多应用架构库ApplMngr

编程指南

V1.0.0

惠尔丰电子（北京）有限公司

2010年9月4日

修订历史记录

**A**-添加，**M**-修改，**D**-删除

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **日期** | **AMD** | **修订者** | **说明** |
| 1.0 | 2010-9-4 | A | 于海涛 | 创建 |
|  |  |  |  |  |
|  |  |  |  |  |

[第一章 前言 4](#_Toc271463754)

[第二章 动手之前的要知道的 5](#_Toc271463755)

[1. 了解一下多应用架构库ApplMngr实现哪些功能？ 5](#_Toc271463756)

[2. 了解一下ApplMngr库的基本模型 6](#_Toc271463757)

[3. ApplMngr库在什么环境下使用？ 7](#_Toc271463758)

[第三章 函数接口API 8](#_Toc271463759)

[1. 应用信息结构 8](#_Toc271463760)

[2. 应用架构处理 9](#_Toc271463761)

[3. 应用事件驱动表 10](#_Toc271463762)

[4. 应用切换 12](#_Toc271463763)

[5. 应用调用 13](#_Toc271463764)

[6. 定时器 14](#_Toc271463765)

[7. 广播通知 15](#_Toc271463766)

[第四章 写一个DEMO 16](#_Toc271463767)

[1. 写一个主控应用MAINAPP 16](#_Toc271463768)

[2. 写一个子应用DEMOA 16](#_Toc271463769)

[3. 写一个子应用DEMOB 16](#_Toc271463770)

[4. 写一个子应用DEMOC 16](#_Toc271463771)

[5. 写一个子应用DEMOD 16](#_Toc271463772)

[第五章 编程要点 17](#_Toc271463773)

[1. 初始化ApplMngr\_AppInit的位置 17](#_Toc271463774)

[2. 子应用注册ApplMngr\_SubAppRegister的时机 17](#_Toc271463775)

[3. 没有默认应用？ 17](#_Toc271463776)

[4. 事件驱动的特点 17](#_Toc271463777)

[5. 键盘缓冲和磁道缓冲的异常处理 17](#_Toc271463778)

[6. MIS-POS模式的实现 17](#_Toc271463779)

[7. 隐身应用的使用 18](#_Toc271463780)

[8. 定时器的使用及循环定时器的用法 18](#_Toc271463781)

[9. 应用之间传递的数据最大能支持多长？ 18](#_Toc271463782)

[10. 休眠模式 18](#_Toc271463783)

[11. 广播事件的使用 18](#_Toc271463784)

[12. 多应用架构库的跟踪调试 19](#_Toc271463785)

[第六章 常见问题及解答 20](#_Toc271463786)

[1. 应用切换失败，总回到当前应用 20](#_Toc271463787)

[2. CALLFUNC调用失败 20](#_Toc271463788)

[3. 多个子应用中，只有部分子应用被运行 20](#_Toc271463789)

[4. 多个子应用中，只看到部分子应用注册信息 20](#_Toc271463790)

[5. Vx670/680待机状态无法进入休眠 20](#_Toc271463791)

# 前言

此文档是指导开发人员在Vx系列的POS上如何使用多应用架构库ApplMngr快速开发应用程序的指南。

此文档的读者需要具备以下能力：

* C语言开发能力
* 对多进程概念有所了解
* 对POS系统有所了解
* 对多应用基本概念有所了解
* Vx平台开发和调试经验

如果您开发过VMAC的程序或者Nurit多应用程序将对深入了解此文档有很大的帮助。

如果需要运行附件中的DEMO程序，您至少还需要：

* 一台带串口的电脑
* VC6.0
* ARM RVDS4.0编译环境
* Verix eVoAps编译库
* Applib库集
* FST（verifone signing tools）V3.03.05
* 签名卡及读卡器

# 动手之前的要知道的

### 了解一下多应用架构库ApplMngr实现哪些功能？

* 建立主应用和子应用的逻辑：

通过简单的调用ApplMngr库的几个API函数，就实现了主控应用和子应用的创建。

* 应用切换：

通过调用ApplMngr库的一个API函数（ApplMngr\_SwitchToApp），实现了应用之间的切换。

* 应用之间的调用：

通过调用ApplMngr库的一个API函数（ApplMngr\_CallFunc），实现了应用间的CALLFUNC功能。

* 设备的管理

设备管理在ApplMngr库内部实现，应用无需关心。

* 省电模式

省电模式在ApplMngr库内部自动判断机型并在恰当的时机处理，应用无需关心。

* 定时器

ApplMngr库提供给应用若干个定时器，由应用自由使用。

* 广播

ApplMngr库提供给应用可以广播通知给所有应用消息的功能。

### 了解一下ApplMngr库的基本模型



### ApplMngr库在什么环境下使用？

* 应用必须使用RVDS4.0编译：

由于新的Vx平台都转向RVDS4.0编译环境，因此ApplMngr库也顺应了潮流。但是由于RVDS4.0不兼容RVDS2.0，原先老的Verix平台将无法使用此库。但是目前的主流机型（如：Vx510、Vx670）都可以通过升级OS的方式使用RVDS4.0编译的程序。

* 应用可以运行的平台

Vx平台：Vx510/670 (OS 12A2以上)

Trident平台：Vx520/680及以后机型

* 应用必须编译成.OUT：

ApplMngr库提供的架构是基于进程方式的多应用，因此无论是主控应用还是子应用，都需要编译成.OUT文件，以独立进程的方式在POS中运行。

# 函数接口API

您也可以参考《ApplMngr.h》中的API定义。

### 应用信息结构

typedef struct \_AppRegInfo

{

char AppDispName[32]; //应用显示TITLE，最长32个字节

char AppName[8]; //应用名称，最长8个字节

char ReleaseVer[32]; //客户定义版本号，从应用版本信息文件中得到

char ManageVer[32]; //应用管理版本号，从应用版本信息文件中得到

char Corp[32]; //公司信息

char BuildDate[32]; //应用生成日期

int HiddenFlag; //是否是隐藏应用

}AppRegInfo;

**说明：**应用信息主要是在子应用中配置，用于子应用向主应用注册使用。

* 一般情况下主应用在收到子应用的信息后，会在列表中显示出子应用的AppDispName。
* 而AppName则用于应用切换、应用调用及内部的消息传递时应用指向名称，所以每个子应用的AppName不能重复，也不可为空。
* 为了便于管理，除了正常外部显示的版本号ReleaseVer外，我们还定义了一个ManageVer方便对应用程序进行管理。
* HiddenFlag等于1表示为隐藏应用，此时主控应用得到的应用信息列表中将不会包括此应用；但是与所有应用都相关的操作依然会涉及到隐藏应用（如休眠广播）。

### 应用架构处理

//应用初始化

extern int ApplMngr\_AppInit(const char \*szCurrAppName);

//子应用注册

extern int ApplMngr\_SubAppRegister(const char \*szMainAppName, AppRegInfo \*p\_CurAppInfo);

//主应用启动子应用并等待子应用注册信息

extern int ApplMngr\_WaitingSubAppInfo(AppRegInfo \*p\_AppRegList,int \*RegAppNum);

//主应用重新获取子应用注册信息

extern int ApplMngr\_ReGetSubAppInfo(AppRegInfo \*p\_AppRegList,int \*RegAppNum);

//应用事件处理引擎

extern int ApplMngr\_AppLoop(TVxEvdHeader \*pEventList);

//判断当前是否是激活应用

extern int ApplMngr\_IsActiveApp(void);

**说明：**

* 搭建主控应用的调用顺序：ApplMngr\_AppInit🡪 ApplMngr\_WaitingSubAppInfo🡪 ApplMngr\_AppLoop
* 搭建子应用的调用顺序：ApplMngr\_AppInit🡪 ApplMngr\_SubAppRegister 🡪 ApplMngr\_AppLoop
* 主控应用的启动通过系统设定的\*GO实现，子应用则由主控应用在调用ApplMngr\_WaitingSubAppInfo函数时自动启动，在主应用启动子应用时，会将主应用名通过参数传递给子应用。子应用可以按照如下方式得到主应用名：

int main (int argc, char \*argv[])

{

if(argc > 0)

strncpy(MainAppName , argv[1] , 8);

……

}

* ApplMngr\_WaitingSubAppInfo函数等待子应用注册信息的时间有限，所以建议子应用在main函数尽量靠前的地方做完ApplMngr\_AppInit🡪 ApplMngr\_SubAppRegister步骤，再执行其他初始化处理。
* 对于出现上述情况的子应用来说，ApplMngr库提供了一个附加函数ApplMngr\_ReGetSubAppInfo,主控可以通过其更新当前已经收到注册信息的子应用列表。

### 应用事件驱动表

//事件响应函数列表

typedef struct \_TVxEvdHead

{

ApplMngr\_CallBack OnActiveEvent; //应用被激活通知函数

ApplMngr\_CallBack OnBroadCastEvent; //广播通知触发事件

ApplMngr\_CallBack IdleLoop\_Scan; //循环函数

ApplMngr\_CallBack Redraw\_Screen; //待机屏幕显示

ApplMngr\_CallBack Call\_Func; //CALLFUNC响应函数

ApplMngr\_CallBack MagCard\_Event; //磁条卡事件触发函数

ApplMngr\_CallBack ICCCard\_Event; //用户IC卡事件触发函数

ApplMngr\_CallBack HotKey\_Event; //用户按键事件触发函数

ApplMngr\_CallBack Bar\_Event; //用户扫描枪事件触发函数

}TVxEvdHeader;

**说明：**

* 应用事件驱动列表用于ApplMngr\_AppLoop应用驱动引擎函数；
* 回调函数指针的原型：

//回调指针函数原型

typedef int (\*ApplMngr\_CallBack)(void \* lpParam);

* 驱动表中回调函数可以为NULL，为NULL时则此事件不会被触发；驱动表在传递给ApplMngr\_AppLoop时也可以为NULL，此时不会触发任何事件。
* 只有在待机状态下才能触发事件，并且相关事件函数处理完毕后才能响应下一次事件。
* 若有多个事件同时触发，其响应顺序如下：

OnActiveEvent🡪Call\_Func🡪HotKey\_Event🡪MagCard\_Event🡪 ICCCard\_Event🡪Bar\_Event🡪 Redraw\_Screen🡪 IdleLoop\_Scan

* OnActiveEvent事件的触发是在应用被激活时（如：被切换为当前应用、或被CALLFUNC调用），主控应用在启动时是被切换为当前应用，所以也会收到此事件触发。应用程序可以通过指针参数知道被激活的原因，参数定义如下：

enum //应用激活事件类型

{

ON\_SWITCHAPP = 1, //不能从0开始

ON\_CALLFUNC,

};

处理函数的DEMO如下：

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: EvdIFunc\_OnActive

\* DESCRIPTION: Be Actived event

\* PARAMETERS

\* pParam - (Input) Pointer of Actived Type

\* RETURN: none

\* NOTES: none.

\* ------------------------------------------------------------------------ \*/

int EvdIFunc\_OnActive(void \* pParam)

{

if(pParam != NULL){

if (pParam == (void \*)ON\_SWITCHAPP){

;

}

else if (pParam == (void \*)ON\_CALLFUNC){

;

}

}

return 0;

}

* OnBroadCastEvent事件是应用接收到广播后触发的事件，由某一个应用通过ApplMngr\_SendBroadCast发起。此事件在背景应用中仍有效。
* IdleLoop\_Scan事件是待机循环事件，默认为每一秒执行一次，无参数需要传递。注意不要在此事件中处理大量耗时（甚至死循环）的逻辑；只有这个正常返回，其他事件才能得到触发。此事件在背景应用中仍有效，但在终端休眠时不会被触发。
* Redraw\_Screen事件是待机界面显示事件，默认为每一秒执行一次。此事件和 IdleLoop\_Scan类似，只是在背景应用和终端休眠时不被触发。
* Call\_Func事件是应用被调用时触发的事件。此事件被触发后，应用回被切换为当前应用，可以操作所有的设备，实现相关功能处理。在Call\_Func事件处理完毕返回后，应用重新回到背景应用状态。在响应时，通过参数能够得到调用者传递过来的数据，在返回时，也可以通过参数传递回数据。如果应用不能被切换为当前应用，则不会处理此事件。
* HotKey\_Event事件是待机界面按键响应事件。其按键的键值缓冲指针通过参数传递给应用。因此，应用想要得到键值，需要对指针进行取值处理。在背景应用时此事件不被触发。在终端休眠时按键会触发此事件，同时唤醒终端退出休眠模式。此事件支持组合按键的键值。

处理函数的DEMO如下：

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: EvdIFunc\_DoHotkey

\* DESCRIPTION: Hot key event handler

\* PARAMETERS pParam - (Input) Pointer of key

\* RETURN:

\* NOTES: none.

\* ------------------------------------------------------------------------ \*/

int EvdIFunc\_DoHotkey(void \*pParam)

{

int \*KeyCode = (int\*)(pParam) ;

if(\*KeyCode == KEY\_CR){

;

}

return 0;

}

* MagCard\_Event、ICCCard\_Event、Bar\_Event事件是待机界面设备响应事件。无参数需要传递。对于磁条卡触发事件MagCard\_Event和用户IC卡触发事件ICCCard\_Event，系统不会清除其相关数据，对数据的读取和处理仍有应用来完成。在背景应用时这些事件不被触发，但是当前应用在终端休眠时会触发此事件，同时唤醒终端退出休眠模式。Bar\_Event为系统预留的扫描枪事件，暂时未处理。

### 应用切换

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: ApplMngr\_SwitchToApp.

\* DESCRIPTION: 切换应用到指定应用

\* PARAMETER:

IN:

\* szToAppName -- 应用名称，如“MAINAPP”

\* NOTES:

\* RETURN: 见返回值定义

\* ------------------------------------------------------------------------ \*/

extern int ApplMngr\_SwitchToApp(const char \* szToAppName);

**说明：**

* 当一个当前应用（应用A）需要切换到另外一个应用（应用B）时，通过此函数实现；
* szToAppName是应用B的名称，即应用B定义的应用信息结构中的AppName；
* 一旦切换成功，应用B将是当前应用，控制所有设备；应用A转为背景应用；
* 在切换成功时，应用B事件驱动表中的OnActiveEvent事件将被触发，并传递参数ON\_SWITCHAPP指针；
* 如果应用B此时不在待机状态，可能会导致切换不成功，此时当前应用仍是应用A，通过函数返回值可以得知失败原因；

### 应用调用

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: ApplMngr\_CallFunc.

\* DESCRIPTION: 调用指定应用的功能函数

\* PARAMETER:

IN:

\* szToAppName -- 被调用的应用名称

inMsgBuf -- 传递过去的数据

inMsgLen -- 传递过去的数据长度

OUT:

outMsgBuf -- 返回的数据

outMsgBufSize -- 接收返回的数据缓冲的大小

outMsgLen -- 返回的数据的实际产度

\* NOTES:

\* RETURN: 见返回值定义

\* ------------------------------------------------------------------------ \*/

extern int ApplMngr\_CallFunc(const char \* szToAppName, void \*inMsgBuf,int inMsgLen, void \*outMsgBuf, int outMsgBufSize,int \*outMsgLen);

**说明：**

* 当一个应用（应用A）需要调用另外一个应用（应用B）的功能时，应用A通过调用此函数实现；
* szToAppName是应用B的名称，即应用B定义的应用信息结构中的AppName；
* 在调用成功时，应用B事件驱动表中的OnActiveEvent事件将被触发，并传递参数ON\_CALLFUNC指针；
* 如果应用B此时不在待机状态，可能会导致调用不成功，此时通过函数返回值可以得知失败原因；
* 如果应用A是背景应用导致应用B无法切换为当前应用，则不会触发应用B的CALLFUNC事件；

### 定时器

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Timer Operation \*/

/\* 定时器功能处理 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MAX\_APP\_TIMER\_NUM 10 //提供给应用使用的定时器最大个数

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: ApplMngr\_NewAppTimer.

\* DESCRIPTION: 创建应用定时器，在指定时间间隔触发定时器事件后被清除

\* PARAMETER:

IN:

\* sec -- 定时器时间，单位second。0无效,非0则设定定时器。

pTimerFunc -- 定时器触发函数

\* NOTES:

1.一个应用创建的定时器个数有限，最大MAX\_APP\_TIMER\_NUM;

2.当终端休眠时，定时器事件不会被触发；

但是当终端一旦被唤醒，所有到期的定时器事件都会被触发

\* RETURN: < 0 超过定时器限制个数或定时器时间无效

> 0 定时器ID

\* ------------------------------------------------------------------------ \*/

extern int ApplMngr\_NewAppTimer(unsigned long sec,ApplMngr\_CallBack pTimerFunc);

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: ApplMngr\_ClearAppTimer.

\* DESCRIPTION: 清除指定的应用定时器

\* PARAMETER:

\* NOTES:

\* RETURN:

\* ------------------------------------------------------------------------ \*/

extern void ApplMngr\_ClearAppTimer(int TimerID);

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: ApplMngr\_ClearAllAppTimer.

\* DESCRIPTION: 清除所有应用定时器

\* PARAMETER:

\* NOTES:

\* RETURN:

\* ------------------------------------------------------------------------ \*/

extern void ApplMngr\_ClearAllAppTimer(void);

**说明：**

* 定时器在应用为背景状态时仍起作用；
* 定时器不是循环定时器，只会触发一次指定的事件；
* 终端休眠时，定时器不会被触发；

### 广播通知

/\* --------------------------------------------------------------------------

\* FUNCTION NAME: ApplMngr\_SendBroadCast.

\* DESCRIPTION: 向所有应用发送广播事件

\* PARAMETER:

IN:

inMsgBuf -- 传递过去的数据

inMsgLen -- 传递过去的数据长度

\* NOTES: 发送对象是所有运行的应用 ，不包括自身

广播事件不保证事件能及时到达所有应用，甚至也不能保证不丢失，

一般只用于通知类消息的传递

\* RETURN: 见返回值定义

\* ------------------------------------------------------------------------ \*/

extern int ApplMngr\_SendBroadCast(void \*inMsgBuf,int inMsgLen);

**说明：**

* 广播通知事件的发起者不会收到广播事件的触发；
* 广播通知事件只适用于非实时性要求的通知消息；

# 写一个DEMO

### 写一个主控应用MAINAPP

* 1. 实现显示应用列表的功能；
  2. 实现切换到子应用的功能；

### 写一个子应用DEMOA

* 1. 实现应用注册的功能；
  2. 实现应用切换回主控的功能；

### 写一个子应用DEMOB

* 1. 实现应用注册的功能；
  2. 实现应用切换回主控的功能；
  3. 实现CALLFUNC到DEMOA的功能；

### 写一个子应用DEMOC

* 1. 实现应用注册的功能；
  2. 实现应用切换回主控的功能；
  3. 实现CALLFUNC到DEMOB再到DEMOA的功能；
  4. 实现使用定时器发送广播消息的功能；

### 写一个子应用DEMOD

* 1. 实现应用隐藏的功能；

# 编程要点

### 初始化ApplMngr\_AppInit的位置

ApplMngr\_AppInit内部对设备进行了处理，只有做完ApplMngr\_AppInit后，才能进行显示。

### 子应用注册ApplMngr\_SubAppRegister的时机

主应用在运行起子应用后，会等待一段时间（约2-3秒）接收子应用的注册信息。如果子应用在启动后没有及时发起ApplMngr\_SubAppRegister，将导致主应用无法及时接收到子应用注册信息。所以建议在做完ApplMngr\_AppInit后立即做ApplMngr\_SubAppRegister的操作。

### 没有默认应用？

多应用架构只提供应用的框架，对于默认应用等概念的处理由开发人员在编写主控程序时按照实际需求处理。

### 事件驱动的特点

事件驱动是由应用引擎检测并触发，应用通过回调函数进行处理的机制。应用在回调函数内部不能阻塞循环，应该在处理完毕后，立刻退出回调函数，这样才能回退到应用引擎中，从而继续检测下一事件。

### 键盘缓冲和磁道缓冲的异常处理

为避免不期望的数据出现，在处理事件时，如果发生了按键或刷卡动作，在此事件处理完成后会被清除，不会被带入到下一个事件触发处理函数中。

### MIS-POS模式的实现

MIS-POS模式一般是串口监听MIS请求数据，收到请求数据后进行处理。在此架构中，可以在IdleLoop\_Scan事件中检测串口，如果有串口数据到达，则进入下一步处理过程，如果没有，则退出IdleLoop\_Scan函数，等待1秒后应用引擎再次调用IdleLoop\_Scan检测串口。

### 隐身应用的使用

有一种子应用需要被运行，但是不需要被显示在主控的应用列表上，子应用可以将其应用信息结构中的HiddenFlag设置为1。但是除了主控应用的应用列表中没有外，其他功能如同未隐藏的子应用。

### 定时器的使用及循环定时器的用法

定时器指定在多久后执行相应处理函数。执行完此函数后，定时器就是失效了。如果需要循环定时器，则在处理函数中新建一个定时器，执行相同的处理函数即可。

### 应用之间传递的数据最大能支持多长？

多应用之间交互数据是使用管道进行，在ApplMngr库中，定义的管道数据最大长度为2048BYTE，因此在应用调用等涉及到应用之间交互数据的处理中，可支持的数据长度最大为2048BYTE。

### 休眠模式

休眠模式只在移动POS（如：Vx670/Vx680）上，且使用电池时起作用。当所有应用都不再动作约60秒后，系统将自动进入休眠模式，此时设备会关闭部分设备以节约电力。当终端进入休眠模式后，IdleLoop\_Scan、Redraw\_Screen和定时器事件将不再被调用，如果在Redraw\_Screen中进行了时钟显示操作，时钟也将不会再被更新；但是此时可以通过刷卡、插卡、按键唤醒终端。

### 广播事件的使用

广播事件是通知类消息，用于没有实时性要求的信息告知。当一个广播消息到达时，如果应用在处理其他事物，广播消息会一直等待到应用退回到待机状态后触发。

如果需要一个实时性和有效的面向所有子应用的处理功能，最好的方法是主控通过CALLFUNC进行遍历来实现。

### 多应用架构库的跟踪调试

系统使用另外一个AppLog库，用于输出调试日志。使用方法如下：

* 1. 下载D\_APPLOG\_01.00.00\_15.ZIP到终端，并解压；
  2. 设置G15中#LOGPORT=COM1(如果使用其他COM口，请修改)；
  3. 如果只需要跟踪某几个应用的LOG，则同时在G15中设置#LOGAPP=APPNAME1,APPNAME2,… (如只需要跟踪主应用MAINAPP的日志，则设置#LOGAPP=MAINAPP；如果是需要同时跟踪MAINAPP和DEMOA的日志，则设置#LOGAPP=MAINAPP,DEMOA)；如果需要跟踪所有应用日志，则无需设置#LOGAPP或设置#LOGAPP=\* 即可。

更加详细的情况请参考《多应用LOG日志编程指南.docx》。

# 常见问题及解答

### 应用切换失败，总回到当前应用

可能原因：

* 1. 切入应用不存在，或者没有运行起来；
  2. 切入应用没有在退回到待机状态，导致没有响应切换请求；

### CALLFUNC调用失败

可能原因：

* 1. 被调用应用不存在，或者没有运行起来；
  2. 被调用应用没有在退回到待机状态，导致没有响应调用请求；
  3. 被调用应用没有取得当前应用状态，导致调用失败；

### 多个子应用中，只有部分子应用被运行

可能原因：

* 1. 检查没有被运行的子应用是否符合是RVDS 4.0编译的；
  2. 检查子应用及主应用的Stack、Heap或Data段使用累计是否已经超过了RAM内存大小；

### 多个子应用中，只看到部分子应用注册信息

可能原因：

* 1. 检查是否所有子应用都已经被运行起来；
  2. 检查没有注册成功子应用是否争取得到主应用名称；
  3. 检查没有注册成功子应用执行ApplMngr\_SubAppRegister前的处理是否耗时过长；

### Vx670/680待机状态无法进入休眠

可能原因：

* 1. 检查终端是否插电或放在底座上带电使用；
  2. 检查是否有应用（包括背景应用）处于函数内部死循环中，没有退回到待机状态；