 2 - HDMIIN 720P	

注意: 此 API 需 1.1.0.20170818 版本 SDK 才能支持; 且此 API 仅用于未打开 HDMIIN 摄像头预览时调用, 一旦检测到信号进入 HDMIIN 预览处理流程, 则只能通过查询安卓系统属性 `getprop sys.hdmiin.resolution` 来判断 HDMIIN 输入信号格式 (返回 1 代表 1080P, 返回 2 代表 720P)。

3 安卓管理

本章描述主板安卓平台相关资源的软件编程接口，包括显示管理、导航条、状态条、应用管理、系统信息、系统升级、Shell 命令执行。

3.1 显示管理

功能概述：读取分辨率、屏幕截屏、屏幕旋转、读取显示密度、设置显示密度（需重启）。

3.1.1 读取分辨率

请使用安卓系统 API 读取分辨率，比如 `WindowManager().getDefaultDisplay().getRealMetrics(dm)` 方法可以得到屏幕的物理分辨率，`WindowManager().getDefaultDisplay().getMetrics(dm)`可以得到去掉导航栏高度的分辨率。

3.1.2 屏幕截屏

➔ `Bitmap screenCapture()`

描述:按照屏幕原始尺寸大小截取屏幕内容并保存返回一个位图对象;如果调用失败则返回 `null`。

参数名/返回值	类型	描述	示例
返回值	Bitmap	Bitmap类对象	

➔ `void screenCapture(String path)`

描述：按照屏幕原始尺寸大小截取屏幕内容并保存到指定路径文件中。

参数名/返回值	类型	描述	示例
path	String	截屏文件保存路径	/mnt/external_sd/1.png

3.1.3 屏幕旋转

请使用安卓系统接口 `Settings.System` 的 `getInt/putInt` 方法对屏幕旋转角度属性读取/设置，属性名称为 `Settings.System.USER_ROTATION`。旋转角度值参数包括 `Surface.ROTATION_0/90/180/270` (比如：`Settings.System.putInt(contentResolver, Settings.System.USER_ROTATION, Surface.ROTATION_270)`)

申明权限：`<uses-permission android:name="android.permission.WRITE_SETTINGS" />`

3.1.4 读取显示密度

➔ `int getDisplayDensity()`

描述：读取安卓系统显示密度值 (`ro.sf.lcd_density`)。常规的密度值如 120 - ldpi、160 - mdpi、240 - hdpi、320 - xhdpi、480 - xxhdpi。

参数名/返回值	类型	描述	示例
返回值	int	系统显示密度值	160

3.1.5 设置显示密度

➔ `void setDisplayDensity(int density)`

描述：设置安卓系统显示密度值，**该 API 调用完成后系统会自动重启**，重启后新的密度值才能生效。常规的密度值如 120 - ldpi、160 - mdpi、240 - hdpi、320 - xhdpi、480 - xxhdpi。**注意：该 API 需要 root 权限！**

参数名/返回值	类型	描述	示例
返回值	int	系统显示密度值(80~640)	160

3.2 导航条

功能概述：显示导航条、隐藏导航条、打开/关闭滑动开关、设置自动隐藏时间。

3.2.1 显示导航条

➔ void showNavigationBar ()

描述：显示安卓底部导航条。如果打开了滑动呼出导航条开关且设置了超时隐藏，则系统空闲超时后导航条一样会被自动隐藏。

3.2.2 隐藏导航条

➔ void hideNavigationBar ()

描述：隐藏安卓底部导航条。

3.2.3 打开/关闭滑动

➔ void navigationBarSlideShow(boolean enable)

描述：打开或关闭滑动显示系统导航条属性。打开滑动开关后，即使系统导航条被隐藏，也可以通过从屏幕底部向上滑动来显示导航条。

参数名/返回值	类型	描述	示例
enable	boolean	true - 打开导航条滑动开关 false - 关闭导航条滑动开关	

3.2.4 导航条自动隐藏

➔ void navigationBarMaxIdle(int seconds)

描述：设置系统无人操作空闲状态多长时间后自动隐藏导航条。如果设置的超时秒数大于 0，则相应秒数无操作自动隐藏导航条，后继通过滑动还可以呼出导航条；如果设置的超时秒数小于或等于 0，则自动关闭滑动功能，若此时导航条已经处于显示状态则不再自动隐藏。

3.3 状态条

安卓状态条承载了安卓系统的网络、时间、设置、消息、通知等动态信息的集中展示，我们不提供方法进行动态隐藏和显示控制，只提供专门的系统版本要么完全不显示状态条、要么显示标准的状态条。并且请注意：如果使用显示状态条的版本则必须同时显示导航条，否则状态条的滑动呼出会无法正常进行。

3.4 应用管理

功能概述：读取应用列表（系统/普通/全部）、应用静默安装、应用静默卸载、应用自动启动。

3.4.1 读取应用列表

请使用系统 API。

3.4.2 应用静默安装

➔ void installApplication(String apkPath)

描述：指定需要安装的 APK 存放路径即可自动完成安装，安装时无需用户交互。**注意：该 API**

需要 root 权限！

参数名/返回值	类型	描述	示例
apkPath	String	需要安装的APK的存放路径	

3.4.3 应用自动安装

请使用如下方法进行自动安装和执行，安装过程中会弹出安装进度条但是无需进行确认操作。

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(Uri.parse("安装包"), "application/vnd.android.package-archive");
intent.putExtra("IMPLUS_INSTALL", "SILENT_INSTALL"); // 自动安装并在安装后自动执行
startActivity(intent);
```

3.4.4 应用静默卸载

➔ void uninstallApplication(String packageName)

描述：指定需要卸载的 APK 包名即可自动完成卸载，卸载时无需用户交互。此处请注意函数的参数并不是 APK 的文件名而是实际的包名，比如要卸载微信程序则包名通常为 com.tencent.mm。

注意：该 API 需要 root 权限！

参数名/返回值	类型	描述	示例
packageName	String	需要卸载的APK的包名	com.tencent.mm

3.4.5 应用自动卸载

请使用如下方法进行自动卸载，卸载过程中无需进行确认操作。

```
Intent intent = new Intent(Intent.ACTION_DELETE);
intent.setData( Uri.parse("package:" + 包名));
intent.putExtra("IMPLUS_UNINSTALL", "SILENT_UNINSTALL"); // 自动卸载应用
startActivity(intent);
```

3.4.6 应用自动启动

参考方法：程序中实现一个 Receiver，监听"android.intent.action.BOOT_COMPLETED"动作并处理自启动，应用需要添加"android.permission.RECEIVE_BOOT_COMPLETED"权限。主板默认发布包中也提供了一个开机启动程序，可以在其中指定需要自动启动的第三方程序。

3.5 系统信息

功能概述：读取安卓版本号、读取安卓序列号、读取系统版本号、读取内核版本、读取 API 版本号、读取 WiFi 硬件地址、读取以太网硬件地址。

3.5.1 读取安卓版本号

推荐使用系统 API：android.os.Build.VERSION.RELEASE，如："4.4.4"。

3.5.2 读取安卓序列号

推荐使用系统 API：android.os.Build.SERIAL，如："QK2141IQAK"。这个序列号是由系统内部基于 WiFi MAC 地址生成的一个硬件序列号,不同硬件序列号通常是不同的,可以作为硬件的唯一标识用。

3.5.3 读取系统版本号

➔ String getSystemVersion()

描述：读取主板安卓系统的 OEM 发布版本号，如 "2.0.0-170514-OEM"。推荐直接使用系统方法如 android.os.Build.DISPLAY 读取该信息。

参数名/返回值	类型	描述	示例
返回值	String	主板软件系统版本号	2.0.0-170514-OEM

3.5.4 读取内核版本

➔ String getKernelVersion()

描述：读取主板 Linux 内核版本号，如 "3.10.96"。

参数名/返回值	类型	描述	示例
返回值	String	主板Linux内核版本号	3.10.96

3.5.5 读取 API 版本

➔ String getApiVersion()

描述：读取本文档所描述的 API 接口统一软件版本号，如 "1.0.0.20170512"。

参数名/返回值	类型	描述	示例
返回值	String	本主板API接口软件版本号	1.0.0.20170512

3.5.6 读取 WiFi 硬件地址

推荐使用系统 API 通过 WifiManager 的 getConnectionInfo()方法读取 WifiInfo 对象，使用 WifiInfo 对象的 getMacAddress()得到 WiFi 硬件地址。

3.5.7 读取以太网硬件地址

请参考 2.1.6 小节的“读取以太网 MAC 地址”内容。

3.6 系统升级

功能概述：指定系统 OTA 升级文件并完成系统升级。

3.6.1 系统 OTA 升级

➔ void updateSystem(String updateFilePath)

描述：指定安卓系统 OTA 升级包（通常为主板厂商提供的 OTA 包 update.zip）并自动重启进入 OTA 升级模式完成系统升级，升级过程会出现一个安卓机器人动图加进度条界面，等待升级进度完成后系统会自动重启并进入安卓。**注意：该 API 需要 root 权限！调用此命令之前请务必保证升级文件的完整性，建议文件拷贝到指定路径后请进行一下文件大小和内容校验，这样做可以确保文件拷贝动作可靠完成且不会有内容在硬件缓存中（尚未真正同步到物理存储器）。升级完成后如果原始升级包不再需要，请自行删除该文件以节省存储空间。**

参数名/返回值	类型	描述	示例
updateFilePath	String	需要升级的安卓系统包路径	

3.7 Shell 命令执行

功能概述：执行指定的 Shell 命令。

3.7.1 Shell 命令执行

➔ void suExec(String command)

描述：以 root 权限执行 Linux Shell 命令。**注意：该 API 需要 root 权限！由于安卓系统的 root 是通过一个后台服务程序中转实现的超级用户权限，本接口 shell 命令的执行并不是完全阻塞同步的，**

比如使用本接口执行一个文件拷贝操作并不能保证文件拷贝完成之后调用才返回。如果需要 shell 命令严格同步执行请参考如下代码自己实现，但是请严格避免进程间等待死锁的情况。

参数名/返回值	类型	描述	示例
command	String	需要执行的Shell命令	chmod 0666 /dev/video0

➔ 严格同步执行的 shell 命令调用方式（未提供 API）：

```
private static String suExecWait (String command, File workingDirectory) {
    if (command == null || (command=command.trim()).length() == 0)
        return null;
    if (workingDirectory == null)
        workingDirectory = new File("/");
    java.io.OutputStream out = null;
    java.io.InputStream in = null;
    java.io.InputStream err = null;
    try {
        Runtime runtime = Runtime.getRuntime();
        Process process = runtime.exec("su", null, workingDirectory);
        StringBuffer inString = new StringBuffer();
        StringBuffer errString = new StringBuffer();
        out = process.getOutputStream();

        out.write(command.endsWith("\n")? command.getBytes() : (command + "\n").getBytes());
        out.write(new byte[]{'e', 'x', 'i', 't', '\n'});

        in = process.getInputStream();
        err = process.getErrorStream();

        process.waitFor(); // 此行会阻塞执行，直到命令返回

        while (in.available() > 0)
            inString.append((char)in.read());
        while (err.available() > 0)
            errString.append((char)err.read());
        return inString.toString();
    } catch (Exception ioex) {
        return null;
    } finally {
        closeStream(out);
        closeStream(in);
        closeStream(err);
    }
}
```



```
}  
}
```

4 定时开关机广播接口

[功能说明]: 定时开关机设置广播接口 (需安装定时开关机 Apk 程序)

注意: 因安卓 8.0 或以上系统静态广播权限限制, 发送广播时必须指定相应包名, 因此发送广播时必须增加 `intent.setPackage("com.lztek.bootmaster.poweralarm7")`。

- ◆ 每天模式定时开关机 `com.lztek.tools.action.ALARM_DAILY`

`onTime` -- 开机时间, 类型 `String`, 格式 `HH:mm`, 24 小时制的时与分如 `08:05`, 空字符串表示无开机时间设置

`offTime` -- 关机时间, 类型 `String`, 格式 `HH:mm`, 24 小时制的时与分如 `20:30`, 空字符串表示无关机时间设置

调用示例(每天 08:05 开机、20:30 关机):

```
Intent intent = new Intent("com.lztek.tools.action.ALARM_DAILY");
intent.putExtra("onTime", "08:05");
intent.putExtra("offTime", "20:30");
intent.setPackage("com.lztek.bootmaster.poweralarm7"); // Android 8.0 or above
context.sendBroadcast(intent);
```

- ◆ 星期模式定时开关机 `com.lztek.tools.action.ALARM_WEEKLY`

`onTime` -- 开机时间, 类型 `String[]`, 数组长度必须为 7(星期日~星期六), 数组元素格式 `HH:mm`, 空字符串表示当天无开机时间设置

`offTime` -- 关机时间, 类型 `String[]`, 数组长度必须为 7(星期日~星期六), 数组元素格式 `HH:mm`, 空字符串表示当天无关机时间设置

调用示例(周一、周二、周三、周四指定时间点开机; 周一、周二、周三、周五指定时间点关机):

```
Intent intent = new Intent("com.lztek.tools.action.ALARM_WEEKLY");  
intent.putExtra("onTime", new String[]{"", "08:05", "09:15", "10:00", "10:00", "", ""});  
intent.putExtra("offTime", new String[]{"", "21:45", "21:05", "21:00", "", "22:00", ""});  
intent.setPackage("com.lztek.bootmaster.poweralarm7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```

- ◆ 清除定时开关机设置 com.lztek.tools.action.ALARM_UNSET

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.ALARM_UNSET");  
intent.setPackage("com.lztek.bootmaster.poweralarm7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```

5 应用启动管理广播接口

[功能说明]: 应用启动管理-开机直达应用守护广播接口 (需安装应用启动管理 Apk 程序)

注意: 因安卓 8.0 或以上系统静态广播权限限制, 发送广播时必须指定相应包名, 因此发送广播时必须增加 `intent.setPackage("com.lztek.bootmaster.autoboot7")`。

◆ 设置应用守护 `com.lztek.tools.action.KEEPALIVE_SETUP`

`packageName` -- 需要设置守护的应用包名, 类型 `String`

`delaySeconds` -- 应用退出后延迟启动的秒数, 类型 `int`, 默认值 0 秒立即启动

`foreground` -- 应用保持前台运行(不在前台则重新打开), 类型 `boolean`, 默认值 `true`

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.KEEPALIVE_SETUP");
intent.putExtra("packageName", "xxx.xxxx.xxx");

//intent.putExtra("delaySeconds", 5); // 应用退出后 5 秒重新启动

//intent.putExtra("foreground", false); // 应用可后台运行, 进程退出后才重新打开

intent.setPackage("com.lztek.bootmaster.autoboot7"); // Android 8.0 or above
context.sendBroadcast(intent);
```

◆ 取消应用守护 `com.lztek.tools.action.KEEPALIVE_UNSET`

`packageName` -- 需要设置守护的应用包名, 类型 `String`

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.KEEPALIVE_UNSET");
```

```
intent.putExtra("packageName", "xxx.xxx.xxx");  
intent.setPackage("com.lztek.bootmaster.autoboot7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```

- ◆ 取消所有应用守护 com.lztek.tools.action.KEEPALIVE_UNSET_ALL

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.KEEPALIVE_UNSET_ALL");  
intent.setPackage("com.lztek.bootmaster.autoboot7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```

-
- ◆ 设置应用开机直达 com.lztek.tools.action.BOOT_SETUP

packageName -- 开机直达启动的应用包名, 类型 String

delaySeconds -- 延迟直达启动的秒数, 类型 int, 默认值 0 秒即开机立即启动

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.BOOT_SETUP");  
intent.putExtra("packageName", "xxx.xxx.xxx");  
  
//intent.putExtra("delaySeconds", 5); // 开机启动完成 5 秒后运行指定 APK  
intent.setPackage("com.lztek.bootmaster.autoboot7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```

- ◆ 取消应用开机直达 com.lztek.tools.action.BOOT_UNSET

packageName -- 开机直达启动的应用包名, 类型 String

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.BOOT_UNSET");
```

```
intent.putExtra("packageName", "xxx.xxxx.xxx");  
intent.setPackage("com.lztek.bootmaster.autoboot7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```

- ◆ 取消所有应用开机直达 com.lztek.tools.action.BOOT_UNSET_ALL

调用示例:

```
Intent intent = new Intent("com.lztek.tools.action.BOOT_UNSET_ALL");  
intent.setPackage("com.lztek.bootmaster.autoboot7"); // Android 8.0 or above  
context.sendBroadcast(intent);
```