

Atlas 300I 推理卡  
1.0.9

## DSMI API 参考 ( 型号 3000, 3010 )

文档版本            03  
发布日期            2022-09-27



版权所有 © 华为技术有限公司 2022。保留一切权利。

非经本公司书面许可，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。

## 商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，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。

## 华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://e.huawei.com>

# 目 录

1 简介.....	1
2 使用约定.....	2
3 设备管理接口.....	4
3.1 dsmi_get_device_count.....	5
3.2 dsmi_get_all_device_count.....	6
3.3 dsmi_list_device.....	7
3.4 dsmi_get_device_health.....	8
3.5 dsmi_get_device_errorcode.....	9
3.6 dsmi_query_errorstring.....	10
3.7 dsmi_get_device_temperature.....	11
3.8 dsmi_get_device_power_info.....	12
3.9 dsmi_get_pcie_info.....	13
3.10 dsmi_get_device_voltage.....	14
3.11 dsmi_get_pcie_bdf.....	14
3.12 dsmi_get_device_utilization_rate.....	15
3.13 dsmi_get_device_frequency.....	17
3.14 dsmi_get_device_flash_count.....	18
3.15 dsmi_get_device_flash_info.....	18
3.16 dsmi_get_memory_info.....	20
3.17 dsmi_get_ecc_info.....	21
3.18 dsmi_passthru_mcu.....	22
3.19 dsmi_get_board_id.....	23
3.20 dsmi_get_board_info.....	24
3.21 dsmi_get_soc_sensor_info.....	25
3.22 dsmi_get_mini2mcu_heartbeat_status.....	27
3.23 dsmi_dft_get_elable.....	28
3.24 dsmi_get_chip_info.....	30
3.25 dsmi_get_device_die.....	31
3.26 dsmi_get_device_alarminfo.....	32
3.27 dsmi_enable_container_service.....	33
3.28 dsmi_get_phyid_from_logicid.....	33
3.29 dsmi_get_logicid_from_phyid.....	34

3.30 dsmi_get_device_cgroup_info.....	35
3.31 dsmi_gpio_direction_output.....	36
3.32 dsmi_gpio_direction_input.....	37
3.33 dsmi_gpio_set_value.....	38
3.34 dsmi_gpio_get_value.....	39
3.35 dsmi_set_device_info.....	40
3.36 dsmi_get_device_info.....	43
3.37 dsmi_get_driver_health.....	44
3.38 dsmi_get_driver_errorcode.....	45
<b>4 MAC 地址管理.....</b>	<b>47</b>
4.1 dsmi_get_mac_count.....	47
4.2 dsmi_get_mac_addr.....	48
4.3 dsmi_set_mac_addr.....	49
<b>5 风扇管理.....</b>	<b>51</b>
5.1 dsmi_get_fan_count.....	51
5.2 dsmi_get_fan_speed.....	52
<b>6 配置管理.....</b>	<b>54</b>
6.1 dsmi_config_ecc_enable.....	54
6.2 dsmi_get_ecc_enable.....	55
6.3 dsmi_get_system_time.....	56
6.4 dsmi_set_device_ip_address.....	57
6.5 dsmi_get_device_ip_address.....	59
6.6 dsmi_set_user_config.....	61
6.7 dsmi_get_user_config.....	64
6.8 dsmi_clear_user_config.....	65
6.9 dsmi_get_pcie_error_rate.....	66
6.10 dsmi_clear_pcie_error_rate.....	68
<b>7 软件升级.....</b>	<b>69</b>
7.1 dsmi_upgrade_start.....	69
7.2 dsmi_upgrade_get_state.....	71
7.3 dsmi_upgrade_get_component_static_version.....	72
7.4 dsmi_get_version.....	73
7.5 dsmi_get_component_count.....	74
7.6 dsmi_get_component_list.....	75
<b>8 昇腾 AI 处理器复位启动.....</b>	<b>78</b>
8.1 dsmi_pre_reset_soc.....	78
8.2 dsmi_rescan_soc.....	79
8.3 dsmi_get_device_boot_status.....	80
8.4 dsmi_pcie_hot_reset.....	81
<b>9 网关地址管理.....</b>	<b>83</b>

9.1 dsmi_get_gateway_addr.....	83
<b>10 不支持的接口.....</b>	<b>85</b>
<b>11 调用示例.....</b>	<b>87</b>

# 1 简介

---

DSMI ( Device System Manage Interface ) 是管理昇腾AI处理器的接口，包括设备管理接口、MAC地址管理接口、风扇管理接口、配置管理接口、软件升级接口、昇腾AI处理器复位启动接口。通过这些接口，用户可以获取昇腾AI处理器的数量、健康状况、温度、传感器、风扇等信息，便于用户监控昇腾AI处理器的状态。

## 说明

- 由于DSMI session限制，普通用户的并发场景下，同一个device最多支持8个线程或者进程同时调用DSMI接口。
- 主进程和其子进程不能同时调用DSMI接口。
- 调用者需要校验本文档中接口的返回值和出参，确保在合法范围内才能使用。

# 2 使用约定

- 本文中接口的所有参数都是必选参数。
- 昇腾310 AI处理器场景下，本文约定如下。
  - PCIe标卡，具体的产品形态包含Atlas 300I 推理卡（型号 3000）和Atlas 300I 推理卡（型号 3010）。
  - mini模块，具体的产品形态为Atlas 200 AI加速模块（型号 3000）。
  - 如果PCIe标卡或mini模块作为EP的场景，DSMI接口都运行在Host侧；mini模块作为RC的场景，DSMI接口都运行在mini侧。
- 当DCMI和DSMI接口都满足客户的需求时，请客户优先使用DCMI接口。
- 本文中的接口涉及返回码如表2-1所示。

表 2-1 返回码列表

返回码	含义
0	成功
-22或22	无效的参数
-1	不支持非root权限/未知错误
-12或12	内存溢出
0X20	调用安全函数失败
-17	获取Device数目失败
1	不支持当前命令或内部异常
2	参数错误或内存不足
3	内部异常返回
5	命令权限不足
0XC0	超时
0x20000201	信号量操作指针为空
0x20000203	信号量操作超时

返回码	含义
0x20000204	信号量P操作失败
0x20000205	信号量V操作失败
0x20000213	信号量类型错误
其它返回码	未知错误



# 3 设备管理接口

---

- [3.1 dsmi\\_get\\_device\\_count](#)
- [3.2 dsmi\\_get\\_all\\_device\\_count](#)
- [3.3 dsmi\\_list\\_device](#)
- [3.4 dsmi\\_get\\_device\\_health](#)
- [3.5 dsmi\\_get\\_device\\_errorcode](#)
- [3.6 dsmi\\_query\\_errorstring](#)
- [3.7 dsmi\\_get\\_device\\_temperature](#)
- [3.8 dsmi\\_get\\_device\\_power\\_info](#)
- [3.9 dsmi\\_get\\_pcie\\_info](#)
- [3.10 dsmi\\_get\\_device\\_voltage](#)
- [3.11 dsmi\\_get\\_pcie\\_bdf](#)
- [3.12 dsmi\\_get\\_device\\_utilization\\_rate](#)
- [3.13 dsmi\\_get\\_device\\_frequency](#)
- [3.14 dsmi\\_get\\_device\\_flash\\_count](#)
- [3.15 dsmi\\_get\\_device\\_flash\\_info](#)
- [3.16 dsmi\\_get\\_memory\\_info](#)
- [3.17 dsmi\\_get\\_ecc\\_info](#)
- [3.18 dsmi\\_passthru\\_mcu](#)
- [3.19 dsmi\\_get\\_board\\_id](#)
- [3.20 dsmi\\_get\\_board\\_info](#)
- [3.21 dsmi\\_get\\_soc\\_sensor\\_info](#)
- [3.22 dsmi\\_get\\_mini2mcu\\_heartbeat\\_status](#)
- [3.23 dsmi\\_dft\\_get\\_elable](#)

- [3.24 dsmi\\_get\\_chip\\_info](#)
- [3.25 dsmi\\_get\\_device\\_die](#)
- [3.26 dsmi\\_get\\_device\\_alarminfo](#)
- [3.27 dsmi\\_enable\\_container\\_service](#)
- [3.28 dsmi\\_get\\_phyid\\_from\\_logicid](#)
- [3.29 dsmi\\_get\\_logicid\\_from\\_phyid](#)
- [3.30 dsmi\\_get\\_device\\_cgroup\\_info](#)
- [3.31 dsmi\\_gpio\\_direction\\_output](#)
- [3.32 dsmi\\_gpio\\_direction\\_input](#)
- [3.33 dsmi\\_gpio\\_set\\_value](#)
- [3.34 dsmi\\_gpio\\_get\\_value](#)
- [3.35 dsmi\\_set\\_device\\_info](#)
- [3.36 dsmi\\_get\\_device\\_info](#)
- [3.37 dsmi\\_get\\_driver\\_health](#)
- [3.38 dsmi\\_get\\_driver\\_errorcode](#)

## 3.1 dsmi\_get\_device\_count

### 函数原型

```
int dsmi_get_device_count(int *device_count)
```

### 功能说明

查询系统中可用的昇腾AI处理器的数量，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_count	输出	Device个数，取值范围：0~64。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
...  
int ret;  
int device_count = 0;  
ret = dsmi_get_device_count(&device_count);  
...
```

## 3.2 dsmi\_get\_all\_device\_count

### 函数原型

```
int dsmi_get_all_device_count(int* all_device_count)
```

### 功能说明

查询系统中所有与Host的PCIe建链成功的昇腾AI处理器的数量，支持PCIe标卡。

### 参数说明

参数名	输入/输出	描述
all_device_count	输出	Device个数，取值范围：0~64。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
int ret = 0;  
int device_count = 0;  
ret = dsmi_get_all_device_count(&device_count);  
if(ret != 0) {  
    //todo: 记录日志  
    return ret;  
}  
...
```

## 3.3 dsmi\_list\_device

### 函数原型

```
int dsmi_list_device(int device_id_list[], int count)
```

### 功能说明

列举所有Device的设备号，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id_list[]	输出	输出所有Device的设备号列表，取值范围：0~63。
count	输入	Device个数；count由dsmi_get_device_count获取。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
...
int ret = 0;
int device_count = 0;
int device_list[64] = {0};
ret=dsmi_get_device_count(&device_count);
if((ret != 0) || (0 == device_count)){
    //todo:记录日志
    return ERROR;
}
ret = dsmi_list_device(&device_list[0], device_count);
...
```

## 3.4 dsmi\_get\_device\_health

### 函数原型

int dsmi\_get\_device\_health(int device\_id, unsigned int \*phealth)

### 功能说明

查询设备总体健康状态，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
phealth	输出	设备总体健康状态的指针，只代表本部件，不包括与本部件存在逻辑关系的其它部件。 例如， <b>phealth</b> 的值以十六进制形式显示时，健康状态值为： <ul style="list-style-type: none"><li>0：正常</li><li>1：一般告警</li><li>2：重要告警</li><li>3：紧急告警</li><li>ffffff：该设备不存在或者未启动</li></ul>

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

如果有多个告警，以最严重的告警作为设备的告警。

### 调用示例

```
...
int ret = 0;
unsigned int health;
ret = dsmi_get_device_health(0, &health);
...
```

## 3.5 dsmi\_get\_device\_errorcode

### 函数原型

```
int dsmi_get_device_errorcode(int device_id, int* errorcount, unsigned int *perrorcode)
```

### 功能说明

查询设备故障码；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
errorcount	输出	错误码数量，取值范围：0~128。
perrorcode	输出	错误码。详细的错误码描述请参见《黑匣子错误码信息列表》。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
#define ERROR_CODE_MAX_NUM      (128)
...
int ret = 0;
int errorcount = 0;
unsigned int perrorcode[ERROR_CODE_MAX_NUM] = {0};
ret = dsmi_get_device_errorcode(0, &errorcount, perrorcode);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

## 3.6 dsmi\_query\_errorstring

### 函数原型

```
int dsmi_query_errorstring(int device_id,unsigned int errorcode, unsigned char
*perrorinfo,int buffsize)
```

### 功能说明

需要先调用[dsmi\\_get\\_device\\_errorcode](#)接口查询到errorcode后，才可以调用dsmi\_query\_errorstring接口查询设备错误码描述；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
errorcode	输入	要查询的错误码。详细的错误码描述请参见《黑匣子异常重启错误码列表》。
perrorinfo	输出	对应的错误字符描述。
buffsize	输入	带入的buff大小，固定为48字节。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
#define ERROR_CODE_MAX_NUM      (128)
#define BUFF_SIZE      ( 48 )
...
int ret = 0;
int errorcount = 0;
unsigned char perrorinfo[BUFF_SIZE] = {0};
unsigned int perrorcode [ERROR_CODE_MAX_NUM] = {0};
ret = dsmi_get_device_errorcode(0, &errorcount, perrorcode);
if((ret != 0) || (errorcount == 0)) {
    //todo:记录日志
    return ret;
}
```

```
}  
ret = dsmi_query_errorstring(0, perrorcode[0], perrorinfo, BUFF_SIZE);  
if(ret != 0) {  
    //todo:记录日志  
    return ret;  
}  
...
```

## 3.7 dsmi\_get\_device\_temperature

### 函数原型

```
int dsmi_get_device_temperature(int device_id, int *ptemperature)
```

### 功能说明

查询昇腾AI处理器的温度，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
ptemperature	输出	昇腾AI处理器的温度：单位摄氏度，精度为1摄氏度，有小数点则四舍五入。16位带符号类型，小字节序。设备侧返回的值就是实际温度。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;  
int temp = 0;  
ret = dsmi_get_device_temperature(0, &temp);  
if(ret != 0) {  
    //todo: 记录日志  
    return ret;  
}  
...
```



## 3.8 dsmi\_get\_device\_power\_info

### 函数原型

```
int dsmi_get_device_power_info(int device_id,struct dsmi_power_info_stru *
pdevice_power_info)
```

### 功能说明

查询设备额定功耗，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pdevice_power_info	输出	设备额定功耗：单位为W，精度为0.1W。16位无符号short类型，小字节序。 计算公式：value=reading*0.1。 struct dsmi_power_info_stru { unsigned short power; };

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;
struct dsmi_power_info_stru powerinfo = {0};
ret = dsmi_get_device_power_info(0, &powerinfo);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

## 3.9 dsmi\_get\_pcie\_info

### 函数原型

`int dsmi_get_pcie_info(int device_id, struct tag_pcie_idinfo *pcie_idinfo)`

### 功能说明

查询PCIe设备信息，只支持PCIe标卡。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pcie_idinfo	输出	PCIe设备信息。 <pre>typedef struct tag_pcie_idinfo{     unsigned int deviceid;//设备ID     unsigned int venderid;//厂商ID     unsigned int subvenderid;//子厂商ID     unsigned int subdeviceid;//子设备ID     unsigned int bdf_deviceid;//BDF ( Bus, Device, Function ) 中的设备ID     unsigned int bdf_busid;//BDF ( Bus, Device, Function ) 中的总线ID     unsigned int bdf_funcid;//BDF ( Bus, Device, Function ) 中的功能ID }TAG_PCIE_IDINFO, tag_pcie_idinfo;</pre>

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;
struct tag_pcie_idinfo pcie_idinfo = {0};
ret = dsmi_get_pcie_info(0, &pcie_idinfo);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

## 3.10 dsmi\_get\_device\_voltage

### 函数原型

```
int dsmi_get_device_voltage(int device_id, unsigned int *pvoltage)
```

### 功能说明

查询昇腾310 AI处理器的CPU核电压，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pvoltage	输出	昇腾310 AI处理器的CPU核电压：单位为V，精度为0.01V。16位无符号short类型，小字节序。计算公式：value=reading*0.01。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;
unsigned int voltage;
ret = dsmi_get_device_voltage(0, &voltage);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

## 3.11 dsmi\_get\_pcie\_bdf

### 函数原型

```
int dsmi_get_pcie_bdf(int device_id, struct tag_pcie_bdfinfo *pcie_idinfo)
```

## 功能说明

查询PCIe设备信息，只支持PCIe标卡。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pcie_idinfo	输出	PCIe设备信息。 typedef struct tag_pcie_bdfinfo{ unsigned int bdf_deviceid;//BDF ( Bus, Device, Function ) 中的设备ID unsigned int bdf_busid;//BDF ( Bus, Device, Function ) 中的总线ID unsigned int bdf_funcid;//BDF ( Bus, Device, Function ) 中的功能ID }TAG_PCIE_BDFINFO, tag_pcie_bdfinfo;

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 异常处理

无。

## 约束说明

无。

## 调用示例

```
int ret = 0;
struct tag_pcie_bdfinfo pcie_bdf = {0};
ret = dsmi_get_pcie_bdf(0, &pcie_bdf);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

# 3.12 dsmi\_get\_device\_utilization\_rate

## 函数原型

```
int dsmi_get_device_utilization_rate(int device_id, int device_type, unsigned
int *putilization_rate)
```

功能说明

获取昇腾AI处理器的占用率，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
device_type	输入	设备类型，目前支持如下几种，数值和具体设备类型对应如下。 <ul style="list-style-type: none"><li>1：内存</li><li>2：AI CORE</li><li>3：AI CPU</li><li>4：控制CPU</li><li>5：内存带宽</li><li>6：HBM</li><li>8：DDR</li></ul>
putilization_rate	输出	昇腾AI处理器的利用率，单位：%。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
int ret = 0;
unsigned int putilization_rate;
ret = dsmi_get_device_utilization_rate(0, 1, &putilization_rate);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

## 3.13 dsmi\_get\_device\_frequency

### 函数原型

```
int dsmi_get_device_frequency(int device_id, int device_type, unsigned int *pfrequency)
```

### 功能说明

获取昇腾AI处理器的频率，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
device_type	输入	设备类型，目前支持如下几种，数值和具体设备类型对应如下。 <ul style="list-style-type: none"><li>• 1：内存</li><li>• 2：控制CPU</li><li>• 6：HBM</li><li>• 7：AI CORE当前频率</li><li>• 9：AI CORE额定频率</li></ul>
pfrequency	输出	频率，单位MHZ。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;
unsigned int frequency;
ret = dsmi_get_device_frequency(0, 1, &frequency);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
```

```
}  
...
```

## 3.14 dsmi\_get\_device\_flash\_count

### 函数原型

```
int dsmi_get_device_flash_count(int device_id, unsigned int *pflash_count)
```

### 功能说明

获取Flash个数，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pflash_count	输出	返回Flash个数，目前固定为1。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;  
unsigned int flash_count = 0;  
ret = dsmi_get_device_flash_count(0, &flash_count);  
if(ret != 0) {  
    //todo: 记录日志  
    return ret;  
}  
...
```

## 3.15 dsmi\_get\_device\_flash\_info

### 函数原型

```
int dsmi_get_device_flash_info(int device_id, unsigned int flash_index,  
dm_flash_info_stru *pflash_info)
```

功能说明

获取flash设备的信息，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
flash_index	输入	Flash索引号，目前固定为0。
pflash_info	输出	返回Flash设备信息。 FLASH 信息结构体定义 typedef struct dm_flash_info_stru { unsigned long long flash_id; /**< combined device & manufacturer code */ unsigned short device_id; /**< device id */ unsigned short vendor; /**< the primary vendor id */ unsigned int state; /**< flash health */ unsigned long long size; /**< total size in bytes */ unsigned int sector_count; /**< number of erase units */ unsigned short manufacturer_id; /** manufacturer id */ } DM_FLASH_INFO_STRU, dm_flash_info_stru;

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
int i;  
int ret = 0;  
dm_flash_info_stru flash_info = {0};  
unsigned int flash_count = 0;  
ret = dsmi_get_device_flash_count(0, &flash_count);  
...  
for (i = 0; i < flash_count; i++){  
    ret = dsmi_get_device_flash_info(0, i, &flash_info);  
    if(ret != 0) {  
        //todo: 记录日志  
    }  
    return ret;  
}  
...  
}
```



## 3.16 dsmi\_get\_memory\_info

### 函数原型

```
int dsmi_get_memory_info(int device_id, struct dsmi_memory_info_stru *  
pdevice_memory_info)
```

### 功能说明

获取内存信息，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pdevice_memory_info	输出	返回内存信息，内存信息结构体： struct dsmi_memory_info_stru { unsigned long long memory_size;//单位 MB unsigned int freq;//频率 unsigned int utiliza;//占用率 };

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;  
struct dsmi_memory_info_stru device_memory_info = {0};  
ret = dsmi_get_memory_info(0, &device_memory_info);  
if(ret != 0) {  
    //todo: 记录日志  
    return ret;  
}  
...
```

## 3.17 dsmi\_get\_ecc\_info

### 函数原型

```
int dsmi_get_ecc_info(int device_id, int device_type, struct dsmi_ecc_info_stru * pdevice_ecc_info)
```

### 功能说明

获取ECC信息，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
device_type	输入	设备类型，目前只支持DSMI_DEVICE_TYPE_DDR。 <pre>typedef enum {     DSMI_DEVICE_TYPE_DDR,     DSMI_DEVICE_TYPE_SRAM,     DSMI_DEVICE_TYPE_HBM,     DSMI_DEVICE_TYPE_NPU,     DSMI_DEVICE_TYPE_NONE = 0xff } DSMI_DEVICE_TYPE;</pre>
pdevice_ecc_info	输出	返回ECC结构体信息。 <pre>struct dsmi_ecc_info_stru{     int enable_flag;     unsigned int single_bit_error_count;     unsigned int double_bit_error_count; };</pre>

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;  
struct dsmi_ecc_info_stru device_ecc_info = {0};  
ret = dsmi_get_ecc_info(0, DSMI_DEVICE_TYPE_DDR, &device_ecc_info);  
if(ret != 0) {  
    //todo: 记录日志
```

```
return ret;
}
```

## 3.18 dsmi\_passthru\_mcu

### 函数原型

```
int dsmi_passthru_mcu(int device_id, struct passthru_message_stru
*passthru_message)
```

### 功能说明

消息转发接口，HOST消息通过Device转发MCU。对于需要通过Device获取MCU信息的功能，在HOST构建消息，把消息发送到Device，Device再转发到MCU，只支持PCIe标卡。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。 一个PCIe标卡有4个Device，但在同一时刻只有1个Device与MCU是连通的，为确保Host消息能通过Device正常转发MCU，您需要先调用 <a href="#">dsmi_get_mini2mcu_heartbeat_status</a> 接口获取哪些Device与MCU之间是连通的（状态为connect），再将处于有连通状态的Device的ID作为入参传入dsmi_passthru_mcu接口。
passthru_message	输入	消息内容，消息结构如下： struct passthru_message_stru{ unsigned int src_len; /*最长传输32个字节*/ unsigned int rw_flag; /*0 read,1 write*/ struct dmp_message_stru src_message; struct dmp_message_stru dest_message; };

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

用该接口的程序必须在物理机下运行，不支持在容器中运行。

### 调用示例

```
int ret = 0;
struct passthru_message_stru assthru_message = {0};
```

```
assthru_message.rw_flag = 1;
assthru_message.src_len = 13;
assthru_message.src_message.data.req.opcode = 0x08;
...
ret = dsmi_passthru_mcu(0,&assthru_message);
if(ret != 0) {
//todo:记录日志
return ret;
}
...
```

## 3.19 dsmi\_get\_board\_id

### 函数原型

```
int dsmi_get_board_id(int device_id, unsigned int *board_id)
```

### 功能说明

获取单板的ID。

昇腾310 AI处理器场景下，该接口支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围由dsmi_get_device_count接口决定。当前实际支持的设备号，通过dsmi_list_device接口获取。
board_id	输出	单板的ID。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 异常处理

无。

### 约束说明

无。

### 调用示例

```
int ret = 0;
unsigned int *board_id = 0;
```

```
ret = dsmi_get_board_id (0,&board_id);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
```

## 3.20 dsmi\_get\_board\_info

### 函数原型

```
int dsmi_get_board_info(int device_id, struct dsmi_board_info_stru
*pboard_info)
```

### 功能说明

获取单板信息，包括单板的board\_id、pcb\_id、bom\_id、slot\_id；支持PCIe标卡（只支持board\_id和slot\_id）、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pboard_info	输出	<pre>struct dsmi_board_info_stru {     unsigned int board_id;//指PCIe卡的board_id，非底板的PCIe插槽ID     unsigned int pcb_id;     unsigned int bom_id;     unsigned int slot_id;//如果单板上有多个昇腾AI处理器，查询是哪个昇腾AI处理器 };</pre>

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;
struct dsmi_board_info_stru board_info = {0};
ret = dsmi_get_board_info(0,&board_info);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

## 3.21 dsmi\_get\_soc\_sensor\_info

### 函数原型

```
int dsmi_get_soc_sensor_info (int device_id, int sensor_id,TAG_SENSOR_INFO
*tsensor_info)
```

### 功能说明

获取SOC传感器信息；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

参数名	输入/输出	描述
sensor_id	输入	<p>指定传感器索引，具体如下值：</p> <p>0：CLUSTER_TEMP_ID，表示CLUSTER温度。返回值对应输出联合体中的uchar成员。</p> <p>1：PERI_TEMP_ID，表示PERI温度。返回值对应输出联合体中的uchar成员。</p> <p>2：AICORE0_TEMP_ID, 表示AICORE0温度。返回值对应输出联合体中的uchar成员。</p> <p>3：AICORE1_TEMP_ID，表示AICORE1温度。返回值对应输出联合体中的uchar成员。</p> <p>4：AICORE_LIMIT_ID，AICORE限核状态返回结果是0，不限核返回结果是1。返回值对应输出联合体中的uchar成员。</p> <p>5：AICORE_TOTAL_PER_ID，表示AICORE脉冲总周期。返回值对应输出联合体中的uchar成员。</p> <p>6：AICORE_ELIM_PER_ID，表示aicore可消除周期。返回值对应输出联合体中的uchar成员。</p> <p>7：AICORE_BASE_FREQ_ID，表示aicore基准频率MHZ。返回值对应输出联合体中的ushort成员。</p> <p>8：NPU_DDR_FREQ_ID，表示DDR频率单位MHZ。返回值对应输出联合体中的ushort成员。</p> <p>9：THERMAL_THRESHOLD_ID，返回值对应输出联合体中的temp[2]成员。temp[0]为温饱限频温度，temp[1]为系统复位温度。</p> <p>10：NTC_TEMP_ID，返回值对应输出联合体中的ntc_tmp[4]成员。ntc_tmp[0] ntc_tmp[1] ntc_tmp[2] ntc_tmp[3]分别对应四个热敏电阻温度。昇腾310 AI处理器（mini模块）场景下，board_id支持000、1000、2000、004、1004、2004，如果board_id不在该范围内，查询热敏电阻温度时，接口返回报错。您可以先调用dsmi_get_board_info接口查询board_id。</p> <p>11：SOC_TEMP_ID，表示SOC最高温。返回值对应输出联合体中的uchar成员。</p>
tsensor_info	输出	<p>类型定义如下：</p> <pre> #define DSMI_TAG_SENSOR_TEMP_LEN 2 #define DSMI_TAG_SENSOR_NTC_TEMP_LEN 4 typedef union tag_sensor_info {     unsigned char uchar;     unsigned short ushort;     unsigned int uint;     signed int iint;     signed char temp[DSMI_TAG_SENSOR_TEMP_LEN]; /**&lt; 2 temp size */     signed int ntc_tmp[DSMI_TAG_SENSOR_NTC_TEMP_LEN]; /**&lt; 4 ntc_tmp size */     unsigned int data[SENSOR_DATA_MAX_LEN]; } TAG_SENSOR_INFO; </pre>

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
int ret;
TAG_SENSOR_INFO sensor_info = {0};

ret = dsmi_get_soc_sensor_info(0, 11, &sensor_info);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.22 dsmi\_get\_mini2mcu\_heartbeat\_status

函数原型

int dsmi\_get\_mini2mcu\_heartbeat\_status(int device\_id, unsigned char \*status,unsigned int \*disconn\_cnt)

功能说明

获取Device对MCU的心跳状态和计数，如果获取了多个Device的状态为connect情况下，disconn\_cnt值越小代表链路越稳定。

只支持PCIe标卡，且同一时刻一块PCIe标卡只有1个Device与MCU之间是连通的。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
status	输出	心跳状态： 0: disconnect 1: connect
disconn_cnt	输出	心跳失联次数： 范围：0~9999



返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。 若返回0xfffe，则表示不支持查询该Device的心跳状态。

约束说明

调用该接口的程序必须在物理机下运行。

调用示例

```
int ret = 0;
unsigned char tmp_status = 0;
unsigned int tmp_disconn_cnt = 0;
ret =dsmi_get_mini2mcu_heartbeat_status(2, &tmp_status, &tmp_disconn_cnt);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.23 dsmi\_dft\_get\_elable

函数原型

int dsmi\_dft\_get\_elable(int device\_id, int item\_type, char \*elable\_data, int \*len)

功能说明

获取指定设备的elable信息。仅支持miniRC场景。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过 dsmi_list_device接口获取。

参数名	输入/输出	描述
item_type	输入	查询指定标签项。 装备标签项： 0x10: Chassis Type 0x11: Chassis Part Number 0x12: Chassis Serial Number 0x20: Mfg. Date / Time 0x21: Board Manufacturer 0x22: Board Product Name 0x23: Board Serial Number 0x24: Board Part Number 0x25: FRU File ID 0x30: Product Manufacturer Name 0x31: Product Name 0x32: Product Part/Model Number 0x33: Product Version 0x34: Product Serial Number 0x35: Asset Tag 0x36: FRU File ID 0x50: 预留 0x60: System Manufacturer Name 0x61: System Product Name 0x62: System Version 0x63: System Serial Number
elable_data	输出	输出标签数据字符串。
len	输出	输出数据长度。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机下运行。

### 调用示例

```
int ret = 0;
int test_item = 0x10;
char elable_data_read[256] = {0};
int len = 0;
ret = dsmi_dft_get_elable(0, test_item, elable_data_read, &len);
if (ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

## 3.24 dsmi\_get\_chip\_info

### 函数原型

**int dsmi\_get\_chip\_info(int device\_id, struct dsmi\_chip\_info\_stru \*chip\_info)**

### 功能说明

获取昇腾AI处理器的相关信息，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
chip_info	输出	获取昇腾AI处理器的相关信息。 chip_info结构体定义： #define MAX_CHIP_NAME (32) struct dsmi_chip_info_stru{ unsigned char chip_type[MAX_CHIP_NAME]; unsigned char chip_name[MAX_CHIP_NAME]; unsigned char chip_ver[MAX_CHIP_NAME]; };

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

## 调用示例

```
int ret = 0;
struct dsmi_chip_info_stru info = {{0},{0},{0}};
ret = dsmi_get_chip_info(0, &info);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

## 3.25 dsmi\_get\_device\_die

### 函数原型

**int dsmi\_get\_device\_die(int device\_id,struct dsmi\_soc\_die\_stru \* pdevice\_die)**

### 功能说明

获取昇腾AI处理器的SOC die id相关信息，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pdevice_die	输出	返回昇腾AI处理器的SOC DIE结构体信息： #define DSMI_SOC_DIE_LEN 5 struct dsmi_soc_die_stru { unsigned int soc_die[DSMI_SOC_DIE_LEN]; /**< 5 soc_die array sizet */ };

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

### 调用示例

```
int ret = 0;
struct dsmi_soc_die_stru pdevice_die = {0};
ret = dsmi_get_device_die (0, &pdevice_die);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
```

```
}  
...
```

## 3.26 dsmi\_get\_device\_alarminfo

### 函数原型

```
int dsmi_get_device_alarminfo(int device_id, int* alarmcount, struct  
dsmi_alarm_info_stru *palarminfo)
```

### 功能说明

查询设备故障告警信息，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
alarmcount	输出	告警数量，取值范围：0~128。
palarminfo	输出	告警信息。详细的告警信息请参见《Ascend 310故障告警处理手册》。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 异常处理

无。

### 约束说明

无。

### 调用示例

```
#define ALARM_INFO_MAX_NUM      (128)  
...  
int ret = 0;  
int alarmcount = 0;  
struct dsmi_alarm_info_stru *palarminfo = NULL;  
  
palarminfo = (struct dsmi_alarm_info_stru *)malloc(sizeof(struct dsmi_alarm_info_stru) *  
ALARM_INFO_MAX_NUM);
```

```
ALARM_INFO_MAX_NUM);  
if (!palarminfo) {  
    //todo:记录日志  
    return -ENOMEM;  
}  
ret = dsmi_get_device_alarminfo(0, &alarmcount, palarminfo);  
if (ret != 0) {  
    //todo:记录日志  
    return ret;  
}  
  
free(palarminfo);  
...
```

## 3.27 dsmi\_enable\_container\_service

### 函数原型

**int dsmi\_enable\_container\_service(void)**

### 功能说明

设置当前容器为监控所有设备的容器，如果调用该接口的监控进程退出，则当前容器不再是监控所有设备的容器。支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

无。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

昇腾310 AI处理器场景下，物理机root用户，特权容器root用户下可以调用，其他场景将会返回失败。

### 调用示例

```
...  
int ret = 0;  
ret = dsmi_enable_container_service();  
...
```

## 3.28 dsmi\_get\_phyid\_from\_logicid

### 函数原型

**int dsmi\_get\_phyid\_from\_logicid(unsigned int logicid, unsigned int \*phyid)**

## 功能说明

通过昇腾AI处理器的逻辑ID获取昇腾AI处理器物理ID。支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

在容器内，逻辑ID是指设备对应的逻辑编号。物理机上，逻辑ID与物理ID相同，都是指设备编号。

## 参数说明

参数名	输入/输出	描述
logicid	输入	昇腾AI处理器的逻辑ID，取值范围：0~63。
phyid	输出	昇腾AI处理器的物理ID，取值范围：0~63。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
...  
int ret = 0;  
unsigned int logicid = 0;  
unsigned int phyid=0;  
ret = dsmi_get_phyid_from_logicid (logicid, &phyid);  
...
```

## 3.29 dsmi\_get\_logicid\_from\_phyid

### 函数原型

```
int dsmi_get_logicid_from_phyid(unsigned int phyid, unsigned int *logicid)
```

### 功能说明

通过昇腾AI处理器物理ID获取昇腾AI处理器逻辑ID。支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

在容器内，逻辑ID是指设备对应的逻辑编号。物理机上，逻辑ID与物理ID相同，都是指设备编号。

参数说明

参数名	输入/输出	描述
phyid	输入	昇腾AI处理器的物理ID，取值范围：0~63。
logid	输出	昇腾AI处理器的逻辑ID，取值范围：0~63。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
...
int ret = 0;
unsigned int phyid = 0;
unsigned int logid =0;
ret = dsmi_get_logid_from_phyid(phyid,&logid);
...
```

3.30 dsmi\_get\_device\_cgroup\_info

函数原型

int dsmi\_get\_device\_cgroup\_info(int device\_id, struct tag\_cgroup\_info \*cg\_info)

功能说明

获取cgroup内存信息，包括cgroup最大内存数、历史使用最大内存数、当前使用内存数。支持PCIe标卡。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。





参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
gpio	输入	管脚号。
value	输入	等级。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。

调用示例

```
int ret = 0;
int deviceid = 0;
unsigned int gpio = 444;
int value = 0;
ret = dsmi_gpio_direction_output(deviceid, gpio, value);
if(ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3.32 dsmi\_gpio\_direction\_input

函数原型

int dsmi\_gpio\_direction\_input(int device\_id, unsigned int gpio)

功能说明

设置相应GPIO管脚为输入模式。  
在EP场景下，存储AI卡配套存储Dorado产品使用。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
gpio	输入	管脚。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。

调用示例

```
int ret = 0;
int deviceid = 0;
unsigned int gpio = 444;
ret = dsmi_gpio_direction_input(deviceid, gpio);
if(ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3.33 dsmi\_gpio\_set\_value

函数原型

int dsmi\_gpio\_set\_value(int device\_id, unsigned int gpio, int value)

功能说明

设置相应GPIO管脚高低电平值。  
在EP场景下，存储AI卡配套存储Dorado产品使用。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
gpio	输入	管脚。
value	输入	高低电平值。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。

调用示例

```
int ret = 0;
int deviceid = 0;
unsigned int gpio = 444;
int value = 0;
ret = dsmi_gpio_set_value(deviceid, gpio, value);
if(ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3.34 dsmi\_gpio\_get\_value

函数原型

int dsmi\_gpio\_get\_value(int device\_id, unsigned int gpio, int \*p\_value)

功能说明

获取相应GPIO管脚高低电平值。  
在EP场景下，存储AI卡配套存储Dorado产品使用。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
gpio	输入	管脚。
p_value	输出	高低电平值。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。

调用示例

```
int ret = 0;
int deviceid = 0;
unsigned int gpio = 444;
int value = 0;
ret = dsmi_gpio_get_value(deviceid, gpio, &value);
if(ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3.35 dsmi\_set\_device\_info

函数原型

int dsmi\_set\_device\_info(unsigned int device\_id, DSMI\_MAIN\_CMD main\_cmd, unsigned int sub\_cmd, const void \*buf, unsigned int buf\_size)

功能说明

设置device的信息的通用接口，对各模块信息进行配置。通过main\_cmd及sub\_cmd区分不同配置项。

昇腾310 AI处理器场景下，该接口仅支持mini模块（包含mini模块作为RC或EP的场景），不同的场景下配置项不同，详细配置内容请参考[命令字说明](#)。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
main_cmd	输入	指定配置项对应主命令字。
sub_cmd	输入	指定配置项对应子命令字。
buf	输入	指定device配置信息，详细配置内容请参考 <a href="#">命令字说明</a> 。
size	输入	指定device配置有效字节数。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

返回值-2或-22表示该接口不支持输入的命令字，其它返回值意义见[表2-1](#)。

命令字说明

主命令字 ( main_cmd )	子命令字 ( sub_cmd )	功能描述	参数说明	使用场景
DSMI_MAIN_CMD_EX_CONTAINER	DSMI_EX_CONTAINER_SUB_CMD_SHARE	支持指定 device 映射到多容器。	<ul style="list-style-type: none"><li>buf: device 映射到多容器的取值。取值为 int 类型。<ul style="list-style-type: none"><li>0x1: 使能 device 映射到多容器。</li><li>0x0: 关闭 device 映射到多容器。</li></ul></li><li>size: device 映射到多容器的数据结构大小固定为 4 字节。</li></ul>	仅支持 mini 模块 ( 包含 mini 模块作为 RC 或 EP 的场景 )。

约束说明

调用该接口的程序必须在物理机的 root 用户下运行，若在物理机的非 root 用户，或在容器下运行，则会返回权限错误。

调用示例

```
int ret = 0;
int device_id = 0;
int dev_share = 1;//开启共享
unsigned int size;
ret=dsmi_set_device_info(device_id, DSMI_MAIN_CMD_EX_CONTAINER,
DSMI_EX_CONTAINER_SUB_CMD_SHARE, &dev_share, size);
if(ret != 0){
//todo:记录日志
return ret;
}
...
```

### 3.36 dsmi\_get\_device\_info

#### 函数原型

```
int dsmi_get_device_info(unsigned int device_id, DSMI_MAIN_CMD main_cmd,
unsigned int sub_cmd, void *buf, unsigned int *size)
```

#### 功能说明

获取device的信息的通用接口，获取各模块中的状态信息。通过main\_cmd及sub\_cmd区分不同配置项。

昇腾310 AI处理器场景下，该接口仅支持mini模块（包含mini模块作为RC或EP的场景），不同的场景下配置项不同，详细配置内容请参考[命令字说明](#)。

#### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
main_cmd	输入	指定配置项对应主命令字。
sub_cmd	输入	指定配置项对应子命令字。
buf	输出	获取device配置信息，详细配置内容请参考 <a href="#">命令字说明</a> 。
size	输出	获取device配置有效字节数。

#### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

返回值-2或-22表示该接口不支持输入的命令字，其它返回值意义见[表2-1](#)。



命令字说明

主命令字 ( main_cmd )	子命令字 ( sub_cmd )	功能描述	参数说明	使用场景
DSMI_MAIN_CMD_EX_CONTAINER	DSMI_EX_CONTAINER_SUB_CMD_SHARE	支持指定 device 映射到多容器。	<ul style="list-style-type: none"><li>• buf: device 映射到多容器的取值。取值为 int 类型。<ul style="list-style-type: none"><li>- 0x1: 使能 device 映射到多容器。</li><li>- 0x0: 关闭 device 映射到多容器。</li></ul></li><li>• size: device 映射到多容器的数据结构大小固定为 4 字节。</li></ul>	仅支持 mini 模块（包含 mini 模块作为 RC 或 EP 的场景）。

约束说明

调用该接口的程序必须在物理机的 root 用户下运行，若在物理机的非 root 用户，或在容器下运行，则会返回权限错误。

调用示例

```
int ret = 0;
int device_id = 0;
int dev_share;
unsigned int size;
ret=dsmi_get_device_info(device_id, DSMI_MAIN_CMD_EX_CONTAINER,
DSMI_EX_CONTAINER_SUB_CMD_SHARE, &dev_share, &size);
if(ret != 0){
//todo:记录日志
return ret;
}
...
```

3.37 dsmi\_get\_driver\_health

函数原型

```
int dsmi_get_driver_health(unsigned int *phealth)
```

功能说明

驱动健康状态。

昇腾310 AI处理器场景下，该接口支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

参数说明

参数名	输入/输出	描述
phealth	输出	驱动健康状态的指针。 例如， <b>phealth</b> 的值以十六进制形式显示时，健康状态值为： <ul style="list-style-type: none"><li>0：正常</li><li>1：一般告警</li><li>2：重要告警</li><li>3：紧急告警</li><li>ffffffff：该设备不存在或者未启动</li></ul>

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

此接口适用于固件版本为1.76.22.3及以上版本。

调用示例

```
...
int ret = 0;
unsigned int health;
ret = dsmi_get_driver_health(&health);
...
```

3.38 dsmi\_get\_driver\_errorcode

函数原型

int dsmi\_get\_driver\_errorcode(int \*errorcount, unsigned int \*perrorcode)

功能说明

查询驱动故障码。

昇腾310 AI处理器场景下，该接口支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

参数说明

参数名	输入/输出	描述
errorcount	输出	错误码数量，取值范围：0~128。
perrorcode	输出	错误码。perrorcode长度至少为128* sizeof(unsigned int)字节。 详细的错误码描述请参见《黑匣子错误码信息列表》。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
#define ERROR_CODE_MAX_NUM      (128)
...
int ret = 0;
int errorcount = 0;
unsigned int perrorcode[ERROR_CODE_MAX_NUM] = {0};
ret = dsmi_get_driver_errorcode(&errorcount, perrorcode);
if(ret != 0) {
//todo:记录日志
return ret;
}
...
```

# 4 MAC 地址管理

- 4.1 dsmi\_get\_mac\_count
- 4.2 dsmi\_get\_mac\_addr
- 4.3 dsmi\_set\_mac\_addr

## 4.1 dsmi\_get\_mac\_count

### 函数原型

```
int dsmi_get_mac_count(int device_id, int* count)
```

### 功能说明

查询MAC地址数量；仅支持miniRC场景。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
count	输出	查询出MAC数，取值范围：0~4。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
int ret = 0;
int count = -1;
ret = dsmi_get_mac_count(0,&count);
if(ret != 0) {
//todo:记录日志
return ret;
}
...
```

# 4.2 dsmi\_get\_mac\_addr

## 函数原型

**int dsmi\_get\_mac\_addr (int device\_id, int mac\_id, char \*pmac\_addr, unsigned int mac\_addr\_len)**

## 功能说明

获取指定设备的MAC地址；仅支持miniRC场景。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
mac_id	输入	取值范围：0~3。
pmac_addr	输出	输出6个字节的MAC地址。
mac_addr_len	输入	MAC地址长度，固定长度6，单位byte。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
int ret = 0;
int count = -1;
char mac_addr[6] = {0};
ret = dsmi_get_mac_count(0,&count);
...
for (i = 0; i < count; i++){
ret = dsmi_get_mac_addr(0, i, mac_addr, 6);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
}
```

## 4.3 dsmi\_set\_mac\_addr

### 函数原型

```
int dsmi_set_mac_addr(int device_id, int mac_id, const char *pmac_addr,
unsigned int mac_addr_len)
```

### 功能说明

设置指定设备的MAC地址；仅支持miniRC场景。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
mac_id	输入	取值范围：0~3。
pmac_addr	输入	设置6个字节的MAC地址。
mac_addr_len	输入	MAC地址长度，固定长度6，单位byte。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

调用该接口的程序必须在物理机的root用户下运行。

## 调用示例

```
int ret = 0;
int count = 1;
char mac_addr[6] = {0x52,0x12,0x36,0x26,0x82,0x66};
ret = dsmi_set_mac_addr(0, count, mac_addr, 6);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

# 5 风扇管理

[5.1 dsmi\\_get\\_fan\\_count](#)

[5.2 dsmi\\_get\\_fan\\_speed](#)

## 5.1 dsmi\_get\_fan\_count

### 函数原型

```
int dsmi_get_fan_count(int device_id, int* count)
```

### 功能说明

获取Device上小风扇数，大部分场景一个昇腾AI处理器只有一个风扇，但可能存在支持多个风扇的情况，如果有多个小风扇，提供查询有几个风扇的接口；支持mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
count	输出	查询小风扇个数，取值范围：目前固定为1。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。



### 约束说明

无。

### 调用示例

```
int ret = 0;
int count = 0;
ret = dsmi_get_fan_count(0, &count);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

## 5.2 dsmi\_get\_fan\_speed

### 函数原型

**int dsmi\_get\_fan\_speed(int device\_id, int fan\_id, int \*speed)**

### 功能说明

查询指定风扇的转速，为风扇实际转速，单位RPM；只支持mini模块（仅限于mini模块作为RC）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
fan_id	输入	如果有多个风扇，fan_id从1开始编号，大于等于1表示查询指定fan_id的风扇转速。 如果fan_id为0，则表示查询所有风扇的平均速度。
speed	输出	输出风扇转速值数组，由调用者申请。 调用成功后，该空间存储为风扇转速，单位为RPM，即转/分钟。 取值范围：0~(18000±10%)

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

无。

## 调用示例

```
int ret = 0;
int speed = 0;
int count = 0;
ret = dsmi_get_fan_count(0, &count);
...
for (i = 1; i <= count; i++){
    ret = dsmi_get_fan_speed(0, i, &speed);
    if(ret != 0) {
        //todo:记录日志
    }
    return ret;
}
...
}
```

# 6 配置管理

---

- 6.1 [dsmi\\_config\\_ecc\\_enable](#)
- 6.2 [dsmi\\_get\\_ecc\\_enable](#)
- 6.3 [dsmi\\_get\\_system\\_time](#)
- 6.4 [dsmi\\_set\\_device\\_ip\\_address](#)
- 6.5 [dsmi\\_get\\_device\\_ip\\_address](#)
- 6.6 [dsmi\\_set\\_user\\_config](#)
- 6.7 [dsmi\\_get\\_user\\_config](#)
- 6.8 [dsmi\\_clear\\_user\\_config](#)
- 6.9 [dsmi\\_get\\_pcie\\_error\\_rate](#)
- 6.10 [dsmi\\_clear\\_pcie\\_error\\_rate](#)

## 6.1 dsmi\_config\_ecc\_enable

### 函数原型

```
int dsmi_config_ecc_enable(int device_id, DSMI_DEVICE_TYPE device_type, int enable_flag)
```

### 功能说明

配置存储ECC的标记为使能或禁用；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。调用该接口配置ECC标记后，需要执行reboot命令重启系统，配置才能生效。

配置ECC标记后，可通过[6.2 dsmi\\_get\\_ecc\\_enable](#)接口查看ECC标记。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
device_type	输入	设备类型，目前支持DSMI_DEVICE_TYPE_DDR。 typedef enum { DSMI_DEVICE_TYPE_DDR, DSMI_DEVICE_TYPE_SRAM, DSMI_DEVICE_TYPE_HBM, DSMI_DEVICE_TYPE_NPU, DSMI_DEVICE_TYPE_NONE = 0xff } DSMI_DEVICE_TYPE;
enable_flag	输入	1：使能 0：禁用

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

- 调用该接口的程序必须在物理机的root用户下运行。
- 使能ECC后，会导致可用DDR内存减少，推理性能下降，且可能存在波动。
- 此接口设置的内容会涉及flash擦写，由于flash擦写次数是有限制的，所以不建议频繁调用该接口。

调用示例

```
int ret = 0;
ret = dsmi_config_ecc_enable(0, DSMI_DEVICE_TYPE_DDR, 1);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
ret = dsmi_config_ecc_enable(0, DSMI_DEVICE_TYPE_DDR, 0);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
```

6.2 dsmi\_get\_ecc\_enable

函数原型

int dsmi\_get\_ecc\_enable(int device\_id, DSMI\_DEVICE\_TYPE device\_type, int\* enable\_flag)

功能说明

查询存储ECC的使能标记；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
device_type	输入	设备类型，目前只支持DSMI_DEVICE_TYPE_DDR。 typedef enum { DSMI_DEVICE_TYPE_DDR, DSMI_DEVICE_TYPE_SRAM, DSMI_DEVICE_TYPE_HBM, DSMI_DEVICE_TYPE_NPU, DSMI_DEVICE_TYPE_NONE = 0xff } DSMI_DEVICE_TYPE;
enable_flag	输出	1：使能 0：禁用

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
int ret = 0;
int enable_flag = 0;
ret = dsmi_get_ecc_enable(0, DSMI_DEVICE_TYPE_DDR, &enable_flag);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

6.3 dsmi\_get\_system\_time

函数原型

```
int dsmi_get_system_time(int device_id, unsigned int *ntime_stamp)
```

## 功能说明

获取系统时间；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
ntime_stamp	输出	表示从1970年00:00:00至今的秒数值。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
int ret = 0;
int time = 0;
ret = dsmi_get_system_time (0, &time);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

# 6.4 dsmi\_set\_device\_ip\_address

## 函数原型

```
int dsmi_set_device_ip_address (int device_id, int port_type, int port_id,
ip_addr_t ip_address, ip_addr_t mask_address)
```

## 功能说明

设置ip地址和mask地址，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景）。

MiniRC场景下，不支持该接口。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
port_type	输入	指定网口类型，取值范围： <ul style="list-style-type: none"> <li>• VNIC: 0x00</li> </ul> 对外提供宏： DSMI_VNIC_PORT (0)
port_id	输入	指定网口号，保留字段。取值范围： 【 0~255 】。
ip_address	输入	<pre>#define DSMI_ARRAY_IPV4_NUM 4 #define DSMI_ARRAY_IPV6_NUM 16  typedef struct ip_addr {     union {         unsigned char ip6[DSMI_ARRAY_IPV6_NUM];         unsigned char ip4[DSMI_ARRAY_IPV4_NUM];     } u_addr;     enum ip_addr_type ip_type; } ip_addr_t;  enum ip_addr_type {     /** IPv4 */     IPADDR_TYPE_V4 = 0U,     /** IPv6 */     IPADDR_TYPE_V6 = 6U,     /**&lt; IPv4+IPv6 ("dual-stack") */     IPADDR_TYPE_ANY = 46U };</pre>
mask_address	输入	<pre>#define DSMI_ARRAY_IPV4_NUM 4 #define DSMI_ARRAY_IPV6_NUM 16  typedef struct ip_addr {     union {         unsigned char ip6[DSMI_ARRAY_IPV6_NUM];         unsigned char ip4[DSMI_ARRAY_IPV4_NUM];     } u_addr;     enum ip_addr_type ip_type; } ip_addr_t;  enum ip_addr_type {     /** IPv4 */     IPADDR_TYPE_V4 = 0U,     /** IPv6 */     IPADDR_TYPE_V6 = 6U,     /**&lt; IPv4+IPv6 ("dual-stack") */     IPADDR_TYPE_ANY = 46U };</pre> <p><b>说明</b> 当前版本不支持IPADDR_TYPE_ANY 字段。 以ip_address的ip_type为准，mask_address的ip_type无效。</p>

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

调用该接口的程序必须在物理机的root用户下运行。

## 调用示例

```
int ret = 0;
int device_id = 0;
int port_type = 0;
int port_id = 0;
unsigned int ip_addr = 0xC801A8C0; //192.168.1.200
unsigned int mask_addr = 0x00FFFFFF; //255.255.255.0
ip_addr_t ip_address = {0};
ip_addr_t ip_mask_address={0};
memcpy(&(ip_address.u_addr.ip4[0]),&ip_addr,4);
memcpy(&(ip_mask_address.u_addr.ip4[0]),&mask_addr,4);
ret = dsmi_set_device_ip_address(device_id, port_type, port_id, ip_address, ip_mask_address);
if(ret != 0) {
//todo:记录日志
return ret;
}
...
```

# 6.5 dsmi\_get\_device\_ip\_address

## 函数原型

```
int dsmi_get_device_ip_address (int device_id, int port_type, int port_id,
ip_addr_t *ip_address, ip_addr_t *mask_address)
```

## 功能说明

获取IP地址和mask地址，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景）。

MiniRC场景下，不支持该接口。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。



参数名	输入/输出	描述
port_type	输入	指定网口类型，取值范围： <ul style="list-style-type: none"><li>• VNIC: 0x00</li></ul> 对外提供宏： DSMI_VNIC_PORT (0)
port_id	输入	指定网口号，保留字段。取值范围： 【 0~255 】。
ip_address	输入和输出	<pre>#define DSMI_ARRAY_IPV4_NUM 4 #define DSMI_ARRAY_IPV6_NUM 16  typedef struct ip_addr {     union {         unsigned char ip6[DSMI_ARRAY_IPV6_NUM];         unsigned char ip4[DSMI_ARRAY_IPV4_NUM];     } u_addr;     enum ip_addr_type ip_type; //这是一个输入值 } ip_addr_t;  enum ip_addr_type {     /** IPv4 */     IPADDR_TYPE_V4 = 0U,     /** IPv6 */     IPADDR_TYPE_V6 = 6U,     /**&lt; IPv4+IPv6 ("dual-stack") */     IPADDR_TYPE_ANY = 46U };</pre>
mask_address	输入和输出	<pre>#define DSMI_ARRAY_IPV4_NUM 4 #define DSMI_ARRAY_IPV6_NUM 16  typedef struct ip_addr {     union {         unsigned char ip6[DSMI_ARRAY_IPV6_NUM];         unsigned char ip4[DSMI_ARRAY_IPV4_NUM];     } u_addr;     enum ip_addr_type ip_type; //这是一个输入值 } ip_addr_t;  enum ip_addr_type {     /** IPv4 */     IPADDR_TYPE_V4 = 0U,     /** IPv6 */     IPADDR_TYPE_V6 = 6U,     /**&lt; IPv4+IPv6 ("dual-stack") */     IPADDR_TYPE_ANY = 46U };</pre> <p><b>说明</b> 当前版本不支持IPADDR_TYPE_ANY 字段。 以ip_address的ip_type为准，mask_address的ip_type无效。</p>

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
int ret = 0;
int device_id = 0;
int port_type = 0;
int port_id = 0;
ip_addr_t ip_address = {0};
ip_addr_t ip_mask_address={0};
ret = dsmi_get_device_ip_address(device_id, port_type, port_id, &ip_address, &mask_address);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
```

## 6.6 dsmi\_set\_user\_config

### 函数原型

```
int dsmi_set_user_config(int device_id, const char *config_name, unsigned int
buf_size, unsigned char *buf)
```

### 功能说明

设置用户配置。

- 设置ddr\_ecc\_enable，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。
- 设置aicpu\_config，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。
- 设置ssh\_status，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景，miniRC场景下的配置不生效）。
- 设置set\_nve\_level，支持mini模块（仅支持mini模块作为RC的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

参数名	输入/输出	描述
config_name	输入	<p>目前支持处理的配置项名称如下，配置项名称的字符串长度最大为254：</p> <ul style="list-style-type: none"> <li>• "ddr_ecc_enable"：用于使能或禁用ECC。</li> <li>• "aicpu_config"：用于配置AI CPU和Control CPU的配比。</li> <li>• "ssh_status"：用于打开或关闭ssh服务。</li> <li>• "set_nve_level"：用于设置功耗和算力档位。</li> </ul> <p><b>说明</b> 调用该接口配置ECC标记、AI CPU和Control CPU的配比、ssh服务开关后，需要执行reboot命令重启系统，配置才能生效。配置生效后，可通过<a href="#">6.7 dsmi_get_user_config</a>接口查看配置结果。</p>
buf_size	输入	<p>buf长度，单位为byte，最大长度为1024 byte。</p> <ul style="list-style-type: none"> <li>• 如果配置"ddr_ecc_enable"：buf_size参数配置为1。</li> <li>• 如果配置"aicpu_config"：buf_size参数配置为1。</li> <li>• 如果配置"ssh_status"：buf_size参数配置为1。</li> <li>• 如果配置"set_nve_level"：buf_size参数配置为1。</li> </ul>

参数名	输入/输出	描述
buf	输入	<p>buf指针，指向配置项内容。</p> <ul style="list-style-type: none"> <li>针对"ddr_ecc_enable"配置项，配置项内容支持如下选项，默认值是0： <ul style="list-style-type: none"> <li>1：表示使能ECC</li> <li>0：表示禁用ECC</li> </ul> </li> <li>针对"aicpu_config"配置项，配置项内容支持如下选项，默认值是240： <ul style="list-style-type: none"> <li>192：表示2个AI CPU，6个Control CPU</li> <li>240：表示4个AI CPU，4个Control CPU</li> <li>252：表示6个AI CPU，2个Control CPU</li> </ul> </li> <li>针对"ssh_status"配置项，配置项内容支持如下选项，默认值是0： <ul style="list-style-type: none"> <li>1：打开ssh服务</li> <li>0：关闭ssh服务</li> </ul> </li> <li>针对"set_nve_level"配置项，配置项内容支持如下选项，默认值是3： <ul style="list-style-type: none"> <li>0：功耗和算力档位为low</li> <li>1：功耗和算力档位为middle</li> <li>2：功耗和算力档位为high</li> <li>3：功耗和算力档位为full</li> </ul> </li> </ul>

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

- 调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。
- 使能ECC后，会导致可用DDR内存减少，推理性能下降。
- 此接口设置的内容会涉及flash擦写，由于flash擦写次数是有限制的，所以不建议频繁调用该接口。

## 调用示例

```
#define BUF_SIZE 1
int ret = 0;
int device_id = 0;
char *config_name = "ddr_ecc_enable";
char buf[BUF_SIZE] = {0};
ret=dsmi_set_user_config(device_id,config_name, BUF_SIZE, buf);
if(ret != 0){
//todo:记录日志
```

```
return ret;  
}  
...
```

## 6.7 dsmi\_get\_user\_config

### 函数原型

```
int dsmi_get_user_config(int device_id, const char *config_name, unsigned int  
buf_size, unsigned char *buf)
```

### 功能说明

获取用户配置。

- 获取ddr\_ecc\_enable配置，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。
- 获取aicpu\_config配置，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。
- 获取ssh\_status配置，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景，miniRC场景下的配置不生效）。
- 获取get\_nve\_level配置，支持mini模块（仅支持mini模块作为RC的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
config_name	输入	目前支持处理的配置项名称如下： <ul style="list-style-type: none"><li>• "ddr_ecc_enable"：用于使能或禁用ECC。</li><li>• "aicpu_config"：用于配置AI CPU和Control CPU的配比。</li><li>• "ssh_status"：用于打开或关闭ssh服务。</li><li>• "get_nve_level"：用于获取功耗和算力档位。</li></ul>
buf_size	输入	buf长度，最大长度为1024 byte。 该参数的配置，请参见 <a href="#">6.6 dsmi_set_user_config</a> 。
buf	输出	buf指针，指向配置项内容。 具体配置项内容的含义，请参见 <a href="#">6.6 dsmi_set_user_config</a> 。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

以下示例以config\_name = "ddr\_mac\_enable"为例。

```
#define BUF_SIZE 1
int ret = 0;
int device_id = 0;
char *config_name = "ddr_mac_enable";
char buf[BUF_SIZE] = {0};
ret=dsmi_get_user_config(device_id, config_name, BUF_SIZE, buf);
if(ret != 0){
//todo:记录日志
return ret;
}
...
```

# 6.8 dsmi\_clear\_user\_config

## 函数原型

**int dsmi\_clear\_user\_config(int device\_id, const char \*config\_name)**

## 功能说明

重置用户配置。

- 重置ddr\_ecc\_enable配置，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。
- 重置aicpu\_config配置，支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。
- 重置ssh\_status配置，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景，miniRC场景下的配置不生效）。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

参数名	输入/输出	描述
config_name	输入	<p>目前支持处理的配置项名称如下：</p> <ul style="list-style-type: none"> <li>"ddr_ecc_enable": 用于使能或禁用ECC。</li> <li>"aicpu_config": 用于配置AI CPU和控制 CPU的配比。</li> <li>"ssh_status": 用于打开或关闭ssh服务。</li> </ul> <p><b>说明</b> 调用该接口配置ECC标记、AI CPU和控制 CPU的配比、ssh服务开关后，需要执行reboot命令重启系统，配置才能生效。配置生效后，可通过<a href="#">6.7 dsmi_get_user_config</a>接口查看配置结果。</p>

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

- 调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。
- 此接口设置的内容会涉及flash擦写，由于flash擦写次数是有限制的，所以不建议频繁调用该接口。

## 调用示例

```
int ret = 0;
int device_id = 0;
char *config_name = "ddr_ecc_enable";
ret=dsmi_clear_user_config(device_id, config_name);
if(ret != 0){
//todo:记录日志
return ret;
}
...
```

## 6.9 dsmi\_get\_pcie\_error\_rate

### 函数原型

```
int dsmi_get_pcie_error_rate(int device_id, struct dsmi_chip_pcie_err_rate_stru
*pcie_err_code_info)
```

### 功能说明

获取PCIe链路误码信息。

昇腾310 AI处理器场景下，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景）。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
pcie_err_code_info	输出	<pre>typedef struct dsmi_chip_pcie_err_rate_stru {     unsigned int reg_deskew_fifo_overflow_intr_status;     unsigned int reg_symbol_unlock_intr_status;     unsigned int reg_deskew_unlock_intr_status;     unsigned int reg_phystatus_timeout_intr_status;     unsigned int symbol_unlock_counter;     unsigned int pcs_rx_err_cnt;     unsigned int phy_lane_err_counter;     unsigned int pcs_rcv_err_status;     unsigned int symbol_unlock_err_status;     unsigned int phy_lane_err_status;     unsigned int dl_lcrc_err_num;     unsigned int dl_dcrc_err_num; } PCIE_ERR_RATE_INFO_STU;</pre>

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。

仅支持固件1.73.5.6.50及以上版本。

调用示例

```
int ret = 0;
int device_id = 0;
char *pcie_err_code_info = NULL;
pcie_err_code_info = (struct dsmi_chip_pcie_err_rate_stru *)
    malloc(sizeof(struct dsmi_chip_pcie_err_rate_stru));
if(!pcie_err_code_info) {
    return -EINVAL;
}
ret=dsmi_get_pcie_err_rate(device_id, pcie_err_code_info);
if(ret != 0){
    //todo:记录日志
    free(pcie_err_code_info);
    return ret;
}
...
```



## 6.10 dsmi\_clear\_pcie\_error\_rate

### 函数原型

**int dsmi\_clear\_pcie\_error\_rate(int device\_id)**

### 功能说明

清除当前PCIe链路误码统计信息。

昇腾310 AI处理器场景下，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号。 昇腾310 AI处理器场景下，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 约束说明

调用该接口的程序必须在物理机的root用户下运行，若在物理机的非root用户，或在容器下运行，则会返回权限错误。

仅支持固件1.73.5.6.50及以上版本。

### 调用示例

```
int ret = 0;
int device_id = 0;
ret=dsmi_clear_pcie_err_rate(device_id);
if(ret != 0){
//todo:记录日志
return ret;
}
...
```

# 7 软件升级

- 7.1 dsmi\_upgrade\_start
- 7.2 dsmi\_upgrade\_get\_state
- 7.3 dsmi\_upgrade\_get\_component\_static\_version
- 7.4 dsmi\_get\_version
- 7.5 dsmi\_get\_component\_count
- 7.6 dsmi\_get\_component\_list

## 7.1 dsmi\_upgrade\_start

### 函数原型

```
int dsmi_upgrade_start(int device_id,DSMI_COMPONENT_TYPE
component_type,const char *file_name)
```

### 功能说明

启动指定昇腾AI处理器的升级，指定升级固件类型、升级的文件名；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

参数名	输入/输出	描述
component_type	输入	固件类型。 目前支持DSMI_COMPONENT_TYPE_NVE、DSMI_COMPONENT_TYPE_XLOADER、DSMI_COMPONENT_TYPE_M3FW、DSMI_COMPONENT_TYPE_UEFI、DSMI_COMPONENT_TYPE_TEE、DSMI_COMPONENT_TYPE_AICPU。 <pre>typedef enum dsmi_component_type {     DSMI_COMPONENT_TYPE_NVE,     DSMI_COMPONENT_TYPE_XLOADER,     DSMI_COMPONENT_TYPE_M3FW,     DSMI_COMPONENT_TYPE_UEFI,     DSMI_COMPONENT_TYPE_TEE,     DSMI_COMPONENT_TYPE_KERNEL,     DSMI_COMPONENT_TYPE_DTB,     DSMI_COMPONENT_TYPE_ROOTFS,     DSMI_COMPONENT_TYPE_IMU,     DSMI_COMPONENT_TYPE_IMP,     DSMI_COMPONENT_TYPE_AICPU,     DSMI_COMPONENT_TYPE_MAX,     UPGRADE_AND_RESET_ALL_COMPONENT = 0xFFFFFFFF7,     UPGRADE_ALL_COMPONENT = 0xFFFFFFFF } DSMI_COMPONENT_TYPE;</pre>
file_name	输入	固件文件名（包含路径）。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

- 非宿主机环境不支持升级功能。
- 将**component\_type**设置为DSMI\_COMPONENT\_TYPE\_AICPU时，调用该接口的程序必须在物理机的root用户下运行；设置为其它类型时，调用该接口的程序必须在物理机的root用户下运行。
- 此接口设置的内容会涉及flash擦写，由于flash擦写次数是有限制的，所以不建议频繁调用该接口。

调用示例

```
int ret = 0;  
ret = dsmi_upgrade_start(0, DSMI_COMPONENT_TYPE_XLOADER, “./xloader.bin”);  
if(ret != 0) {  
    //todo: 记录日志  
    return ret;  
}  
...
```

## 7.2 dsmi\_upgrade\_get\_state

### 函数原型

```
int dsmi_upgrade_get_state(int device_id, unsigned char *schedule, unsigned char *upgrade_status)
```

### 功能说明

查询固件升级状态及进度；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
schedule	输出	升级进度，0~100百分比。
upgrade_status	输出	升级状态，目前支持如下几种： 0：UPGRADE_IDLE_STATE表示处于空闲状态 1：IS_UPGRADING表示处于正在升级的状态 2：UPGRADE_NOT_SUPPORT表示不支持升级，错误状态，不可触发升级 3：UPGRADE_UPGRADE_FAIL表示升级失败的状态 4：UPGRADE_STATE_NONE表示未进行固件升级，处于初始化状态 5：UPGRADE_WAITTING_RESTART表示正在等待重启 6：UPGRADE_WAITTING_SYNC表示正在等待同步 7：UPGRADE_SYNCHRONIZING表示正在同步升级状态

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 调用示例

```
int ret = 0;
unsigned char schedule;
unsigned char upgrade_status;
ret = dsmi_upgrade_get_state(0, &schedule, &upgrade_status);
```

```
if(ret != 0) {  
    //todo: 记录日志  
    return ret;  
}
```

## 7.3 dsmi\_upgrade\_get\_component\_static\_version

### 函数原型

```
int dsmi_upgrade_get_component_static_version(int device_id,  
DSMI_COMPONENT_TYPE component_type, unsigned char* version_str,  
unsigned int version_len, unsigned int *ret_len)
```

### 功能说明

查询FLASH上固件的版本；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
component_type	输入	固件类型。 目前支持DSMI_COMPONENT_TYPE_NVE、DSMI_COMPONENT_TYPE_XLOADER、DSMI_COMPONENT_TYPE_M3FW、DSMI_COMPONENT_TYPE_UEFI、DSMI_COMPONENT_TYPE_TEE、DSMI_COMPONENT_TYPE_AICPU。 <pre>typedef enum dsmi_component_type {     DSMI_COMPONENT_TYPE_NVE,     DSMI_COMPONENT_TYPE_XLOADER,     DSMI_COMPONENT_TYPE_M3FW,     DSMI_COMPONENT_TYPE_UEFI,     DSMI_COMPONENT_TYPE_TEE,     DSMI_COMPONENT_TYPE_KERNEL,     DSMI_COMPONENT_TYPE_DTB,     DSMI_COMPONENT_TYPE_ROOTFS,     DSMI_COMPONENT_TYPE_IMU,     DSMI_COMPONENT_TYPE_IMP,     DSMI_COMPONENT_TYPE_AICPU,     DSMI_COMPONENT_TYPE_MAX,     UPGRADE_AND_RESET_ALL_COMPONENT = 0xFFFFFFFF7,     UPGRADE_ALL_COMPONENT = 0xFFFFFFFFF } DSMI_COMPONENT_TYPE;</pre>
version_str	输出	用户申请的空间，存放返回的固件版本号。
version_len	输入	version_str的内存大小，单位Byte。
ret_len	输出	版本号长度。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

调用该接口的程序必须在物理机、容器的root用户下运行。

## 调用示例

```
int ret = 0;
unsigned int len = 0;
unsigned char version_str[64] = {0};
ret = dsmi_upgrade_get_component_static_version(0, DSMI_COMPONENT_TYPE_XLOADER, version_str, 64,
&len);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

# 7.4 dsmi\_get\_version

## 函数原型

```
int dsmi_get_version(int device_id, char* version_str, unsigned int version_len,
unsigned int *ret_len)
```

## 功能说明

查询系统版本；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
version_str	输出	返回系统版本。 该空间由用户申请，大小为64Byte。
version_len	输入	version_str的内存空间大小，单位Byte。
ret_len	输出	返回版本号长度。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

无。

## 调用示例

```
int ret = 0;
int len = -1;
unsigned char version_str[64] = {0};
ret = dsmi_get_version (0, version_str, 64, &len);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
...
```

# 7.5 dsmi\_get\_component\_count

## 函数原型

```
int dsmi_get_component_count(int device_id, unsigned int
*component_count)
```

## 功能说明

获取可升级组件的个数；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
component_count	输出	返回组件的个数。

## 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

## 约束说明

调用该接口的程序必须在物理机、容器的root用户下运行。

## 调用示例

```
int ret = 0;
unsigned int component_num = 0;
ret =dsmi_get_component_count(0, &component_num);
if(ret != 0) {
//todo: 记录日志
return ret;
}
```

# 7.6 dsmi\_get\_component\_list

## 函数原型

**int dsmi\_get\_component\_list(int device\_id, DSMI\_COMPONENT\_TYPE  
\*component\_table, unsigned int component\_count)**

## 功能说明

获取可升级组件列表；支持PCIe标卡、mini模块（包含mini模块作为RC或EP的场景）。

需要先调用[dsmi\\_get\\_component\\_count](#)接口获取可升级组件的个数后，再调用dsmi\_get\_component\_list接口获取组件列表。

## 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
component_count	输入	组件个数。



参数名	输入/输出	描述
component_table	输出	<div>返回可升级组件列表，具体值含义如下： 目前支持DSMI_COMPONENT_TYPE_NVE、 DSMI_COMPONENT_TYPE_XLOADER、 DSMI_COMPONENT_TYPE_M3FW、 DSMI_COMPONENT_TYPE_UEFI、 DSMI_COMPONENT_TYPE_TEE、 DSMI_COMPONENT_TYPE_AICPU。</div> <div>typedef enum dsmi_component_type {     DSMI_COMPONENT_TYPE_NVE,     DSMI_COMPONENT_TYPE_XLOADER,     DSMI_COMPONENT_TYPE_M3FW,     DSMI_COMPONENT_TYPE_UEFI,     DSMI_COMPONENT_TYPE_TEE,     DSMI_COMPONENT_TYPE_KERNEL,     DSMI_COMPONENT_TYPE_DTB,     DSMI_COMPONENT_TYPE_ROOTFS,     DSMI_COMPONENT_TYPE_IMU,     DSMI_COMPONENT_TYPE_IMP,     DSMI_COMPONENT_TYPE_AICPU,     DSMI_COMPONENT_TYPE_MAX,     UPGRADE_AND_RESET_ALL_COMPONENT = 0xFFFFFFFF7,     UPGRADE_ALL_COMPONENT = 0xFFFFFFFFF } DSMI_COMPONENT_TYPE;</div>

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机、容器的root用户下运行。

调用示例

```
int ret = -EINVAL;
unsigned int component_num = 0;
DSMI_COMPONENT_TYPE *component_table = NULL;
ret =dsmi_get_component_count(0, &component_num);
if(ret != 0) {
    //todo: 记录日志
    return ret;
}
component_table = (DSMI_COMPONENT_TYPE*)malloc(sizeof(DSMI_COMPONENT_TYPE) *
component_num);
if(component_table == NULL) {
    //todo: 记录日志
    return ret;
}
ret = dsmi_get_component_list(0, component_table, component_num);
if(ret != 0) {
    //todo: 记录日志
    return ret;
```

```
}  
...
```

# 8 昇腾 AI 处理器复位启动

- 8.1 dsmi\_pre\_reset\_soc
- 8.2 dsmi\_rescan\_soc
- 8.3 dsmi\_get\_device\_boot\_status
- 8.4 dsmi\_pcie\_hot\_reset

## 8.1 dsmi\_pre\_reset\_soc

### 函数原型

```
int dsmi_pre_reset_soc(int device_id)
```

### 功能说明

昇腾AI处理器预复位，发起昇腾AI处理器预复位接口，预复位目的是解除上层驱动及软件对此昇腾AI处理器的依赖。预复位完成后可以对此昇腾AI处理器进行隔离或实际复位操作。

一般用于实际复位的操作流程：dsmi\_pre\_reset\_soc -> reset -> dsmi\_rescan\_soc；支持PCIe标卡。

reset功能是通过BMC（Baseboard Management Controller）带外通道完成，仅支持华为服务器；为保证复位功能正常使用，请升级服务器的BMC到最新版本。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行。  
调用该接口前，请停掉昇腾AI处理器上的所有业务。

调用示例

```
int ret = 0;
ret = dsmi_pre_reset_soc(0);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

8.2 dsmi\_rescan\_soc

函数原型

int dsmi\_rescan\_soc(int device\_id)

功能说明

对指定昇腾AI处理器重新扫描；支持PCIe标卡。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

调用该接口的程序必须在物理机的root用户下运行。

调用示例

```
int ret = 0;
ret = dsmi_rescan_soc(0);
if(ret != 0) {
//todo: 记录日志
return ret;
}
...
```

8.3 dsmi\_get\_device\_boot\_status

函数原型

```
int dsmi_get_device_boot_status(int device_id, enum dsmi_boot_status
*boot_status)
```

功能说明

获取昇腾AI处理器系统的启动状态。  
昇腾310 AI处理器场景下，支持PCIe标卡。

参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
boot_status	输出	昇腾AI处理器的启动状态： enum dsmi_boot_status { DSMI_BOOT_STATUS_UNINIT = 0, /*未初始化*/ DSMI_BOOT_STATUS_BIOS, /* BIOS启动中*/ DSMI_BOOT_STATUS_OS, /* OS启动中*/ DSMI_BOOT_STATUS_FINISH /*启动完成*/ };

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

无。

调用示例

```
int ret = 0;
enum dsmi_boot_status boot_status = 0;
ret = dsmi_get_device_boot_status (0, &boot_status);
if(ret != 0) {
```

```
//todo: 记录日志
return ret;
}
...
```

## 8.4 dsmi\_pcie\_hot\_reset

### 函数原型

```
int dsmi_pcie_hot_reset(int device_id)
```

### 功能说明

昇腾AI处理器带内复位接口，用于发起昇腾AI处理器复位。该接口调用成功表示复位命令执行完毕，并不代表芯片处于可用状态，可通过获取芯片健康状态等接口（如：dsmi\_get\_device\_health）来确保芯片处于可用状态。该接口仅支持PCIe标卡，支持通过创建子进程并行复位所有芯片。

### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，取值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。

### 返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

### 异常处理

无。

### 约束说明

调用该接口的程序必须在物理机的root用户下运行；若在物理机的非root用户，或在容器下运行，则返回权限错误。

调用该接口前，请停掉昇腾AI处理器上的所有业务。

### 调用示例

```
pid_t pid;
int ret;
pid = fork(); //创建子进程运行hot_reset
if (pid < 0)
    printf("fork error\n");
else if (pid == 0) {
    ret = dsmi_pcie_hot_reset(0);
    if (ret != 0) {
```

```
//todo: 记录日志  
return ret;  
}  
...
```

# 9 网关地址管理

## 9.1 dsmi\_get\_gateway\_addr

### 9.1 dsmi\_get\_gateway\_addr

#### 函数原型

```
int dsmi_get_gateway_addr (int device_id, int port_type, int port_id, ip_addr_t *gtw_address)
```

#### 功能说明

查询网关地址，支持PCIe标卡、mini模块（仅支持mini模块作为EP的场景）。  
MiniRC场景下，不支持该接口。

#### 参数说明

参数名	输入/输出	描述
device_id	输入	指定设备号，有效值范围：0~63，当前实际支持的设备号，通过dsmi_list_device接口获取。
port_type	输入	指定网口类型，取值范围： <ul style="list-style-type: none"><li>• VNIC: 0x00</li></ul> 用户在调用接口时可以调用如下宏： DSMI_VNIC_PORT (0)
port_id	输入	指定网口号，保留字段。取值范围： 【 0~255 】。



参数名	输入/输出	描述
gtw_address	输出&输入	<pre>#define DSMI_ARRAY_IPV4_NUM 4 #define DSMI_ARRAY_IPV6_NUM 16  typedef struct ip_addr {     union {         unsigned char ip6[DSMI_ARRAY_IPV6_NUM];         unsigned char ip4[DSMI_ARRAY_IPV4_NUM];     } u_addr;     enum ip_addr_type ip_type; //这是一个输入值 } ip_addr_t;  enum ip_addr_type {     /** IPv4 */     IPADDR_TYPE_V4 = 0U,     /** IPv6 */     IPADDR_TYPE_V6 = 6U,     /** IPv4+IPv6 ("dual-stack") */     IPADDR_TYPE_ANY = 46U };</pre>

返回值

类型	描述
int	处理结果，返回0成功，失败返回错误码。

约束说明

只支持IPV4地址。

调用示例

```
int ret = 0;
int device_id = 0;
int port_type = 0;
int port_id = 0;
ip_addr_t gateway_address={0};
ret = dsmi_get_gateway_addr(device_id, port_type, port_id, &gateway_address);
if(ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

# 10 不支持的接口

如下接口不支持。当调用如下接口时，系统将返回不支持或调用失败的提示信息，返回值信息见[表2-1](#)。

接口名称	功能说明
dsmi_get_emmc_status	查询eMMC控制器状态信息。
dsmi_get_lp_status	获取低功耗的状态信息。
dsmi_get_hiss_status	获取HISS子系统的状态信息。
dsmi_get_can_status	查询指定CAN控制器状态信息。
dsmi_get_can_config	获取CAN的配置信息。
dsmi_get_ufs_status	获取UFS的状态信息。
dsmi_get_ufs_config	获取UFS控制器配置信息。
dsmi_set_ufs_config	设置UFS控制器配置信息。
dsmi_get_sensorhub_status	查询SensorHub状态信息。
dsmi_get_sensorhub_config	获取SensorHub配置信息。
dsmi_get_gpio_status	获取gpio的状态信息。
dsmi_get_softwarefault	获取芯片的硬件错误信息。
dsmi_get_safetyisland_status	获取safetyisland的自身的状态信息。
dsmi_register_fault_event_handler	注册故障回调接口。
dsmi_set_upgrade_attr	通过设置升级接口属性DSMI_UPGRADE_ATTR来调用相应的升级接口功能。
dsmi_set_power_state	设置系统模式。

接口名称	功能说明
dsmi_get_vectorcore_info	获取vectorcore信息。
dsmi_get_uart_status	获取串口的状态信息。
dsmi_set_uart_config	配置串口波特率。
dsmi_set_cpu_nums	设置cpu配比。
dsmi_get_cpu_nums	获取cpu配比。
dsmi_get_hbm_info	查询hbm的频率、容量、利用率信息。
dsmi_get_aicore_info	查询aicore的当前频率与额定频率信息。
dsmi_get_network_health	查询RoCE网卡的IP地址的联通状态。
dsmi_get_llc_perf_para	查询LLC性能参数，包括LLC读命中率、写命中率和吞吐量。
dsmi_get_aicpu_info	查询AICPU的个数、最大运行频率、当前运行频率和利用率。
dsmi_set_sec_revocation	实现密钥吊销功能，当前只支持Soc二级密钥吊销。
dsmi_create_capability_group	创建算力配置信息。
dsmi_delete_capability_group	删除算力分组配置信息。
dsmi_get_capability_group_info	获取算力分组配置信息。
dsmi_set_device_info	设置device的信息的通用接口，对各模块信息进行配置。
dsmi_get_device_info	获取device的信息的通用接口，获取各模块中的状态信息。

# 11 调用示例

各接口说明处的“调用示例”是一个代码示例片段，此处以调用 `dsmi_get_device_health` 接口为例，给出完整的调用示例，说明需要include的头文件（`dsmi_common_interface.h`和`ascend_hal_error.h`）、调用接口的逻辑、返回值的打印等。

您需要参见《[CANN 软件安装指南](#)》部署开发环境，部署完成后，从开发环境的“`/usr/local/Ascend/driver/include`”目录下获取`dsmi_common_interface.h`和`ascend_hal_error.h`文件。

## 说明

`/usr/local/Ascend`表示软件包的默认安装目录，需根据实际情况替换。

## 示例代码文件 `get_device_health.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <unistd.h>
#include "dsmi_common_interface.h"
int main(int argc, char *argv[])
{
    int ret = 0;
    int opt = 0;
    int test_case_num = 0;
    char optstring[20] = "p:s:gr";
    static struct option long_options[] =
    {
        {"process", 1, NULL, 'p'},
        {0, 0, 0, 0}
    };
    while ((opt = getopt_long(argc, argv, optstring, long_options, NULL)) != -1)
    {
        switch (opt)
        {
            case 'p':
            {
                if (argc < 3)
                {
                    printf("process test the para num: %d test_error. Input num should not be smaller than 3.test_fail \n",argc);
                    return -1;
                }
                test_case_num = strtol(argv[2], NULL, 10);
                switch (test_case_num)
                {
```

```
case 1:
{
    printf("begin query health\n");
    unsigned int phealth = 0;
    ret = dsmi_get_device_health(0, &phealth);
    if (ret)
    {
        printf("call dsmi_get_device_health fail, ret = %d\n", ret);
        return -1;
    }
    else
    {
        /*phealth type is unsigned int, printf value should use %u*/
        printf("dsmi_get_device_health success,phealth:%u.\n", phealth);
    }
    break;
}
default:
    break;
}
}
}
}
return 0;
}
```

## Ascend EP 场景下使用调用示例

**步骤1** 以HwHiAiUser用户将get\_device\_health.c、dsmi\_common\_interface.h和ascend\_hal\_error.h上传到Host侧服务器的同一个目录下。

**步骤2** 以HwHiAiUser用户登录到Host侧服务器。

**步骤3** 执行如下命令，编译get\_device\_health.c中的代码，生成可执行文件get\_device\_health。

```
gcc get_device_health.c /usr/local/Ascend/driver/lib64/libdrvdsmi_host.so -L -o get_device_health
```

### 说明

- /usr/local/Ascend表示Driver组件的默认安装路径，请根据实际情况替换。
- Driver组件的安装，请参见《[CANN 软件安装指南](#)》。

**步骤4** 执行可执行文件get\_device\_health。

```
./get_device_health -p 1
```

----结束