|  |
| --- |
| OpenCPU |
| SDK OS API 说明 |
| 版本1.3日期2025-03-28 |

版权声明

版权所有 © 深圳市有方科技股份有限公司2025。深圳市有方科技股份有限公司保留所有权利。

未经深圳市有方科技股份有限公司书面同意，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

是深圳市有方科技股份有限公司所有商标。

本文档中出现的其他商标，由商标所有者所有。

说明

本文档的使用对象为系统工程师，开发工程师及测试工程师。

本设计指南为客户产品设计提供支持，客户须按照本文中的规范和参数进行产品设计和调试。如因客户操作不当造成的人身伤害和财产损失，有方概不承担责任。

由于产品版本升级或其它原因，本文档内容会在不预先通知的情况下进行必要的更新。

除非另有约定，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市有方科技股份有限公司为客户提供全方位的技术支持，任何垂询请直接联系您的客户经理或发送邮件至以下邮箱：

Sales@neoway.com

Support@neoway.com

公司网址：http://www.neoway.com

目录

[关于本文档 viii](#_Toc194070240)

[范围 viii](#_Toc194070241)

[读者对象 viii](#_Toc194070242)

[修订记录 viii](#_Toc194070243)

[符号约定 ix](#_Toc194070244)

[1 概述 10](#_Toc194070245)

[2 数据结构 11](#_Toc194070246)

[2.1 DM 11](#_Toc194070247)

[2.2 OS 11](#_Toc194070248)

[2.3 File 12](#_Toc194070249)

[2.4 Socket 12](#_Toc194070250)

[2.5 Virtual AT 12](#_Toc194070251)

[3 函数 13](#_Toc194070252)

[3.1 设备信息 13](#_Toc194070253)

[3.1.1 nwy\_dm\_dev\_model\_get 13](#_Toc194070254)

[3.1.2 nwy\_dm\_inner\_version\_get 13](#_Toc194070255)

[3.1.3 nwy\_dm\_imei\_get 13](#_Toc194070256)

[3.1.4 nwy\_dm\_hw\_version\_get 13](#_Toc194070257)

[3.1.5 nwy\_dm\_app\_version\_set 14](#_Toc194070258)

[3.1.6 nwy\_dm\_sn\_get 14](#_Toc194070259)

[3.1.7 nwy\_dm\_rftemperature\_get 14](#_Toc194070260)

[3.1.8 nwy\_dm\_heapinfo 14](#_Toc194070261)

[3.2 消息队列 15](#_Toc194070262)

[3.2.1 nwy\_msg\_queue\_create 15](#_Toc194070263)

[3.2.2 nwy\_msg\_queue\_delete 15](#_Toc194070264)

[3.2.3 nwy\_msg\_queue\_send 15](#_Toc194070265)

[3.2.4 nwy\_msg\_queue\_recv 16](#_Toc194070266)

[3.2.5 nwy\_msg\_queue\_pending\_num\_get 16](#_Toc194070267)

[3.2.6 nwy\_msg\_queue\_space\_num\_get 16](#_Toc194070268)

[3.3 文件操作 16](#_Toc194070269)

[3.3.1 nwy\_file\_open 16](#_Toc194070270)

[3.3.2 nwy\_file\_close 17](#_Toc194070271)

[3.3.3 nwy\_file\_read 17](#_Toc194070272)

[3.3.4 nwy\_file\_write 17](#_Toc194070273)

[3.3.5 nwy\_file\_seek 17](#_Toc194070274)

[3.3.6 nwy\_file\_path\_size 18](#_Toc194070275)

[3.3.7 nwy\_file\_fd\_size 18](#_Toc194070276)

[3.3.8 nwy\_file\_exist 18](#_Toc194070277)

[3.3.9 nwy\_file\_remove 18](#_Toc194070278)

[3.3.10 nwy\_file\_rename 18](#_Toc194070279)

[3.3.11 nwy\_file\_sync 18](#_Toc194070280)

[3.3.12 nwy\_file\_fd\_trunc 19](#_Toc194070281)

[3.3.13 nwy\_file\_path\_trunc 19](#_Toc194070282)

[3.3.14 nwy\_file\_path\_stat\_get 19](#_Toc194070283)

[3.3.15 nwy\_file\_fd\_stat\_get 19](#_Toc194070284)

[3.3.16 nwy\_dir\_mk 19](#_Toc194070285)

[3.3.17 nwy\_dir\_rm 20](#_Toc194070286)

[3.3.18 nwy\_dir\_recursive\_rm 20](#_Toc194070287)

[3.3.19 nwy\_dir\_open 20](#_Toc194070288)

[3.3.20 nwy\_dir\_close 20](#_Toc194070289)

[3.3.21 nwy\_dir\_read 20](#_Toc194070290)

[3.3.22 nwy\_dir\_tell 20](#_Toc194070291)

[3.3.23 nwy\_dir\_seek 21](#_Toc194070292)

[3.3.24 nwy\_dir\_rewind 21](#_Toc194070293)

[3.3.25 nwy\_vfs\_free\_size\_get 21](#_Toc194070294)

[3.3.26 nwy\_vfs\_total\_size\_get 21](#_Toc194070295)

[3.3.27 nwy\_vfs\_logical\_block\_device\_create 22](#_Toc194070296)

[3.3.28 nwy\_vfs\_mount 22](#_Toc194070297)

[3.3.29 nwy\_vfs\_unmount 22](#_Toc194070298)

[3.3.30 nwy\_vfs\_mkfs 22](#_Toc194070299)

[3.4 线程 23](#_Toc194070300)

[3.4.1 nwy\_sdk\_log\_printf 23](#_Toc194070301)

[3.4.2 nwy\_thread\_create 23](#_Toc194070302)

[3.4.3 nwy\_thread\_get\_current 23](#_Toc194070303)

[3.4.4 nwy\_thread\_get\_priority 23](#_Toc194070304)

[3.4.5 nwy\_thread\_set\_priority 24](#_Toc194070305)

[3.4.6 nwy\_thread\_suspend 24](#_Toc194070306)

[3.4.7 nwy\_thread\_resume 24](#_Toc194070307)

[3.4.8 nwy\_thread\_exit 24](#_Toc194070308)

[3.4.9 nwy\_thread\_sleep 24](#_Toc194070309)

[3.4.10 nwy\_thread\_usleep 25](#_Toc194070310)

[3.4.11 nwy\_thread\_event\_send 25](#_Toc194070311)

[3.4.12 nwy\_thread\_event\_front\_send 25](#_Toc194070312)

[3.4.13 nwy\_thread\_event\_wait 25](#_Toc194070313)

[3.4.14 nwy\_thread\_pendingevt\_num\_get 26](#_Toc194070314)

[3.4.15 nwy\_thread\_spaceevt\_cnt\_get 26](#_Toc194070315)

[3.4.16 nwy\_thread\_info\_get 26](#_Toc194070316)

[3.4.17 nwy\_thread\_list\_get 26](#_Toc194070317)

[3.4.18 nwy\_thread\_runtime\_stats\_get 26](#_Toc194070318)

[3.5 同步锁 27](#_Toc194070319)

[3.5.1 nwy\_sdk\_mutex\_create 27](#_Toc194070320)

[3.5.2 nwy\_sdk\_mutex\_lock 27](#_Toc194070321)

[3.5.3 nwy\_sdk\_mutex\_unlock 27](#_Toc194070322)

[3.5.4 nwy\_sdk\_mutex\_delete 27](#_Toc194070323)

[3.6 信号量 27](#_Toc194070324)

[3.6.1 nwy\_semaphore\_create 28](#_Toc194070325)

[3.6.2 nwy\_semaphore\_acquire 28](#_Toc194070326)

[3.6.3 nwy\_semahpore\_release 28](#_Toc194070327)

[3.6.4 nwy\_semahpore\_delete 28](#_Toc194070328)

[3.6.5 nwy\_semahpore\_count\_get 28](#_Toc194070329)

[3.7 日期时间 29](#_Toc194070330)

[3.7.1 nwy\_date\_set 29](#_Toc194070331)

[3.7.2 nwy\_date\_get 29](#_Toc194070332)

[3.7.3 nwy\_uptime\_get 29](#_Toc194070333)

[3.7.4 nwy\_sys\_timestamp\_set 29](#_Toc194070334)

[3.7.5 nwy\_sys\_timestamp\_get 30](#_Toc194070335)

[3.7.6 nwy\_sys\_timezone\_set 30](#_Toc194070336)

[3.7.7 nwy\_date\_to\_timestamp 30](#_Toc194070337)

[3.8 定时器 31](#_Toc194070338)

[3.8.1 nwy\_sdk\_timer\_create 31](#_Toc194070339)

[3.8.2 nwy\_sdk\_timer\_destory 31](#_Toc194070340)

[3.8.3 nwy\_sdk\_timer\_start 31](#_Toc194070341)

[3.8.4 nwy\_sdk\_timer\_stop 31](#_Toc194070342)

[3.9 日志 31](#_Toc194070343)

[3.10 Socket 32](#_Toc194070344)

[3.10.1 nwy\_socket\_errno 32](#_Toc194070345)

[3.10.2 nwy\_socket\_open 32](#_Toc194070346)

[3.10.3 nwy\_socket\_send 32](#_Toc194070347)

[3.10.4 nwy\_socket\_recv 33](#_Toc194070348)

[3.10.5 nwy\_socket\_ sendto 33](#_Toc194070349)

[3.10.6 nwy\_socket\_recvfrom 33](#_Toc194070350)

[3.10.7 nwy\_socket\_setsockopt 33](#_Toc194070351)

[3.10.8 nwy\_socket\_getsockopt 34](#_Toc194070352)

[3.10.9 nwy\_socket\_gethost\_by\_name 34](#_Toc194070353)

[3.10.10 nwy\_socket\_close 34](#_Toc194070354)

[3.10.11 nwy\_socket\_connect 34](#_Toc194070355)

[3.10.12 nwy\_socket\_bind 35](#_Toc194070356)

[3.10.13 nwy\_socket\_listen 35](#_Toc194070357)

[3.10.14 nwy\_socket\_accept 35](#_Toc194070358)

[3.10.15 nwy\_socket\_select 35](#_Toc194070359)

[3.10.16 nwy\_socket\_shutdown 36](#_Toc194070360)

[3.10.17 nwy\_socket\_get\_state 36](#_Toc194070361)

[3.10.18 nwy\_socket\_lport\_bind 36](#_Toc194070362)

[3.10.19 nwy\_socket\_ set\_nonblock 36](#_Toc194070363)

[3.10.20 nwy\_socket\_ set\_ reuseaddr 37](#_Toc194070364)

[3.10.21 nwy\_socket\_ set\_ nodelay 37](#_Toc194070365)

[3.10.22 nwy\_socket\_ set\_ keepalive 37](#_Toc194070366)

[3.10.23 nwy\_socket\_get\_keepalive 37](#_Toc194070367)

[3.10.24 nwy\_socket\_ set \_linger 37](#_Toc194070368)

[3.10.25 nwy\_socket\_ get\_ack 38](#_Toc194070369)

[3.10.26 nwy\_socket\_ get\_sent 38](#_Toc194070370)

[3.10.27 nwy\_socket\_ inet\_ntop 38](#_Toc194070371)

[3.10.28 nwy\_socket\_ inet\_pton 38](#_Toc194070372)

[3.10.29 nwy\_socket\_htons 38](#_Toc194070373)

[3.10.30 nwy\_socket\_ntohs 39](#_Toc194070374)

[3.10.31 nwy\_socket\_htonl 39](#_Toc194070375)

[3.10.32 nwy\_socket\_ntonl 39](#_Toc194070376)

[3.11 Virtual AT 39](#_Toc194070377)

[3.11.1 nwy\_virt\_at\_parameter\_init 39](#_Toc194070378)

[3.11.2 nwy\_virt\_at\_cmd\_send 39](#_Toc194070379)

[3.11.3 nwy\_virt\_at\_unsolicited\_cb\_reg 40](#_Toc194070380)

[3.11.4 nwy\_virt\_at\_get\_original\_rsp\_cb 40](#_Toc194070381)

[3.11.5 nwy\_virt\_at\_forward\_cb 40](#_Toc194070382)

[3.11.6 nwy\_virt\_at\_forward\_send 40](#_Toc194070383)

关于本文档

范围

本文档对应产品为N706B系列（RTOS系统）。

读者对象

本文档的使用对象为系统工程师，开发工程师及测试工程师。

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 变更 | 作者 |
| 1.0 | 2025-01 | 初始版本 | Zhang Yanhong |
| 1.1 | 2025-02 | * 新增了nwy\_vfs\_total\_size\_get * 新增了nwy\_thread\_event\_front\_send * 新增了nwy\_date\_to\_timestamp * 删除了nwy\_socket\_fcntl * 新增了nwy\_socket\_get\_keepalive * 新增了nwy\_socket\_htons * 新增了nwy\_socket\_ntohs * 新增了nwy\_socket\_htonl * 新增了nwy\_socket\_ntonl * 修改部分API注意事项 * 修改了数据结构 | Hu Jun |
| 1.2 | 2025-03 | * 更新3.2.5 nwy\_msg\_queue\_get\_pending\_num * 更新3.2.6 nwy\_msg\_queue\_get\_space\_num | Hu Jun |
| 1.3 | 2025-03 | 新增文件操作API：  nwy\_vfs\_logical\_block\_device\_create  nwy\_vfs\_mount  nwy\_vfs\_unmount  nwy\_vfs\_mkfs | Hu Jun |

符号约定

|  |  |
| --- | --- |
| 符号 | 含义 |
|  | 危险或警告，用户必须遵从的规则，否则会造成模块或客户设备不可逆的故障损坏，甚至可能造成人员身体伤害。 |
|  | 注意，警示用户使用模块时应该特别注意的地方，如不遵从，模块或客户设备可能出现故障。 |
|  | 说明或提示，提供模块使用的意见或建议。 |

# 概述

本文档介绍了OpenCPU SDK中系统操作相关的API，主要包括了以下几个方面：

* 设备信息

主要用于获取模块的型号, 版本, imei号以及其他一些系统资源等信息.

* 消息队列

主要用于消息队列创建, 消息发送接收.

* 文件操作

主要定义了文件, 以及路径相关的操作, 如打开,关闭,读,写等等.

* 线程操作

主要用于线程的创建, 状态机,以及线程事件处理.

* 同步锁

主要用于同步锁的创建,加锁,解锁,删除锁.

* 信号量

主要用于信号量的创建,获取,释放等.

* 定时器

主要用于定时器创建,启动,停止,销毁.

* 日期时间

主要用于获取系统时间

* 系统日志

主要用来提供LOG输出

# 数据结构

## DM

详见nwy\_dm\_api.h

## OS

（1）event事件结构体

typedef struct nwy\_event\_msg\_t

{

uint32 id; ///< event identifier

uint32 param1; ///< 1st parameter

uint32 param2; ///< 2nd parameter

uint32 param3; ///< 3rd parameter

} nwy\_event\_msg\_t;

（2）timer相关结构体

typedef void (\*nwy\_timer\_cb\_func)(void \* ctx);//timer回调函数

typedef void \* nwy\_timer\_cb\_para\_t;

typedef struct nwy\_timer\_para\_t

{

int expired\_time; //timer定时器执行时间

nwy\_timer\_type\_e type; //timer类型，0是单次执行，1是循环执行

nwy\_osi\_thread\_t thread\_hdl; //timer处理的对应thread的句柄

nwy\_timer\_cb\_func cb; //timer回调函数

nwy\_timer\_cb\_para\_t cb\_para; //timer回调函数参数

}nwy\_timer\_para\_t;

（3）thread优先级定义

typedef enum nwy\_thread\_priority\_e

{

NWY\_OSI\_PRIORITY\_IDLE = 1, // reserved

NWY\_OSI\_PRIORITY\_LOW = 8,

NWY\_OSI\_PRIORITY\_BELOW\_NORMAL = 16,

NWY\_OSI\_PRIORITY\_NORMAL = 24,

NWY\_OSI\_PRIORITY\_ABOVE\_NORMAL = 32,

NWY\_OSI\_PRIORITY\_HIGH = 40,

NWY\_OSI\_PRIORITY\_REALTIME = 48,

NWY\_OSI\_PRIORITY\_HISR = 56, // reserved

} nwy\_thread\_priority\_e;

其他定义详见nwy\_osi\_api.h

## File

详见nwy\_file\_api.h及nwy\_adapt\_platform.h

## Socket

#define NWY\_DEFAULT\_NETIF\_ID (1) //默认cid

typedef enum {

NWY\_UNKNOWN = -1,

NWY\_CLOSED = 0,

NWY\_LISTEN = 1,

NWY\_SYN\_SENT = 2,

NWY\_SYN\_RCVD = 3,

NWY\_ESTABLISHED = 4,

NWY\_FIN\_WAIT\_1 = 5,

NWY\_FIN\_WAIT\_2 = 6,

NWY\_CLOSE\_WAIT = 7,

NWY\_CLOSING = 8,

NWY\_LAST\_ACK = 9,

NWY\_TIME\_WAIT = 10

}nwy\_tcp\_state\_e;

其他详见nwy\_socket\_api.h及nwy\_adapt\_platform.h

## Virtual AT

#define NWY\_AT\_BUF\_MAX 2048

#define NWY\_AT\_TIMEOUT\_MAX (30)//30S

#define NWY\_AT\_TIMEOUT\_MIN (0)//0S

#define NWY\_AT\_TIMEOUT\_DEFAULT (30)//30S

#define NWY\_AT\_UNISOLICITE\_REG\_MAX (50) //最大注册的主动上报at的数目

typedef int (\*nwy\_at\_unsolicite\_handle)(uint8 \*data\_str, int len); //主动上报at的回调处理函数

typedef struct nwy\_at\_info

{

char at\_command[NWY\_AT\_BUF\_MAX + 1]; //at命令字符串

char ok\_fmt[64];//at回码的ok格式

char err\_fmt[64]; ;//at回码的error格式

int ok\_flag;

int length;

}nwy\_at\_info\_t;

typedef struct nwy\_at\_unsolicite\_table

{

char at\_prefix[32]; //主动上报at前缀

nwy\_at\_unsolicite\_handle nwy\_at\_unsolicite\_cb; //主动上报at处理函数

}nwy\_at\_unsolicite\_table\_t;

# 函数

## 设备信息

本章节提供设备相关的操作接口。

此接口函数定义位于nwy\_common.h及nwy\_dm\_api.h，用于进行设备相关操作。

### nwy\_dm\_dev\_model\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_dev\_model\_get(char \*model\_buf, int buf\_len); |
| 描述 | 获取设备信息。 |
| 参数说明 | model\_buf：设备信息(如:N723)  buf\_len：长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dm\_inner\_version\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_inner\_version\_get(char \*version\_buf, int buf\_len); |
| 描述 | 查询版本号。 |
| 参数说明 | version\_buf：底版本号  buf\_len：长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dm\_imei\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_imei\_get(nwy\_dev\_simid\_e simid, char\* imei\_buf, int buf\_len); |
| 描述 | 获取设备的国际移动设备识别码（IMEI）。 |
| 参数说明 | simid: sim卡id值, 参考nwy\_dev\_simid\_e定义  imei\_buf：用于返回设备IMEI的buffer，长度保证超过16字节。  buf\_len：imei\_buf的长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dm\_hw\_version\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_hw\_version\_get(char \*version\_buf, int buf\_len); |
| 描述 | 获取硬件版本信息。 |
| 参数说明 | version\_buf：用于返回设备硬件版本号的buffer，长度保证超过16字节。  buf\_len：version\_buf的长度。 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dm\_app\_version\_set

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_app\_version\_set(char \* version\_buf, int buf\_len); |
| 描述 | 设置客户APP版本信息, 设置成功后，可以通过AT+NAPPCHECK?查询客户APP版本号。 |
| 参数说明 | version\_buf：设置客户APP版本号内容，长度不超过128字节。  buf\_len：version\_buf的长度。 |
| 返回值 | NA |

### nwy\_dm\_sn\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_sn\_get(char \*sn, int len); |
| 描述 | 获取产品的SN号. |
| 参数说明 | sn, 输出产品的序列字符串 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dm\_rftemperature\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_rftemperature\_get(float \*outvalue) |
| 描述 | 获取设备射频温度 |
| 参数说明 | outvalue: 输出获取的射频温度值 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dm\_heapinfo

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dm\_heapinfo(nwy\_heapinfo\_t \* heapinfo); |
| 描述 | 动态RAM剩余内获取 |
| 参数说明 | typedef struct  {  uint32 start;  uint32 size;  uint32 avail\_size;  uint32 max\_block\_size;  }nwy\_heapinfo\_t;  start:堆起始地址  size:总堆大小  available\_ram：ram剩余可用大小  max\_block\_size：RAM的最大可分配大小 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

## 消息队列

此接口函数定义位于nwy\_osi\_api.h，用于进行消息队列相关操作。

### nwy\_msg\_queue\_create

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_msg\_queue\_create (nwy\_osi\_msg\_queue\_t \*\*msg\_q, uint32 msg\_size, uint32 msg\_num) |
| 描述 | 初始化消息队列。 |
| 参数说明 | msg\_num：消息队最大可以容纳的消息数目  msg\_size：消息内容大小  msg\_q: 创建的消息队列，指针的指针 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他枚举值 |

### nwy\_msg\_queue\_delete

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_msg\_queue\_delete (nwy\_osi\_msg\_queue\_t \*msg\_q) |
| 描述 | 删除消息队列。 |
| 参数说明 | msq\_q：消息队列 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他枚举值 |
| 说明 | 客户在使用消息队列完毕后，必须删除消息队列，否则会发生内存泄漏。 |

### nwy\_msg\_queue\_send

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_msg\_queue\_send(nwy\_osi\_msg\_queue\_t \*msg\_q, uint32 size, void\* msg\_ptr, int timeout) |
| 描述 | 发送消息。 |
| 参数说明 | msg\_q：消息队列  size:消息长度  \*msg\_ptr：消息内容  timeout：如果msgqueue满时, timeout设置 0: no block; 1: wait for ever; 其他: wait for time ms |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他枚举值 |

### nwy\_msg\_queue\_recv

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_msg\_queue\_recv(nwy\_osi\_msg\_queue\_t \*msg\_q, uint32 size, void\* msg\_ptr, int timeout) |
| 描述 | 接收消息。 |
| 参数说明 | msg\_q：消息队列  size:消息长度  \*msg\_ptr：消息内容  timeout：如果msgqueue空时, timeout设置 0: no block; 1: wait for ever; 其他: wait for time ms |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他枚举值 |

### nwy\_msg\_queue\_pending\_num\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_msg\_queue\_pending\_num\_get(nwy\_osi\_msg\_queue\_t msg\_q, uint32 \*pnum); |
| 描述 | 获取队列中待处理的消息数目。 |
| 参数说明 | msg\_q：消息队列  pnum:返回消息数目 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他枚举值 |

### nwy\_msg\_queue\_space\_num\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_msg\_queue\_space\_num\_get(nwy\_osi\_msg\_queue\_t msg\_q, uint32 \*pnum); |
| 描述 | 获取队列中还可以存放msg的个数 |
| 参数说明 | msg\_q：消息队列  pnum:返回消息队列space个数 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他枚举值 |

## 文件操作

此接口函数定义位于nwy\_file.h，用于进行消息队列相关操作。

### nwy\_file\_open

|  |  |
| --- | --- |
| 格式 | int nwy\_file\_open(const char \*path, nwy\_file\_openmode\_e mode); |
| 描述 | 打开文件。 |
| 参数说明 | \*path：文件全路径名  mode：打开文件的方式，参考nwy\_file\_open\_mode\_e |
| 返回值 | 成功：返回文件描述符  失败：小于0 |

### nwy\_file\_close

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_file\_close(int fd); |
| 描述 | 关闭文件。 |
| 参数说明 | fd：文件描述符 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_read

|  |  |
| --- | --- |
| 格式 | int nwy\_file\_read(int fd, void \*dst, int size); |
| 描述 | 读文件。 |
| 参数说明 | fd：文件描述符  \*dst：读取数据buffer  Size：字节数 |
| 返回值 | >0： 成功读取的字节数  =0：到达文件尾  <0：发生错误 |

### nwy\_file\_write

|  |  |
| --- | --- |
| 格式 | int nwy\_file\_write(int fd, const void \*data, int size); |
| 描述 | 写文件。 |
| 参数说明 | fd：文件描述符  \* data：写入数据  Size：字节数 |
| 返回值 | 成功：写入的字节数  失败：其他 |

### nwy\_file\_seek

|  |  |
| --- | --- |
| 格式 | int nwy\_file\_seek(int fd, int offset, nwy\_fseek\_offset\_e mode) |
| 描述 | 设置文件指针。 |
| 参数说明 | fd：文件描述符  offset：偏移量  mode：从哪里开始偏移，参见nwy\_fseek\_offset\_e |
| 返回值 | 成功：偏移量  失败：其他 |

### nwy\_file\_path\_size

|  |  |
| --- | --- |
| 格式 | long nwy\_file\_path\_size(const char \*path); |
| 描述 | 获取文件大小。 |
| 参数说明 | path：文件名 |
| 返回值 | 成功：文件大小字节数  失败：-1 |

### nwy\_file\_fd\_size

|  |  |
| --- | --- |
| 格式 | long nwy\_file\_fd\_size(int fd); |
| 描述 | 获取文件大小。 |
| 参数说明 | fd:文件描述符 |
| 返回值 | 成功：文件大小字节数  失败：-1 |

### nwy\_file\_exist

|  |  |
| --- | --- |
| 格式 | bool nwy\_file\_exist(const char \*path); |
| 描述 | 判断文件是否存在。 |
| 参数说明 | path：文件路径 |
| 返回值 | 存在：true  不存在：false |

### nwy\_file\_remove

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_file\_remove(const char\* path); |
| 描述 | 删除文件。 |
| 参数说明 | path：文件路径 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_rename

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_file\_rename(const char \*oldpath, const char \*newpath); |
| 描述 | 重命名文件。 |
| 参数说明 | oldpath：旧的文件名  newpath：新文件名 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_sync

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_file\_sync(int fd); |
| 描述 | 同步文件 |
| 参数说明 | fd：文件描述符 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_fd\_trunc

|  |  |
| --- | --- |
| 格式 | int nwy\_file\_fd\_trunc(int fd, long len); |
| 描述 | 文件截取 |
| 参数说明 | fd：文件描述符  len：截取长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_path\_trunc

|  |  |
| --- | --- |
| 格式 | int nwy\_file\_path\_trunc(const char \*path, long len); |
| 描述 | 文件截取 |
| 参数说明 | path：文件名  len：截取长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_path\_stat\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_file\_path\_stat\_get(const char \*path, nwy\_file\_stat \*st); |
| 描述 | 获取文件status |
| 参数说明 | path：文件名  st：文件信息，详见stat.h |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_file\_fd\_stat\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_file\_fd\_stat\_get(int fd, nwy\_file\_stat \*st); |
| 描述 | 获取文件status |
| 参数说明 | fd：文件描述符  st：文件信息，详见stat.h |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dir\_mk

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir\_mk(const char \*path); |
| 描述 | 创建目录 |
| 参数说明 | path：目录路径 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dir\_rm

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir \_rm(const char \*path); |
| 描述 | 删除目录 |
| 参数说明 | path：目录路径 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 无法对非空文件夹进行删除。 |

### nwy\_dir\_recursive\_rm

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir\_recursive\_rm(const char \*path); |
| 描述 | 删除目录 |
| 参数说明 | path：目录路径 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 强制删除，可删除非空文件夹 |

### nwy\_dir\_open

|  |  |
| --- | --- |
| 格式 | nwy\_dir\_info\_t \*nwy\_dir\_open(const char \*name); |
| 描述 | 打开一个目录 |
| 参数说明 | name：目录名 |
| 返回值 | nwy\_dir\_info\_t: 打开的目录指针，如果失败返回NULL |

### nwy\_dir\_close

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir\_close(nwy\_dir\_info\_t \*pdir); |
| 描述 | 关闭打开的目录 |
| 参数说明 | pdir: 要关闭的目录地址 |
| 返回值 | 无 |

### nwy\_dir\_read

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir\_read(nwy\_dir\_info\_t \*pdir, nwy\_dirent\_t \*d\_info) |
| 描述 | 读取目录内容 |
| 参数说明 | pdir: 要读取的目录地址 |
| 返回值 | nwy\_dirent 读取到的目录信息 |

### nwy\_dir\_tell

|  |  |
| --- | --- |
| 格式 | int nwy\_dir\_tell(nwy\_dir\_info\_t \*pdir); |
| 描述 | 获取参数目录流当前的读取位置 |
| 参数说明 | pdir：目录地址  d\_info:读取到的信息 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dir\_seek

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir\_seek(nwy\_dir\_info\_t \*pdir, long loc); |
| 描述 | 用来设置参数pdir 目录流目前的读取位置, 在调用readdir()时便从此新位置开始读取 |
| 参数说明 | pdir：目录地址  loc：距离目录文件开头的偏移量 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_dir\_rewind

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_dir\_rewind(nwy\_dir\_info\_t \*pdir); |
| 描述 | 用来将位置指针移动到文件开头 |
| 参数说明 | pdir：目录地址 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_vfs\_free\_size\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_vfs\_free\_size\_get(unsigned long long \*avail\_space); |
| 描述 | 获取文件系统剩余可用空间 |
| 参数说明 | avail\_space：输出可用空间 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | RDA平台需要知道vfs mount的点，通常会由基线定义 |

### nwy\_vfs\_total\_size\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_vfs\_total\_size\_get(const char\* path,unsigned long long \*total\_space) |
| 描述 | 获取path路径下文件系统大小 |
| 参数说明 | Path：path路径  total\_space：获取的总大小 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 若获取整个文件系统的大小，path传NULL |

### nwy\_vfs\_logical\_block\_device\_create

|  |  |
| --- | --- |
| 格式 | nwy\_block\_device\_t \*nwy\_vfs\_logical\_block\_device\_create(char \*name, uint32\_t offset, uint32\_t size) |
| 描述 | 创建逻辑块设备 |
| 参数说明 | name：物理块设备名  offset：起始  Size ：大小 |
| 返回值 | 成功：逻辑块设备  失败：NULL |

### nwy\_vfs\_mount

|  |  |
| --- | --- |
| 格式 | int nwy\_vfs\_mount(const char \*base\_path, nwy\_block\_device\_t \*nwy\_bdev) |
| 描述 | 块设备挂载 |
| 参数说明 | base\_path：用于文件系统访问的挂载点  nwy\_bdev：块设备 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 入参nwy\_bdev是nwy\_vfs\_logical\_block\_device\_create创建的逻辑块设备 |

### nwy\_vfs\_unmount

|  |  |
| --- | --- |
| 格式 | int nwy\_vfs\_unmount(const char \*base\_path) |
| 描述 | 下载块设备 |
| 参数说明 | base\_path：块设备的挂载点 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_vfs\_mkfs

|  |  |
| --- | --- |
| 格式 | int nwy\_vfs\_mkfs(nwy\_block\_device\_t \*nwy\_bdev) |
| 描述 | 格式化块设备 |
| 参数说明 | nwy\_bdev：块设备 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 格式化后，如需再次使用，需要重新挂载 |

## 线程

此接口函数定义位于nwy\_osi\_api.h，用于进行线程相关操作。

### nwy\_sdk\_log\_printf

|  |  |
| --- | --- |
| 格式 | void nwy\_sdk\_log\_printf(nwy\_log\_level\_e lev, char\* fmt, ...) |
| 描述 | Log打印 |
| 参数说明 | Lev：log级别  Fmt：log打印格式 |
| 返回值 | NA |

### nwy\_thread\_create

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_create(nwy\_osi\_thread\_t \*\*hdl, char \*task\_name, uint8 priority, nwy\_task\_cb\_func cb, void \*argv, uint32 event\_count, uint32 stack\_size, void \*stack) |
| 描述 | 创建一个线程 |
| 参数说明 | hdl:输出创建的线程句柄,指针的指针  task\_name:线程名字;  stack: 指定栈起始地址, 默认为NULL  stack\_size: 栈大小  priority: 线程优先级  cb: 线程执行函数  argv: cb的参数  event\_count: 线程持有event的个数,每个线程都有一个event queue |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_get\_current

|  |  |
| --- | --- |
| 格式 | nwy\_osi\_thread\_t \*nwy\_thread\_get\_current(void); |
| 描述 | 获取当前函数所在的任务。 |
| 参数说明 | NA |
| 返回值 | 成功：当前函数所在任务的句柄  失败：NULL |

### nwy\_thread\_get\_priority

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_get\_priority(nwy\_osi\_thread\_t \*hdl, uint8 \*priority); |
| 描述 | 获取优先级。 |
| 参数说明 | hdl：线程  priority：优先级 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_set\_priority

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_set\_priority(nwy\_osi\_thread\_t \*hdl, uint8 new\_pri); |
| 描述 | 设置线程优先级。 |
| 参数说明 | hdl：线程  new\_pri：新优先级 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_suspend

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_suspend(nwy\_osi\_thread\_t \*hdl); |
| 描述 | 任务挂起。 |
| 参数说明 | hdl：线程 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_resume

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_resume(nwy\_osi\_thread\_t \* hdl); |
| 描述 | 任务唤醒。 |
| 参数说明 | hdl：线程 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_exit

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_exit(nwy\_osi\_thread\_t \*hdl); |
| 描述 | 退出线程。 |
| 参数说明 | hdl：线程 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 在需要退出线程时，必须执行线程退出函数，释放thread资源，对于RDA当hdl为空时，退出当前线程，当hdl为非空时，退出对应线程，退出前将任务suspend。对应ASR（ASR采用threadx）无法在当前线程的上下文中退出当前线程，当hdl为空，则返回参数error，当hdl不为空时，退出对应thead，并且调用该函数退出前要保证对应hdl的线程处于suspend状态，否则退出失败， |

### nwy\_thread\_sleep

|  |  |
| --- | --- |
| 格式 | void nwy\_thread\_sleep(uint32 ms); |
| 描述 | Sleep延时。 |
| 参数说明 | sleep：延时时间，单位ms |
| 返回值 | NA |

### nwy\_thread\_usleep

|  |  |
| --- | --- |
| 格式 | void nwy\_thread\_usleep(uint32 us); |
| 描述 | Sleep延时。 |
| 参数说明 | sleep：延时时间，单位us |
| 返回值 | NA |

### nwy\_thread\_event\_send

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_event\_send(nwy\_osi\_thread\_t \*hdl, nwy\_event\_msg\_t \*event, int timeout) |
| 描述 | 给线程发送一个event |
| 参数说明 | hdl: 线程句柄  event: 要发送的event  timeout: 如果msgqueue满时, timeout设置 0: no block; 1: wait for ever; 其他: wait for  time ms. |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_event\_front\_send

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_event\_front\_send(nwy\_osi\_thread\_t hdl, nwy\_event\_msg\_t \*event, int timeout) |
| 描述 | 给线程发送一个event，将event放入队列头部 |
| 参数说明 | hdl: 线程句柄  event: 事件指针  timeout: 当thread event queue为空, timeout设置 0: no block, 直接返回失败 1: wait for ever 直到有event; 其他: wait for time ms. |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_event\_wait

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_event\_wait(nwy\_osi\_thread\_t \*hdl, nwy\_event\_msg\_t \*event, int timeout) |
| 描述 | 等待线程事件 |
| 参数说明 | hdl: 线程句柄  event: 事件指针  timeout: 当thread event queue为空, timeout设置 0: no block, 直接返回失败 1: wait for ever 直到有event; 其他: wait for time ms. |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_pendingevt\_num\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_pendingevt\_num\_get(nwy\_osi\_thread\_t \*hdl, uint32 \*pnum) |
| 描述 | 获取线程事件队列中待处理的事件数目 |
| 参数说明 | hdl: 线程句柄  pnum: pending event 数目 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_spaceevt\_cnt\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_spaceevt\_cnt\_get(nwy\_osi\_thread\_t \*hdl, uint32 \*pnum) |
| 描述 | 获取线程事件队列中还可以存储的事件数目 |
| 参数说明 | hdl: 线程句柄  pnum: space event 数目 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_info\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_info\_get(nwy\_osi\_thread\_t \*hdl, nwy\_thread\_info\_t \*thread\_info) |
| 描述 | 获取某一个thread 信息 |
| 参数说明 | hdl: 线程句柄  thread\_info: 输出获取到的thread信息 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_thread\_list\_get

|  |  |
| --- | --- |
| 格式 | void nwy\_thread\_list\_get(char \*thread\_list) |
| 描述 | 获取系统任务列表信息 |
| 参数说明 | thread\_list: 输出任务列表信息 |
| 返回值 | NA |

### nwy\_thread\_runtime\_stats\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_thread\_runtime\_stats\_get(char \*stats\_buf) |
| 描述 | 获取系统任务运行时间统计信息 |
| 参数说明 | stats\_buf: 输出任务运行时间信息 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

## 同步锁

此接口函数定义位于nwy\_osi\_api.h，用于进行mutex相关操作。

### nwy\_sdk\_mutex\_create

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_mutex\_create(nwy\_osi\_mutex\_t\*\* mutex) |
| 描述 | 初始化一个互斥锁。 |
| 参数说明 | mutex:创建的互斥锁，指针的指针 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_sdk\_mutex\_lock

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_mutex\_lock(nwy\_osi\_mutex\_t \*mutex, int time\_out); |
| 描述 | 加锁 |
| 参数说明 | mutex：互斥锁  time\_out: 若此时已经被锁, timeout设置为 0: no block; 1: wait for ever; 其他: wait for time ms |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_sdk\_mutex\_unlock

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_mutex\_unlock(nwy\_osi\_mutex\_t \*mutex); |
| 描述 | 解锁 |
| 参数说明 | mutex：互斥锁 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_sdk\_mutex\_delete

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_mutex\_delete(nwy\_osi\_mutex\_t \*mutex); |
| 描述 | 销毁互斥锁。 |
| 参数说明 | mutex：互斥锁 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | Mutex使用完成后,一定要释放掉,否则会资源浪费. |

## 信号量

此接口函数定义位于nwy\_osi\_api.h，用于进行信号量相关操作。

### nwy\_semaphore\_create

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_semaphore\_create(nwy\_osi\_semaphore\_t \*\*sem,uint32 sem\_count) |
| 描述 | 初始化一个信号量。 |
| 参数说明 | sem：信号量句柄，指针的指针  sem\_count：initial count of the semaphore |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_semaphore\_acquire

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_semaphore\_acquire(nwy\_osi\_semaphore\_t \*sem, int time\_out); |
| 描述 | 请求信号量。 |
| 参数说明 | sem：信号量  timeout：超时时间，单位ms（0：不超时） |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_semahpore\_release

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_semahpore\_release(nwy\_osi\_semaphore\_t \*sem); |
| 描述 | 释放信号量。 |
| 参数说明 | sem：信号量 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_semahpore\_delete

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_semahpore\_delete(nwy\_osi\_semaphore\_t \*sem); |
| 描述 | 删除信号量。 |
| 参数说明 | sem：信号量 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_semahpore\_count\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_semahpore\_count\_get(nwy\_osi\_semaphore\_t sem, uint32 \*count) |
| 描述 | 获取信号量个数 |
| 参数说明 | sem：信号量 count：存储获取的信号量数 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

## 日期时间

此接口函数定义位于nwy\_api.h，相关头文件nwy\_common.h。用于进行日期时间的读取和设置。

### nwy\_date\_set

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_date\_set(nwy\_time\_t \*p\_time, int timezone) |
| 描述 | 设置时间接口。 |
| 参数说明 | p\_time：输入时间  timezone：时区信息，时区的单位是十五分钟，相当于现代时间的四分之一小时 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 1.设置year的范围为2000-2100。  2. 时区是指地球上按照经度划分的区域，每个区域都有统一的标准时间。时区的单位是十五分钟，这相当于现代时间的四分之一小时，也就是15分钟。全球被划分为24个时区，每个时区相隔15度经度，相差1小时，这种划分方式有助于不同地区的人按照当地时间进行日常生活和活动 |

### nwy\_date\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_date\_get(nwy\_time\_t \*p\_time, int\* timezone) |
| 描述 | 获取时间接口。 |
| 参数说明 | julian\_time：输出时间信息  timezone：时区信息，时区的单位是十五分钟，相当于现代时间的四分之一小时 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 时区是指地球上按照经度划分的区域，每个区域都有统一的标准时间。时区的单位是十五分钟，这相当于现代时间的四分之一小时，也就是15分钟。全球被划分为24个时区，每个时区相隔15度经度，相差1小时，这种划分方式有助于不同地区的人按照当地时间进行日常生活和活动 |

### nwy\_uptime\_get

|  |  |
| --- | --- |
| 格式 | int64 nwy\_uptime\_get(void) |
| 描述 | 获取最近一次系统启动运行了多长时间 |
| 参数说明 | 无 |
| 返回值 | 返回时间的毫秒数。 |

### nwy\_sys\_timestamp\_set

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sys\_timestamp\_set(nwy\_timeval\_t \*time); |
| 描述 | 设置系统时间戳 |
| 参数说明 | \*time：时间戳（Timestamp）信息，是指从1970年1月1日（UTC时间）开始计算的秒数 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_sys\_timestamp\_get

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sys\_timestamp\_set(nwy\_timeval\_t \*time); |
| 描述 | 获取系统时间戳 |
| 参数说明 | \*time：时间戳（Timestamp）信息，是指从1970年1月1日（UTC时间）开始计算的秒数 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_sys\_timezone\_set

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sys\_timezone\_set(int tz); |
| 描述 | 设置系统时区 |
| 参数说明 | tz：时区信息，时区的单位是十五分钟，相当于现代时间的四分之一小时 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |
| 说明 | 接口nwy\_date\_get和nwy\_date\_set 也支持查询和设置时区信息  时区是指地球上按照经度划分的区域，每个区域都有统一的标准时间。时区的单位是十五分钟，这相当于现代时间的四分之一小时，也就是15分钟。全球被划分为24个时区，每个时区相隔15度经度，相差1小时，这种划分方式有助于不同地区的人按照当地时间进行日常生活和活动  举例：当你调用 nwy\_sys\_timezone\_set(32); 时，意味着你正在尝试将系统的时区设置为32个十五分钟单位的时区。由于每个时区单位是15分钟，32个单位相当于32 \* 15 = 480分钟，也就是8小时。所以这个调用是将系统时区设置为比UTC时间快8小时的时区。 |

### nwy\_date\_to\_timestamp

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_date\_to\_timestamp(nwy\_time\_t \*tm, nwy\_timeval\_t \*timestamp); |
| 描述 | 将时间转换为时间戳 |
| 参数说明 | tm：需要转换的时间  timestamp：获取的时间戳 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

## 定时器

此接口函数定义位于nwy\_osi\_api.h，用于进行定时器相关操作。

### nwy\_sdk\_timer\_create

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_timer\_create(nwy\_osi\_timer\_t \*\*timer, nwy\_timer\_para\_t \*para) |
| 描述 | 创建定时器 |
| 参数说明 | timer: 指向创建则timer，指针的指针;  para：创建timer的参数，详见nwy\_timer\_para\_t结构体 |
| 返回值 | 成功：定时器指针  失败：NULL |
| 说明 |  |

### nwy\_sdk\_timer\_destory

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_timer\_destory(nwy\_osi\_timer\_t \*timer) |
| 描述 | 删除定时器。 |
| 参数说明 | timer：定时器指针 |
| 返回值 | 无 |
| 说明 | 定时器使用完毕后一定要调用，防止内存泄漏。 |

### nwy\_sdk\_timer\_start

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_timer\_start(nwy\_osi\_timer\_t \*timer, nwy\_timer\_para\_t \*para) |
| 描述 | 启动定时器开始计时 |
| 参数说明 | timer：启动的timer指针  para：创建timer的参数，详见nwy\_timer\_para\_t结构体 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_sdk\_timer\_stop

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_timer\_stop(nwy\_osi\_timer\_t \*timer) |
| 描述 | 停止定时器 |
| 参数说明 | timer：定时器 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

## 日志

此接口函数定义位于nwy\_log.h，用于调试日志输出。

/\* LOG LEVERL DEFINITION \*/

#define LOG\_EMERG 0 /\* systemis unusable \*/

#define LOG\_ALERT 1 /\* actionmust bet taken immediately \*/

#define LOG\_CRIT 2 /\* critical conditions \*/

#define LOG\_ERR 3 /\* error conditions \*/

#define LOG\_WARNING 4 /\* warning conditions \*/

#define LOG\_NOTICE 5 /\* normalbut significant \*/

#define LOG\_INFO 6 /\* informational \*/

#define LOG\_DEBUG 7 /\* debug-level messages \*/

#define NWY\_LOG\_OUTPUT\_LEVEL 7

#define NWY\_LOG\_PRINTF(level, tag, fmt, ...) \

do \

{ \

if (NWY\_LOG\_OUTPUT\_LEVEL >= level) \

osiTracePrintf("nwy open ", fmt, ##\_\_VA\_ARGS\_\_); \

} while (0)

#define NWY\_LOGV(level, fmt, args...) NWY\_LOG\_PRINTF(level, fmt, ###args)

#define NWY\_LOGI(fmt, args...) NWY\_LOG\_PRINTF(LOG\_INFO, fmt, ###args)

#define NWY\_LOGE(fmt, args...) NWY\_LOG\_PRINTF(LOG\_ERR, fmt, ###args)

#define NWY\_LOGD(fmt, args...) NWY\_LOG\_PRINTF(LOG\_DEBUG, fmt, ###args) nwy\_open\_sdk\_log

## Socket

### nwy\_socket\_errno

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_errno(void); |
| 描述 | 获取socket 错误码 |
| 参数说明 | NA |
| 返回值 | 返回对应错误码，见errno.h |

### nwy\_socket\_open

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_open(int domain, int type, int protocol, int cid); |
| 描述 | 创建socket |
| 参数说明 | domain：协议族(family), 常用的协议族有，AF\_INET (IPV4)/ AF\_INET6(IPV6) type：套接字类型  protocol：使用的通信协议,IPPROTO\_TCP、IPPROTO\_TCP等  cid:链路cid，1-7，若使用默认链接cid为NWY\_DEFAULT\_NETIF\_ID |
| 返回值 | 返回socket id |

### nwy\_socket\_send

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_send(int socket, const void \*data, size\_t size, int flags); |
| 描述 | Socket发送数据 |
| 参数说明 | socket：socket id data：发送数据  size：发送数据长度  flags：未使用，填0 |
| 返回值 | 返回实际发送数据长度 |

### nwy\_socket\_recv

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_recv(int socket, void \*data, size\_t len, int flags); |
| 描述 | Socket接收数据 |
| 参数说明 | socket：socket id data：接收数据缓存buffer  len：缓存buffer长度  flags：未使用，填0 |
| 返回值 | 返回实际接收数据长度 |

### nwy\_socket\_ sendto

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_sendto(int socket, const void \*data, size\_t size, int flags,const struct sockaddr \*to, socklen\_t tolen); |
| 描述 | Socket向指定地址发送数据 |
| 参数说明 | socket：socket id data：发送的数据  size：发送数据长度  flags：未使用，填0  to：发送目的地  tolen：目的地地址长度 |
| 返回值 | 返回实际发送数据长度 |

### nwy\_socket\_recvfrom

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_recvfrom(int socket, void \*data, size\_t len, int flags,struct sockaddr \*from, socklen\_t \*fromlen); |
| 描述 | Socket从指定地址接收数据 |
| 参数说明 | socket：socket id data：接收数据缓存buffer  len：缓存buffer长度  flags：未使用，填0  from：指定地址  fromlen：指定地址长度 |
| 返回值 | 返回实际接收数据长度 |
| 返回值 | 返回实际接收数据长度 |

### nwy\_socket\_setsockopt

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_setsockopt(int socket, int level, int optname,  const void \* optval, socklen\_t optlen); |
| 描述 | 设置套接字选项值。 |
| 参数说明 | socket: 套接字描述符  level：选项所在协议层, 如SOL\_SOCKET, IPPROTO\_IP, IPPROTO\_TCP  optname：需要访问的选项名  optval：指向包含新选项值的缓冲  optlen：选项长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_getsockopt

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_socket\_getsockopt(int socket, int level, int optname, void \* optval, socklen\_t \* optlen); |
| 描述 | 返回套接字选项值。 |
| 参数说明 | socket: 套接字描述符  level：选项所在协议层  optname：需要访问的选项名  optval：指向返回选项值的缓冲  optlen：选项值最大长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_gethost\_by\_name

|  |  |
| --- | --- |
| 格式 | char\* nwy\_socket\_gethost\_by\_name(const char \*name, int \*isipv6,int cid); |
| 描述 | 根据域名获取主机地址。 |
| 参数说明 | name:域名  isipv6：返回是否为ipv6地址; 1: ipv6, 0:ipv4  cid:链路cid，1-7，若使用默认链接cid为NWY\_DEFAULT\_NETIF\_ID |
| 返回值 | 成功返回host ip地址; 失败返回NULL |

### nwy\_socket\_close

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_close(int socket); |
| 描述 | 关闭套接字。 |
| 参数说明 | socket:套接字描述符 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_connect

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_connect(int socket, const struct sockaddr \*name, socklen\_t namelen); |
| 描述 | 套接字连接服务器 |
| 参数说明 | socket:套接字描述符  name：服务器套结子地址  namelen：服务器套接字信息长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_bind

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_sdk\_socket\_bind(int socket, struct sockaddr \*addr, socklen\_t \*addrlen) |
| 描述 | bind套接字连接。 |
| 参数说明 | socket:套接字描述符  addr：套接字信息  addrlen：长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_listen

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_listen(int socket, int backlog) |
| 描述 | 套接字监听。 |
| 参数说明 | socket:套接字描述符  backlog：监听最大数 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_accept

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_accept (int socket, struct sockaddr \*addr, socklen\_t \*addrlen) |
| 描述 | 接受连接套接字连接。 |
| 参数说明 | socket:套接字描述符  addr：套接字信息  addrlen：长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_select

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_select(int maxfdp1, fd\_set \*readset, fd\_set \*writeset, fd\_set \*exceptset, struct timeval \*timeout) |
| 描述 | 套接字select监听。 |
| 参数说明 | maxfdp1：所有文件描述符的范围，即所有文件描述符的最大值加1  readset：要监视这些文件描述符的读变化的，即我们关心是否可以从这些文件中读取数据了，如果这个集合中有一个文件可读，select就会返回一个大于0的值，表示有文件可读。  writeset：监视这些文件描述符的写变化的，即我们关心是否可以向这些文件中写入数据了，如果这个集合中有一个文件可写，select就会返回一个大于0的值，表示有文件可写。  exceptset:同上面两个参数的意图，用来监视文件错误异常。  timeout：用来指定select函数的超时时间:   * null,将select置于阻塞状态，select会永远等待下去，直到有一个文件描述符就绪，select返回。 * 0秒0毫秒：将select设置为一个纯粹的非阻塞函数，不管文件描述符是否有变化，都立刻返回继续执行，文件无变化返回0，有变化返回一个正值 * 大于0的值：这就是等待的超时时间，即select在timeout时间内阻塞，超时时间之内有相关事件到来，则返回大于0数值，若设置时间超时，select同样返回。 |
| 返回值 | * 小于0：select错误 * 大于0：某些文件可读写。 * 0：等待超时，没有可读写或错误的文件 |
| 说明 | 在使用select机制接收到消息后，需要通过event将消息发送到另一线程（比如线程A）进行具体的消息处理。因为在进行select接收时，当前线程的处理是阻塞的，会影响event的接收，导致消息处理异常。 |

### nwy\_socket\_shutdown

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_shutdown(int socket, int how) |
| 描述 | 关闭socket全双工连接的一端。 |
| 参数说明 | * socket:套结字描述符 * how：SHUT\_RD ,SHUT\_WR, SHUT\_RDWR |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_get\_state

|  |  |
| --- | --- |
| 格式 | nwy\_tcp\_state\_e nwy\_socket\_get\_state(int socket) |
| 描述 | 获取tcp连接状态 |
| 参数说明 | socket:套结字描述符 |
| 返回值 | 状态码 |

### nwy\_socket\_lport\_bind

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_lport\_bind(int socket, uint16\_t lport); |
| 描述 | socket bind指定端口 |
| 参数说明 | * socket：socket id * lport：端口号 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ set\_nonblock

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_set\_nonblock(int socket); |
| 描述 | 设置socket非阻塞模式 |
| 参数说明 | socket：socket id |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ set\_ reuseaddr

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_set\_reuseaddr(int socket); |
| 描述 | 设置socket重复使用地址 |
| 参数说明 | socket：socket id |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ set\_ nodelay

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_set\_nodelay(int socket); |
| 描述 | 设置TCP\_NODELAY选项，该选项是用来控制是否开启Nagle算法 |
| 参数说明 | * socket：socket id |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ set\_ keepalive

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_set\_keepalive(int socket ,uint32\_t mode ,uint32\_t idle ,uint32\_t interval); |
| 描述 | 设置tcp keepalive |
| 参数说明 | * socket：socket id * mode：1打开，0关闭 * idle：通道空闲间隔 * nterval：发包间隔 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_get\_keepalive

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_get\_keepalive(int socket ,uint32\_t \*mode ,uint32\_t \*idle ,uint32\_t \*interval) |
| 描述 | 获取tcp keepalive |
| 参数说明 | * socket：socket id * mode：获取keepalive状态，1打开，0关闭 * idle：通道空闲间隔 * nterval：发包间隔 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ set \_linger

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_set\_linger(int socket); |
| 描述 | 设置socket linger |
| 参数说明 | socket：socket id |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ get\_ack

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_get\_ack(int socket); |
| 描述 | 获取socket收到ack |
| 参数说明 | socket：socket id |
| 返回值 | ack总个数，即实际发送成功的数据数 |

### nwy\_socket\_ get\_sent

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_get\_sent(int socket); |
| 描述 | 获取socket 的send的数据数 |
| 参数说明 | socket：socket id |
| 返回值 | socket sent的总数据 |

### nwy\_socket\_ inet\_ntop

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_inet\_ntop(int family, void \*src, char \*dst, int size); |
| 描述 | 将IPv4或IPv6 Internet网络地址转换为 Internet标准格式的字符串 |
| 参数说明 | * family：协议类型，AF\_INET6和AF\_INET * src：网络地址 * dst：转换后字符串地址 * size：src地址的长度 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_ inet\_pton

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_socket\_inet\_pton(int family, char \*src, void \*dst); |
| 描述 | 将标准文本表示形式的IPv4或IPv6 Internet网络地址转换为数字二进制形式 |
| 参数说明 | * family：协议类型，AF\_INET6和AF\_INET * src：ip地址字符串 * dst：转换后的二进制地址 |
| 返回值 | 成功：NWY\_SUCCESS  失败：其他 |

### nwy\_socket\_htons

|  |  |
| --- | --- |
| 格式 | int16 nwy\_socket\_htons(int16 value) |
| 描述 | 用于将一个16位的无符号整数从主机字节序转换为网络字节序 |
| 参数说明 | Value：转换前的value |
| 返回值 | 转换后的value |

### nwy\_socket\_ntohs

|  |  |
| --- | --- |
| 格式 | int16 nwy\_socket\_ntohs(int16 value) |
| 描述 | 用于将一个16位的无符号整数从网络字节序转换为主机字节序 |
| 参数说明 | Value：转换前的value |
| 返回值 | 转换后的value |

### nwy\_socket\_htonl

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_htonl(int value) |
| 描述 | 用于将一个32位的无符号整数从主机字节序转换为网络字节序 |
| 参数说明 | Value：转换前的value |
| 返回值 | 转换后的value |

### nwy\_socket\_ntonl

|  |  |
| --- | --- |
| 格式 | int nwy\_socket\_ntohl(int value) |
| 描述 | 用于将一个32位的无符号整数从网络字节序转换为主机字节序 |
| 参数说明 | Value：转换前的value |
| 返回值 | 转换后的value |

## Virtual AT

### nwy\_virt\_at\_parameter\_init

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_virt\_at\_parameter\_init(); |
| 描述 | 虚拟AT初始化 |
| 参数说明 | NA |
| 返回值 | 返回对应错误码，见errno.h |

### nwy\_virt\_at\_cmd\_send

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_virt\_at\_cmd\_send(nwy\_at\_info\_t \*pInfo, char \*resp, int resp\_len, int timeout); |
| 描述 | 发送虚拟AT |
| 参数说明 | pInfo：虚拟at信息  resp\_len：回码buffer长度  timeout：超时时间默认最长30S  resp：回码信息 |
| 返回值 | 返回对应错误码，见errno.h |

### nwy\_virt\_at\_unsolicited\_cb\_reg

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_virt\_at\_unsolicited\_cb\_reg(char \*at\_prefix, void \*p\_func); |
| 描述 | 注册主动上报AT |
| 参数说明 | at\_prefix：主动上报前缀  p\_func：主动上报处理函数 |
| 返回值 | 返回对应错误码，见errno.h |

### nwy\_virt\_at\_get\_original\_rsp\_cb

|  |  |
| --- | --- |
| 格式 | nwy\_error\_enwy\_virt\_at\_get\_original\_rsp\_cb(nwy\_get\_original\_at\_rsp cb); |
| 描述 | 获取原生AT回码，注册处理函数 |
| 参数说明 | cb：处理函数 |
| 返回值 | 返回对应错误码，见errno.h |
| 说明 | ASR平台不支持 |

### nwy\_virt\_at\_forward\_cb

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_virt\_at\_forward\_cb(int index,const char\* name, nwy\_at\_forward\_process\_cb cb); |
| 描述 | 注册自定义AT |
| 参数说明 | index：序列号  name：at名称  cb：处理函数 |
| 返回值 | 返回对应错误码，见errno.h |
| 说明 | SR平台不支持 |

### nwy\_virt\_at\_forward\_send

|  |  |
| --- | --- |
| 格式 | nwy\_error\_e nwy\_virt\_at\_forward\_send(void\* handle, char\* buf, int len); |
| 描述 | 发送自定义AT |
| 参数说明 | handle：uart or usb, or vir at handle  buf:at参数 |
| 返回值 | 返回对应错误码，见errno.h |
| 说明 | ASR平台不支持 |