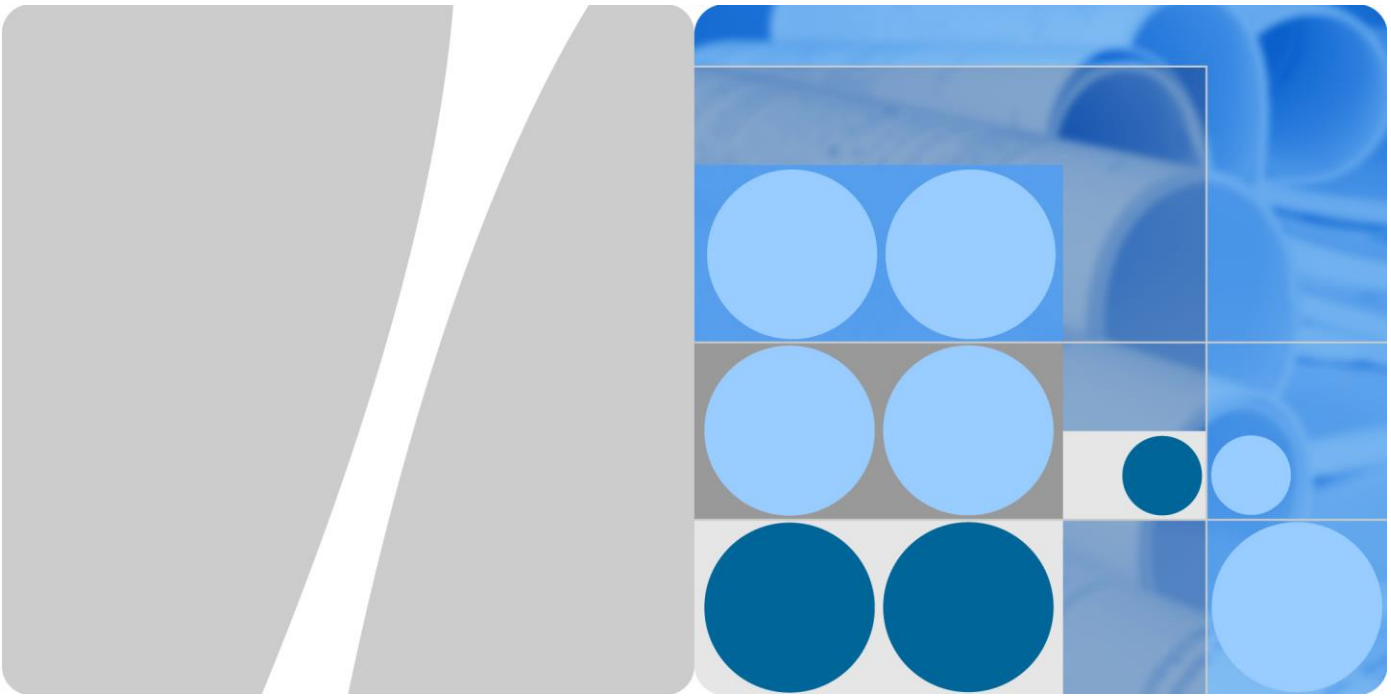


HiAI DDK V320

IR 模型构建使用说明书

文档版本 02

发布日期 2019-12-31

华为技术有限公司

HUAWEI

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

法律声明

本文所描述内容可能包含但不限于对非华为或开源软件的介绍或引用，使用它们时请遵循对方的版权要求。

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 HiAI 申请方式

发送申请邮件到邮箱：developer@huawei.com

邮件名称：HUAWEI HiAI+公司名称+产品名称

邮件正文：合作公司+联系人+联系方式+联系邮箱

我们将在收到邮件的 5 个工作日内邮件给您反馈结果，请您注意查收。

官网地址 https://developer.huawei.com/consumer/cn/

前 言

概述

本文提供对 IR 模型生成操作的说明，包括算子 API、离线、在线构建说明等。

修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	修订版本	修改描述
2019-12-31	02	新增 V320 版本内容
2019-09-04	01	新增 V310 版本内容

目 录

插图目录

表格目录

1 模型构建

1.1 概述

IR（Intermediate Representation）模型构建，是让用户通过 HiAI 开发的算子 IR 结构，自主地构造网络模型。

目前，构建模型主要包括两种方式：在线构建模型、离线构建模型。

在线构建模型，主要针对第三方框架：用户可以参照 HiAI IR API 定义，将第三方框架代码映射到 IR API，在应用运行期间构造模型，并执行推理。也可以将构造的模型进行序列化，供后面 APP 执行期使用。

离线构建模型，主要针对有特殊需求的高级用户：用户可以直接使用 IR API 进行编程，构造推理模型。

说明

HiAI DDK 100.320.020.010 原始模型权值为 float32 的场景，维持权值数据类型不变，预期 IR 模型大小较历史版本会增大。

1.2 限制条件

手机芯片上 ROM 和 RAM 空间有限，需要对模型大小和运行内存进行限制：

- 建议模型大小不超过 100M
- 建议峰值运行内存占用大小不超过 200M

1.3 支持的算子

hiai::op 算子定义所在目录： /ddk/ai_ddk_lib/include/graph/op/。

算子名	所属分类	所在文件
Acos	math_defs	math_defs.h
Acosh	math_defs	math_defs.h

算子名	所属分类	所在文件
Activation	nn_defs	nn_defs.h
Add	math_defs	math_defs.h
ArgMaxExt2	math_defs	math_defs.h
Asin	math_defs	math_defs.h
Asinh	math_defs	math_defs.h
Atan	math_defs	math_defs.h
Atanh	math_defs	math_defs.h
AxisAlignedBboxTransform	nn_defs	nn_defs.h
BatchMatMul	math_defs	math_defs.h
BatchReindex	array_defs	array_defs.h
BatchToSpaceND	array_defs	array_defs.h
BiasAdd	nn_defs	nn_defs.h
BidirectionLSTM	nn_defs	nn_defs.h
BNInference	nn_defs	nn_defs.h
CastT	math_defs	math_defs.h
Ceil	math_defs	math_defs.h
ChannelAxpy	math_defs	math_defs.h
Clipboxes	nn_defs	nn_defs.h
ClipByValue	math_defs	math_defs.h
ConcatD	array_defs	array_defs.h
Const	const_defs	const_defs.h
Convolution	nn_defs	nn_defs.h
ConvolutionDepthwise	nn_defs	nn_defs.h
ConvTranspose	nn_defs	nn_defs.h
Cos	math_defs	math_defs.h
Cosh	math_defs	math_defs.h
Crop	image_defs	image_defs.h
CropAndResize	image_defs	image_defs.h
Data	array_defs	array_defs.h
DecodeBBox	nn_defs	nn_defs.h
DepthToSpace	array_defs	array_defs.h

算子名	所属分类	所在文件
Dequantize	array_defs	array_defs.h
DetectionPostprocessing	detection_defs	detection_defs.h
DynamicImageData	image_defs	image_defs.h
Eltwise	nn_defs	nn_defs.h
Equal	math_defs	math_defs.h
Exp	math_defs	math_defs.h
ExpandDims	array_defs	array_defs.h
Expm1	math_defs	math_defs.h
FakeQuantWithMinMaxVars	array_defs	array_defs.h
FakeQuantWithMinMaxVarsPerChannel	nn_defs	nn_defs.h
Fill	array_defs	array_defs.h
Flatten	array_defs	array_defs.h
Floor	math_defs	math_defs.h
FloorDiv	math_defs	math_defs.h
FloorMod	math_defs	math_defs.h
FractionalPooling	nn_defs	nn_defs.h
FSRDetectionOutput	detection_defs	detection_defs.h
FullyConnection	nn_defs	nn_defs.h
GatherNd	array_defs	array_defs.h
GatherV2D	array_defs	array_defs.h
Greater	math_defs	math_defs.h
GreaterEqual	math_defs	math_defs.h
HardSwish	nn_defs	nn_defs.h
ImageChannelSwap	image_defs	image_defs.h
ImageColorSpaceConvertion	image_defs	image_defs.h
ImageCrop	image_defs	image_defs.h
ImageData	image_defs	image_defs.h
ImageDataTypeConversion	image_defs	image_defs.h
ImagePadding	image_defs	image_defs.h
ImageResize	image_defs	image_defs.h

算子名	所属分类	所在文件
InstanceNorm	nn_defs	nn_defs.h
Interp	image_defs	image_defs.h
InvertPermutation	array_defs	array_defs.h
L2Normalize	math_defs	math_defs.h
LayerNorm	nn_defs	nn_defs.h
Less	math_defs	math_defs.h
LessEqual	math_defs	math_defs.h
Log	math_defs	math_defs.h
Log1p	math_defs	math_defs.h
LogicalAnd	math_defs	math_defs.h
LogicalNot	math_defs	math_defs.h
LogicalOr	math_defs	math_defs.h
LogicalXor	nn_defs	nn_defs.h
LogSoftmax	nn_defs	nn_defs.h
LRN	nn_defs	nn_defs.h
LSTM	nn_defs	nn_defs.h
MatMul	math_defs	math_defs.h
Maximum	math_defs	math_defs.h
Minimum	math_defs	math_defs.h
MirrorPad	array_defs	array_defs.h
Mul	math_defs	math_defs.h
Multinomial	random_defs	random_defs.h
Neg	math_defs	math_defs.h
NonMaxSuppression	image_defs	image_defs.h
Normalize	nn_defs	nn_defs.h
NotEqual	math_defs	math_defs.h
OneHot	array_defs	array_defs.h
Pack	array_defs	array_defs.h
Pad	array_defs	array_defs.h
PadV2	array_defs	array_defs.h
Permute	detection_defs	detection_defs.h

算子名	所属分类	所在文件
PoolingD	nn_defs	nn_defs.h
Pow	math_defs	math_defs.h
Power	math_defs	math_defs.h
PriorBox	nn_defs	nn_defs.h
PSROIPooling	nn_defs	nn_defs.h
Quantize	array_defs	array_defs.h
QuantizedConvolution	nn_defs	nn_defs.h
QuantizedConvolutionDepthwise	nn_defs	nn_defs.h
QuantizedFullyConnection	nn_defs	nn_defs.h
RandomNormalNoSeed	random_defs	random_defs.h
RandomShuffleNoSeed	random_defs	random_defs.h
RandomUniformNoSeed	random_defs	random_defs.h
Range	math_defs	math_defs.h
Rank	nn_defs	nn_defs.h
RealDiv	math_defs	math_defs.h
Reciprocal	math_defs	math_defs.h
ReduceAllD	math_defs	math_defs.h
ReduceMax	math_defs	math_defs.h
ReduceMean	nn_defs	nn_defs.h
ReduceMin	math_defs	math_defs.h
ReduceProdD	math_defs	math_defs.h
ReduceSum	math_defs	math_defs.h
Reshape	array_defs	array_defs.h
ResizeBilinear	image_defs	image_defs.h
ResizeBilinearV2	image_defs	image_defs.h
ResizeNearestNeighbor	image_defs	image_defs.h
ReverseSequence	array_defs	array_defs.h
Rint	math_defs	math_defs.h
ROIPooling	nn_defs	nn_defs.h
Round	math_defs	math_defs.h

算子名	所属分类	所在文件
Rsqrt	math_defs	math_defs.h
Scale	nn_defs	nn_defs.h
ScatterNd	nn_defs	nn_defs.h
SegmentMax	math_defs	math_defs.h
SegmentMean	math_defs	math_defs.h
SegmentMin	math_defs	math_defs.h
SegmentProd	math_defs	math_defs.h
SegmentSum	math_defs	math_defs.h
Select	array_defs	array_defs.h
Shape	array_defs	array_defs.h
ShuffleChannel	nn_defs	nn_defs.h
ShuffleChannelV2	nn_defs	nn_defs.h
Sign	math_defs	math_defs.h
Sin	math_defs	math_defs.h
Sinh	math_defs	math_defs.h
Size	array_defs	array_defs.h
Slice	array_defs	array_defs.h
Softmax	nn_defs	nn_defs.h
SpaceToBatchND	array_defs	array_defs.h
SpaceToDepth	array_defs	array_defs.h
SplitD	array_defs	array_defs.h
SplitV	array_defs	array_defs.h
Sqrt	math_defs	math_defs.h
Square	math_defs	math_defs.h
SquaredDifference	math_defs	math_defs.h
Squeeze	array_defs	array_defs.h
SSDDetectionOutput	detection_defs	detection_defs.h
StopGradient	nn_defs	nn_defs.h
StridedSlice	array_defs	array_defs.h
Sub	math_defs	math_defs.h
SVDF	nn_defs	nn_defs.h

算子名	所属分类	所在文件
Tan	math_defs	math_defs.h
Threshold	nn_defs	nn_defs.h
Tile	array_defs	array_defs.h
TopK	nn_defs	nn_defs.h
Unpack	array_defs	array_defs.h
Upsample	image_defs	image_defs.h

ge::op 命名空间下的算子定义所在目录： /ddk/ai_ddk_lib/include/graph/compatible/。

算子名	所属分类	所在文件
Add	math_defs	cpt_math_defs.h
Mul	math_defs	cpt_math_defs.h
Expm1	math_defs	cpt_math_defs.h
Ceil	math_defs	cpt_math_defs.h
Sin	math_defs	cpt_math_defs.h
Cos	math_defs	cpt_math_defs.h
Floor	math_defs	cpt_math_defs.h
Log1p	math_defs	cpt_math_defs.h
LogicalAnd	math_defs	cpt_math_defs.h
LogicalNot	math_defs	cpt_math_defs.h
Maximum	math_defs	cpt_math_defs.h
Minimum	math_defs	cpt_math_defs.h
Acosh	math_defs	cpt_math_defs.h
Asinh	math_defs	cpt_math_defs.h
Equal	math_defs	cpt_math_defs.h
Reciprocal	math_defs	cpt_math_defs.h
Sqrt	math_defs	cpt_math_defs.h
Square	math_defs	cpt_math_defs.h
ReduceAll	math_defs	cpt_math_defs.h
Cast	math_defs	cpt_math_defs.h
Sign	math_defs	cpt_math_defs.h

算子名	所属分类	所在文件
Cosh	math_defs	cpt_math_defs.h
Exp	math_defs	cpt_math_defs.h
FloorMod	math_defs	cpt_math_defs.h
GreaterEqual	math_defs	cpt_math_defs.h
Less	math_defs	cpt_math_defs.h
MatMul	math_defs	cpt_math_defs.h
RealDiv	math_defs	cpt_math_defs.h
Rint	math_defs	cpt_math_defs.h
Round	math_defs	cpt_math_defs.h
Rsqrt	math_defs	cpt_math_defs.h
Sinh	math_defs	cpt_math_defs.h
Sub	math_defs	cpt_math_defs.h
Range	math_defs	cpt_math_defs.h
Acos	math_defs	cpt_math_defs.h
Asin	math_defs	cpt_math_defs.h
Atanh	math_defs	cpt_math_defs.h
Log	math_defs	cpt_math_defs.h
LogicalOr	math_defs	cpt_math_defs.h
Neg	math_defs	cpt_math_defs.h
ReduceProd	math_defs	cpt_math_defs.h
ReduceSum	math_defs	cpt_math_defs.h
Tan	math_defs	cpt_math_defs.h
ArgMax	math_defs	cpt_math_defs.h
FloorDiv	math_defs	cpt_math_defs.h
Const	const_defs	cpt_const_defs.h
RandomUniform	random_defs	cpt_random_defs.h
Multinomial	random_defs	cpt_random_defs.h
Permute	detection_defs	cpt_detection_defs.h
Activation	nn_defs	cpt_nn_defs.h
BatchNorm	nn_defs	cpt_nn_defs.h
Convolution	nn_defs	cpt_nn_defs.h

算子名	所属分类	所在文件
Deconvolution	nn_defs	cpt_nn_defs.h
BiasAdd	nn_defs	cpt_nn_defs.h
Eltwise	nn_defs	cpt_nn_defs.h
LRN	nn_defs	cpt_nn_defs.h
ConvolutionDepthwise	nn_defs	cpt_nn_defs.h
FullConnection	nn_defs	cpt_nn_defs.h
Pooling	nn_defs	cpt_nn_defs.h
Scale	nn_defs	cpt_nn_defs.h
ShuffleChannel	nn_defs	cpt_nn_defs.h
Softmax	nn_defs	cpt_nn_defs.h
TopK	nn_defs	cpt_nn_defs.h
LogSoftmax	nn_defs	cpt_nn_defs.h
QuantizedConvolution	nn_defs	cpt_nn_defs.h
QuantizedFullConnection	nn_defs	cpt_nn_defs.h
QuantizedConvolutionDepthwise	nn_defs	cpt_nn_defs.h
BatchNormExt2	nn_defs	cpt_nn_defs.h
CropAndResize	image_defs	cpt_image_defs.h
ResizeBilinear	image_defs	cpt_image_defs.h
ResizeNearestNeighbor	image_defs	cpt_image_defs.h
Data	array_defs	cpt_array_defs.h
Concat	array_defs	cpt_array_defs.h
Reshape	array_defs	cpt_array_defs.h
Split	array_defs	cpt_array_defs.h
SplitV	array_defs	cpt_array_defs.h
Unpack	array_defs	cpt_array_defs.h
Flatten	array_defs	cpt_array_defs.h
Slice	array_defs	cpt_array_defs.h
ExpandDims	array_defs	cpt_array_defs.h
Gather	array_defs	cpt_array_defs.h
GatherNd	array_defs	cpt_array_defs.h

算子名	所属分类	所在文件
Pack	array_defs	cpt_array_defs.h
SpaceToDepth	array_defs	cpt_array_defs.h
StridedSlice	array_defs	cpt_array_defs.h
SpaceToBatchND	array_defs	cpt_array_defs.h
BatchToSpaceND	array_defs	cpt_array_defs.h
Tile	array_defs	cpt_array_defs.h
Size	array_defs	cpt_array_defs.h
Fill	array_defs	cpt_array_defs.h
InvertPermutation	array_defs	cpt_array_defs.h
ReverseSequence	array_defs	cpt_array_defs.h

2 算子 API 说明

算子 API 是算子的输入、输出以及属性的定义。具体说明请参见“5 API 参考”。

下面结合算子定义，讲解通用 IR API 的使用方式。

说明

- 算子输入不支持大于 4 维的输入，只支持 4 维及以下的输入。
- DDK 100.320.030.010 版本中 算子定义更改为 hiai::op::命名空间，原有的 ge::op 命名空间下的算子定义，移到./graph/compatible/。推荐用户使用新的 hiai::op::命名空间下的 IR API。
- 如果用户继续使用老的 ge::op 命名空间下的算子，需要将#include <op/all_ops.h>修改为：#include <compatible/all_ops.h>。
- 由于 ge::op 命名空间下的算子不再支持更多的扩展，如果用户使用两种命名空间下的算子，则需要同时包含 op/all_ops.h、compatible/all_ops.h。

2.1 输入

IR 算子 API 定义了算子的输入描述名称、输入的属性以及输入支持的数据类型。

根据输入的属性，又可将输入设置为可选输入或可变输入，分别以 OPTIONAL_INPUT 和 DYNAMIC_INPUT 来标识。OPTIONAL_INPUT 一般用在权重的配置上，当输入个数不固定时，使用 DYNAMIC_INPUT。通过以下几个例子具体说明。

例 1：Convolution 算子

```
REG_OP(Convolution)
.INPUT(x, TensorType({ DT_FLOAT }))
.INPUT(filter, TensorType({ DT_FLOAT }))
.OPTIONAL_INPUT(bias, TensorType({ DT_FLOAT }))
```

算子的一个权重 bias，是作为可选输入设置的，即 OPTIONAL_INPUT，OPTIONAL_INPUT 同 INPUT 调用算子 API 接口相同，即通过 set_input_接口配置。

```
auto conv_op= hiai::op::Convolution("convolution")
.set_input_x(data)
.set_input_filter(conv1_const_0)
.set_input_bias(conv1_const_1).
```

例 2：ConcatD 算子


```
REG_OP(ConcatD)
.DYNAMIC_INPUT(x, TensorType({ DT_FLOAT, DT_INT32 }))
.OUTPUT(y, TensorType({ DT_FLOAT, DT_INT32 }))
```

算子的输入 x 是作为可变输入设置的，即 DYNAMIC_INPUT，通过 create_dynamic_input_和 set_dynamic_input_接口设置。

```
auto fire2_concat = hiai::op::ConcatD("fire2/concat")
.create_dynamic_input_x(2)
.set_dynamic_input_x(1,fire2_relu_expand1x1)
.set_dynamic_input_x(2,fire2_relu_expand3x3)
```

其中，create_dynamic_input_x(2)的 x 表示算子输入名称，参数 2 表示可变输入个数；

set_dynamic_input_x(1,fire2_relu_expand1x1)的 x 表示算子输入名称，1 表示输入 index，默认从 1 开始，fire2_relu_expand1x1 表示输入 value。

2.2 输出

单算子接口类定义了算子的输出描述名称、输出的属性以及支持的数据类型。

根据其属性，又可将输出设置为 OUTPUT 、DYNAMIC_OUTPUT，当输出个数不固定时，使用 DYNAMIC_OUTPUT 标识。

如：SplitD 算子

```
REG_OP(SplitD)
.INPUT(x, TensorType({ DT_FLOAT, DT_INT8, DT_INT32, DT_BOOL }))
.DYNAMIC_OUTPUT(y, TensorType({ DT_FLOAT, DT_INT8, DT_INT32, DT_BOOL }))
```

算子的输出 y 是作为可变输出设置的，即 DYNAMIC_ OUTPUT，通过 create_dynamic_output_接口设置。

创建可变输出：

```
auto split_op = hiai::op::SplitD("split")
.set_input_x(data)
.create_dynamic_output_y(2)
.set_attr_split_dim(0)
```

可变输出使用：

```
auto acosh_op= hiai::op::Acosh("acosh")
.set_input_x(split_op.get_output(0))
```

其中，create_dynamic_output_y(2)的 y 表示算子输出名称，参数 2 表示可变输出个数；

set_input_x(split_op .get_output(0))的 split_op 表示输入 value，get_output(0)表示取构建的动态输出的第 1 个，create_dynamic_output_y 中的 y 和输出的序号拼接来的，序号从 1 开始。

2.3 属性

单算子接口类定义了算子的属性名称、属性类型、属性支持的数据类型、属性的默认值及其取值范围。根据其属性类型，又可将属性设置为 ATTR、REQUIRED_ATTR，对于必填属性，以 REQUIRED_ATTR 来标识。

如：ConcatD 算子

```
REG_OP(ConcatD)
.DYNAMIC_INPUT(x, TensorType({ DT_FLOAT, DT_INT32 }))
.OUTPUT(y, TensorType({ DT_FLOAT, DT_INT32 }))
.REQUIRED_ATTR(concat_dim, AttrValue::INT)
.ATTR(N, AttrValue::INT { 1 })
.OP_END()
```

该算子有一个必填属性 concat_dim，REQUIRED_ATTR 同 ATTR 调用算子属性 API 相同，即通过 set_attr_接口配置。

```
.set_attr_concat_dim(1)
.set_attr_N(2)
```

2.3.1 量化属性

IR API 提供量化因子，供用户提供自己训练出的量化因子。以 QuantizedConvolution 为例：

```
REG_OP(QuantizedConvolution)
.INPUT(x, TensorType ({ DT_FLOAT }))
.INPUT(filter, TensorType({ DT_FLOAT, DT_INT8 }))
.OPTIONAL_INPUT(bias, TensorType ({ DT_FLOAT, DT_INT32 }))
.OUTPUT(y, TensorType ({ DT_FLOAT }))
.REQUIRED_ATTR(strides, AttrValue::LIST_INT)
.ATTR(dilations, AttrValue::LIST_INT ({ 1, 1 }))
.ATTR(pads, AttrValue::LIST_INT ({ 0, 0, 0, 0 }))
.ATTR(pad_mode, AttrValue::STR { "SPECIFIC" })
.ATTR(groups, AttrValue::INT { 1 })
.ATTR(data_format, AttrValue::STR { "NCHW" })
.ATTR(x_quant_type, AttrValue::INT { 0 })
.ATTR(filter_quant_type, AttrValue::INT { 0 })
.ATTR(x_quant_scale, AttrValue::FLOAT { 1.0 })
.ATTR(x_quant_offset, AttrValue::INT { 0 })
.ATTR(filter_quant_scales, AttrValue::LIST_FLOAT({}))
.OP_END()
```

- 为支持量化新增 x_quant_type、filter_quant_type、x_quant_scale、x_quant_offset、filter_quant_scales 五个字段。
- 非量化卷积流程时，x_quant_type、filter_quant_type 设置为 0 或者不设。
- 量化卷积流程时，当前只支持 8bit 量化，x_quant_type、filter_quant_type 只能设置为 1。
- 输入 scale x_quant_scale 值必须大于 0。
- 权值 scale filter_quant_scales 个数必须大于等于 1，且值大于 0。
- 当前 filter_quant_scales 个数为 1 时，权值的 scale type 为 SCALER_TYPE,当 filter_quant_scales 个数大于 1 时，权值的 scale type 为 VECTOR_TYPE。

- 整网所有需要量化的算子，权值的 scale type 必须一致。
- 权值由用户自行量化，量化好的权值通过 w 输入传递，类型设置为 DT_INT8。
- 偏置由用户自行量化（如果存在），量化好的 bias 通过 b 输入传递，类型设置为 DT_INT32。

2.3.2 AIPP 属性

AIPP 支持通过 IR 定义的方式，生成带 AIPP 功能的 DaVinci 模型，相关的 IR 算子定义详见：/ddk/ai_ddk_lib/include/graph/op/image_defs.h。

以 ImageData 为例：

```
REG_OP(ImageData)
.INPUT(x, TensorType({ DT_UINT8 }))
.OUTPUT(y, TensorType({ DT_UINT8 }))
.REQUIRED_ATTR(input_format, AttrValue::STR)
.REQUIRED_ATTR(src_image_size_w, AttrValue::INT)
.REQUIRED_ATTR(src_image_size_h, AttrValue::INT)
.ATTR(image_type, AttrValue::STR { "JPEG" })
.OP_END()
```

- input_format：必填属性，原始图片的格式，目前支持 YUV420SP_U8, YUV422SP_U8, AYUV444_U8, YUYV_U8, YUV400_U8, XRGB8888_U8, ARGB8888_U8。
- src_image_size_w，src_image_size_h：必填属性，原始图片的长宽。
- image_type：必填属性，可接受格式为 JPEG, BT_601_NARROW, BT_601_FULL, BT_709_NARROW，默认值为 JPEG。

3 在线构建模型

3.1 构建模型 Demo

3.1.1 依赖条件

- 依赖工具：Android Studio 开发工具
- 源码包目录：
  ddk/app_sample/IR_model_demo/IR_Demo_Soure_Code.rar
  说明：在线样例包，Demo APK 完整的代码和工程文件。
- 其他依赖项：
  ddk/ddk/ai_ddk_lib/
  说明：APP 开发编译所依赖头文件以及库文件。
  组件：
  - include：工程依赖头文件（含 IR API 接口、算子定义等头文件）。
  - lib/lib64：APK 集成相关库文件。

3.1.2 Demo 运行步骤

步骤 1 解压源码包 IR_Demo_Soure_Code.rar;

步骤 2 使用 Android Studio 打开源码工程：IR_Demo_Soure_Code；

步骤 3 根据原生模型，修改样例文件 createModel.cpp 的参数和配置，如下操作；

1. 从原生模型中找到所有的单算子模型；
2. 构建网络单算子：调用“5.8 Graph 类接口”，构建单算子，并根据原生模型中算子之间的关系，进行算子级联;
3. 构建 graph，设置 graph 的输入、输出；
4. 构建 model，将 graph 对象添加到模型中;
5. 构建网络：调用“5.10 模型构建类接口”，构建模型生成文件;

步骤 4 使用 Android Studio 工具编译生成 APK;

步骤 5 将 APK 安装到手机上，运行这个 APP，生成 OM 模型。

----结束

3.2 构建 IR 模型

3.2.1 原生模型

以 squeezenet 网络为例，具体代码详见 jni 目录下的 c++样例文件 createModel.cpp。

此模型中包含以下几种算子：

- Data
- Const
- Convolution
- Activation
- PoolingD
- ConcatD
- Softmax

3.2.2 单算子构建

调用“5.8 Graph 类接口”，构建网络单算子，并根据原生模型中算子之间的关系，进行算子级联。

3.2.2.1 Data & Const

用户可以自行构造 Data 算子作为整个网络的输入，样例如下：

```
TensorDesc desc(Shape({in_channels, out_channels, h, w}), format,
datatype);
string data_name = op_name;
data = hiai::op::Data(data_name);
data.update_input_desc_x(desc);
```

权值等信息可以用 Const 算子作为算子的输入，样例如下：

```
hiai::op::Const conv10_const_1=
hiai::op::Const("conv10_filter").set_attr_value(weightList0[0]);
```

3.2.2.2 AIPP

说明

HiAI DDK V320 新增功能。

AIPP 算子包含 AIPP 输入算子和 AIPP 功能算子。

AIPP 输入算子：ImageData、DynamicImageData。

AIPP 功能算子：ImageCrop、ImageChannelSwap、ImageColorSpaceConvertion、ImageResize、ImageDataTypeConversion 和 ImagePadding。

静态/动态 AIPP

表3-1 AIPP 输入算子使用注意

算子名称	使用场景	是否能级联 AIPP 功能算子
ImageData	用于静态 AIPP，输入图片类型、长宽由 input_format，src_image_size_w 和 src_image_size_h 定义	能
DynamicImageData	用于动态 AIPP，输入图片类型、长宽不确定	不能

AIPP 的 IR 模型定义支持多输入场景，支持网络中定义多个 ImageData 算子，也支持一个 ImageData 算子之后，连接多组不同的 AIPP 功能算子。

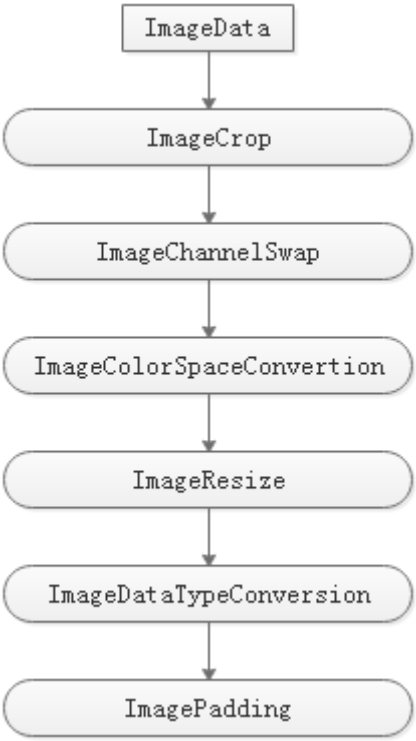
静态 AIPP 算子级联顺序

对于所有的 AIPP 功能，硬件仅支持固定的处理顺序，IR 构建时必须按图 3-1 顺序进行级联。

说明

一组 AIPP 功能算子之间，可以省略不必要的算子，但是不能插入其他类型的算子，也不能重复添加相同的 AIPP 功能算子。

图3-1 AIPP 算子级联顺序图



使用范例

假定需要对模型输入进行 Crop 处理，其 IR 代码实现如下：

```
// 步骤1. 构建ImageData节点
TensorDesc desc(Shape({n, channels, h, w}), format,  datatype);
hiai::op::ImageData data =
op::ImageData(data_name).set_attr_input_format("XRGB8888_U8").set_attr_sr
c_image_size_w(32).set_attr_src_image_size_h(32);
data.update_input_desc_x(desc);
// 步骤2. 构建AIPP功能算子cropOp和resizeOp
auto cropOp =
op::ImageCrop("crop").set_input_x(data).set_attr_load_start_pos_w(0).set_
attr_load_start_pos_h(0).set_attr_crop_size_w(16).set_attr_crop_size_h(16
);
auto resizeOp =
op::ImageResize("resize").set_input_x(cropOp).set_attr_resize_output_h(32
).set_attr_resize_output_w(32);
// 步骤3. 衔接其他功能算子，如pool op
auto poolOp =op:: Pooling ("pool"). set_input_x(resizeOp).
set_attr_data_mode(0).set_attr_pad_mode(4).set_attr_ceil_mode(1).set_attr
_mode(0).set_attr_pad(AttrValue::LIST_INT({ 0, 0, 0,
0})).set_attr_window(AttrValue::LIST_INT({ 3,
3})).set_attr_stride(AttrValue::LIST_INT({ 2,
2})).set_attr_global_pooling(0);
```

3.2.2.3 Convolution

根据算子定义，Convolution 算子有以下输入和属性：

```
REG_OP(Convolution)
.INPUT(x, TensorType({ DT_FLOAT }))
.INPUT(filter, TensorType({ DT_FLOAT }))
.OPTIONAL_INPUT(bias, TensorType({ DT_FLOAT }))
.OPTIONAL_INPUT(offset_w, TensorType({ DT_INT8}))
.OUTPUT(y, TensorType({ DT_FLOAT }))
.REQUIRED_ATTR(strides, AttrValue::LIST_INT)
.ATTR(dilations, AttrValue::LIST_INT ({ 1, 1 }))
.ATTR(pads, AttrValue::LIST_INT ({ 0, 0, 0, 0 }))
.ATTR(pad_mode, AttrValue::STR { "SPECIFIC" })
.ATTR(groups, AttrValue::INT { 1 })
.ATTR(data_format, AttrValue::STR { "NCHW" })
.ATTR(offset_x, AttrValue::INT { 0 })
.OP_END()
```

对于每个节点，根据网络模型中该节点的详细信息，调用单算子模型“5.8 Graph 类接口”来设置其输入和属性值。

网络中的第一个 Convolution 节点，输入包括一个 input 和两个 weight，分别通过 set_input_x、set_input_w、set_input_b 接口配置，同时因为输入 b 为可选输入，用户可根据需要选择是否设置。

模型中指定了属性 stride、num_output 等的取值，其他未指定属性值的属性则给予默认值，通过 set_attr_属性名接口设置。

```
auto conv0 = hiai::op::Convolution("conv0")
.set_input_x(data)
.set_input_filter(conv0_const_0)
.set_attr_strides(AttrValue::LIST_INT({2,2}))
.set_attr_dilations(AttrValue::LIST_INT({1,1}))
.set_attr_groups(1)
.set_attr_pads(AttrValue::LIST_INT({3, 3, 3, 3}))
.set_attr_pad_mode("SAME")
```

3.2.2.4 Activation

根据算子定义，Activation 算子有以下输入和属性：

```
REG_OP(Activation)
.INPUT(x, TensorType({ DT_INT8, DT_INT32, DT_FLOAT, DT_BOOL, DT_INT64 }))
.OUTPUT(y, TensorType({ DT_INT8, DT_INT32, DT_FLOAT, DT_BOOL, DT_INT64 }))
.ATTR(mode, AttrValue::INT { 1 })
.ATTR(coef, AttrValue::FLOAT { 0.0 })
.ATTR(negative_slope, AttrValue::FLOAT { 0.0 })
.OP_END()
```

对于每个节点，根据网络模型中该节点的详细信息，调用单算子模型“5.8 Graph 类接口”来设置其输入和属性值。

网络中的第一个 Activation 节点，输入包括一个 input，通过 set_input_x 接口配置。

模型中没有指定属性的取值，则通过 set_attr_属性名接口给予默认值。

```
auto relu_conv1 = hiai::op::Activation("relu_conv1")
.set_input_x(conv1)
.set_attr_coef(0.000000)
.set_attr_mode(1);
```

3.2.2.5 ConcatD

根据算子定义，Concat 算子有以下输入和属性，输入是可变输入：

```
REG_OP(ConcatD)
.DYNAMIC_INPUT(x, TensorType({ DT_FLOAT, DT_INT32 }))
.OUTPUT(y, TensorType({ DT_FLOAT, DT_INT32 }))
.REQUIRED_ATTR(concat_dim, AttrValue::INT)
.ATTR(N, AttrValue::INT { 1 })
.OP_END()
```

对于每个节点，根据网络模型中该节点的详细信息，调用单算子模型“5.8 Graph 类接口”来设置其输入和属性值。

网络中的第一个 Concat 节点，此算子配置了两个输入，通过 create_dynamic_input_x 接口对输入个数进行限制。

模型中没有指定属性的取值，则通过 set_attr_属性名接口给予默认值。

```
auto fire2_concat = hiai::op::ConcatD("fire2/concat")
.create_dynamic_input_x(2)
.set_dynamic_input_x(1,fire2_relu_expand1x1)
.set_dynamic_input_x(2,fire2_relu_expand3x3)
.set_attr_concat_dim(1);
```

3.2.2.6 PoolingD

根据算子定义，PoolingD 算子有以下输入和属性：

```
REG_OP(PoolingD)
.INPUT(x, TensorType({ DT_FLOAT }))
.OUTPUT(y, TensorType({ DT_FLOAT }))
.ATTR(mode, AttrValue::INT { 0 })
.ATTR(pad_mode, AttrValue::INT { 0 })
.ATTR(global_pooling, AttrValue::BOOL { false })
.ATTR(window, AttrValue::LIST_INT({ 1, 1 }))
.ATTR(pad, AttrValue::LIST_INT({ 0, 0,  0,0 }))
.ATTR(stride, AttrValue::LIST_INT({ 1, 1 }))
.ATTR(ceil_mode, AttrValue::INT { 0 })
.ATTR(data_mode, AttrValue::INT { 1 })
.OP_END()
```

对于每个节点，根据网络模型中该节点的详细信息，调用单算子模型“5.8 Graph 类接口”来设置其输入和属性值。

网络中的第一个 PoolingD 节点，输入包括一个 input，通过 set_input_x 接口配置。

模型中指定了属性 stride 的取值，其他未指定属性值的属性则给予默认值，通过 set_attr_属性名接口设置。

```
auto pool1 = op::Pooling("pool1").set_input_x(relu_conv1)
.set_attr_data_mode(0)
.set_attr_pad_mode(4)
.set_attr_ceil_mode(1)
.set_attr_mode(0)
.set_attr_pad(AttrValue::LIST_INT({ 0, 0, 0, 0}))
.set_attr_window(AttrValue::LIST_INT({ 3, 3}))
.set_attr_stride(AttrValue::LIST_INT({ 2, 2}))
.set_attr_global_pooling(false);
```

3.2.2.7 Softmax

根据算子定义，Softmax 算子有以下输入和属性：

```
REG_OP(Softmax)
.INPUT(x, TensorType { DT_FLOAT })
.OUTPUT(y, TensorType { DT_FLOAT })
.ATTR(axis, AttrValue::INT { 0 })
.ATTR(algo, AttrValue::INT { 1 })
```


对于每个节点，根据网络模型中该节点的详细信息，调用单算子模型“5.8 Graph 类接口”来设置其输入和属性值。

网络中的第一个 Softmax 节点，输入包括一个 input，通过 set_input_x 接口配置。

模型中没有指定属性的取值，则通过 set_attr_属性名接口给予默认值。

```
auto prob = op::Softmax("prob").set_input_x(pool10)
.set_attr_axis(1)
.set_attr_algo(1);
```

3.2.3 网络构建

Demo 示例中 Java 通过 JNI 调用 Native 代码，Native 代码调用 libhiai_ir.so 和 libhiai_ir_build.so 库的函数实现 IR 模型的构建，如图 3-2 所示，其中调用 buildSqueezentGraph 函数的详细步骤如图 3-3 所示。

代码实现在 DDK 包 IR_model_demo 工程的 Demo_Soure_Code\app\src\main\jni\createModel.cpp 中。文件中的 Java_com_huawei_hiaidemo_utils_ModelManager_createOmModel 函数为 Java 方法的 Native 代码。

图3-2 网络构建步骤

图3-3 调用 buildSqueezenetGraph 函数

4 离线构建模型

4.1 构建模型 Demo

4.1.1 依赖条件

- 依赖环境：Linux\Ubuntu
- 源码包目录：

  ddk/tools/tools_omg/IR_Model_Offline_Demo/

  说明：离线样例包。

  组件：makefile：自动化编译文件。

  squeezenet.cpp：IR 离线构建样例文件。

  squeezenet：IR 离线构建可执行文件。
- 其他依赖项：

  ddk/tools/tools_omg/

  说明：离线模型生成器，主要负责适配各种框架下的不同网络文件和权值文件，统一转化 IR 离线模型。

  组件：lib64：离线构建所依赖的库文件。

  ddk/ddk/ai_ddk_lib/include/

  说明：工程依赖的头文件（含 IR API 构建、算子定义等头文件）；

  工程依赖的部分头文件需自行下载，链接如下：

  https://android.googlesource.com/platform/external/libcxx/+/refs/heads/pie-release/include/

  https://android.googlesource.com/platform/external/libcxxabi/+/refs/heads/pie-release/include/

  存放位置参考 makefile 注释。
- 编译操作参考 makefile（ddk/tools/tools_omg/IR_Model_Offline_Demo/makefile）。

4.1.2 离线构建模型步骤

步骤 1 拷贝源码包 IR_Model_Offline_Demo。

步骤 2 参考离线构建样例 squeezenet.cpp，编写自己的 IR 模型生成文件，即：

- 从原生模型中找到所有的单算子模型；
- 构建网络单算子：调用“5.8 Graph 类接口”，构建单算子，并根据原生模型中算子之间的关系，进行算子级联;
- 构建 graph，设置 graph 的输入、输出；
- 构建 model，将 graph 对象添加到模型中；
- 网络构建：调用“5.10 模型构建类接口”，构建模型生成文件；

步骤 3 在 Linux\Ubuntu 环境下，编译自己的 IR 模型生成可执行文件，然后执行该可执行程序，生成 OM 模型，如下操作：

1. 链接离线构建依赖的动态库文件 *.so；

   命令：export LD_LIBRARY_PATH=路径（即依赖的动态库文件所在路径，如：ddk/tools/tools_omg/lib64）

2. 执行./squeezenet 生成 om

说明

- 编译依赖 Make 工具和 g++编译器，用户需自己安装，推荐 Make 4.1 和 g++；
- 编译命令：make squeezenet

----结束

4.2 构建 IR 模型

4.2.1 原生模型

以 squeezenet 网络为例，具体代码详见 squeezenet.cpp 样例。

此模型中包含以下几种算子：

- Data
- Const
- Convolution
- Activation
- PoolingD
- ConcatD
- Softmax

4.2.2 单算子构建

同在线构建模型的网络单算子模型构建，请参见“3.2.2 单算子构建”。

4.2.3 构建 model

1. 调用“5.8 Graph 类接口”，设置 graph 的输入、输出，构建 graph 对象；
2. 调用“5.9 Model 类接口”，将 graph 对象添加到模型中，构建 model。

构建 graph 对象：


```
std::vector<Operator> inputs{data};
std::vector<Operator> outputs{prob};
graph.SetInputs(inputs).SetOutputs(outputs);
```

构建 model：

```
ge::Model model("model", "model_v00001");
model.SetGraph(graph);
```

4.2.4 构建网络

调用“5.10 模型构建类接口”，完成模型生成。

```
domi::HiaiIrBuild ir_build;
domi::ModelBufferData om_model_buff;
ir_build.CreateModelBuff(model,om_model_buff);
bool ret = ir_build.BuildIRModel(model,om_model_buff);
if(ret){
ret = WriteToOMFile(om_model_buff,"./squeezenet.ir.offline.om");
}
ir_build.ReleaseModelBuff(om_model_buff);
```

4.2.5 生成模型

在 Linux\Ubuntu 环境下，经过编译，执行 IR 模型生成的可执行文件，最终生成.om 模型。

5 API 参考

华为 HiAI DDK V320 提供了安全易用的 IR 构图接口集合，用户可以调用这些接口构建网络模型，设置模型所包含的图、图内的算子、以及模型和算子的属性等。该集合划分为如下两大类型：

- 辅助（数据）类型

类型	主要功能	文件定义
NamedAttrs 类	属性的实例，按照 key-value 的格式存储。	attr_value.h、attributes_holder.h
AttrHolder 类	用于存放 Operator 的属性值。	
Tensor 类	用于设置/获取 Tensor 数据以及实现对 Tensor 的描述。	tensor.h
TensorDesc 类	用于描述 Tensor 的信息，包括 Shape、DataType、Format 等。	
Shape 类	用于 Tensor 的形状描述。	
缓存类	用于实现模型数据的序列化和反序列化功能。	buffer.h

- 接口类型

类型	主要功能	文件定义
Operator 类	用于存取算子属性、设置输入/输出。	operator.h
Operator 注册类	用于设置算子注册时的输入输出、校验属性、获取维度等。	operator_reg.h
Graph 类	用于设置图的 Input/Output、校验图的有效性。	graph.h
Model 类	用于存取模型版本号和属性以及模型内部 Graph。	model.h
模型构建类	用于模型构建。	hiai_ir_build.h

您可以在发布的 DDK 包“ddk/ai_ddk_lib/include”目录下，查看这些接口的定义文件。每个接口的定义可在对应的头文件中查看具体的描述。

5.1 属性类接口

该类接口在 attr_value.h、attributes_holder.h 中定义。

5.1.1 SetAttr

函数原型

GraphErrCodeStatus SetAttr(const string& name, const AttrValue& value);

功能说明

设置属性值。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称
value	输入	const AttrValue&	属性值

返回值

类型	描述
GraphErrCodeStatus	成功返回 GRAPH_SUCCESS， 否则，返回 GRAPH_FAILED

异常处理

无。

约束说明

无。

5.1.2 GetAttr

函数原型

GraphErrCodeStatus GetAttr(const string& name, AttrValue& value) const;

功能说明

获取属性值。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称
value	输出	AttrValue&	属性值

返回值

类型	描述
GraphErrCodeStatus	成功返回 GRAPH_SUCCESS， 否则，返回 GRAPH_FAILED

异常处理

无。

约束说明

无。

5.1.3 HasAttr

函数原型

bool HasAttr(const string& name) const;

功能说明

判断是否有对应名称的属性

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称

返回值

类型	描述
bool	true：有名称为 name 的属性 false：无名称为 name 的属性

异常处理

无。

约束说明

无。

5.1.4 DelAttr

函数原型

GraphErrCodeStatus DelAttr(const string& name);

功能说明

删除属性。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称

返回值

类型	描述
GraphErrCodeStatus	成功返回 GRAPH_SUCCESS， 否则，返回 GRAPH_FAILED

异常处理

无。

约束说明

无。

5.1.5 SetName

函数原型

void SetName(const std::string& name);

功能说明

设置属性的名称。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称

返回值

无。

异常处理

无。

约束说明

无。

5.1.6 GetName

函数原型

string GetName() const;

功能说明

获取属性的名称。

参数说明

无。

返回值

类型	描述
string	属性名称

异常处理

无。

约束说明

无。

5.1.7 GetItem

函数原型

AttrValue GetItem(const string& key) const;

功能说明

获取对应名称的属性值。

参数说明

参数名	输入/输出	类型	描述
key	输入	const string&	属性名称

返回值

类型	描述
AttrValue	获取到的属性值

异常处理

无。

约束说明

无。

5.1.8 SetValue

函数原型

GraphErrCodeStatus SetValue(std::initializer_list<DT>&& val);

GraphErrCodeStatus SetValue(const std::vector<DT>& val);

GraphErrCodeStatus SetValue(DT&& val);

GraphErrCodeStatus SetValue(const T& t);

GraphErrCodeStatus SetValue(const vector<T>& t);

功能说明

设置指定模板类型的参数值。

参数说明

参数名	输入/输出	类型	描述
val	输入	std::initializer_list<DT>&& 或 const std::vector<DT>& 或 DT&&	待设置的值
t	输入	const T& 或 const vector<T>&	待设置的值

返回值

类型	描述
GraphErrCodeStatus	GRAPH_SUCCESS：设置成功 GRAPH_FAILED：设置失败 GRAPH_PARAM_INVALID：设置的参数无效

异常处理

无。

约束说明

无。

5.1.9 GetValue

函数原型

GraphErrCodeStatus GetValue(std::vector<DT>& val) const;

GraphErrCodeStatus GetValue(DT& val) const;

GraphErrCodeStatus GetValue(T& t);

GraphErrCodeStatus GetValue(vector<T>& t);

功能说明

获取指定模板类型的参数值。

参数说明

参数名	输入/输出	类型	描述
val	输入	std::vector<DT>& 或 DT&	保存获取值的变量
t	输入	T& 或 vector<T>&	保存获取值的变量

返回值

类型	描述
GraphErrCodeStatus	GRAPH_SUCCESS：获取成功 GRAPH_FAILED：获取失败 GRAPH_PARAM_INVALID：获取的参数无效

异常处理

无。

约束说明

无。

5.1.10 CreateFrom

函数原型

static AttrValue CreateFrom(DT&& val);

static AttrValue CreateFrom(const T& val);

static AttrValue CreateFrom(const vector<T>& val);

static AttrValue CreateFrom(std::initializer_list<DT>&& val);

功能说明

创建指定模板类型的属性值对象。

参数说明

参数名	输入/输出	类型	描述
val	输入	或 DT&& 或 const T& 或 const vector<T>& 或 std::initializer_list<DT>&&	创建对象时设置的值

返回值

类型	描述
AttrValue	创建的属性值对象

异常处理

无。

约束说明

无。

5.1.11 GetValueType

函数原型

ValueType GetValueType() const;

功能说明

获取当前属性值的类型。

参数说明

无。

返回值

类型	描述
ValueType	返回对应的属性类型

异常处理

无。

约束说明

无。

5.1.12 IsEmpty

函数原型

bool IsEmpty() const;

功能说明

判断属性值是否设置。

参数说明

无。

返回值

类型	描述
bool	true：已设置属性值 false：未设置值

异常处理

无。

约束说明

无。

5.1.13 Copy

函数原型

AttrValue Copy() const;

功能说明

复制属性值，返回一个属性对象。

参数说明

无。

返回值

类型	描述
AttrValue	属性对象

异常处理

无。

约束说明

无。

5.1.14 operator==

函数原型

bool operator==(const AttrValue& other) const；

功能说明

重载“==”等于操作符。

参数说明

参数名	输入/输出	类型	描述
other	输入	const AttrValue&	AttrValue 类型的操作数的 const 引用。

返回值

类型	描述
bool	true：判断结果为相等 false：判断结果为不等

异常处理

无。

约束说明

无。

5.1.15 MutableTensor

函数原型

GraphErrCodeStatus MutableTensor(TensorPtr& tensor);

功能说明

获取可以修改的 tensor 引用。

参数说明

参数名	输入/输出	类型	描述
tensor	输出	TensorPtr&	tensor 引用。

返回值

类型	描述
GraphErrCodeStatus	成功返回 GRAPH_SUCCESS， 否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.1.16 MutableListTensor

函数原型

GraphErrCodeStatus MutableListTensor(vector<TensorPtr>& list_tensor);

功能说明

获取可以修改的 Tensor 列表的引用。

参数说明

参数名	输入/输出	类型	描述
list_tensor	输出	vector<TensorPtr>&	Tensor 列表的引用。

返回值

类型	描述
GraphErrCodeStatus	成功返回 GRAPH_SUCCESS， 否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.1.17 InitDefault

函数原型

void InitDefault();

功能说明

初始化默认值。

参数说明

无。

返回值

无。

异常处理

无。

约束说明

无

5.1.18 GetProtoOwner

函数原型

inline const ProtoMsgOwner& GetProtoOwner() const;

功能说明

获取 ProtoMsgOwner 对象。

参数说明

无。

返回值

类型	描述
const ProtoMsgOwner&	返回获取到的 ProtoMsgOwner 对象。

异常处理

无。

约束说明

无。

5.1.19 GetProtoMsg

函数原型

inline ProtoType* GetProtoMsg() const;

功能说明

获取 ProtoType 对象。

参数说明

无。

返回值

类型	描述
ProtoType*	返回获取到的 ProtoType 对象。

异常处理

无。

约束说明

无。

5.1.20 CopyValueFrom

函数原型

void CopyValueFrom(const GeIrProtoHelper<ProtoType>& other);

功能说明

拷贝对象。

参数说明

参数名	输入/输出	类型	描述
other	输入	const GeIrProtoHelper<ProtoType>&	被拷贝 ProtoType 对象。

返回值

无。

异常处理

无。

约束说明

无。

5.1.21 MoveValueFrom

函数原型

void MoveValueFrom(GeIrProtoHelper<ProtoType>&& other);

功能说明

移动对象。

参数说明

参数名	输入/输出	类型	描述
other	输入	GeIrProtoHelper<ProtoType>&&	被移动的 ProtoType 对象。

返回值

无。

异常处理

无。

约束说明

无。

5.2 Tensor 类接口

该类接口在 tensor.h 中定义。

5.2.1 GetTensorDesc

函数原型

TensorDesc GetTensorDesc() const;

功能说明

获取 Tensor 的描述符（TensorDesc）。

参数说明

无。

返回值

类型	描述
TensorDesc	返回当前 Tensor 的描述符。

异常处理

无。

约束说明

无。

5.2.2 MutableTensorDesc

函数原型

TensorDesc& MutableTensorDesc();

功能说明

获取当前 Tensor 的描述符，可修改。

参数说明

无。

返回值

类型	描述
TensorDesc&	当前 Tensor 的描述符，可修改。

异常处理

无。

约束说明

无。

5.2.3 SetTensorDesc

函数原型

GraphErrCodeStatus SetTensorDesc(const TensorDesc &tensorDesc);

功能说明

设置 Tensor 的描述符。

参数说明

参数名	输入/输出	类型	描述
tensorDesc	输入	const TensorDesc &	需设置的 Tensor 描述符。

返回值

类型	描述
GraphErrCodeStatus	设置成功返回 GRAPH_SUCCESS，否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.2.4 GetData

函数原型

const Buffer GetData() const;

功能说明

获取 Tensor 中的数据。

参数说明

无。

返回值

类型	描述
const Buffer	Tensor 中所存放的数据。

异常处理

无。

约束说明

无。

5.2.5 MutableData

函数原型

Buffer MutableData();

功能说明

获取 Tensor 中存放的数据。

参数说明

无。

返回值

类型	描述
Buffer	Tensor 中存放的数据。

异常处理

无。

约束说明

无。

5.2.6 SetData

函数原型

GraphErrCodeStatus SetData(std::vector<uint8_t> &&data);

GraphErrCodeStatus SetData(const std::vector<uint8_t> &data);

GraphErrCodeStatus SetData(const Buffer &data);

GraphErrCodeStatus SetData(const uint8_t *data, size_t size);

功能说明

向 Tensor 中设置数据。

参数说明

参数名	输入/输出	类型	描述
data	输入	std::vector<uint8_t> && 或 const std::vector<uint8_t> & 或 const Buffer & 或 const uint8_t *	需设置的数据。
size	输入	size_t	数据的长度，单位为字节。

返回值

类型	描述
GraphErrCodeStatus	设置成功返回 GRAPH_SUCCESS，否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.2.7 Clone

函数原型

Tensor Clone() const;

功能说明

拷贝 Tensor。

参数说明

无。

返回值

类型	描述
Tensor	返回拷贝的 Tensor 对象。

异常处理

无。

约束说明

无。

5.2.8 operator=

函数原型

Tensor& operator= (const Tensor &other);

功能说明

重载“=”赋值操作符。

参数说明

参数名	输入/输出	类型	描述
other	输入	const Tensor&	不可更改的 Tensor 对象的引用。

返回值

类型	描述
Tensor&	Tensor 对象的引用。

异常处理

无。

约束说明

无。

5.3 TensorDesc 类接口

该类接口在 tensor.h 中定义。

5.3.1 GetName

函数原型

string GetName() const;

功能说明

获取 Tensor 名称。

参数说明

无。

返回值

类型	描述
string	Tensor 名称。

异常处理

无。

约束说明

无。

5.3.2 SetName

函数原型

void SetName(const string& name);

功能说明

设置 Tensor 名称。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	需设置的 Tensor 名称。

返回值

无。

异常处理

无。

约束说明

无。

5.3.3 Update

函数原型

void Update(Shape shape, Format format = FORMAT_NCHW, DataType dt = DT_FLOAT);

功能说明

更新 TensorDesc 对象的 shape、format、datatype 属性。

参数说明

参数名	输入/输出	类型	描述
shape	输入	Shape	需刷新的 shape 对象。
format	输入	Format	需刷新的 format 对象，默认取值 FORMAT_NCHW。 关于 Format 数据类型的定义，请参见•Format。
dt	输入	DataType	需刷新的 datatype 对象，默认取值 DT_FLOAT。 关于 DataType 数据类型的定义，请参见•DataType。

返回值

无。

异常处理

无。

约束说明

无。

数据类型说明

- Format

```
enum Format {
      FORMAT NCHW = 0,        /**< NCHW */
      FORMAT NHWC,           /**< NHWC */
      FORMAT ND,             /**< Nd Tensor */
      ...
   };
```

- DataType

```
enum DataType {
      DT FLOAT = 0,          // float type
      DT FLOAT16 = 1,        // fp16 type
      DT INT8 = 2,           // int8 type
      DT UINT8 = 4,          // uint8 type
      DT INT32 = 3,          //
      DT INT64 = 9,          // int64 type
      DT BOOL = 12,          // bool type
      ...
   };
```


5.3.4 GetShape

函数原型

Shape GetShape() const;

功能说明

获取 TensorDesc 描述的 shape。

参数说明

无。

返回值

类型	描述
Shape	TensorDesc 描述的 shape。

异常处理

无。

约束说明

返回的 shape 为 const，不可修改。

5.3.5 MutableShape

函数原型

Shape& MutableShape();

功能说明

获取 TensorDesc 中可修改的 shape 引用。

参数说明

无。

返回值

类型	描述
Shape &	返回 TensorDesc 中的 shape 引用，可修改。

异常处理

无。

约束说明

无。

5.3.6 SetShape

函数原型

void SetShape(Shape shape);

功能说明

设置 TensorDesc 的 shape。

参数说明

参数名	输入/输出	类型	描述
shape	输入	Shape	需向 TensorDesc 设置的 shape 对象。

返回值

无。

异常处理

无。

约束说明

无。

5.3.7 GetFormat

函数原型

Format GetFormat() const;

功能说明

获取 TensorDesc 所描述的 Tensor 的 format 信息。

参数说明

无。

返回值

类型	描述
Format	TensorDesc 所描述的 Tensor 的 format 信息。

异常处理

无。

约束说明

无。

5.3.8 SetFormat

函数原型

void SetFormat(Format format);

功能说明

设置 TensorDesc 所描述的 Tensor 的 format 信息。

参数说明

参数名	输入/输出	类型	描述
format	输入	Format	需设置的 format 信息。

返回值

无。

异常处理

无。

约束说明

无。

5.3.9 GetDataType

函数原型

DataType GetDataType() const;

功能说明

获取 TensorDesc 所描述的 Tensor 的数据类型。

参数说明

无。

返回值

类型	描述
DataType	TensorDesc 所描述的 Tensor 的数据类型。

异常处理

无。

约束说明

无。

5.3.10 SetDataType

函数原型

void SetDataType(DataType dt);

功能说明

设置 TensorDesc 所描述的 Tensor 的数据类型。

参数说明

参数名	输入/输出	类型	描述
dt	输入	DataType	需设置的 dt 信息。

参数名	输入/输出	类型	描述
			关于 DataType，请参见•DataType。

返回值

无。

异常处理

无。

约束说明

无。

5.3.11 Clone

函数原型

TensorDesc Clone() const;

功能说明

拷贝 TensorDesc。

参数说明

无。

返回值

类型	描述
TensorDesc	拷贝的 TensorDesc 对象。

异常处理

无。

约束说明

无。

5.3.12 IsValid

函数原型

GraphErrCodeStatus IsValid();

功能说明

判断 tensor 对象是否有效。

参数说明

无。

返回值

类型	描述
GraphErrCodeStatus	有效返回 GRAPH_SUCCESS，否则返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.3.13 operator=

函数原型

TensorDesc& operator=(const TensorDesc& other);

TensorDesc& operator=(TensorDesc&& other);

功能说明

重载“=”赋值操作符。

参数说明

参数名	输入/输出	类型	描述
other	输入	const TensorDesc&	不可更改的 TensorDesc 对象的引用。

参数名	输入/输出	类型	描述
other	输入	TensorDesc&&	TensorDesc 对象的右值引用。

返回值

类型	描述
TensorDesc&	TensorDesc 对象的引用。

异常处理

无。

约束说明

无。

5.4 Shape 类接口

该类接口在 tensor.h 中定义。

5.4.1 GetDimNum

函数原型

size_t GetDimNum() const;

功能说明

获取 Shape 的维度个数。

参数说明

无。

返回值

类型	描述
size_t	Tensor Shape 的维度个数。

异常处理

无。

约束说明

无。

5.4.2 GetDim

函数原型

int64_t GetDim(size_t idx) const;

功能说明

获取 Shape 第 idx 维的长度。

参数说明

参数名	输入/输出	类型	描述
idx	输入	size_t	维度索引，索引从 0 开始。

返回值

类型	描述
int64_t	第 idx 维的长度。

异常处理

无

约束说明

无。

5.4.3 GetDims

函数原型

std::vector<int64_t> GetDims() const;

功能说明

获取 Shape 所有维度组成的向量。

参数说明

无。

返回值

类型	描述
std::vector<int64_t>	Shape 的所有维度组成的向量。

异常处理

无。

约束说明

无。

5.4.4 SetDim

函数原型

GraphErrCodeStatus SetDim(size_t idx, int64_t value);

功能说明

将 Shape 中第 idx 维度的值设置为 value。

参数说明

参数名	输入/输出	类型	描述
idx	输入	size_t	Shape 维度的索引，索引从 0 开始。
value	输入	int64_t	需设置的值。

返回值

类型	描述
GraphErrCodeStatus	设置成功返回 GRAPH_SUCCESS，否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.4.5 GetShapeSize

函数原型

int64_t GetShapeSize() const;

功能说明

获取 Shape 中所有 dim 的累乘结果。

参数说明

无。

返回值

类型	描述
int64_t	返回所有 dim 的累乘结果。

异常处理

无。

约束说明

无。

5.4.6 operator=

函数原型

Shape& operator=(const Shape& other);

Shape& operator=(Shape&& other);

功能说明

重载“=”赋值操作符。

参数说明

参数名	输入/输出	类型	描述
other	输入	const Shape&	不可更改的 shape 对象的引用。
other	输入	Shape&&	Shape 对象的右值引用。

返回值

类型	描述
Shape&	Shape 对象的引用。

异常处理

无。

约束说明

无。

5.5 缓存类接口

该类接口在 buffer.h 中定义。

5.5.1 ClearBuffer

函数原型

void ClearBuffer();

功能说明

清空 Buffer, 将保存数据的 Buffer 指针置为 null。

参数说明

无。

返回值

无。

异常处理

无。

约束说明

无

5.5.2 GetData

函数原型

```
std::uint8_t* GetData();
const std::uint8_t* GetData() const;
```

功能说明

获取 Buffer 中存放的数据的指针。

参数说明

无。

返回值

类型	描述
uint8_t*	返回 Buffer 中存放的数据的指针。

异常处理

异常场景	说明
Buffer 为空	返回空指针。

约束说明

无。

5.5.3 GetSize

函数原型

```
std::size_t GetSize() const;
```


功能说明

获取 Buffer 中存放的数据的大小。

参数说明

无。

返回值

类型	描述
size_t	返回 Buffer 中存放的数据的大小。

异常处理

异常场景	说明
Buffer 为空	返回数据大小为 0。

约束说明

无。

5.5.4 CopyFrom

函数原型

static Buffer CopyFrom(std::uint8_t* data, std::size_t bufferSize);

功能说明

根据 data 中指针所指向的地址，拷贝 bufferSize 大小的数据，并将数据地址存放到 Buffer 中。

参数说明

参数名	输入/输出	类型	描述
data	输入	uint8_t*	存放要拷贝的数据指针。
bufferSize	输入	size_t	拷贝数据的大小。

返回值

类型	描述
Buffer	用于存放拷贝数据的 Buffer 对象。

异常处理

无。

约束说明

无。

5.5.5 data

函数原型

std::uint8_t* data();

const std::uint8_t* data() const;

功能说明

获取 Buffer 中存放的数据的指针。

参数说明

无。

返回值

类型	描述
uint8_t*	返回 Buffer 中存放的数据的指针。

异常处理

异常场景	说明
Buffer 为空	返回空指针

约束说明

无。

5.5.6 size

函数原型

std::size_t size() const;

功能说明

获取 Buffer 中存放的数据的大小。

参数说明

无。

返回值

类型	描述
size_t	返回 Buffer 中存放的数据的大小。

异常处理

异常场景	说明
Buffer 为空	返回数据大小为 0。

约束说明

无。

5.5.7 clear

函数原型

void clear();

功能说明

清除 Buffer 中的数据。

参数说明

无。

返回值

类型	描述
void	清除 Buffer 中的数据。

异常处理

异常场景	说明
Buffer 为空	不处理。

约束说明

无。

5.5.8 operator=

函数原型

Buffer& operator=(const Buffer& other);

功能说明

重载“=”赋值操作符。

参数说明

参数名	输入/输出	类型	描述
other	输入	const Buffer&	Buffer 类型的操作数的 const 引用。

返回值

类型	描述
Buffer&	Buffer 类型的引用。

异常处理

无。

约束说明

无。

5.5.9 operator[]

函数原型

uint8_t operator[](size_t index) const;

功能说明

重载“[]”操作符。

参数说明

参数名	输入/输出	类型	描述
index	输入	size_t	索引

返回值

类型	描述
uint8_t	buffer 非空时：反馈索引为 index 处的 buffer 地址。 buffer 为空时：返回默认无效值 0xff。

异常处理

无。

约束说明

无。

5.6 Operator 类接口

该类接口在 operator.h 中定义。

5.6.1 GetName

函数原型

string GetName() const;

功能说明

获取算子名称。

参数说明

无。

返回值

类型	描述
string	算子名称。

异常处理

无。

约束说明

无。

5.6.2 SetInput

函数原型

Operator& SetInput(const string& dstName, const Operator& srcOprt);

Operator& SetInput(const string& dstName, const Operator& srcOprt, const string &name);

Operator& SetInput(int dstIndex, const Operator& srcOprt, int srcIndex);

功能说明

设置当前算子的 Input 信息。

参数说明

参数名	输入/输出	类型	描述
dstName	输入	const string&	当前算子输入边的目的节点名称。

参数名	输入/输出	类型	描述
srcOprt	输入	const Operator&	当前算子输入边的起始节点所在的源算子对象的引用。
name	输入	const string&	当前算子输入边的起始节点名称。
dstIndex	输入	int	当前算子输入边的目的节点的索引。
srcIndex	输入	int	当前算子输入边的起始节点的索引。

返回值

类型	描述
Operator&	当前被设置 Input 的算子对象引用。

异常处理

无。

约束说明

要求 Operator& srcOprt 只存在一个输出。

5.6.3 GetInputDesc

函数原型

TensorDesc GetInputDesc(const string& name) const;

TensorDesc GetInputDesc(uint32_t index) const;

功能说明

根据算子 Input 名称或 Input 索引获取 Input 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子 Input 名称。 无算子 Input 名称，则返回

参数名	输入/输出	类型	描述
			TensorDesc 默认构造的对象。
index	输入	uint32_t	算子 Input 索引。 无算子 Input 索引，则返回 TensorDesc 默认构造的对象。

返回值

类型	描述
TensorDesc	算子 Input 的 TensorDesc。

异常处理

无。

约束说明

无。

5.6.4 TryGetInputDesc

函数原型

bool TryGetInputDesc(const string& name, TensorDesc& tensorDesc) const;

功能说明

根据算子 Input 名称获取 Input 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	Input 名称。
tensorDesc	输出	TensorDesc&	TensorDesc 对象的引用。

返回值

类型	描述

类型	描述
bool	获取 TensorDesc 成功， 返回 true；否则，返回 false。

异常处理

无。

约束说明

无。

5.6.5 UpdateInputDesc

函数原型

GraphErrCodeStatus UpdateInputDesc(const string& name, const TensorDesc& tensorDesc);

功能说明

根据算子 Input 名称更新 Input 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子 Input 名称。
tensorDesc	输入	const TensorDesc&	TensorDesc 对象的引用。

返回值

类型	描述
GraphErrCodeStatus	更新 TensorDesc 成功，返回 GRAPH_SUCCESS；否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.6.6 GetOutputDesc

函数原型

TensorDesc GetOutputDesc(const string& name) const;

TensorDesc GetOutputDesc(uint32_t index) const;

功能说明

根据算子 Output 名称或 Output 索引获取 Output 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子 Output 名称。
index	输入	uint32_t	算子 Output 索引。 无算子 Output 索引，则返回 TensorDesc 默认构造的对象。

返回值

类型	描述
TensorDesc	获取 TensorDesc 成功，返回对应 TensorDesc， 否则，返回 TensorDesc 默认构造的对象。

异常处理

无。

约束说明

无。

5.6.7 UpdateOutputDesc

函数原型

GraphErrCodeStatus UpdateOutputDesc (const string& name, const TensorDesc& tensorDesc);

功能说明

根据算子 Output 名称更新 Output 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子 Output 名称。
tensorDesc	输入	const TensorDesc&	TensorDesc 对象的引用。

返回值

类型	描述
GraphErrCodeStatus	更新 TensorDesc 成功，返回 GRAPH_SUCCESS；否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.6.8 GetDynamicInputDesc

函数原型

TensorDesc GetDynamicInputDesc(const string& name, const unsigned int index) const;

功能说明

根据 name 和 index 的组合获取算子动态 Input 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子动态 Input 的名称。
index	输入	const unsigned int	算子动态 Input 编号，编号从 1 开始。

返回值

类型	描述
TensorDesc	获取 TensorDesc 成功，则返回算子动态 Input 的 TensorDesc；获取失败，则返回 TensorDesc 默认构造的对象。

异常处理

无。

约束说明

无。

5.6.9 UpdateDynamicInputDesc

函数原型

GraphErrCodeStatus UpdateDynamicInputDesc(const string& name, const unsigned int index, const TensorDesc& tensorDesc);

功能说明

根据 name 和 index 的组合更新动态 Input 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子动态 Input 的名称。
index	输入	const unsigned int	算子动态 Input 编号，编号从 1 开始。

参数名	输入/输出	类型	描述
tensorDesc	输入	const TensorDesc&	TensorDesc 对象的引用。

返回值

类型	描述
GraphErrCodeStatus	更新动态 Input 成功，返回 GRAPH_SUCCESS；否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.6.10 GetDynamicOutputDesc

函数原型

TensorDesc GetDynamicOutputDesc (const string& name, const unsigned int index) const;

功能说明

根据 name 和 index 的组合获取算子动态 Output 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子动态 Output 的名。
index	输入	const unsigned int	算子动态 Output 编号，编号从 1 开始。

返回值

类型	描述

类型	描述
TensorDesc	获取 TensorDesc 成功，则返回算子动态 Output 的 TensorDesc；获取失败，则返回 TensorDesc 默认构造的对象。

异常处理

无。

约束说明

无。

5.6.11 UpdateDynamicOutputDesc

函数原型

GraphErrCodeStatus UpdateDynamicOutputDesc (const string& name, const unsigned int index, const TensorDesc& tensorDesc);

功能说明

根据 name 和 index 的组合更新算子动态 Output 的 TensorDesc。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	算子动态 Output 的名称。
index	输入	const unsigned int	算子动态 Output 编号。
tensorDesc	输入	const TensorDesc&	TensorDesc 对象的引用。

返回值

类型	描述
GraphErrCodeStatus	更新动态 Output 成功，返回 GRAPH_SUCCESS，否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.6.12 SetAttr

函数原型

Operator& SetAttr(const string& name, AttrValue&& attrValue);

功能说明

设置算子属性的属性值。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称。
attrValue	输入	AttrValue&&	需设置的属性值。

返回值

类型	描述
Operator&	算子对象本身。

异常处理

无。

约束说明

无。

5.6.13 GetAttr

函数原型

GraphErrCodeStatus GetAttr(const string& name, AttrValue& attrValue) const;

功能说明

根据属性名称获取其属性值。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	属性名称。
attrValue	输入	AttrValue&	属性值。

返回值

类型	描述
GraphErrCodeStatus	获取属性值成功，返回 GRAPH_SUCCESS，否则返回 GRAPH_FAILED

异常处理

无。

约束说明

无。

5.7 Operator 注册类接口

注册算子类型以 REG_OP 为起始，以“.”链接 INPUT、OUTPUT、ATTR 等接口注册算子的输入、输出和属性信息，最终以 OP_END 接口结束。注册算子类型成功后，自动生成以算子类型名称命名的类。

例如：

```
REG_OP(Activation)
.INPUT(x, TensorType({    DT_FLOAT, DT_BOOL }))
.OUTPUT(y, TensorType({ DT_FLOAT, DT_BOOL }))
.ATTR(mode, AttrValue::INT { 1 })
.ATTR(coef, AttrValue::FLOAT { 0.0 })
.ATTR(negative_slope, AttrValue::FLOAT { 0.0 })
.OP_END()
```


Operator 注册类接口在 operator_hiai_reg.h 中定义，原有 DDK 100.320.030.010 版本注册算子接口在 compatible/operator_reg.h 中。已注册的算子及对应的头文件，请参见“1.3 支持的算子”。

5.7.1 REG_OP

函数原型

REG_OP(x)

功能说明

注册算子类型，同时自动生成算子类型的两个构造函数。

例如，注册算子的类型名称 Conv2D，可调用 REG_OP(Conv2D)接口，调用该接口后，定义了算子的类型名称 Conv2D，同时产生 Conv2D 的两个构造函数，其中，Conv2D(const string& name)需指定算子名称，Conv2D()使用默认算子名称，例如“Conv2D_XX”(编号 XX 唯一)。

```
class Conv2D : public Operator {
   typedef Conv2D  THIS TYPE;
public:
   explicit Conv2D(const string& name);
   explicit Conv2D();
}
```

参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，被注册算子的类型名称。

返回值

无。

异常处理

无。

约束说明

注册的算子类型名称需保持唯一，不能重复。

5.7.2 ATTR

函数原型

ATTR(x, default_value)

功能说明

注册算子属性，必须指定默认值，用户不设置算子对象的属性值时使用默认值。

注册算子属性成功后，自动生成算子属性的 3 个对外接口，用于获取属性的名称、获取属性的值、设置属性的值。

现以注册类型为 int64_t 的属性、类型为 int64_t 列表两种场景为例，说明所生成的算子属性接口：

- 调用 ATTR(mode, AttrValue::INT{1})接口，注册属性 mode，属性类型为 int64_t，默认值为 1。

  注册属性成功后，自动生成以下接口：

  ```
  static const string name attr mode();       // 返回属性的名称，即"mode"
  int64 t get attr mode() const;             // 返回 mode 属性的值
  _THIS_TYPE& set_attr_mode(int64_t v);       // 设置 mode 属性的值，返回算子对象本身
  ```

- 调用 ATTR(pad, AttrValue::LIST_INT{0, 0, 0, 0})接口，注册属性 pad，属性类型为 int64_t 列表，默认值为{0,0,0,0}。

  注册属性成功后，自动生成以下接口：

  ```
  static const string name attr pad();       // 返回属性的名称，即"pad"
  vector<int64 t> get attr pad() const;       // 返回属性 pad 的值
  _THIS_TYPE& set_attr_pad(vector<int64_t> v);// 设置属性 pad 的值，返回算子对象本身
  ```

参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子属性的名称。
default_value	输入	-	算子属性的值，根据不同类型指定默认值，支持的属性类型包括： • AttrValue::INT，属性类型为 int64_t • AttrValue::FLOAT，属性类型为 float • AttrValue::STR，属性类型为 string • AttrValue::BOOL，属性类型为 bool • AttrValue::TENSOR，属性类型为 Tensor • AttrValue::LIST_INT，属性类型为

参数名	输入/输出	类型	描述
			vector<int64_t>, int64_t 列表 • AttrValue::LIST_FLOAT，属性类型为 vector<float>, float 列表 • AttrValue::LIST_STR，属性类型为 vector<string>，string 列表 • AttrValue::LIST_BOOL，属性类型为 vector<bool>，bool 列表 • AttrValue::LIST_TENSOR，属性类型为 vector<Tensor>，Tensor 列表

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子属性名称需保持唯一，不能重复。

5.7.3 REQUIRED_ATTR

函数原型

REQUIRED_ATTR (x, type)

功能说明

注册算子属性，没有默认值，用户必须设置算子对象的属性值。

注册算子属性成功后，自动生成算子属性的 3 个对外接口，用于获取属性的名称、获取属性的值、设置属性的值。

例如，注册类型为 int64_t 的属性 mode，可调用 **REQUIRED_ATTR (mode, Int)**接口，注册算子属性成功后，会自动生成如下接口：

```
static const string name_attr_mode();         // 返回属性的名称，即“mode”
OpInt get_attr_mode() const;                 // 返回 mode 属性的值，OpInt 即 int64_t
_THIS_TYPE& set_attr_mode(const OpInt& v);    // 设置 mode 属性的值，返回 this 对象
```


参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子属性的名称。
type	输入	-	算子属性的类型，包括： • AttrValue::INT，属性类型为 int64_t • AttrValue::FLOAT，属性类型为 float • AttrValue::STR，属性类型为 string • AttrValue::BOOL，属性类型为 bool • AttrValue::TENSOR，属性类型为 Tensor • AttrValue::LIST_INT，属性类型为 vector<int64_t>, int64_t 列表 • AttrValue::LIST_FLOAT，属性类型为 vector<float>, float 列表 • AttrValue::LIST_STR，属性类型为 vector<string>，string 列表 • AttrValue::LIST_BOOL，属性类型为 vector<bool>，bool 列表 • AttrValue::LIST_TENSOR，属性类型为 vector<Tensor>，Tensor 列表

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子属性名称需保持唯一，不能重复。

5.7.4 INPUT

函数原型

INPUT (x, t)

功能说明

注册算子输入信息。

注册算子输入信息成功后，自动生成算子输入的相关接口，用于获取算子输入的名称、设置算子输入的对应描述等。

例如，注册算子输入 x，算子输入接收的数据类型为 TensorType{DT_FLOAT}，可调用**INPUT(x, TensorType{DT_FLOAT})**接口，注册算子输入成功后，自动生成以下相关接口：

```
static const string name_in_x();                    // 返回输入的名称，即"x"
_THIS_TYPE& set_input_x(Operator& v, const string& srcName);
                                                     // 指定输入 x 与算子对象 v 的输出 srcName 存在连接关系，返回算子对象本身
_THIS_TYPE& set_input_x(Operator& v);                // 指定输入 x 与算子对象 v 的索引 0 的输出存在连接关系，返回算子对象本身
TensorDesc get_input_desc_x();                       // 返回输入 x 对应的描述
GraphErrCodeStatus update input desc x(const TensorDesc& tensorDesc);
                                                     // 设置输入 x 对应的描述，包括 Shape、DataType、Format 等信息，GraphErrCodeStatus 即 uint32_t 类型，返回非 0 表示出错
```

参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子输入的名称
t	输入	-	算子输入接收的数据类型，可以是 TensorType 定义的一个或多个，如果多个，通过“,”隔离，例如： TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} 关于 TensorType 类，请参见 TensorType 类说明。

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子输入名称需保持唯一，不能重复。

TensorType 类说明

TensorType 类用以定义输入或者输出支持的数据类型，TensorType 提供以下接口指定支持的数据类型：

- TensorType(DataType dt)：指定仅支持一个数据类型；
- TensorType(std::initializer_list<DataType> types)：指定支持多个数据类型；
- static TensorType ALL()：指定支持所有数据类型；
- static TensorType FLOAT()：指定支持 DT_FLOAT 和 DT_FLOAT16 数据类型。

5.7.5 OPTIONAL_INPUT

函数原型

OPTIONAL_INPUT(x, t)

功能说明

注册可选算子输入信息。

注册可选算子输入信息成功后，自动生成算子输入的相关接口，用于获取算子输入的名称、设置算子输入的对应描述等。

例如，注册算子输入 b，算子输入接收的数据类型为 TensorType{DT_FLOAT}，可调用 OPTIONAL_INPUT(b, TensorType{DT_FLOAT})接口，注册算子输入成功后，自动生成以下相关接口：

```
static const string name_in_b();             // 返回输入的名称，即“b”
_THIS_TYPE& set_input_b(Operator& v, const string& srcName);
                                        // 指定输入 b 与算子对象 v 的输出 srcName 存在连接关系，返回算子对象本身
_THIS_TYPE& set_input_b(Operator& v);         // 指定输入 b 与算子对象 v 的索引 0 的输出存在连接关系，返回算子对象本身
TensorDesc get_input_desc_b();               // 返回输入 b 对应的描述
GraphErrCodeStatus update_input_desc_b(const TensorDesc& tensorDesc);
                                        // 设置输入 b 对应的描述，包括 Shape、DataType、Format 等信息
```

参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子输入的名称。
t	输入	-	算子输入接收的数据类型，可以是 TensorType 定义的一个或多个，如果多个，通过“,”隔离，例如： TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} 关于 TensorType 类，请参见

参数名	输入/输出	类型	描述
			“TensorType 类说明”。

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子输入名称需保持唯一，不能重复。

5.7.6 OPTIONAL_OUTPUT

函数原型

OPTIONAL_OUTPUT(x, t)

功能说明

注册可选算子输出信息。

注册可选算子输出信息成功后，自动生成算子输出的相关接口，用于获取算子输出的名称、获取算子输出的对应描述等。

例如，注册算子输出 b，算子输出接收的数据类型为 TensorType{DT_FLOAT}，可调用 OPTIONAL_OUTPUT(b, TensorType{DT_FLOAT})接口，注册算子输出成功后，自动生成以下相关接口：

```
static const string name_out_b();              // 返回输出的名称，即“b”
TensorDesc get_output_desc_b();                // 返回输出 b 对应的描述
GraphErrCodeStatus update_output_desc_b(const TensorDesc& tensorDesc);
                                               // 设置输出 b 对应的描述，包括 Shape、DataType、
Format 等信息
```

参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子输入的名称。
t	输入	-	算子输入接收的数据类型，可以是 TensorType 定义的一个或多个，如果多个，通过“,”隔离，例如：

参数名	输入/输出	类型	描述
			TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} 关于 TensorType 类，请参见“TensorType 类说明”。

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子输出名称需保持唯一，不能重复。

5.7.7 DYNAMIC_INPUT

函数原型

DYNAMIC_INPUT (x, t)

功能说明

注册动态算子输入信息。

注册动态算子输入信息成功后，自动生成算子输入的相关接口，用于创建动态输入、设置算子输入的对应描述等。

例如，注册动态算子输入 d，算子输入接收的数据类型为 TensorType{DT_FLOAT}，可调用 DYNAMIC_INPUT(d, TensorType{DT_FLOAT})接口，注册动态算子输入成功后，自动生成以下相关接口：

```
_THIS_TYPE& create_dynamic_input_d(unsigned int num);
                                        // 创建动态输入 d，包括 num 个输入
TensorDesc get_dynamic_input_desc_d(unsigned int index);
                                        // 返回动态输入 d 第 index 个描述，包括 Shape、
DataType、Format 等信息
GraphErrCodeStatus update_dynamic_input_desc_d(unsigned int index, const
TensorDesc& tensorDesc);                    // 更新动态输入 d 的第 index 个描述
_THIS_TYPE& set_dynamic_input_d(unsigned int dstIndex, Operator &v);
                                        // 指定输入 d 的第 dstIndex 个输入与算子对象 v 的索引 0
的输出存在连接关系，返回算子对象本身
_THIS_TYPE& set_dynamic_input_d(unsigned int dstIndex, Operator &v, const string
&srcName);                    //指定动态输入 d 的第 dstIndex 个输入与算子对象 v 的输出 srcName
存在连接关系，返回算子对象本身
```


参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子输入的名称
t	输入	-	算子输入接收的数据类型，可以是TensorType 定义的一个或多个，如果多个，通过“,”隔离，例如： TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} 关于 TensorType 类，请参见“TensorType 类说明”。

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子输入名称需保持唯一，不能重复。

5.7.8 OUTPUT

函数原型

OUTPUT (x, t)

功能说明

注册算子输出信息。

注册算子输出信息成功后，自动生成算子输出的相关接口，用户获取算子输出的名称、获取算子输出的描述、设置算子输出的描述。

例如，注册算子输出 y，算子输出接收的数据类型为 TensorType{DT_FLOAT}，可调用 OUTPUT(y, TensorType{DT_FLOAT})接口，注册算子输出成功后，自动生成以下相关接口。

```
static const string name out y();            // 返回输出的名称，即“y”
TensorDesc get output desc y();              // 返回输出 y 对应的描述
GraphErrCodeStatus update output desc y(const TensorDesc& tensorDesc); );
                                      // 设置输出 y 对应的描述，包括 Shape、DataType、
Format 等信息
```


参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子输出的名称。
t	输入	-	算子输出接收的数据类型，可以是TensorType 定义的一个或多个，如果多个，通过“,”隔离，例如： TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} 关于 TensorType 类，请参见“TensorType 类说明”。

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子输出名称需保持唯一，不能重复。

5.7.9 DYNAMIC_OUTPUT

函数原型

DYNAMIC_OUTPUT (x, t)

功能说明

注册动态算子输出信息。

注册动态算子输出信息成功后，自动生成动态算子输出的相关接口，包括用于创建动态输出、设置算子输出的对应描述等

例如，注册动态算子输出 d，算子输出接收的数据类型为 TensorType{DT_FLOAT}，可调用 DYNAMIC_OUTPUT (d, TensorType{DT_FLOAT})接口，注册动态算子输出成功后，自动生成以下相关接口。

```
 THIS TYPE& create dynamic output d(unsigned int num);
                                        // 创建动态输出 d，包括 num 个输出
TensorDesc get dynamic output desc d(unsigned int index);
                                        // 返回动态输出 d 第 index 个描述，包括 Shape、
DataType、Format 等信息
```



```
GraphErrCodeStatus update_dynamic_output_desc_d(unsigned int index, const
TensorDesc& tensorDesc);                    // 更新动态输出 d 的第 index 个描述
```

参数说明

参数名	输入/输出	类型	描述
x	输入	-	宏参数，算子输出的名称。
t	输入	-	算子输出接收的数据类型，可以是 TensorType 定义的一个或多个，如果多个，通过“,”隔离，例如： TensorType{DT_FLOAT} TensorType({DT_FLOAT, DT_INT8} 关于 TensorType 类，请参见“TensorType 类说明”。

返回值

无。

异常处理

无。

约束说明

对于同一个算子，注册的算子输出名称需保持唯一，不能重复。

5.7.10 OP_END

函数原型

OP_END ()

功能说明

结束算子注册。

参数说明

无。

返回值

无。

异常处理

无。

约束说明

无。

5.7.11 DECLARE_INFERFUNC

函数原型

DECLARE_INFERFUNC (op_name, func_name)

功能说明

声明算子的 infershape 处理接口。

参数说明

参数名	输入/输出	类型	描述
op_name	输入	-	待获取 shape 信息的算子名称
func_name	输入	-	用来获取算子 shape 的 infershape 函数接口名称

返回值

无

异常处理

无。

约束说明

无。

5.7.12 IMPLEMT_INFERFUNC

函数原型

IMPLEMT_INFERFUNC (op_name, func_name)

功能说明

实现算子 infershape 函数。

参数说明

参数名	输入/输出	类型	描述
op_name	输入	-	待获取 shape 信息的算子名称
func_name	输入	-	用来获取算子 shape 的 infershape 函数接口名称

返回值

类型	描述
GraphErrCodeStatus	获取特定算子 shape 信息的操作状态返回值

异常处理

无。

约束说明

无。

5.7.13 DECLARE_VERIFIER

函数原型

DECLARE_VERIFIER (op_name, func_name)

功能说明

算子功能验证函数声明

参数说明

参数名	输入/输出	类型	描述
op_name	输入	-	待验证的算子名称
func_name	输入	-	用来验证算子信息的函数接口

返回值

类型	描述
GraphErrCodeStatus	获取特定算子 shape 信息的操作状态返回值

异常处理

无。

约束说明

无。

5.7.14 IMPLEMT_VERIFIER

函数原型

IMPLEMT_VERIFIER (op_name, func_name)

功能说明

算子功能验证接口的实现

参数说明

参数名	输入/输出	类型	描述
op_name	输入	-	待验证的算子名称
func_name	输入	-	用来验证算子信息的函数接口

返回值

无。

异常处理

无。

约束说明

无。

5.7.15 GET_INPUT_SHAPE

函数原型

GET_INPUT_SHAPE(op, name)

功能说明

获取算子 Input 的 shape。

参数说明

参数名	输入/输出	类型	描述
op	输入	-	算子对象。
name	输入	-	算子 Input 的名称。

返回值

类型	描述
-	算子 Input 对应的 shape。

返回值

无。

异常处理

无。

约束说明

无。

5.7.16 GET_DYNAMIC_INPUT_SHAPE

函数原型

GET_DYNAMIC_INPUT_SHAPE(op, name, index)

功能说明

获取算子动态 Input 的 shape。

参数说明

参数名	输入/输出	类型	描述
op	输入	-	算子对象。
name	输入	-	算子的动态 Input 的名称。
index	输入	-	算子的动态 Input 的索引。

返回值

类型	描述
-	算子动态 Input 对应的 shape。

异常处理

无。

5.7.17 SET_OUTPUT_SHAPE

函数原型

SET_OUTPUT_SHAPE(op, name, shape)

功能说明

设置 Output 算子的 shape。

参数说明

参数名	输入/输出	类型	描述
op	输入	-	算子对象。
name	输入	-	算子 Output 的名称。
shape	输入	-	要设置的算子 Output 的 shape。

返回值

无。

异常处理

无。

5.7.18 SET_DYNAMIC_OUTPUT_SHAPE

函数原型

SET_DYNAMIC_OUTPUT_SHAPE(op, name, index, shape)

功能说明

设置动态算子 Output 的 shape。

参数说明

参数名	输入/输出	类型	描述
op	输入	-	算子对象。
name	输入	-	算子动态 Output 的名称。
index	输入	-	要设置的算子动态 Output 的索引。
shape	输入	-	要设置的算子动态 Output 的 shape。

返回值

无。

异常处理

无。

5.7.19 GET_ATTR

函数原型

GET_ATTR(op, name, type, val)

功能说明

获取算子的属性值。

参数说明

参数名	输入/输出	类型	描述

参数名	输入/输出	类型	描述
op	输入	-	算子对象。
name	输入	-	算子属性名称。
type	输入	-	要获取的算子属性值类型。
val	输入	-	要获取的算子属性值。

返回值

类型	描述
-	算子属性值。

异常处理

无。

5.8 Graph 类接口

该类接口在 graph.h 中定义。

5.8.1 SetInputs

函数原型

Graph& SetInputs(std::vector<Operator>& inputs);

功能说明

设置 Graph 内的输入算子。

参数说明

参数名	输入/输出	类型	描述
inputs	输入	std::vector<Operator>&	Graph 内的输入算子。

返回值

类型	描述
Graph&	返回调用者本身。

异常处理

无。

约束说明

无。

5.8.2 SetOutputs

函数原型

Graph& SetOutputs(std::vector<Operator>& outputs);

功能说明

设置 Graph 关联的输出算子。

参数说明

参数名	输入/输出	类型	描述
outputs	输入	std::vector<Operator>&	与 Graph 关联的输出算子。

返回值

类型	描述
Graph&	返回调用者本身。

异常处理

无。

约束说明

无。

5.8.3 IsValid

函数原型

bool IsValid() const;

功能说明

判断 Graph 对象是否有效。

参数说明

无。

返回值

类型	描述
bool	true：构建 Graph 对象有效，非空。 false：构建 Graph 对象无效，空指针。

异常处理

无。

约束说明

无。

5.8.4 AddOp

函数原型

GraphErrCodeStatus AddOp(ge::Operator& op);

功能说明

在 Graph 中增加算子。

参数说明

参数名	输入/输出	类型	描述
op	输入	ge::Operator&	需增加的算子。

返回值

类型	描述
GraphErrCodeStatus	GRAPH_SUCCESS：操作成功 GRAPH_FAILED： 操作失败

异常处理

无。

约束说明

无。

5.8.5 FindOpByName

函数原型

ge::Operator FindOpByName(const string& name) const;

功能说明

根据算子的名称，返回 Graph 中的算子实例。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	需指定的算子名称。

返回值

类型	描述
ge::Operator	如果 Graph 中包含对应名称算子，返回 Graph 中的这个算子；否则返回 type 为“NULL”的算子。

异常处理

无。

约束说明

无。

5.8.6 CheckOpByName

函数原型

GraphErrCodeStatus CheckOpByName(const string& name) const;

功能说明

在 Graph 中检查指定名称的算子是否存在。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	需指定的算子名称。

返回值

类型	描述
GraphErrCodeStatus	GRAPH_SUCCESS： 指定名称的算子存在 GRAPH_FAILED：指定名称的算子不存在

异常处理

无。

约束说明

无。

5.8.7 GetAllOpName

函数原型

GraphErrCodeStatus GetAllOpName(std::vector<string>& opName) const ;

功能说明

返回 Graph 中所有算子的名称。

参数说明

参数名	输入/输出	类型	描述
opName	输出	std::vector<string>&	返回 Graph 中的所有算子的名称。

返回值

类型	描述
GraphErrCodeStatus	GRAPH_SUCCESS：操作成功 GRAPH_FAILED： 操作失败

异常处理

无。

约束说明

无。

5.9 Model 类接口

该类接口在 model.h 中定义。

5.9.1 SetName

函数原型

void SetName(const string& name);

功能说明

设置模型的名称。

参数说明

参数名	输入/输出	类型	描述
name	输入	const string&	模型名称。

返回值

无。

异常处理

无。

约束说明

无。

5.9.2 GetName

函数原型

string GetName() const;

功能说明

获取模型的名称。

参数说明

无。

返回值

类型	描述
string	返回模型的名称。

异常处理

无。

约束说明

无。

5.9.3 SetVersion

函数原型

void SetVersion(uint32_t version)

功能说明

设置模型的版本号。

参数说明

参数名	类型	描述
version	uint32_t	模型的版本号。

返回值

无。

异常处理

无。

约束说明

无。

5.9.4 GetVersion

函数原型

uint32_t GetVersion() const;

功能说明

获取模型的版本号。

参数说明

无。

返回值

类型	描述
uint32_t	返回模型的版本号。

异常处理

无。

约束说明

无。

5.9.5 SetPlatformVersion

函数原型

void SetPlatformVersion(string version);

功能说明

设置用户自定义模型的版本号。

参数说明

参数名	类型	描述
version	string	用户自定义模型的版本号。

返回值

无。

异常处理

无。

约束说明

无。

5.9.6 GetPlatformVersion

函数原型

std::string GetPlatformVersion() const;

功能说明

获取用户自定义模型的版本号，版本号是常量。

参数说明

无。

返回值

类型	描述
string	模型版本号。

异常处理

无。

约束说明

无。

5.9.7 GetGraph

函数原型

Graph GetGraph() const;

功能说明

获取模型中的 Graph 对象。

参数说明

无。

返回值

类型	描述
Graph	模型中的 Graph 对象。

异常处理

无。

约束说明

无。

5.9.8 SetGraph

函数原型

void SetGraph(const Graph& graph);

功能说明

设置模型的 Graph 对象。

参数说明

参数名	输入/输出	类型	描述
graph	输入	const Graph&	graph 对象。

返回值

无。

异常处理

无。

约束说明

无。

5.9.9 Save

函数原型

GraphErrCodeStatus Save(Buffer& buffer) const;

功能说明

将模型对象序列化。

参数说明

参数名	输入/输出	类型	描述
buffer	输入	Buffer&	序列化输出的对象的引用。

返回值

类型	描述
GraphErrCodeStatus	序列化成功，返回 GRAPH_SUCCESS，否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.9.10 Load

函数原型

static GraphErrCodeStatus Load(const uint8_t* data, size_t len, Model& model);

功能说明

加载序列化数据，反序列化构建模型对象。

参数说明

参数名	输入/输出	类型	描述
data	输入	const uint8_t *	序列化数据指针。
len	输入	size_t	序列化数据长度。
model	输出	Model &	承载反序列化后的模型对象。

返回值

类型	描述
GraphErrCodeStatus	执行成功，返回 GRAPH_SUCCESS，否则，返回 GRAPH_FAILED。

异常处理

无。

约束说明

无。

5.9.11 IsValid

函数原型

bool IsValid() const;

功能说明

判断模型中 Graph 对象是否有效，如果 Graph 对象为空指针，则无效。

参数说明

无。

返回值

类型	描述
bool	Model 中 Graph 对象是否有效，true 表示有效，false 表示无效。

异常处理

无。

约束说明

无。

5.10 模型构建类接口

在 hiai_ir_build.h 中提供了此类接口的声明。

模型构建时的接口调用顺序如下：

CreateModelBuff-->BuildIRModel-->ReleaseModelBuff

5.10.1 CreateModelBuff

函数原型

bool CreateModelBuff(ge::Model& irModel,ModelBufferData& output);

功能说明

创建模型 Buffer。

参数说明

参数名	输入/输出	类型	描述
irModel	输入	ge::Model&	模型对象。

output	输出	ModelBufferData&	离线模型结构体对象。
			struct ModelBufferData
			{
			void* data;
			uint32_t length;
			};

返回值

类型	描述
bool	true：创建模型 Buffer 成功。 false：创建模型 Buffer 失败。

异常处理

无。

约束说明

无。

5.10.2 CreateModelBuff

函数原型

bool CreateModelBuff(ge::Model& irModel, ModelBufferData& output, uint32_t customSize);

功能说明

按指定大小创建模型 Buffer。

参数说明

参数名	输入/输出	类型	描述
irModel	输入	ge::Model&	模型对象。

output	输出	ModelBufferData&	离线模型结构体对象。
			struct ModelBufferData
			{
			void* data;
			uint32_t length;
			};

customSize	输入	uint32_t	用户指定 Buffer 大小，单位字节，必须为非负整数，建议范围 200M 以内，当 customSize 设置为 0 时，接口内部根据 irModel 自动计算合适的 Buffer 大小

返回值

类型	描述
bool	true：创建模型 Buffer 成功。 false：创建模型 Buffer 失败。

异常处理

无。

约束说明

本接口为 HiAI DDK V320 新增。

5.10.3 BuildIRModel

函数原型

bool BuildIRModel(ge::Model& irModel,ModelBufferData& output);

功能说明

离线模型构建接口，输入模型对象，输出离线模型。

参数说明

参数名	输入/输出	类型	描述
irModel	输入	ge::Model&	模型对象。

output	输出	ModelBufferData&	离线模型结构体对象。
			struct ModelBufferData
			{
			void* data;
			uint32_t length;
			};

返回值

类型	描述
bool	true：构建模型成功。 false：构建模型失败。

异常处理

无。

约束说明

无。

5.10.4 ReleaseModelBuff

函数原型

void ReleaseModelBuff(ModelBufferData& output);

功能说明

释放模型 Buffer。

参数说明

参数名	输入/输出	类型	描述
output	输出	ModelBufferData&	离线模型结构体对象。

参数名	输入/输出	类型	描述

			struct ModelBufferData
			{
			void* data;
			uint32_t length;
			};

返回值

无。

异常处理

无。

约束说明

无。

