密级状态：绝密(　)　秘密(　)　内部(　)　公开(√)

RKNN C API 参考手册

（图形计算平台中心）

文件状态： [ ] 正在修改 [√] 正式发布	当前版本：	v2.0.0-beta0
	作　者：	HPC/NPU 团队
	完成日期：	2024-03-18
	审　核：	熊伟
	完成日期：	2024-03-18

瑞芯微电子股份有限公司

Rockchips Semiconductor Co . , Ltd

更新记录

版本	修改人	修改日期	修改说明	核定人
v0.6.0	HPC 团队	2021-03-01	初始版本	熊伟
v0.7.0	HPC 团队	2021-04-22	删除输入通道转换流程说明	熊伟
v1.0.0	HPC 团队	2021-04-30	正式发布版本	熊伟
v1.1.0	HPC 团队	2021-08-13	1. 增加 rknn_tensor_mem_flags 标志 2. 增加输入/输出 tensor 原生属性的查询命令 3. 增加 NC1HWC2 的内存布局	熊伟
v1.2.0b1	NPU 团队	2021-12-04	1. 增加 RK3588/RK3588s 平台说明 2. 增加 rknn_set_core_mask 接口 3. 增加 rknn_dup_context 接口 4. 增加输入输出 API 详细说明	熊伟
v1.2.0	HPC 团队	2022-01-14	1. 增加关键字说明 2. 增加 NPU SDK 目录和编译说明 3. 增加调试方法章节 4. 增加 NATIVE_LAYOUT 中 C2 取值说明	熊伟
v1.3.0	NPU/HPC 团队	2022-05-13	1. 修复命名 destroy 变为 destroy 2. 增加 RV1106/RV1103 的使用说明 3. 增加 NATIVE_LAYOUT 的细节说明 4. 增加 C API 硬件平台支持说明 5. 增加 NPU 版本、利用率查询以及 NPU 电源手动开关的指令	熊伟

版本	修改人	修改日期	修改说明	核定人
v1.4.0	NPU/HPC 团队	2022-08-31	1. RV1106/RV1103 增加 rknn_create_mem_from_phys/ rknn_create_mem_from_fd/ rknn_set_weight_mem/ rknn_set_internal_mem 接口支持 2. 新增权重共享的功能 3. RK3588 新增 sram 功能支持 4. RK3588 新增单 batch 多核支持 5. NPU 新版本驱动增加查询频率、电压、设置延时关闭时间等功能	熊伟
v1.4.2	HPC 团队	2023-02-13	1. 增加 RK3562 的使用说明 2. 增加 rknn_init 接口中 RKNN_FLAG_COLLECT_MODEL_INFO_ONLY 标志说明	熊伟
v1.5.0	HPC 团队	2023-05-22	1. 增加动态形状输入 API 使用说明和相关数据结构说明 2. 增加 Matmul API 使用说明	熊伟
v1.5.2	HPC 团队	2023-08-22	1. 增加 rknn_init 接口中 RKNN_FLAG_EXECUTE_FALLBACK_PRIOR_DEVICE_GPU 和 RKNN_FLAG_INTERNAL_ALLOC_OUTSIDE 标志说明 2. 移除旧动态输入形状功能的 rknn_set_input_shape 接口说明，新增 rknn_set_input_shapes 接口说明	熊伟
v1.6.0	HPC 团队	2023-11-28	1. 增加 rknn_init 接口中 RKNN_FLAG_ENABLE_SRAM 、RKNN_FLAG_SHARE_SRAM 以及 RKNN_FLAG_DISABLE_PROC_HIGH_PRIORITY 标志说明 2. 增加 rknn_set_batch_core_num 接口说明 3. 增加 rknn_mem_sync 接口说明	熊伟

版本	修改人	修改日期	修改说明	核定人
v2.0.0-beta0	HPC 团队	2024-03-18	1. 增加 rknn_create_mem2 接口说明 2. 修改 rknn_matmul_info 和 rknn_matmul_type 结构体 3. 增加 rknn_quant_params 和 rknn_matmul_shape 结构体 4. 增加 rknn_matmul_set_quant_params, rknn_matmul_get_quant_params, rknn_matmul_create_dyn_shape, rknn_matmul_set_dynamic_shape 接口	熊伟

目 录

1 概述

RKNN C API 是 RKNPU Runtime（运行时库）的 C 语言接口。通过使用 RKNN C API，开发者可以利用 NPU 的计算能力完成高效的 RKNN 模型推理或矩阵乘法计算任务。本文对 RKNN C API 的各个函数、数据结构以及返回码定义进行说明。

2 硬件平台

本文档适用如下硬件平台：

- RK3576 系列
- RK3566 系列
- RK3568 系列
- RK3588 系列
- RV1103
- RV1106
- RK3562

3 RKNPU 编译说明

开发者编译应用时要包含接口函数所在的头文件，并且根据使用的硬件平台和系统类型，链接相应 RKNPU 运行时库。以下对 RKNN C API 头文件和运行时库文件进行说明。

3.1 RKNN C API 头文件

根据不同的功能特点，RKNN C API 的接口分为三个部分，各个部分函数、数据结构定义和头文件对应关系如下：

1. “rknn_api.h”定义部署 RKNN 模型的基础接口和数据结构。
2. “rknn_matmul_api.h”定义矩阵乘法接口。
3. “rknn_custom_op.h”定义用户自定义算子接口。

3.2 Linux 平台 RKNPU 运行时库

1. 对于 RK3576、RK3566 系列、RK3568 系列、RK3588 系列、RK3562 硬件平台，RKNPU 运行时库文件为<sdk_path>/rknpu2/runtime 目录下的 librknnrt.so，其中<sdk_path>是瑞芯微 NPU 软件开发包的路径。
2. 对于 RV1106、RV1103 硬件平台，RKNPU 运行时库文件为<sdk_path>/rknpu2/runtime 目录下的 librknnmrt.so。

3.3 Android 平台 RKNPU 运行时库

Android 平台有两种方式来调用 RKNN C API：

1） 应用直接链接 librknnrt.so。

2） 应用链接 Android 平台 HIDL 实现的 librknn_api_android.so。

对于需要通过 CTS/VTS 测试的 Android 设备需要使用基于 Android 平台 HIDL 实现的 RKNN API（**librknn_api_android.so 不包含矩阵乘法和自定义算子功能**）。如果不需要通过 CTS/VTS

测试的设备建议直接使用 librknnrt.so（包含矩阵乘法和自定义算子功能），对各个接口调用流程的链路更短，可以提供更好的性能。

对于使用 Android HIDL 实现的 RKNN API 的代码位于 RK3576/RK3588/RK3562/RK3566/RK3568 Android 系统 SDK 的 vendor/rockchip/hardware/interfaces/neuralnetworks 目录下。当完成 Android 系统编译后，将会生成 RKNPU 相关的一系列库文件（对于应用开发只需要链接使用 librknn_api_android.so 即可），如下所示：

```
/vendor/lib/librknn_api_android.so
/vendor/lib/librknnhal_bridge.rockchip.so
/vendor/lib64/librknn_api_android.so
/vendor/lib64/librknnhal_bridge.rockchip.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0-adapter-helper.so
/vendor/lib64/hw/rockchip.hardware.neuralnetworks@1.0-impl.so
/vendor/bin/hw/rockchip.hardware.neuralnetworks@1.0-service
```

也可以使用如下命令单独重新编译生成以上的库文件：

```
mmm vendor/rockchip/hardware/interfaces/neuralnetworks/ -j8
```

4 RKNN C API 说明

4.1 各个硬件平台的 C API 支持情况

由于不同的芯片平台的硬件特性不同，RKNN C API 的接口以及接口参数的支持情况也不同。各个硬件平台的 RKNN C API 接口支持情况如表 4-1 所示：

表 4-1 各个硬件平台的 RKNN C API 接口支持情况

	RKNN C API	RK3562/ RK3566/ RK3568	RK3588/ RK3576	RV1106/RV1103
1	rknn_init	√	√	√
2	rknn_set_core_mask	×	√	×
3	rknn_dup_context	√	√	×
4	rknn_destroy	√	√	√
5	rknn_query	√	√	√
6	rknn_inputs_set	√	√	×
7	rknn_run	√	√	√
8	rknn_wait	×	×	×
9	rknn_outputs_get	√	√	×
10	rknn_outputs_release	√	√	√
11	rknn_create_mem_from_mb_blk	×	×	×
12	rknn_create_mem_from_phys	√	√	√
13	rknn_create_mem_from_fd	√	√	√
14	rknn_create_mem	√	√	√
15	rknn_destroy_mem	√	√	√
16	rknn_set_weight_mem	√	√	√
17	rknn_set_internal_mem	√	√	√
18	rknn_set_io_mem	√	√	√
19	rknn_set_input_shapes	√	√	×
20	rknn_mem_sync	√	√	√
21	rknn_matmul_create	√	√	×
22	rknn_matmul_set_io_mem	√	√	×
23	rknn_matmul_set_core_mask	×	√	×
24	rknn_matmul_run	√	√	×
25	rknn_matmul_destroy	√	√	×
26	rknn_register_custom_ops	√	√	×
27	rknn_custom_op_get_op_attr	√	√	×
28	rknn_set_batch_core_num	×	√	×
29	rknn_matmul_set_quant_params	√	√	×
30	rknn_matmul_get_quant_params	√	√	×
31	rknn_matmul_create_dyn_shape	√	√	×
32	rknn_matmul_set_dynamic_shape	√	√	×
33	rknn_B_normal_layout_to_native_layout	√	√	×

各个硬件平台使用 rknn_query 函数支持的查询参数如表 4-2 所示：

表 4-2 各个硬件平台 rknn_query 函数支持的查询参数

	rknn_query 参数	RK3562/RK3566/ RK3568	RK3576/ RK3588	RV1106/ RV1103
1	RKNN_QUERY_IN_OUT_NUM	√	√	√
2	RKNN_QUERY_INPUT_ATTR	√	√	√
3	RKNN_QUERY_OUTPUT_ATTR	√	√	√
4	RKNN_QUERY_PERF_DETAIL	√	√	×
5	RKNN_QUERY_PERF_RUN	√	√	×
6	RKNN_QUERY_SDK_VERSION	√	√	√
7	RKNN_QUERY_MEM_SIZE	√	√	√
8	RKNN_QUERY_CUSTOM_STRING	√	√	√
9	RKNN_QUERY_NATIVE_INPUT_ATTR	√	√	√
10	RKNN_QUERY_NATIVE_OUTPUT_ATTR	√	√	√
11	RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR	√	√	√
12	RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR	√	√	√
13	RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR	√	√	√
14	RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR	√	√	√
15	RKNN_QUERY_INPUT_DYNAMIC_RANGE	√	√	×
16	RKNN_QUERY_CURRENT_INPUT_ATTR	√	√	×
17	RKNN_QUERY_CURRENT_OUTPUT_ATTR	√	√	×
18	RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR	√	√	×
19	RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR	√	√	×

4.2 基础数据结构定义

4.2.1 rknn_sdk_version

结构体 rknn_sdk_version 用来表示 RKNN SDK 的版本信息，结构体的定义如下：

成员变量	数据类型	含义
api_version	char[]	SDK 的版本信息。
drv_version	char[]	SDK 所基于的驱动版本信息。

4.2.2 rknn_input_output_num

结构体 rknn_input_output_num 表示输入输出 tensor 个数，其结构体成员变量如下表所示：

成员变量	数据类型	含义
n_input	uint32_t	输入 tensor 个数。
n_output	uint32_t	输出 tensor 个数。

4.2.3 rknn_input_range

结构体 rknn_input_range 表示一个输入的支持形状列表信息。它包含了输入的索引、支持的形状个数、数据布局格式、名称以及形状列表，具体的结构体的定义如下表所示：

成员变量	数据类型	含义
index	uint32_t	表示该形状对应输入的索引位置。
shape_number	uint32_t	表示 RKNN 模型支持的输入形状个数。
fmt	rknn_tensor_format	表示形状对应的数据布局格式。
name	char[]	表示输入的名称。
dyn_range	uint32_t[][]	表示输入形状列表，它是包含多个形状数组的二维数组，形状优先存储。
n_dims	uint32_t	表示每个形状数组的有效维度个数。

4.2.4 rknn_tensor_attr

结构体 rknn_tensor_attr 表示模型的 tensor 的属性，结构体的定义如下表所示：

成员变量	数据类型	含义
index	uint32_t	表示输入输出 tensor 的索引位置。
n_dims	uint32_t	Tensor 维度个数。
dims	uint32_t[]	Tensor 各维度值。
name	char[]	Tensor 名称。
n_elems	uint32_t	Tensor 数据元素个数。
size	uint32_t	Tensor 数据所占内存大小。
fmt	rknn_tensor_format	Tensor 维度的格式，有以下格式： **RKNN_TENSOR_NCHW** **RKNN_TENSOR_NHWC** **RKNN_TENSOR_NC1HWC2** **RKNN_TENSOR_UNDEFINED**
type	rknn_tensor_type	Tensor 数据类型，有以下数据类型： **RKNN_TENSOR_FLOAT32** **RKNN_TENSOR_FLOAT16** **RKNN_TENSOR_INT8** **RKNN_TENSOR_UINT8** **RKNN_TENSOR_INT16** **RKNN_TENSOR_UINT16** **RKNN_TENSOR_INT32** **RKNN_TENSOR_INT64** **RKNN_TENSOR_BOOL**
qnt_type	rknn_tensor_qnt_type	Tensor 量化类型，有以下的量化类型： **RKNN_TENSOR_QNT_NONE**：未量化； **RKNN_TENSOR_QNT_DFP**：动态定点量化； **RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC**：非对称量化。
fl	int8_t	RKNN_TENSOR_QNT_DFP 量化类型的参数。
scale	float	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC 量化类型的参数。

w_stride	uint32_t	实际存储一行图像数据的像素数目，等于一行的有效数据像素数目 + 为硬件快速跨越到下一行而补齐的一些无效像素数目，单位是像素。
size_with_stride	uint32_t	实际存储图像数据所占的存储空间的大小（包括了补齐的无效像素的存储空间大小）。
pass_through	uint8_t	0 表示未转换的数据，1 表示转换后的数据，转换包括归一化和量化。
h_stride	uint32_t	仅用于多 batch 输入场景，且**该值由用户设置**。目的是 NPU 正确地读取每 batch 数据的起始地址，它等于原始模型的输入高度+跨越下一列而补齐的无效像素数目。如果设置成 0，表示与原始模型输入高度一致，单位是像素。

4.2.5 rknn_perf_detail

结构体 rknn_perf_detail 表示模型的性能详情，结构体的定义如下表所示（RV1106/RV1103 暂不支持）：

成员变量	数据类型	含义
perf_data	char*	性能详情包含网络每层运行时间，能够直接打印出来查看。
data_len	uint64_t	存放性能详情的字符串数组的长度。

4.2.6 rknn_perf_run

结构体 rknn_perf_run 表示模型的总体性能，结构体的定义如下表所示（RV1106/RV1103 暂不支持）：

成员变量	数据类型	含义
run_duration	int64_t	网络总体运行（不包含设置输入/输出）时间，单位是微秒。

4.2.7 rknn_mem_size

结构体 rknn_mem_size 表示初始化模型时的内存分配情况，结构体的定义如下表所示：

成员变量	数据类型	含义
total_weight_size	uint32_t	模型的权重占用的内存大小。
total_internal_size	uint32_t	模型的中间 tensor 占用的内存大小。
total_dma_allocated_size	uint64_t	模型申请的所有 dma 内存之和。
total_sram_size	uint32_t	只针对 RK3588 有效，为 NPU 预留的系统 SRAM 大小（具体使用方式参考《RK3588_NPU_SRAM_usage.md》）。
free_sram_size	uint32_t	只针对 RK3588 有效，当前可用的空闲 SRAM 大小（具体使用方式参考《RK3588_NPU_SRAM_usage.md》）。
reserved[12]	uint32_t	预留。

4.2.8 rknn_tensor_mem

结构体 rknn_tensor_mem 表示 tensor 的内存信息。结构体的定义如下表所示：

成员变量	数据类型	含义
virt_addr	void*	该 tensor 的虚拟地址。
phys_addr	uint64_t	该 tensor 的物理地址。
fd	int32_t	该 tensor 的文件描述符。
offset	int32_t	相较于文件描述符和虚拟地址的偏移量。
size	uint32_t	该 tensor 占用的内存大小。
flags	uint32_t	rknn_tensor_mem 的标志位，有以下标志： **RKNN_TENSOR_MEMORY_FALGS_ALLOC_INSIDE:** 表明 rknn_tensor_mem 结构体由运行时创建； **RKNN_TENSOR_MEMORY_FLAGS_FROM_FD:** 表明 rknn_tensor_mem 结构体由 fd 构造； **RKNN_TENSOR_MEMORY_FLAGS_FROM_PHYS:** 表明 rknn_tensor_mem 结构体由物理地址构造； **用户不用关注该标志。**
priv_data	void*	内存的私有数据。

4.2.9 rknn_input

结构体 rknn_input 表示模型的一个数据输入，用来作为参数传入给 rknn_inputs_set 函数。结构体的定义如下表所示：

成员变量	数据类型	含义
index	uint32_t	该输入的索引位置。
buf	void*	输入数据的指针。
size	uint32_t	输入数据所占内存大小。
pass_through	uint8_t	设置为 1 时会将 buf 存放的输入数据直接设置给模型的输入节点，不做任何预处理。
type	rknn_tensor_type	输入数据的类型。
fmt	rknn_tensor_format	输入数据的格式。

4.2.10 rknn_output

结构体 rknn_output 表示模型的一个数据输出，用来作为参数传入给 rknn_outputs_get 函数，在函数执行后，结构体对象将会被赋值。结构体的定义如下表所示：

成员变量	数据类型	含义
want_float	uint8_t	标识是否需要将输出数据转为 float 类型输出，该字段由用户设置。
is_prealloc	uint8_t	标识存放输出数据是否是预分配，该字段由用户设置。
index	uint32_t	该输出的索引位置，该字段由用户设置。
buf	void*	输出数据的指针，该字段由接口返回。
size	uint32_t	输出数据所占内存大小，该字段由接口返回。

4.2.11 rknn_init_extend

结构体 rknn_init_extend 表示初始化模型时的扩展信息。结构体的定义如下表所示（RV1106/RV1103 暂不支持）：

成员变量	数据类型	含义
ctx	rknn_context	已初始化的 rknn_context 对象。
real_model_offset	int32_t	真正 rknn 模型在文件中的偏移，只有以文件路径为参数初始化时才生效。
real_model_size	uint32_t	真正 rknn 模型在文件中的大小，只有以文件路径为参数初始化时才生效。
reserved	uint8_t[120]	预留数据位。

4.2.12 rknn_run_extend

结构体 rknn_run_extend 表示模型推理时的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

成员变量	数据类型	含义
frame_id	uint64_t	表示当前推理的帧序号。
non_block	int32_t	0 表示阻塞模式，1 表示非阻塞模式，非阻塞即 rknn_run 调用直接返回。
timeout_ms	int32_t	推理超时的上限，单位毫秒。
fence_fd	int32_t	用于非阻塞执行推理，**暂不支持**。

4.2.13 rknn_output_extend

结构体 rknn_output_extend 表示获取输出的扩展信息，**目前暂不支持使用**。结构体的定义如下表所示：

成员变量	数据类型	含义
frame_id	int32_t	输出结果的帧序号。

4.2.14 rknn_custom_string

结构体 rknn_custom_string 表示转换 RKNN 模型时，用户设置的自定义字符串，结构体的定义如下表所示：

成员变量	数据类型	含义
string	char[]	用户自定义字符串。

4.3 基础 API 说明

4.3.1 rknn_init

rknn_init 初始化函数功能为创建 rknn_context 对象、加载 RKNN 模型以及根据 flag 和 rknn_init_extend 结构体执行特定的初始化行为。

API	rknn_init
功能	初始化 rknn 上下文。
参数	rknn_context *context：rknn_context 指针。
	void *model：RKNN 模型的二进制数据或者 RKNN 模型路径。当参数 size 大于 0 时，model 表示二进制数据；当参数 size 等于 0 时，model 表示 RKNN 模型路径。
	uint32_t size：当 model 是二进制数据，表示模型大小，当 model 是路径，则设置为 0。
	uint32_t flag：初始化标志，默认初始化行为需要设置为 0。
	rknn_init_extend：特定初始化时的扩展信息。没有使用，传入 NULL 即可。如果需要共享模型 weight 内存，则需要传入另个模型 rknn_context 指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
```

各个初始化标志说明如下：

RKNN_FLAG_COLLECT_PERF_MASK：用于运行时查询网络各层时间；

RKNN_FLAG_MEM_ALLOC_OUTSIDE：用于表示模型输入、输出、权重、中间 tensor 内存全部由用户分配，它主要有两方面的作用：

1）所有内存均是用户自行分配，便于对整个系统内存进行统筹安排。

2）用于内存复用，特别是针对 RV1103/RV1106 这种内存极为紧张的情况。

假设有模型 A、B 两个模型，这两个模型在设计上串行运行的，那么这两个模型的中间 tensor 的内存就可以复用。示例代码如下：

```
rknn_context ctx_a, ctx_b;

rknn_init(&ctx_a, model_path_a, 0, RKNN_FLAG_MEM_ALLOC_OUTSIDE, NULL);
rknn_query(ctx_a, RKNN_QUERY_MEM_SIZE, &mem_size_a, sizeof(mem_size_a));

rknn_init(&ctx_b, model_path_b, 0, RKNN_FLAG_MEM_ALLOC_OUTSIDE, NULL);
rknn_query(ctx_b, RKNN_QUERY_MEM_SIZE, &mem_size_b, sizeof(mem_size_b));

max_internal_size = MAX(mem_size_a.total_internal_size, mem_size_b.total_internal_size);
internal_mem_max = rknn_create_mem(ctx_a, max_internal_size);

internal_mem_a = rknn_create_mem_from_fd(ctx_a, internal_mem_max->fd,
      internal_mem_max->virt_addr, mem_size_a.total_internal_size, 0);
rknn_set_internal_mem(ctx_a, internal_mem_a);

internal_mem_b = rknn_create_mem_from_fd(ctx_b, internal_mem_max->fd,
      internal_mem_max->virt_addr, mem_size_b.total_internal_size, 0);
rknn_set_internal_mem(ctx_b, internal_mem_b);
```

RKNN_FLAG_SHARE_WEIGHT_MEM：用于共享另一个模型的 weight 权重。主要用于模拟不定长度模型输入（RKNPU 运行时库版本大于等于 1.5.0 后该功能被动态 shape 功能替代）。比如对于某些语音模型，输入长度不定，但由于 NPU 无法支持不定长输入，因此需要生成几个不同分辨率的 RKNN 模，其中，只有一个 RKNN 模型的保留完整权重，其他 RKNN 模型不带权重。在初始化不带权重 RKNN 模型时，使用该标志能让当前上下文共享完整 RKNN 模型的权重。假设需要分辨率 A、B 两个模型，则使用流程如下：

1）使用 RKNN-Toolkit2 生成分辨率 A 的模型。

2）使用 RKNN-Toolkit2 生成不带权重的分辨率 B 的模型，rknn.config()中，remove_weight 要设置成 True，主要目的是减少模型 B 的大小。

3）在板子上，正常初始化模型 A。

4）通过 RKNN_FLAG_SHARE_WEIGHT_MEM 的 flags 初始化模型 B。

5）其他按照原来的方式使用。板端参考代码如下：

```
rknn_context ctx_a, ctx_b;
rknn_init(&ctx_a, model_path_a, 0, 0, NULL);

rknn_init_extend extend;
extend.ctx = ctx_a;
rknn_init(&ctx_b, model_path_b, 0, RKNN_FLAG_SHARE_WEIGHT_MEM, &extend);
```

RKNN_FLAG_COLLECT_MODEL_INFO_ONLY：用于初始化一个空上下文，仅用于调用

rknn_query 接口查询模型 weight 内存总大小和中间 tensor 总大小，无法进行推理；

RKNN_FLAG_INTERNAL_ALLOC_OUTSIDE: 表示模型中间 tensor 由用户分配，常用于用户自行管理和复用多个模型之间的中间 tensor 内存；

RKNN_FLAG_EXECUTE_FALLBACK_PRIOR_DEVICE_GPU: 表示所有 NPU 不支持的层优先选择运行在 GPU 上，但并不保证运行在 GPU 上，实际运行的后端设备取决于运行时对该算子的支持情况；

RKNN_FLAG_ENABLE_SRAM: 表示中间 tensor 内存尽可能分配在 SRAM 上；

RKNN_FLAG_SHARE_SRAM: 用于当前上下文尝试共享另一个上下文的 SRAM 内存地址空间，要求当前上下文初始化时必须同时启用 RKNN_FLAG_ENABLE_SRAM 标志；

RKNN_FLAG_DISABLE_PROC_HIGH_PRIORITY: 表示当前上下文使用默认进程优先级。不设置该标志，进程 nice 值是-19；

RKNN_FLAG_DISABLE_FLUSH_INPUT_MEM_CACHE: runtime 内部不主动刷新输入 tensor 缓存，用户必须确保输入 tensor 在调用 rknn_run 之前已刷新缓存。主要用于当输入数据没有 CPU 访问时，减少 runtime 内部刷 cache 的耗时。

RKNN_FLAG_DISABLE_FLUSH_OUTPUT_MEM_CACHE： runtime 不主动清除输出 tensor 缓存。此时用户不能直接访问 output_mem->virt_addr，这会导致缓存一致性问题。 如果用户想使用 output_mem->virt_addr ， 必须使用 rknn_mem_sync (ctx, mem, RKNN_MEMORY_SYNC_FROM_DEVICE) 来刷新缓存。该标志一般在 NPU 的输出数据不被 CPU 访问时使用，比如输出数据由 GPU 或 RGA 访问以减少刷新缓存所需的时间。

4.3.2 rknn_set_core_mask

rknn_set_core_mask 函数指定工作的 NPU 核心，该函数仅支持 RK3576/RK3588 平台，在 RK3562/RK3566/RK3568/RV1106/RV1103 平台上设置会返回错误。

API	rknn_set_core_mask
功能	设置运行的 NPU 核心。
参数	rknn_context context：rknn_context 对象。
	rknn_core_mask core_mask：NPU 核心的枚举类型，目前有如下方式配置：

	RKNN_NPU_CORE_AUTO：表示自动调度模型，自动运行在当前空闲的 NPU 核上； **RKNN_NPU_CORE_0**：表示运行在 NPU0 核上； **RKNN_NPU_CORE_1**：表示运行在 NPU1 核上； **RKNN_NPU_CORE_2**：表示运行在 NPU2 核上； **RKNN_NPU_CORE_0_1**：表示同时工作在 NPU0、NPU1 核上； **RKNN_NPU_CORE_0_1_2**：表示同时工作在 NPU0、NPU1、NPU2 核上。 **RKNN_NPU_CORE_ALL**：表示根据使用所有的 NPU 核心；
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_context ctx;
rknn_core_mask core_mask = RKNN_NPU_CORE_0;
int ret = rknn_set_core_mask(ctx, core_mask);
```

在 RKNN_NPU_CORE_0_1 及 RKNN_NPU_CORE_0_1_2 模式下，目前以下 OP 能获得更好的加速：Conv、DepthwiseConvolution、Add、Concat、Relu、Clip、Relu6、ThresholdedRelu、PRelu、LeakyRelu，其余类型 OP 将 fallback 至单核 Core0 中运行，部分类型 OP（如 Pool 类、ConvTranspose 等）将在后续更新版本中支持。

4.3.3 rknn_set_batch_core_num

rknn_set_batch_core_num 函数指定多 batch RKNN 模型（RKNN-Toolkit2 转换时设置 rknn_batch_size 大于 1 导出的模型）的 NPU 核心数量，该函数仅支持 RK3588/RK3576 平台。

API	rknn_set_batch_core_num
功能	设置多 batch RKNN 模型运行的 NPU 核心数量。
参数	rknn_context context：rknn_context 对象。
	int core_num：指定运行的核心数量。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_set_batch_core_num(ctx, 2);
```

4.3.4 rknn_dup_context

rknn_dup_context 生成一个指向同一个模型的新 context，可用于多线程执行相同模型时的权重复用。**RV1106/RV1103 平台暂不支持** 。

API	rknn_dup_context
功能	生成同一个模型的两个 ctx，复用模型的权重信息。
参数	rknn_context * context_in：rknn_context 指针。初始化后的 rknn_context 对象。
	rknn_context * context_out：rknn_context 指针。生成新的 rknn_context 对象。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_context ctx_in;
rknn_context ctx_out;
int ret = rknn_dup_context(&ctx_in, &ctx_out);
```

4.3.5 rknn_destroy

rknn_destroy 函数将释放传入的 rknn_context 及其相关资源。

API	rknn_destroy
功能	销毁 rknn_context 对象及其相关资源。
参数	rknn_context context：要销毁的 rknn_context 对象。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_destroy (ctx);
```

4.3.6 rknn_query

rknn_query 函数能够查询获取到模型输入输出信息、逐层运行时间、模型推理的总时间、

SDK 版本、内存占用信息、用户自定义字符串等信息。

API	rknn_query
功能	查询模型与 SDK 的相关信息。
参数	rknn_context context：rknn_context 对象。
	rknn_query_cmd ： 查询命令。
	void* info：存放返回结果的结构体变量。
	uint32_t size：info 对应的结构体变量的大小。
返回值	int 错误码（见 RKNN 返回值错误码）。

当前 SDK 支持的查询命令如下表所示：

查询命令	返回结果结构体	功能
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	查询输入输出 tensor 个数。
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	查询输入 tensor 属性。
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	查询输出 tensor 属性。
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	查询网络各层运行时间，需要调用 rknn_init 接口时，设置 RKNN_FLAG_COLLECT_PERF_MASK 标志才能生效。
RKNN_QUERY_PERF_RUN	rknn_perf_run	查询推理模型（不包含设置输入/输出）的耗时，单位是微秒。
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	查询 SDK 版本。
RKNN_QUERY_MEM_SIZE	rknn_mem_size	查询分配给权重和网络中间 tensor 的内存大小。
RKNN_QUERY_CUSTOM_STRING	rknn_custom_string	查询 RKNN 模型里面的用户自定义字符串信息。
RKNN_QUERY_NATIVE_INPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输入 tensor 属性，它是 NPU 直接读取的模型输入属性。
RKNN_QUERY_NATIVE_OUTPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输出 tensor 属性，它是 NPU 直接输出的模型输出属性。
RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输入 tensor 属性，它是 NPU 直接读取的模型输入属性 与 RKNN_QUERY_NATIVE_INPUT_ATTR 查询结果一致。
RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输出 tensor 属性，它是 NPU 直接输出的模型输出属与 RKNN_QUERY_NATIVE_OU

		TPUT_ATTR 查询结果一致性。
RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输入 tensor 属性与 RKNN_QUERY_NATIVE_INPUT_ATTR 查询结果一致 。
RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR	rknn_tensor_attr	使用零拷贝 API 接口时,查询原生输出 NHWC tensor 属性。
RKNN_QUERY_INPUT_DYNAMIC_RANGE	rknn_input_range	使用支持动态形状 RKNN 模型时，查询模型支持输入形状数量、列表、形状对应的数据布局和名称等信息。
RKNN_QUERY_CURRENT_INPUT_ATTR	rknn_tensor_attr	使用支持动态形状 RKNN 模型时，查询模型当前推理所使用的输入属性。
RKNN_QUERY_CURRENT_OUTPUT_ATTR	rknn_tensor_attr	使用支持动态形状 RKNN 模型时，查询模型当前推理所使用的输出属性。
RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR	rknn_tensor_attr	使用支持动态形状 RKNN 模型时，查询模型当前推理所使用的 NPU 原生输入属性。
RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR	rknn_tensor_attr	使用支持动态形状 RKNN 模型时，查询模型当前推理所使用的 NPU 原生输出属性。

各个指令用法的详细说明，如下：

1) 查询 SDK 版本

传入 RKNN_QUERY_SDK_VERSION 命令可以查询 RKNN SDK 的版本信息。其中需要先创建 rknn_sdk_version 结构体对象。

示例代码如下：

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version,
                 sizeof(rknn_sdk_version));
printf("sdk api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

2) 查询输入输出 tensor 个数

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_IN_OUT_NUM 命令可以查询模型输入输出 tensor 的个数。其中需要先创建 rknn_input_output_num 结构体对象。

示例代码如下：

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num,
                 sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input,
                 io_num.n_output);
```

3) 查询输入 tensor 属性(用于通用 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_INPUT_ATTR 命令可以查询模型输入 tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象 (**注意：RV1106/RV1103 查询出来的 tensor 是原始输入 native 的 tensor**) 。

示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
   input_attrs[i].index = i;
   ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
                    sizeof(rknn_tensor_attr));
}
```

4) 查询输出 tensor 属性(用于通用 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_OUTPUT_ATTR 命令可以查询模型输出 tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
   output_attrs[i].index = i;
   ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
                                 sizeof(rknn_tensor_attr));
}
```

5) 查询模型推理的逐层耗时

在 rknn_run 接口调用完毕后，rknn_query 接口传入 RKNN_QUERY_PERF_DETAIL 可以查询网络推理时逐层的耗时，单位是微秒。使用该命令的前提是，在 rknn_init 接口的 flag 参数需要包含 RKNN_FLAG_COLLECT_PERF_MASK 标志。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size,
                        RKNN_FLAG_COLLECT_PERF_MASK, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail,
                        sizeof(perf_detail));
```

6) 查询模型推理的总耗时

在 rknn_run 接口调用完毕后，rknn_query 接口传入 RKNN_QUERY_PERF_RUN 可以查询上模型推理（不包含设置输入/输出）的耗时，单位是微秒。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
...
ret = rknn_run(ctx,NULL);
...
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run,
                        sizeof(perf_run));
```

7) 查询模型的内存占用情况

在 rknn_init 接口调用完毕后，当用户需要自行分配网络的内存时，rknn_query 接口传入 RKNN_QUERY_MEM_SIZE 可以查询模型的权重、网络中间 tensor 的内存（不包括输入和输出）、推演模型所用的所有 DMA 内存的以及 SRAM 内存（如果 sram 没开或者没有此项功能则为 0）的占用情况。使用该命令的前提是在 rknn_init 接口的 flag 参数需要包含 RKNN_FLAG_MEM_ALLOC_OUTSIDE 标志。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size,
                    RKNN_FLAG_MEM_ALLOC_OUTSIDE , NULL);
rknn_mem_size mem_size;
ret = rknn_query(ctx, RKNN_QUERY_MEM_SIZE, &mem_size,
                    sizeof(mem_size));
```

8) 查询模型中用户自定义字符串

在 rknn_init 接口调用完毕后，当用户需要查询生成 RKNN 模型时加入的自定义字符串，rknn_query 接口传入 RKNN_QUERY_CUSTOM_STRING 可以获取该字符串。例如，在转换 RKNN 模型时，用户填入“RGB”的自定义字符来标识 RKNN 模型输入是 RGB 格式三通道图像而不是 BGR 格式三通道图像，在运行时则根据查询到的“RGB”信息将数据转换成 RGB 图像。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
rknn_custom_string custom_string;
ret = rknn_query(ctx, RKNN_QUERY_CUSTOM_STRING, &custom_string,
                    sizeof(custom_string));
```

9) 查询原生输入 tensor 属性(用于零拷贝 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_NATIVE_INPUT_ATTR 命令（同 RKNN_QUERY_NATIVE_NC1HWC2_INPUT_ATTR）可以查询模型原生输入 tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
   input_attrs[i].index = i;
   ret = rknn_query(ctx, RKNN_QUERY_NATIVE_INPUT_ATTR,
                                &(input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

10) 查询原生输出 tensor 属性(用于零拷贝 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_NATIVE_OUTPUT_ATTR 命令（同 RKNN_QUERY_NATIVE_NC1HWC2_OUTPUT_ATTR）可以查询模型原生输出 tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
   output_attrs[i].index = i;
   ret = rknn_query(ctx, RKNN_QUERY_NATIVE_OUTPUT_ATTR,
                                &(output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

11) 查询 NHWC 格式原生输入 tensor 属性(用于零拷贝 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR 命令可以查询模型 **NHWC 格式**输入 tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
   input_attrs[i].index = i;
   ret = rknn_query(ctx, RKNN_QUERY_NATIVE_NHWC_INPUT_ATTR,
                                &(input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

12) 查询 NHWC 格式原生输出 tensor 属性(用于零拷贝 API 接口)

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR 命令可以查询模型 **NHWC 格式**输出 tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
   output_attrs[i].index = i;
   ret = rknn_query(ctx, RKNN_QUERY_NATIVE_NHWC_OUTPUT_ATTR,
                                  &(output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

13) 查询 RKNN 模型支持的动态输入形状信息（注：RV1106/RV1103 不支持该接口）

在 rknn_init 接口调用完毕后，传入 RKNN_QUERY_INPUT_DYNAMIC_RANGE 命令可以查询模型支持的输入形状信息，包含输入形状个数 、输入形状列表、输入形状对应的布局和名称等信息。其中需要先创建 rknn_input_range 结构体对象。

示例代码如下：

```
rknn_input_range dyn_range[io_num.n_input];
memset(dyn_range, 0, io_num.n_input * sizeof(rknn_input_range));
for (uint32_t i = 0; i < io_num.n_input; i++) {
  dyn_range[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_INPUT_DYNAMIC_RANGE,
        &dyn_range[i], sizeof(rknn_input_range));
}
```

14) 查询 RKNN 模型当前使用的输入动态形状

在 rknn_set_input_shapes 接口调用完毕后，传入 RKNN_QUERY_CURRENT_INPUT_ATTR 命令可以查询模型当前使用的输入属性信息。其中需要先创建 rknn_tensor_attr 结构体（注：RV1106/RV1103 不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_input_attrs[io_num.n_input];
memset(cur_input_attrs, 0, io_num.n_input * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_input; i++) {
  cur_input_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_INPUT_ATTR,
        &(cur_input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

15) 查询 RKNN 模型当前使用的输出动态形状

在 rknn_set_input_shapes 接口调用完毕后，传入 RKNN_QUERY_CURRENT_OUTPUT_ATTR

命令可以查询模型当前使用的输出属性信息。其中需要先创建 rknn_tensor_attr 结构体（注：RV1106/RV1103 不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_output_attrs[io_num.n_output];
memset(cur_output_attrs, 0, io_num.n_output * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_output; i++) {
  cur_output_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_OUTPUT_ATTR,
        &(cur_output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

16) 查询 RKNN 模型当前使用的原生输入动态形状

在 rknn_set_input_shapes 接口调用完毕后，传入 RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR 命令可以查询模型当前使用的原生输入属性信息。其中需要先创建 rknn_tensor_attr 结构体（注：RV1106/RV1103 不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_input_attrs[io_num.n_input];
memset(cur_input_attrs, 0, io_num.n_input * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_input; i++) {
  cur_input_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_NATIVE_INPUT_ATTR,
        &(cur_input_attrs[i]), sizeof(rknn_tensor_attr));
}
```

17) 查询 RKNN 模型当前使用的原生输出动态形状

在 rknn_set_input_shapes 接口调用完毕后，传入 RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR 命令可以查询模型当前使用的原生输出属性信息。其中需要先创建 rknn_tensor_attr 结构体（注：RV1106/RV1103 不支持该命令）。

示例代码如下：

```
rknn_tensor_attr cur_output_attrs[io_num.n_output];
memset(cur_output_attrs, 0, io_num.n_output * sizeof(rknn_tensor_attr));
for (uint32_t i = 0; i < io_num.n_output; i++) {
  cur_output_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_CURRENT_NATIVE_OUTPUT_ATTR,
        &(cur_output_attrs[i]), sizeof(rknn_tensor_attr));
}
```

4.3.7 rknn_inputs_set

通过 rknn_inputs_set 函数可以设置模型的输入数据。该函数能够支持多个输入，其中每个输入是 rknn_input 结构体对象，在传入之前用户需要设置该对象（**注：RV1106/RV1103 不支持该接口**）。

API	rknn_inputs_set
功能	设置模型输入数据。
参数	rknn_context context：rknn_context 对象。
	uint32_t n_inputs：输入数据个数。
	rknn_input inputs[]：输入数据数组，数组每个元素是 rknn_input 结构体对象。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;
inputs[0].pass_through = 0;

ret = rknn_inputs_set(ctx, 1, inputs);
```

4.3.8 rknn_run

rknn_run 函数将执行一次模型推理，调用之前需要先通过 rknn_inputs_set 函数或者零拷贝的接口设置输入数据。

API	rknn_run
功能	执行一次模型推理。
参数	rknn_context context：rknn_context 对象。
	rknn_run_extend* extend：保留扩展，当前没有使用，传入 NULL 即可。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
ret = rknn_run(ctx, NULL);
```

4.3.9 rknn_outputs_get

rknn_outputs_get 函数可以获取模型推理的输出数据。该函数能够一次获取多个输出数据。其中每个输出是 rknn_output 结构体对象，在函数调用之前需要依次创建并设置每个 rknn_output 对象。

对于输出数据的 buffer 存放可以采用两种方式：一种是用户自行申请和释放，此时 rknn_output 对象的 is_prealloc 需要设置为 1，并且将 buf 指针指向用户申请的 buffer；另一种是由 rknn 来进行分配，此时 rknn_output 对象的 is_prealloc 设置为 0 即可，函数执行之后 buf 将指向输出数据。**（注：RV1106/RV1103 不支持该接口）**

API	rknn_outputs_get
功能	获取模型推理输出。
参数	rknn_context context：rknn_context 对象。
	uint32_t n_outputs：输出数据个数。
	rknn_output outputs[]：输出数据的数组，其中数组每个元素为 rknn_output 结构体对象，代表模型的一个输出。
	rknn_output_extend* extend：保留扩展，当前没有使用，传入 NULL 即可。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
    outputs[i].index = i;
    outputs[i].is_prealloc = 0;
    outputs[i].want_float = 1;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

4.3.10 rknn_outputs_release

rknn_outputs_release 函数将释放 rknn_outputs_get 函数得到的输出的相关资源。

API	rknn_outputs_release
功能	释放 rknn_output 对象。
参数	rknn_context context：rknn_context 对象。
	uint32_t n_outputs：输出数据个数。
	rknn_output outputs[]：要销毁的 rknn_output 数组。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

4.3.11 rknn_create_mem_from_phys

当用户需要自己分配内存让 NPU 使用时，通过 rknn_create_mem_from_phys 函数可以创建一个 rknn_tensor_mem 结构体并得到它的指针，该函数通过传入物理地址、虚拟地址以及大小，外

部内存相关的信息会赋值给 rknn_tensor_mem 结构体。

API	rknn_create_mem_from_phys
功能	通过物理地址创建 rknn_tensor_mem 结构体并分配内存。
参数	rknn_context context：rknn_context 对象。
	uint64_t phys_addr：内存的物理地址。
	void *virt_addr：内存的虚拟地址。
	uint32_t size：内存的大小。
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

示例代码如下：

```
//suppose we have got buffer information as input_phys, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_phys(ctx, input_phys, input_virt, size);
```

4.3.12 rknn_create_mem_from_fd

当用户要自己分配内存让 NPU 使用时，rknn_create_mem_from_fd 函数可以创建一个 rknn_tensor_mem 结构体并得到它的指针，该函数通过传入文件描述符 fd、偏移、虚拟地址以及大小，外部内存相关的信息会赋值给 rknn_tensor_mem 结构体。

API	rknn_create_mem_from_fd
功能	通过文件描述符创建 rknn_tensor_mem 结构体。
参数	rknn_context context：rknn_context 对象。
	int32_t fd：内存的文件描述符。
	void *virt_addr：内存的虚拟地址，**fd 对应的内存的首地址**。
	uint32_t size：内存的大小。
	int32_t offset：内存相对于文件描述符和虚拟地址的偏移量。
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

示例代码如下：

```
//suppose we have got buffer information as input_fd, input_virt and size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem_from_fd(ctx, input_fd, input_virt, size, 0);
```

4.3.13 rknn_create_mem

当用户要 NPU 内部分配内存时，rknn_create_mem 函数可以分配用户指定的内存大小，并返回一个 rknn_tensor_mem 结构体。

API	rknn_create_mem
功能	创建 rknn_tensor_mem 结构体并分配内存。
参数	rknn_context context：rknn_context 对象。
	uint32_t size：内存的大小。
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

示例代码如下：

```
//suppose we have got buffer size
rknn_tensor_mem* input_mems [1];
input_mems[0] = rknn_create_mem(ctx, size);
```

4.3.14 rknn_create_mem2

当用户要 NPU 内部分配内存时，rknn_create_mem2 函数可以分配用户指定的内存大小及内存类型，并返回一个 rknn_tensor_mem 结构体。

API	rknn_create_mem2
功能	创建 rknn_tensor_mem 结构体并分配内存。
参数	rknn_context context：rknn_context 对象。
	uint64_t size：内存的大小。
	uint64_t alloc_flags: 控制内存是否是 cacheable 的。 RKNN_FLAG_MEMORY_CACHEABLE：创建 cacheable 内存 RKNN_FLAG_MEMORY_NON_CACHEABLE: 创建 non -cacheable 内存 RKNN_FLAG_MEMORY_FLAGS_DEFAULT ： 同 RKNN_FLAG_MEMORY_CACHEABLE
返回值	rknn_tensor_mem*： tensor 内存信息结构体指针。

rknn_create_mem2 与 rknn_create_mem 的主要区别是 rknn_create_mem2 带了一个 alloc_flags，可以指定分配的内存是否 cacheable 的，而 rknn_create_mem 不能指定，默认就是 cacheable。

示例代码如下：

```
    //suppose we have got buffer size
    rknn_tensor_mem* input_mems [1];
    input_mems[0] = rknn_create_mem2(ctx, size,
RKNN_FLAG_MEMORY_NON_CACHEABLE);
```

4.3.15 rknn_destroy_mem

rknn_destroy_mem 函数会销毁 rknn_tensor_mem 结构体，用户分配的内存需要自行释放。

API	rknn_destroy_mem
功能	销毁 rknn_tensor_mem 结构体。
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：tensor 内存信息结构体指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* input_mems [1];
int ret = rknn_destroy_mem(ctx, input_mems[0]);
```

4.3.16 rknn_set_weight_mem

如果用户自己为网络权重分配内存，初始化相应的 rknn_tensor_mem 结构体后，在调用 rknn_run 前，通过 rknn_set_weight_mem 函数可以让 NPU 使用该内存。

API	rknn_set_weight_mem
功能	设置包含权重内存信息的 rknn_tensor_mem 结构体。
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：权重 tensor 内存信息结构体指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* weight_mems [1];
int ret = rknn_set_weight_mem(ctx, weight_mems[0]);
```

4.3.17 rknn_set_internal_mem

如果用户自己为网络中间 tensor 分配内存，初始化相应的 rknn_tensor_mem 结构体后，在调用 rknn_run 前，通过 rknn_set_internal_mem 函数可以让 NPU 使用该内存。

API	rknn_set_internal_mem
功能	设置包含中间 tensor 内存信息的 rknn_tensor_mem 结构体。
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：模型中间 tensor 内存信息结构体指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_tensor_mem* internal_tensor_mems [1];
int ret = rknn_set_internal_mem(ctx, internal_tensor_mems[0]);
```

4.3.18 rknn_set_io_mem

如果用户自己为网络输入/输出 tensor 分配内存，初始化相应的 rknn_tensor_mem 结构体后，在调用 rknn_run 前，通过 rknn_set_io_mem 函数可以让 NPU 使用该内存。

API	rknn_set_io_mem
功能	设置包含模型输入/输出内存信息的 rknn_tensor_mem 结构体。
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem*：输入/输出 tensor 内存信息结构体指针。
	rknn_tensor_attr *: 输入/输出 tensor 的属性。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
    rknn_tensor_attr output_attrs[1];
    rknn_tensor_mem* output_mems[1];

    ret = rknn_query(ctx, RKNN_QUERY_NATIVE_OUTPUT_ATTR, &(output_attrs[0]),
sizeof(rknn_tensor_attr));
    output_mems[0] = rknn_create_mem(ctx, output_attrs[0].size_with_stride);
    rknn_set_io_mem(ctx, output_mems[0], &output_attrs[0]);
```

4.3.19 rknn_set_input_shape（deprecated）

该接口已经废弃，请使用 rknn_set_input_shapes 接口绑定输入形状。当前版本不可用，如要继续使用该接口，请使用 1.5.0 版本 SDK 并参考 1.5.0 版本的使用指南文档。

4.3.20 rknn_set_input_shapes

对于动态形状输入 RKNN 模型，在推理前必须指定当前使用的输入形状。该接口传入输入个数和 rknn_tensor_attr 数组，包含了每个输入形状和对应的数据布局信息，将每个 rknn_tensor_attr 结构体对象的索引、名称、形状（dims）和内存布局信息（fmt）必须填充，rknn_tensor_attr 结构体其他成员无需设置。在使用该接口前，可先通过 rknn_query 函数查询 RKNN 模型支持的输入形状数量和动态形状列表，要求输入数据的形状在模型支持的输入形状列表中。初次运行或每次切换新的输入形状，需要调用该接口设置新的形状，否则，不需要重复调用该接口。

API	rknn_set_input_shapes
功能	设置模型当前使用的输入形状。
参数	rknn_context context：rknn_context 对象。
	uint32_t n_inputs：输入 Tensor 的数量。
	rknn_tensor_attr *: 输入 tensor 的属性数组指针，传递所有输入的形状信息，用户需要设置每个输入属性结构体中的 index、name、dims、fmt、n_dims 成员，其他成员无需设置。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
for (int i = 0; i < io_num.n_input; i++) {
    for (int j = 0; j < input_attrs[i].n_dims; ++j) {
        //使用第一个动态输入形状
        input_attrs[i].dims[j] = dyn_range[i].dyn_range[0][j];
    }
}
ret = rknn_set_input_shapes(ctx, io_num.n_input, input_attrs);
if (ret < 0) {
    fprintf(stderr, "rknn_set_input_shapes error! ret=%d\n", ret);
    return -1;
}
```

4.3.21 rknn_mem_sync

rknn_create_mem 函数创建的内存默认是带 cacheable 标志的，对于带 cacheable 标志创建的内存，在被 CPU 和 NPU 同时使用时，由于 cache 行为会导致数据一致性问题。该接口用于同步一块带 cacheable 标志创建的内存，保证 CPU 和 NPU 访问这块内存的数据是一致的。

API	rknn_mem_sync
功能	同步 CPU cache 和 DDR 数据。
参数	rknn_context context：rknn_context 对象。
	rknn_tensor_mem* mem：tensor 内存信息结构体指针。
	rknn_mem_sync_mode mode: 表示刷新 CPU cache 和 DDR 数据的模式。 **RKNN_MEMORY_SYNC_TO_DEVICE**：表示 CPU cache 数据同步到 DDR 中，通常用于 CPU 写入内存后，NPU 访问相同内存前使用该模式将 cache 中的数据写回 DDR。 **RKNN_MEMORY_SYNC_FROM_DEVICE**：表示 DDR 数据同步到 CPU cache，通常用于 NPU 写入内存后，使用该模式让下次 CPU 访问相同内存时，cache 数据无效，CPU 从 DDR 重新读取数据。 **RKNN_MEMORY_SYNC_BIDIRECTIONAL**：表示 CPU cache 数据同步到 DDR 同时令 CPU 重新从 DDR 读取数据。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
ret =rknn_mem_sync(ctx, &outputs[0].mem,
                           RKNN_MEMORY_SYNC_FROM_DEVICE);
if (ret < 0) {
    fprintf(stderr, " rknn_mem_sync error! ret=%d\n", ret);
    return -1;
}
```

4.4 矩阵乘法数据结构定义

4.4.1 rknn_matmul_info

rknn_matmul_info 表示用于执行矩阵乘法的规格信息，它包含了矩阵乘法的规模、输入和输出矩阵的数据类型和内存排布。结构体的定义如下表所示：

成员变量	数据类型	含义
M	int32_t	A 矩阵的行数
K	int32_t	A 矩阵的列数
N	int32_t	B 矩阵的列数
type	rknn_matmul_type	输入输出矩阵的数据类型： RKNN_FLOAT16_MM_FLOAT16_TO_FLOAT32：表示矩阵 A 和 B 是 float16 类型，矩阵 C 是 float32 类型； RKNN_INT8_MM_INT8_TO_INT32：表示矩阵 A 和 B 是 int8 类型，矩阵 C 是 int32 类型； RKNN_INT8_MM_INT8_TO_INT8：表示矩阵 A、B 和 C 是 int8 类型； RKNN_FLOAT16_MM_FLOAT16_TO_FLOAT16：表示矩阵 A、B 和 C 是 float16 类型； RKNN_FLOAT16_MM_INT8_TO_FLOAT32：表示矩阵 A 是 float16 类型，矩阵 B 是 int8 类型，矩阵 C 是 float32 类型； RKNN_FLOAT16_MM_INT8_TO_FLOAT16：表示矩阵 A 是 float16 类型，矩阵 B 是 int8 类型，矩阵 C 是 float16 类型； RKNN_FLOAT16_MM_INT4_TO_FLOAT32：表示矩阵 A 是 float16 类型，矩阵 B 是 int4 类型，矩阵 C 是 float32 类型； RKNN_FLOAT16_MM_INT4_TO_FLOAT16：表示矩阵 A 是 float16 类型，矩阵 B 是 int4 类型，矩阵 C 是 float16 类型；RKNN_INT4_MM_INT4_TO_INT16：表示矩阵 A 和 B 是 int4 类型，矩阵 C 是 int16 类型；

		RKNN_INT8_MM_INT4_TO_INT32：表示矩阵 A 和 B 是 int4 类型，矩阵 C 是 int32 类型
B_layout	int16_t	指定矩阵 B 的数据排列方式。 0：表示矩阵 B 按照原始形状排列 1：表示矩阵 B 按照高性能形状排列
B_quant_type	int16_t	指定矩阵 B 的量化方式类型。 0：表示矩阵 B 按照 Per-Layer 方式量化 1：表示矩阵 B 按照 Per-Channel 方式量化
AC_layout	int16_t	指定矩阵 A 和矩阵 C 的数据排列方式。 0：表示矩阵 A 和 C 按照原始形状排列 1：表示矩阵 A 和 C 按照高性能形状排列
AC_quant_type	int16_t	指定矩阵 A 和 C 的量化方式类型。 0：表示矩阵 A 和 C 按照 Per-Layer 方式量化 1：表示矩阵 A 和 C 按照 Per-Channel 方式量化
iommu_domain_id	int32_t	矩阵上下文所在的 IOMMU 地址空间域的索引。IOMMU 地址空间与上下文一一对应，每个 IOMMU 地址空间大小为 4GB。该参数主要用于矩阵 A、B 和 C 的参数规格较大，某个域内 NPU 分配的内存超过 4GB 以后需切换另一个域时使用。
reserved	int8_t[]	预留字段

4.4.2 rknn_matmul_tensor_attr

rknn_matmul_tensor_attr 表示每个矩阵 tensor 的属性，它包含了矩阵的名字、形状、大小和数据类型。结构体的定义如下表所示：

成员变量	数据类型	含义
name	char[]	矩阵的名字
n_dims	uint32_t	矩阵的维度个数
dims	uint32_t[]	矩阵的形状
size	uint32_t	矩阵的大小，以字节为单位
type	rknn_tensor_type	矩阵的数据类型

4.4.3 rknn_matmul_io_attr

rknn_matmul_io_attr 表示矩阵所有输入和输出 tensor 的属性，它包含了矩阵 A、B 和 C 的属性。结构体的定义如下表所示：

成员变量	数据类型	含义
A	rknn_matmul_tensor_attr	矩阵 A 的 tensor 属性
B	rknn_matmul_tensor_attr	矩阵 B 的 tensor 属性
C	rknn_matmul_tensor_attr	矩阵 C 的 tensor 属性

4.4.4 rknn_quant_params

rknn_quant_params 表示矩阵的量化参数，包括 name 以及 scale 和 zero_point 数组的指针和长度，name 用来标识矩阵的名称，它可以从初始化矩阵上下文时得到的 rknn_matmul_io_attr 结构体中获取。结构体定义如下表所示:

成员变量	数据类型	含义
name	char[]	矩阵的名字
scale	float*	矩阵的 scale 数组指针
scale_len	int32_t	矩阵的 scale 数组长度
zp	int32_t*	矩阵的 zero_point 数组指针
zp_len	int32_t	矩阵的 zero_point 数组长度

4.4.5 rknn_matmul_shape

rknn_matmul_shape 表示某个特定 shape 的矩阵乘法的 M、K 和 N，在初始化动态 shape 的矩阵乘法上下文时，需要提供 shape 的数量，并使用 rknn_matmul_shape 结构体数组表示所有的输入的 shape。结构体定义如下表所示:

成员变量	数据类型	含义
M	int32_t	矩阵 A 的行数
K	int32_t	矩阵 A 的列数

N	int32_t	矩阵 B 的列数

4.5 矩阵乘法 API 说明

4.5.1 rknn_matmul_create

该函数的功能是根据传入的矩阵乘法规格等信息，完成矩阵乘法上下文的初始化，并返回输入和输出 tensor 的形状、大小和数据类型等信息。

API	rknn_matmul_create
功能	初始化矩阵乘法上下文。
参数	rknn_matmul_ctx* ctx：矩阵乘法上下文指针。
	rknn_matmul_info* info：矩阵乘法的规格信息结构体指针。
	rknn_matmul_io_attr* io_attr：矩阵乘法输入和输出 tensor 属性结构体指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_matmul_info info;
memset(&info, 0, sizeof(rknn_matmul_info));
info.M          = 4;
info.K          = 64;
info.N          = 32;
info.type       = RKNN_INT8_MM_INT8_TO_INT32;
info.B_layout  = 0;
info.AC_layout    = 0;

rknn_matmul_io_attr io_attr;
memset(&io_attr, 0, sizeof(rknn_matmul_io_attr));

int ret = rknn_matmul_create(&ctx, &info, &io_attr);
if (ret < 0) {
  printf("rknn_matmul_create fail! ret=%d\n", ret);
  return -1;
}
```

4.5.2 rknn_matmul_set_io_mem

该函数用于设置矩阵乘法运算的输入/输出内存。在调用该函数前，先使用 rknn_create_mem 接口创建的 rknn_tensor_mem 结构体指针，接着将其与 rknn_matmul_create 函数返回的矩阵 A、B 或 C 的 rknn_matmul_tensor_attr 结构体指针传入该函数，把输入和输出内存设置到矩阵乘法上下文中。在调用该函数前，要根据 rknn_matmul_info 中配置的内存排布准备好矩阵 A 和矩阵 B 的数

据。

API	rknn_matmul_set_io_mem
功能	设置矩阵乘法的输入/输出内存。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
	rknn_tensor_mem* mem：tensor 内存信息结构体指针。
	rknn_matmul_tensor_attr* attr：矩阵乘法输入和输出 tensor 属性结构体指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
// Create A
rknn_tensor_mem* A = rknn_create_mem(ctx, io_attr.A.size);
if (A == NULL) {
  printf("rknn_create_mem fail!\n");
  return -1;
}
memset(A->virt_addr, 1, A->size);
rknn_matmul_io_attr io_attr;
memset(&io_attr, 0, sizeof(rknn_matmul_io_attr));

int ret = rknn_matmul_create(&ctx, &info, &io_attr);
if (ret < 0) {
  printf("rknn_matmul_create fail! ret=%d\n", ret);
  return -1;
}
// Set A
ret = rknn_matmul_set_io_mem(ctx, A, &io_attr.A);
if (ret < 0) {
  printf("rknn_matmul_set_io_mem fail! ret=%d\n", ret);
  return -1;
}
```

4.5.3 rknn_matmul_set_core_mask

该函数用于设置矩阵乘法运算时可用的 NPU 核心（仅支持 RK3588 和 RK3576）。在调用该函数前，需要先通过 rknn_matmul_create 函数初始化矩阵乘法上下文。可通过该函数设置的掩码值，指定需要使用的核心，以提高矩阵乘法运算的性能和效率。

API	rknn_matmul_set_core_mask
功能	设置矩阵乘法运算的 NPU 核心掩码。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
	rknn_core_mask core_mask：矩阵乘法运算的 NPU 核心掩码值，用于指定可用的 NPU 核心。掩码的每一位代表一个核心，如果对应位为 1，则表示该核心可用；否则，表示该核心不可用（详细掩码说明见 rknn_set_core_mask API 参数）。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_matmul_set_core_mask(ctx, RKNN_NPU_CORE_AUTO);
```

4.5.4 rknn_matmul_set_quant_params

rknn_matmul_set_quant_params 用于设置每个矩阵的量化参数，支持 Per-Channel 量化和 Per-Layer 量化两种方式的量化参数设置。当使用 Per-Channel 量化时，rknn_quant_params 中的 scale 和 zp 数组的长度等于 N。当使用 Per-Layer 量化时，rknn_quant_params 中的 scale 和 zp 数组的长度为 1。在 rknn_matmul_run 之前调用此接口设置所有矩阵的量化参数。如果不调用此接口，则默认量化方式为 Per-Layer 量化，scale=1.0，zero_point=0。

API	rknn_matmul_set_quant_params
功能	设置矩阵的量化参数。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
	rknn_quant_params* params： 矩阵的量化参数信息。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
rknn_quant_params params_a;
memcpy(params_a.name, io_attr.A.name, RKNN_MAX_NAME_LEN);
params_a.scale_len = 1;
params_a.scale = (float *)malloc(params_a.scale_len * sizeof(float));
params_a.scale[0] = 0.2;
params_a.zp_len = 1;
params_a.zp = (int32_t *)malloc(params_a.zp_len * sizeof(int32_t));
params_a.zp[0] = 0;
rknn_matmul_set_quant_params(ctx, &params_a);
```

4.5.5 rknn_matmul_get_quant_params

rknn_matmul_get_quant_params 用于 rknn_matmul_type 类型等于 RKNN_INT8_MM_INT8_TO_INT32 并且 Per-Channel 量化方式时，获取矩阵 B 所有通道 scale 归一化后的 scale 值，获取的 scale 值和 A 的原始 scale 值相乘可以得到 C 的 scale 值。可以用于在矩阵 C 没有真实 scale 时，近似计算得到 C 的 scale。

API	rknn_matmul_get_quant_params
功能	获取矩阵 B 的量化参数。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
	rknn_quant_params* params：矩阵 B 的量化参数信息。
	float* scale: 矩阵 B 的 scale 指针。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
float b_scale;
rknn_matmul_get_quant_params(ctx, &params_b, &b_scale);
```

4.5.6 rknn_matmul_create_dyn_shape

rknn_matmul_create_dyn_shape 用于创建动态 shape 矩阵乘法上下文，该接口需要传入 rknn_matmul_info 结构体、shape 数量以及对应的 shape 数组，shape 数组会记录多个 M、K 和 N 值。在初始化成功后，会得到 rknn_matmul_io_attr 的数组，数组中包含了所有的输入输出矩阵的 shape、大小和数据类型等信息。目前仅支持设置多个不同的 M，而 K 和 N 固定。

API	rknn_matmul_create_dyn_shape
功能	初始化动态 shape 矩阵乘法的上下文。
参数	rknn_matmul_ctx *ctx：矩阵乘法上下文指针。
	rknn_matmul_info* info：矩阵乘法的规格信息结构体指针。其中 M、K 和 N 不需要设置。
	int shape_num：矩阵上下文支持的 shape 数量。
	rknn_matmul_shape dynamic_shapes[]：矩阵上下文支持的 shape 数组。
	rknn_matmul_io_attr io_attrs[]：矩阵乘法输入和输出 tensor 属性结构体数组。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
const int          shape_num = 2;
 rknn_matmul_shape shapes[shape_num];
for (int i = 0; i < shape_num; ++i) {
  shapes[i].M = i+1;
  shapes[i].K = 64;
  shapes[i].N = 32;
}  rknn_matmul_io_attr io_attr[shape_num];
memset(io_attr, 0, sizeof(rknn_matmul_io_attr) * shape_num);

int ret = rknn_matmul_create_dyn_shape(&ctx, &info, shape_num, shapes, io_attr);
if (ret < 0) {
  fprintf(stderr, " rknn_matmul_create_dyn_shape fail! ret=%d\n", ret);
  return -1;
}
```

4.5.7 rknn_matmul_set_dynamic_shape

rknn_matmul_set_dynamic_shape 用于指定矩阵乘法使用的某一个 shape。在创建动态 shape 的矩阵乘法上下文后，选取其中一个 rknn_matmul_shape 结构体作为输入参数，调用此接口设置运算使用的 shape。

API	rknn_matmul_set_dynamic_shape
功能	设置矩阵乘法 shape。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
	rknn_matmul_shape* shape：指定矩阵乘法使用的 shape。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
ret = rknn_matmul_set_dynamic_shape(ctx, &shapes[0]);
if (ret != 0) {
  fprintf(stderr, "rknn_matmul_set_dynamic_shapes fail!\n");
  return -1;
}
```

4.5.8 rknn_B_normal_layout_to_native_layout

rknn_B_normal_layout_to_native_layout 用于将矩阵 B 的原始形状排列的数据（KxN）转换为高性能数据排列方式的数据。

API	rknn_B_normal_layout_to_native_layout
功能	将矩阵 B 的数据排列从原始形状转换成高性能形状。
参数	void* B_input：原始形状的矩阵 B 数据指针。
	void* B_output：高性能形状的矩阵 B 数据指针。
	int K：矩阵 B 的行数。
	int N：矩阵 B 的列数。
	int subN：等于 rknn_matmul_io_attr 结构体中的 B.dims[2]。
	int subK：等于 rknn_matmul_io_attr 结构体中的 B.dims[3]。
	rknn_matmul_type type：输入输出矩阵的数据类型。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
int32_t subN = io_attr.B.dims[2];
int32_t subK = io_attr.B.dims[3];
rknn_B_normal_layout_to_native_layout(B_Matrix, B->virt_addr, K, N, subN, subK,
                                      info.type);
```

4.5.9 rknn_matmul_run

该函数用于运行矩阵乘法运算，并将结果保存在输出矩阵 C 中。在调用该函数前，输入矩阵 A 和 B 需要先准备好数据，并通过 rknn_matmul_set_io_mem 函数设置到输入缓冲区。输出矩阵 C 需要先通过 rknn_matmul_set_io_mem 函数设置到输出缓冲区，而输出矩阵的 tensor 属性则通过 rknn_matmul_create 函数获取。

API	rknn_matmul_run
功能	运行矩阵乘法运算。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
int ret = rknn_matmul_run(ctx);
```

4.5.10 rknn_matmul_destroy

该函数用于销毁矩阵乘法运算上下文，释放相关资源。在使用完 rknn_matmul_create 函数创建的矩阵乘法上下文指针后，需要调用该函数进行销毁。

API	rknn_matmul_destroy
功能	销毁矩阵乘法运算上下文。
参数	rknn_matmul_ctx ctx：矩阵乘法上下文。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
int ret = rknn_matmul_destroy(ctx);
```

4.6 自定义算子数据结构定义

4.6.1 rknn_gpu_op_context

rknn_gpu_op_context 表示指定 GPU 运行的自定义算子的上下文信息。结构体的定义如下表所示：

成员变量	数据类型	含义
cl_context	void*	OpenCL 的 cl_context 对象，使用时请强制类型转换成 cl_context。
cl_command_queue	void*	OpenCL 的 cl_command_queue 对象，使用时请强制类型转换成 cl_command_queue。
cl_kernel	void*	OpenCL 的 cl_kernel 对象，使用时请强制类型转换成 cl_kernel。

4.6.2 rknn_custom_op_context

rknn_custom_op_context 表示自定义算子的上下文信息。结构体的定义如下表所示：

成员变量	数据类型	含义
target	rknn_target_type	执行自定义算子的后端设备： RKNN_TARGET_TYPE_CPU： CPU RKNN_TARGET_TYPE_CPU： GPU
internal_ctx	rknn_custom_op_interal_context	算子内部的私有上下文。
gpu_ctx	rknn_gpu_op_context	包含自定义算子的 OpenCL 上下文信息，当执行后端设备是 GPU 时，在回调函数中从该结构体获取 OpenCL 的 cl_context 等对象。
priv_data	void*	留给开发者管理的数据指针。

4.6.3 rknn_custom_op_tensor

rknn_custom_op_tensor 表示自定义算子的输入/输出的 tensor 信息。结构体的定义如下表所示：

成员变量	数据类型	含义
attr	rknn_tensor_attr	包含 tensor 的名称、形状、大小等信息。
mem	rknn_tensor_mem	包含 tensor 的内存地址、fd、有效数据偏移等信息。

4.6.4 rknn_custom_op_attr

rknn_custom_op_attr 表示自定义算子的参数或属性信息。结构体的定义如下表所示：

成员变量	数据类型	含义
name	char[]	自定义算子的参数名。
dtype	rknn_tensor_type	每个元素的数据类型。
n_elems	uint32_t	元素数量。
data	void*	参数数据内存段的虚拟地址。

4.6.5 rknn_custom_op

rknn_custom_op 表示自定义算子的注册信息。结构体的定义如下表所示：

成员变量	数据类型	含义
version	uint32_t	自定义算子版本号。
target	rknn_target_type	自定义算子执行后端类型。
op_type	char[]	自定义算子类型。
cl_kernel_name	char[]	OpenCL 的 kernel 函数名。
cl_kernel_source	char*	OpenCL 的资源名称。当 cl_source_size 等于 0 时，表示文件绝对路径；当 cl_source_size 大于 0 时，表示 kernel 函数代码的字符串。
cl_source_size	uint64_t	当 cl_kernel_source 是字符串，表示字符串长度；当 cl_kernel_source 是文件路径，则设置为 0。
cl_build_options	char[]	OpenCL kernel 的编译选项。
init	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	自定义算子初始化回调函数指针。在注册时，调用一次。不需要时可以设置为 NULL。
prepare	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	预处理回调函数指针。在 rknn_run 时调用一次。不需要时可以设置为 NULL。
compute	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	自定义算子功能的回调函数指针。在 rknn_run 时调用一次。**不能设置为 NULL**。
compute_native	int (*)(rknn_custom_op_context* op_ctx, rknn_custom_op_tensor* inputs, uint32_t n_inputs, rknn_custom_op_tensor* outputs, uint32_t n_outputs);	高性能计算的回调函数指针，它与 compute 回调函数区别是输入和输出的 tensor 的格式存在差异。暂不支持，目前设置为 NULL。
destroy	int (*)(rknn_custom_op_context* op_ctx);	销毁资源的回调函数指针。在 rknn_destroy 时调用一次。

4.7 自定义算子 API 说明

4.7.1 rknn_register_custom_ops

在初始化上下文成功后，该函数用于在上下文中注册若干个自定义算子的信息，包括自定义算子类型、运行后端类型、OpenCL 内核信息以及回调函数指针。注册成功后，在推理阶段，rknn_run 接口会调用开发者实现的回调函数。

API	rknn_register_custom_ops
功能	注册若干个自定义算子到上下文中。
参数	rknn_context *context：rknn_context 指针。函数调用之前，context 必须已经初始化成功。
	rknn_custom_op* op：自定义算子信息数组，数组每个元素是 rknn_custom_op 结构体对象。
	uint32_t custom_op_num：自定义算子信息数组长度。
返回值	int 错误码（见 RKNN 返回值错误码）。

示例代码如下：

```
// CPU operators
rknn_custom_op user_op[2];
memset(user_op, 0, 2 * sizeof(rknn_custom_op));
strncpy(user_op[0].op_type, "cstSoftmax", RKNN_MAX_NAME_LEN - 1);
user_op[0].version = 1;
user_op[0].target  = RKNN_TARGET_TYPE_CPU;
user_op[0].init    = custom_op_init_callback;
user_op[0].compute = compute_custom_softmax_float32;
user_op[0].destroy = custom_op_destroy_callback;

strncpy(user_op[1].op_type, "ArgMax", RKNN_MAX_NAME_LEN - 1);
user_op[1].version = 1;
user_op[1].target  = RKNN_TARGET_TYPE_CPU;
user_op[1].init    = custom_op_init_callback;
user_op[1].compute = compute_custom_argmax_float32;
user_op[1].destroy = custom_op_destroy_callback;

ret = rknn_register_custom_ops(ctx, user_op, 2);
if (ret < 0) {
  printf("rknn_register_custom_ops fail! ret = %d\n", ret);
  return -1;
}
```

4.7.2 rknn_custom_op_get_op_attr

该函数用于在自定义算子的回调函数中获取自定义算子的参数信息，例如 Softmax 算子的 axis 参数。它传入自定义算子参数的字段名称和一个 rknn_custom_op_attr 结构体指针，调用该接口后，参数值会存储在 rknn_custom_op_attr 结构体中的 data 成员中，开发者根据返回的结构体内 dtype 成员将该指针强制转换成 C 语言中特定数据类型的数组首地址，再按照元素数量读取出完整参数值。

API	rknn_custom_op_get_op_attr
功能	获取自定义算子的参数或属性。
参数	rknn_custom_op_context* op_ctx：自定义算子上下文指针。
	const char* attr_name：自定义算子参数的字段名称。
	rknn_custom_op_attr* op_attr：表示自定义算子参数值的结构体。
返回值	无

示例代码如下：

```
  rknn_custom_op_attr op_attr;
  rknn_custom_op_get_op_attr(op_ctx, "axis", &op_attr);
  if (op_attr.n_elems == 1 && op_attr.dtype == RKNN_TENSOR_INT64) {
    axis = ((int64_t*)op_attr.data)[0];
  }
…
```

5 RKNN 返回值错误码

RKNN API 函数的返回值错误码定义如下表所示：

错误码	错误详情
RKNN_SUCC(0)	执行成功。
RKNN_ERR_FAIL(-1)	执行出错。
RKNN_ERR_TIMEOUT(-2)	执行超时。
RKNN_ERR_DEVICE_UNAVAILABLE(-3)	NPU 设备不可用。
RKNN_ERR_MALLOC_FAIL(-4)	内存分配失败。
RKNN_ERR_PARAM_INVALID(-5)	传入参数错误。
RKNN_ERR_MODEL_INVALID(-6)	传入的 RKNN 模型无效。
RKNN_ERR_CTX_INVALID(-7)	传入的 rknn_context 无效。
RKNN_ERR_INPUT_INVALID(-8)	传入的 rknn_input 对象无效。
RKNN_ERR_OUTPUT_INVALID(-9)	传入的 rknn_output 对象无效。
RKNN_ERR_DEVICE_UNMATCH(-10)	版本不匹配。
RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION(-12)	RKNN 模型设置了优化等级的选项，但是和当前驱动不兼容。
RKNN_ERR_TARGET_PLATFORM_UNMATCH(-13)	RKNN 模型和当前平台不兼容。