

云天励飞 DESDK API 手册

正在修改 [<input checked="" type="checkbox"/>] 正式发布 [<input type="checkbox"/>]	当前版本:	V1.1.0
	完稿日期:	2020.9.22
	审核日期:	2020.9.22

深圳云天励飞技术有限公司

声明

本文档是云天励飞提供接口使用文档，便于合作的企事业单位使用，云天励飞保留所有接口使用权利，未经许可不得公开接口使用信息给任何非授权企业和个人。

更新记录

版本	修改日期	修改说明
V0.1.0	2020.4.8	初始版本
V1.0.0	2020.7.15	v1.0.0 版本更新
V1.0.1	2020.8.12	增加 CV 算子
V1.0.2	2020.9.9	增加跟踪算子接口说明，删除部分章节
V1.1.0	2020.9.22	增加模型加载卸载接口说明；增加芯片资源使用情况查询接口说明

目录

声明.....	2
更新记录.....	3
目录.....	4
1. 概述.....	6
2. 基本概念.....	6
3. 类型和结构定义.....	6
3.1. 数据类型.....	6
3.1.1. NDArrary.....	6
3.1.2. Prop.....	7
3.1.3. 其它类型.....	8
3.1.4. Thread.....	8
3.1.5. 常量定义.....	10
3.2. 内存接口.....	10
3.2.1. 内存申请.....	10
3.2.2. 内存释放.....	11
3.2.3. 虚拟地址转物理地址.....	12
3.3. 流程接口.....	13
3.3.1. 系统初始化.....	13
3.3.2. 系统去初始化.....	13
3.3.3. SDK 初始化.....	13
3.3.4. SDK 去初始化.....	14
3.3.5. SDK 状态查询.....	14
3.3.6. 读取 SDK 配置文件.....	15
3.3.7. 读取 SDK 版本信息.....	15
3.3.8. 类型注册.....	15
3.3.9. 方法获取.....	16
3.3.10. 模型加载.....	16
3.3.11. 模型卸载.....	17
3.4. 图像处理接口.....	17
3.4.1. 图像格式转换.....	17
3.4.2. 图像尺寸转换.....	18
3.4.3. 图像格式尺寸转换.....	19
3.4.4. 图像翻转.....	20
3.4.5. 图像旋转.....	21
3.4.6. 抠图.....	21
3.4.7. 边缘检测.....	22
3.4.8. 高斯模糊.....	23
3.4.9. 自然指数运算.....	23
3.4.10. np.where.....	24
3.4.11. TopN 选择.....	24
3.4.12. 向量乘法.....	25

3.4.13. 特征比对.....	25
3.4.14. 亮度计算.....	26
3.4.15. 拉普拉斯变换.....	27
3.4.16. 兴趣区标注.....	27
3.4.17. 均值方差.....	28
3.4.18. 仿射变换.....	29
3.5. kcf 跟踪.....	29
3.6. Python 接口.....	32
3.6.1. 函数接口.....	32
3.6.1.1. 设备初始化.....	32
3.6.2. Python 类.....	33
3.6.2.1. NNDeploy 类.....	33
3.6.2.2. NNModule 类.....	34
3.7. 设备管理接口.....	35
3.7.1. dcmi 模块初始化.....	35
3.7.2. dcmi 模块去初始化.....	35
3.7.3. 获取支持的最大 DeepEye 芯片个数.....	36
3.7.4. 设置事件回调函数.....	36
3.7.5. 执行生产测试.....	37
3.7.6. DeepEye 芯片版本升级.....	37
3.7.7. DeepEye 芯片版本下载.....	38
3.7.8. 获取设备日志信息（暂未实现）.....	39
3.7.9. 获取设备状态.....	39
3.8. 资源使用情况查询接口.....	40
3.8.1. 数据类型定义.....	40
3.8.2. 启动资源使用统计.....	41
3.8.3. 获取资源使用情况.....	41
3.8.4. 停止资源使用统计.....	42
4. FAQ.....	42

1. 概述

本文档介绍如何使用 DESDK API 接口开发应用程序。

2. 基本概念

DESDK API 接口按照功能层级可以分为核心层（CORE）、硬件抽象层（HAL）和功能模块层（FML）。核心层提供基本数据结构定义，初始化、序列化、线程等，硬件抽象层封装各种硬件接口，功能模块层通过调用硬件抽象层接口，实现项目中所涉及到的各片外功能模块，隐藏具体的模块操作细节，并为上层提供简单清晰的调用接口。

3. 类型和结构定义

3.1. 数据类型

3.1.1. NDArrary

de::NDArrary 为数据保存和传递的基本结构，用户无需了解内部结构，可以通过接口从图片中获取，再传递给后面的处理流程。定义在 de_ndarray.h 中。

de::NDArrary 使用引用计数机制来实现数据的共享和零拷贝。在使用值传递的方式传递 NDArrary 对象作为参数的函数中，仅增加 NDArrary 内部的引用计数，并不会拷贝 NDArrary 承载的数据。当引用计数减为 0 时，自动调用 NDArrary 的析构函数，释放第一个 NDArrary 对象创建时申请的控制域、数据域空间。

NDArrary 扩展了内存布局，以支持序列化/反序列化、在不同平台间传输。完整的 NDArrary 内存布局如下图所示：



变量说明

变量名	类型	作用
-----	----	----

data_	Container*	保存 NDArry 的实际控制信息和数据域信息，实现 NDArry 的引用计数
-------	------------	---

方法说明

常用方法见下表中所列。

方法名	作用
TotalSize	返回 NDArry 完整内存布局总大小
GetTensorData	返回指向 tensor 数据域内存的地址
GetTensorDataSize	返回 tensor 数据域空间的大小
GetTransportData	返回指向传输控制头内存的地址
GetTransportDataCap	返回传输控制头所占空间的大小
GetBearData	返回指向 bear 数据域内存的地址
GetBearDataCap	返回 bear 数据空间的大小
GetTensor	返回 NDArry 内部的 DLTensor 成员的地址
Empty	<p>NDArry 的静态成员函数，创建一个 NDArry 对象，根据输入的 data size，申请完整的所需内存空间。</p> <p>Empty 函数有多个重载形式。最常用的形式只用传递一个 tensor data size，提供所需数据域空间大小即可。</p> <p>用户可在调用 Empty 获得一个 NDArry 对象后，再调用 GetTensorData，获取 tensor 数据域空间地址，对这块地址做读写操作</p>
Shrink	<p>调整 NDArry 的 size 信息，重新写入到传输控制头中。</p> <p>该函数的典型使用场景：先按照某一固定的 size（例如 4M），调用 Empty 构建一个 NDArry 对象。NDArry 使用完成后，tensor data 实际使用了 3M。在将该 NDArry 对象传输到另外一个平台前，调用 Shrink 把有效数据的 size（3M）写入到传输控制头中，以防止在另外一个平台上访问到无效数据。</p> <p>Shrink 只是把调整后的 tensor data size 信息写入到传输控制头中，并不会真正地调整内存空间而已</p>
CalcCrc	<p>计算 NDArry 的 CRC 值</p> <p>该函数的典型使用场景：在把一个 NDArry 对象发送到另外一个平台前，调用该函数计算得到 CRC，并写入到传输控制头中。在另外一个平台上接收到 NDArry 完整数据后，恢复出 NDArry 对象，调用该函数计算得到 CRC，并和传输控制头中的 CRC 值比较，在判断传输控制中是否发生错误</p>

3.1.2. Prop

Prop 为属性类型的基本结构，为 Key-Value 对，支持多种结构的设置、获取操作。定义在 de_prop.h 中。

Prop 类通常不会在顶层代码中使用，而是在一个类的内部，定义一个 Prop 类对象成员。类的设计者预先设计好该类提供哪些属性以及属性的默认值。用户可按需重设某些属性的默认值，但不能给该类增加新的属性。类的函数执行时，把所需属性读取出来，使用用户重设值或者默认值。

变量说明

变量名	类型	作用
-----	----	----

int_paras_	map<string, int>	string-int 属性集
double_paras_	map<string, float>	string-double 属性集
string_paras_	map<string, string>	string-string 属性集

方法说明

方法名	作用
SetDefault	预设一个属性的默认值 该函数应由类的设计者调用
Set	重设一个属性的默认值 该函数通常由类的使用者调用。用户只能修改已有属性的默认值，而不能增加新的属性。也即只能在类的设计者预先调用过 SetDefault(keyXXX, ...)之后，用户调用 Set(keyXXX, ...)才有效，否则报错。这样做是为了避免代码拼写错误，导致属性设置不生效，程序执行结果不符合预期，在更早阶段就报告出错误，方便调试。
Get	获取某个属性的值

3.1.3. 其它类型

de_c_type.h 中定义了通用的一些数据类型，包括图像格式类型、Tensor 格式类型等。需要关注的数据类型如下：

- deRect：基本 Rect 类型
- deShapeCode：Tensor 中存储的数据形状类型
- deSize：基本 Size 类型
- dePixelFormat：常用的图像格式类型
- DecBearData：解码器输出的附加数据类型
- ExtInfo：NDArray 中携带的扩展信息
- DeNodeState：算子具体状态

变量说明

无

方法说明

无

3.1.4. Thread

Thread 类是各种业务接口、Node 算子的父类，通过任务队列将各种任务串联起来，驱动任务执行。子类可通过重写 Start、Proc 等虚函数实现自己的业务逻辑。定义在 de_thread.h 中。

变量说明

变量名	类型	作用
kMaxPoutNum	int	pout 最大数目，默认为 16

KMaxUserPoutNum	int	用户可用的 pout 最大数目，默认为 15
pin	PIN	用于 Thread 的输入
pout	POUT[kMaxPoutNum]	用于 Thread 的输出
name_	string	线程名
resource_	ASyncRes*	若该值不为空，说明 Thread 工作在资源受限场景下
reorder_db_	ASyncDisorder* [kMaxPoutNum]	异步调度，输出任务重排数据库
kTxMaxQueueSz	int	最大输出队列个数，默认为 8
prev_	ThreadPrevPtr*	前一个异步线程指针
next_	ThreadNextPtr* [kMaxPoutNum]	后续异常线程指针数组
dbg_pin_grab_	int	是否抓取输入任务数据到文件中
dbg_pin_grab_file	string	用于存储输入任务数据的文件名
dbg_pin_grab_max_size_	int	用于存储输入任务数据的文件最大容量限制，默认为 4MB
profile_enable_	int	是否开启 Thread 的 profile 功能
profile_stat_period_	int	Thread 的 profile 统计周期
debug_level_	int	Thread 的 LOG 打印等级
frame_no	int	帧号 (frame number)
proc_average_time	int	处理一条任务平均所需时间，单位为 ms
proc_max_time	int	处理一条任务最长所需时间
pdb_	void*	Thread 关联的数据库指针
de_state_	DeNodeState	Thread 具体状态
tx_task_num_	int	Thread 发送出去的 Task 总数量
rx_task_num_	int	Thread 接收到的 Task 总数量
max_rx_queue_size_	int	输入队列最大深度，也即 pin 中堆积的任务最大数目
rx_queue_stat_size_ _rx_queue_stat_cnt_	int	这两个变量用于统计输入队列平均深度，也即 pin 中堆积的任务平均数量
thread_active_	bool	用于标识任务处理线程是否为 active 状态
filter_cb_func_	PackedFunc	任务过滤函数，可以通过该函数把不希望处理的任务过滤掉

方法说明

方法名	作用
Start	创建处理线程并启动，虚函数，子类可重写
Stop	停止处理线程。 该函数是一个纯虚函数，子类必须重写。子类的 Stop 函数，应保证 Start 函数创建的线程停止执行并退出。
SetName	设置线程名，以方便调试
SetRxQue	为 pin 创建一个新的队列，或使用一个已经存在的队列作为输入

SetTxQue	为第 n 个 pout 创建一个新的队列，或者使用一个已经存在的队列作为输入
GetOutput	从第 n 个 pout 中取出输出任务
SetOutput	为第 n 个 pout 设置输出任务
SetInput	为 pin 设置输入任务
SetResources	设置线程需要的资源数据库
GetResource	获取线程的资源数据库
PutResource	释放资源
SeDisorderDb	为第 n 个 pout 设置线程支持乱序排序功能数据库
SetStartTag	发送一个任务，启动乱序排序功能和异步调度
ReorderTxTask	重排输出任务
Set	设置某个属性的 key-value 对
UpdatePara	更新属性参数
SetDatabase	设置关联的数据库指针
GetThreadActive	返回处理线程是否为 active 状态
SetThreadActive	设置处理线程的 active 状态 如果用户重写了 Start 函数，创建自己的处理线程，则应在处理线程启动前、后分别调用该函数，以正确设置处理线程的状态
GetDeState	返回 Thread 的具体状态
GetTotalCpuOccup y	静态函数，获取整个系统的 cpu 使用时间（仅在 Linux 系统下有效）
GetThreadCpuOcc upy	静态函数，获取指定线程的 cpu 使用时间（仅在 Linux 系统下有效）
GetThreadMemOc cupy	静态函数，获取指定线程的 memory 使用大小（仅在 Linux 系统下有效）
Proc	用户线程的任务处理函数 该函数是一个纯虚函数。用户设计的继承于 Thread 的类必须重写该函数。
ThreadProc	通用的线程调度函数
WriteDatabase	写数据库
ReadDatabase	从数据库中读取数据，并清空数据库

3.1.5. 常量定义

参数	类型	说明
kMaxDevNum	int	支持最大的 device 数，目前为 32

3.2. 内存接口

3.2.1. 内存申请

若用户希望直接申请内存，可使用以下两个接口，alignment 用于申请有指定对齐需求的内存。

1. C 函数接口 DEMemAlloc

函数格式

```
void* DEMEMAlloc(size_t size, size_t alignment);
```

参数说明

参数	类型	说明	取值范围
size	size_t	申请内存大小，单位为 Byte	
alignment	size_t	返回地址对齐边界，只能为 2 的 N 次幂	4, 8, 16, 32, ...

返回值

申请到的内存的起始虚拟地址。其数值一定是对齐到 alignment 边界的。

函数说明

该函数只能在 dp1000 上使用。

2. C++ 函数接口 de::mem::Alloc

函数格式

```
void* de::mem::Alloc(size_t size, size_t alignment = 16);
```

参数说明

参数	类型	说明	取值范围
size	size_t	申请内存大小，单位为 Byte	
alignment	size_t	返回地址对齐边界，只能为 2 的 N 次幂	4, 8, 16, 32, ...

返回值

申请到的内存的起始虚拟地址。其数值一定是对齐到 alignment 边界的。

函数说明

该函数只能在 dp1000 上使用。

3.2.2. 内存释放

与内存申请接口相对应的，分别是以下两个内存释放接口。

1. C 函数接口 DEMemFree

函数格式

```
void DEMemFree(void* ptr);
```

参数说明

参数	类型	说明	取值范围
ptr	void*	释放内存的起始虚拟地址，必须是此前调用 DEMemAlloc 的返回值	

返回值

无。

函数说明

该函数只能在 dp1000 上使用。

2. C++ 函数接口 de::mem::Free

函数格式

```
void de::mem::Free(void* ptr);
```

参数说明

参数	类型	说明	取值范围
ptr	void*	释放内存的起始虚拟地址，必须是此前调用 de::mem::Alloc 的返回值	

返回值

无。

函数说明

该函数只能在 dp1000 上使用。

3.2.3. 虚拟地址转物理地址

在 dp1000 上，通过 DEMemAlloc 或者 de::mem::Alloc 申请到的内存，除返回虚拟地址外，还可以获取到对应的物理地址。可通过以下两个接口获取虚拟地址对应的物理地址：

1. C 函数接口 DEMemGetPhyaddr

函数格式

```
void* DEMemGetPhyaddr(void* ptr);
```

参数说明

	参数	类型	说明	取值范围
返回	ptr	void*	要查询的目的虚拟地址	

回

值

虚拟地址 ptr 对应的物理地址。

函数说明

该函数只能在 dp1000 上使用，因为其他平台 ptr 无对应的物理地址。

在 dp1000 平台上，若 ptr 在 de::mem 模块所维护的物理地址-虚拟地址映射范围内，则返回对应的物理地址；若 ptr 不在这个范围内，则返回 nullptr。

2. C++ 函数接口 de::mem::GetPhyAddr

函数格式

```
void* de::mem::GetPhyAddr (void* ptr)
```

参数说明

	参数	类型	说明	取值范围
返回	ptr	void*	要查询的目的虚拟地址	

回

值

虚拟地址 ptr 对应的物理地址。

函数说明

该函数只能在 dp1000 上使用。

若 ptr 在 de::mem 模块所维护的物理地址-虚拟地址映射范围内，则返回对应的物理地址；若 ptr 不在这个范围内，则返回 nullptr。

需注意的是，物理地址仅能在 dp1000 平台上供驱动函数（例如 DMA、USB）使用。用户代码不能直接读、写物理地址。

3.3. 流程接口

3.3.1. 系统初始化

函数格式


```
void DeSystemInit(intdev_id = 0);
```

参数说明

参数	类型	说明	取值范围
dev_id	int	device id, 目标设备的编号	0 ~ kMaxDevNum - 1

返回值

无

函数说明

在 Host 端：该函数会在和编号为 dev_id 的设备首次建立 RPC 连接的时候调用一次，之后不会再调用。dev_id 的取值范围为 0 ~ kMaxDevNum - 1，可通知指定设备完成一些初始化工作。该函数是在 RPC 连接建立的时候自动调用的，而非由用户显式调用。用户只需关注何时需要建立 RPC 连接即可。

在 Device 端：在单设备 Device 平台上，dev_id 参数会被忽略。该函数目前完成的功能包括：初始化 Device 侧 LOG 等级、启动 DSP、初始化 DSP LOG 等级。

3.3.2. 系统去初始化

函数格式

```
voidDeSystemDeinit(intdev_id = 0);
```

参数说明

参数	类型	说明	取值范围
dev_id	int	device id, 目标设备的编号	0 ~ kMaxDevNum - 1

返回值

无

函数说明

在 Host 端：Host 上的程序退出时，会检查和哪些设备建立有 RPC 连接，使用这些设备的编号作为参数，调用该函数。该函数在程序执行结束后自动调用，不需要由用户显式调用。

在 Device 端：函数体为空。

3.3.3. SDK 初始化

函数格式

```
voidInit(void);
```

参数说明

参数	类型	说明	取值范围
无			

返回值

无

函数说明

在 Host 端：函数体为空。

在 Device 端：建立 SDK RPC Server，等待 Host 端与其建立 RPC 连接。

3.3.4. SDK 去初始化

函数格式

```
void Delnit(void);
```

参数说明

参数	类型	说明	取值范围
无			

返回值

无

函数说明

在 Host 端：函数体为空。

在 Device 端：函数体为空

3.3.5. SDK 状态查询

函数格式

```
voidDeSDKAlive(void);
```

参数说明

参数	类型	说明	取值范围
无			

返回值

SDK RPC Server 是否在工作状态

函数说明

在 Host 端：总是返回 true。

在 Device 端：若 RPC Server 还在工作状态，返回 true。若 RPC Server 已退出，返回 false。

3.3.6. 读取 SDK 配置文件

函数格式

```
void DeSDKReadConfig(const std::string& cfg_file = "/root/cfg/desdk.cfg");
```

参数说明

参数	类型	说明	取值范围
cfg_file	string	SDK 配置文件路径	

返回值

无

函数说明

在 Host 端：函数体为空

在 Device 端：从一个 JSON 格式的配置文件中，读取 SDK 配置，默认读取文件路径为 "/root/cfg/desdk.cfg"。如果希望在 SDK 启动时，使用自定义的配置，可以在调用 DeSystemInit 之前调用该函数。如果没有调用该函数，或者指定的配置文件并不存在，则使用 SDK 的默认配置。

3.3.7. 读取 SDK 版本信息

函数格式

```
std::string Version(void);
```

参数说明

参数	类型	说明	取值范围
无			

返回值

SDK 版本信息

函数说明

返回包括 SDK 仓库名、分支名、提交标识的字符串，例如 "desdk:master:4464dd02"。

3.3.8. 类型注册

函数格式

DE_CLASS_REGISTER(OpName, Type)

参数说明

参数	类型	说明	取值范围
OpName	string	注册的名称	
Type	Class	注册的类型	

返回值

无

函数说明

宏定义，对类型进行全局注册，注册后可通过 OpName 字符串找到该类型。

3.3.9. 方法获取

函数格式

PackedFuncGetFunc(int dev_id, const std::string& name);

参数说明

参数	类型	说明	取值范围
dev_id	int	方法所在的设备 ID	
name	string	要获取的目标方法的名字	

返回值

保存函数信息的一个 PackedFunc 对象。

函数说明

获取已在 device 上注册的函数。如果目标函数并没有在 device 上注册，返回的 PackedFunc 对象中存储的是一个空函数对象。可以将返回值与 nullptr 做比较以判断是否成功获取到了 device 上的函数。

用法：

1) 先通过名字获取到函数对象，再调用，例如：static auto pf = Device::GetFunc(0, "FuncAddint"); int res = pf(2);

2) 直接通过名字和参数调用函数，例如：int res = Device::GetFunc(0, "FuncAddint")(2)

3.3.10. 模型加载

函数格式

```
ModelInfo* Load(const std::string& model_file, const std::string& para_file = "", bool  
is_encrypt = false);
```

参数说明

参数	类型	说明	取值范围
model_file	string	模型文件	
para_fil	string	参数文件	
is_encrypt	bool	模型加密指示	

返回值

模型信息。

函数说明

模型加载成功，返回模型信息，供后续推理等过程调用；模型加载失败，返回 nullptr。

该接口既支持加载主控侧模型，也支持加载设备侧模型。如果加载主控侧模型模型文件路径前需添加 host: 前缀，例如 host:../models/xxx.bin。

该接口需在推理等业务过程前调用。

3.3.11. 模型卸载

函数格式

```
void Unload(const std::string& model_file);
```

参数说明

参数	类型	说明	取值范围
model_file	string	模型文件	

返回值

无。

函数说明

model_file 必需与加载模型时设置的模型文件一致。一般在业务结束后调用，必须和 Load 接口配对使用。

3.4. 图像处理接口

3.4.1. 图像格式转换

目前支持类型：

```
/**@brief color space convert code
*/
typedef enum{
    kCvtYuvNv12Rgb = 0,      ///< YUV420NV12=>RGB
    kCvtYuvUyvy2Rgb,        ///< YUV422UYVY=>RGB
    kCvtYuvNv21Rgb,         ///< YUV420NV21=>RGB
    kCvtYuvYuyv2Rgb,        ///< YUV422YUYV=>RGB
    kCvtYuvNv16Rgb,         ///< YUV422NV16=>RGB
    kCvtYuvNv16YuvUYVY,     ///< YUV422NV16=>YUV422UYVY
    kCvtYuvNv16YuvYUYV,     ///< YUV422NV16=>YUV422YUYV
    kCvtYuvNv12YuvUYVY,     ///< YUV420NV12=>YUV422UYVY
    kCvtYuvNv12YuvYUYV,     ///< YUV420NV12=>YUV422YUYV
    kCvtYuvNv21YuvYUYV,     ///< YUV420NV21=>YUV422YUYV
    kCvtYuvNv21YuvUYVY,     ///< YUV420NV21=>YUV422UYVY
    kCvtYuvUYVYYuvNv12,     ///< YUV422UYVY=>YUV420NV12
    kCvtYuvUYVYYuvNv21,     ///< YUV422UYVY=>YUV420NV21
    kCvtYuvYUYVYuvNv12,     ///< YUV422YUYV=>YUV420NV12
    kCvtYuvYUYVYuvNv21,     ///< YUV422YUYV=>YUV420NV21
    kCvtRgbYuvNv12,         ///< RGB=>YUV420NV12
    kCvtRgbYuvUYVY,         ///< RGB=>YUV422UYVY
    kCvtYuvNv12Rgb888,      ///< YUV420NV12=>RGB888
    kCvtYuvUyvy2Rgb888,     ///< YUV422UYVY=>RGB888, width must be even
    kCvtYuvNv21Rgb888,      ///< YUV420NV21=>RGB888
    kCvtYuvYuyv2Rgb888,     ///< YUV422YUYV=>RGB888, width must be even
    kCvtRgbYuvNv21,         ///< RGB=>YUV420NV21
    kCvtRgbYuvYUYV          ///< RGB=>YUV422YUYV
}CvtColorCode;
```

注意：输出的 RGB 格式为 R...G...B, RGB888 为 RGBRGB....

函数格式

```
intCvtColor(de::NDArray& src, de::NDArray& dst, int cvtCode);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArra y 数据结构引用,需要用户构建实体并申请物理连续空间	具体参考 NDArra y 结构定义
dst	de::NDArray	输出图像 NDArra y 数据结构引用	具体参考 NDArra y 结构定义
cvtCode	Int	色域转换码, 参考 CvtColorCode 定义	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.2. 图像尺寸转换

支持 type:

```
enum ResizeType {
    kNormal,          ///< 普通类型
    kNormScale,      ///< 等比例缩放
};
```

函数格式

```
int Resize(de::NDArray& src, de::NDArray& dst, Size sz, int type);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArra y 数据结构引用,需要用户构建实体并申请物理连续空间	注意: RGB 格式, 默认为 plane, 即 R...G...B
dst	de::NDArray	输出图像 NDArra y 数据结构引用	注意: RGB 格式, 默认为 plane, 即 R...G...B
Type	ResizeType	Resize 类型	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.3. 图像格式尺寸转换

```

/**@brief image size in pixel
*/
typedef struct{
    int width;    ///< width
    int height;   ///< height
}Size;

```

函数格式

```
int CvtResize(de::NDArray& src, de::NDArray& dst, int cvtCode, Size sz);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	注意：RGB 格式，默认为 plane，即 R...G...B
dst	de::NDArray	输出图像 NDArray 数据结构引用	注意：RGB 格式，默认为 plane，即 R...G...B
cvtCode	Int	色域转换码，参考 CvtColorCode 定义	只支持 kCvtYuvNv12Rgb，kCvtYuvUyvy2Rgb，kCvtYuvYuyv2Rgb，kCvtYuvNv21Rgb
sz	Size	Resize 后的目标分辨率，参考 size 结构定义	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.4. 图像翻转

目前只支持 yuv420nv12/nv21、yuv422 uyvy/yuyv

/**@brief cv 翻转类型

*/

enumFlipType {

kImageFlipVert = 0, ///<垂直翻转

kImageFlipHorz, ///<水平翻转

};

函数格式

```
int Flip(de::NDArray& src, de::NDArray& dst, int flipCode);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出图像 NDArray 数据结构引用	
flipCode	FlipType	参考 FlipType 定义	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.5. 图像旋转

/**@brief cv 旋转类型

*/

enumRotationType {

kRotation_90 = 0, ///<顺时针旋转 90 度

kRotation_180, ///<顺时针旋转 180 度

kRotation_270, ///<顺时针旋转 270 度

kRotation_angle ///<任意角度

};

函数格式

```
int Rotate(de::NDArray& src, de::NDArray& dst, int rotateCode);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出图像 NDArray 数据结构引用	
rotateCode	RotationType	参考 RotationType 定义	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.6. 抠图

```
/* @struct Rect
 * @brief rect
 */
typedef struct {
    int x;
    int y;
    int w;
    int h;
} Rect;
```

函数格式

```
int Crop(de::NDArray& src, de::NDArray& dst, Rect roi);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出图像 NDArray 数据结构引用	
roi	Rect	参考 Rect 定义	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.7. 边缘检测

```
/* @enum EdgeDetectionType
 * @brief EdgeDetectionType
 */
enum EdgeDetectionType {
    kHorizontal = 0,
    kVertical
};
```

函数格式

```
int EdgeDetection(de::NDArray& src, de::NDArray& dst, int type);
```

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出图像 NDArray 数据结构引用	
type	int	参考 EdgeDetectionType 定义	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.8. 高斯模糊

函数格式

```
int GaussianBlur(de::NDArray& src, de::NDArray& dst);
```

说明：目前只支持 5x5 的高斯 blur

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出图像 NDArray 数据结构引用	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.9. 自然指数运算

函数格式

```
int NpExp(de::NDArray& src, de::NDArray& dst);
```

说明：输入输出数据都只支持半精度浮点数

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArry 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出图像 NDArry 数据结构引用	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.10. np.where

函数格式

```
int NpWhere(de::NDArray& src, de::NDArray& vals, de::NDArray& index, int* count, float threshold);
```

说明：输入输出数据都只支持半精度浮点数

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArry 数据结构引用,需要用户构建实体并申请物理连续空间	
vals	de::NDArray	输出结果 NDArry 数据结构引用	
index	de::NDArray	输出索引 NDArry 数据结构引用	
count	int	输出满足条件的个数	
threshold	float	输入的条件阈值	

返回值

部分返回值请参考下方错误码示例

错误码示例

3.4.11

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

TopN 选择

函数格式

```
int TopNSelect(de::NDArray& src, de::NDArray& out_vals, de::NDArray& index, int n, int batch_num, int threshold);
```

说明：输入输出数据都只支持 int16 类型

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入图像 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
out_vals	de::NDArray	输出结果 NDArray 数据结构引用	
index	de::NDArray	输出索引 NDArray 数据结构引用	
n	int	Topn	
batch_num	int	输入的 batch 数	
threshold	int	输入阈值, 函数返回大于阈值的最大的 n 个数	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.12. 向量乘法

函数格式

```
int VectorMultiply(de::NDArray& src0, de::NDArray& src1, de::NDArray&dst, int n, int num0, int num1);
```

说明：输入只支持 int8 类型，输出为 int16 类型，函数不会处理中间溢出的情况

参数说明

参数	类型	说明	取值范围
src0	de::NDArray	输入数据 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
src1	de::NDArray	输入数据 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出索引 NDArray 数据结构引用	
n	int	向量的维度	
num0	int	输入 0 向量个数	
num1	int	输入 1 向量个数	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
3.4.13.	0	执行成功
1	-1	执行失败

特征比对

函数格式

```
int FeCompare(de::NDArray& src, de::NDArray& ref, de::NDArray& dst, de::NDArray& index,
```

```
int fe_num, int n, int src_num, int ref_num, int threshold);
```

说明：输入输出只支持 int8 类型

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入数据 NDArra y 数据结构引用,需要用户构建实体并申请物理连续空间	
ref	de::NDArray	输入数据 NDArra y 数据结构引用,需要用户构建实体并申请物理连续空间	
dst	de::NDArray	输出结果 NDArra y 数据结构引用	
index	de::NDArray	输出索引 NDArra y 数据结构引用	
fe_num	int	单个特征的维度	
n	int	Topn	
src_num	int	输入特征个数	
ref_num	int	特征库特征个数	
threshold	int	特征阈值	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
3.4.14.	0	执行成功
1	-1	执行失败

亮度计算

函数格式

```
int Luma(std::vector<de::NDArray>& src, std::vector<std::vector<deRect>>& rois,
std::vector<std::vector<uint8_t>>& lumas);
```

说明：输入必须为图像类型

参数说明

参数	类型	说明	取值范围
src	Std::vector<de::NDArray>	输入数据 NDArra y 数组数据结构引用,需要用户构建实体并申请物理连续空间	数组最大支持：8
Roi	std::vector<std::vector<deRect>>	输入二维数组引用，元素表示为 src 中每个 image 兴趣区范围，即计算亮度的数据区	数组一维支持：8 数组二维支持：128
Lumas	std::vector<std::vector<uint8_t>>	输出结果二维数组，元素表示为 src 中每个 image 兴趣区的亮度计算值	数组一维支持：8 数组二维支持：128

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.15. 拉普拉斯变换

函数格式

```
int Laplacian(const de::NDArray&nd, std::vector<deRect>rois,
std::vector<de::NDArray>&dsts, int ksize, bool normalize);
```

说明：输入必须为图像类型

参数说明

参数	类型	说明	取值范围
Nd	de::NDArray	输入数据 NDArray 数组数据结构引用, 需要用户构建实体并申请物理连续空间	
Rois	std::vector<deRect>	输入数组引用, 元素表示为 src 中每个 image 兴趣区范围	数组最大支持: 64
dsts	std::vector<de::NDArray>	输出结果数组, 元素表示为 src 中每个 image 兴趣区的计算值	数组最大支持: 64
ksize	Int	目前仅支持配置为 1	
normalize	bool	支持 true or false 配置	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.16. 兴趣区标注

函数格式

```
de::NDArray Range(de::NDArray&nd, int row_start, int row_end, int col_start, int col_end);
```

说明：标注修改某个 NDArry 的兴趣区

参数说明

参数	类型	说明	取值范围
Nd	de::NDArray	输入数据 NDArry 数组数据结构引用,需要用户构建实体并申请物理连续空间	
row_start	Int	Roi 左边 y 坐标值	
row_end	Int	实际 height 计算: row_end- row_start	[row_start, row_end-1]
col_start	Int	Roi 左边 x 坐标值	
col_end	Int	实际 width 计算: col_end- col_start	[col_start, col_end-1]

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.17. 均值方差

函数格式

```
int MeanStddev(std::vector<de::NDArray>& srcs, std::vector<float>& means, std::vector<float>& stddevs);
```

参数说明

参数	类型	说明	取值范围
src	Std::vector<de::NDArray>	输入数据 NDArry 数组数据结构引用,需要用户构建实体并申请物理连续空间	数组最大支持：64
means	std::vector<float>	输入 src 的每个 NDArry 对应的数据均值	数组最大支持：64
stddevs	std::vector<float>	输入 src 的每个 NDArry 对应的数据标准方差	数组最大支持：64

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.4.18. 仿射变换

函数格式

```
int WarpAffine(de::NDArray& src, std::vector<float>& matrix, de::NDArray& dst, int width_out, int height_out);
```

说明：输入必须为图像类型，**只支持灰度和 BGR888_planar 格式**

参数说明

参数	类型	说明	取值范围
src	de::NDArray	输入数据 NDArray 数据结构引用,需要用户构建实体并申请物理连续空间	
matrix	std::vector<float>	变换矩阵	
Dst	de::NDArray	输出数据 NDArray 数据结构引用,不需要用户构建实体	
width_out	Int	目标宽	
height_out	Int	目标高	

返回值

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	0	执行成功
1	-1	执行失败

3.5. kcf 跟踪

函数格式

```
KCFMultiTracker(const deRect roi, const int max_track_num=40, bool
small_scale=false);
```

说明：构造函数

参数说明

函数	参数	类型	说明	取值范围
数 格 式	roi	deRect	跟踪兴趣 Roi，设置后，基于原图上的这个 Roi 内进行跟踪，不在这个 Roi 内的目标将不跟踪。	
	max_track_num	int	最大跟踪目标数，最大 40	
	small_scale	bool	是否用小尺度标识，默认大尺度，用户暂可不设置。	

```
KCFMultiTracker (const int max_track_num=40, bool small_scale=false);
```

说明：构造函数，默认 Roi 为整张原图

参数说明

函数	参数	类型	说明	取值范围
数 格 式	max_track_num	int	最大跟踪目标数，最大 40	
	small_scale	bool	是否用小尺度标识，默认大尺度，用户暂可不设置。	

格式

```
void Init (NDArray image, const std::vector<deRect> bounding_boxes);
```

说明：清除历史跟踪信息，初始化多个跟踪目标

参数说明

函数	参数	类型	说明	取值范围
数 格 式	image	de::NDArray	跟踪图像	
	bounding_boxes	std::vector<deRect>	跟踪目标 roi	

格式

```
void Init (NDArray image, const deRect& bounding_box);
```

说明：清除历史跟踪信息，初始化跟踪目标

参数说明

参数	类型	说明	取值范围
image	de::NDArray	跟踪图像	
bounding_boxes	deRect	跟踪目标 roi	

函数格式

```
void Add(NDArray image,const std::vector<deRect> bounding_boxes);
```

说明：增加新的目标，当前帧先进行 update 操作，然后才允许 add 操作

参数说明

参数	类型	说明	取值范围
image	de::NDArray	跟踪图像	
bounding_boxes	std::vector<deRect>	跟踪目标 roi	

函数格式

```
void Add(NDArray image, deRect& bounding_boxes);
```

说明：增加新的目标，当前帧先进行 update 操作，然后才允许 add 操作

参数说明

参数	类型	说明	取值范围
image	de::NDArray	跟踪图像	
bounding_boxes	deRect	跟踪目标 roi	

函数格式

```
bool Update(NDArray image, std::vector<std::pair<bool, deRect>>& bounding_boxes);
```

说明：跟踪主函数，获取当前图像的跟踪结果

参数说明**返****回****值**

参数	类型	说明	取值范围
image	de::NDArray	跟踪图像	
bounding_boxes	std::vector<std::pair<bool, deRect>>	跟踪结果目标 roi, bool 值为 true 表示目标跟踪成功, false 表示跟踪失败	

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	True	执行成功
1	False	执行失败

函数格式

bool Update(NDArray image);

说明：跟踪主函数，跟踪结果内部保存

参数说明

返回	参数	类型	说明	取值范围
值	image	de::NDArray	跟踪图像	

部分返回值请参考下方错误码示例

错误码示例

编号	错误码	错误码描述
0	True	执行成功
1	False	执行失败

函数格式

void SetTrackerMaxNum(int max_num);

说明：设置最大跟踪目标数，最大 40，该接口不能在跟踪过程中调用，调用该接口，会对所有对象信息进行初始化。

参数说明

参数	类型	说明	取值范围
max_num	int	最大跟踪目标数	0-40

函数格式

void SetTrackRoi(deRect& rect);

说明：设置跟踪兴趣 ROI，超出该区域的目标将不跟踪

参数说明

参数	类型	说明	取值范围
rect	deRect	兴趣 roi	

3.6. Python 接口

DESDK python 接口目前仅用于模型评估过程，只提供同步的模型推理接口。

3.6.1. 函数接口

3.6.1.1. 设备初始化

函数格式

```
device_init(dev_id, host = "192.168.33.101", port = 9200);
```

参数说明

参数	类型	说明	取值范围
dev_id	int	设备 id	0~3
host	str	设备 ip	
port	int	设备端口	

返回值

与设备连接的 rpc 对象

3.6.2. Python 类

3.6.2.1. NNDeploy 类

NNDeploy 类			
说明： nn 部署类			
nn.NNDeploy	NNDeploy	NNDeploy(remote)	
		构造函数	
		输入	remote: device_init 函数返回的对象
		返回	NNDeploy 对象
		注意	
	load_host	加载模型 load_host(self, net_bin_host, model_bin_host, batch_max = 8, resize_type = 0, tensor_num_per_batch = 1, profile = 1)	
		输入	net_bin_host: 模型文件 netbin 路径

			model_bin_host: 模型文件 modelbin 路径 batch_max: 最大 batch 数 resize_type: 缩放类型 0: 正常缩放 1: 等比例缩放 tensor_num_per_batch: 模型一个 batch 有几个输入 profile: 是否打开 dspprofile 功能
		返回	nn.Module 对象
		注意	
	to_ndarray	to_ndarray(self, data, img_format, shapes) 将图片转化为 ndarray	
		输入	data: 输入图片, 可以是离线数据, 也可以是 jpeg 文件
		返回	Ndarray 对象
		注意	
	data_to_ndarray	data_to_ndarray(self,data,type_code,bits,shape_code,shapes) 将纯数据转为 ndarray	
		输入	data: 输入数据 type_code: 如下定义 class TypeCode(Enum): Int = 0 UInt = 1 Float = 2 bits: 8, 16, 32 shape_code: 如下定义 class ShapeCode(Enum): NDHW = 0 DHW = 1 HW = 2 W = 3 shape: 数据维度 list, 如[3,224,224]
		返回	ndarray
		注意	
	un_load	un_load(self, module) 卸载模型	
		输入	module: 通过 load_host 返回的 NNModule 对象
		返回	无
		注意	

3.6.2.2. NNModule 类

NNModule 类

说明： nn 部署类			
nn.NNModule	set_input	set_input(self, *args)	
		设置输入	
		输入	*args: 模型输入的 nd 对象，支持多个
		返回	无
		注意	
	run	run(self)	
		执行模型	
		输入	无
		返回	无
		注意	
	get_outputs	get_outputs(self)	
		获取输出	
		输入	无
		返回	模型输出，返回 numpy 对象的 list
		注意	
	profile	profile(self)	
		获取 profile 信息	
		输入	无
		输出	Profile 信息
		注意	
	display_profile_info	display_profile_info(self, model_time)	
		打印 profile 信息	
		输入	profile(self)返回的 profile 信息
		输出	无
		注意	
	batch_num	batch_num(self, batch_num)	
		设置 batch 数目	
		输入	batch_num: batch 数目
		输出	无
		注意	
	dsp_split	dsp_split(self, dsp_split)	
		设置是否两个 dsp 调度	
		输入	dsp_split: 0: 单个 dsp 调度 1: 两个 dsp 调度
		输出	无
		注意	

3.7. 设备管理接口

3.7.1. dcmi 模块初始化

函数格式

DE_API ErrorCodeInit(unsigned int nums);

参数说明

参数	类型	说明	取值范围
nums	unsigned int	使用 dcmi 升级模块前必须调用	Nums 芯片数目

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_UNKNOWN	失败

3.7.2. dcmi 模块去初始化

函数格式

DE_API ErrorCodeDeInit(); 未实现

参数说明

参数	类型	说明	取值范围
无			

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1		
2		

3.7.3. 获取支持的最大 DeepEye 芯片个数

函数格式

DE_API int MaxDevNum();

参数说明

参数	类型	说明	取值范围
无			

返回值

Desdk 所能支持芯片的个数

错误码示例

编号	错误码	错误码描述
1	NULL	
2		

3.7.4. 设置事件回调函数

函数格式

DE_API ErrorCodeSetEventCallback(DevId dev_id, EventCallback cb);

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备的 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
cb	EventCallback	事件回调函数	

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_PARAM_INVALID	参数无效
3	DE_UNKNOWN	其他错误

3.7.5. 执行生产测试

函数格式

```
DE_API ErrorCodeAmtTest(DevId dev_id, int time_out, std::vector<AmtErrorCode>& result);
```

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
time_out	int	生产测试超时, 单位 ms	大于 3000
result	std::vector<AmtErrorCode>	设备生产测试结果	

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_NO_DEVICE_FOUND	测试的设备不存在
3	DE_LOAD_DLL_FAIL	测试函数未注册
4	DE_TIMEOUT	生产测试超时
5	DE_PART_SUCCESS	部分测试成功

3.7.6. DeepEye 芯片版本升级

函数格式

```
DE_API ErrorCodeUpgrade(const std::vector<VersionCfg>& config,  
std::vector<ErrorCode>& result);
```

参数说明

参数	类型	说明	取值范围
config	const std::vector<VersionCfg>	设备 ID 执行升级的固件	不能超过实际设备 ID 和 desdk 所支持的最大设备数
result	std::vector<ErrorCode>	执行升级结果返回	

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_UNKNOWN	失败

3.7.7. DeepEye 芯片版本下载

函数格式

```
DE_API ErrorCodeDownload(const std::vector<VersionCfg>& config,  
std::vector<ErrorCode>& result);
```

参数说明

参数	类型	说明	取值范围
config	(const std::vector<VersionCfg>	设备 ID 下载版本固件路径	不能超过实际设备 ID 和 desdk 所支持的最大设备数
result	std::vector<ErrorCode>	执行返回结果	

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_UNKNOWN	失败

3.7.8. 获取设备日志信息（暂未实现）

函数格式

DE_API ErrorCodeGetDevLog(DevId dev_id, std::vector<std::string>& log);

参数说明

参数	类型	说明	取值范围
无			

返回值

错误码示例

编号	错误码	错误码描述
1		
2		

3.7.9. 获取设备状态

函数格式

```
DE_API ErrorCodeGetDevStatus(DevId dev_id, std::vector<DevStatus>& status);
```

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
status	std::vector<DevStatus>	设备在线或者离线状态	

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_PARAM_INVALID	参数无效

3.7.10. 启动主控发送心跳

函数格式

```
DE_API ErrorCode StartHostHeartBeat(DevId dev_id, HeartbeatConfigInfo*  
cfg, bool device_reboot = true);
```

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
cfg	HeartbeatConfigInfo*	心跳配置指针	非空
device_reboot	bool	超时后设备是否自动重启	默认为 true, 重启

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_PARAM_INVALID	参数无效

3.7.11. 停止主控发送心跳

函数格式

```
DE_API ErrorCode StopHostHeartBeat(DevId dev_id, bool device_ok = true);
```

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
device_ok	bool	设备是否还正常, 设备正常会通知设备主控心跳停止了; 设备不正常就不通知了, 因此可能会触发芯片侧接收主控心跳超时。	默认为 true, 设备正常

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_PARAM_INVALID	参数无效

3.7.12. 启动设备硬件看门狗

函数格式

DE_API ErrorCode StartDeviceWdt(DevId dev_id, WatchdogConfigInfo* cfg);

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
cfg	WatchdogConfigInfo*	看门狗配置指针	非空

返回值

DE_OK: 成功, 其他: 失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_PARAM_INVALID	参数无效

3.7.13. 停止设备硬件看门狗

函数格式

DE_API ErrorCode StopDeviceWdt(DevId dev_id, bool device_ok = true);

参数说明

参数	类型	说明	取值范围
dev_id	DevId	设备 ID	不能超过实际设备 ID 和 desdk 所支持的最大设备数
device_ok	bool	设备是否还正常，设备正常会通知设备停止看门狗功能了；设备不正常就不通知了，因此芯片侧可能会看门狗超时重启。	默认为 true，设备正常

返回值

DE_OK: 成功，其他：失败

错误码示例

编号	错误码	错误码描述
1	DE_OK	成功
2	DE_PARAM_INVALID	参数无效

3.8. 资源使用情况查询接口

支持查询芯片侧以下资源的使用情况：

1. NNP 调度率
2. DDR 带宽
3. FastMalloc 内存总大小、已使用大小、空闲大小
4. 总 CPU 占用率

3.8.1. 数据类型定义

资源使用情况数据结构定义如下：

```
///  
//< NNP 调度率  
struct NNPSchedRate  
{  
    //单位：百分比  
    double period_sched_rate; //最近一个统计周期内的调度率  
    double tot_sched_rate;    //总调度率  
};
```

```
///  
//< DDR 带宽  
struct BandwidthItem  
{  
    //单位: GB/s  
    float r_bw;  
    float w_bw;  
    float rw_bw;  
  
    //对齐到 8 字节整数倍  
    float reserved;  
};  
struct DDRBandwidth  
{  
    struct BandwidthItem total;  
    struct BandwidthItem port[5];  
};  
  
///  
//< FastMalloc 内存使用  
struct FastMallocMem  
{  
    //单位: MB  
    float total_size; //总大小  
    float used_size;  //已使用大小  
    float free_size;  //空闲大小  
  
    //对齐到 8 字节整数倍  
    float reserved;  
};  
  
///  
//< CPU 使用率  
struct CPUUsage  
{  
    //单位: 百分比, 例如 12.3 表示 CPU 使用率为 12.3%  
    double cpu_percent; //最近一个统计周期内的 cpu 使用率  
};  
  
struct ResourceUsageInfo  
{  
    static const int kNnpNumber = 4;  
  
    NNPSchedRate nnp_shced_rate[kNnpNumber];  
    DDRBandwidth ddr_bandwidth;  
    FastMallocMem fast_malloc_mem;
```

```
CPUUsage cpu_usage;
};
```

3.8.2. 启动资源使用统计

函数格式

```
void StartDevResProfile(int dev_id)
```

参数说明

参数	类型	说明	取值范围
dev_id	int	要启动资源使用统计功能的设备编号	Nums 芯片数目

返回值

无

函数说明

该函数可以调用多次，芯片侧会判断资源统计功能是否已经开启，如已开启会忽略多余的调用。

3.8.3. 获取资源使用情况

函数格式

```
ResourceUsageInfo GetDevResUseInfo(int dev_id);
```

参数说明

参数	类型	说明	取值范围
dev_id	int	要获取资源使用情况的设备编号	Nums 芯片数目

返回值

资源使用情况结构体

3.8.4. 停止资源使用统计

函数格式

```
void StopDevResProfile(int dev_id);
```

参数说明

参数	类型	说明	取值范围
dev_id	int	要停止资源使用统计功能的设备编号	Nums 芯片数目

返回值

无

函数说明

该函数可以调用多次，芯片侧会忽略多余的调用。

4. FAQ