Ascend 310B
23.0.RC1

DCMI API 参考

文档版本 01

发布日期 2023-05-08

华为技术有限公司

商标声明


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

前言

概述

本文档详细的描述了DCMI（DaVinci Card Management Interface）接口，用户可使用这些接口进行设备管理、配置管理、芯片复位启动等操作。

适用于Atlas 200I DK A2 开发者套件和Atlas 200I A2 加速模块。

读者对象

本文档主要适用于以下人员：

- 企业管理员
- 企业终端用户

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示如不避免将会导致死亡或严重伤害的具有高等级风险的危害。
警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2023-05-08	第一次正式发布。

目 录

1 简介

DCMI接口包含设备管理接口、配置管理接口、MCU升级控制接口、芯片复位接口以及用户自定义信息配置接口。用户可调用接口完成二次开发。

须知

- 本文“部署场景”中的“Y”表示支持，“N”表示不支持，“NA”表示不涉及。
- 不支持多线程并发使用DCMI接口。
- DSMI接口从1.0.10版本开始废弃，计划于2023年版本不再提供，请使用对应的DCMI接口进行替代。
- 建议用户在对应的执行环境上编译DCMI相关的可执行文件，否则可能会出现glibc版本前后不兼容的情况。

2 设备管理接口

2.1 dcmi_init 接口原型

函数原型

int dcmi_init(void)

功能说明

完成DCMI初始化。

参数说明

无

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

在调用其他DCMI接口之前必须先调用该接口进行初始化。

表 2-1 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
ret = dcmi_init();
...
```

2.2 dcmi_get_dcmi_version 接口原型

函数原型

int dcmi_get_dcmi_version(char *dcmi_ver, unsigned int len)

功能说明

查询dcmi版本。

参数说明

参数名称	输入/输出	类型	描述
dcmi_ver	输出	char *	用户申请的空间，用于存放返回的版本号。
len	输入	unsigned int	dcmi_ver空间的长度，长度至少为16。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-2 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
char dcmi_ver[16] = {0};
ret = dcmi_get_dcmi_version(dcmi_ver, sizeof(dcmi_ver));
...
```

2.3 dcmi_get_driver_version 接口原型

函数原型

int dcmi_get_driver_version(char *driver_ver, unsigned int len)

功能说明

查询驱动版本。

参数说明

参数名称	输入/输出	类型	描述
driver_ver	输出	char *	用户申请的空间，用于存放返回的版本号，大小为64Byte。
len	输入	unsigned int	driver_ver空间的长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-3 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
char driver_ver[64] = {0};
ret = dcmi_get_driver_version(driver_ver, sizeof(driver_ver));
...
```

2.4 dcmi_get_version 接口原型

函数原型

int dcmi_get_version(int card_id, int device_id, char *verison_str, unsigned int version_len, int *len)

功能说明

查询驱动版本。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
version_str	输出	char *	返回驱动版本。 该空间由用户申请。

参数名称	输入/输出	类型	描述
version_len	输入	unsigned int	version_str空间的大小，长度至少为64。
len	输出	int *	返回版本号长度。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.3 dcmi_get_driver_version接口原型。

表 2-4 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

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

2.5 dcmi_get_card_list 接口原型

函数原型

int dcmi_get_card_list(int *card_num, int *card_list, int list_len)

功能说明

获取NPU单元个数和NPU单元ID编号。

参数说明

参数名称	输入/输出	类型	描述
card_num	输出	int *	NPU管理单元个数。最大支持64个。
card_list	输出	int *	NPU管理单元的ID列表。
list_len	输入	int	参数card_list数组的长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-5 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
```

```
int ret;
int card_num = 0;
int card_list[16] = {0};
ret = dcmi_get_card_list(&card_num, card_list, 16);
...
```

2.6 dcmi_get_card_num_list 接口原型

函数原型

int dcmi_get_card_num_list(int *card_num, int *card_list, int list_len)

功能说明

获取NPU单元个数和NPU单元ID编号。

参数说明

参数名称	输入/输出	类型	描述
card_num	输出	int	NPU管理单元个数。最大支持64个。
card_list	输出	int	NPU管理单元的ID列表。
list_len	输入	int	输出参数card_list的数组长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**2.5 dcmi_get_card_list接口原型**。

表 2-6 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

Y	Y	Y

调用示例

```
...
int ret;
int device_count = 0;
int card_id_list[16];
ret = dcmi_get_card_num_list(&device_count, card_id_list, 16);
...
```

2.7 dcmi_get_device_num_in_card 接口原型

函数原型

int dcmi_get_device_num_in_card(int card_id, int *device_num)

功能说明

查询指定NPU管理单元的上的NPU芯片数量。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_num	输出	int *	NPU芯片数量。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-7 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int device_num = 0;
int card_id = 0;
ret = dcmi_get_device_num_in_card(card_id, &device_num);
...
```

2.8 dcmi_get_device_id_in_card 接口原型

函数原型

int dcmi_get_device_id_in_card(int card_id, int *device_id_max, int *mcu_id, int *cpu_id)

功能说明

查询指定NPU管理单元上的芯片数量、MCU ID和CPU ID。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id_max	输出	int *	NPU管理单元中NPU芯片最大个数。
mcu_id	输出	int *	NPU管理单元中MCU的ID。取值为-1，表示无MCU。
cpu_id	输出	int *	NPU管理单元中控制CPU的ID。取值为-1，表示无控制CPU。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-8 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int device_id_max = 0;
int mcu_id = 0;
int cpu_id = 0;
int card_id = 0;
ret = dcmi_get_device_id_in_card(card_id, &device_id_max, &mcu_id, &cpu_id);
...
```

2.9 dcmi_get_device_type 接口原型

函数原型

int dcmi_get_device_type(int card_id, int device_id, enum dcmi_unit_type *device_type)

功能说明

查询设备类型。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

device_type	输出	enum dcmi_unit_type *	设备类型，目前支持如下几种：
			enum dcmi_unit_type {
			ASCEND_TYPE = 0,//昇腾芯片
			MCU_TYPE = 1,//控制MCU（Multipoint Control Unit）
			CPU_TYPE = 2,//控制CPU（Central Processing Unit）
			INVALID_TYPE = 0xFF
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-9 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int device_id = 0;
enum dcmi_unit_type device_type = INVALID_TYPE;
ret = dcmi_get_device_type(card_id, device_id, &device_type);
...
```

2.10 dcmi_get_device_chip_info 接口原型

函数原型

int dcmi_get_device_chip_info(int card_id, int device_id, struct dcmi_chip_info *chip_info)

功能说明

获取指定设备的chip信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

chip_info	输出	struct dcmi_chip_info *	芯片信息。
			struct dcmi_chip_info {
			unsigned char chip_type[MAX_CHIP_NAME_LEN];
			unsigned char chip_name[MAX_CHIP_NAME_LEN];
			unsigned char chip_ver[MAX_CHIP_NAME_LEN];
			unsigned int aicore_cnt;
			};
			其中MAX_CHIP_NAME_LEN为32
			输入device_id为NPU芯片ID时，aicore_cnt无效。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-10 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int device_id = 0;
struct dcmi_chip_info chip_info = {0};
ret = dcmi_get_device_chip_info(card_id, device_id, &chip_info);
...
```

2.11 dcmi_get_device_pcie_info 接口原型

函数原型

int dcmi_get_device_pcie_info(int card_id, int device_id, struct dcmi_pcie_info *pcie_info)

功能说明

获取指定设备的PCIe（Peripheral Component Interconnect Express）信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

pcie_info	输出	struct dcmi_pcie_info *	PCIe信息
			struct dcmi_pcie_info {
			unsigned int deviceid;
			unsigned int venderid;
			unsigned int subvenderid;
			unsigned int subdeviceid;
			unsigned int bdf_deviceid;
			unsigned int bdf_busid;
			unsigned int bdf_funcid;
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-11 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret;
int card_id = 0;
int device_id = 0;
struct dcmi_pcie_info pcie_info = {0};
ret = dcmi_get_device_pcie_info(card_id, device_id, &pcie_info);
...
```

2.12 dcmi_get_device_pcie_info_v2 接口原型

函数原型

int dcmi_get_device_pcie_info_v2(int card_id, int device_id, struct dcmi_pcie_info_all *pcie_info)

功能说明

获取指定设备的PCIe信息，包括PCIe domain域信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

pcie_info	输出	struct dcmi_pcie_info_all *	PCIe信息
			struct dcmi_pcie_info_all {
			unsigned int venderid;
			unsigned int subvenderid;
			unsigned int deviceid;
			unsigned int subdeviceid;
			int domain;
			unsigned int bdf_busid;
			unsigned int bdf_deviceid;
			unsigned int bdf_funcid;
			unsigned char reserve[32];
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-12 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret;
int card_id = 0;
int device_id = 0;
struct dcmi_pcie_info_all pcie_info = {0};
ret = dcmi_get_device_pcie_info_v2(card_id, device_id, &pcie_info);
...
```

2.13 dcmi_get_pcie_info 接口原型

函数原型

int dcmi_get_pcie_info(int card_id, int device_id, struct dcmi_tag_pcie_idinfo *pcie_idinfo)

功能说明

查询芯片的PCIe（Peripheral Component Interconnect Express）信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

pcie_idinfo	输出	struct dcmi_tag_pcie_idinfo *	PCIe信息。
			struct dcmi_tag_pcie_idinfo{
			unsigned int deviceid;
			unsigned int venderid;
			unsigned int subvenderid;
			unsigned int subdeviceid;
			unsigned int bdf_deviceid;
			unsigned int bdf_busid;
			unsigned int bdf_funcid;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.11 dcmi_get_device_pcie_info接口原型。

表 2-13 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

N	N	N

调用示例

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

2.14 dcmi_get_device_board_info 接口原型

函数原型

int dcmi_get_device_board_info(int card_id, int device_id, struct dcmi_board_info *board_info)

功能说明

获取指定设备的board信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

board_info	输出	struct dcmi_board_info*	board信息。
			struct dcmi_board_info {
			unsigned int board_id;
			unsigned int pcb_id;
			unsigned int bom_id;
			unsigned int slot_id;
			};
			输入device_id为NPU芯片的id时，只有board_id和slot_id有效，其中slot_id为芯片在卡上位置标识。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-14 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int device_id = 0;
struct dcmi_board_info board_info = {0};
ret = dcmi_get_device_board_info(card_id, device_id, &board_info);
...
```

2.15 dcmi_get_board_info 接口原型

函数原型

int dcmi_get_board_info(int card_id, int device_id, struct dcmi_board_info_stru *board_info)

功能说明

获取指定设备的board信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

board_info	输出	struct dcmi_board_info_stru*	board信息。
			struct dcmi_board_info_stru {
			unsigned int board_id;
			unsigned int pcb_id;
			unsigned int bom_id;
			unsigned int slot_id;
			};
			输入device_id为NPU芯片的id时，只有board_id和slot_id有效，其中slot_id为芯片在卡上位置标识。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**2.14 dcmi_get_device_board_info接口原型**。

表 2-15 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_board_info_stru board_info = {0};
ret = dcmi_get_board_info(card_id, device_id, &board_info);
...
```

2.16 dcmi_get_device_elabel_info 接口原型

函数原型

int dcmi_get_device_elabel_info(int card_id, int device_id, struct dcmi_elabel_info *elabel_info)

功能说明

获取设备的电子标签信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

elabel_info	输出	struct dcmi_elabel_info *	电子标签信息。
			struct dcmi_elabel_info {
			char product_name[MAX_LENTH];
			char model[MAX_LENTH];
			char manufacturer[MAX_LENTH];
			char manufacturer_date[MAX_LENTH];
			char serial_number[MAX_LENTH];
			};
			其中manufacturer_date字段当前预留，此字段无效。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-16 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
struct dcmi_elabel_info elabel_info = {0};
int ret = 0;
int card_id = 0;
int device_id = 0;
ret = dcmi_get_device_elabel_info(card_id, device_id, &elabel_info);
...
```

2.17 dcmi_get_device_power_info 接口原型

函数原型

int dcmi_get_device_power_info(int card_id, int device_id, int *power)

功能说明

查询设备功耗。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。

参数名称	输入/输出	类型	描述
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
power	输出	int*	设备功耗：单位为0.1W。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-17 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
int ret = 0;
int card_id = 0;
int device_id = 0;
int power = 0;
ret = dcmi_get_device_power_info(card_id, device_id, &power);
...
```

2.18 dcmi_get_device_die_v2 接口原型

函数原型

int dcmi_get_device_die_v2(int card_id, int device_id, enum dcmi_die_type input_type, struct dcmi_die_id *die_id)

功能说明

获取指定设备的DIE ID。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

input_type	输入	enum dcmi_die_type	die类型
			enum dcmi_die_type {
			NDIE,
			VDIE
			};
			不支持NDIE。

die_id	输出	struct dcmi_die_id *	die id信息
			struct dcmi_die_id {
			unsigned int soc_die[DIE_ID_COUNT];
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-18 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
enum dcmi_die_type input_type = NDIE;
struct dcmi_die_id die_id = {0};
ret = dcmi_get_device_die_v2(card_id, device_id, input_type, &die_id);
...
```

2.19 dcmi_get_device_die 接口原型

函数原型

int dcmi_get_device_die(int card_id, int device_id, struct dcmi_soc_die_stru *device_die)

功能说明

获取指定设备的VDIE ID。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

device_die	输出	struct dcmi_soc_die_stru *	返回DIE结构体信息：
			struct dcmi_soc_die_stru{
			unsigned int soc_die[5];
			}

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.18 dcmi_get_device_die_v2接口原型。

表 2-19 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_soc_die_stru pdevice_die = {0};
ret = dcmi_get_device_die(card_id, device_id, &pdevice_die);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.20 dcmi_get_device_ndie 接口原型

函数原型

int dcmi_get_device_ndie(int card_id, int device_id, struct dsmi_soc_die_stru *device_ndie)

功能说明

获取指定设备的NDIE ID。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

device_ndie	输出	struct dsmi_soc_die_stru *	返回NDIE结构体信息：
			struct dsmi_soc_die_stru {
			unsigned int soc_die[5];
			}

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.18 dcmi_get_device_die_v2接口原型。

表 2-20 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0x3;
```



```
int device_id = 0;
struct dsmi_soc_die_stru device_ndie = {0};
ret = dcmi_get_device_ndie(card_id, device_id, &device_ndie);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.21 dcmi_get_device_health 接口原型

函数原型

int dcmi_get_device_health(int card_id, int device_id, unsigned int *health)

功能说明

查询芯片的总体健康状态。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
health	输出	unsigned int *	设备总体健康状态，只代表本部件，不包括与本部件存在逻辑关系的其它部件，内容定义为： ● 0：正常 ● 1：一般告警 ● 2：重要告警 ● 3：紧急告警 ● 0xFFFFFFFF：该设备不存在

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

如果有多个告警，以最严重的告警作为设备的健康状态。

表 2-21 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned int health = 0;
ret = dcmi_get_device_health(card_id, device_id, &health);
...
```

2.22 dcmi_get_driver_health 接口原型

函数原型

int dcmi_get_driver_health(unsigned int *health)

功能说明

获取驱动的健康状态。

参数说明

参数名称	输入/输出	类型	描述
health	输出	unsigned int *	驱动健康状态的指针。 **health**的值以十六进制形式显示时，健康状态值为： ● 0：正常 ● 1：一般告警（预留） ● 2：重要告警（预留） ● 3：紧急告警 ● ffffffff：该设备不存在或者未启动。（预留）

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-22 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

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

2.23 dcmi_get_device_errorcode_v2 接口原型

函数原型

int dcmi_get_device_errorcode_v2(int card_id, int device_id, int *error_count, unsigned int *error_code_list, unsigned int list_len)

功能说明

查询设备故障码。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
errorcount	输出	int *	错误码数量，取值范围：0~128。
error_code_list	输出	unsigned int *	错误码列表。 若打印信息中提示有错误码，请参考对应产品的《黑匣子错误码信息列表》进行查看。
list_len	输入	unsigned int	error_code_list空间大小。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-23 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
#define ERROR_CODE_MAX_NUM              (128)
...
```

```
int ret = 0;
int card_id = 0;
int device_id = 0;
int errorcount = 0;
unsigned int error_code_list[ERROR_CODE_MAX_NUM] = {0};
ret = dcmi_get_device_errorcode_v2(card_id, device_id, &errorcount, error_code_list,
ERROR_CODE_MAX_NUM);
...
```

2.24 dcmi_get_device_errorcode 接口原型

函数原型

int dcmi_get_device_errorcode(int card_id, int device_id, int *error_count, unsigned int *error_code, int *error_width)

功能说明

查询设备故障码。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。
errorcount	输出	int *	错误码数量，取值范围：0~128。
error_code	输出	unsigned int *	错误码。数组长度至少为128。 若打印信息中提示有错误码，请参考对应产品的《黑匣子错误码信息列表》进行查看。
errorwidth	输出	int *	每个错误码占用的字节空间。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.23 dcmi_get_device_errorcode_v2接口原型。

表 2-24 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

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

2.25 dcmi_get_driver_errorcode 接口原型

函数原型

int dcmi_get_driver_errorcode(int *error_count, unsigned int *error_code_list, unsigned int list_len)

功能说明

查询驱动故障码。

参数说明

参数名称	输入/输出	类型	描述
error_count	输出	int *	错误码数量，取值范围：0~128。

参数名称	输入/输出	类型	描述
error_code_list	输出	unsigned int *	错误码列表。 若打印信息中提示有错误码，请参考对应产品的《黑匣子错误码信息列表》。
list_len	输入	unsigned int	error_code_list空间大小。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-25 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
#define ERROR_CODE_MAX_NUM              (128)
...
int ret = 0;
int error_count = 0;
unsigned int error_code_list[DCMI_ERROR_CODE_MAX_COUNT] = {0};
ret = dcmi_get_driver_errorcode(&error_count, error_code_list, ERROR_CODE_MAX_NUM);
...
```

2.26 dcmi_get_device_errorcode_string 接口原型

函数原型

int dcmi_get_device_errorcode_string(int card_id, int device_id, unsigned int error_code, unsigned char *error_info, int buf_size)

功能说明

查询设备故障描述。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
errorcode	输入	unsigned int	要查询的错误码，通过**dcmi_get_device_errorcode_v2**接口获取。
error_info	输出	unsigned char *	对应的错误描述。
buf_size	输入	int	传入的error_info大小，固定为48字节。若设置的buffer大小大于48字节，则默认用48字节。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-26 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
#define ERROR_CODE_MAX_NUM  (128)
#define BUF_SIZE            (48)
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int errorcount = 0;
unsigned int error_code_list[ERROR_CODE_MAX_NUM] = {0};
unsigned char error_info[BUF_SIZE] = {0};
ret = dcmi_get_device_errorcode_v2(card_id, device_id, &errorcount, error_code_list,
ERROR_CODE_MAX_NUM);
if ((ret != 0) || (errorcount == 0)){
    //todo:记录日志
    return ret;
}
ret = dcmi_get_device_errorcode_string(card_id, device_id, error_code_list[0], error_info, BUF_SIZE);
if (ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

2.27 dcmi_get_device_errorinfo 接口原型

函数原型

int dcmi_get_device_errorinfo(int card_id, int device_id, int errorcode, unsigned char *errorinfo, int buf_size)

功能说明

查询设备故障描述。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
errorcode	输入	int	要查询的错误码，通过**dcmi_get_device_errorcode_v2**接口获取。
errorinfo	输出	unsigned char *	对应的错误字符描述。
buf_size	输入	int	传入的errorinfo大小，固定为48字节。若设置的buffer大小大于48字节，则默认用48字节。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.26 dcmi_get_device_errorcode_string接口原型。

表 2-27 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
#define ERROR_CODE_MAX_NUM   (128)
#define BUF_SIZE          (48)
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int errorcount = 0;
int error_code_width = 0;
unsigned char errorinfo[BUF_SIZE] = {0};
unsigned char errorcode [ERROR_CODE_MAX_NUM] = {0};
ret = dcmi_get_device_errorcode_v2(card_id, device_id, &errorcount, errorcode, &error_code_width);
if ((ret != 0) || (errorcount == 0)){
    //todo:记录日志
    return ret;
}
ret = dcmi_get_device_errorinfo(card_id, device_id, errorcode[0], errorinfo, BUF_SIZE);
if (ret != 0){
    //todo:记录日志
    return ret;
}
...
```

2.28 dcmi_get_device_flash_count 接口原型

函数原型

int dcmi_get_device_flash_count(int card_id, int device_id, unsigned int *flash_count)

功能说明

获取设备中Flash个数。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
flash_count	输出	unsigned int *	返回Flash个数。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-28 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
unsigned int flash_count = 0;
ret = dcmi_get_device_flash_count(card_id, device_id, &flash_count);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.29 dcmi_get_device_flash_info_v2 接口原型

函数原型

int dcmi_get_device_flash_info_v2(int card_id, int device_id, unsigned int flash_index, struct dcmi_flash_info *flash_info)

功能说明

获取设备内Flash的信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。
flash_index	输入	unsigned int	Flash索引号，通过dcmi_get_device_flash_count获取。取值范围：[0, flash_count-1]

参数名称	输入/输出	类型	描述

flash_info	输出	struct dcmi_flash_info *	返回Flash信息。
			Flash信息结构体定义：
			struct dcmi_flash_info {
			unsigned long long flash_id;
			unsigned short device_id;
			unsigned short vendor;
			unsigned int state; /*flash health, 0x8表示正常，0x10表示非正常*/
			unsigned long long size;
			unsigned int sector_count;
			unsigned short manufacturer_id;
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-29 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
int i;
int ret = 0;
int card_id = 0x3;
int device_id = 0;
struct dcmi_flash_info flash_info = {0};
```



```
unsigned int flash_count = 0;
ret = dcmi_get_device_flash_count(card_id, device_id,&flash_count);
if (ret != 0){
    //todo：记录日志
    return ret;
}
for (int i = 0; i < flash_count; i++){
    ret = dcmi_get_device_flash_info_v2(card_id, device_id, i, &flash_info);
    if(ret != 0){
        //todo：记录日志
        return ret;
    }
}
...
```

2.30 dcmi_get_device_flash_info 接口原型

函数原型

int dcmi_get_device_flash_info(int card_id, int device_id, unsigned int flash_index, struct dcmi_flash_info_stru *flash_info)

功能说明

获取芯片内Flash的信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。
flash_index	输入	unsigned int	Flash索引号，通过dcmi_get_device_flash_count获取。取值范围：[0, flash_count-1]

参数名称	输入/输出	类型	描述

flash_info	输出	struct dcmi_flash_info_stru *	返回Flash信息。
			Flash信息结构体定义：
			struct dcmi_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, 0x8表示正常，0x10表示非正常*/
			unsigned long long size; /* total size in bytes */
			unsigned int sector_count; /* number of erase units */
			unsigned short manufacturer_id; /* manufacturer id */
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**2.29 dcmi_get_device_flash_info_v2接口原型**。

表 2-30 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

Y	Y	Y

调用示例

```
...
int i;
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_flash_info_stru flash_info = {0};
unsigned int flash_count = 0;
ret = dcmi_get_device_flash_count(card_id, device_id,, &flash_count);
...
For (i = 0; i < flash_count; i++){
    ret = dcmi_get_device_flash_info(card_id, device_id, i, &flash_info);
    if (ret != 0){
        //todo：记录日志
        return ret;
    }
...
}
...
```

2.31 dcmi_get_device_aicore_info 接口原型

函数原型

int dcmi_get_device_aicore_info(int card_id, int device_id, struct dcmi_aicore_info *aicore_info)

功能说明

查询aicore信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

aicore_info	输出	struct dcmi_aicore_info*	返回aicore信息，信息结构体：
			struct dcmi_aicore_info {
			unsigned int freq; //额定频率，单位是MHz
			unsigned int cur_freq;//当前频率单位是MHz
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-31 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
int ret = 0;
int card_id = 0x3;
int device_id = 0;
struct dcmi_aicore_info aicore_info = {0};
ret = dcmi_get_device_aicore_info(card_id, device_id, &aicore_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.32 dcmi_get_aicore_info 接口原型

函数原型

int dcmi_get_aicore_info(int card_id, int device_id, struct dsmi_aicore_info_stru *aicore_info)

功能说明

获取aicore信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

aicore_info	输出	struct dsmi_aicore_info_stru *	返回aicore信息，信息结构体：
			struct dsmi_aicore_info_stru {
			unsigned int freq; // normal freq (MHz)
			unsigned int curfreq; // current freq (MHz)
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.31 dcmi_get_device_aicore_info接口原型。

表 2-32 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
struct dsmi_aicore_info_stru aicore_info = {0};
ret = dcmi_get_aicore_info(card_id, device_id, &aicore_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.33 dcmi_get_device_aicpu_info 接口原型

函数原型

int dcmi_get_device_aicpu_info(int card_id, int device_id, struct dcmi_aicpu_info *aicpu_info)

功能说明

查询AICPU信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

aicpu_info	输出	struct dcmi_aicpu_info *	AICPU信息，信息结构体：
			struct dcmi_aicpu_info {
			unsigned int max_freq;
			unsigned int cur_freq;
			unsigned int aicpu_num;
			unsigned int util_rate[MAX_CORE_NUM];
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-33 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0x3;
int device_id = 0;
struct dcmi_aicpu_info aicpu_info = {0};
ret = dcmi_get_device_aicpu_info(card_id, device_id, &aicpu_info);
if (ret){
    //todo : 记录日志
    return ERROR;
}
...
```

2.34 dcmi_get_aicpu_info 接口原型

函数原型

int dcmi_get_aicpu_info(int card_id, int device_id, struct dsmi_aicpu_info_stru *aicpu_info)

功能说明

获取aicpu相关信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

aicpu_info	输出	struct dsmi_aicpu_info_stru *	struct dsmi_aicpu_info_stru{
			unsigned int maxFreq; // AICPU的最大运行频率，单位是MHz
			unsigned int curFreq; // AICPU的当前运行频率，单位是MHz
			unsigned int aicpuNum; //AICPU的个数
			unsigned int utilRate [TAISHAN_CORE_NUM]; //AICPU的利用率
			}

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.33 dcmi_get_device_aicpu_info接口原型。

表 2-34 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dsmi_aicpu_info_stru aicpu_info = {0};
ret = dcmi_get_aicpu_info(card_id, dev_id, &aicpu_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.35 dcmi_get_device_system_time 接口原型

函数原型

int dcmi_get_device_system_time(int card_id, int device_id, unsigned int *time)

功能说明

查询芯片的系统时间。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
time	输出	unsigned int *	表示从1970年1月1日00:00:00至今的秒数值。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-35 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
unsigned int time = 0;
int card_id = 0x3;
int device_id = 0;
ret = dcmi_get_device_system_time(card_id, device_id, &time);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.36 dcmi_get_system_time 接口原型

函数原型

int dcmi_get_system_time(int card_id, int device_id, unsigned int *time)

功能说明

查询芯片系统时间。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
time	输出	unsigned int *	表示从1970年1月1日00:00:00至今的秒数值。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.35 dcmi_get_device_system_time接口原型。

表 2-36 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
unsigned int time = 0;
int card_id = 0x3;
int device_id = 0;
ret = dcmi_get_system_time(card_id, device_id, &time);
```

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

2.37 dcmi_get_device_temperature 接口原型

函数原型

int dcmi_get_device_temperature(int card_id, int device_id, int *temperature)

功能说明

查询设备温度。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
temperature	输出	int *	芯片温度：单位为摄氏度。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-37 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int temperature = 0;
ret = dcmi_get_device_temperature(card_id, device_id, &temperature);
...
```

2.38 dcmi_get_device_voltage 接口原型

函数原型

int dcmi_get_device_voltage(int card_id, int device_id, unsigned int *voltage)

功能说明

查询设备电压。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
voltage	输出	unsigned int *	芯片电压：精度为0.01V。转换为V的计算公式：value=voltage*0.01。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-38 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned int voltage = 0;
ret = dcmi_get_device_voltage(card_id, device_id, &voltage);
...
```

2.39 dcmi_get_device_pcie_error_cnt 接口原型

函数原型

int dcmi_get_device_pcie_error_cnt(int card_id, int device_id, struct dcmi_chip_pcie_err_rate *pcie_err_code_info)

功能说明

查询芯片的PCIe 链路误码信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

pcie_err_code_info	输出	struct dcmi_chip_pcie_err_rate *	struct dcmi_chip_pcie_err_rate {
			unsigned int reg_deskew_fifo_overflow_intr_status; //是否发生deskew_fifo溢出：1表示已发生，0表示未发生。
			unsigned int reg_symbol_unlock_intr_status;//是否发生symbol_unlock事件：1表示已发生，0表示未发生。
			unsigned int reg_deskew_unlock_intr_status;//是否发生deskew_unlock事件：1表示已发生，0表示未发生。
			unsigned int reg_phystatus_timeout_intr_status;//是否发生phystatus超时事件：1表示已发生，0表示未发生。
			unsigned int symbol_unlock_counter;
			unsigned int pcs_rx_err_cnt;
			unsigned int phy_lane_err_counter;
			unsigned int pcs_rcv_err_status;//PCS层接收错误状态，每bit映射到每个使用的通道：1表示有错误，0表示正常。
			unsigned int symbol_unlock_err_status;// symbol_unlock标志，每bit映射到每个使用的通道：1表示有错误，0表示正常。
			unsigned int phy_lane_err_status;//lane错误，每bit映射到每个使用的通道：1表示有错误，0表示正常。
			unsigned int dl_lcrc_err_num;
			unsigned int dl_dcrc_err_num;
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

设备重新启动后请先清除芯片的PCIe链路误码信息。

表 2-39 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_chip_pcie_err_rate pcie_err_code_info = {0};
ret = dcmi_get_device_pcie_error_cnt(card_id, device_id, &pcie_err_code_info);
…
```

2.40 dcmi_get_pcie_error_cnt 接口原型

函数原型

int dcmi_get_pcie_error_cnt(int card_id, int device_id, struct dcmi_chip_pcie_err_rate_stru *pcie_err_code_info)

功能说明

查询芯片的PCIe（Peripheral Component Interconnect Express）链路误码信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

pcie_err_code_info	输出	struct dcmi_chip_pcie_err_rate_stru *	struct dcmi_chip_pcie_err_rate_stru {
			unsigned int reg_deskew_fifo_overflow_intr_status;
			//是否发生deskew_fifo溢出：1表示已发生，0表示未发生。
			unsigned int reg_symbol_unlock_intr_status;
			//是否发生symbol_unlock事件：1表示已发生，0表示未发生。
			unsigned int reg_deskew_unlock_intr_status;
			//是否发生deskew_unlock事件：1表示已发生，0表示未发生。
			unsigned int reg_phystatus_timeout_intr_status;
			//是否发生phystatus超时事件：1表示已发生，0表示未发生。
			unsigned int symbol_unlock_counter;
			unsigned int pcs_rx_err_cnt;
			unsigned int phy_lane_err_counter;
			unsigned int pcs_rcv_err_status;//PCS层接收错误状态，每bit映射到每个使用的通道：1表示有错误，0表示正常。
			unsigned int symbol_unlock_err_status;
			//symbol_unlock标志，每bit映射到每个使用的通道：1表示有错误，0表示正常。
			unsigned int phy_lane_err_status;
			//lane错误，每bit映射到每个使用的通道：1表示有错误，0表示正常。
			unsigned int dl_lcrc_err_num;
			unsigned int dl_dcrc_err_num;
			}

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.39 dcmi_get_device_pcie_error_cnt接口原型。

表 2-40 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_chip_pcie_err_rate_stru pcie_err_code_info = {0};
ret = dcmi_get_pcie_error_cnt(card_id, device_id, &pcie_err_code_info);
if (ret != 0){
    //todo:记录日志
    return ret;
}
...
```

2.41 dcmi_get_device_ecc_info 接口原型

函数原型

int dcmi_get_device_ecc_info(int card_id, int device_id, enum dcmi_device_type input_type, struct dcmi_ecc_info *device_ecc_info)

功能说明

获取ECC信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。

参数名称	输入/输出	类型	描述
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

device_type	输入	enum dcmi_device_type	组件类型
			enum dcmi_device_type {
			DCMI_DEVICE_TYPE_DDR,
			DCMI_DEVICE_TYPE_SRAM,
			DCMI_DEVICE_TYPE_HBM,
			DCMI_DEVICE_TYPE_NPU,
			DCMI_HBM_RECORDED_SINGLE_ADDR,
			DCMI_HBM_RECORDED_MULTI_ADDR,
			DCMI_DEVICE_TYPE_NONE = 0xff
			};
			目前支持DCMI_DEVICE_TYPE_DDR。

device_ecc_info	输出	struct dcmi_ecc_info *	返回ECC结构体信息：
			struct dcmi_ecc_info {
			int enable_flag;
			unsigned int single_bit_error_cnt;
			unsigned int double_bit_error_cnt;
			unsigned int total_single_bit_error_cnt;
			unsigned int total_double_bit_error_cnt;
			unsigned int single_bit_isolated_pages_cnt;
			unsigned int double_bit_isolated_pages_cnt;
			};
			其中，enable_flag输出0，表示ecc检测未使能；enable_flag输出1，表示ecc检测使能。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-41 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	N

调用示例

```
...
int ret = 0;
int card_id = 1;
int device_id = 0;
struct dcmi_ecc_info device_ecc_info = {0};
ret = dcmi_get_device_ecc_info(card_id, device_id, DCMI_DEVICE_TYPE_DDR, &device_ecc_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.42 dcmi_get_ecc_info 接口原型

函数原型

int dcmi_get_ecc_info(int card_id, int device_id, int device_type, struct dsmi_ecc_info_stru *pdevice_ecc_info)

功能说明

获取ECC信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

参数名称	输入/输出	类型	描述
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

device_type	输入	int	组件类型
			enum dcmi_device_type {
			DCMI_DEVICE_TYPE_DDR,
			DCMI_DEVICE_TYPE_SRAM,
			DCMI_DEVICE_TYPE_HBM,
			DCMI_DEVICE_TYPE_NPU,
			DCMI_HBM_RECORDED_SINGLE_ADDR,
			DCMI_HBM_RECORDED_MULTI_ADDR,
			DCMI_DEVICE_TYPE_NONE = 0xff
			};
			目前支持DCMI_DEVICE_TYPE_DDR。

pdevice_ecc_info	输出	struct dsmi_ecc_info_stru *	返回ECC结构体信息：
			struct dsmi_ecc_info_stru {
			int enable_flag;（0或1，分别代表禁用和使能）
			unsigned int single_bit_error_count;
			unsigned int double_bit_error_count;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**2.41 dcmi_get_device_ecc_info接口原型**。

表 2-42 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dsmi_ecc_info_stru device_ecc_info = {0};
ret = dcmi_get_ecc_info(card_id, device_id, DCMI_DEVICE_TYPE_DDR, &device_ecc_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.43 dcmi_get_device_frequency 接口原型

函数原型

int dcmi_get_device_frequency(int card_id, int device_id, enum dcmi_freq_type input_type, unsigned int *frequency)

功能说明

获取设备的频率。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
input_type	输入	enum dcmi_freq_type	设备类型，数值和具体设备类型对应如下： ● 1：内存 ● 2：控制CPU ● 6：HBM ● 7：AI CORE当前频率 ● 9：AI CORE额定频率 ● 12 ： Vector Core当前频率 当前支持1、2、7、9这几种类型。 **说明** AI CORE额定频率：表示AI CORE在TDP功耗和场景下，能够持续运行的频率。
frequency	输出	unsigned int *	频率，单位为MHz。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-43 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
```



```
int device_id = 0;
unsigned int frequency = 0;
ret = dcmi_get_device_frequency(card_id, device_id, DCMI_FREQ_DDR, &frequency);
...
```

2.44 dcmi_get_device_hbm_info 接口原型

函数原型

int dcmi_get_device_hbm_info(int card_id, int device_id, struct dcmi_hbm_info *hbm_info)

功能说明

获取芯片的HBM内存信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

hbm_info	输出	struct dcmi_hbm_info*	返回HBM信息，HBM信息结构体：
			struct dcmi_hbm_info {
			unsigned long long memory_size;
			unsigned int freq;
			unsigned long long memory_usage;
			int temp;
			unsigned int bandwith_util_rate;
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-44 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_hbm_info device_hbm_info = {0};
ret = dcmi_get_device_hbm_info(card_id, device_id, &device_hbm_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.45 dcmi_get_hbm_info 接口原型

函数原型

int dcmi_get_hbm_info(int card_id, int device_id, struct dsmi_hbm_info_stru *pdevice_hbm_info)

功能说明

获取芯片的HBM内存信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

pdevice_hbm_info	输出	struct dsmi_hbm_info_stru*	返回HBM信息，HBM信息结构体：
			struct dsmi_hbm_info_stru {
			unsigned long long memory_size; /**< HBM total size, KB */
			unsigned int freq; /**< HBM freq, MHz */
			unsigned long long memory_usage; /**< HBM memory_usage, KB */
			int temp; /**< HBM temperature */
			unsigned int bandwith_util_rate;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.44 dcmi_get_device_hbm_info接口原型。

表 2-45 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
```



```
struct dsmi_hbm_info_stru device_hbm_info = {0};
ret = dcmi_get_hbm_info(card_id, device_id, &device_hbm_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.46 dcmi_get_device_memory_info_v3 接口原型

函数原型

int dcmi_get_device_memory_info_v3(int card_id, int device_id, struct dcmi_get_memory_info_stru *memory_info)

功能说明

获取芯片的内存信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

memory_info	输出	struct dcmi_get_memory_info_stru*	返回内存信息，内存信息结构体：
			struct dcmi_get_memory_info_stru {
			unsigned long long memory_size; /* unit:MB */
			unsigned long long memory_available; /* free + hugepages_free * hugepagesize * /
			unsigned int freq;
			unsigned long hugepagesize; /* unit:KB */
			unsigned long hugepages_total;
			unsigned long hugepages_free;
			unsigned int utiliza; /* ddr memory info usages */
			unsigned char reserve[60]; /* the size of dcmi_memory_info is 96 */
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-46 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_get_memory_info_stru memory_info = {0};
ret = dcmi_get_device_memory_info_v3(card_id, device_id, &memory_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.47 dcmi_get_device_memory_info_v2 接口原型

函数原型

int dcmi_get_device_memory_info_v2(int card_id, int device_id, struct dcmi_memory_info *memory_info)

功能说明

获取芯片的内存信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

memory_info	输出	struct dcmi_memory_info*	返回内存信息，内存信息结构体：
			struct dcmi_memory_info {
			unsigned long long memory_size; // 单位 MB
			unsigned int freq;
			unsigned int utiliza;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**2.46 dcmi_get_device_memory_info_v3接口原型**。

表 2-47 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_memory_info memory_info = {0};
ret = dcmi_get_device_memory_info_v2(card_id, device_id, &memory_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.48 dcmi_get_memory_info 接口原型

函数原型

int dcmi_get_memory_info(int card_id, int device_id, struct dcmi_memory_info_stru *pdevice_memory_info)

功能说明

获取芯片的内存信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定卡编号，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

pdevice_memory_info	输出	struct
		dcmi_memory_info_stru *
			返回内存信息，内存信息结构体：
			struct dcmi_memory_info_stru {
			unsigned long long memory_size; // 单位 MB
			unsigned int freq;
			unsigned int utiliza;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.46 dcmi_get_device_memory_info_v3接口原型。

表 2-48 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
struct dcmi_memory_info_stru pdevice_memory_info = {0};
ret = dcmi_get_memory_info(card_id, device_id, &pdevice_memory_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.49 dcmi_get_device_utilization_rate 接口原型

函数原型

int dcmi_get_device_utilization_rate(int card_id, int device_id, int input_type, unsigned int *utilization_rate)

功能说明

获取芯片占用率。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
input_type	输入	int	设备类型，数值和具体设备类型对应如下： ● 1：内存 ● 2：AI CORE ● 3：AI CPU ● 4：控制CPU ● 5：内存带宽 ● 6：HBM ● 8：DDR ● 10：HBM带宽 ● 12：vector core 当前支持1、2、3、4、5、12这几种类型。
utilization_rate	输出	unsigned int *	处理器利用率，单位：%。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-49 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
unsigned int utilization_rate = 0;
int input_type = DCMI_UTILIZATION_RATE_DDR;
ret = dcmi_get_device_utilization_rate(card_id, device_id, input_type, &utilization_rate);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.50 dcmi_get_device_sensor_info 接口原型

函数原型

int dcmi_get_device_sensor_info(int card_id, int device_id, enum dcmi_manager_sensor_id sensor_id, union dcmi_sensor_info *sensor_info)

功能说明

获取设备的传感器信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

sensor_id	输入	enum dcmi_manager_sensor_id	enum dcmi_manager_sensor_id {
			DCMI_CLUSTER_TEMP_ID = 0,
			DCMI_PERI_TEMP_ID = 1,
			DCMI_AICORE0_TEMP_ID,
			DCMI_AICORE1_TEMP_ID,
			DCMI_AICORE_LIMIT_ID,
			DCMI_AICORE_TOTAL_PER_ID,
			DCMI_AICORE_ELIM_PER_ID,
			DCMI_AICORE_BASE_FREQ_ID,
			DCMI_NPU_DDR_FREQ_ID,
			DCMI_THERMAL_THRESHOLD_ID,
			DCMI_NTC_TEMP_ID,
			DCMI_SOC_TEMP_ID,
			DCMI_FP_TEMP_ID,
			DCMI_N_DIE_TEMP_ID,
			DCMI_HBM_TEMP_ID,
			DCMI_SENSOR_INVALID_ID = 255
			};
			指定传感器索引，具体如下值：
			0：DCMI_CLUSTER_TEMP_ID， 表示CLUSTER温度；返回值对应输出联合体中的uchar成员；
			1：DCMI_PERI_TEMP_ID，表示PERI温度；返回值对应输出联合体中的uchar成员；
			2：DCMI_AICORE0_TEMP_ID, 表示AICORE0温度；返回值对应输出联合体中的uchar成员；
			3：DCMI_AICORE1_TEMP_ID，表示AICORE1温度；返回值对应输出联合体中的uchar成员；
			4：DCMI_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；
			5：DCMI_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；
			6：DCMI_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；

参数名称	输入/输出	类型	描述
			7：DCMI_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； 8：DCMI_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； 9：DCMI_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； 10：DCMI_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。 11：DCMI_SOC_TEMP_ID，表示SOC最高温；返回值对应输出联合体中的uchar成员； 12： DCMI_FP_TEMP_ID， 表示光模块最高温度；返回值对应输出联合体中的signed int iint成员； 13： DCMI_N_DIE_TEMP_ID， 表示N_DIE温度；返回值对应输出联合体中的signed int iint成员； 14： DCMI_HBM_TEMP_ID， 表示HBM最高温度；返回值对应输出联合体中的signed int iint成员。 该场景支持0、1、2、7、8、11。

sensor_info	输出	union dcmi_sensor_info *	返回传感器结构体信息：
			union dcmi_sensor_info {
			unsigned char uchar;
			unsigned short ushort;
			unsigned int uint;
			signed int iint;
			signed char temp[DCMI_SENSOR_TEMP_LEN];
			signed int ntc_tmp[DCMI_SENSOR_NTC_TEMP_LEN];
			unsigned int data[DCMI_SENSOR_DATA_MAX_LEN];
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-50 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
enum dcmi_manager_sensor_id sensor_id = DCMI_CLUSTER_TEMP_ID;
union dcmi_sensor_info sensor_info = {0};
ret = dcmi_get_device_sensor_info(card_id, device_id, sensor_id, &sensor_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.51 dcmi_get_soc_sensor_info 接口原型

函数原型

int dcmi_get_soc_sensor_info(int card_id, int device_id, int sensor_id, union tag_sensor_info *sensor_info)

功能说明

获取设备的传感器信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

sensor_id	输入	int	enum dcmi_manager_sensor_id {
			DCMI_CLUSTER_TEMP_ID = 0,
			DCMI_PERI_TEMP_ID = 1,
			DCMI_AICORE0_TEMP_ID,
			DCMI_AICORE1_TEMP_ID,
			DCMI_AICORE_LIMIT_ID,
			DCMI_AICORE_TOTAL_PER_ID,
			DCMI_AICORE_ELIM_PER_ID,
			DCMI_AICORE_BASE_FREQ_ID,
			DCMI_NPU_DDR_FREQ_ID,
			DCMI_THERMAL_THRESHOLD_ID,
			DCMI_NTC_TEMP_ID,
			DCMI_SOC_TEMP_ID,
			DCMI_FP_TEMP_ID,
			DCMI_N_DIE_TEMP_ID,
			DCMI_HBM_TEMP_ID,
			DCMI_SENSOR_INVALID_ID = 255
			};
			指定传感器索引，具体如下值：
			0：DCMI_CLUSTER_TEMP_ID， 表示CLUSTER温度；返回值对应输出联合体中的uchar成员；
			1：DCMI_PERI_TEMP_ID，表示PERI温度；返回值对应输出联合体中的uchar成员；
			2：DCMI_AICORE0_TEMP_ID, 表示AICORE0温度；返回值对应输出联合体中的uchar成员；
			3：DCMI_AICORE1_TEMP_ID，表示AICORE1温度；返回值对应输出联合体中的uchar成员；
			4：DCMI_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；
			5：DCMI_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；
			6：DCMI_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；

参数名称	输入/输出	类型	描述
			7：DCMI_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； 8：DCMI_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； 9：DCMI_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； 10：DCMI_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。 11：DCMI_SOC_TEMP_ID，表示SOC最高温；返回值对应输出联合体中的uchar成员； 12： DCMI_FP_TEMP_ID， 表示光模块最高温度；返回值对应输出联合体中的signed int iint成员； 13： DCMI_N_DIE_TEMP_ID， 表示N_DIE温度；返回值对应输出联合体中的signed int iint成员； 14： DCMI_HBM_TEMP_ID， 表示HBM最高温度；返回值对应输出联合体中的signed int iint成员。 该场景支持0、1、2、7、8、11。

sensor_info	输出	union tag_sensor_info *	返回温度传感器结构体信息：
			union tag_sensor_info {
			unsigned char uchar;
			unsigned short ushort;
			unsigned int uint;
			signed int iint;
			signed char temp[2];
			signed int ntc_tmp[4];
			unsigned int data[16];
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.50 dcmi_get_device_sensor_info接口原型。

表 2-51 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
union tag_sensor_info sensor_info = {0};
int sensor_id = 1;
ret = dcmi_get_soc_sensor_info(card_id, device_id, sensor_id, &sensor_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.52 dcmi_set_container_service_enable 接口原型

函数原型

int dcmi_set_container_service_enable(void)

功能说明

初始化当前容器所有设备，用于后续逻辑id与物理id之间转换。

参数说明

无。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-52 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

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

2.53 dcmi_get_device_board_id 接口原型

函数原型

int dcmi_get_device_board_id(int card_id, int device_id, unsigned int *board_id)

功能说明

获取Board ID。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定卡的编号，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
board_id	输出	unsigned int*	单板的ID。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-53 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int board_id = 0;
ret = dcmi_get_device_board_id(card_id, device_id, &board_id);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.54 dcmi_get_board_id 接口原型

函数原型

int dcmi_get_board_id(int card_id, int device_id, int *board_id)

功能说明

获取指定设备的Board ID信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
board_id	输出	int *	Board ID信息。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**2.53 dcmi_get_device_board_id接口原型**。

表 2-54 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int board_id = 0;
ret = dcmi_get_board_id(card_id, device_id, &board_id);
...
```

2.55 dcmi_get_device_component_count 接口原型

函数原型

int dcmi_get_device_component_count(int card_id, int device_id, unsigned int *component_count)

功能说明

获取可升级组件的个数。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
component_count	输出	unsigned int *	返回组件的个数。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-55 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	Y

调用示例

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

2.56 dcmi_get_device_component_list 接口原型

函数原型

int dcmi_get_device_component_list(int card_id, int device_id, enum dcmi_component_type *component_table, unsigned int component_count)

功能说明

获取可升级组件列表。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
component_table	输出	enum dcmi_component_type *	返回可升级组件列表，具体值如下： enum dcmi_component_type { DCMI_COMPONENT_TYPE_NVE, DCMI_COMPONENT_TYPE_XLOADER, DCMI_COMPONENT_TYPE_M3FW, DCMI_COMPONENT_TYPE_UEFI, DCMI_COMPONENT_TYPE_TEE, DCMI_COMPONENT_TYPE_KERNEL, DCMI_COMPONENT_TYPE_DTB, DCMI_COMPONENT_TYPE_ROOTFS, DCMI_COMPONENT_TYPE_IMU, DCMI_COMPONENT_TYPE_IMP, DCMI_COMPONENT_TYPE_AICPU, DCMI_COMPONENT_TYPE_HBOOT1_A, DCMI_COMPONENT_TYPE_HBOOT1_B, DCMI_COMPONENT_TYPE_HBOOT2, DCMI_COMPONENT_TYPE_DDR, DCMI_COMPONENT_TYPE_LP, DCMI_COMPONENT_TYPE_HSM, DCMI_COMPONENT_TYPE_SAFETY_ISLAND, DCMI_COMPONENT_TYPE_HILINK, DCMI_COMPONENT_TYPE_RAWDATA, DCMI_COMPONENT_TYPE_SYSDRV, DCMI_COMPONENT_TYPE_ADSAPP, DCMI_COMPONENT_TYPE_COMISOLATOR, DCMI_COMPONENT_TYPE_CLUSTER, DCMI_COMPONENT_TYPE_CUSTOMIZED, DCMI_COMPONENT_TYPE_SYS_BASE_CONFIG, DCMI_COMPONENT_TYPE_RECOVERY, DCMI_COMPONENT_TYPE_HILINK2, DCMI_COMPONENT_TYPE_LOGIC_BIST, DCMI_COMPONENT_TYPE_MEMORY_BIST, DCMI_COMPONENT_TYPE_ATF,

参数名称	输入/输出	类型	描述
			DCMI_COMPONENT_TYPE_USER_BASE_CONFIG, DCMI_COMPONENT_TYPE_BOOTROM, DCMI_COMPONENT_TYPE_MAX, DCMI_UPGRADE_AND_RESET_ALL_COMPONENT = 0xFFFFFFF7, DCMI_UPGRADE_ALL_IMAGE_COMPONENT = 0xFFFFFFFD, DCMI_UPGRADE_ALL_FIRMWARE_COMPONENT = 0xFFFFFFFE, DCMI_UPGRADE_ALL_COMPONENT = 0xFFFFFFFF }; 当前仅支持: DCMI_COMPONENT_TYPE_AICPU, DCMI_COMPONENT_TYPE_HBOOT1_A, DCMI_COMPONENT_TYPE_HBOOT1_B, DCMI_COMPONENT_TYPE_HBOOT2， DCMI_COMPONENT_TYPE_HSM， DCMI_COMPONENT_TYPE_HILINK， DCMI_COMPONENT_TYPE_SYS_BASE_CONFIG， DCMI_COMPONENT_TYPE_ATF， DCMI_COMPONENT_TYPE_USER_BASE_CONFIG
component_count	输入	unsigned int	“component_table”数组的长度。

返回值

类型	描述
int	处理结果: • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-56 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned int component_num = 0;
enum dcmi_component_type *component_table = NULL;

ret = dcmi_get_device_component_count(card_id, device_id, &component_num);
if (ret != 0) {
    // todo：记录日志
    return ret;
}

component_table = (enum dcmi_component_type *)malloc(sizeof(enum dcmi_component_type) *
component_num);
if (component_table == NULL) {
    // todo：记录日志
    return ret;
}

ret = dcmi_get_device_component_list(card_id, device_id, component_table, component_num);
if (ret != 0) {
    // todo：记录日志
    free(component_table);
    return ret;
}
...
```

2.57 dcmi_get_device_component_static_version 接口原型

函数原型

int dcmi_get_device_component_static_version(int card_id, int device_id, enum dcmi_component_type component_type, unsigned char *version_str, unsigned int len)

功能说明

查询静态组件版本。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
component_type	输入	enum dcmi_component_type	固件类型 enum dcmi_component_type { DCMI_COMPONENT_TYPE_NVE, DCMI_COMPONENT_TYPE_XLOADER, DCMI_COMPONENT_TYPE_M3FW, DCMI_COMPONENT_TYPE_UEFI, DCMI_COMPONENT_TYPE_TEE, DCMI_COMPONENT_TYPE_KERNEL, DCMI_COMPONENT_TYPE_DTB, DCMI_COMPONENT_TYPE_ROOTFS, DCMI_COMPONENT_TYPE_IMU, DCMI_COMPONENT_TYPE_IMP, DCMI_COMPONENT_TYPE_AICPU, DCMI_COMPONENT_TYPE_HBOOT1_A, DCMI_COMPONENT_TYPE_HBOOT1_B, DCMI_COMPONENT_TYPE_HBOOT2, DCMI_COMPONENT_TYPE_DDR, DCMI_COMPONENT_TYPE_LP, DCMI_COMPONENT_TYPE_HSM, DCMI_COMPONENT_TYPE_SAFETY_ISLAND, DCMI_COMPONENT_TYPE_HILINK, DCMI_COMPONENT_TYPE_RAWDATA, DCMI_COMPONENT_TYPE_SYSDRV, DCMI_COMPONENT_TYPE_ADSAPP, DCMI_COMPONENT_TYPE_COMISOLATOR, DCMI_COMPONENT_TYPE_CLUSTER, DCMI_COMPONENT_TYPE_CUSTOMIZED, DCMI_COMPONENT_TYPE_SYS_BASE_CONFIG, DCMI_COMPONENT_TYPE_RECOVERY, DCMI_COMPONENT_TYPE_HILINK2, DCMI_COMPONENT_TYPE_LOGIC_BIST, DCMI_COMPONENT_TYPE_MEMORY_BIST, DCMI_COMPONENT_TYPE_ATF,

参数名称	输入/输出	类型	描述
			DCMI_COMPONENT_TYPE_USER_BASE_CONFIG, DCMI_COMPONENT_TYPE_BOOTROM, DCMI_COMPONENT_TYPE_MAX, DCMI_UPGRADE_AND_RESET_ALL_COMPONENT = 0xFFFFFFF7, DCMI_UPGRADE_ALL_IMAGE_COMPONENT = 0xFFFFFFFD, DCMI_UPGRADE_ALL_FIRMWARE_COMPONENT = 0xFFFFFFFE, DCMI_UPGRADE_ALL_COMPONENT = 0xFFFFFFFF }; 当前仅支持: DCMI_COMPONENT_TYPE_AICPU, DCMI_COMPONENT_TYPE_HBOOT1_A, DCMI_COMPONENT_TYPE_HBOOT1_B, DCMI_COMPONENT_TYPE_HBOOT2， DCMI_COMPONENT_TYPE_HSM， DCMI_COMPONENT_TYPE_HILINK， DCMI_COMPONENT_TYPE_SYS_BASE_CONFIG， DCMI_COMPONENT_TYPE_ATF， DCMI_COMPONENT_TYPE_USER_BASE_CONFIG
version_str	输出	unsigned char *	用户申请的空间，存放返回的固件版本号。
len	输入	unsigned int	version_str的内存大小，大小不能小于64Byte。

返回值

类型	描述
int	处理结果: • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-57 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned char version_str[64] = {0};
ret = dcmi_get_device_component_static_version(card_id, device_id,
DCMI_COMPONENT_TYPE_NVE,version_str, 64);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.58 dcmi_get_device_cgroup_info 接口原型

函数原型

int dcmi_get_device_cgroup_info(int card_id, int device_id, struct dcmi_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。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

cg_info	输出	struct dcmi_cgroup_info*	cgroup信息。
			struct dcmi_cgroup_info {
			unsigned long limit_in_bytes;
			unsigned long max_usage_in_bytes;
			unsigned long usage_in_bytes;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-58 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_cgroup_info cg_info= {0};
ret = dcmi_get_device_cgroup_info(card_id, device_id, &cg_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.59 dcmi_get_device_llc_perf_para 接口原型

函数原型

int dcmi_get_device_llc_perf_para(int card_id, int device_id, struct dcmi_llc_perf *perf_para)

功能说明

查询LLC性能参数，包括LLC读命中率、写命中率和吞吐量。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

perf_para	输出	struct dcmi_llc_perf*	LLC性能参数信息，包括LLC读命中率、写命中率和吞吐量。
			struct dcmi_llc_perf {
			unsigned int wr_hit_rate; // LLC写命中率，单位是%（百分比）
			unsigned int rd_hit_rate; // LLC读命中率，单位是%（百分比）
			unsigned int throughput; // LLC吞吐量，单位是KB/s
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-59 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_llc_perf perf_para = {0};
ret = dcmi_get_device_llc_perf_para(card_id, device_id, &perf_para);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.60 dcmi_set_device_info 接口原型

函数原型

int dcmi_set_device_info(int card_id, int device_id, enum dcmi_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。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

main_cmd	输入	enum dcmi_main_cmd	模块cmd信息，执行用于获取对应模块的信息
			enum dcmi_main_cmd {
			DCMI_MAIN_CMD_DVPP = 0,
			DCMI_MAIN_CMD_ISP,
			DCMI_MAIN_CMD_TS_GROUP_NUM,
			DCMI_MAIN_CMD_CAN,
			DCMI_MAIN_CMD_UART,
			DCMI_MAIN_CMD_UPGRADE = 5,
			DCMI_MAIN_CMD_UFS,
			DCMI_MAIN_CMD_OS_POWER,
			DCMI_MAIN_CMD_LP,
			DCMI_MAIN_CMD_MEMORY,
			DCMI_MAIN_CMD_RECOVERY,
			DCMI_MAIN_CMD_TS,
			DCMI_MAIN_CMD_CHIP_INF,
			DCMI_MAIN_CMD_QOS,
			DCMI_MAIN_CMD_SOC_INFO,
			DCMI_MAIN_CMD_SILS,
			DCMI_MAIN_CMD_HCCS,
			DCMI_MAIN_CMD_TEMP = 50,
			DCMI_MAIN_CMD_SVM,
			DCMI_MAIN_CMD_VDEV_MNG,
			DCMI_MAIN_CMD_SEC,
			DCMI_MAIN_CMD_EX_COMPUTING = 0x8000,
			DCMI_MAIN_CMD_DEVICE_SHARE = 0x8001,
			DCMI_MAIN_CMD_MAX
			};
			Atlas 200I A2 加速模块当前支持DCMI_MAIN_CMD_DEVICE_SHARE和DCMI_MAIN_CMD_LP。
			Atlas 200I DK A2 开发者套件当前仅支持DCMI_MAIN_CMD_DEVICE_SHARE，详细可参见**3.16 dcmi_set_device_share_enable接口原型**。

参数名称	输入/输出	类型	描述

sub_cmd	输入	unsigned int	子命令，只支持取0。
			0表示默认值。
			/* DCMI sub commond for Low power */
			typedef enum {
			DCMI_LP_SUB_CMD_SET_WORK_TOPS = 7, // 设置工作档位
			DCMI_LP_SUB_CMD_GET_WORK_TOPS = 8, // 获取当前工作档位
			DCMI_LP_SUB_CMD_MAX,
			} DCMI_LP_SUB_CMD;

buf	输入	const void *	用于配置相应设备的配置信息。
buf_size	输入	unsigned int	buf数组的长度。

表 2-60 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_LP_SUB_CMD_SET_WORK_TOPS	DCMI_LP_WORK_TOPS_STRU	sizeof(DCMI_LP_WORK_TOPS_STRU)	#define DCMI_LP_WORK_TOPS_RESERVE 32 typedef struct dcmi_lp_work_tops_stru { unsigned int work_tops; unsigned int is_in_flash; unsigned char reserve[DCMI_LP_WORK_TOPS_RESERVE]; } DCMI_LP_WORK_TOPS_STRU; work_tops的值来源于DCMI_LP_SUB_CMD_TOPS_DETAILS查询结果中的档位列表。 is_in_flash设置非0时，会写入到flash；否则不会写入flash中。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-61 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int buf = 0;
unsigned int size = sizeof(int);
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_DVPP, 0, &buf, size);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.61 dcmi_get_device_info 接口原型

函数原型

int dcmi_get_device_info(int card_id, int device_id, enum dcmi_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。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

main_cmd	输入	enum dcmi_main_cmd	指定查询项对应主命令字。
			模块cmd信息，执行用于获取对应模块的信息
			enum dcmi_main_cmd {
			DCMI_MAIN_CMD_DVPP = 0,
			DCMI_MAIN_CMD_ISP,
			DCMI_MAIN_CMD_TS_GROUP_NUM,
			DCMI_MAIN_CMD_CAN,
			DCMI_MAIN_CMD_UART,
			DCMI_MAIN_CMD_UPGRADE = 5,
			DCMI_MAIN_CMD_UFS,
			DCMI_MAIN_CMD_OS_POWER,
			DCMI_MAIN_CMD_LP,
			DCMI_MAIN_CMD_MEMORY,
			DCMI_MAIN_CMD_RECOVERY,
			DCMI_MAIN_CMD_TS,
			DCMI_MAIN_CMD_CHIP_INF,
			DCMI_MAIN_CMD_QOS,
			DCMI_MAIN_CMD_SOC_INFO,
			DCMI_MAIN_CMD_SILS,
			DCMI_MAIN_CMD_HCCS,
			DCMI_MAIN_CMD_TEMP = 50,
			DCMI_MAIN_CMD_SVM,
			DCMI_MAIN_CMD_VDEV_MNG,
			DCMI_MAIN_CMD_SEC,
			DCMI_MAIN_CMD_EX_COMPUTING = 0x8000,
			DCMI_MAIN_CMD_DEVICE_SHARE = 0x8001,
			DCMI_MAIN_CMD_MAX
			};
			Atlas 200I A2 加速模块当前支持DCMI_MAIN_CMD_DEVICE_SHARE，DCMI_MAIN_CMD_LP，DCMI_MAIN_CMD_MEMORY，DCMI_MAIN_CMD_DVPP和DCMI_MAIN_CMD_TS。
			Atlas 200I DK A2 开发者套件当前仅支持DCMI_MAIN_CMD_DEVICE_SHARE，DCMI_MAIN_CMD_MEMORY，DCMI_MAIN_CMD_DVPP和DCMI_MAIN_CMD_TS，详细可参见

参数名称	输入/输出	类型	描述
			3.17 dcmi_get_device_share_enable 接口原型。

sub_cmd	输入	unsigned int	指定查询项对应子命令字。
			/* DCMI sub commond for Low power */
			typedef enum {
			DCMI_LP_SUB_CMD_SET_WORK_TOPS = 7, // 设置工作档位
			DCMI_LP_SUB_CMD_GET_WORK_TOPS = 8, // 获取当前工作档位
			DCMI_LP_SUB_CMD_MAX,
			} DCMI_LP_SUB_CMD;

buf	输出	void *	用于接收设备信息的返回值。
size	输入/输出	unsigned int *	buf数组的输入/输出长度。

表 2-62 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	返回值说明
DCMI_LP_SUB_CMD_GET_WORK_TOPS	DCMI_LP_CUR_TOPS_STRU	sizeof(DCMI_LP_CUR_TOPS_STRU)	typedef struct dcmi_lp_cur_tops_stru { unsigned int work_tops; unsigned int tops_nums; } DCMI_LP_CUR_TOPS_STRU; work_tops为当前工作档位。 tops_nums代表工作档位的总数。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-63 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int buf = 0;
unsigned int size = sizeof(int);
unsigned int sub_cmd = 0;
ret = dcmi_get_device_info(card_id, device_id, DCMI_MAIN_CMD_DVPP, sub_cmd, &buf, &size);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.62 dcmi_set_device_sec_revocation 接口原型

函数原型

int dcmi_set_device_sec_revocation(int card_id, int device_id, enum dcmi_revo_type input_type, const unsigned char *file_data, unsigned int file_size)

功能说明

实现密钥吊销功能。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

input_type	输入	enum dcmi_revo_type	吊销类型。
			enum dcmi_revo_type {
			DCMI_REVOCATION_TYPE_SOC = 0, /* for SOC revocation */
			DCMI_REVOCATION_TYPE_CMS_CRL = 1, /* for MDC CMS CRL file upgrade */
			DCMI_REVOCATION_TYPE_CMS_CRL_EXT = 2, /* for extended CRL upgrade */
			DCMI_REVOCATION_TYPE_MAX
			};

file_data	输入	const unsigned char *	吊销文件的数据地址。
file_size	输入	unsigned int	吊销文件的数据长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

- 密钥吊销操作是不可逆的过程，吊销操作执行成功后，无法再进行恢复，需要谨慎使用。
- 该接口在确定需要进行对应的吊销操作时才可以调用，并且需要正确的吊销文件才可以吊销成功，否则，调用该接口返回失败。
- 执行吊销操作成功后，设备不可用。

表 2-64 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
#define REVOCATION_FILE_LEN  544
int card_id = 0;
int dev_id = 0;
int ret = 0;
int dev_count = 0;
unsigned char revocation_file_buf[REVOCATION_FILE_LEN] = {0};
unsigned int buf_size = REVOCATION_FILE_LEN;
ret = dcmi_set_device_sec_revocation(card_id, dev_id, DCMI_REVOCATION_TYPE_SOC, (const unsigned char
*)revocation_file_buf, buf_size);
if (ret != 0){
    // todo:记录日志
    return ret;
}
...
```

2.63 dcmi_get_device_mac_count 接口原型

函数原型

int dcmi_get_device_mac_count(int card_id, int device_id, int *count)

功能说明

查询MAC地址数量。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
count	输出	int *	查询出MAC数，取值范围：0~4。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-65 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

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

2.64 dcmi_get_device_network_health 接口原型

函数原型

int dcmi_get_device_network_health(int card_id, int device_id, enum dcmi_rdfx_detect_result *result)

功能说明

查询RoCE网卡的IP地址的连通状态。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

result	输出	enum dcmi_rdfx_detect_result *	查询RoCE网卡的IP地址的连通状态，内容定义为：
			enum dcmi_rdfx_detect_result {
			DCMI_RDFX_DETECT_OK = 0,
			DCMI_RDFX_DETECT_SOCK_FAIL = 1,
			DCMI_RDFX_DETECT_RECV_TIMEOUT = 2,
			DCMI_RDFX_DETECT_UNREACH = 3,
			DCMI_RDFX_DETECT_TIME_EXCEEDED = 4,
			DCMI_RDFX_DETECT_FAULT = 5,
			DCMI_RDFX_DETECT_INIT = 6,
			DCMI_RDFX_DETECT_THREAD_ERR = 7,
			DCMI_RDFX_DETECT_IP_SET = 8,
			DCMI_RDFX_DETECT_MAX = 0xFF
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-66 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
enum dcmi_rdfx_detect_result health = DCMI_RDFX_DETECT_MAX;
int card_id = 0;
int device_id = 0;
ret = dcmi_get_device_network_health(card_id, device_id, &health);
...
```

2.65 dcmi_get_device_logic_id 接口原型

函数原型

int dcmi_get_device_logic_id(int *device_logic_id, int card_id, int device_id)

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。

参数说明

参数名称	输入/输出	类型	描述
device_logic_id	输出	int*	昇腾AI处理器的逻辑ID。
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-67 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int device_logic_id = 0;
int card_id = 0;
int device_id = 0;
ret = dcmi_get_device_logic_id(&device_logic_id, card_id, device_id);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.66 dcmi_get_device_fan_count 接口原型

函数原型

int dcmi_get_device_fan_count(int card_id, int device_id, int *count)

功能说明

获取Device上小风扇数。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
count	输出	int *	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： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-68 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

2.67 dcmi_get_device_fan_speed 接口原型

函数原型

int dcmi_get_device_fan_speed(int card_id, 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。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
fan_id	输入	int	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	int *	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： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-69 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int count = 0;
int card_id = 0;
int device_id = 0;
int speed;
ret = dcmi_get_device_fan_count(card_id, device_id, &count);
...
for (i = 1; i <= count; i++){
    speed = 0;
    ret = dcmi_get_device_fan_speed(card_id, device_id, i, &speed);
    if (ret != 0){
        //todo
        return ret;
    }
...
}
...
```

2.68 dcmi_get_card_elabel_v2 接口原型

函数原型

int dcmi_get_card_elabel_v2(int card_id, struct dcmi_elabel_info *elabel_info)

功能说明

获取NPU管理单元的电子标签信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

参数名称	输入/输出	类型	描述

elabel_info	输出	struct dcmi_elabel_info *	#define MAX_LENTH 256
			struct dcmi_elabel_info {
			char product_name[MAX_LENTH];
			char model[MAX_LENTH];
			char manufacturer[MAX_LENTH];
			char manufacturer_date[MAX_LENTH];
			char serial_number[MAX_LENTH];
			};
			其中manufacturer_date字段当前预留，此字段无效。
			该接口在容器中不支持查询Serial Number。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-70 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
struct dcmi_elabel_info elableInfo;
int ret = 0;
int card_id = 0;
memset(&elableInfo, 0, sizeof(elableInfo));
ret = dcmi_get_card_elabel_v2(card_id, &elableInfo);
```

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

2.69 dcmi_get_card_elabel 接口原型

函数原型

int dcmi_get_card_elabel(int card_id, struct dcmi_elabel_info_stru *elabel_info)

功能说明

获取NPU管理单元的电子标签信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

elabel_info	输出	struct dcmi_elabel_info_stru*	电子标签信息
			#define MAX_LENTH 256
			struct dcmi_elabel_info_stru {
			char product_name[MAX_LENTH];
			char model[MAX_LENTH];
			char manufacturer[MAX_LENTH];
			char serial_number[MAX_LENTH];
			}
			该接口在容器中不支持查询Serial Number。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用2.68 dcmi_get_card_elabel_v2接口原型。

表 2-71 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
struct dcmi_elabel_info_stru elableInfo;
int ret = 0;
int card_id = 0;
memset(&elableInfo, 0, sizeof(elableInfo));
ret = dcmi_get_card_elabel(card_id, &elableInfo);
if (ret != 0) {
    //todo:记录日志
    return ERROR;
}
...
```

2.70 dcmi_mcu_get_chip_temperature 接口原型

函数原型

int dcmi_mcu_get_chip_temperature(int card_id, char *data_info, int buf_size, int *data_len)

功能说明

查询NPU卡设备温度。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

参数名称	输入/输出	类型	描述
data_info	输出	char *	BYTE[0]：温度传感器个数 BYTE[1:8]：温度传感器1名称 BYTE[9:10]：温度传感器1的温度值 … BYTE[10n-9:10n-2]：温度传感器n名称 BYTE[10n-1:10n]：温度传感器n的温度值。
buf_size	输入	int	data_info空间的最大长度。
data_len	输出	int *	输出数据长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-72 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int enable_flag = 0;
char data_info[256] = {0};
int data_len = 0;
ret = dcmi_mcu_get_chip_temperature(card_id, data_info, sizeof(data_info), &data_len);
if (ret != 0) {
    //todo:记录日志
    return ERROR;
```



```
}
...
```

2.71 dcmi_get_device_ssh_enable 接口原型

函数原型

int dcmi_get_device_ssh_enable(int card_id, int device_id, int *enable_flag)

功能说明

获取设备ssh使能状态。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
enable_flag	输出	int *	ssh使能状态：分为禁用、使能 ● 0：禁用 ● 1：使能

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-73 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int enable_flag = 0;
ret = dcmi_get_device_ssh_enable(card_id, device_id, &enable_flag);
if (ret != 0) {
    //todo:记录日志
    return ERROR;
}
...
```

2.72 dcmi_get_card_board_info 接口原型

函数原型

int dcmi_get_card_board_info(int card_id, struct dcmi_board_info *board_info)

功能说明

获取指定NPU管理单元的board信息，包括board id，pcb id，bom id，slot_id信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

board_info	输出	struct dcmi_board_info*	board信息
			struct dcmi_board_info {
			unsigned int board_id;
			unsigned int pcb_id;
			unsigned int bom_id;
			unsigned int slot_id;
			};
			pcb_id、bom_id、slot_id为无效值。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-74 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
struct dcmi_board_info board_info = {0};
ret = dcmi_get_card_board_info(card_id, &board_info);
...
```

2.73 dcmi_get_card_pcie_info 接口原型

函数原型

int dcmi_get_card_pcie_info(int card_id, char *pcie_info, int pcie_info_len)

功能说明

查询指定NPU管理单元的PCIe信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

参数名称	输入/输出	类型	描述
pcie_info	输出	char *	获取的PCIe信息。
pcie_info_len	输入	int	pcie_info长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-75 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
char pcie_info[256] = {0};
ret = dcmi_get_card_pcie_info(card_id, pcie_info, sizeof(pcie_info));
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.74 dcmi_get_card_pcie_slot 接口原型

函数原型

int dcmi_get_card_pcie_slot(int card_id, int *pcie_slot)

功能说明

查询指定NPU管理单元的PCIe slot ID。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
pcie_slot	输出	int *	pcie slot id

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-76 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

2.75 dcmi_get_fault_device_num_in_card 接口原型

函数原型

int dcmi_get_fault_device_num_in_card(int card_id, int *device_num)

功能说明

查询指定NPU管理单元中故障芯片数量。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_num	输出	int *	故障的芯片数量。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-77 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
```



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

2.76 dcmi_mcu_check_i2c 接口原型

函数原型

int dcmi_mcu_check_i2c(int card_id, int *health_status, int buf_size)

功能说明

查询NPU与MCU之间的IIC通道是否正常。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
health_status	输出	int *	IIC通道状态： Fault，OK，Unknown
buf_size	输入	int	health_status空间长度,长度至少为6。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-78 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int health_status[6] = {0};
int size = 6;
ret = dcmi_mcu_check_i2c(card_id, &health_status, size);
if(ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

2.77 dcmi_mcu_collect_log 接口原型

函数原型

int dcmi_mcu_collect_log(int card_id, int log_type)

功能说明

收集MCU的日志。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
log_type	输入	int	日志类型 ● 0：错误日志 ● 1：操作日志 ● 2：维护日志

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-79 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

2.78 dcmi_get_device_chip_slot 接口原型

函数原型

int dcmi_get_device_chip_slot(int card_id, int device_id, int *chip_pos_id)

功能说明

查询指定芯片在卡上的位置标识信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
chip_pos_id	输出	int *	芯片在卡上的位置标识信息。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-80 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int chip_pos_id = 0;
ret = dcmi_get_device_chip_slot(card_id, device_id, &chip_pos_id);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

2.79 dcmi_get_product_type 接口原型

函数原型

int dcmi_get_product_type(int card_id, int device_id, char *product_type_str, int buf_size)

功能说明

查询指定设备的产品类型。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
product_type_str	输出	char *	用户申请的空间，存放返回的产品类型。
buf_size	输入	int	product_type_str空间的长度，长度至少为32。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-81 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
char driver_ver[16] = {0};
int card_id = 0;
int device_id = 0;
char product_type_str[64] = {0};
int buf_size = 64;
ret = dcmi_get_product_type(card_id, device_id, product_type_str, buf_size);
...
```

2.80 dcmi_get_device_outband_channel_state 接口原型

函数原型

int dcmi_get_device_outband_channel_state(int card_id, int device_id, int *channel_state)

功能说明

查询带外通道状态。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	NPU管理单元的card id。通过dcmi_get_card_num_list接口获取
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。
channel_state	输出	int *	带外通道状态。 1：通道正常 0：通道异常

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-82 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret;
int card_id = 0;
int device_id = 0;
int state = 0;
ret = dcmi_get_device_outband_channel_state(card_id, device_id, &state);
...
```

2.81 dcmi_mcu_get_board_info 接口原型

函数原型

int dcmi_mcu_get_board_info(int card_id, struct dcmi_board_info *pboard_info)

功能说明

获取指定NPU管理单元的board信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

pboard_info	输出	struct dcmi_board_info*	board信息
			struct dcmi_board_info{
			unsigned int board_id;
			unsigned int pcb_id;
			unsigned int bom_id;
			unsigned int slot_id;
			};
			其中slot_id为无效值。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-83 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

2.82 dcmi_mcu_get_power_info 接口原型

函数原型

int dcmi_mcu_get_power_info(int card_id, int *power)

功能说明

查询设备功耗。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
power	输出	int*	功耗。单位为0.1W。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-84 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

2.83 dcmi_get_computing_power_info 接口原型

函数原型

int dcmi_get_computing_power_info(int card_id, int device_id, int type, struct dsmi_computing_power_info *pcomputing_power)

功能说明

查询算力信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
type	输入	int	获取算力信息的类型。 取值说明：当前取值仅支持1，表示查询AIcore核数。

pcomputing_power	输出	struct dsmi_computing_power_info*	返回算力信息，算力信息结构体：
			struct dsmi_computing_power_info {
			unsigned int data1;
			unsigned int reserve[3];
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-85 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
struct dsmi_computing_power_info computing_power = {0};
int type = 1;
ret = dcmi_get_computing_power_info(card_id, device_id, type, &computing_power);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.84 dcmi_get_device_aicpu_count_info 接口原型

函数原型

int dcmi_get_device_aicpu_count_info(int card_id, int device_id, unsigned char *count_info)

功能说明

获取芯片的aicpu数量信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
count_info	输出	unsigned char *	获取芯片的aicpu数量信息。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-86 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0x3;
int device_id = 0;
unsigned char count_info = 0;
ret = dcmi_get_device_aicpu_count_info(card_id, device_id, &count_info);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

2.85 dcmi_get_first_power_on_date 接口原型

函数原型

int dcmi_get_first_power_on_date(int card_id, unsigned int *first_power_on_date)

功能说明

获取指定设备的首次上电日期。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
first_power_on_date	输出	unsigned int *	首次上电日期信息为1970/1/1 00:00:00到当前时间的秒数，时间精确到日。全0表示当前还未将首次上电时间写入flash，需要等待24小时后查询。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-87 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
unsigned int first_power_on_date = 0;
ret = dcmi_get_first_power_on_date(card_id, &first_power_on_date);
...
```

2.86 dcmi_get_fault_event 接口原型

函数原型

int dcmi_get_fault_event(int card_id, int device_id, int timeout, struct dcmi_event_filter filter, struct dcmi_event *event)

功能说明

订阅设备故障或恢复事件的接口。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	NPU管理单元ID，当前实际支持的ID通过**dcmi_get_card_num_list**接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。
timeout	输入	int	timeout >= 0：阻塞等待timeout(ms)时间，最大阻塞时间为30000ms(30s)； timeout = -1：阻塞等待永不超时。

参数名称	输入/输出	类型	描述

filter	输入	struct dcmi_event_filter	可只订阅满足指定条件的事件，过滤条件如下：
			#define DCMI_EVENT_FILTER_FLAG_EVENT_ID (1UL << 0)
			#define DCMI_EVENT_FILTER_FLAG_SERVERITY (1UL << 1)
			#define DCMI_EVENT_FILTER_FLAG_NODE_TYPE (1UL << 2)
			#define DCMI_MAX_EVENT_RESV_LENGTH 32
			struct dcmi_event_filter {
			unsigned long long filter_flag; /* 可单独使能某个过滤条件，也可将全部条件同时使能，过滤条件如下：
			0: 不使能过滤条件
			DCMI_EVENT_FILTER_FLAG_EVENT_ID: 只接收指定的事件
			DCMI_EVENT_FILTER_FLAG_SERVERITY: 只接收指定级别及以上的事件
			DCMI_EVENT_FILTER_FLAG_NODE_TYPE: 只接收指定节点类型的事件 */
			unsigned int event_id; /* 接收指定的事件*/
			unsigned char severity; /* 接收指定级别及以上的事件：见struct dcmi_dms_fault_event结构体中severity定义 */
			unsigned char node_type; /* 接收指定节点类型的事件*/
			unsigned char resv[DCMI_MAX_EVENT_RESV_LENGTH]; /* < reserve 32bytes */
			};

参数名称	输入/输出	类型	描述

event	输出	struct dcmi_event *	输出事件结构体定义如下：
			struct dcmi_event {
			enum dcmi_event_type type; /* 事件类型 */
			union {
			struct dcmi_dms_fault_event dms_event; /* 事件内容 */
			} event_t;
			};
			type：
			当前支持DCMI_DMS_FAULT_EVENT类型，枚举定义如下：
			enum dcmi_event_type {
			DCMI_DMS_FAULT_EVENT = 0,
			DCMI_EVENT_TYPE_MAX
			};
			dms_event：
			DCMI_DMS_FAULT_EVENT类型对应的事件内容定义如下：
			#define DCMI_MAX_EVENT_NAME_LENGTH 256
			#define DCMI_MAX_EVENT_DATA_LENGTH 32
			#define DCMI_MAX_EVENT_RESV_LENGTH 32
			struct dcmi_dms_fault_event {
			unsigned int event_id; /* 事件id */
			unsigned short deviceid; /* 设备号 */
			unsigned char node_type; /* 节点类型 */
			unsigned char node_id; /* 节点id */
			unsigned char sub_node_type; /* 子节点类型 */
			unsigned char sub_node_id; /* 子节点id */
			unsigned char severity; /* 事件级别 0：提示，1：次要，2：重要，3：紧急 */
			unsigned char assertion; /* 事件类型 0：故障恢复，1：故障产生，2：一次性事件 */
			int event_serial_num; /* 告警序列号 */
			int notify_serial_num; /* 通知序列号 */

参数名称	输入/输出	类型	描述
			unsigned long long alarm_raised_time; /* 事件产生时间：自1970年1月1日0点0分0秒开始至今的毫秒数 */ char event_name[DCMI_MAX_EVENT_NAME_LENGTH]; /* 事件描述信息 */ char additional_info[DCMI_MAX_EVENT_DATA_LENGTH]; /* 事件附加信息 */ unsigned char resv[DCMI_MAX_EVENT_RESV_LENGTH]; /**< reserve 32bytes */ };

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

- 该接口可获取故障产生时正在上报故障或恢复事件，不能获取已经产生的历史事件。
- 该接口支持多进程不支持多线程，最大支持64个进程同时调用。

表 2-88 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int timeout = 1000;
struct dcmi_event_filter filter = {0};
struct dcmi_event event = {0};
filter.filter_flag = DCMI_EVENT_FILTER_FLAG_SERVERITY | DCMI_EVENT_FILTER_FLAG_NODE_TYPE;
filter.severity = 2; /* 只订阅2~3级别的事件 */
filter.node_type = 0x40; /* 只订阅模块ID为SOC类型的事件 */
ret = dcmi_get_fault_event(card_id, device_id, timeout, filter, &event);
if (ret != DCMI_OK) {
    printf("dcmi_get_fault_event failed. err is %d\n", ret);
}
// todo
...
```

2.87 dcmi_get_device_resource_info 接口原型

函数原型

int dcmi_get_device_resource_info (int card_id, int device_id, struct dcmi_proc_mem_info *proc_info, int *proc_num)

功能说明

获取指定设备上的业务进程及其占用的内存。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。

proc_info	输出	struct dcmi_proc_mem_info *	结构体包含进程id和进程占用的内存（byte），进程id是host侧id，内存是device侧OS占用的内存和业务分配的内存总和。
			结构体定义如下：
			struct dcmi_proc_mem_info {
			int proc_id;
			unsigned long proc_mem_usage;
			};

proc_num	输出	int *	进程个数，最多32个，无业务时进程为0。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 2-89 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
#define MAX_PROC_NUM_IN_DEVICE 32
int ret;
int card_id = 0;
int chip_id = 0;
struct dcmi_proc_mem_info proc_info[MAX_PROC_NUM_IN_DEVICE] = {0};
int proc_num = 0;
char proc_name[16] = {0};
int name_len = 0;
ret = dcmi_get_device_resource_info(card_id, chip_id, proc_info, &proc_num);
if (ret != DCMI_OK) {
    printf("dcmi_get_device_resource_info failed. err is %d\n", ret);
}
return ret;
...
```

2.88 dcmi_get_device_dvpp_ratio_info 接口原型

函数原型

int dcmi_get_device_dvpp_ratio_info(int card_id, int device_id, struct dcmi_dvpp_ratio *usage)

功能说明

获取并整合DVPP的5种查询信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。

usage	输出	struct dcmi_dvpp_ratio *	返回对应DVPP命令内容
			struct dcmi_dvpp_ratio {
			int vdec_ratio; // H.264/H.265的视频解码功能
			int vpc_ratio; //对图片和视频其它方面的处理功能
			int venc_ratio; //输出视频的编码功能
			int jpege_ratio; //对JPEG格式的图片进行编码功能
			int jpegd_ratio; //对JPEG格式的图片进行解码功能
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-90 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
struct dcmi_dvpp_ratio usage = {0};
ret = dcmi_get_device_dvpp_ratio_info(card_id, device_id, &usage);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

2.89 dcmi_get_device_phyid_from_logicid 接口原型

函数原型

int dcmi_get_device_phyid_from_logicid(unsigned int logicid, unsigned int *phyid)

功能说明

根据设备逻辑ID获取物理ID。

参数说明

参数名称	输入/输出	类型	描述
logicid	输入	unsigned int	NPU设备逻辑ID，当前实际支持的ID通过dcmi_get_device_logic_id接口获取。
phyid	输出	unsigned int	NPU设备物理ID。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-91 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int chip_id = 0;
int logic_id = 0;
unsigned int phy_id = 0;
ret = dcmi_get_device_logic_id(&logic_id, card_id, chip_id);
if (ret != DCMI_OK) {
    printf("dcmi_get_device_logic_idfailed. err is %d\n", ret);
    return ret;
}
ret = dcmi_get_device_phyid_from_logicid((unsigned int)logic_id, &phy_id);
if (ret != DCMI_OK) {
    printf("dcmi_get_device_phyid_from_logicidfailed. err is %d\n", ret);
}
return ret;
...
```

2.90 dcmi_get_device_logicid_from_phyid 接口原型

函数原型

int dcmi_get_device_logicid_from_phyid(unsigned int phyid, unsigned int *logicid)

功能说明

根据设备物理ID获取逻辑ID。

参数说明

参数名称	输入/输出	类型	描述
phyid	输入	unsigned int	NPU设备物理ID。 **说明** 可执行 ls /dev/davinci* 命令获取设备的物理ID，如显示/dev/davinci0，则表示设备的物理ID为0。
logicid	输出	unsigned int	NPU设备逻辑ID。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在物理机场景和特权容器场景下支持，在其他容器场景下获取的结果不保证正确性。

表 2-92 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
unsigned int logicid;
unsigned int phyid = 0;
ret = dcmi_get_device_logicid_from_phyid(phyid, &logicid);
if (ret != DCMI_OK) {
    printf("dcmi_get_device_logicid_from_phyid failed. err is %d\n", ret);
}
return ret;
...
```

2.91 dcmi_get_card_id_device_id_from_phyid 接口原型

函数原型

int dcmi_get_card_id_device_id_from_phyid(int *card_id, int *device_id, unsigned int device_phy_id)

功能说明

根据设备物理ID查询NPU管理单元ID以及NPU管理单元上的设备编号。

参数说明

参数名称	输入/输出	类型	描述
card_id	输出	int	NPU管理单元ID。
device_id	输出	int	NPU管理单元上设备编号。
device_phy_id	输入	unsigned int	NPU设备物理ID。 **说明** 可执行 ls /dev/davinci* 命令获取设备的物理ID，如显示/dev/davinci0，则表示设备的物理ID为0。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-93 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int chip_id = 0;
unsigned int device_phy_id = 0;
ret = dcmi_get_card_id_device_id_from_phyid(&card_id, &chip_id, device_phy_id);
if (ret != DCMI_OK) {
    printf("dcmi_get_card_id_device_id_from_phyid failed. err is %d\n", ret);
}
return ret;
...
```

2.92 dcmi_get_card_id_device_id_from_logicid 接口原型

函数原型

int dcmi_get_card_id_device_id_from_logicid(int *card_id, int *device_id, unsigned int device_logic_id)

功能说明

根据设备逻辑ID查询NPU管理单元ID以及NPU管理单元上的设备编号。

参数说明

参数名称	输入/输出	类型	描述
card_id	输出	int	NPU管理单元ID。
device_id	输出	int	NPU管理单元上设备编号。
device_logic_id	输入	unsigned int	NPU设备逻辑ID。当前支持ID范围通过dcmi_get_device_logic_id接口获取。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-94 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int chip_id = 0;
unsigned int device_logic_id = 0;
ret = dcmi_get_card_id_device_id_from_logicid(&card_id, &chip_id, device_logic_id);
if (ret != DCMI_OK) {
    printf("dcmi_get_card_id_device_id_from_logicid failed. err is %d\n", ret);
}
return ret;
...
```

2.93 dcmi_get_vdevice_mode 接口原型

函数原型

int dcmi_get_vdevice_mode(int *mode)

功能说明

查询算力切分模式。

参数说明

参数名称	输入/输出	类型	描述
mode	输出	int*	算力切分模式。输出为0或1： ● 0：算力切分容器模式 ● 1：算力切分虚拟机模式

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-95 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

2.94 dcmi_get_vnpu_config_recover_mode 接口原型

函数原型

int dcmi_get_vnpu_config_recover_mode(unsigned int *mode)

功能说明

查询vNPU配置恢复使能状态。

参数说明

参数名称	输入/输出	类型	描述
mode	输出	unsigned int	vNPU的配置恢复使能状态。分为禁用、使能。默认为使能状态。 ● 0：禁用 ● 1：使能

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-96 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret;
unsigned int mode = 0;
ret = dcmi_get_vnpu_config_recover_mode(&mode);
if (ret != DCMI_OK) {
    printf("dcmi_get_vnpu_config_recover_mode failed. err is %d\n", ret);
}
return ret;
...
```

2.95 dcmi_get_device_boot_status 接口原型

函数原型

int dcmi_get_device_boot_status(int card_id, int device_id, enum dcmi_boot_status *boot_status)

功能说明

获取设备的启动状态。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

boot_status	输出	enum dcmi_boot_status *	enum dcmi_boot_status
			{
			DCMI_BOOT_STATUS_UNINIT = 0, /*未初始化*/
			DCMI_BOOT_STATUS_BIOS, * BIOS启动中*/
			DCMI_BOOT_STATUS_OS, /* OS启动中*/
			DCMI_BOOT_STATUS_FINISH /*启动完成*/
			DCMI_SYSTEM_START_FINISH = 16, /*dcmi服务进程启动完成*/
			}

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-97 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
enum dcmi_boot_status boot_status = 0;
```



```
ret = dcmi_get_device_boot_status(card_id, device_id, &boot_status);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

2.96 dcmi_sm_encrypt 接口原型

函数原型

int dcmi_sm_encrypt(int card_id, int device_id, struct dcmi_sm_parm* parm, struct dcmi_sm_data *data)

功能说明

调用此接口，输入需加密的明文、加密的密钥以及国密加密算法类型，获取加密后的密文。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

parm	输入	struct dcmi_sm_parm*	srtuct sm_parm {
			unsigned int key_type;
			unsigned int key_len;//SM4密钥长度为16字节，不涉及SM3
			unsigned int iv_len;//SM4初始值长度为16字节，不涉及SM3
			unsigned int reserves;//预留
			unsigned char iv[64];
			unsigned char key[512];
			unsigned char reserved[512];//预留
			};
			key_type 入参范围：
			enum sm_key_type{
			SM3_NORMAL_SUMMARY = 0,
			SM4_CBC_ENCRYPT = 1,
			SM4_CBC_DECRYPT = 2,
			};

参数名称	输入/输出	类型	描述

data	输入；输出	struct dcmi_sm_data *	struct sm_data {
			const unsigned char *in_buf;
			unsigned in_len;//SM3长度、SM4长度最多为3072字节，且SM4长度必须为16字节的整数倍
			unsigned char *out_buf;
			unsigned int *out_len;
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-98 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
int ret;
  int card_id = 0;
  int device_id = 0;
  struct dcmi_sm_parm parm = {0};
  parm.key_type = 0;
  unsigned char hash1[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};
  unsigned char *out_buf =  (unsigned char *)malloc(100);
  unsigned int *out_len =(unsigned int *)malloc(sizeof(unsigned int));
  *out_len = 100;
  struct dcmi_sm_data data = {(const unsigned char *)hash1, sizeof(hash1), out_buf, out_len};
  dcmi_init();
  ret = dcmi_sm_encrypt(0, 0, &parm, &data);
  if (ret != 0) {
```



```
    //todo:记录日志
    free(out_buf);
    free(out_len);
    return ret;
}
//data.out_buf中记录加密后的数据,data.out_len记录加密后的数据长度
free(out_buf);
free(out_len);
```

2.97 dcmi_sm_decrypt 接口原型

函数原型

int dcmi_sm_decrypt(int card_id, int device_id, struct sm_parm* parm, struct sm_data*data)

功能说明

调用此接口，输入加密后的密文、解密的密钥以及国密解密算法类型，获取解密后的明文。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

parm	输入	struct sm_parm*	srtuct sm_parm {
			unsigned int key_type;
			unsigned int key_len;//SM4密钥长度为16字节，不涉及SM3
			unsigned int iv_len;//SM4初始值长度为16字节，不涉及SM3
			unsigned int reserves;//预留
			unsigned char iv[64];
			unsigned char key[512];
			unsigned char reserved[512];//预留
			};
			key_type 入参范围：
			enum sm_key_type{
			SM3_NORMAL_SUMMARY = 0,
			SM4_CBC_ENCRYPT = 1,
			SM4_CBC_DECRYPT = 2,
			};

data	输入；输出	struct sm_data*	struct sm_data {
			const unsigned char *in_buf;
			unsigned in_len;//SM3长度、SM4长度最多为3072字节，且SM4长度必须为16字节的整数倍
			unsigned char *out_buf;
			unsigned int *out_len;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-99 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
int ret;
int card_id = 0;
int device_id = 0;
unsigned char data1[] = {   // 待解密的密文
   /密文/
};
unsigned char key_in1[] = { // 加密/解密输入的key
  /16字节的密钥/
};
unsigned char iv_in1[] = {  // 加密/解密输入的iv值
  /16字节的iv值/
};
struct sm_parm sm_test_param = {0};
sm_test_param.key_type = SM4_CBC_DECRYPT;
memcpy(sm_test_param.key, key_in1, sizeof(key_in1));
memcpy(sm_test_param.key, iv_in1, sizeof(iv_in1));
sm_test_param.key_len = 16;
sm_test_param.iv_len = 16;
unsigned char *out_buf =  (unsigned char *)malloc(sizeof(data1));
unsigned int *out_len =(unsigned int *)malloc(sizeof(unsigned int));
*out_len = sizeof(data1);
struct dcmi_sm_data sm_test_data = {(const unsigned char *)data1, sizeof(data1), out_buf, out_len};
dcmi_init();
ret = dcmi_sm_decrypt(0, 0, &sm_test_param, &sm_test_data);
if (ret != 0) {
    //todo:记录日志
    free(out_buf);
    free(out_len);
    return ret;
}
//data.out_buf中记录解密后的数据,data.out_len记录解密后的数据长度
free(out_buf);
free(out_len);
```

2.98 dcmi_set_power_state 接口原型

函数原型

int dcmi_set_power_state(int card_id, int device_id, struct dcmi_power_state_info_stru power_info)

功能说明

设置系统模式。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

type	输入	struct dcmi_power_state_info_stru	设置系统状态参数。
			typedef enum {
			DCMI_POWER_STATE_SUSPEND,
			DCMI_POWER_STATE_POWEROFF,
			DCMI_POWER_STATE_RESET,
			DCMI_POWER_STATE_MAX,
			} DCMI_POWER_STATE;
			typedef enum {
			DCMI_POWER_RESUME_MODE_BUTTON, /* resume by button */
			DCMI_POWER_RESUME_MODE_TIME, /* resume by time */
			DCMI_POWER_RESUME_MODE_MAX,
			} DCMI_LP_RESUME_MODE;
			#define DCMI_POWER_INFO_RESERVE_LEN 8
			struct dcmi_power_state_info_stru {
			DCMI_POWER_STATE type;
			DCMI_LP_RESUME_MODE mode;
			unsigned int value;
			unsigned int reserve[DCMI_POWER_INFO_RESERVE_LEN];
			};
			说明
			type的值当前仅支持DCMI_POWER_STATE_SUSPEND。
			当mode为DCMI_POWER_RESUME_MODE_TIME时，value的取值区间为[200ms,604800000ms]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

命令执行后设备立即进入休眠状态，休眠时间结束后自动唤醒。

表 2-100 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
    int ret;
    int card_id = 0;
    int device_id = 0;
    struct dcmi_power_state_info_stru power_info = {0};

    power_info.type = DCMI_POWER_STATE_SUSPEND;
    power_info.mode = DCMI_POWER_RESUME_MODE_TIME;
    power_info.value = 300;// 300ms

    ret = dcmi_set_power_state(card_id, device_id, power_info);
    if (ret) {
        // todo
    }
    // todo
...
```

2.99 dcmi_get_npu_work_mode 接口原型

函数原型

int dcmi_get_npu_work_mode(int card_id, unsigned char *work_mode)

功能说明

查询NPU工作模式。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
work_mode	输出	unsigned char *	NPU工作模式。输出为0或1： ● 0：AMP ● 1：SMP

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 2-101 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int card_id;
unsigned char *work_mode;
ret= dcmi_get_npu_work_mode(&work_mode);
if (ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3 配置管理接口

3.1 dcmi_set_device_clear_pcie_error 接口原型

函数原型

int dcmi_set_device_clear_pcie_error(int card_id, int device_id)

功能说明

清除设备的PCIe链路误码信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-1 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
int ret = 0;
int card_id = 0;
int device_id = 0;
ret = dcmi_set_device_clear_pcie_error(card_id, device_id);
...
```

3.2 dcmi_clear_pcie_error_cnt 接口原型

函数原型

int dcmi_clear_pcie_error_cnt(int card_id, int device_id)

功能说明

清除设备的PCIe链路误码信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用3.1 dcmi_set_device_clear_pcie_error接口原型。

表 3-2 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

3.3 dcmi_get_device_p2p_enable 接口原型

函数原型

int dcmi_get_device_p2p_enable(int card_id, int device_id, int *enable_flag)

功能说明

查询Flash存储的P2P使能标记。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
enable_flag	输出	int *	• 0：禁用 • 1：使能

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 3-3 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
int ret = 0;
int card_id = 0;
int device_id = 0;
int enable_flag = 0;
ret = dcmi_get_device_p2p_enable(card_id, device_id, &enable_flag);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.4 dcmi_get_p2p_enable 接口原型

函数原型

int dcmi_get_p2p_enable(int card_id, int device_id, int* enable_flag)

功能说明

查询Flash存储的P2P使能标记。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
enable_flag	输出	int *	• 0：禁用 • 1：使能

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用3.3 dcmi_get_device_p2p_enable接口原型。

表 3-4 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int enable_flag = 1;
int card_id = 0;
int device_id = 0;
ret = dcmi_get_p2p_enable(card_id, device_id, &enable_flag);
```



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

3.5 dcmi_set_device_clear_ecc_statistics_info 接口原型

函数原型

int dcmi_set_device_clear_ecc_statistics_info(int card_id, int device_id)

功能说明

清除ecc统计信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-5 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

N	N	N

调用示例

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

3.6 dcmi_set_device_mac 接口原型

函数原型

int dcmi_set_device_mac(int card_id, int device_id, int mac_id, const char *mac_addr, unsigned int len)

功能说明

设置指定设备的MAC地址。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
mac_id	输入	int	取值范围：0~3。 该参数在当前版本不使用。默认值为0，请保持默认值即可。
mac_addr	输入	const char *	设置6个字节的MAC地址。
len	输入	unsigned int	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： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-6 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
char mac_addr[6] = {0xXX,0xXX,0xXX,0xXX,0xXX,0xXX};  // XX 表示mac地址的各段数据，以实际写入值为准
ret = dcmi_set_device_mac(card_id, dev_id, 0, mac_addr, 6);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.7 dcmi_get_device_mac 接口原型

函数原型

int dcmi_get_device_mac(int card_id, int device_id, int mac_id, char *mac_addr, unsigned int len)

功能说明

获取指定设备的MAC地址。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
mac_id	输入	int	取值范围：0~3。 该参数在当前版本不使用。默认值为0，请保持默认值即可。
mac_addr	输出	char *	输出6个字节的MAC地址。
len	输入	unsigned int	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： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-7 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
```



```
char mac_addr[6] = {0};
ret = dcmi_get_device_mac(card_id, device_id, 0, mac_addr, 6);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.8 dcmi_get_device_gateway 接口原型

函数原型

int dcmi_get_device_gateway(int card_id, int device_id, enum dcmi_port_type input_type, int port_id, struct dcmi_ip_addr *gateway)

功能说明

获取网关地址。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

input_type	输入	enum dcmi_port_type	指定网口类型。
			enum dcmi_port_type {
			DCMI_VNIC_PORT = 0,
			DCMI_ROCE_PORT = 1,
			DCMI_INVALID_PORT
			};

port_id	输入	int	指定网口号，保留字段。取值范围：[0, 255]。

gateway	输出	struct dcmi_ip_addr *	网关地址
			struct dcmi_ip_addr {
			union {
			unsigned char ip6[16];
			unsigned char ip4[4];
			} u_addr;
			enum dcmi_ip_addr_type ip_type;
			};

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 3-8 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int port_id = 0;
struct dcmi_ip_addr ip_gateway_address = {0};
ret = dcmi_get_device_gateway(card_id,device_id, DCMI_ROCE_PORT, port_id, &ip_gateway_address);
if (ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.9 dcmi_set_device_gateway 接口原型

函数原型

int dcmi_set_device_gateway(int card_id, int device_id, enum dcmi_port_type input_type, int port_id, struct dcmi_ip_addr *gateway)

功能说明

设置网关地址。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

input_type	输入	enum dcmi_port_type	指定网口类型。
			enum dcmi_port_type {
			DCMI_VNIC_PORT = 0,
			DCMI_ROCE_PORT = 1,
			DCMI_INVALID_PORT
			};

port_id	输入	int	指定网口号，保留字段。取值范围：[0, 255]。

gateway	输入	struct dcmi_ip_addr *	网关地址
			struct dcmi_ip_addr {
			union {
			unsigned char ip6[16];
			unsigned char ip4[4];
			} u_addr;
			enum dcmi_ip_addr_type ip_type;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-9 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int port_id = 0;
unsigned int gateway_address = 0xC801A8C0; //192.168.1.200（仅用作示例，以实际配置网关地址为准）
struct dcmi_ip_addr ip_gateway_address = {0};
memcpy(&(ip_gateway_address.u_addr.ip4[0]),&gateway_address,4);
ret = dcmi_set_device_gateway(card_id,device_id, DCMI_ROCE_PORT, port_id, &ip_gateway_address);
if (ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.10 dcmi_get_device_ip 接口原型

函数原型

int dcmi_get_device_ip(int card_id, int device_id, enum dcmi_port_type input_type, int port_id, struct dcmi_ip_addr *ip, struct dcmi_ip_addr *mask)

功能说明

获取IP地址和mask地址。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

input_type	输入	enum dcmi_port_type	指定网口类型。
			enum dcmi_port_type {
			DCMI_VNIC_PORT = 0,
			DCMI_ROCE_PORT = 1,
			DCMI_INVALID_PORT
			};

port_id	输入	int	指定网口号，保留字段。取值范围：[0, 255]。

ip	输出	struct dcmi_ip_addr *	IP地址
			struct dcmi_ip_addr {
			union {
			unsigned char ip6[16];
			unsigned char ip4[4];
			} u_addr;
			enum dcmi_ip_addr_type ip_type;
			};

mask	输出	struct dcmi_ip_addr *	mask地址
			struct dcmi_ip_addr {
			union {
			unsigned char ip6[16];
			unsigned char ip4[4];
			} u_addr;
			enum dcmi_ip_addr_type ip_type;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-10 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int port_id = 0;
struct dcmi_ip_addr ip_address = {0};
struct dcmi_ip_addr ip_mask_address = {0};
ret = dcmi_get_device_ip(card_id,device_id, DCMI_ROCE_PORT, port_id, &ip_address, &ip_mask_address);
if (ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.11 dcmi_set_device_ip 接口原型

函数原型

int dcmi_set_device_ip(int card_id, int device_id, enum dcmi_port_type input_type, int port_id, struct dcmi_ip_addr *ip, struct dcmi_ip_addr *mask)

功能说明

设置IP地址和mask地址。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

input_type	输入	enum dcmi_port_type	指定网口类型。
			enum dcmi_port_type {
			DCMI_VNIC_PORT = 0,
			DCMI_ROCE_PORT = 1,
			DCMI_INVALID_PORT
			};

port_id	输入	int	指定网口号，保留字段。取值范围：[0, 255]。

ip	输入	struct dcmi_ip_addr *	IP地址
			struct dcmi_ip_addr {
			union {
			unsigned char ip6[16];
			unsigned char ip4[4];
			} u_addr;
			enum dcmi_ip_addr_type ip_type;
			};

mask	输入	struct dcmi_ip_addr *	mask地址
			struct dcmi_ip_addr {
			union {
			unsigned char ip6[16];
			unsigned char ip4[4];
			} u_addr;
			enum dcmi_ip_addr_type ip_type;
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-11 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int port_id = 0;
unsigned int ip_addr = 0xC801A8C0; //192.168.1.200（仅用作示例，以实际配置网关地址为准）
unsigned int mask_addr = 0x00FFFFFF; //255.255.255.0（仅用作示例，以实际配置网关地址为准）
struct dcmi_ip_addr ip_address = {0};
struct dcmi_ip_addr 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 = dcmi_set_device_ip(card_id,device_id, DCMI_ROCE_PORT, port_id, &ip_address, &ip_mask_address);
if (ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.12 dcmi_set_device_ecc_enable 接口原型

函数原型

int dcmi_set_device_ecc_enable(int card_id, int device_id, enum dcmi_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，调用该接口配置ECC标记后，需要重启系统，配置才能生效。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述

device_type	输入	enum dcmi_device_type	设备类型
			enum dcmi_device_type {
			DCMI_DEVICE_TYPE_DDR,
			DCMI_DEVICE_TYPE_SRAM,
			DCMI_DEVICE_TYPE_HBM,
			DCMI_DEVICE_TYPE_NPU,
			DCMI_HBM_RECORDED_SINGLE_ADDR,
			DCMI_HBM_RECORDED_MULTI_ADDR,
			DCMI_DEVICE_TYPE_NONE = 0xff
			};
			目前支持DCMI_DEVICE_TYPE_DDR。

enable_flag	输入	int	• 0：禁用 • 1：使能

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-12 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
int ret = 0;
```



```
int card_id = 0;
int device_id = 0;
ret = dcmi_set_device_ecc_enable(card_id, device_id, DCMI_DEVICE_TYPE_DDR, 1);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.13 dcmi_config_ecc_enable 接口原型

函数原型

int dcmi_config_ecc_enable(int card_id, int device_id, int enable_flag)

功能说明

配置存储ECC的标记为使能或禁用，调用该接口配置ECC标记后，需要重启系统，配置才能生效。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
enable_flag	输入	int	• 0：禁用 • 1：使能

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用3.12 dcmi_set_device_ecc_enable接口原型。

表 3-13 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
int enable_flag = 1;
ret = dcmi_config_ecc_enable(card_id, device_id, enable_flag);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.14 dcmi_get_nve_level 接口原型

函数原型

int dcmi_get_nve_level(int card_id, int device_id, int *nve_level)

功能说明

查询指定芯片的算力等级。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
level	输出	int*	获取功耗和算力档位 配置项内容支持如下选项，默认值是3： 0：功耗和算力档位为low 1：功耗和算力档位为middle 2：功耗和算力档位为high 3：功耗和算力档位为full

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-14 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int level = 0;
int card_id = 0;
int device_id = 0;
ret = dcmi_get_nve_level(card_id, device_id, &level);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```


3.15 dcmi_set_nve_level 接口原型

函数原型

int dcmi_set_nve_level(int card_id, int device_id, int level)

功能说明

设置指定芯片的算力等级。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
level	输入	int	设置功耗和算力档位 配置项内容支持如下选项，默认值是3： 0：功耗和算力档位为low 1：功耗和算力档位为middle 2：功耗和算力档位为high 3：功耗和算力档位为full

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-15 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

3.16 dcmi_set_device_share_enable 接口原型

函数原型

int dcmi_set_device_share_enable(int card_id, int device_id, int enable_flag)

功能说明

配置指定芯片的容器共享使能标记为使能或禁用。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
enable_flag	输入	int	1：使能 0：禁用 默认禁用。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

设置容器共享模式后，重启后容器使能状态默认为禁用。

表 3-16 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
ret = dcmi_set_device_share_enable(card_id, device_id, 1);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.17 dcmi_get_device_share_enable 接口原型

函数原型

int dcmi_get_device_share_enable(int card_id, int device_id, int *enable_flag)

功能说明

查询指定芯片的容器共享使能标记。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
enable_flag	输出	int *	1：使能 0：禁用 默认禁用。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

设置容器共享模式后，重启后容器使能状态默认为禁用。

表 3-17 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
int ret = 0;
int enable_flag = -1;
int card_id = 0;
int device_id = 0;
```



```
ret = dcmi_get_device_share_enable(card_id, device_id, &enable_flag);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.18 dcmi_set_device_user_config 接口原型

函数原型

int dcmi_set_device_user_config(int card_id, int device_id, const char *config_name, unsigned int buf_size, char *buf)

功能说明

设置用户配置。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
config_name	输入	const char *	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32。 已实现功能的配置项：ddr_ecc_enable、cpu_num_cfg。支持用户自定义名称配置。 配置项功能说明如下： ● "ddr_ecc_enable"：用于使能或禁用ECC。 ● "cpu_num_cfg"：用于系统cpu配比设置。 **说明** cpu_num_cfg： ● data CPU不使用，要求配置为0。 ● control CPU和aicpu的数量总和要和实际device的cpu核数量一致，并且control CPU数量不能小于1。 调用该接口配置标记后，需要复位芯片，配置才能生效。配置生效后，可通过**3.20 dcmi_get_user_config接口原型**接口查看配置结果。

参数名称	输入/输出	类型	描述
buf_size	输入	unsigned int	buf长度，单位为byte，最大长度为1K byte。 支持ddr_ecc_enable、cpu_num_cfg配置。 目前支持处理的配置项名称如下： ● 如果配置"ddr_ecc_enable"：buf_size参数配置为1。 ● 如果配置"cpu_num_cfg"：传入的data的结构体都是uc_cpu_cfg_t，size都是结构体的实际长度16。 除了如上固定的名称外，其他最大长度不超过1024 byte。

buf	输入	char *	buf指针，指向配置项内容。
			支持ddr_ecc_enable、cpu_num_cfg配置。
			目前支持处理的配置项名称如下：
			● 针对"ddr_ecc_enable"配置项，配置项内容支持如下选项，默认值是0：
			1：表示使能ECC
			0：表示禁用ECC
			● 针对"cpu_num_cfg"配置项，用户需要自定义配比的数据结构，该数据结构为：
			typedef struct uc_cpu_cfg_stru {
			unsigned char ctrl_cpu_num; // control CPU数量
			unsigned char data_cpu_num; // datacpu 数量
			unsigned char ai_cpu_num; // aicpu 数量
			unsigned char reserved_1;
			unsigned int reserved_2[3]; /* 3 word 12 bytes, total 16 bytes */
			} uc_cpu_cfg_t;
			除了如上固定的名称外，其他配置项内容请根据实际情况配置。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-18 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

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

3.19 dcmi_set_user_config 接口原型

函数原型

int dcmi_set_user_config(int card_id, int device_id, const char *config_name, unsigned int buf_size, unsigned char *buf)

功能说明

设置用户配置。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。

参数名称	输入/输出	类型	描述
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
config_name	输入	const char *	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32。 已实现功能的配置项：ddr_ecc_enable、cpu_num_cfg。支持用户自定义名称配置。 配置项功能说明如下： ● "ddr_ecc_enable"：用于使能或禁用ECC。 ● "cpu_num_cfg"：用于系统cpu配比设置。 **说明** cpu_num_cfg： ● data CPU不使用，要求配置为0。 ● control CPU和aicpu的数量总和要和实际device的cpu核数量一致，并且control CPU数量不能小于1。 调用该接口配置标记后，需要复位芯片，配置才能生效。配置生效后，可通过**3.20 dcmi_get_user_config接口原型**接口查看配置结果。
buf_size	输入	unsigned int	buf长度，单位为byte，最大长度为1K byte。 支持ddr_ecc_enable、cpu_num_cfg配置。 目前支持处理的配置项名称如下： ● 如果配置"ddr_ecc_enable"：buf_size参数配置为1。 ● 如果配置"cpu_num_cfg"：传入的data的结构体都是uc_cpu_cfg_t，size都是结构体的实际长度16。 除了如上固定的名称外，其他最大长度不超过1024 byte。

参数名称	输入/输出	类型	描述

buf	输入	unsigned char *	buf指针，指向配置项内容。
			支持ddr_ecc_enable、cpu_num_cfg配置。
			目前支持处理的配置项名称如下：
			• 针对"ddr_ecc_enable"配置项，配置项内容支持如下选项，默认值是0：
			1：表示使能ECC
			0：表示禁用ECC
			• 针对"cpu_num_cfg"配置项，用户需要自定义配比的数据结构，该数据结构为：
			typedef struct uc_cpu_cfg_stru {
			unsigned char ctrl_cpu_num; //control CPU数量
			unsigned char data_cpu_num; //datacpu 数量
			unsigned char ai_cpu_num; //aicpu 数量
			unsigned char reserved_1;
			unsigned int reserved_2[3]; /* 3 word 12 bytes, total 16 bytes */
			} uc_cpu_cfg_t;
			除了如上固定的名称外，其他配置项内容请根据实际情况配置。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**3.18 dcmi_set_device_user_config接口原型**。

表 3-19 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

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

3.20 dcmi_get_user_config 接口原型

函数原型

int dcmi_get_user_config(int card_id, int device_id, const char *config_name, unsigned int buf_size, unsigned char *buf)

功能说明

获取用户配置。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
config_name	输入	const char *	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32。支持用户自定义名称配置。 已实现功能的配置项：ddr_ecc_enable、cpu_num_cfg。 配置项功能说明如下： • "ddr_ecc_enable"：用于使能或禁用ECC。 • "cpu_num_cfg"：用于系统cpu配比设置。
buf_size	输入	unsigned int	buf长度，最大长度为1K byte。 该参数的配置，请参见**3.18 dcmi_set_device_user_config接口原型**。
buf	输出	unsigned char *	buf指针，指向配置项内容。 具体配置项内容的含义，请参见**3.18 dcmi_set_device_user_config接口原型**。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-20 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

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

3.21 dcmi_clear_device_user_config 接口原型

函数原型

int dcmi_clear_device_user_config(int card_id, int device_id, const char *config_name)

功能说明

重置用户配置。

默认值请参见**3.18 dcmi_set_device_user_config接口原型**。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

参数名称	输入/输出	类型	描述
config_name	输入	const char *	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32。 已实现功能的配置项：ddr_ecc_enable、cpu_num_cfg。支持用户自定义名称配置。 配置项功能说明如下： • "ddr_ecc_enable"：用于使能或禁用ECC。 • "cpu_num_cfg"：用于系统cpu配比设置。 **说明** 调用该接口配置标记后，需要复位芯片，配置才能生效。配置生效后，可通过**3.20 dcmi_get_user_config接口原型**接口查看配置结果。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-21 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
```



```
const char *config_name = "mac_info";
ret = dcmi_clear_device_user_config(card_id, device_id, config_name);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

3.22 dcmi_create_vdevice 接口原型

函数原型

int dcmi_create_vdevice(int card_id, int device_id, struct dcmi_create_vdev_res_stru*vdev, struct dcmi_create_vdev_out *out)

功能说明

创建指定NPU单元的vNPU设备。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。取值范围如下： NPU芯片：[0, device_id_max-1]。

vdev	输入	struct dcmi_create_vdev_res_stru*	输入创建虚拟设备信息。
			struct dcmi_create_vdev_res_stru {
			unsigned int vdev_id;
			unsigned int vfg_id;
			char template_name[32];
			unsigned char reserved[64];
			};
			说明
			• vdev_id表示指定虚拟设备对应的虚拟设备ID，默认自动分配，默认值为0xFFFFFFFF。
			• vfg_id表示指定虚拟设备所属的虚拟分组ID，默认自动分配，默认值为0xFFFFFFFF。
			• template_name表示算力切分模板名称。可通过**npu-smi info -t template-info**命令查询其详细信息。
			• reserved表示预留参数。

参数名称	输入/输出	类型	描述

out	输出	struct dcmi_create_vdev_out *	输出创建的虚拟设备信息。当前只支持vdev_id
			struct dcmi_create_vdev_out {
			unsigned int vdev_id;
			unsigned int pcie_bus;
			unsigned int pcie_device;
			unsigned int pcie_func;
			unsigned int vfg_id;
			unsigned char reserved[DCMI_VDEV_FOR_RESERVE];
			};
			说明
			pcie_bus、pcie_device、pcie_func为预留参数。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-22 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
int card_id = 0;
int device_id = 0;
struct dcmi_create_vdev_out out = {0};
int ret;
```



```
struct dcmi_create_vdev_res_stru vdev = {0};
vdev.vdev_id = 0xFFFFFFFF;
strncpy_s(vdev.template_name, sizeof(vdev.template_name), "vir02", strlen("vir02"));
ret = dcmi_create_vdevice(card_id, device_id, &vdev, &out);
if (ret != 0) {
    //todo
    return ret;
}
```

3.23 dcmi_set_destroy_vdevice 接口原型

函数原型

int dcmi_set_destroy_vdevice(int card_id, int device_id, unsigned int vdevid)

功能说明

销毁指定NPU单元的vNPU设备。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
vdevid	输入	unsigned int	已创建的虚拟设备id，当vdevid为65535时，删除所属设备下的所有虚拟设备。 可通过接口**dcmi_get_device_info**查询一个设备下的虚拟设备的信息。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-23 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int card_id = 0;
int device_id = 0;
unsigned int vdevid = 65535;
int ret;
ret = dcmi_set_destroy_vdevice(card_id, device_id, vdevid);
if (ret != 0) {
    //todo
    return ret;
}
...
```

3.24 dcmi_get_device_cpu_num_config 接口原型

函数原型

int dcmi_get_device_cpu_num_config(int card_id, int device_id, unsigned char *buf, unsigned int buf_size)

功能说明

获取系统AI CPU、control CPU、data CPU个数配置。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
buf	输出	unsigned char	buf指针，指向配置项内容。具体请参见**3.25 dcmi_set_device_cpu_num_config接口原型**。

参数名称	输入/输出	类型	描述
buf_size	输入	unsigned int	buf长度，buf_size参数固定配置为16。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-24 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	Y	Y

调用示例

```
...
#define BUF_SIZE 16
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned char buf[BUF_SIZE] = {0};
ret=dcmi_get_device_cpu_num_config(card_id, device_id, buf, BUF_SIZE );
if (ret != 0) {
    //todo:记录日志
    return ret;
}
...
```

3.25 dcmi_set_device_cpu_num_config 接口原型

函数原型

int dcmi_set_device_cpu_num_config(int card_id, int device_id, unsigned char *buf, unsigned int buf_size)

功能说明

配置系统AI CPU、control CPU、data CPU个数。需要复位系统配置才能生效。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。
buf	输入	unsigned char	buf指针，指向配置项内容。具体包含如下： ● buf[0]：control CPU数量。 ● buf[1]：data CPU数量。 ● buf[2]：AI CPU数量。 **说明** ● CPU总核数为4（AI CPU数量：control CPU数量：data CPU数量默认配置为1:3:0） ● data CPU固定配置为0 ● AI CPU数量取值为：[0，3] ● control CPU数量取值为：[1，4]
buf_size	输入	unsigned int	buf长度，buf_size参数固定配置为16。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 3-25 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
Y	N	N

调用示例

```
...
#define BUF_SIZE 16
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned char buf[BUF_SIZE] = {1,0,3};
ret=dcmi_set_device_cpu_num_config(card_id, device_id, buf, BUF_SIZE );
if (ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3.26 dcmi_set_vdevice_mode 接口原型

函数原型

int dcmi_set_vdevice_mode(int mode)

功能说明

设置算力切分模式。

参数说明

参数名称	输入/输出	类型	描述
mode	输入	int	算力切分模式。取值为0或1： ● 0：算力切分容器模式 ● 1：算力切分虚拟机模式 取0和1时，返回不支持。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 3-26 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

3.27 dcmi_set_vnpu_config_recover_mode 接口原型

函数原型

int dcmi_set_vnpu_config_recover_mode(unsigned int mode)

功能说明

设置vNPU配置恢复使能状态。

参数说明

参数名称	输入/输出	类型	描述
mode	输入	unsigned int	vNPU的配置恢复使能状态。分为禁用、使能。默认为使能状态。 ● 0：禁用 ● 1：使能

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

表 3-27 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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


4 MCU 升级控制接口

4.1 dcmi_get_mcu_version 接口原型

函数原型

int dcmi_get_mcu_version(int card_id, char *version, int len)

功能说明

查询MCU版本号。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
version	输出	char*	用户申请的空间，存放返回的固件版本号。

参数名称	输入/输出	类型	描述
len	输入	int	version空间的最大长度。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 4-1 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
char version_str[16] = {0};
int card_id = 1;
ret = dcmi_get_mcu_version(card_id, version_str, sizeof(version_str));
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

4.2 dcmi_mcu_get_version 接口原型

函数原型

int dcmi_mcu_get_version(int card_id, char *version_str, int max_version_len, int *len)

功能说明

查询MCU版本号。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
version_str	输出	char*	用户申请的空间，存放返回的固件版本号。 x.y[.z]格式，x表示Major版本，y表示Minor版本，z表示Revision版本。
max_version_le	输入	int	version_str空间的最大长度。
len	输出	int *	版本号长度。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**4.1 dcmi_get_mcu_version接口原型**。

表 4-2 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
char version_str[16] = {0};
int card_id = 0;
int len = 0;
ret = dcmi_mcu_get_version(card_id, version_str, sizeof(version_str), &len);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

4.3 dcmi_set_mcu_upgrade_stage 接口原型

函数原型

int dcmi_set_mcu_upgrade_stage(int card_id, enum dcmi_upgrade_type input_type)

功能说明

控制MCU升级。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。

input_type	输入	enum dcmi_upgrade_type	升级阶段。
			enum dcmi_upgrade_type {
			MCU_UPGRADE_START = 1,
			MCU_UPGRADE_VALIDETE = 3,
			MCU_UPGRADE_NONE//表示无效
			};

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 4-3 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 1;
enum dcmi_upgrade_type input_type = MCU_UPGRADE_START;
ret = dcmi_set_mcu_upgrade_stage(card_id,input_type);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

4.4 dcmi_mcu_upgrade_control 接口原型

函数原型

int dcmi_mcu_upgrade_control(int card_id, int upgrade_type)

功能说明

控制MCU（Multipoint Control Unit）升级。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
upgrade_type	输入	int	取值范围： ● 1：启动升级 ● 3：生效

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用4.3 dcmi_set_mcu_upgrade_stage接口原型。

表 4-4 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

4.5 dcmi_set_mcu_upgrade_file 接口原型

函数原型

int dcmi_set_mcu_upgrade_file(int card_id, const char *file)

功能说明

将MCU的升级包传至MCU。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
file	输入	const char *	MCU升级包。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 4-5 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 1;
ret = dcmi_set_mcu_upgrade_file(card_id,"./mcu.bin");
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

4.6 dcmi_mcu_upgrade_transfile 接口原型

函数原型

int dcmi_mcu_upgrade_transfile(int card_id, const char *file)

功能说明

将MCU的升级包传至MCU。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的编号通过dcmi_get_card_num_list接口原型获取。
file	输入	char *	MCU升级包。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**4.5 dcmi_set_mcu_upgrade_file接口原型**。

表 4-6 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
ret = dcmi_mcu_upgrade_transfile(card_id,"./mcu.hpm");
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

4.7 dcmi_get_mcu_upgrade_status 接口原型

函数原型

int dcmi_get_mcu_upgrade_status(int card_id, int *status, int *progress)

功能说明

查询MCU升级状态及进度。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
status	输出	int *	升级状态，目前支持如下几种： 0：升级成功 1：升级中 2：不支持升级 3：升级失败 4：获取状态失败
progress	输出	int *	升级进度，0~100百分比。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 4-7 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int status = 0;
int progress = 0;
ret = dcmi_get_mcu_upgrade_status(card_id,  &status, &progress);
if (ret != 0){
    //todo：记录日志
    return ret;
}
...
```

4.8 dcmi_mcu_get_upgrade_status 接口原型

函数原型

int dcmi_mcu_get_upgrade_status(int card_id, int *status, int *progress)

功能说明

查询MCU升级状态及进度。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
status	输出	int *	升级状态，目前支持如下几种： 0：升级成功 1：升级中 2：不支持升级 3：升级失败 4：获取状态失败
progress	输出	int *	升级进度，0~100百分比。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用4.7 dcmi_get_mcu_upgrade_status接口原型。

表 4-8 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int status = 0;
int progress = 0;
ret = dcmi_mcu_get_upgrade_status(card_id,  &status, &progress);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

4.9 dcmi_mcu_get_upgrade_statues 接口原型

函数原型

int dcmi_mcu_get_upgrade_statues(int card_id, int *status, int *progress)

功能说明

查询MCU升级状态及进度。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
status	输出	int *	升级状态，目前支持如下几种： 0：升级成功 1：升级中 2：不支持升级 3：升级失败 4：获取状态失败
progress	输出	int *	升级进度，0~100百分比。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用4.7 dcmi_get_mcu_upgrade_status接口原型。

表 4-9 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
```

```
int status = 0;
int progress = 0;
ret = dcmi_mcu_get_upgrade_statues(card_id,  &status, &progress);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

5 芯片复位接口

5.1 使用前必读

- 带内复位：通过PCIe标准热复位流程复位昇腾AI处理器。使用接口：**dcmi_set_device_reset**（channel_type设置为1）
- 带外复位：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。

  带外复位触发操作流程：**dcmi_set_device_pre_reset**-> **dcmi_set_device_reset**-> **dcmi_set_device_rescan**

须知

- 调用芯片复位接口前，请停掉该芯片的NPU相关业务。
- 调用**dcmi_set_device_reset**接口之后需等待3秒，才可调用**dcmi_set_device_rescan**接口。

5.2 接口说明

5.2.1 dcmi_set_device_pre_reset 接口原型

函数原型

int dcmi_set_device_pre_reset(int card_id, int device_id)

功能说明

设备预复位，发起设备预复位接口，预复位目的是解除上层驱动及软件对此设备的依赖。必须在预复位接口返回成功后，才能对此芯片进行隔离或实际复位操作。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 5-1 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
ret = dcmi_set_device_pre_reset(card_id, device_id);
if (ret != 0) {
    //todo：记录日志
```

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

5.2.2 dcmi_pre_reset_soc 接口原型

函数原型

int dcmi_pre_reset_soc(int card_id, int device_id)

功能说明

芯片预复位，发起芯片预复位接口，预复位目的是解除上层驱动及软件对此芯片的依赖。必须在预复位接口返回成功后，才能对此芯片进行隔离或实际复位操作。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**5.2.3 dcmi_set_device_reset接口原型**。

表 5-2 部署场景

Linux物理机	Linux物理机容器

root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

5.2.3 dcmi_set_device_reset 接口原型

函数原型

int dcmi_set_device_reset(int card_id, int device_id, enum dcmi_reset_channel channel_type)

功能说明

复位芯片。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

channel_type	输入	enum dcmi_reset_channel	复位通道。
			enum dcmi_reset_channel {
			OUTBAND_CHANNEL = 0, // 带外复位
			INBAND_CHANNEL // 带内复位
			}

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

需要调用5.2.1 dcmi_set_device_pre_reset接口原型预复位芯片并返回成功后，才能调用该接口进行带外复位。

表 5-3 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int device_id = 0;
enum dcmi_reset_channel channel_type = INBAND_CHANNEL;
ret = dcmi_set_device_reset(card_id, device_id, channel_type);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

5.2.4 dcmi_reset_device 接口原型

函数原型

int dcmi_reset_device(int card_id, int device_id)

功能说明

通过带外复位芯片。

在调用该接口前，需要调用5.2.2 dcmi_pre_reset_soc接口原型预复位芯片。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**5.2.3 dcmi_set_device_reset接口原型**。

表 5-4 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

5.2.5 dcmi_set_device_rescan 接口原型

函数原型

int dcmi_set_device_rescan(int card_id, int device_id)

功能说明

对指定设备重新扫描。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 5-5 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

5.2.6 dcmi_rescan_soc 接口原型

函数原型

int dcmi_rescan_soc(int card_id, int device_id)

功能说明

对指定芯片重新扫描。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**5.2.5 dcmi_set_device_rescan接口原型**。

表 5-6 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

5.2.7 dcmi_reset_device_inband 接口原型

函数原型

int dcmi_reset_device_inband(int card_id, int device_id)

功能说明

通过带内复位芯片。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
device_id	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。 取值范围如下： NPU芯片：[0, device_id_max-1]。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用5.2.3 dcmi_set_device_reset接口原型。

表 5-7 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

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

5.3 带内复位调用示例

```
#include <stdio.h>
#include "dcmi_interface_api.h"
#define MAX_CARD_NUMBER (16)
#define NPU_OK (0)
int main()
{
    int ret;
    int card_count = 0;
    int device_count = 0;
    int card_id;
    int card_id_list[8] = {0};
    int device_id;
    enum dcmi_reset_channel inband_channel = INBAND_CHANNEL;

    ret = dcmi_init();
```

```
    if (ret != NPU_OK) {
        printf("Failed to init dcmi.\n");
        return ret;
    }
    ret = dcmi_get_card_num_list(&card_count, card_id_list, MAX_CARD_NUMBER);
    if (ret != NPU_OK)
    {
        printf("Failed to get card number,ret is %d\n",ret);
        return ret;
    }
    for (card_id = 0; card_id < card_count; card_id++)
    {
        ret = dcmi_get_device_num_in_card(card_id_list[card_id], &device_count);
        if(ret != NPU_OK) {
           printf("dcmi_get_device_num_in_card failed! card_id is %d ,ret: %d\n", card_id_list[card_id], ret);
            return ret;
        }
        for (device_id = 0; device_id <= device_count; device_id++)
        {
            // 复位
            ret = dcmi_set_device_reset(card_id_list[card_id], device_id, inband_channel);
            if(ret != NPU_OK) {
                if (device_id == device_count) {
                    if (ret == -8255) {
                        printf("dcmi_set_device_reset fail! card_id is %d , device_id is %d, channel_type: %d, ret: %d
\n", card_id_list[card_id], device_id, inband_channel, ret);
                    } else {
                        printf("dcmi_set_device_reset fail! card_id is %d , device_id is %d, channel_type: %d, ret: %d
\n", card_id_list[card_id], device_id, inband_channel, ret);
                    }
                } else {
                    printf("dcmi_set_device_reset fail! card_id is %d , device_id is %d, channel_type: %d, ret: %d
\n", card_id_list[card_id], device_id, inband_channel, ret);
                }
            } else {
                printf("dcmi_set_device_reset successful! card_id is %d, device_id:%d, channel_type: %d\n",
card_id_list[card_id], device_id, inband_channel);
            }
        }
    }
    return 0;
}
```


6 用户自定义信息配置接口

6.1 dcmi_set_card_customized_info 接口原型

函数原型

int dcmi_set_card_customized_info(int card_id, char *info, int len)

功能说明

向MCU写入用户自定义信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
info	输入	char *	写入的信息。大小为1-255个字节。
len	输入	int	写入的信息长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口如果设置过信息，需要清除上次配置才能再次设置。

表 6-1 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
char info[4] = {0};
ret = dcmi_set_card_customized_info(card_id, info, sizeof(info));
...
```

6.2 dcmi_set_customized_info_api 接口原型

函数原型

int dcmi_set_customized_info_api(int card_id, const char *data_info, int len)

功能说明

向MCU写入用户自定义信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
data_info	输入	const char *	信息的内容。大小为1-255个字节。
len	输入	int	信息的长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用6.1 dcmi_set_card_customized_info接口原型。

该接口如果设置过信息，需要清除上次配置才能再次设置。

表 6-2 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
ret = dcmi_set_customized_info_api(card_id, "info", 5);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

6.3 dcmi_mcu_set_license_info 接口原型

函数原型

int dcmi_mcu_set_license_info(int card_id, char *license, int len)

功能说明

向MCU写入用户License证书。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
license	输入	char *	写入证书的内容。大小为1-255个字节。
len	输入	int	写入证书的长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用6.1 dcmi_set_card_customized_info接口原型。

该接口如果设置过信息，需要清除上次配置才能再次设置。

表 6-3 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户

N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
char license[4] = {0};
int len = 4;
ret = dcmi_mcu_set_license_info(card_id, license, len);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

6.4 dcmi_get_card_customized_info 接口原型

函数原型

int dcmi_get_card_customized_info(int card_id, char *info, int len)

功能说明

查询存放在MCU的用户信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元的ID，当前实际支持的ID通过dcmi_get_card_list接口获取。
info	输出	char *	写入的信息。
len	输入	int	info空间的长度。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

表 6-4 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
char info[256] = {0};
ret = dcmi_get_card_customized_info(card_id, info, sizeof(info));
...
```

6.5 dcmi_get_customized_info_api 接口原型

函数原型

int dcmi_get_customized_info_api(int card_id, char *data_info, int *len)

功能说明

查询存放在MCU的用户信息。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
data_info	输出	char *	返回信息的内容。长度至少为256字节。
len	输出	int *	返回信息的长度。

返回值

类型	描述
int	处理结果： ● 成功：返回0 ● 失败：返回码请参见**8 返回码**。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用**6.4 dcmi_get_card_customized_info接口原型**。

表 6-5 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
char data_info[256] = {0};
int len = 0;
ret = dcmi_get_customized_info_api(card_id, data_info, &len);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

6.6 dcmi_mcu_get_license_info 接口原型

函数原型

int dcmi_mcu_get_license_info(int card_id, char *data_info, int *len)

功能说明

查询存放在MCU的用户License证书。

参数说明

参数名称	输入/输出	类型	描述
card_id	输入	int	指定NPU管理单元ID，当前实际支持的ID通过dcmi_get_card_num_list接口获取。
data_info	输出	char *	返回证书的内容。长度至少为256字节。
len	输出	int *	返回证书的长度。

返回值

类型	描述
int	处理结果： • 成功：返回0 • 失败：返回码请参见8 返回码。

异常处理

无。

约束说明

该接口在后续版本将会删除，推荐使用6.4 dcmi_get_card_customized_info接口原型。

表 6-6 部署场景

Linux物理机		Linux物理机容器
root用户	运行用户组（非root用户）	root用户
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
char license[256] = {0};
int len = 0;
ret = dcmi_mcu_get_license_info(card_id, license, &len);
if (ret != 0) {
    //todo：记录日志
    return ret;
}
...
```

7 调用示例

各接口说明处给的是接口的调用片段，此处以调用dcmi_get_card_num_list接口为例，给出调用示例，说明需要include的头文件（dcmi_interface_api.h）、调用接口的逻辑等。

示例代码文件 get_card_num_list.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "dcmi_interface_api.h"

#define MAX_CARD_NUM (16)
#define NPU_OK (0)

int main(int argc, char ** argv)
{
    int ret;
    int card_count = 0;
    int card_id_list[MAX_CARD_NUM] = {0};

    ret = dcmi_init();
    if (ret != NPU_OK) {
        printf("Failed to init dcmi.\n");
        return ret;
    }

    ret = dcmi_get_card_num_list(&card_count, card_id_list, MAX_CARD_NUM);
    if (ret != NPU_OK) {
        printf("Failed to get card number.\n");
        return ret;
    }
    printf("card count is %d\n", card_count);

    return ret;
}
```

从已安装驱动和npu-smi工具的环境的“/usr/local/include/”目录下获取dcmi_interface_api.h文件。

步骤1 以HwHiAiUser用户将get_card_num_list.c、dcmi_interface_api.h传到同一个目录下。

步骤2 以HwHiAiUser用户登录到Host侧系统。

步骤3 执行如下命令，将上传的get_card_num_list.c、dcmi_interface_api.h所属的用户和组修改为HwHiAiUser。

```
chown HwHiAiUser:HwHiAiUser dcmi_interface_api.h
chown HwHiAiUser:HwHiAiUser get_card_num_list.c
```


步骤4 执行如下命令，编译get_card_num_list.c中的代码，生成可执行文件card_list。编译时需要使用驱动中的dsmi相关的驱动，具体如下：

```
gcc get_card_num_list.c -L/usr/lib64 -ldcmi -lascend_hal -ldevmmap -ldrvdsmi -o card_list
```

步骤5 执行可执行文件card_list。

```
./card_list
```

----结束

8 返回码

表 8-1 返回码

DCMI返回码	值	含义
DCMI_OK	0	执行成功。
DCMI_ERR_CODE_INVALID_PARAMETER	-8001	输入参数错误。
DCMI_ERR_CODE_OPER_NOT_PERMITTED	-8002	权限错误。
DCMI_ERR_CODE_MEM_OPERATE_FAIL	-8003	内存接口操作失败。
DCMI_ERR_CODE_SECURE_FUN_FAIL	-8004	安全函数执行失败。
DCMI_ERR_CODE_INNER_ERR	-8005	内部错误。
DCMI_ERR_CODE_TIME_OUT	-8006	响应超时。
DCMI_ERR_CODE_INVALID_DEVICE_ID	-8007	无效的device id。
DCMI_ERR_CODE_DEVICE_NOT_EXIST	-8008	设备不存在。
DCMI_ERR_CODE_IOCTL_FAIL	-8009	ioctl返回失败。
DCMI_ERR_CODE_SEND_MSG_FAIL	-8010	消息发送失败。
DCMI_ERR_CODE_RECV_MSG_FAIL	-8011	消息接收失败。
DCMI_ERR_CODE_NOT_REDAY	-8012	尚未就绪，请重试。
DCMI_ERR_CODE_NOT_SUPPORT_IN_CONTAINER	-8013	在容器中不支持该接口。
DCMI_ERR_CODE_RESET_FAIL	-8015	复位失败。

DCMI返回码	值	含义
DCMI_ERR_CODE_ABORT_OPERATE	-8016	复位取消。
DCMI_ERR_CODE_IS_UPGRADING	-8017	正在升级中。
DCMI_ERR_CODE_RESOURCE_OCCUPIED	-8020	设备资源被占用。
DCMI_ERR_CODE_NOT_SUPPORT	-8255	设备ID/功能不支持。