

Atlas A2 中心推理和训练硬件
24.1.0

DCMI API 参考

文档版本 04
发布日期 2025-04-01



版权所有 © 华为技术有限公司 2025。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

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

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编： 518129

网址： <https://e.huawei.com>

安全声明

产品生命周期政策

华为公司对产品生命周期的规定以“产品生命周期终止政策”为准，该政策的详细内容请参见如下网址：
<https://support.huawei.com/ecolumnsweb/zh/warranty-policy>

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
<https://www.huawei.com/cn/psirt/vul-response-process>
如企业客户须获取漏洞信息，请参见如下网址：
<https://securitybulletin.huawei.com/enterprise/cn/security-advisory>

华为初始证书权责说明

华为公司对随设备出厂的初始数字证书，发布了“华为设备初始数字证书权责说明”，该说明的详细内容请参见如下网址：
<https://support.huawei.com/enterprise/zh/bulletins-service/ENEWS2000015766>

华为企业业务最终用户许可协议(EULA)

本最终用户许可协议是最终用户（个人、公司或其他任何实体）与华为公司就华为软件的使用所缔结的协议。最终用户对华为软件的使用受本协议约束，该协议的详细内容请参见如下网址：
<https://e.huawei.com/cn/about/eula>

产品资料生命周期策略

华为公司针对随产品版本发布的售后客户资料（产品资料），发布了“产品资料生命周期策略”，该策略的详细内容请参见如下网址：
<https://support.huawei.com/enterprise/zh/bulletins-website/ENEWS2000017760>

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 200T A2 Box16 异构子框、Atlas 800T A2 训练服务器、Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件。





仅**Ascend HDK 23.0.RC3及以上**版本支持Atlas 900 A2 PoDc 集群基础单元。由于Atlas 900 A2 PoDc 集群基础单元与Atlas 900 A2 PoD 集群基础单元使用的Ascend HDK软件相同，因此本文以Atlas 900 A2 PoD 集群基础单元统称，不作区分。


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
	表示如不可避免将会导致死亡或严重伤害的具有高等级风险的危害。
	表示如不可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。
	表示如不可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。
	用于传递设备或环境安全警示信息。如不可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。

符号	说明
 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
04	2025-04-01	第四次正式发布。 修改 2.120 dcmi_get_device_cpu_freq_info接口原型 章节的调用示例。
03	2025-03-21	第三次正式发布。 2.111 dcmi_prbs_operate接口原型 章节“参数说明”表中prbs码型添加多次出现满误码情况说明。
02	2025-02-12	第二次正式发布。 <ul style="list-style-type: none">补充超链接优化2.44 dcmi_get_device_hbm_info接口原型、2.49 dcmi_get_device_utilization_rate接口原型章节的约束说明。
01	2025-01-23	第一次正式发布。

目 录

前言..... iii

1 简介..... 1

2 设备管理接口..... 2

2.1 dcmi_init 接口原型..... 6

2.2 dcmi_get_dcmi_version 接口原型..... 7

2.3 dcmi_get_driver_version 接口原型..... 8

2.4 dcmi_get_version 接口原型..... 9

2.5 dcmi_get_card_list 接口原型..... 10

2.6 dcmi_get_card_num_list 接口原型..... 11

2.7 dcmi_get_device_num_in_card 接口原型..... 12

2.8 dcmi_get_device_id_in_card 接口原型..... 13

2.9 dcmi_get_device_type 接口原型..... 15

2.10 dcmi_get_device_chip_info 接口原型..... 16

2.11 dcmi_get_device_pcie_info 接口原型..... 18

2.12 dcmi_get_device_pcie_info_v2 接口原型..... 19

2.13 dcmi_get_pcie_info 接口原型..... 21

2.14 dcmi_get_device_board_info 接口原型..... 23

2.15 dcmi_get_board_info 接口原型..... 24

2.16 dcmi_get_device_elabel_info 接口原型..... 26

2.17 dcmi_get_device_power_info 接口原型..... 28

2.18 dcmi_get_device_die_v2 接口原型..... 29

2.19 dcmi_get_device_die 接口原型..... 31

2.20 dcmi_get_device_ndie 接口原型..... 32

2.21 dcmi_get_device_health 接口原型..... 33

2.22 dcmi_get_driver_health 接口原型..... 35

2.23 dcmi_get_device_errorcode_v2 接口原型..... 36

2.24 dcmi_get_device_errorcode 接口原型..... 37

2.25 dcmi_get_driver_errorcode 接口原型..... 39

2.26 dcmi_get_device_errorcode_string 接口原型..... 40

2.27 dcmi_get_device_errorinfo 接口原型..... 42

2.28 dcmi_get_device_flash_count 接口原型..... 44

2.29 dcmi_get_device_flash_info_v2 接口原型..... 45

2.30 dcmi_get_device_flash_info 接口原型	47
2.31 dcmi_get_device_aicore_info 接口原型	49
2.32 dcmi_get_aicore_info 接口原型	51
2.33 dcmi_get_device_aicpu_info 接口原型	52
2.34 dcmi_get_aicpu_info 接口原型	54
2.35 dcmi_get_device_system_time 接口原型	55
2.36 dcmi_get_system_time 接口原型	56
2.37 dcmi_get_device_temperature 接口原型	58
2.38 dcmi_get_device_voltage 接口原型	59
2.39 dcmi_get_device_pcie_error_cnt 接口原型	60
2.40 dcmi_get_pcie_error_cnt 接口原型	63
2.41 dcmi_get_device_ecc_info 接口原型	66
2.42 dcmi_get_ecc_info 接口原型	69
2.43 dcmi_get_device_frequency 接口原型	71
2.44 dcmi_get_device_hbm_info 接口原型	73
2.45 dcmi_get_hbm_info 接口原型	74
2.46 dcmi_get_device_memory_info_v3 接口原型	76
2.47 dcmi_get_device_memory_info_v2 接口原型	78
2.48 dcmi_get_memory_info 接口原型	79
2.49 dcmi_get_device_utilization_rate 接口原型	81
2.50 dcmi_get_device_sensor_info 接口原型	83
2.51 dcmi_get_soc_sensor_info 接口原型	87
2.52 dcmi_set_container_service_enable 接口原型	90
2.53 dcmi_get_device_board_id 接口原型	91
2.54 dcmi_get_board_id 接口原型	93
2.55 dcmi_get_device_component_count 接口原型	94
2.56 dcmi_get_device_component_list 接口原型	95
2.57 dcmi_get_device_component_static_version 接口原型	99
2.58 dcmi_get_device_cgroup_info 接口原型	103
2.59 dcmi_get_device_llc_perf_para 接口原型	105
2.60 dcmi_set_device_info 接口原型	106
2.60.1 DCMI_MAIN_CMD_LP 命令说明	109
2.60.2 DCMI_MAIN_CMD_QOS 命令说明	111
2.60.3 DCMI_MAIN_CMD_EX_CERT 命令说明	118
2.61 dcmi_get_device_info 接口原型	120
2.61.1 DCMI_MAIN_CMD_DVPP 命令说明	124
2.61.2 DCMI_MAIN_CMD_LP 命令说明	127
2.61.3 DCMI_MAIN_CMD_TS 命令说明	131
2.61.4 DCMI_MAIN_CMD_QOS 命令说明	134
2.61.5 DCMI_MAIN_CMD_HCCS 命令说明	138
2.61.6 DCMI_MAIN_CMD_EX_COMPUTING 命令说明	142
2.61.7 DCMI MAIN CMD EX CERT 命令说明	144

2.61.8 DCMI_MAIN_CMD_PCIE 命令说明.....	148
2.62 dcmi_set_device_sec_revocation 接口原型.....	150
2.63 dcmi_get_device_mac_count 接口原型.....	152
2.64 dcmi_get_device_network_health 接口原型.....	153
2.65 dcmi_get_device_logic_id 接口原型.....	155
2.66 dcmi_get_device_fan_count 接口原型.....	157
2.67 dcmi_get_device_fan_speed 接口原型.....	158
2.68 dcmi_get_card_elabel_v2 接口原型.....	159
2.69 dcmi_get_card_elabel 接口原型.....	161
2.70 dcmi_mcu_get_chip_temperature 接口原型.....	162
2.71 dcmi_get_device_ssh_enable 接口原型.....	164
2.72 dcmi_get_card_board_info 接口原型.....	165
2.73 dcmi_get_card_pcie_info 接口原型.....	167
2.74 dcmi_get_card_pcie_slot 接口原型.....	168
2.75 dcmi_get_fault_device_num_in_card 接口原型.....	169
2.76 dcmi_mcu_check_i2c 接口原型.....	170
2.77 dcmi_mcu_collect_log 接口原型.....	172
2.78 dcmi_get_device_chip_slot 接口原型.....	173
2.79 dcmi_get_product_type 接口原型.....	174
2.80 dcmi_get_device_outband_channel_state 接口原型.....	175
2.81 dcmi_mcu_get_board_info 接口原型.....	177
2.82 dcmi_mcu_get_power_info 接口原型.....	178
2.83 dcmi_get_computing_power_info 接口原型.....	179
2.84 dcmi_get_device_aicpu_count_info 接口原型.....	181
2.85 dcmi_get_first_power_on_date 接口原型.....	182
2.86 dcmi_get_fault_event 接口原型.....	184
2.87 dcmi_get_device_resource_info 接口原型.....	188
2.88 dcmi_get_device_dvpp_ratio_info 接口原型.....	189
2.89 dcmi_get_device_phyid_from_logicid 接口原型.....	191
2.90 dcmi_get_device_logicid_from_phyid 接口原型.....	192
2.91 dcmi_get_card_id_device_id_from_phyid 接口原型.....	194
2.92 dcmi_get_card_id_device_id_from_logicid 接口原型.....	195
2.93 dcmi_get_device_boot_status 接口原型.....	196
2.94 dcmi_sm_encrypt 接口原型.....	198
2.95 dcmi_sm_decrypt 接口原型.....	200
2.96 dcmi_set_power_state 接口原型.....	203
2.97 dcmi_get_npu_work_mode 接口原型.....	205
2.98 dcmi_subscribe_fault_event 接口原型.....	206
2.99 dcmi_get_device_compatibility 接口原型.....	210
2.100 dcmi_get_all_device_count 接口原型.....	212
2.101 dcmi_get_multi_ecc_time_info 接口原型.....	213
2.102 dcmi_get_multi_ecc_record_info 接口原型.....	214

2.103 dcmi_get_pcie_link_bandwidth_info 接口原型.....	216
2.104 dcmi_get_topo_info_by_device_id 接口原型.....	218
2.105 dcmi_get_affinity_cpu_info_by_device_id 接口原型.....	220
2.106 dcmi_get_netdev_pkt_stats_info 接口原型.....	221
2.107 dcmi_get_rdma_bandwidth_info 接口原型.....	227
2.108 dcmi_get_device_hbm_product_info 接口原型.....	229
2.109 dcmi_get_serdes_quality_info 接口原型.....	230
2.110 dcmi_get_mainboard_id 接口原型.....	232
2.111 dcmi_prbs_operate 接口原型.....	233
2.112 dcmi_set_traceroute 接口原型.....	239
2.113 dcmi_get_pfc_duration_info 接口原型.....	242
2.114 dcmi_clear_pfc_duration 接口原型.....	244
2.115 dcmi_get_netdev_tc_stat_info 接口原型.....	245
2.116 dcmi_set_device_clear_tc_pkt_stats 接口原型.....	247
2.117 dcmi_get_qpn_list 接口原型.....	248
2.118 dcmi_get_qp_info 接口原型.....	250
2.119 dcmi_get_hccs_link_bandwidth_info 接口原型.....	252
2.120 dcmi_get_device_cpu_freq_info 接口原型.....	254
2.121 dcmi_get_device_chip_info_v2 接口原型.....	255
3 配置管理接口.....	258
3.1 dcmi_set_device_clear_pcie_error 接口原型.....	259
3.2 dcmi_clear_pcie_error_cnt 接口原型.....	260
3.3 dcmi_get_device_p2p_enable 接口原型.....	261
3.4 dcmi_get_p2p_enable 接口原型.....	262
3.5 dcmi_set_device_clear_ecc_statistics_info 接口原型.....	264
3.6 dcmi_set_device_mac 接口原型.....	265
3.7 dcmi_get_device_mac 接口原型.....	267
3.8 dcmi_get_device_gateway 接口原型.....	268
3.9 dcmi_set_device_gateway 接口原型.....	270
3.10 dcmi_get_device_ip 接口原型.....	272
3.11 dcmi_set_device_ip 接口原型.....	274
3.12 dcmi_set_device_ecc_enable 接口原型.....	276
3.13 dcmi_config_ecc_enable 接口原型.....	278
3.14 dcmi_get_nve_level 接口原型.....	279
3.15 dcmi_set_nve_level 接口原型.....	281
3.16 dcmi_set_device_share_enable 接口原型.....	282
3.17 dcmi_get_device_share_enable 接口原型.....	283
3.18 dcmi_set_device_user_config 接口原型.....	285
3.19 dcmi_set_user_config 接口原型.....	287
3.20 dcmi_get_user_config 接口原型.....	289
3.21 dcmi_clear_device_user_config 接口原型.....	291
3.22 dcmi_get_device_cpu_num_config 接口原型.....	293

3.23 dcmi_set_device_cpu_num_config 接口原型.....	294
3.24 dcmi_create_capability_group 接口原型.....	296
3.25 dcmi_delete_capability_group 接口原型.....	299
3.26 dcmi_get_capability_group_info 接口原型.....	300
3.27 dcmi_get_capability_group_aicore_usage 接口原型.....	304
4 MCU 升级控制接口.....	306
4.1 dcmi_get_mcu_version 接口原型.....	306
4.2 dcmi_mcu_get_version 接口原型.....	307
4.3 dcmi_set_mcu_upgrade_stage 接口原型.....	309
4.4 dcmi_mcu_upgrade_control 接口原型.....	310
4.5 dcmi_set_mcu_upgrade_file 接口原型.....	312
4.6 dcmi_mcu_upgrade_transfile 接口原型.....	313
4.7 dcmi_get_mcu_upgrade_status 接口原型.....	314
4.8 dcmi_mcu_get_upgrade_status 接口原型.....	315
4.9 dcmi_mcu_get_upgrade_statues 接口原型.....	317
5 芯片复位接口.....	319
5.1 使用前必读.....	319
5.2 接口说明.....	319
5.2.1 dcmi_set_device_pre_reset 接口原型.....	319
5.2.2 dcmi_pre_reset_soc 接口原型.....	321
5.2.3 dcmi_set_device_reset 接口原型.....	322
5.2.4 dcmi_reset_device 接口原型.....	323
5.2.5 dcmi_set_device_rescan 接口原型.....	325
5.2.6 dcmi_rescan_soc 接口原型.....	326
5.2.7 dcmi_reset_device_inband 接口原型.....	327
5.3 调用示例.....	329
5.3.1 带内复位调用示例.....	329
5.3.2 带外复位调用示例.....	330
6 用户自定义信息配置接口.....	332
6.1 dcmi_set_card_customized_info 接口原型.....	332
6.2 dcmi_set_customized_info_api 接口原型.....	333
6.3 dcmi_mcu_set_license_info 接口原型.....	335
6.4 dcmi_get_card_customized_info 接口原型.....	336
6.5 dcmi_get_customized_info_api 接口原型.....	337
6.6 dcmi_mcu_get_license_info 接口原型.....	338
7 调用示例.....	340
8 常用操作.....	342
8.1 准备 ipmitool 软件.....	342
8.2 查询 NPU 业务进程.....	342

9 返回码..... 344

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。

接口场景释义：

表 1-1 场景释义

场景	说明
Linux物理机	物理机场景
Linux物理机容器	物理机映射容器场景

说明

本文涉及的容器场景除特殊说明外，均指普通容器。

2 设备管理接口

- [2.1 dcmi_init接口原型](#)
- [2.2 dcmi_get_dcmi_version接口原型](#)
- [2.3 dcmi_get_driver_version接口原型](#)
- [2.4 dcmi_get_version接口原型](#)
- [2.5 dcmi_get_card_list接口原型](#)
- [2.6 dcmi_get_card_num_list接口原型](#)
- [2.7 dcmi_get_device_num_in_card接口原型](#)
- [2.8 dcmi_get_device_id_in_card接口原型](#)
- [2.9 dcmi_get_device_type接口原型](#)
- [2.10 dcmi_get_device_chip_info接口原型](#)
- [2.11 dcmi_get_device_pcie_info接口原型](#)
- [2.12 dcmi_get_device_pcie_info_v2接口原型](#)
- [2.13 dcmi_get_pcie_info接口原型](#)
- [2.14 dcmi_get_device_board_info接口原型](#)
- [2.15 dcmi_get_board_info接口原型](#)
- [2.16 dcmi_get_device_elabel_info接口原型](#)
- [2.17 dcmi_get_device_power_info接口原型](#)
- [2.18 dcmi_get_device_die_v2接口原型](#)
- [2.19 dcmi_get_device_die接口原型](#)
- [2.20 dcmi_get_device_ndie接口原型](#)
- [2.21 dcmi_get_device_health接口原型](#)
- [2.22 dcmi_get_driver_health接口原型](#)
- [2.23 dcmi_get_device_errorcode_v2接口原型](#)

- [2.24 dcmi_get_device_errorcode接口原型](#)
- [2.25 dcmi_get_driver_errorcode接口原型](#)
- [2.26 dcmi_get_device_errorcode_string接口原型](#)
- [2.27 dcmi_get_device_errorinfo接口原型](#)
- [2.28 dcmi_get_device_flash_count接口原型](#)
- [2.29 dcmi_get_device_flash_info_v2接口原型](#)
- [2.30 dcmi_get_device_flash_info接口原型](#)
- [2.31 dcmi_get_device_aicore_info接口原型](#)
- [2.32 dcmi_get_aicore_info接口原型](#)
- [2.33 dcmi_get_device_aicpu_info接口原型](#)
- [2.34 dcmi_get_aicpu_info接口原型](#)
- [2.35 dcmi_get_device_system_time接口原型](#)
- [2.36 dcmi_get_system_time接口原型](#)
- [2.37 dcmi_get_device_temperature接口原型](#)
- [2.38 dcmi_get_device_voltage接口原型](#)
- [2.39 dcmi_get_device_pcie_error_cnt接口原型](#)
- [2.40 dcmi_get_pcie_error_cnt接口原型](#)
- [2.41 dcmi_get_device_ecc_info接口原型](#)
- [2.42 dcmi_get_ecc_info接口原型](#)
- [2.43 dcmi_get_device_frequency接口原型](#)
- [2.44 dcmi_get_device_hbm_info接口原型](#)
- [2.45 dcmi_get_hbm_info接口原型](#)
- [2.46 dcmi_get_device_memory_info_v3接口原型](#)
- [2.47 dcmi_get_device_memory_info_v2接口原型](#)
- [2.48 dcmi_get_memory_info接口原型](#)
- [2.49 dcmi_get_device_utilization_rate接口原型](#)
- [2.50 dcmi_get_device_sensor_info接口原型](#)
- [2.51 dcmi_get_soc_sensor_info接口原型](#)
- [2.52 dcmi_set_container_service_enable接口原型](#)
- [2.53 dcmi_get_device_board_id接口原型](#)
- [2.54 dcmi_get_board_id接口原型](#)
- [2.55 dcmi_get_device_component_count接口原型](#)
- [2.56 dcmi_get_device_component_list接口原型](#)

- [2.57 dcmi_get_device_component_static_version接口原型](#)
- [2.58 dcmi_get_device_cgroup_info接口原型](#)
- [2.59 dcmi_get_device_llc_perf_para接口原型](#)
- [2.60 dcmi_set_device_info接口原型](#)
- [2.61 dcmi_get_device_info接口原型](#)
- [2.62 dcmi_set_device_sec_revocation接口原型](#)
- [2.63 dcmi_get_device_mac_count接口原型](#)
- [2.64 dcmi_get_device_network_health接口原型](#)
- [2.65 dcmi_get_device_logic_id接口原型](#)
- [2.66 dcmi_get_device_fan_count接口原型](#)
- [2.67 dcmi_get_device_fan_speed接口原型](#)
- [2.68 dcmi_get_card_elabel_v2接口原型](#)
- [2.69 dcmi_get_card_elabel接口原型](#)
- [2.70 dcmi_mcu_get_chip_temperature接口原型](#)
- [2.71 dcmi_get_device_ssh_enable接口原型](#)
- [2.72 dcmi_get_card_board_info接口原型](#)
- [2.73 dcmi_get_card_pcie_info接口原型](#)
- [2.74 dcmi_get_card_pcie_slot接口原型](#)
- [2.75 dcmi_get_fault_device_num_in_card接口原型](#)
- [2.76 dcmi_mcu_check_i2c接口原型](#)
- [2.77 dcmi_mcu_collect_log接口原型](#)
- [2.78 dcmi_get_device_chip_slot接口原型](#)
- [2.79 dcmi_get_product_type接口原型](#)
- [2.80 dcmi_get_device_outband_channel_state接口原型](#)
- [2.81 dcmi_mcu_get_board_info接口原型](#)
- [2.82 dcmi_mcu_get_power_info接口原型](#)
- [2.83 dcmi_get_computing_power_info接口原型](#)
- [2.84 dcmi_get_device_aicpu_count_info接口原型](#)
- [2.85 dcmi_get_first_power_on_date接口原型](#)
- [2.86 dcmi_get_fault_event接口原型](#)
- [2.87 dcmi_get_device_resource_info接口原型](#)
- [2.88 dcmi_get_device_dvpp_ratio_info接口原型](#)
- [2.89 dcmi_get_device_phyid_from_logicid接口原型](#)

- [2.90 dcmi_get_device_logicid_from_phyid接口原型](#)
- [2.91 dcmi_get_card_id_device_id_from_phyid接口原型](#)
- [2.92 dcmi_get_card_id_device_id_from_logicid接口原型](#)
- [2.93 dcmi_get_device_boot_status接口原型](#)
- [2.94 dcmi_sm_encrypt接口原型](#)
- [2.95 dcmi_sm_decrypt接口原型](#)
- [2.96 dcmi_set_power_state接口原型](#)
- [2.97 dcmi_get_npu_work_mode接口原型](#)
- [2.98 dcmi_subscribe_fault_event接口原型](#)
- [2.99 dcmi_get_device_compatibility接口原型](#)
- [2.100 dcmi_get_all_device_count接口原型](#)
- [2.101 dcmi_get_multi_ecc_time_info接口原型](#)
- [2.102 dcmi_get_multi_ecc_record_info接口原型](#)
- [2.103 dcmi_get_pcie_link_bandwidth_info接口原型](#)
- [2.104 dcmi_get_topo_info_by_device_id接口原型](#)
- [2.105 dcmi_get_affinity_cpu_info_by_device_id接口原型](#)
- [2.106 dcmi_get_netdev_pkt_stats_info接口原型](#)
- [2.107 dcmi_get_rdma_bandwidth_info接口原型](#)
- [2.108 dcmi_get_device_hbm_product_info接口原型](#)
- [2.109 dcmi_get_serdes_quality_info接口原型](#)
- [2.110 dcmi_get_mainboard_id接口原型](#)
- [2.111 dcmi_prbs_operate接口原型](#)
- [2.112 dcmi_set_traceroute接口原型](#)
- [2.113 dcmi_get_pfc_duration_info接口原型](#)
- [2.114 dcmi_clear_pfc_duration 接口原型](#)
- [2.115 dcmi_get_netdev_tc_stat_info接口原型](#)
- [2.116 dcmi_set_device_clear_tc_pkt_stats接口原型](#)
- [2.117 dcmi_get_qpn_list接口原型](#)
- [2.118 dcmi_get_qp_info接口原型](#)
- [2.119 dcmi_get_hccs_link_bandwidth_info接口原型](#)
- [2.120 dcmi_get_device_cpu_freq_info接口原型](#)
- [2.121 dcmi_get_device_chip_info_v2接口原型](#)

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。默认取值为-1，无实际意义。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
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） INVALID_TYPE = 0xFF };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。

参数名称	输入/输出	类型	描述
chip_info	输出	struct dcmi_chip_info *	芯片信息。 struct dcmi_chip_info { unsigned char chip_type[MAX_CHIP_NAME_LEN]; // 芯片类型 unsigned char chip_name[MAX_CHIP_NAME_LEN]; // 芯片名称 unsigned char chip_ver[MAX_CHIP_NAME_LEN]; // 芯片版本 unsigned int aicore_cnt; // aicore数量（仅NPU类型具有） }; 其中MAX_CHIP_NAME_LEN为32 输入device_id为MCU芯片ID时，aicore_cnt无效。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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; //设备id unsigned int venderid; //厂商id unsigned int subvenderid; //厂商子id unsigned int subdeviceid; //设备子id unsigned int bdf_deviceid; // BDF（Bus, Device, Function）中的设备ID unsigned int bdf_busid; // BDF（Bus, Device, Function）中的总线ID unsigned int bdf_funcid; // BDF（Bus, Device, Function）中的功能ID };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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; //厂商id unsigned int subvenderid; //厂商子id unsigned int deviceid; //设备id unsigned int subdeviceid; //设备子id int domain; // pcie domain unsigned int bdf_busid; //BDF (Bus, Device, Function) 中的总线ID unsigned int bdf_deviceid; //BDF (Bus, Device, Function) 中的设备ID unsigned int bdf_funcid; //BDF (Bus, Device, Function) 中的功能ID unsigned char reserve[32]; };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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]。

参数名称	输入/输出	类型	描述
pcie_idinfo	输出	struct dcmi_tag_pcie_idinfo *	PCIe信息。 struct dcmi_tag_pcie_idinfo{ unsigned int deviceid; //设备id unsigned int venderid; //厂商id unsigned int subvenderid; //厂商子id unsigned int subdeviceid; //设备子id unsigned int bdf_deviceid; // BDF (Bus, Device, Function) 中的设备ID unsigned int bdf_busid; // BDF (Bus, Device, Function) 中的总线ID unsigned int bdf_funcid; // BDF (Bus, Device, Function) 中的功能ID };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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]。 MCU芯片：mcu_id。
board_info	输出	struct dcmi_board_info*	board信息。 struct dcmi_board_info { unsigned int board_id; //单板的board_id unsigned int pcb_id; //PCB版本编号 unsigned int bom_id; //BOM版本编号 unsigned int slot_id; //槽位号信息 }; 在物理机场景下： <ul style="list-style-type: none">输入device_id为NPU芯片ID时，只有board_id和slot_id有效，其中slot_id标识芯片所在的pcie槽位号。输入device_id为MCU芯片ID时，board_id、pcb_id、bom_id、slot_id有效，其中slot_id标识芯片在卡上位置标识。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
board_info	输出	struct dcmi_board_info_stru*	board信息。 struct dcmi_board_info_stru { unsigned int board_id; //单板的board_id unsigned int pcb_id; //PCB版本编号 unsigned int bom_id; //BOM版本编号 unsigned int slot_id; //槽位号信息 }; 在物理机场景下： <ul style="list-style-type: none">输入device_id为NPU芯片ID时，只有board_id和slot_id有效，其中slot_id标识芯片所在的pcie槽位号。输入device_id为MCU芯片ID时，board_id、pcb_id、bom_id、slot_id有效，其中slot_id标识芯片在卡上位置标识。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。

参数名称	输入/输出	类型	描述
elabel_info	输出	struct dcmi_elabel_info *	电子标签信息。 struct dcmi_elabel_info { char product_name[MAX_LENTH]; //产品名称 char model[MAX_LENTH]; //产品型号 char manufacturer[MAX_LENTH]; //生产厂商 char manufacturer_date[MAX_LENTH]; //生产日期 char serial_number[MAX_LENTH]; //产品序列号 }; 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。
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。 不支持输入MCU芯片ID，查询功耗。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 };

参数名称	输入/输出	类型	描述
die_id	输出	struct dcmi_die_id *	die id信息 struct dcmi_die_id { unsigned int soc_die[DIE_ID_COUNT]; //芯片Die编号 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]; //芯片Die编号 }

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]; //芯片Die编号 }

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
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： <ul style="list-style-type: none">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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">0：正常1：一般告警（预留）2：重要告警（预留）3：紧急告警fffffff：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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
error_count	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《黑匣子错误码信息列表》或《健康管理故障定义》进行查看。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
error_count	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》或《健康管理故障定义》进行查看。
error_width	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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
N	N	N

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。
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《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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
error_code	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取值范围是大于等于48字节。 <ul style="list-style-type: none">若设置的error_info小于48字节，则系统报错。若设置的error_info在48~255字节之间，则在《健康管理故障定义》中的故障码，查询出来的故障信息为简化信息。若设置的error_info大于等于256字节，则查询出来的故障信息为实际故障信息。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

调用该接口查询到的信息仅代表当前芯片设计了这种错误码类型，具有上报这种故障类型的能力，但不代表当前已经使用这个错误码。当前芯片已经支持的错误码请按照[参数说明](#)表中error_code参数的描述获取。

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 (256)
...
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。

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]。 MCU芯片：mcu_id。
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取值范围是大于等于48字节。 <ul style="list-style-type: none">• 若设置的errorinfo小于48字节，则系统报错。• 若设置的errorinfo在48~255字节之间，则在《健康管理故障定义》中的故障码，查询出来的故障信息为简化信息。• 若设置的errorinfo大于等于256字节，则查询出来的故障信息为实际故障信息。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">• 成功：返回0。• 失败：返回码请参见9 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 (256)
...
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]。

参数名称	输入/输出	类型	描述
flash_count	输出	unsigned int *	返回Flash个数。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]
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; //Flash id unsigned short device_id; //设备id unsigned short vendor; // 厂商id unsigned int state; //Flash health, 0x8表示正常，0x10表示非正常 unsigned long long size; //Flash大小，单位Byte unsigned int sector_count; //擦除单元数量 unsigned short manufacturer_id; //制造商id };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。

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]。
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]
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; //Flash_id unsigned short device_id; //设备id unsigned short vendor; //厂商id unsigned int state; //state=0时，表示flash正常，state≠0时，表示flash异常 unsigned long long size; //Flash的总大小 unsigned int sector_count; //擦除单元数量 unsigned short manufacturer_id; //制造商id };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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; //额定频率，单位是MHz unsigned int curfreq; //当前频率，单位是MHz };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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; // AICPU的最大运行频率，单位是MHz unsigned int cur_freq; // AICPU的当前运行频率，单位是MHz unsigned int aicpu_num; //AICPU的个数 unsigned int util_rate[MAX_CORE_NUM]; //AICPU的利用率 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 [16]; //AICPU的利用率 }

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
temperature	输出	int *	芯片温度：单位为摄氏度。 输入device_id为MCU芯片ID时，针对I2C协议，如果温度为无效数据则为0x7ffd，如果温度读取失败则为0x7fff。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。

参数名称	输入/输出	类型	描述
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。 输入device_id为MCU芯片ID时，针对I2C协议，如果电压为无效数据则为0x7ffd，如果电压读取失败则为0x7fff。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 *	<div>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;//symbol_unlock错误计数 unsigned int pcs_rx_err_cnt;//PCS层接收错误计数 unsigned int phy_lane_err_counter;//lane错误计数 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;//PCIe DLLP Lcrc的错误计数 unsigned int dl_dcrc_err_num;//PCIe DLLP Dcrc的错误计数 };</div>

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 *	<div>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;//symbol_unlock错误计数 unsigned int pcs_rx_err_cnt;//PCS层接收错误计数 unsigned int phy_lane_err_counter;//lane错误计数 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;//PCIe DLLP Lcrc的错误计数 unsigned int dl_dcrc_err_num;//PCIe DLLP Dcrc的错误计数 }</div>

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。
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 }; 仅支持DCMI_DEVICE_TYPE_HBM 片上内存类型内存。

参数名称	输入/输出	类型	描述
device_ecc_info	输出	struct dcmi_ecc_info *	<p>返回ECC结构体信息：</p> <pre>struct dcmi_ecc_info { int enable_flag; unsigned int single_bit_error_cnt; //单比特错误数量 unsigned int double_bit_error_cnt; //多比特错误数量 unsigned int total_single_bit_error_cnt; // 生命周期内所有单比特ecc错误统计 unsigned int total_double_bit_error_cnt; // 生命周期内所有多比特ecc错误统计 unsigned int single_bit_isolated_pages_cnt; //单比特错误隔离内存页数量 unsigned int double_bit_isolated_pages_cnt; // 多比特错误隔离内存页数量 };</pre> <p>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。</p> <p>说明</p> <ul style="list-style-type: none">• 当前支持单比特和多比特错误查询，如果查询到多比特错误，需要进行相关问题定位和处理。• 该接口获取的是实时的数据，系统重启之后单比特错误数量和多比特错误数量清零。

返回值

类型	描述
int	<p>处理结果：</p> <ul style="list-style-type: none">• 成功：返回0。• 失败：返回码请参见9 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	Y

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。
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	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_ADD R, DCMI_HBM_RECORDED_MULTI_ADDR, DCMI_DEVICE_TYPE_NONE = 0xff }; 仅支持DCMI_DEVICE_TYPE_HBM //片上内存类型内存。
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; //单bit错误数量 unsigned int double_bit_error_count; //多bit错误数量 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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	Y

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，目前支持如下几种，数值和具体设备类型对应如下。 支持2、6、7、9这几种类型。 <ul style="list-style-type: none">1：内存2：控制CPU6：片上内存7：AI Core当前频率9：AI Core额定频率12：Vector Core当前频率 说明 AI Core额定频率：AI Core表示在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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件的片上内存为32G，无业务时系统占用3G；Atlas 900 A2 PoD 集群基础单元、Atlas 800T A2 训练服务器、Atlas 200T A2 Box16 异构子框、Atlas 800I A2 推理服务器、A200I A2 Box 异构组件的片上内存为64G，无业务时系统占用4G。

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

获取芯片的片上内存信息。

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*	返回片上内存信息，片上内存信息结构体： struct dcmi_hbm_info { unsigned long long memory_size; //片上内存总大小，单位MB unsigned int freq; // 片上内存 频率，单位MHz unsigned long long memory_usage; // 片上内存已用内存 int temp; //片上内存温度 unsigned int bandwidth_util_rate; //带宽利用率 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件的片上内存为32G，无业务时系统占用3G；Atlas 900 A2 PoD 集群基础单元、Atlas 800T A2 训练服务器、Atlas 200T A2 Box16 异构子框、Atlas 800I A2 推理服务器、A200I A2 Box 异构组件的片上内存为64G，无业务时系统占用4G。

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
Y	Y	Y

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

获取芯片的片上内存信息。

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]。
pdevice_hbm_info	输出	struct dsmi_hbm_info_stru*	返回片上内存信息，片上内存信息结构体： struct dsmi_hbm_info_stru { unsigned long long memory_size; //片上内存总大小，单位MB unsigned int freq; // 片上内存 频率，单位MHz unsigned long long memory_usage; // 片上内存已用内存 int temp; //片上内存温度 unsigned int bandwidth_util_rate; //带宽利用率 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件的片上内存为32G，无业务时系统占用3G；Atlas 900 A2 PoD 集群基础单元、Atlas

800T A2 训练服务器、Atlas 200T A2 Box16 异构子框、Atlas 800I A2 推理服务器、A200I A2 Box 异构组件的片上内存为64G，无业务时系统占用4G。

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
Y	Y	Y

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]。

参数名称	输入/输出	类型	描述
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; //内存大小， 单位MB unsigned long long memory_available; //可用内存大小， 计算方法： free + hugepages_free * hugepagesize unsigned int freq; //频率MHz unsigned long hugepagesize; //大页的大小， 单位KB unsigned long hugepages_total; //总大页数量 unsigned long hugepages_free; //可用大页数量 unsigned int utiliza; //DDR内存的使用量 unsigned char reserve[60]; //保留 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

推荐使用[2.44 dcmi_get_device_hbm_info接口原型](#)。

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

Linux物理机		Linux物理机容器
N	N	N

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; //内存大小，单位MB unsigned int freq; //频率MHz unsigned int utiliza; //利用率 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 2-47 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
N	N	N

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;//内存大小，单位MB unsigned int freq; //频率MHz unsigned int utiliza; //利用率 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 2-48 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
N	N	N

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]。

参数名称	输入/输出	类型	描述
input_type	输入	int	<p>设备类型，目前支持如下几种，数值和具体设备类型对应如下。</p> <p>支持2、3、4、6、10、12、13这几种类型。</p> <ul style="list-style-type: none">1：内存2：AI Core3：AI CPU4：控制CPU5：内存带宽6：片上内存8：DDR10：片上内存带宽12：vector core13：NPU整体利用率 <p>说明</p> <p>当设备类型为AI Core或vector core时开启 profiling，查询占用率为0，实际无意义。</p> <p>在直通虚拟机场景、直通虚拟机容器场景仅支持AI Core（2）、片上内存（6）、片上内存带宽（10）、vector core（12），其它参数返回不支持。该场景下获取的片上内存带宽为0，实际无意义。</p>
utilization_rate	输出	unsigned int *	处理器利用率，单位：%。

返回值

类型	描述
int	<p>处理结果：</p> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件的片上内存为32G，无业务时系统占用3G；Atlas 900 A2 PoD 集群基础单元、Atlas 800T A2 训练服务器、Atlas 200T A2 Box16 异构子框、Atlas 800I A2 推理服务器、A200I A2 Box 异构组件的片上内存为64G，无业务时系统占用4G。

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	<p>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 };</p> <p>指定传感器索引，具体如下值：</p> <p>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；</p> <p>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；</p> <p>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；</p> <p>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；</p> <p>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；</p> <p>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；</p> <p>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；</p>

参数名称	输入/输出	类型	描述
			<p>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;</p> <p>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;</p> <p>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;</p> <p>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。</p> <p>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;</p> <p>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;</p> <p>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;</p> <p>14: DCMI_HBM_TEMP_ID, 表示片上内存最高温度; 返回值对应输出联合体中的signed int iint成员。</p> <p>该场景支持11~14。</p> <p>说明</p> <p>DCMI_FP_TEMP_ID:</p> <ul style="list-style-type: none">• 不插光模块时, 会返回-8008。• 若插入光模块时, 显示正常温度。• 若插入铜缆时, 不支持温度查询, 返回0x7EFF。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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	<div>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 };</div> <div>指定传感器索引，具体如下值：</div> <div>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；</div> <div>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；</div> <div>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；</div> <div>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；</div> <div>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；</div> <div>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；</div> <div>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；</div>

参数名称	输入/输出	类型	描述
			<p>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;</p> <p>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;</p> <p>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;</p> <p>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。</p> <p>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;</p> <p>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;</p> <p>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;</p> <p>14: DCMI_HBM_TEMP_ID, 表示片上内存最高温度; 返回值对应输出联合体中的signed int iint成员。</p> <p>该场景支持11~14。</p> <p>说明</p> <p>DCMI_FP_TEMP_ID:</p> <ul style="list-style-type: none">• 不插光模块时, 会返回-8008。• 若插入光模块时, 显示正常温度。• 若插入铜缆时, 不支持温度查询, 返回0x7EFF。
sensor_info	输出	union tag_sensor_info *	<p>返回温度传感器结构体信息:</p> <pre>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]; };</pre>

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
board_id	输出	unsigned int*	单板的ID。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。 MCU芯片：mcu_id。
board_id	输出	int *	Board ID信息。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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，不包含recovery组件。

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 = 0xFFFFFFFF7, DCMI_UPGRADE_ALL_IMAGE_COMPONENT = 0xFFFFFFFFD, DCMI_UPGRADE_ALL_FIRMWARE_COMPONENT = 0xFFFFFFFFE, DCMI_UPGRADE_ALL_COMPONENT = 0xFFFFFFFFF }; 当前支持: DCMI_COMPONENT_TYPE_HBOOT1_A、 DCMI_COMPONENT_TYPE_HBOOT1_B、 DCMI_COMPONENT_TYPE_HILINK、 DCMI_COMPONENT_TYPE_HILINK2。
component_count	输入	unsigned int	“component_table”数组的长度，表示获取的组件个数。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
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。

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]。

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 = 0xFFFFFFFF7, DCMI_UPGRADE_ALL_IMAGE_COMPONENT = 0xFFFFFFFFD, DCMI_UPGRADE_ALL_FIRMWARE_COMPONENT = 0xFFFFFFFFE, DCMI_UPGRADE_ALL_COMPONENT = 0xFFFFFFFFF }; 当前支持 DCMI_COMPONENT_TYPE_HBOOT1_A、 DCMI_COMPONENT_TYPE_HBOOT1_B、 DCMI_COMPONENT_TYPE_HILINK、 DCMI_COMPONENT_TYPE_HILINK2。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口需调入TEEOS，耗时较长，不支持在接口调用时触发休眠唤醒，如果触发休眠，有较大可能造成休眠失败。

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_cgrou p_info*	cgroup信息。 struct dcmi_cgroup_info { unsigned long limit_in_bytes; //允许最大使用内存数 unsigned long max_usage_in_bytes; //历史记录中使用的最大内存数 unsigned long usage_in_bytes; //当前内存使用率 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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_HOST_AICPU, DCMI_MAIN_CMD_TEMP = 50, DCMI_MAIN_CMD_SVM, DCMI_MAIN_CMD_VDEV_MNG, DCMI_MAIN_CMD_SEC, DCMI_MAIN_CMD_PCIE = 55, DCMI_MAIN_CMD_SIO = 56, DCMI_MAIN_CMD_EX_COMPUTING = 0x8000, DCMI_MAIN_CMD_DEVICE_SHARE = 0x8001, DCMI_MAIN_CMD_EX_CERT = 0x8003, DCMI_MAIN_CMD_MAX }; 仅支持DCMI_MAIN_CMD_LP //低功耗模块主命令字、 DCMI_MAIN_CMD_QOS //qos模块主命令字、 DCMI_MAIN_CMD_EX_CERT //证书管理模块主命令字

参数名称	输入/输出	类型	描述
sub_cmd	输入	unsigned int	详细参见子章节中的功能说明。
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。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-60 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;
}
...
```

2.60.1 DCMI_MAIN_CMD_LP 命令说明

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LP相关配置。

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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_LP

参数名称	输入/输出	类型	描述
sub_cmd	输入	unsigned int	<pre>typedef enum { // 获取AICORE电压电流的寄存器值 DCMI_LP_SUB_CMD_AICORE_VOLTAGE_CURRENT = 0, // 获取HYBIRD电压电流的寄存器值 DCMI_LP_SUB_CMD_HYBIRD_VOLTAGE_CURRENT, // 获取CPU电压电流的寄存器值 DCMI_LP_SUB_CMD_TAISHAN_VOLTAGE_CURRENT, // 获取DDR电压电流的寄存器值 DCMI_LP_SUB_CMD_DDR_VOLTAGE_CURRENT, // 获取ACG调频计数值 DCMI_LP_SUB_CMD_ACG, // 获取低功耗总状态 DCMI_LP_SUB_CMD_STATUS, // 获取所有工作档位 DCMI_LP_SUB_CMD_TOPS_DETAILS, // 设置工作档位 DCMI_LP_SUB_CMD_SET_WORK_TOPS, // 获取当前工作档位 DCMI_LP_SUB_CMD_GET_WORK_TOPS, // 获取当前降频原因 DCMI_LP_SUB_CMD_AICORE_FREQREDUC_CAUSE, // 获取功耗信息 DCMI_LP_SUB_CMD_GET_POWER_INFO, // 设置IDLE模式开关 DCMI_LP_SUB_CMD_SET_IDLE_SWITCH, DCMI_LP_SUB_CMD_MAX, } DCMI_LP_SUB_CMD;</pre> 当前仅支持DCMI_LP_SUB_CMD_SET_IDLE_SWITCH命令。
buf	输入	const void *	详见本节约束说明。
buf_size	输入	unsigned int	buf数组的长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-61 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_IDLE_SWITCH	unsigned char	长度为： sizeof(unsigned char)	0：表示idle模式关闭 1：表示idle模式开启 其他值无效。

表 2-62 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;
DCMI_MAIN_CMD main_cmd = DCMI_MAIN_CMD_LP;
unsigned int sub_cmd = DCMI_LP_SUB_CMD_SET_WORK_TOPS;
DCMI_LP_WORK_TOPS_STRU set_works_tops_lv = {0};
set_works_tops_lv.work_tops = 1;
set_works_tops_lv.is_in_flash = 0;
unsigned int size = sizeof(DCMI_LP_WORK_TOPS_STRU);
ret = dcmi_set_device_info(card_id, device_id, main_cmd, sub_cmd, (void *)&set_works_tops_lv, size);
if (ret) {
    // todo
}
// todo
...
```

2.60.2 DCMI_MAIN_CMD_QOS 命令说明

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

配置QOS相关信息，包括配置mpamid对应的QOS信息、配置master对应的QOS信息、配置带宽统计功能。

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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_QOS
sub_cmd	输入	unsigned int	<pre>typedef enum { // 配置指定mpamid的信息 DCMI_QOS_SUB_MATA_CONFIG, // 配置指定master的信息 DCMI_QOS_SUB_MASTER_CONFIG, // 配置带宽的统计功能 DCMI_QOS_SUB_BW_DATA, // 配置通用信息 DCMI_QOS_SUB_GLOBAL_CONFIG, } DCMI_QOS_SUB_INFO;</pre>
buf	输入	const void *	详细见本节约束说明。
buf_size	输入	unsigned int	buf数组的长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-63 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_QOS_SUB_MATA_CONFIG	struct dcmi_qos_mata_config { int mpamid; unsigned int bw_high; unsigned int bw_low; int hardlimit; int reserved[DCMI_QOS_CFG_RESERVE_D_LEN]; };	sizeof(dcmi_qos_mata_config)	<ul style="list-style-type: none">● mpamid：取值范围是 [0,127]。● bw_high：上水线(GB/s)，取值范围是 [0,1638]。【注意】取值范围上限随颗粒主频变化，计算公式为： bw_high_max= ddr_freq * 2 * max_chn * 8 / 1000，以实际主频和通道数为准。● bw_low：下水线(GB/s)，取值范围是[0, bw_high]● hardlimit：<ul style="list-style-type: none">- 1表示开启- 0表示不开启 <p>用户通过该接口读取水线，可能同先前用户配置值存在误差。误差值计算方式为：处理器最大带宽/ MAX_REG_VALUE。其中，MAX_REG_VALUE 取值为1024。</p>

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_QOS_SUB_MASTER_CONFIG	struct dcmi_qos_master_config { int master; int mpamid; int qos; int pmg; unsigned long long bitmap[4]; /* max support 64 * 4 */ int reserved[DCMI_QOS_CFG_RESERVE D_LEN]; };	sizeof(dcmi_qos_master_config)	<ul style="list-style-type: none">• master: master id, 支持配置的项为: vdec=1,vpc=2,jpge=3,jpgd=4,pcie=7,sdma=13• mpamid: 取值范围是 [0,127]。• qos: 带宽调度优先级, 取值范围[0,7], 0作为hardlimit专用qos, 7为调度绿色通道qos• pmg: mpamid分组, 取值范围是[0,3]• bitmap: 因框架限制, 不支持。

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_QOS_SUB_BW_DATA	struct dcmi_qos_bw_config { u8 mode; u8 state; u8 cnt; u8 method; u32 interval; u32 target_set[16]; int reserved_1[8]; };	sizeof(dcmi_qos_bw_config)	<p>interval: 带宽采样时间间隔(单位us) 需要配置大于1000</p> <ul style="list-style-type: none">mode: 带宽统计的模式, 0表示自动模式, 1表示手动模式。该配置只有在MATA/DHA侧生效, DDRC侧配置不生效且只支持手动模式。method: 选择带宽统计的节点, 0表示在MATA/DHA侧对带宽进行统计, 1表示在DDRC侧对带宽进行统计。state: 带宽采样下发命令。<ul style="list-style-type: none">2表示开启, 此时读取的值有效。1表示初始化。0表示关闭。target_set: 采样对象的mpamid, 最多可存放16个, 实际支持的数量与昇腾AI处理器有关。cnt: 有效带宽统计对象的数量, 最多可支持16个。

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_QOS_SUB_GLOBAL_CONFIG	<pre>struct dcmi_qos_gbl_config { unsigned int enable; unsigned int autoqos_fuse_en; /* 0--enable, 1-- disable */ unsigned int mpamqos_fuse_m ode; /* 0-- average, 1--max, 2--replace */ unsigned int mpam_subtype; /* 0--all, 1--wr, 2-- rd, 3--none */ int reserved[DCMI_Q OS_CFG_RESERVE D_LEN]; };</pre>	sizeof(dcmi_qos_gbl_config)	<ul style="list-style-type: none">● enable: 是否使能QOS功能。<ul style="list-style-type: none">- 0表示不使能。- 1表示使能。● autoqos_fuse_en: qos的融合开关。当前不支持设置autoqos_fuse_en, 默认值为1。<ul style="list-style-type: none">- 0表示关闭qos融合。- 1表示开启qos融合。● mpamqos_fuse_mode: qos的融合模式, autoqos_fuse_en开启的条件下生效。<ul style="list-style-type: none">- 0表示均值融合。- 1表示取随路qos和mpamqos之间的最大值作为融合结果。- 2表示使用随路qos替换mpamqos。● mpam_subtype: 带宽统计的模式。<ul style="list-style-type: none">- 0表示统计读+写带宽。- 1表示统计写带宽。- 2表示统计读带宽。

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
Y	N	N

调用示例

```
...
int32_t ret;
int card_id = 0;
int device_id = 0;
// 配置mpamid上下水线，用于带宽限制
struct qos_mata_config mataCfg = {0};
mataCfg.mpamid = 0; // 举例，mpamid配置为0
mataCfg.bw_high = 20;
mataCfg.bw_low = 10;
mataCfg.hardlimit = 1;
ret = dcmi_set_device_info(card_id, device_id,
DCMI_MAIN_CMD_QOS,static_cast<uint32_t>(DCMI_QOS_SUB_MATA_CONFIG),
static_cast<void*>(&mataCfg), sizeof(struct qos_mata_config));
if (ret != 0) {
printf("[dev:%d]set mata qos config failed, ret = %d\n", devId, ret);
return ret;
}
// 配置master对应的mpamid、qos、pmg
struct qos_master_config masterCfg = {0};
masterCfg.master = 0;
masterCfg.mpamid = 1; // 举例，mpamid配置为1
masterCfg.qos = 3;
masterCfg.pmg = 0;
masterCfg.bitmap[0] = 0xffff0000;
masterCfg.bitmap[1] = 0xffff;
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_QOS,
static_cast<uint32_t>(DCMI_QOS_SUB_MASTER_CONFIG),
static_cast<void*>(&cfg), sizeof(struct qos_master_config));
if (ret != 0) {
printf("[dev:%d]set master qos config failed, ret = %d\n", devId, ret);
return ret;
}
// 以手动模式开启带宽监控：需指定监控对象
struct qos_bw_config bwCfg = {0};
bwCfg.mode = 1;
bwCfg.state = 2;
bwCfg.target_set[0] = 1; // 举例,监控对象[0]配置为1
bwCfg.target_set[1] = 2; // 举例,监控对象[1]配置为2
bwCfg.cnt = 2;
bwCfg.interval = 1000;
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_QOS, DCMI_QOS_SUB_BW_DATA, \
static_cast<void*>(&(bwCfg)), sizeof(struct qos_bw_config));
if (ret != 0) {
printf("[dev:%d]set mbwu failed, ret = %d\n", devId, ret);
return ret;
}
// 终止带宽监控
bwCfg.state = 0;
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_QOS, DCMI_QOS_SUB_BW_DATA, \
static_cast<void*>(&(bwCfg)), sizeof(bwCfg));
if (ret != 0) {
printf("[dev:%d]set mbwu failed, ret = %d\n", devId, ret);
return ret;
}
// 以自动模式开启带宽监控：无需指定监控对象，程序自动读取配置过的监控对象
```



```
bwCfg.mode = 0;
bwCfg.interval = 1000;
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_QOS, DCMI_QOS_SUB_BW_DATA, \
static_cast<void *>(&(bwCfg)), sizeof(bwCfg));
if (ret != 0) {
printf("[dev:%d]set mbwu failed, ret = %d\n", devId, ret);
return ret;
}
// 终止带宽监控
bwCfg.state = 0;
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_QOS, DCMI_QOS_SUB_BW_DATA, \
static_cast<void *>(&(bwCfg)), sizeof(bwCfg));
if (ret != 0) {
printf("[dev:%d]set mbwu failed, ret = %d\n", devId, ret);
return ret;
}
// 配置全局开关：开启qos功能
struct qos_gbl_config gblCfg = {0};
gblCfg.enable = 1;
ret = dcmi_set_device_info(card_id, device_id, DCMI_MAIN_CMD_QOS,
static_cast<uint32_t>(DCMI_QOS_SUB_GLOBAL_CONFIG),
static_cast<void *>(&gblCfg), sizeof(struct qos_gbl_config));
if (ret != 0) {
QOS_LOG_ERROR("[dev:%d]set gbl qos config failed, ret = %d\n", devId, ret);
return ret;
}
...
```

2.60.3 DCMI_MAIN_CMD_EX_CERT 命令说明

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

用于配置TLS证书。

参数说明

参数名	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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_VDEV_MNG

参数名	输入/输出	类型	描述
sub_cmd	输入	unsigned int	typedef enum { DCMI_CERT_SUB_CMD_INIT_TLS_PUB_KEY = 0, // 初始化TLS证书（用于获取CSR） DCMI_CERT_SUB_CMD_INIT_RESERVE, // 初始化预留字段 DCMI_CERT_SUB_CMD_TLS_CERT_INFO, // 预置/更新/获取TLS证书信息 DCMI_CERT_SUB_CMD_MAX, } DCMI_EX_CERT_SUB_CMD; 只支持DCMI_CERT_SUB_CMD_TLS_CERT_INFO
buf	输入	const void *	详见本节约束说明。
buf_size	输入	unsigned int	buf数组的长度。

表 2-65 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_CERT_SUB_CMD_TLS_CERT_INFO	struct dcmi_certs_chain_data { unsigned int count; // 证书数量 unsigned int data_len[MAX_CERT_COUNT]; // 证书长度 unsigned char data[0]; // 证书内容 };	sizeof(struct dcmi_certs_chain_data) + cert_count * NPU_CERT_MAX_SIZE	无

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无

约束说明

表 2-66 Atlas 800T A2 训练服务器、Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件部署场景

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

表 2-67 Atlas 200T A2 Box16 异构子框部署场景

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_certs_chain_data certs_chain_data = {0};
DCMI_MAIN_CMD main_cmd = DCMI_MAIN_CMD_EX_CERT;
unsigned int sub_cmd = DCMI_CERT_SUB_CMD_INIT_TLS_PUB_KEY;
unsigned int size = sizeof(struct dcmi_certs_chain_data) + cert_count * NPU_CERT_MAX_SIZE;
ret = dcmi_set_device_info(card_id, device_id, main_cmd, sub_cmd, (void *)&certs_chain_data, size);
if (ret) {
    // todo
}
// todo
...
```

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	<p>指定查询项对应主命令字。</p> <p>模块cmd信息，执行用于获取对应模块的信息</p> <pre>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_PCIE = 55, DCMI_MAIN_CMD_SIO = 56, DCMI_MAIN_CMD_EX_COMPUTING = 0x8000, DCMI_MAIN_CMD_DEVICE_SHARE = 0x8001, DCMI_MAIN_CMD_EX_CERT = 0x8003, DCMI_MAIN_CMD_MAX };</pre> <p>仅支持</p> <p>DCMI_MAIN_CMD_DVPP //dvpp算子模块主命令字、DCMI_MAIN_CMD_LP //lp低功耗模块主命令字、DCMI_MAIN_CMD_TS //ts任务调度模块主命令字、DCMI_MAIN_CMD_QOS //qos模块主命</p>

参数名称	输入/输出	类型	描述
			令字、DCMI_MAIN_CMD_HCCS //hccs 模块主命令字、 DCMI_MAIN_CMD_EX_COMPUTING // 算力扩展模块主命令字、 DCMI_MAIN_CMD_EX_CERT //证书管理 模块主命令字
sub_cmd	输入	unsigned int	详细参见子章节中的功能说明。
buf	输入/输出	void *	用于输入指定获取信息，并接收设备信息的返回数据。
size	输入/输出	unsigned int *	buf数组的输入/输出长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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;
}
...
```

2.61.1 DCMI_MAIN_CMD_DVPP 命令说明

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dvpp相关状态，配置信息。

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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_DVPP

参数名称	输入/输出	类型	描述
sub_cmd	输入	unsigned int	sub_cmd获取对应模块下子属性信息。 /* DCMI sub command for DVPP module */ #define DCMI_SUB_CMD_DVPP_STATUS 0 // dvpp状态，buf为0表示状态正常，非0表示状态异常 #define DCMI_SUB_CMD_DVPP_VDEC_RATE 1 // vdec利用率，正常值范围0-100 #define DCMI_SUB_CMD_DVPP_VPC_RATE 2 // vpc利用率，正常值范围0-100 #define DCMI_SUB_CMD_DVPP_VENC_RATE 3 // venc利用率，正常值范围0-100 #define DCMI_SUB_CMD_DVPP_JPEGE_RATE 4 // jpege利用率，正常值范围0-100 #define DCMI_SUB_CMD_DVPP_JPEGD_RATE 5 // jpegd利用率，正常值范围0-100 目前不支持 DCMI_SUB_CMD_DVPP_VENC_RATE命令的查询。
buf	输入/输出	void *	详细见本节约束说明。
size	输入/输出	unsigned int *	buf数组的长度/返回结果实际数据长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

获取dvpp设备时sub_cmd，buf和size之间必须要满足以下关系，如果不满足会导致接口调用失败。

表 2-69 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size
DCMI_SUB_CMD_DVPP_STATUS	unsigned int	长度为：sizeof(unsigned int)
DCMI_SUB_CMD_DVPP_VDEC_RATE DCMI_SUB_CMD_DVPP_VPC_RATE DCMI_SUB_CMD_DVPP_VENC_RATE DCMI_SUB_CMD_DVPP_JPEGE_RATE DCMI_SUB_CMD_DVPP_JPEGD_RATE	unsigned int	长度为：sizeof(unsigned int)

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

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
int ratio = 0;
int sub_cmd = 0;
unsigned int ratio_size = sizeof(int);
ret = dcmi_get_device_info(card_id, dev_id, DCMI_MAIN_CMD_DVPP, sub_cmd, (void *)&ratio, &ratio_size);
if (ret != 0) {
    printf("DCMI_MAIN_CMD_DVPP error\n");
    printf("card_id : %d, dev_id : %d, sub_cmd:0x%x, ratio:(0x%x,%d) fail, value = %d\n", card_id, dev_id,
sub_cmd, ratio, ratio_size);
    return ret;
} else {
    //todo
}
```

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

2.61.2 DCMI_MAIN_CMD_LP 命令说明

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

获取系统中AICORE、HYBIRD、CPU和DDR的电压和电流的寄存器值等LP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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_LP

参数名称	输入/输出	类型	描述
sub_cmd	输入	unsigned int	<div>/* DCMI sub command for Low power */</div> <div>typedef enum { // 获取AICORE电压电流的寄存器值 DCMI_LP_SUB_CMD_AICORE_VOLTAGE_CURRENT = 0, // 获取HYBIRD电压电流的寄存器值 DCMI_LP_SUB_CMD_HYBIRD_VOLTAGE_CURRENT, // 获取CPU电压电流的寄存器值 DCMI_LP_SUB_CMD_TAISHAN_VOLTAGE_CURRENT, // 获取DDR电压电流的寄存器值 DCMI_LP_SUB_CMD_DDR_VOLTAGE_CURRENT, // 获取ACG调频计数值 DCMI_LP_SUB_CMD_ACG, // 获取低功耗总状态 DCMI_LP_SUB_CMD_STATUS, // 获取所有工作档位 DCMI_LP_SUB_CMD_TOPS_DETAILS, // 设置工作档位 DCMI_LP_SUB_CMD_SET_WORK_TOPS, // 获取当前工作档位 DCMI_LP_SUB_CMD_GET_WORK_TOPS, // 获取当前降频原因 DCMI_LP_SUB_CMD_AICORE_FREQREDUC_CAUSE, // 获取功耗信息 DCMI_LP_SUB_CMD_GET_POWER_INFO, // 设置IDLE模式开关 DCMI_LP_SUB_CMD_SET_IDLE_SWITCH, DCMI_LP_SUB_CMD_MAX, } DCMI_LP_SUB_CMD;</div> <div>支持 DCMI_LP_SUB_CMD_AICORE_FREQREDUC_CAUSE、 DCMI_LP_SUB_CMD_GET_POWER_INFO、DCMI_LP_SUB_CMD_STATUS命令。</div>
buf	输入/输出	void *	详细见本节约束说明。
size	输入/输出	unsigned int *	buf数组的长度/返回结果实际数据长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 通过本接口获取电压和电流信息为寄存器数值。
- AI Core降频原因每100ms更新一次，不能查询历史降频原因。

表 2-71 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_AICORE_FREQREDUC_CAUSE	Unsigned long long	sizeof(unsigned long long)	buf为8字节内存空间，每个bit对应一种降频原因
DCMI_LP_SUB_CMD_GET_POWER_INFO	DCMI_LP_POWER_INFO_STRU	sizeof(DCMI_LP_POWER_INFO_STRU)	#define DCMI_LP_POWER_RESERVED_LEN 32 typedef struct DCMI_lp_power_info { unsigned int soc Rated power; unsigned char reserved[DCMI_LP_POWER_RESERVED_LEN]; } DCMI_LP_POWER_INFO_STRU; 其中，soc Rated power表示soc额定功率，其余为预留扩展空间。 soc额定功率正确范围 [150000,600000]
DCMI_LP_SUB_CMD_STATUS	unsigned int	sizeof(unsigned int)	非空闲：0；空闲：1

子命令DCMI_LP_SUB_CMD_AICORE_FREQREDUC_CAUSE下，AI Core降频原因由一个64位的值表示，每个bit对应一种降频原因。当值为0时，说明AI Core以额定的频率

运行。当值为1时，说明是由于某种原因引起AI Core不能以额定频率运行。同时由于降频原因可能由多个因素引起，所以可能存在多个bit被同时置1的情况。

表 2-72 buf 值各 bit 对应的含义

名称	Bit位	说明
IDLE	0	AI Core处于空闲状态，通过将AI Core频率降低到空闲时的频率来降低功耗。空闲状态需要持续一段时间，频率才会切换。
THERMAL	2	昇腾AI处理器温度超过了允许的范围导致底层软件将AI Core的频率限制在一定的范围，从而来降低芯片的温度。
SW_EDP	3	在昇腾AI处理器上，AI Core模块的供电电流超过了允许的范围导致底层软件将AI Core的频率限制在一定的范围，保证AI Core模块供电稳定。
HW_EDP	4	在昇腾AI处理器上，AI Core模块的瞬态供电电流超过了允许的范围导致主板上的电流传感器触发AI Core模块快速降频。
POWER_BREAK	5	主板上的功率监测模块监测到供电功率超过了允许的最大上限，通知昇腾AI处理器将AI Core的频率快速降低，维持供电稳定。
SVFD	8	昇腾AI处理器上的AI Core供电监测模块监测到AI Core模块的供电不稳（有噪声），触发AI Core快速降频，维持供电稳定。

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
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
int sub_cmd=0;
unsigned int voltage_cruuent_buf = 0;
unsigned int buf_size = 8;
```

```
ret = dcmi_get_device_info(card_id, dev_id, DCMI_MAIN_CMD_LP, sub_cmd, &voltage_cruuent_buf,
&buf_size);
if (ret != 0) {
//todo
return ret;
} else {
// todo
return ret;
}
...
```

2.61.3 DCMI_MAIN_CMD_TS 命令说明

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

获取系统中TS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]。
main_cmd	输入	enum dcmi_main_c md	DCMI_MAIN_CMD_TS。

参数名称	输入/输出	类型	描述
sub_cmd	输入	unsigned int	<pre>typedef enum { // 获取 AICORE 的单核利用率，正常值范围 0-100 DCMI_TS_SUB_CMD_AICORE_UTILIZATION_RATE = 0, // 获取 VECTOR CORE 的单核利用率/获取 AICORE单核中Vector单元利用率，正常值范围0-100 DCMI_TS_SUB_CMD_VECTORCORE_UTILIZATION_RATE, // 获取FFTS或者FFTS+的类型，0表示FFTS，1表示FFTS+ DCMI_TS_SUB_CMD_FFTS_TYPE, // 设置硬件屏蔽AICORE ERR的掩码 DCMI_TS_SUB_CMD_SET_FAULT_MASK, // 获取硬件屏蔽AICORE ERR的掩码 DCMI_TS_SUB_CMD_GET_FAULT_MASK, DCMI_TS_SUB_CMD_MAX, } DCMI_TS_SUB_CMD;</pre> 不支持 DCMI_TS_SUB_CMD_SET_FAULT_MASK、 DCMI_TS_SUB_CMD_GET_FAULT_MASK 开启profiling时，查询单核利用率结果为0xEF。
buf	输入/输出	void *	详细见本节约束说明。
Size	输入/输出	unsigned int *	buf数组的长度/返回结果数据长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

查询Vector Core的单核利用率时，buf至少为50字节内存空间，查询AI Core的单核利用率时，buf至少为25字节空间。

表 2-74 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size
DCMI_TS_SUB_CMD_AICORE_UTILIZATION_RATE、 DCMI_TS_SUB_CMD_VECTOR_CORE_UTILIZATION_RATE	查询Vector Core的单核利用率时，buf至少为50字节内存空间，查询AI Core的单核利用率时，buf至少为25字节空间。 异常值： 0xEE：表示对应的core损坏； 0xEF：无效值； 出参时每个字节表示一个核的利用率，正常范围0-100	作为入参时表示buf 的大小； 作为出参时表示buf内填充的有效值的个数。
DCMI_TS_SUB_CMD_FFTS_TY PE	unsigned int	unsigned int

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
Y	Y	Y

调用示例

```
...
int ret;
int card_id = 0;
int dev_id = 0;
void *buf = NULL;
int buf_size = 10;
buf = calloc(buf_size, sizeof(char));
if (buf == NULL) {
    printf("calloc buf failed.\n");
    return -1;
}
ret = dcmi_get_device_info(card_id,dev_id,DCMI_MAIN_CMD_TS,
DCMI_TS_SUB_CMD_AICORE_UTILIZATION_RATE, buf, &buf_size);
if (ret != 0) {
    printf("dcmi_get_device_info failed, ret = %d.\n", ret);
    return -1;
}
return 0;
...
```


2.61.4 DCMI_MAIN_CMD_QOS 命令说明

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QOS相关配置，包含获取指定的mpamid对应的QOS配置、指定master对应的QOS配置、带宽的统计值。

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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_QOS
sub_cmd	输入	unsigned int	<pre>typedef enum { // 获取 指定MATA的配置信息 DCMI_QOS_SUB_MATA_CONFIG, // 获取 指定master的配置信息 DCMI_QOS_SUB_MASTER_CONFIG, // 获取 带宽的统计信息 DCMI_QOS_SUB_BW_DATA, // 获取 通用配置信息 DCMI_QOS_SUB_GLOBAL_CONFIG, // 配置 完成指令 DCMI_QOS_SUB_CONFIG_DONE, } DCMI_QOS_SUB_INFO;</pre> 当前仅支持DCMI_QOS_SUB_MATA_CONFIG、DCMI_QOS_SUB_MASTER_CONFIG、DCMI_QOS_SUB_BW_DATA、DCMI_QOS_SUB_GLOBAL_CONFIG。
buf	输入/输出	void *	详细见本节约束说明。
Size	输入/输出	unsigned int *	buf数组的长度/返回结果数据长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 受soc特性约束，需要调用DCMI_set_device_info后再调用相应get接口读取配置是否生效，否则读取值不可信。
- resctrl获取带宽功能与DCMI接口获取实时带宽功能冲突，如果已经使能其中一种，请勿并行使用另外一种。

表 2-76 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	buf_size	参数说明
DCMI_QOS_SUB_MATA_CONFIG	struct qos_mata_config { int mpamid; u32 bw_high; u32 bw_low; int hardlimit; int reserved[8]; };	sizeof(qos_mata_config)	mpamid：取值范围是[0,127]。 bw_high：上水线(GB/s)取值范围是[0,1638]。 bw_low：下水线(GB/s)，取值范围是[0, bw_high] hardlimit：1表示开启，0表示不开启 用户通过该接口读取水线，可能同先前用户配置值存在误差。误差值计算方式为：处理器最大带宽/MAX_REG_VALUE。其中，MAX_REG_VALUE取值为1024。

sub_cmd	buf对应的数据类型	buf_size	参数说明
DCMI_QOS_SUB_MASTER_CONFIG	<pre>struct qos_master_config { int master; int mpamid; int qos; int pmg; u64 bitmap[4]; int reserved[8]; };</pre>	sizeof(qos_master_config)	<p>master: master id, 支持配置的项为: vdec=1,vpc=2,jpge=3,jpgd=4,pcie=7,sdma=13 mpamid : 取值范围是[0,127] qos : 带宽调度优先级, 取值范围[0,7], 0作为hardlimit专用qos, 7为调度绿色通道qos pmg: mpamid分组, 取值范围是[0,3](当前不支持) bitmap: 因框架限制, 不支持</p>
DCMI_QOS_SUB_BW_DATA	<pre>struct qos_bw_result { int mpamid; u32 curr; u32 bw_max; u32 bw_min; u32 bw_mean; int reserved[8]; };</pre>	sizeof(qos_bw_result)	<p>mpamid: 获取带宽的目标mpamid。 取值范围是[0,127]。 curr: 最近时间点获取的带宽值(MB/s) bw_max: 采样时间段内最大值(MB/s) bw_min: 采样时间段内最小值(MB/s) bw_mean: 采样时间段内的平均值(MB/s)</p>

sub_cmd	buf对应的数据类型	buf_size	参数说明
DCMI_QOS_SUB_GLOBAL_CONFIG	struct qos_gbl_config { u32 enable; u32 autoqos_fuse_en; u32 mpamqos_fuse_mode; u32 mpam_subtype; int reserved[8]; };	sizeof(qos_gbl_config)	<ul style="list-style-type: none">• enable: 是否使能QOS功能<ul style="list-style-type: none">- 0表示不使能- 1表示使能• autoqos_fuse_en: qos的融合开关<ul style="list-style-type: none">- 0表示关闭qos融合- 1表示开始qos融合• mpamqos_fuse_mode: qos的融合模式, autoqos_fuse_en开启的条件下生效<ul style="list-style-type: none">- 0表示均值融合- 1表示取随路qos和mpamqos之间的最大值作为融合结果- 2表示使用随路qos替换mpamqos• mpam_subtype: 带宽统计的模式。<ul style="list-style-type: none">- 0表示统计读+写带宽- 1表示统计写带宽- 2表示统计读带宽

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	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
int sub_cmd=0;
int size = sizeof(struct dcmi_qos_mata_config);
struct dcmi_qos_mata_config mataCfg = {0};
mataCfg.mpamid = 127;
unsigned int subCmd = (unsigned int)(DCMI_QOS_SUB_CMD_MAKE(mataCfg.mpamid,
```

```
DCMI_QOS_SUB_MATA_CONFIG));
ret = dcmi_get_device_info(card_id, dev_id, DCMI_MAIN_CMD_QOS, subCmd, (void *)&mataCfg, &size);
if (ret != 0) {
    //todo
    return ret;
} else {
    // todo
    return ret;
}
...
```

2.61.5 DCMI_MAIN_CMD_HCCS 命令说明

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HCCS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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_HCCS
sub_cmd	输入	unsigned int	typedef enum { DCMI_HCCS_CMD_GET_STATUS = 0, DCMI_HCCS_CMD_GET_LANE_INFO = 1, DCMI_HCCS_CMD_GET_STATISTIC_INFO = 3, DCMI_HCCS_CMD_MAX, } DCMI_HCCS_SUB_CMD;
buf	输出	void *	详见本节约束说明。
size	输入/输出	unsigned int *	返回结果数据长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-78 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_HCCS_CMD_GET_STATUS	struct dcmi_hccs_status { unsigned int pcs_status; unsigned char reserve[DCMI_HCCS_STATUS_RESERVED_LEN]; };	sizeof(struct dcmi_hccs_status)	查询HCCS状态，当前支持查询pcs状态。 pcs_status表示HCCS pcs状态，其值含义如下： <ul style="list-style-type: none">0 表示状态正常。非0表示状态异常。其中：<ul style="list-style-type: none">0bit在状态异常下固定为1。1-7bit当前固定为0，预留扩展。8-15bit表示当前存在问题的PCS序号，当前序号的范围为0-7，此值表示从0开始第1个有问题的索引号。16-23bit表示PCS当前发送的lane模式。 0表示0lane。 1表示1lane。 2表示4lane。 3表示8lane。24-31bit表示PCS当前发送的lane。

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_HCCS_CMD_GET_LANE_INFO	struct dcmi_hccs_lane_info { unsigned int hccs_port_pcs_bitmap; unsigned int pcs_lane_bitmap[DCMI_HCCS_MAX_PCS_NUM]; unsigned int reserve[DCMI_HCCS_MAX_PCS_NUM]; };	sizeof(struct dcmi_hccs_lane_info)	<ul style="list-style-type: none">• hccs_port_pcs_bitmap: 当前用于hccs连接PCS的bit位，对应序号从低位开始计数，写0代表非用于hccs，写1代表用于hccs。• pcs_lane_bitmap: 当前用于hccs连接的lane信息数组，数组顺序与hccs_port_pcs_bitmap从低位开始写‘1’的bit位对应，其值含义如下：<ul style="list-style-type: none">- 0bit: 是否切换到hccs完成。 1表示完成。 0表示其余bit位的值均无效。- 1-8bit: 当前hccs连接用的lane序号，从低位开始，计数范围0-3。 值为1表示该lane为当前发送使用。 值为0表示未被使用。- 9-10bit: 当前hccs连接用的lane模式。 00为0lane。 01为1lane。 10为4lane。 11为2lane。

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_HCCS_CMD_GET_STATISTIC_INFO	struct dcmi_hccs_statistic_info { unsigned int tx_cnt[DCMI_HCCS_MAX_PCS_NUM]; unsigned int rx_cnt[DCMI_HCCS_MAX_PCS_NUM]; unsigned int crc_err_cnt[DCMI_HCCS_MAX_PCS_NUM]; unsigned int retry_cnt[DCMI_HCCS_MAX_PCS_NUM]; unsigned int reserved_field_cnt[DCMI_HCCS_RESERVED_FIELD_NUM]; // 预留64个字段供后续扩充使用 };	sizeof(struct dcmi_hccs_statistic_info)	获取HCCS收发报文和误码统计信息。 <ul style="list-style-type: none">当前芯片仅使用x4模式，共获取8个HDLC链路的统计信息（每device）。每个链路的统计信息包含3个u32整数，分别是发送报文、接收报文、接收报文crc错误的计数，单位是flit。tx_cnt：表示累积发包数量，范围：0~4,294,967,295。rx_cnt：表示累积收包数量，范围：0~4,294,967,295。crc_err_cnt：表示误码数量，范围：0~4,294,967,295。retry_cnt：表示数据包的重传次数，范围：0~4,294,967,295。<ul style="list-style-type: none">0：表示无重传其他：表示重传的次数

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
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
struct dcmi_hccs_statues status = {0};
unsigned int buf_size = sizeof(struct dcmi_hccs_statues);
ret = dcmi_get_device_info(card_id, dev_id, DCMI_MAIN_CMD_HCCS,
DCMI_HCCS_CMD_GET_STATUS, &status, &buf_size);
if (ret != 0) {
//todo
}
```



```
return ret;
} else {
// todo
return ret;
}
...
```

2.61.6 DCMI_MAIN_CMD_EX_COMPUTING 命令说明

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

获取算力token值。

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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_EX_COMPUTING
sub_cmd	输入	unsigned int	DCMI_EX_COMPUTING_SUB_CMD_TOKEN
buf	输出	void *	详细见本节约束说明。
size	输入/输出	unsigned int *	返回结果数据长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-80 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_EX_COMPUTING_SUB_CMD_TOKEN	struct dcmi_computing_token_stru { float value; u8 type; u8 reserve_c; u16 reserve_s; } a. value表示算力值。 b. type表示license BOM编码转换后的类型值。 c. reserve_c表示保留字，当前预留，不使用。 d. reserve_s表示保留字，当前预留，不使用。	sizeof(dcmi_computing_token_stru)	获取算力token值 value值的正确范围：[0,65535]

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 = 0;
int card_id = 0;int device_id = 0;
struct dcmi_computing_token_stru dcmi_computing_token = {0};
unsigned int buf_size = sizeof(struct dcmi_computing_token_stru);
ret = dcmi_get_device_info(card_id, device_id, DCMI_MAIN_CMD_EX_COMPUTING,
DCMI_EX_COMPUTING_SUB_CMD_TOKEN, &dcmi_computing_token, &buf_size);
if (ret != 0) {
//todo
return ret;
```

```
} else {  
    // todo  
    return ret;  
}  
...
```

2.61.7 DCMI_MAIN_CMD_EX_CERT 命令说明

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, const void *buf, unsigned int buf_size)
```

功能说明

用于获取TLS证书相关信息。

参数说明

参数名	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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_EX_CERT
sub_cmd	输入	unsigned int	typedef enum { DCMI_CERT_SUB_CMD_INIT_TLS_PUB_KEY = 0, // 初始化TLS证书（用于获取CSR） DCMI_CERT_SUB_CMD_INIT_RESERVE, // 初始化预留字段 DCMI_CERT_SUB_CMD_TLS_CERT_INFO, // 预置/更新/获取TLS证书信息 DCMI_CERT_SUB_CMD_MAX, } DCMI_EX_CERT_SUB_CMD;
buf	输入	const void *	详细见本节约束说明。
buf_size	输入	unsigned int	buf数组的长度。

表 2-82 sub_cmd 对应的 buf 格式

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_CERT_SUB_CMD_INIT_TLS_PUB_KEY	<pre>#define CERT_NAME_MAX_LEN 16 #define CERT_COMMON_NAME_LEN 64 struct dcmi_csr_info { char country_name[CERT_NAME_MAX_LEN]; char province_name[CERT_NAME_MAX_LEN]; char city_name[CERT_NAME_MAX_LEN]; char organization_name[CERT_NAME_MAX_LEN]; char department_name[CERT_NAME_MAX_LEN]; char reserve_name[CERT_COMMON_NAME_LEN]; int reserve; int csr_len; char csr_data[NPU_CERT_MAX_SIZE]; };</pre>	返回公钥或者证书信息长度	<ul style="list-style-type: none">country_name：国家名称province_name：州或省名称city_name：地区名称organization_name：组织名称department_name：组织内的部门名称reserve_name：保留名称reserve：保留csr_len：csr请求内容的长度csr_data：csr请求内容

sub_cmd	buf对应的数据类型	size	参数说明
DCMI_CERT_SUB_CMD_TLS_CERT_INFO	<pre>#define CERT_COMMON_NAME_LEN 64 #define CERT_NAME_MAX_LEN 16 #define TIME_LEN 32 typedef struct dcmi_cert_info { unsigned int alarm_stat; unsigned int reserve; char start_time[TIME_LEN]; char end_time[TIME_LEN]; char country_name[CERT_NAME_MAX_LEN]; char province_name[CERT_NAME_MAX_LEN]; char city_name[CERT_NAME_MAX_LEN]; char organization_name[CERT_NAME_MAX_LEN]; char department_name[CERT_NAME_MAX_LEN]; char reserve_name[CERT_COMMON_NAME_LEN]; char common_name[CERT_COMMON_NAME_LEN]; } CERT_INFO;</pre>	返回公钥或者证书信息长度	<ul style="list-style-type: none">alarm_stat: 证书告警状态, 0-正常, 1-证书即将过期, 2-证书已过期/失效 (只有在读取时呈现证书即将过期, 不主动呈现)reserve: 保留start_time: 证书生效的起始时间end_time: 证书生效的结束时间country_name: 国家名称province_name: 州或省名称city_name: 地区名称organization_name: 组织名称

sub_cmd	buf对应的数据类型	size	参数说明
			<ul style="list-style-type: none">depart ment_n ame：组织内的部门名称reserve _name ：保留commo n_name ：证书的通用名，这里填充的是 device 的DIE-ID

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	N	N

调用示例

```
...
int ret;
```

```
int card_id = 0;
int device_id = 0;
struct dcmi_cert_info cert_info = { 0 };
DCMI_MAIN_CMD main_cmd = DCMI_MAIN_CMD_EX_CERT;
unsigned int sub_cmd = DCMI_CERT_SUB_CMD_INIT_TLS_PUB_KEY;
unsigned int size = sizeof(struct dcmi_cert_info);
ret = dcmi_get_device_info(card_id, device_id, main_cmd, sub_cmd, (void *)&cert_info, size);
if (ret) {
    // todo
}
// todo
...
```

2.61.8 DCMI_MAIN_CMD_PCIE 命令说明

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PCIe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]。
main_cmd	输入	enum dcmi_main_cmd	DCMI_MAIN_CMD_PCIE
sub_cmd	输入	unsigned int	DCMI_PCIE_SUB_CMD_PCIE_ERROR_IN FO
buf	输出	void *	详细见本章 约束说明 。
size	输入/输出	Unsigned int *	Buf数组长度

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-84 sub_cmd 对应的 buf 格式

sub_cmd	Buf对应的数据类型	参数说明
DCMI_PCIE_SUB_CMD_PCIE_ERROR_INFO	struct dcmi_pcie_link_error_info { unsigned int tx_err_cnt; unsigned int rx_err_cnt; unsigned int lcrc_err_cnt; unsigned int ecrc_err_cnt; unsigned int retry_cnt; unsigned int rsv[32]; };	获取pcie link error相关值。 <ul style="list-style-type: none">tx_err_cnt表示PCIe发送错误计数rx_err_cnt表示PCIe接收错误计数lcrc_err_cnt表示PCIe DLLP LCRC校验错误计数ecrc_err_cnt表示PCIe TLP ECRC校验错误计数retry_cnt表示PCIe链路重传计数rsv保留字段

该接口在Linux物理机特权容器场景下支持。

host与device采用HCCS互联的设备另采用HCCS命令查询链路信息。

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
Y	Y	Y

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
struct dcmi_pcie_link_error_info pcie_link_error_info = {0};
unsigned int info_leng = sizeof(struct dcmi_pcie_link_error_info);
ret = dcmi_get_device_info(card_id, dev_id, DCMI_MAIN_CMD_PCIE,
DCMI_PCIE_SUB_CMD_PCIE_ERROR_INFO, &pcie_link_error_info, &info_leng);
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, // 用于吊销SOC密钥 DCMI_REVOCATION_TYPE_CMS_CRL = 1, //用于MDC CMS CRL文件升级 DCMI_REVOCATION_TYPE_CMS_CRL_EXT = 2, //用于扩展CRL文件升级 DCMI_REVOCATION_TYPE_MAX }; 当前仅支持 DCMI_REVOCATION_TYPE_SOC。
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，Soc二级密钥吊销操作的文件长度固定为544字节。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。
- 对于SMP系统，在执行吊销操作前必须先获取设备个数，然后对所有的设备均执行吊销操作。
- 该接口需调入TEEOS，耗时较长，不支持在接口调用时触发休眠唤醒，如果触发休眠，有较大可能造成休眠失败。

● 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
Y	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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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, // 网络健康状态正常 DCMI_RDFX_DETECT_SOCK_FAIL = 1, // 网络套接字创建失败 DCMI_RDFX_DETECT_RECV_TIMEOUT = 2, // 网口收包超时 DCMI_RDFX_DETECT_UNREACH = 3, // 侦测ip地址不可达 DCMI_RDFX_DETECT_TIME_EXCEEDED = 4, // 发送侦测报文执行超时 DCMI_RDFX_DETECT_FAULT = 5, // 发送侦测报文失败 DCMI_RDFX_DETECT_INIT = 6, // 侦测任务初始化中 DCMI_RDFX_DETECT_THREAD_ERR = 7, // 侦测任务创建失败 DCMI_RDFX_DETECT_IP_SET = 8, // 正在设置侦测ip地址 DCMI_RDFX_DETECT_MAX = 0xFF };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

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	Y

在调用dcmi_get_device_network_health接口前，需在Host侧以root用户执行如下命令设置IP地址：

- 配置RoCE网卡的IP地址、子网掩码
hccn_tool -i *devid* -ip -s *address* %s *netmask* %s
您需要根据实际情况，修改如下内容：
 - devid*：设置为设备ID。
 - address*后的 %s：设置为RoCE网卡的IP地址。
 - netmask*后的 %s：设置为子网掩码。
- 配置用于检测RoCE网卡的IP地址是否连通的IP地址
hccn_tool -i *devid* -netdetect -s *address* %s
您需要根据实际情况，修改如下内容：
 - devid*：设置为设备ID。
 - address*后的 %s：设置为用于检测RoCE网卡的IP地址是否连通的IP地址，例如路由器的IP地址。

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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。
elabel_info	输出	struct dcmi_elabel_info *	<pre>#define MAX_LENTH 256 struct dcmi_elabel_info { char product_name[MAX_LENTH]; //产品名称 char model[MAX_LENTH]; //产品型号 char manufacturer[MAX_LENTH]; //生产厂家 char manufacturer_date[MAX_LENTH]; //生产日期 char serial_number[MAX_LENTH]; //产品序列号 };</pre> 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

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

```
...
struct dcmi_elabel_info elabelInfo;
int ret = 0;
int card_id = 0;
memset(&elabelInfo, 0, sizeof(elabelInfo));
ret = dcmi_get_card_elabel_v2(card_id, &elabelInfo);
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]; //产品名称 char model[MAX_LENTH]; //产品型号 char manufacturer[MAX_LENTH]; //生产厂家 char serial_number[MAX_LENTH]; //产品序列号 }

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 2-93 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
N	N	N

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。
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。 温度传感器名称请参见相应产品的带外管理接口说明。 针对I2C协议，如果温度为无效数据则为0x7ffd，如果温度读取失败则为0x7fff 说明 OPTICAL温度传感器在未插入光模块或者接铜缆时，返回的温度为无效值0x7ffd。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
Y	Y	Y

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。 <ul style="list-style-type: none">0：禁用1：使能

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 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*	<div>board信息</div> <div>struct dcmi_board_info { unsigned int board_id; //单板id unsigned int pcb_id; //PCB版本编号 unsigned int bom_id; //BOM版本编号 unsigned int slot_id; //槽位号信息 };</div> <div>Atlas 200T A2 Box16 异构子框、Atlas 900 A2 PoD 集群基础单元和Atlas 800T A2 训练服务器、Atlas 800I A2 推理服务器、A200I A2 Box 异构组件的 pcb_id、bom_id、slot_id为无效值。</div>

返回值

类型	描述
int	<div>处理结果：</div> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

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
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。 <ul style="list-style-type: none">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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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]。 MCU芯片：mcu_id。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-102 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。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-103 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;
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-104 Atlas 200T A2 Box16 异构子框、Atlas 800T A2 训练服务器、Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件部署场景

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; //单板id unsigned int pcb_id; //PCB版本编号 unsigned int bom_id; //BOM版本编号 unsigned int slot_id; //槽位号信息 }; 在物理机场景下，slot_id为无效值。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-105 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_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。 针对I2C协议，如果功耗为无效数据则为0x7ffd，如果功耗读取失败则为0x7fff。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-106 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AI Core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]; }; type输入为NPU_TYPE即1时，data1代表AI Core核数。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-107 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

Linux物理机		Linux物理机容器
Y	Y	Y

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-108 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-109 Atlas 200T A2 Box16 异构子框、Atlas 800T A2 训练服务器、Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件部署场景

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	<p>可只订阅满足指定条件的事件，过滤条件如下：</p> <pre>#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：参考《健康管理故障定义》 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：参考《健康管理故障定义》 unsigned char resv[DCMI_MAX_EVENT_RESV_LENGTH]; //保留 };</pre>

参数名称	输入/输出	类型	描述
event	输出	struct dcmi_event *	<p>输出事件结构体定义如下：</p> <pre>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; };</pre> <p>type: 当前支持DCMI_DMS_FAULT_EVENT类型，枚举定义如下：</p> <pre>enum dcmi_event_type { DCMI_DMS_FAULT_EVENT = 0, DCMI_EVENT_TYPE_MAX };</pre> <p>dms_event: DCMI_DMS_FAULT_EVENT类型对应的事件内容定义如下：</p> <pre>#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</pre>

参数名称	输入/输出	类型	描述
			<div>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</div> <div>char event_name[DCMI_MAX_EVENT_NAME_LENGTH]; //事件描述信息</div> <div>char additional_info[DCMI_MAX_EVENT_DATA_LENGTH]; // 事件附加信息</div> <div>unsigned char resv[DCMI_MAX_EVENT_RESV_LENGTH]; //保留</div> <div>};</div>

返回值

类型	描述
int	<div>处理结果：</div> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 2-110 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 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SVM模块相关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; //进程id unsigned long proc_mem_usage; //内存占用量 };
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

在宿主机和特权容器场景下执行该命令时可查询到宿主机、普通容器、特权容器中运行的所有进程，在普通容器场景下执行该命令时仅能查询到普通容器中运行的所有进程。

表 2-111 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 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-112 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-113 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在物理机场景下支持，在其他容器场景下获取的结果不保证正确性。

表 2-114 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-115 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-116 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_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]。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-117 Atlas 800T A2 训练服务器、Atlas 800I A2 推理服务器、Atlas 900 A2 PoD 集群基础单元、A200I A2 Box 异构组件部署场景

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

Linux物理机		Linux物理机容器
Y	N	N

表 2-118 Atlas 200T A2 Box16 异构子框部署场景

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;
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;
}
...
```

2.94 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]。

参数名称	输入/输出	类型	描述
parm	输入	struct dcmi_sm_parm*	struct 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];//CBC算法初始化向量 unsigned char key[512];//密钥 unsigned char reserved[512];//预留 }; key_type入参范围： enum sm_key_type{ SM3_NORMAL_SUMMARY = 0,//SM3杂凑算法操作 SM4_CBC_ENCRYPT = 1, //SM4 CBC加密算法 SM4_CBC_DECRYPT = 2,[x(1) //SM4 CBC解密算法 };
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;//输出缓存 unsigned int *out_len;//输出缓存长度 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-119 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);
```

2.95 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]。
parm	输入	struct sm_parm*	struct 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];//CBC算法初始化向量 unsigned char key[512];//密钥 unsigned char reserved[512];//预留 }; key_type入参范围： enum sm_key_type{ SM3_NORMAL_SUMMARY = 0,//SM3杂凑算法操作 SM4_CBC_ENCRYPT = 1, //SM4 CBC加密算法 SM4_CBC_DECRYPT = 2,[x(1) //SM4 CBC解密算法 };
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;//输出缓存 unsigned int *out_len;//输出缓存长度 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-120 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.iv, 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 dcml_sm_data sm_test_data = {(const unsigned char *)data1, sizeof(data1), out_buf, out_len};
dcml_init();
ret = dcml_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);
```

2.96 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]。

参数名称	输入/输出	类型	描述
type	输入	struct dcmi_power_state_info_stru	设置系统状态参数。 typedef enum { DCMI_POWER_STATE_SUSPEND, //暂停 DCMI_POWER_STATE_POWEROFF, //下电 DCMI_POWER_STATE_RESET, //复位 DCMI_POWER_STATE_MAX, } DCMI_POWER_STATE; typedef enum { DCMI_POWER_RESUME_MODE_BUTTON, //电源恢复模式按钮 DCMI_POWER_RESUME_MODE_TIME, //系统按时间恢复 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;//休眠唤醒状态 DCMI_LP_RESUME_MODE mode;//休眠唤醒模式 unsigned int value; unsigned int reserve[DCMI_POWER_INFO_RESERVE_LEN];//保留 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 2-121 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_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
...
```

2.97 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： <ul style="list-style-type: none">0：AMP1：SMP

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-122 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;
}
...
```

2.98 dcmi_subscribe_fault_event 接口原型

函数原型

int dcmi_subscribe_fault_event(int card_id, int device_id, struct dcmi_event_filter filter, dcmi_fault_event_callback handler)

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_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]。
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：参考《健康管理故障定义》 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：参考《健康管理故障定义》 unsigned char resv[DCMI_MAX_EVENT_RESV_LENGTH]; //保留 };

参数名称	输入/输出	类型	描述
handler	输入	dcmi_fault_event_callback	<pre>typedef void (*dcmi_fault_event_callback) (struct dcmi_event *event);</pre> 输出事件结构体定义如下： <pre>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; };</pre> type: 当前支持DCMI_DMS_FAULT_EVENT类型，枚举定义如下： <pre>enum dcmi_event_type { DCMI_DMS_FAULT_EVENT = 0, DCMI_EVENT_TYPE_MAX };</pre> dms_event: DCMI_DMS_FAULT_EVENT类型对应的事件内容定义如下： <pre>#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</pre>

参数名称	输入/输出	类型	描述
			<div>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</div> <div>char event_name[DCMI_MAX_EVENT_NAME_LENGTH]; //事件描述信息</div> <div>char additional_info[DCMI_MAX_EVENT_DATA_LENGTH]; //事件附加信息</div> <div>unsigned char resv[DCMI_MAX_EVENT_RESV_LENGTH]; //保留</div> <div>};</div>

返回值

类型	描述
int	<div>处理结果：</div> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 当card_id和device_id的取值都为-1时，支持订阅当前环境中所有设备的故障事件。
- 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。同一个进程中，不能与dcmi_get_fault_event接口同时调用。
- 一个进程内只能调用一次，且物理机和特权容器共享64个进程资源，进程退出时才会释放该进程资源。
- 在SMP模式下，对于OS类的故障，只能从SMP下头节点的device_id订阅到该OS类故障。

表 2-123 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

```
void event_handler(struct dcmi_event *event){
    // todo
}
struct dcmi_event_filter filter;
filter.filter_flag = DCMI_EVENT_FILTER_FLAG_SERVERITY | DCMI_EVENT_FILTER_FLAG_NODE_TYPE;
filter.severity = 1; /* 只订阅2~3级别的事件 */
filter.node_type = 112;
struct dcmi_event event = {0};
int dev_id, ret;
.....
ret = dcmi_subscribe_fault_event(card_id_list[card_id], device_id, filter, event_handler);
if (ret != 0) {
} else {
}
```

2.99 dcmi_get_device_compatibility 接口原型

函数原型

int dcmi_get_device_compatibility (int card_id, int device_id, enum dcmi_device_compat *compatibility)

功能说明

查询NPU芯片驱动与固件兼容性。

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
compatibility	输出	enum dcmi_device_compat*	<div>enum dcmi_device_compat{ DCMI_COMPAT_OK = 1, DCMI_COMPAT_NOK = 2, DCMI_COMPAT_UNKNOWN = 3 };</div> <div>输出说明如下：</div> <ul style="list-style-type: none">DCMI_COMPAT_OK (1) 表示NPU驱动和NPU固件兼容。DCMI_COMPAT_NOK (2) 表示NPU驱动和NPU固件不兼容。DCMI_COMPAT_UNKNOWN (3) 表示NPU驱动和NPU固件配套关系未知。 <div>说明 当安装或升级固件run包异常时，或使用NPU 23.0.RC2之前版本的固件run包时，会出现DCMI_COMPAT_UNKNOWN的情况。</div>

返回值

类型	描述
int	<div>处理结果：</div> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

调用该接口前NPU固件和NPU驱动的安装升级必须生效。

表 2-124 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;
enum dcmi_device_compat compatibility = DCMI_COMPAT_UNKNOWN;
ret = dcmi_get_device_compatibility (card_id, device_id, & compatibility);
...
```

2.100 dcmi_get_all_device_count 接口原型

函数原型

```
int dcmi_get_all_device_count(int *all_device_count)
```

功能说明

host启动后，查询与host PCIe建链成功的昇腾AI处理器设备个数。

参数说明

参数名称	输入/输出	类型	描述
all_device_count	输出	Int *	查询到的设备个数。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

调用该接口前NPU固件和NPU驱动的安装升级必须生效。

表 2-125 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

Linux物理机		Linux物理机容器
Y	Y	N

调用示例

```
int ret = 0;
int all_device_count;
ret = dcmi_get_all_device_count(&all_device_count);
if(ret != 0) {
    //todo
    return ret;
}
...
```

2.101 dcmi_get_multi_ecc_time_info 接口原型

函数原型

int dcmi_get_multi_ecc_time_info(int card_id, struct dcmi_multi_ecc_time_data *multi_ecc_time_data)

功能说明

查询片上内存ECC多比特时间戳信息。

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。
multi_ecc_time_data	输出	struct dcmi_multi_ecc_time_data	多比特时间戳信息。 struct dcmi_multi_ecc_time_data { unsigned int multi_record_count;//不同地址的多bit错误个数 unsigned int multi_ecc_times[MAX_RECORD_ECC_ADDR_COUNT];//ECC地址出现错误次数 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-126 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 dev_id=0;
struct dcmi_multi_ecc_time_data multi_ecc_time_data = {0};
ret = dcmi_get_multi_ecc_time_info(dev_id, &multi_ecc_time_data);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.102 dcmi_get_multi_ecc_record_info 接口原型

函数原型

```
int dcmi_get_multi_ecc_record_info(int card_id, unsigned int *ecc_count,
unsigned char read_type, unsigned char module_type, struct
dcmi_ecc_common_data *ecc_common_data_s)
```

功能说明

查询片上内存ECC详细地址信息。

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。
ecc_count	输出	unsigned int	片上内存ECC错误个数。
read_type	输入	unsigned char	enum ECC_INFO_READ { MULTI_ECC_TIMES_READ = 0, //获取多bit错误次数记录 SINGLE_ECC_INFO_READ, //获取单bit错误信息 MULTI_ECC_INFO_READ, //获取多bit错误信息 ECC_ADDRESS_COUNT_READ, //ecc错误地址计数 ECC_MAX_READ_CMD }; 仅支持SINGLE_ECC_INFO_READ和MULTI_ECC_INFO_READ。
module_type	输入	unsigned char	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 }; 仅支持DCMI_DEVICE_TYPE_HBM //片上内存类型内存。
ecc_common_data_s	输出	struct dcmi_ecc_common_data	struct dcmi_ecc_common_data { unsigned long long physical_addr; //物理地址 unsigned int stack_pc_id; //hbmc_id unsigned int reg_addr_h; //寄存器的行和列信息 unsigned int reg_addr_l; //sid bank信息 unsigned int ecc_count; //ecc数量 int timestamp; //ecc发生时间 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-127 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 dev_id=0;
unsigned int ecc_count = 0;
unsigned char read_type = SINGLE_ECC_INFO_READ;
unsigned char module_type = DCMI_DEVICE_TYPE_HBM;
struct dcmi_ecc_common_data ecc_common_data_s[MAX_RECORD_ECC_ADDR_COUNT] = {0};
ret = dcmi_get_multi_ecc_record_info(dev_id, &ecc_count, read_type, module_type, ecc_common_data_s);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.103 dcmi_get_pcie_link_bandwidth_info 接口原型

函数原型

```
int dcmi_get_pcie_link_bandwidth_info (int card_id, int device_id, struct
dcmi_pcie_link_bandwidth_info *pcie_link_bandwidth_info)
```

功能说明

查询指定采样时间内，NPU设备与host OS间PCIe带宽信息。

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。
pcie_link_bandwidth_info	输入/输出	dcmi_pcie_link_bandwidth_info	<pre>struct dcmi_pcie_link_bandwidth_info { int profiling_time; //输入，控制采样时长 unsigned int tx_p_bw[AGENTDRV_PROF_DATA_NUM]; //输出，device设备向远端写带宽 unsigned int tx_np_bw[AGENTDRV_PROF_DATA_NUM]; //输出，device设备从远端读带宽 unsigned int tx_cpl_bw[AGENTDRV_PROF_DATA_NUM]; //输出，device设备回复远端读操作CPL的带宽 unsigned int tx_np_latency[AGENTDRV_PROF_DATA_NUM]; //输出，device设备从远端读的延时（ns） unsigned int rx_p_bw[AGENTDRV_PROF_DATA_NUM]; //输出，device设备接收远端写的带宽 unsigned int rx_np_bw[AGENTDRV_PROF_DATA_NUM]; //输出，device设备接收远端读的带宽 unsigned int rx_cpl_bw[AGENTDRV_PROF_DATA_NUM]; //输出，device设备从远端读收到CPL回复的带宽 }; profiling_time取值范围： 0ms~2000ms。 带宽单位均为MB/ms。</pre>

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 该接口在开启profiling场景下数据无意义。
- 该命令在Linux物理机特权容器场景下支持使用。

表 2-128 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 dcmi_pcie_link_bandwidth_info bandwidth_info = {0};
bandwidth_info.profiling_time = 1000;
ret = dcmi_get_pcie_link_bandwidth_info(card_id, device_id, &bandwidth_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.104 dcmi_get_topo_info_by_device_id 接口原型

函数原型

int dcmi_get_topo_info_by_device_id (int card_id1, int device_id1, int card_id2, int device_id2, int * topo_type)

功能说明

查询指定两个Device之间的拓扑关系。

参数说明

参数名称	输入/输出	类型	描述
card_id1/ card_id2	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_id1/ device_id2	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。
topo_type	输出	int *	输出Device之间拓扑关系的数值，对应关系如下： typedef enum { DCMI_TOPO_TYPE_SELF = 0,//芯片本身 DCMI_TOPO_TYPE_SYS,//通过PCIe连接且穿过NUMA nodes，nodes之间通过SMP连接，如：QPI、UPI DCMI_TOPO_TYPE_PHB,//通过PCIe连接且穿过同一个CPU的PCIe host bridge DCMI_TOPO_TYPE_HCCS,//通过HCCS链接 DCMI_TOPO_TYPE_PXB,//通过PCIe连接穿过多个PCIe switch DCMI_TOPO_TYPE_PIX,//通过PCIe连接穿过一个PCIe switch DCMI_TOPO_TYPE_BUTT = 6,//未知关系 DCMI_TOPO_TYOE_MAX, };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-129 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_id1 = 0;
int card_id2 = 1;
int device_id1 = 0;
int device_id2 = 1;
int result;
ret = dcmi_get_topo_info_by_device_id(card_id1, device_id1, card_id2, device_id2, &result);
...
```

2.105 dcmi_get_affinity_cpu_info_by_device_id 接口原型

函数原型

```
int dcmi_get_affinity_cpu_info_by_device_id(int card_id, int device_id, char
*affinity_cpu, int *length)
```

功能说明

查询指定NPU的CPU亲和性。

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。
affinity_cpu	输出	char *	获取指定NPU的CPU亲和性。
length	输出	int *	输出亲和性字符串的长度。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-130 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;
char affinity_cpu[TOPO_INFO_MAX_LENTH] ={0};
int length;
ret = dcmi_get_affinity_cpu_info_by_device_id(card_id, device_id, affinity_cpu, &length);
...
```

2.106 dcmi_get_netdev_pkt_stats_info 接口原型

函数原型

```
int dcmi_get_netdev_pkt_stats_info(int card_id, int device_id, int port_id,
struct dcmi_network_pkt_stats_info *network_pkt_stats_info)
```

功能说明

查询NPU设备网口当前收发包数统计。

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。
port_id	输入	int	NPU设备的网口端口号，当前仅支持配置0。

参数名称	输入/输出	类型	描述
network_pkt_stats_info	输出	dcmi_network_pkt_stats_info	struct dcmi_network_pkt_stats_info { unsigned long long mac_tx_mac_pause_num; MAC发送的pause帧总报文数 unsigned long long mac_rx_mac_pause_num; MAC接收的pause帧总报文数 unsigned long long mac_tx_pfc_pkt_num; MAC发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri0_pkt_num; MAC 0号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri1_pkt_num; MAC 1号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri2_pkt_num; MAC 2号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri3_pkt_num; MAC 3号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri4_pkt_num; MAC 4号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri5_pkt_num; MAC 5号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri6_pkt_num; MAC 6号调度队列发送的PFC帧总报文数 unsigned long long mac_tx_pfc_pri7_pkt_num; MAC 7号调度队列发送的PFC帧总报文数 unsigned long long mac_rx_pfc_pkt_num; MAC接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri0_pkt_num; MAC 0号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri1_pkt_num; }

参数名称	输入/输出	类型	描述
			MAC 1号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri2_pkt_num; MAC 2号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri3_pkt_num; MAC 3号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri4_pkt_num; MAC 4号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri5_pkt_num; MAC 5号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri6_pkt_num; MAC 6号调度队列接收的PFC帧总报文数 unsigned long long mac_rx_pfc_pri7_pkt_num; MAC 7号调度队列接收的PFC帧总报文数 unsigned long long mac_tx_total_pkt_num; MAC发送的总报文数 unsigned long long mac_tx_total_oct_num; MAC发送的总报文字节数 unsigned long long mac_tx_bad_pkt_num; MAC发送的坏包总报文数 unsigned long long mac_tx_bad_oct_num; MAC发送的坏包总报文字节数 unsigned long long mac_rx_total_pkt_num; MAC接收的总报文数 unsigned long long mac_rx_total_oct_num; MAC接收的总报文字节数 unsigned long long mac_rx_bad_pkt_num; MAC接收的坏包总报文数 unsigned long long mac_rx_bad_oct_num;

参数名称	输入/输出	类型	描述
			<div>MAC接收的坏包总报文字节数 unsigned long long mac_rx_fcs_err_pkt_num; MAC接收的存在FCS错误的报文数 unsigned long long roce_rx_rc_pkt_num; RoCEE接收的RC类型报文数 unsigned long long roce_rx_all_pkt_num; RoCEE接收的总报文数 unsigned long long roce_rx_err_pkt_num; RoCEE接收的坏包总报文数 unsigned long long roce_tx_rc_pkt_num; RoCEE发送的RC类型报文数 unsigned long long roce_tx_all_pkt_num; RoCEE发送的总报文数 unsigned long long roce_tx_err_pkt_num; RoCEE发送的坏包总报文数 unsigned long long roce_cqe_num; RoCEE任务完成的总元素个数 unsigned long long roce_rx_cnp_pkt_num; RoCEE接收的CNP类型报文数 unsigned long long roce_tx_cnp_pkt_num; RoCEE发送的CNP类型报文数 unsigned long long roce_err_ack_num; RoCEE接收的非预期ACK报文数，NPU做 丢弃处理，不影响业务 unsigned long long roce_err_psn_num; RoCEE接收的PSN>预期PSN的报文，或 重复PSN报文数。乱序或丢包，会触发重 传 unsigned long long roce_verification_err_num; RoCEE接收的域段校验错误的报文数， 如：icrc、报文长度、目的端口号等校验 失败</div>

参数名称	输入/输出	类型	描述
			<div>unsigned long long roce_err_qp_status_num; RoCEE接收的QP连接状态异常产生的报文数</div> <div>unsigned long long roce_new_pkt_rty_num; RoCEE发送的超次重传的数量统计</div> <div>unsigned long long roce_ecn_db_num; RoCEE接收的存在ECN标记位的报文数</div> <div>unsigned long long nic_tx_all_pkg_num; NIC发送的总报文数</div> <div>unsigned long long nic_tx_all_oct_num; NIC发送的总报文字节数</div> <div>unsigned long long nic_rx_all_pkg_num; NIC接收的总报文数</div> <div>unsigned long long nic_rx_all_oct_num; NIC接收的总报文字节数</div> <div>long tv_sec; 查询发生时的当前系统时间（单位s）</div> <div>long tv_usec; 查询发生时的当前系统时间（单位us）</div> <div>unsigned char reserved[64]; };</div>

返回值

类型	描述
int	<div>处理结果：</div> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-131 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	NA

调用示例

```
...
int ret = 0;
int card_id=0;
int device_id=0;
int port_id=0;
struct dcmi_network_pkt_stats_info network_pkt_stats_info = {0};
ret = dcmi_get_netdev_pkt_stats_info (card_id, device_id, port_id, &network_pkt_stats_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.107 dcmi_get_rdma_bandwidth_info 接口原型

函数原型

```
int dcmi_get_rdma_bandwidth_info(int card_id, int device_id, int port_id,
unsigned int prof_time, struct dcmi_network_rdma_bandwidth_info
*network_rdma_bandwidth_info)
```

功能说明

查询指定采样时间内，NPU设备网口的rdma带宽信息。

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。
port_id	输入	int	NPU设备的网口端口号，当前仅支持配置0。
prof_time	输入	unsigned int	采样时间，取值范围：100ms~10000ms。

参数名称	输入/输出	类型	描述
network_rdma_bandwidth_info	输出	dcmi_network_rdma_bandwidth_info	struct dcmi_network_rdma_bandwidth_info { unsigned int tx_bandwidth;接收方向带宽 unsigned int rx_bandwidth;发送方向带宽 }; 单位为MB/s。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-132 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	NA

调用示例

```
...
int ret = 0;
int card_id=0;
int device_id=0;
int port_id=0;
unsigned int prof_time = 1000;
struct dcmi_network_rdma_bandwidth_info bandwidth_info = {0};
ret = dcmi_get_rdma_bandwidth_info (card_id, device_id, port_id, prof_time, &bandwidth_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.108 dcmi_get_device_hbm_product_info 接口原型

函数原型

```
int dcmi_get_device_hbm_product_info(int card_id, int device_id, struct  
dcmi_hbm_product_info *hbm_product_info)
```

功能说明

查询指定NPU的片上内存厂商信息。

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。
hbm_product_info	输出	struct dcmi_hbm_product_info	struct dcmi_hbm_product_info { unsigned short manufacturer_id; // 片上内存厂商类型 unsigned char reserve[62]; };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-133 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_hbm_product_info hbm_product_info = {0};
ret = dcmi_get_device_hbm_product_info (card_id, device_id, &hbm_product_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.109 dcmi_get_serdes_quality_info 接口原型

函数原型

int dcmi_get_serdes_quality_info (int card_id, int device_id, unsigned int macro_id, struct dcmi_serdes_quality_info *serdes_quality_info)

功能说明

查询指定NPU的某个macro端口的眼图质量。

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。
macro_id	输入	unsigned int	需查询的macro端口号。 <ul style="list-style-type: none">0-7为h60 macro端口9-12为h32 macro端口

参数名称	输入/输出	类型	描述
serdes_quality_info	输出	struct dcmi_serdes_quality_info *	<p>查询眼图质量信息结构体</p> <ul style="list-style-type: none">当端口类型为h60时，base_info中snr与heh数据有效，其他数据为0（无效）。当端口类型为h32时，base_info中snr与heh数据为0（无效），其他数据有效。当端口断开时，对应端口的lane信息全为0。 <p>相关结构体定义如下</p> <pre>#define SERDES_RESERVED_LEN 64 #define SERDES_INFO_NUM 8 struct dcmi_serdes_quality_base { unsigned int snr; //信噪比 unsigned int heh; //半眼高 signed int bottom; //眼图的底部 signed int top; //眼图的顶部 signed int left; //眼图的左侧 signed int right; //眼图的右侧 }; typedef struct dcmi_serdes_quality_info { unsigned int macro_id; // 端口id unsigned int reserved1; //保留 struct dcmi_serdes_quality_base serdes_quality_info[SERDES_INFO_NUM]; // 端口每条lane眼图信息 unsigned int reserved2[SERDES_RESERVED_LEN]; // 预留字节 } DCMI_SERDES_QUALITY_INFO;</pre>

返回值

类型	描述
int	<p>处理结果：</p> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-134 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 macro_id = 0;
struct dcmi_serdes_quality_info serdes_quality_info = {0};
ret = dcmi_get_serdes_quality_info (card_id, device_id, macro_id, &serdes_quality_info);
...
```

2.110 dcmi_get_mainboard_id 接口原型

函数原型

int dcmi_get_mainboard_id(int card_id, int device_id, unsigned int *mainboard_id)

功能说明

查询指定NPU的mainboard_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	输入	int	指定设备编号，通过dcmi_get_device_id_in_card接口获取。
mainboard_id	输出	unsigned int *	获取指定NPU的mainboard_id。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-135 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 mainboard_id;
int card_id = 0;
int device_id = 0;
ret = dcmi_get_mainboard_id (card_id, device_id, &mainboard_id);
...
```

2.111 dcmi_prbs_operate 接口原型

函数原型

```
int dcmi_prbs_operate(int card_id, int device_id, struct
dcmi_prbs_operate_param operate_para, struct dcmi_prbs_operate_result
*operate_result)
```

功能说明

对昇腾NPU芯片打流和获取打流结果。

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
operate_param	输入	DCMI_PRBS_OPERATE_PARAM	<p>prbs码流操作入参，通过设置以下结构体中的参数区分打流和查询打流结果。</p> <pre>typedef struct dcmi_serdes_prbs_param_base { unsigned int serdes_prbs_macro_id; //仅支持macro_0打流 } DCMI_SERDES_PRBS_PARAM_BASE;</pre> <p>说明 macro_id取值范围为0~7，9~12，说明如下：</p> <ul style="list-style-type: none">• macro_0用于ROCE协议；• macro_1~macro_7用于HCCS协议；• macro_9~macro_12用于PCIe协议。 <pre>unsigned int serdes_prbs_start_lane_id; // lane的开始id，最小值为0，最大值为3 unsigned int serdes_prbs_lane_count; // lane的数量，最小值1，最大值为4 typedef DCMI_SERDES_PRBS_PARAM_BASE DCMI_SERDES_PRBS_GET_PARAM; // 获取打流结果时仅需传入基础信息param base，具体获取类型通过sub_cmd区分 typedef struct dcmi_serdes_prbs_set_param { DCMI_SERDES_PRBS_PARAM_BASE param_base; //打流设置基础参数 unsigned int serdes_prbs_type; //打流的码型，参考prbs码型枚举 unsigned int serdes_prbs_direction; //打流的方向，参考prbs打流方向枚举 } DCMI_SERDES_PRBS_SET_PARAM;</pre> <pre>typedef struct dcmi_prbs_operate_param { unsigned int main_cmd; //打流的命令，参考主命令字枚举 unsigned int sub_cmd; // 标识是设置打流命令还是查询打流结果命令 union { DCMI_SERDES_PRBS_SET_PARAM set_param; //设置打流参数 DCMI_SERDES_PRBS_GET_PARAM get_param; //查询打流结果 } operate_param; } DCMI_PRBS_OPERATE_PARAM;</pre> <p>// 主命令字枚举</p>

参数名称	输入/输出	类型	描述
			<div>enum dcmi_prbs_main_cmd_list { DSMI_SERDES_CMD_PRBS = 0,//打流的命令 DSMI_SERDES_CMD_MAX }; // 子命令字枚举 enum dcmi_prbs_sub_cmd_list { SERDES_PRBS_SET_CMD = 0,//设置打流的子命令 SERDES_PRBS_GET_RESULT_CMD, // 查结果 SERDES_PRBS_GET_STATUS_CMD, // 查码型 SERDES_PRBS_SUB_CMD_MAX }; // prbs码型枚举 enum dcmi_serdes_prbs_type_list { SERDES_PRBS_TYPE_END = 0, SERDES_PRBS_TYPE_7, SERDES_PRBS_TYPE_9, SERDES_PRBS_TYPE_10, SERDES_PRBS_TYPE_11, SERDES_PRBS_TYPE_15, SERDES_PRBS_TYPE_20, SERDES_PRBS_TYPE_23, SERDES_PRBS_TYPE_31, SERDES_PRBS_TYPE_58, SERDES_PRBS_TYPE_63, SERDES_PRBS_TYPE_MAX }; 说明<ul style="list-style-type: none">当前不支持58、63码型。如果使用当前prbs码型打流时多次出现满误码情况，可能光模块本身不支持，建议更换其他prbs码型进行尝试。通常建议使用prbs31码型进行打流。 // prbs打流方向枚举 enum dcmi_serdes_prbs_direction { SERDES_PRBS_DIRECTION_TX = 0, SERDES_PRBS_DIRECTION_RX,</div>

参数名称	输入/输出	类型	描述
			SERDES_PRBS_DIRECTION_TXRX, SERDES_PRBS_DIRECTION_MAX };
operate_result	输出	DCMI_PRBS_OPERATE_RESULT	查询打流结果或查询prbs链路状态。 #define MAX_LANE_NUM 8 typedef struct dcmi_prbs_operate_result { union { SERDES_PRBS_STATUS_S result[MAX_LANE_NUM];//打流返回结果 DCMI_SERDES_PRBS_LANE_STATUS lane_status[MAX_LANE_NUM];//lane的状态 } prbs_result; } DCMI_PRBS_OPERATE_RESULT; struct dcmi_serdes_prbs_lane_status { unsigned int lane_prbs_tx_status;//tx方向打流码型 unsigned int lane_prbs_rx_status;//rx方向打流码型 }; typedef struct { unsigned int check_en;//查询打流是否打开 unsigned int check_type;//查询打流码型 unsigned int error_status;//查询打流状态 unsigned int error_cnt;//错误的码型数量统计 unsigned long error_rate; //打流错误率的倒数，1/error_rate的值小于10 ⁻⁵ 为正常 unsigned int alos_status; //输入的信号幅度，0表示正常，1表示过低 unsigned long time_val;//设置打流和查寻打流结果之间的时间间隔 } SERDES_PRBS_STATUS_S;

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 打流前请关闭所有运行的业务。
- Linux物理机容器场景仅支持特权容器场景。
- driver包安装或NPU复位之后需要等待120s之后才能做打流动作。
- 打流动作会使NPU网口down，会产生BMC告警；若后续不关闭打流动作，则网口无法up。
- 打流功能支持光模块环回（光模块需要有环回能力）、光模块外接自环头环回、Cdr环回场景。

表 2-136 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;
DCMI_PRBS_OPERATE_PARAM operate_para = {0};
operate_para.main_cmd = DSMI_SERDES_CMD_PRBS;
operate_para.sub_cmd = SERDES_PRBS_SET_CMD;
operate_para.operate_para.set_param.param_base = {0, 0, 1};
operate_para.operate_para.set_param.serdes_prbs_type = SERDES_PRBS_TYPE_7;
operate_para.operate_para.set_param.serdes_prbs_direction = SERDES_PRBS_DIRECTION_TXRX;
DCMI_PRBS_OPERATE_RESULT operate_result;
ret = dcmi_prbs_operate(card_id, device_id, operate_para, &operate_result);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.112 dcmi_set_traceroute 接口原型

函数原型

`int dcmi_set_traceroute (int card_id, int device_id, struct traceroute param_info, struct node_info *ret_info[], unsigned int ret_info_size)`

功能说明

配置traceroute参数探测报文途径的网络节点信息。

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
param_info	输入	struct traceroute	<p>traceroute接口的入参</p> <pre>struct tracerout_result { int max_ttl; //探测报文的最大跳数。取值范围：-1，1~255。 设置为-1时，表示默认值为30 int tos; //IPv4设置TOS优先级，取值范围：-1~63。IPv6设置流量控制值，取值范围：-1~255。数值越大优先级越高，设置为-1时，表示默认值为0 int waittime; //设置等待探测响应的最大时间。取值范围：-1，1~60，单位：s，设置为-1时，表示默认值为3s int sport; //设置源端口号，取值范围：-1~65535。设置为-1时，表示默认值为大于30000的随机值 int dport; //设置目的端口号，取值范围：-1~65535。设置为-1时，表示默认值为大于30000的随机值 char dest_ip[48]; //目标主机IP bool ipv6_flag; //是否使用ipv6协议。0表示不使用，1表示使用 bool reset_flag; //终止device侧traceroute所有后台进程，当traceroute测试异常时配置为1结束device侧进程。 };</pre> <p>说明 端口号设置为0时，系统会使用大于30000的随机值。</p>

参数名称	输入/输出	类型	描述
ret_info[]	输出	struct node_info *	tracertime接口的返回信息 struct tracertime_result { int mask; //掩码，表示后面的数据是否有效，例如：0xFF代表mask后的8个字节是有效的 char ip[48]; //路由节点的ip地址 int sent; //发送ICMP请求报文的数量 double loss; //对应节点的丢包率 double last; //最新报文的响应时间，单位ms double avg; //所有报文的平均响应时间，单位ms double best; //报文最快响应时间，单位ms double worst; //报文最慢响应时间，单位ms double stdev; //标准偏差值，越大说明相应节点越不稳定 char reserve[64]; //附加扩展信息 }; 说明 返回信息的使用结构体数组存储，每个节点信息使用一个结构体。
ret_info_size	输入	unsigned int	传入的节点信息结构体的大小。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

- 为防止由于icmp报文被丢弃出现tracertime指令卡死的情况，建议IPv4的两条指令间隔3s执行。
- 若强制中断host侧指令或执行指令时返回值显示为-8020，则需配置param_info.reset_flag=1，调用dcmi_set_tracertime接口结束device侧进程。
- tracertime指令结束后，建议配置param_info.reset_flag=1，调用dcmi_set_tracertime接口结束device侧进程，防止进程持续运行影响NPU性能。

- 若param_info.reset_flag=1，则其他参数不生效。

约束说明

表 2-137 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;
struct traceroute param_info = {0};
struct node_info ret_info[10] = {0};
size_t ret_info_size = sizeof(ret_info);
param_info.sport = 40000;
param_info.waittime = 5;
strncpy_s(param_info.dest_ip, sizeof(param_info.dest_ip), "x.x.x.x", strlen("x.x.x.x"));

ret = dcmi_set_traceroute_get_info(card_id, device_id, param_info, &ret_info, ret_info_size);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

 **说明**

x.x.x.x表示目的IP地址。

2.113 dcmi_get_pfc_duration_info 接口原型

函数原型

```
int dcmi_get_pfc_duration_info(int card_id, int device_id, struct
dcmi_pfc_duration_info *pfc_duration_info)
```

功能说明

获取指定设备的PFC反压帧持续时间统计值。

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]。
pfc_duration_info	输出	struct dcmi_pfc_duration_info*	PFC pause duration值，结构体内容为： unsigned long long tx[PRI_NUM];//发送的反压帧持续时间 unsigned long long rx[PRI_NUM];//接收的反压帧持续时间 PRI_NUM=8;//8条优先级队列

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在Linux物理机特权容器场景下支持使用。

表 2-138 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_pfc_duration_info pfc_duration_info = {0};
ret = dcmi_get_pfc_duration_info(card_id, device_id, &pfc_duration_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.114 dcmi_clear_pfc_duration 接口原型

函数原型

```
int dcmi_clear_pfc_duration(int card_id, int device_id)
```

功能说明

清除指定设备的PFC反压帧持续时间统计值。

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在Linux物理机特权容器场景下支持使用。

表 2-139 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_clear_pfc_duration(card_id, device_id);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.115 dcmi_get_netdev_tc_stat_info 接口原型

函数原型

dcmi_get_netdev_tc_stat_info(int card_id, int device_id, struct dcmi_network_tc_stat_info *network_tc_stat_info)

功能说明

获取每个NPU 8个TC的累计流量。

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_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
network_tc_stat_info	输出	dcmi_network_tc_stat_info	struct dcmi_tc_stat_data { unsigned long long tx_packet[TC_QUE_NUM];//获取发送的数据流量 unsigned long long rx_packet[TC_QUE_NUM];//获取接收的数据流量 unsigned long long reserved[TC_QUE_NUM];//预留 }

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在Linux物理机特权容器场景下支持使用。

表 2-140 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_tc_stat_data network_tc_stat_info = {0};
ret = dcmi_get_netdev_tc_stat_info (card_id, device_id, &network_tc_stat_info);
if (ret != 0){
    //todo: 记录日志
}
```

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

2.116 dcmi_set_device_clear_tc_pkt_stats 接口原型

函数原型

```
dcmi_set_device_clear_tc_pkt_stats(int card_id, int device_id)
```

功能说明

清除每个NPU 8个TC的累计流量。

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_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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在Linux物理机特权容器场景下支持使用。

表 2-141 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

Linux物理机		Linux物理机容器
Y	N	N

调用示例

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

2.117 dcmi_get_qpn_list 接口原型

函数原型

int dcmi_get_qpn_list (int card_id, int device_id, int port_id, struct dcmi_qpn_list *list)

功能说明

获取qp的数量和对应的qpn_list。

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]。
port_id	输入	int	NPU设备的网口端口号，当前仅支持配置0。

参数名称	输入/输出	类型	描述
list	输出	struct dcmi_qpn_list*	#define MAX_QPN_NUM 8192//qpn列表最大支持的条数 struct dcmi_qpn_list { unsigned int qp_num;//获取到 active状态的qp的数量 unsigned int qpn_list[MAX_QPN_NUM];//与 qp_num对应的qpn列表 }; qpn: 表示qp number。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在Linux物理机特权容器场景下支持使用。

表 2-142 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 port_id = 0;
struct dcmi_qpn_list list = {0};
ret = dcmi_get_qpn_list(card_id, device_id, port_id, &list);
```

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

2.118 dcmi_get_qp_info 接口原型

函数原型

```
int dcmi_get_qp_info (int card_id, int device_id, int port_id, unsigned int qpn,
struct dcmi_qp_info *qp_info)
```

功能说明

根据qpn获取qp context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]。
port_id	输入	int	NPU设备的网口端口号，当前仅支持配置0。
qpn	输入	unsigned int	qp number

参数名称	输入/输出	类型	描述
qp_info	输出	struct dcmi_qp_info*	查询到的qp信息 #define IP_ADDRESS_LEN 40 struct dcmi_qp_info { unsigned char status; // qp状态 unsigned char type; // qp类型 char ip[IP_ADDRESS_LEN]; // 目标ip地址 unsigned short src_port; // 源端口 unsigned int src_qpn; // 源qp号 unsigned int dst_qpn; // 目标qp号 unsigned int send_psn; // 发送的psn unsigned int recv_psn; // 接收的psn char reserved[32]; // 保留字段 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口在Linux物理机特权容器场景下支持使用。

表 2-143 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 port_id = 0;
unsigned int qpn = 0;
struct dcmi_qp_info qp_info = {0};
ret = dcmi_get_qp_info(card_id, device_id, port_id, qpn, &qp_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.119 dcmi_get_hccs_link_bandwidth_info 接口原型

函数原型

int dcmi_get_hccs_link_bandwidth_info(int card_id, int device_id, struct dcmi_hccs_bandwidth_info *hccs_bandwidth_info)

功能说明

查询HCCS链路带宽。

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
hccs_bandwidth_info	输出	struct dcmi_hccs_bandwidth_info *	<pre>#define DCMI_HCCS_MAX_PCS_NUM 16 struct dcmi_hccs_bandwidth_info { int profiling_time; //采样间隔时间，取值范围1~1000，单位ms double total_txbw; //总发送数据带宽，单位GB/S double total_rxbw; //总接收数据带宽，单位GB/S double tx_bandwidth[DCMI_HCCS_MAX_PCS_NUM]; //单链路发送数据带宽，单位GB/S double rx_bandwidth[DCMI_HCCS_MAX_PCS_NUM]; //单链路接收数据带宽，单位GB/S };</pre>

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口支持物理机特权容器场景。

表 2-144 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 id = 0;
int chipid = 0;
int ret;
struct dcmi_hccs_bandwidth_info hccs_bandwidth_info = {0};
hccs_bandwidth_info.profilng_time = 1000;//采样间隔时间
ret = dcmi_get_hccs_link_bandwidth_info(id, chipid, &hccs_bandwidth_info);
if (ret != 0) {
    return ret;
}
...
```

2.120 dcmi_get_device_cpu_freq_info 接口原型

函数原型

```
int dcmi_get_device_cpu_freq_info(int card_id, int device_id, int *enable_flag)
```

功能说明

获取CPU频率信息。

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]。
enable_flag	输出	Int *	CPU频率值： 0: 1.9GHz 1: 1.0GHz

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-145 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_cpu_freq_info (card_id, device_id, &enable_flag);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

2.121 dcmi_get_device_chip_info_v2 接口原型

函数原型

```
int dcmi_get_device_chip_info_v2(int card_id, int device_id, struct
dcmi_chip_info_v2 *chip_info)
```

功能说明

获取NPU、MCU、CPU的dcmi_chip_info_v2信息，包含chip_name, chip_ver, aicore_cnt, npu_name。

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
chip_info	输出	struct dcmi_chip_info_v2 *	chip-info信息，包含chip_name, chip_ver, aicore_cnt, npu_name。 struct dcmi_chip_info_v2 { unsigned char chip_type[MAX_CHIP_NAME_LEN];//芯片类型 unsigned char chip_name[MAX_CHIP_NAME_LEN];//芯片名称 unsigned char chip_ver[MAX_CHIP_NAME_LEN];//芯片版本号 unsigned int aicore_cnt;//aicore数量 unsigned char npu_name[MAX_CHIP_NAME_LEN];//芯片名称 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

表 2-146 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_chip_info_v2 chip_info;  
ret = dcmi_get_device_chip_info_v2(card_id, device_id, &chip_info);  
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](#)
- [3.2 dcmi_clear_pcie_error_cnt接口原型](#)
- [3.3 dcmi_get_device_p2p_enable接口原型](#)
- [3.4 dcmi_get_p2p_enable接口原型](#)
- [3.5 dcmi_set_device_clear_ecc_statistics_info接口原型](#)
- [3.6 dcmi_set_device_mac接口原型](#)
- [3.7 dcmi_get_device_mac接口原型](#)
- [3.8 dcmi_get_device_gateway接口原型](#)
- [3.9 dcmi_set_device_gateway接口原型](#)
- [3.10 dcmi_get_device_ip接口原型](#)
- [3.11 dcmi_set_device_ip接口原型](#)
- [3.12 dcmi_set_device_ecc_enable接口原型](#)
- [3.13 dcmi_config_ecc_enable接口原型](#)
- [3.14 dcmi_get_nve_level接口原型](#)
- [3.15 dcmi_set_nve_level接口原型](#)
- [3.16 dcmi_set_device_share_enable接口原型](#)
- [3.17 dcmi_get_device_share_enable接口原型](#)
- [3.18 dcmi_set_device_user_config接口原型](#)
- [3.19 dcmi_set_user_config接口原型](#)
- [3.20 dcmi_get_user_config接口原型](#)
- [3.21 dcmi_clear_device_user_config接口原型](#)
- [3.22 dcmi_get_device_cpu_num_config接口原型](#)
- [3.23 dcmi_set_device_cpu_num_config接口原型](#)

- [3.24 dcmi_create_capability_group接口原型](#)
- [3.25 dcmi_delete_capability_group接口原型](#)
- [3.26 dcmi_get_capability_group_info接口原型](#)
- [3.27 dcmi_get_capability_group_aicore_usage接口原型](#)

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。

参数名称	输入/输出	类型	描述
enable_flag	输出	int *	<ul style="list-style-type: none">0：禁用1：使能

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
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_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 *	<ul style="list-style-type: none">0：禁用1：使能

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
N	N	N

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

须知

该接口会导致需要隔离的片上内存多bit ECC故障地址被清除，故障地址冷隔离失效，具有较高风险，请谨慎使用。

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
Y	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]。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}; // 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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	N

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]。

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, //虚拟网口 DCMI_ROCE_PORT = 1, //roce网口 DCMI_INVALID_PORT }; 不支持DCMI_VNIC_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]; //ipv6地址 unsigned char ip4[4]; //ipv4地址 } u_addr; enum dcmi_ip_addr_type ip_type; //ip 类型 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
Y	Y	Y

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, //虚拟网口 DCMI_ROCE_PORT = 1, //roce网口 DCMI_INVALID_PORT }; 不支持DCMI_VNIC_PORT。

参数名称	输入/输出	类型	描述
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]; //ipv6地址 unsigned char ip4[4]; //ipv4地址 } u_addr; enum dcmi_ip_addr_type ip_type; //ip类型 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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（仅作为示例，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]。
input_type	输入	enum dcmi_port_type	指定网口类型。 enum dcmi_port_type { DCMI_VNIC_PORT = 0, //虚拟网口 DCMI_ROCE_PORT = 1, //roce网口 DCMI_INVALID_PORT };
port_id	输入	int	指定网口号，保留字段。取值范围：[0, 255]。

参数名称	输入/输出	类型	描述
ip	输出	struct dcmi_ip_addr *	IP地址 struct dcmi_ip_addr { union { unsigned char ip6[16]; //ipv6地址 unsigned char ip4[4]; //ipv4地址 } u_addr; enum dcmi_ip_addr_type ip_type; //ip类型 }; 说明 支持IPv6协议，ip6数组中第一位元素的值为该IP信息的前缀，合法值为0~128，其他元素无意义。
mask	输出	struct dcmi_ip_addr *	mask地址 struct dcmi_ip_addr { union { unsigned char ip6[16]; //ipv6地址 unsigned char ip4[4]; //ipv4地址 } u_addr; enum dcmi_ip_addr_type ip_type; //ip类型 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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, //虚拟网口 DCMI_ROCE_PORT = 1, //roce网口 DCMI_INVALID_PORT }; 不支持DCMI_VNIC_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]; //ipv6地址 unsigned char ip4[4]; //ipv4地址 } u_addr; enum dcmi_ip_addr_type ip_type; //ip类型 }; 说明 支持IPv6协议，ip6数组中第一位元素的值为该IP信息的前缀，合法值为0~128，其他元素无意义。
mask	输入	struct dcmi_ip_addr *	mask地址 struct dcmi_ip_addr { union { unsigned char ip6[16]; //ipv6地址 unsigned char ip4[4]; //ipv4地址 } u_addr; enum dcmi_ip_addr_type ip_type; //ip类型 };

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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，需要复位标卡，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。

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	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_ADD R, DCMI_HBM_RECORDED_MULTI_ADDR, DCMI_DEVICE_TYPE_NONE = 0xff }; 目前支持 DCMI_DEVICE_TYPE_DDR //ddr类型内 存。
enable_flag	输入	int	<ul style="list-style-type: none">0：禁用1：使能

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
N	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，需要复位标卡，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	<ul style="list-style-type: none">0：禁用1：使能

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
N	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]。
level	输出	int*	查询功耗和算力档位。 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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]。

参数名称	输入/输出	类型	描述
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

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
N	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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

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
N	N	N

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：mac_info。支持用户自定义名称配置。 配置项功能说明如下：
buf_size	输入	unsigned int	buf长度，单位为Byte，最大长度为1K Byte。 支持mac_info配置。 目前支持处理的配置项名称如下： 如果配置"mac_info"：buf_size参数固定配置为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	char *	buf指针，指向配置项内容。 支持mac_info配置。 目前支持处理的配置项名称如下： <ul style="list-style-type: none">针对"mac_info"配置项，默认值为空，包含内容如下：<ul style="list-style-type: none">buf[0]~buf[1]: crc_value，对应于buf[2]~buf[10]的CRC校验值；buf[2]: data_length，固定为9；buf[3]: mac_id，对应于MAC_ID；buf[4]: mac_type，对应于MAC类型；buf[5]~buf[10]: mac_addr，对应于MAC地址，占6个字节；buf[11]~buf[15]: 预留，当前没有使用，可置为全0。 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。
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：mac_info、cert_expired_threshold。支持用户自定义名称配置。 配置项功能说明如下： "mac_info"：用于设置MAC地址。 "cert_expired_threshold"：用于设置证书过期时间阈值。

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mac_info、cert_expired_threshold配置。 目前支持处理的配置项名称如下： 如果配置"mac_info"：buf_size参数固定配置为16。 如果配置“cert_expired_threshold”：buf_size参数固定配置为1。 除了如上固定的名称外，其他最大长度不超过1024 Byte。
buf	输入	unsigned char *	buf指针，指向配置项内容。 支持mac_info配置。 目前支持处理的配置项名称如下： <ul style="list-style-type: none">针对"mac_info"配置项，默认值为空，包含内容如下：<ul style="list-style-type: none">buf[0]~buf[1]：crc_value，对应于buf[2]~buf[10]的CRC校验值；buf[2]：data_length，固定为9；buf[3]：mac_id，对应于MAC_ID；buf[4]：mac_type，对应于MAC类型；buf[5]~buf[10]：mac_addr，对应于MAC地址，占6个字节；buf[11]~buf[15]：预留，当前没有使用，可置为全0。 支持cert_expired_threshold配置。 <ul style="list-style-type: none">针对"cert_expired_threshold"配置项，默认值为空，包含内容如下：<ul style="list-style-type: none">buf[0]：证书过期时间阈值。 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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。

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：mac_info、cert_expired_threshold。支持用户自定义名称配置。 配置项功能说明如下： "mac_info"：用于获取MAC地址。 "cert_expired_threshold"：用于获取证书过期时间阈值。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 = "mac_info";
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：mac_info。支持用户自定义名称配置。 配置项功能说明如下： "mac_info"：用于设置MAC地址。 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

该接口设置的内容会涉及Flash擦写，由于Flash擦写次数是有限制的，所以不建议频繁调用该接口。

该接口需调入TEEOS，耗时较长，不支持在接口调用时触发休眠唤醒，如果触发休眠，有较大可能造成休眠失败。

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_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_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]。
buf	输出	unsigned char	buf指针，指向配置项内容。具体请参见 3.23 dcmi_set_device_cpu_num_config接口原型 。
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

```
...
#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;
}
...
```

3.23 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。需要复位标卡配置才能生效。

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]。

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。具体包含如下： <ul style="list-style-type: none">buf[0]：Control CPU数量；buf[1]：data CPU数量；buf[2]：AI CPU数量；
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

```
...
#define BUF_SIZE 16
int ret = 0;
int card_id = 0;
int device_id = 0;
unsigned char buf[BUF_SIZE] = {1,0,7};
ret=dcmi_set_device_cpu_num_config(card_id, device_id, buf, BUF_SIZE );
if (ret != 0){
    //todo:记录日志
    return ret;
}
...
```

3.24 dcmi_create_capability_group 接口原型

函数原型

```
int dcmi_create_capability_group(int card_id, int device_id, int ts_id, struct dcmi_capability_group_info *group_info)
```

功能说明

创建昇腾虚拟化实例配置信息。

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]。
ts_id	输入	int	算力类型id： typedef enum { DCMI_TS_AICORE = 0, DCMI_TS_AIVECTOR, } DCMI_TS_ID; 目前不支持DCMI_TS_AIVECTOR

参数名称	输入/输出	类型	描述
group_info	输入	struct dcmi_capability_group_info *	<p>算力组信息。</p> <p>算力信息结构体为：</p> <pre>#define DCMI_COMPUTE_GROUP_INFO_RES_NUM - AICORE_MASK_NUM 8 #define DCMI_AICORE_MASK_NUM 2 struct dcmi_capability_group_info { unsigned int group_id; unsigned int state; unsigned int extend_attribute; unsigned int aicore_number; unsigned int aivector_number; unsigned int sdma_number; unsigned int aicpu_number; unsigned int active_sq_number; unsigned int aicore_mask [DCMI_AICORE_MASK_NUM]; unsigned int res[DCMI_COMPUTE_GROUP_INFO_RES_NUM - DCMI_AICORE_MASK_NUM]; };</pre> <ul style="list-style-type: none">group id表示算力组id，全局唯一。可配置范围：0~3，创建重复失败。state表示算力组创建状态。0： not created, 1: created。该字段作为输入时，无意义。extend_attribute表示默认昇腾虚拟化实例标志，1：表示系统默认使用该昇腾虚拟化实例，其他值表示不使用该昇腾虚拟化实例为默认组，缺省值：0。aicore_number可配置范围：0~8或255，具体参照如下说明。aivector_number、sdma_number、aicpu_number、active_sq_number仅支持配置为255，表示配置为剩余可用值，不支持切分。aicore_mask表示aicore的编号掩码，其中每个bit表示一个core。1：该算力组内aicore，0：非算力组内aicore。该字段作为输入时，无意义。res表示预留参数。

参数名称	输入/输出	类型	描述
			<div>说明</div> <div>group_info参数的约束说明：</div> <ul style="list-style-type: none">group_id取值范围为0~3，且每次创建时需要使用未使用过的group id，否则将会返回失败。当创建不同分组时，各参数变量需要满足以下条件：<div>$\sum_{group_id} aicore_number_{group_id} \leq total_aicore_number$</div><ul style="list-style-type: none">total_aicore_number为8。当aicore_number配置为255时，表示配置为剩余可用值，如果多个group配置255，这几个group共用剩余可用值。

返回值

类型	描述
int	<div>处理结果：</div> <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 仅Control CPU开放形态使用该接口。
- 调用该接口前，请先调用3.26 dcmi_get_capability_group_info接口原型查询已配置的资源信息，结合ts_id参数的说明和group_info参数的约束说明进行配置，否则可能出现接口调用失败。如果需要删除已有配置，可以调用3.25 dcmi_delete_capability_group接口原型接口进行删除。
- 不建议与dcmi_create_vdevice相关算力功能共用，共用情况有可能存在异常。

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
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
int ts_id = 0;
struct dcmi_capability_group_info group_info = {0};
group_info.group_id = 0;
group_info.aivector_number = 255;
group_info.sdma_number = 255;
group_info.aicpu_number = 255;
group_info.active_sq_number = 255;
group_info.aicore_number = 8;

ret = dcmi_create_capability_group(card_id, dev_id, ts_id, &group_info);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

3.25 dcmi_delete_capability_group 接口原型

函数原型

```
int dcmi_delete_capability_group(int card_id, int device_id, int ts_id, int
group_id)
```

功能说明

删除昇腾虚拟化实例配置信息。

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]。
ts_id	输入	int	算力类型id： typedef enum { DCMI_TS_AICORE = 0, DCMI_TS_AIVECTOR, } DCMI_TS_ID; 目前不支持DCMI_TS_AIVECTOR
group_id	输入	int	算力组id。算力组id未创建时，会返回失败。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

仅Control CPU开放形态使用该接口。

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
N	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
int dev_id = 0;
int ts_id = 0;
int group_id = 0;
ret = dcmi_delete_capability_group(card_id, dev_id, ts_id, group_id);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

3.26 dcmi_get_capability_group_info 接口原型

函数原型

int dcmi_get_capability_group_info(int card_id, int device_id, int ts_id, int group_id, struct dcmi_capability_group_info *group_info, int group_count)

功能说明

获取昇腾虚拟化实例配置信息。

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]。
ts_id	输入	int	算力类型id： typedef enum { DCMI_TS_AICORE = 0, DCMI_TS_AIVECTOR, } DCMI_TS_ID; 目前不支持DCMI_TS_AIVECTOR
group_id	输入	int	算力组id，范围：[0, 3]或者-1。当group_id为-1时获取所有的group信息。

参数名称	输入/输出	类型	描述
group_info	输出	struct dcmi_capability_group_info*	<p>算力组信息。</p> <p>算力信息结构体为：</p> <pre>#define DCMI_COMPUTE_GROUP_INFO_RES_NUM #define DCMI_AICORE_MASK_NUM 2 struct dcmi_capability_group_info { unsigned int group_id; unsigned int state; unsigned int extend_attribute; unsigned int aicore_number; unsigned int aivector_number; unsigned int sdma_number; unsigned int aicpu_number; unsigned int active_sq_number; unsigned int aicore_mask [DCMI_AICORE_MASK_NUM]; unsigned int res[DCMI_COMPUTE_GROUP_INFO_RES_NUM - DCMI_AICORE_MASK_NUM]; };</pre> <ul style="list-style-type: none">group id表示算力组id，全局唯一。可配置范围：0~3。state表示算力组创建状态。0: not created, 1: created。extend_attribute表示默认昇腾虚拟化实例标志，1：系统默认使用该昇腾虚拟化实例，其他值表示不使用该昇腾虚拟化实例为默认组，缺省值：0。aicore_number表示算力组的aicore数量。aivector_number、sdma_number、aicpu_number、active_sq_number分别表示aivector、sdma、aicpu、active资源数量，目前会返回255，表示当前资源为共享资源，不支持切分。aicore_mask表示aicore的编号掩码，其中每个bit表示一个core。1：该算力组内aicore，0：非算力组内aicore。res表示预留参数。

参数名称	输入/输出	类型	描述
group_count	输入	int	参数group_info的数组长度。 在group_id为-1时，group_count需要不小于4。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

仅Control CPU开放形态使用该接口。

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 ret = 0;
int card_id = 0;
int dev_id = 0;
int ts_id = 0;
struct dcmi_capability_group_info group_info[4] = {0};
ret = dcmi_get_capability_group_info(card_id, dev_id, ts_id, -1, group_info, 4);
if (ret != 0){
    //todo: 记录日志
    return ret;
}
...
```

3.27 dcmi_get_capability_group_aicore_usage 接口原型

函数原型

```
int dcmi_get_capability_group_aicore_usage(int card_id, int device_id, int group_id, int *rate)
```

功能说明

获取昇腾虚拟化实例的aicore利用率。

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]。
group_id	输入	int	算力组id 范围：[0, 3]。
rate	输出	int*	aicore利用率。 当前group的aicore_number为0，aicore利用率为0，无实际意义。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

仅Control CPU开放形态使用该接口。

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;
int card_id = 0;
int device_id = 0;
int group_id = 0;
int rate = 0;
ret = dcml_get_capability_group_aicore_usage(card_id, device_id, group_id, &rate);
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](#)
- 4.2 [dcmi_mcu_get_version接口原型](#)
- 4.3 [dcmi_set_mcu_upgrade_stage接口原型](#)
- 4.4 [dcmi_mcu_upgrade_control接口原型](#)
- 4.5 [dcmi_set_mcu_upgrade_file接口原型](#)
- 4.6 [dcmi_mcu_upgrade_transfile接口原型](#)
- 4.7 [dcmi_get_mcu_upgrade_status接口原型](#)
- 4.8 [dcmi_mcu_get_upgrade_status接口原型](#)
- 4.9 [dcmi_mcu_get_upgrade_statues接口原型](#)

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	Y	Y

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： <ul style="list-style-type: none">• 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： <ul style="list-style-type: none">• 成功：返回0。• 失败：返回码请参见9 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,//设置mcu升级状态为开始 MCU_UPGRADE_VALIDATE = 3,//设置mcu升级状态为校验 VRD_UPGRADE_START = 11, MCU_UPGRADE_NONE //表示无效 }; 仅支持MCU_UPGRADE_START、MCU_UPGRADE_VALIDATE

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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： <ul style="list-style-type: none">1：启动升级3：生效

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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升级包对应的bin包。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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升级包对应的bin包。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	N	N

调用示例

```
...
int ret = 0;
int card_id = 0;
ret = dcmi_mcu_upgrade_transfile(card_id, "./mcu.bin");
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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
Y	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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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_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

5.2 接口说明

5.3 调用示例

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，NPU相关业务可通过fuser软件查询，具体参考[8.2 查询NPU业务进程](#)。
- 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

带外标卡复位功能依赖ipmitool软件，需要提前下载并加载驱动。详细操作请参见8.1 准备ipmitool软件章节。

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

带外标卡复位功能依赖ipmitool软件，需要提前下载并加载驱动。详细操作请参见[8.1 准备ipmitool软件](#)章节。

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 } 说明 Atlas 200T A2 Box16 异构子框、Atlas 900 A2 PoD 集群基础单元支持 INBAND_CHANNEL。

返回值

类型	描述
int	处理结果： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 返回码。

异常处理

无。

约束说明

- 带外标卡复位功能依赖ipmitool软件，需要提前下载并加载驱动。详细操作请参见8.1 准备ipmitool软件章节。
- 该命令在Linux物理机特权容器场景下支持。
- 对于Atlas 900 A2 PoD 集群基础单元和Atlas 200T A2 Box16 异构子框，正常启动的NPU不支持单NPU复位。若同一设备上的多个NPU中部分降P启动，部分正常启动，则降P启动的NPU支持单NPU复位；正常启动的NPU不支持单NPU复位，即执行单NPU复位命令时所有NPU都将被复位，包括被降P启动的NPU。查询指定NPU是否为降P启动的方法请参见《 Atlas A2 中心推理和训练硬件 24.1.0 npu-smi 命令参考 》中的“[查询指定NPU是否为降P启动](#)” 章节或《 Atlas A2 中心推理和训练硬件 24.1.0 npu-smi 命令参考 》中的“[对指定NPU进行降P启动](#)” 章节。

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
Y	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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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 调用示例

5.3.1 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;
}
```

5.3.2 带外复位调用示例

```
#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;
    int channel_state = 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_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++)
        {
            // 检查带外通道
            ret = dcmi_get_device_outband_channel_state(card_id_list[card_id], device_id, &channel_state);
            if (ret == DCMI_ERR_CODE_NOT_SUPPORT) {
                printf("dcmi_get_device_outband_channel_state is not support.\n");
            }
            if (ret != 0 || channel_state != 1) {
                printf("\nChip reset is not supported. Please check BMC version and IPMI driver.\n");
                printf("Please reboot OS to reset chip if there is no BMC.\n");
                return -9005;
            }
            // 预复位
            ret = dcmi_pre_reset_soc(card_id_list[card_id], device_id);
            if (ret != 0) {
                printf("Failed to start prereset chip %d.\n", device_id);
                printf("Please reboot OS to reset card.\n");
                return ret;
            }
            // 复位
            ret = dcmi_reset_device(card_id_list[card_id], device_id);
            if (ret != 0) {
                printf("Failed to reset chip %d in card %d.\n", device_id, card_id_list[card_id]);
                printf("Please reboot OS to reset card.\n");
                return ret;
            }
            sleep(3); // 延时3s

            // 重新扫描
            ret = dcmi_rescan_soc(card_id_list[card_id], device_id);
            if (ret != 0) {
                printf("Failed to rescan chip %d.\n", device_id);
                printf("Please reboot OS to reset card.\n");
                return ret;
            }
        }
    }
}
```

```
        // 重新扫描的操作只是触发驱动启动扫描，这里等待2秒  
        sleep(2);  
        printf("reset successful! card_id is %d, device_id:%d\n", card_id_list[card_id], device_id);  
    }  
}  
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](#)
- 6.2 [dcmi_set_customized_info_api接口原型](#)
- 6.3 [dcmi_mcu_set_license_info接口原型](#)
- 6.4 [dcmi_get_card_customized_info接口原型](#)
- 6.5 [dcmi_get_customized_info_api接口原型](#)
- 6.6 [dcmi_mcu_get_license_info接口原型](#)

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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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

Linux物理机		Linux物理机容器
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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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： <ul style="list-style-type: none">成功：返回0。失败：返回码请参见9 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;
}
```

您可以从已安装驱动和npu-smi工具环境的“/usr/local/dcmi/”目录下获取dcmi_interface_api.h文件。

使用调用示例

步骤1 将get_card_num_list.c、dcmi_interface_api.h传到Host侧服务器的同一个目录下。

步骤2 登录到Host侧服务器。

步骤3 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。

```
gcc get_card_num_list.c /usr/local/dcmi/libdcmi.so -o card_list -lm
```

步骤4 执行可执行文件card_list。

```
./card_list
```

----结束

8 常用操作

8.1 准备ipmitool软件

8.2 查询NPU业务进程

8.1 准备 ipmitool 软件

带外标卡复位功能依赖ipmitool软件，需要提前下载并加载驱动。

操作步骤

步骤1 下载ipmitool软件，例如：

- CentOS: **yum install ipmitool**
- Ubuntu: **apt-get install ipmitool**

步骤2 执行如下命令，加载驱动。

```
modprobe ipmi_si  
modprobe ipmi_devintf  
modprobe ipmi_msghandler
```

步骤3 执行如下命令，查看驱动是否加载成功。

```
lsmod | grep ipmi
```

如果没有加载成功，执行[步骤2](#)重新加载驱动。

📖 说明

如果重启OS后，需要重新执行[步骤2](#)~[步骤3](#)。

----结束

8.2 查询 NPU 业务进程

可使用fuser软件查询NPU业务进程。

使用方法

步骤1 安装fuser软件，例如：

- CentOS: **yum install psmisc**
- Ubuntu: **apt-get install psmisc**

步骤2 执行如下命令查询NPU当前时刻的业务进程。

fuser -uv /dev/davinci* /dev/devmm_svm /dev/hisi_hdc

若无回显信息，表示当前时刻无业务进程。

----结束

9 返回码

须知

对应返回码的处理方法需要联系华为技术支持。

表 9-1 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_OPERATION_FAIL	-8003	内存接口操作失败。
DCMI_ERR_CODE_SECURE_FUNCTION_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返回码	值	含义
DCMI_ERR_CODE_NOT_REDAY	-8012	尚未就绪，请重试。
DCMI_ERR_CODE_NOT_SUPPO RT_IN_CONTAINER	-8013	在容器中不支持该接口。
DCMI_ERR_CODE_RESET_FAIL	-8015	复位失败。
DCMI_ERR_CODE_ABORT_OPE RATE	-8016	复位取消。
DCMI_ERR_CODE_IS_UPGRADI NG	-8017	正在升级中。
DCMI_ERR_CODE_RESOURCE_ OCCUPIED	-8020	设备资源被占用。
DCMI_ERR_CODE_CONFIG_INF O_NOT_EXIST	-8023	查询的配置信息不存在。
DCMI_ERR_CODE_NOT_SUPPO RT	-8255	设备ID/功能不支持。