# TensorRT基础

# hello

学习使用TensorRT-CPP的API构建模型，并进行编译的流程

**重点提炼：**

1. 必须使用createNetworkV2，并指定为1（表示显性batch）。createNetwork已经废弃，非显性batch官方不推荐。这个方式直接影响推理时enqueue还是enqueueV2
2. builder、config等指针，记得释放，否则会有内存泄漏，使用ptr->destroy()释放
3. markOutput表示是该模型的输出节点，mark几次，就有几个输出，addInput几次就有几个输入。这与推理时相呼应
4. workspaceSize是工作空间大小，某些layer需要使用额外存储时，不会自己分配空间，而是为了内存复用，直接找tensorRT要workspace空间。指的这个意思

# hello

5. 一定要记住，保存的模型只能适配编译时的trt版本、编译时指定的设备。也只能保证在这种配置下是最优的。如果用trt跨不同设备执行，有时候可以运行，但不是最优的，也不推荐

# inference

这里主要讲推理，我们关注的基础中的基础对吗

**重点提炼：**

1. bindings是tensorRT对输入输出张量的描述，bindings = input-tensor + output-tensor。比如input有a，output有b, c, d，那么bindings = [a, b, c, d]，bindings[0] = a，bindings[2] = c。此时看到engine->getBindingDimensions(0)你得知道获取的是什么
2. enqueueV2是**异步推理**，加入到stream队列等待执行。输入的bindings则是tensors的指针（注意是device pointer）。其shape对应于编译时指定的输入输出的shape（这里只演示全部shape静态）
3. createExecutionContext可以执行多次，允许一个引擎具有多个执行上下文，不过看看就好，别当真

# 动态shape

动态shape，即编译时指定可动态的范围[L-H]，推理时可以允许 L <= shape <= H

**重点提炼：**

1. OptimizationProfile是一个优化配置文件，用来指定输入的shape可以变换的范围的，不要被优化两个字蒙蔽了双眼
2. 如果onnx的输入某个维度是-1，表示该维度动态，否则表示该维度是明确的，明确维度的minDims, optDims, maxDims一定是一样的

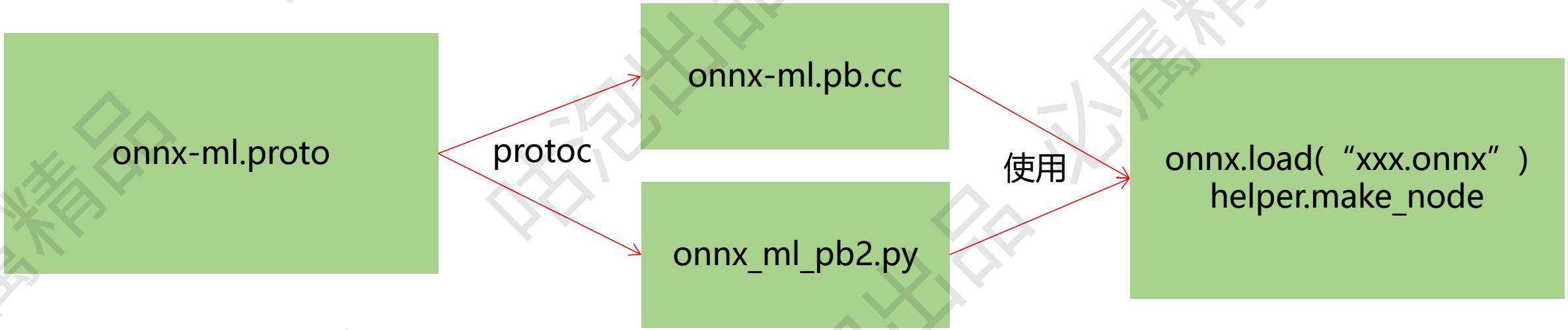
# ONNX

1、ONNX的本质，是一种Protobuf格式文件

2、Protobuf则通过onnx-ml.proto编译得到onnx-ml.pb.h和onnx-ml.pb.cc或onnx_ml_pb2.py

3、然后用onnx-ml.pb.cc和代码来操作onnx模型文件，实现增删改

4、onnx-ml.proto则是描述onnx文件如何组成的，具有什么结构，他是操作onnx经常参照的东西




https://github.com/onnx/onnx/blob/main/onnx/onnx-ml.proto

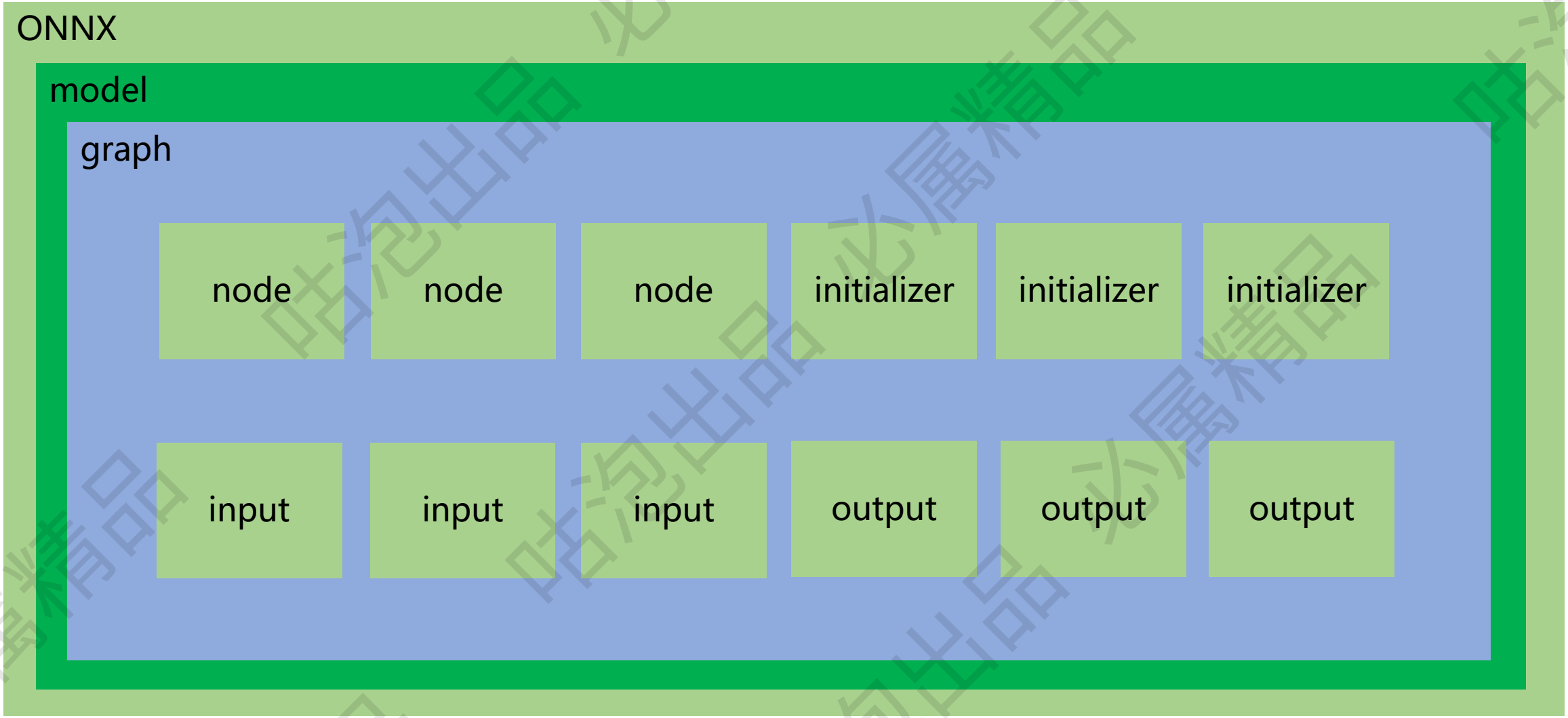
# ONNX

```
message NodeProto {
  repeated string input = 1;    // namespace Value
  repeated string output = 2;   // namespace Value

  // An optional identifier for this node in a graph.
  // This field MAY be absent in ths version of the IR.
  optional string name = 3;     // namespace Node

  // The symbolic identifier of the Operator to execute.
  optional string op_type = 4;  // namespace Operator
  // The domain of the OperatorSet that specifies the operator named by op_type.
  optional string domain = 7;   // namespace Domain

  // Additional named attributes.
  repeated AttributeProto attribute = 5;

  // A human-readable documentation for this node. Markdown is allowed.
  optional string doc_string = 6;
}
```

表示onnx中有节点类型叫node
他有input属性，是repeated，即重复类型，数组
他有output属性，是repeated，即重复类型，数组
他有name属性是string类型

对于repeated是数组，对于optional无视他
对于input = 1，后面的数字是id，无视他
**我们只关心是否数组，类型是什么**

# ONNX



# ONNX

**model**：表示整个onnx的模型，包含图结构和解析器格式、opset版本、导出程序类型

**model.graph**：表示图结构，通常是我们netron看到的主要结构

**model.graph.node**：表示图中的所有节点，数组，例如conv、bn等节点就是在这里的，通过input、output表示节点之间的连接关系

**model.graph.initializer**：权重类的数据大都储存在这里

**model.graph.input**：整个模型的输入储存在这里，表明哪个节点是输入节点，shape是多少

**model.graph.output**：整个模型的输出储存在这里，表明哪个节点是输出节点，shape是多少

对于anchorgrid类的常量数据，通常会储存在model.graph.node中，并指定类型为Constant，该类型节点在netron中可视化时不会显示出来

# ONNX

**ONNX重点：**

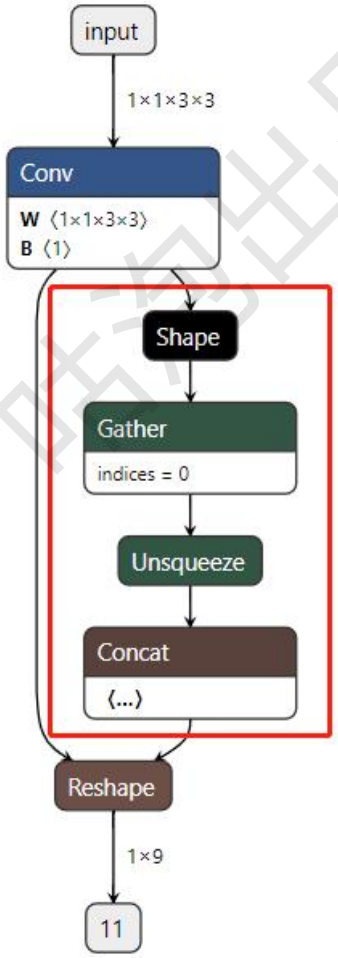
1. ONNX的主要结构：graph、graph.node、graph.initializer、graph.input、graph.output
2. ONNX的节点构建方式：onnx.helper，各种make函数
3. ONNX的proto文件，https://github.com/onnx/onnx/blob/main/onnx/onnx-ml.proto
4. 理解模型结构的储存、权重的储存、常量的储存、netron的解读对应到代码中的部分
5. ONNX的解析器的理解，包括如何使用nv发布的解析器源代码https://github.com/onnx/onnx-tensorrt

# 正确导出ONNX

1. 对于任何用到shape、size返回值的参数时，例如：tensor.view(tensor.size(0), -1)这类操作，避免直接使用tensor.size的返回值，而是加上int转换，tensor.view(int(tensor.size(0)), -1)，断开跟踪
2. 对于nn.Upsample或nn.functional.interpolate函数，使用scale_factor指定倍率，而不是使用size参数指定大小
3. 对于reshape、view操作时，-1的指定请放到batch维度。其他维度可以计算出来即可。batch维度禁止指定为大于-1的明确数字
4. torch.onnx.export指定dynamic_axes参数，并且只指定batch维度，禁止其他动态
5. 使用opset_version=11，不要低于11
6. 避免使用inplace操作，例如y[..., 0:2] = y[..., 0:2] * 2 - 0.5
7. 尽量少的出现5个维度，例如ShuffleNet Module，可以考虑合并wh避免出现5维
8. 尽量把让后处理部分在onnx模型中实现，降低后处理复杂度
9. 掌握了这些，就可以保证后面各种情况的顺利了

这些做法的必要性体现在，简化过程的复杂度，去掉gather、shape类的节点，很多时候，部分不这么改看似也是可以但是需求复杂后，依旧存在各类问题。按照说的这么修改，基本总能成。做了这些，就不需要使用onnx-simplifer了

# 正确导出ONNX

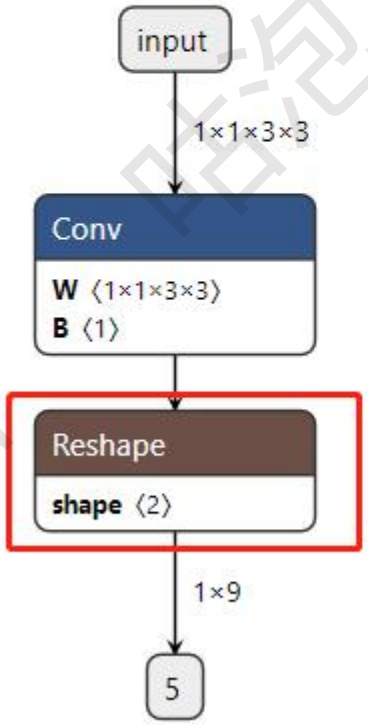



lesson1.py U lesson2.py U

workspace > lesson1.py > ...

```python
import torch
import torch.nn as nn

class Model(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv = nn.Conv2d(1, 1, 3, stride=1, padding=1, bias=True)
        self.conv.weight.data.fill_(0.3)
        self.conv.bias.data.fill_(0.2)

    def forward(self, x):
        x = self.conv(x)
        return x.view(x.size(0), -1)

model = Model().eval()

x = torch.full((1, 1, 3, 3), 1.0)
y = model(x)

torch.onnx.export(
    model, (x, ), "lesson1.onnx", verbose=True
)
```

# 正确导出ONNX




lesson1.py U

workspace > lesson1.py > ...

```python

import torch
import torch.nn as nn

class Model(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv = nn.Conv2d(1, 1, 3, stride=1, padding=1, bias=True)
        self.conv.weight.data.fill_(0.3)
        self.conv.bias.data.fill_(0.2)

    def forward(self, x):
        x = self.conv(x)
        # return x.view(int(x.size(0)), -1)
        return x.view(-1, int(x.numel() // x.size(0)))

model = Model().eval()

x = torch.full((1, 1, 3, 3), 1.0)
y = model(x)

torch.onnx.export(
    model, (x, ), "lesson1.onnx", verbose=True
)

```

# ONNX解析器

onnx解析器有两个选项，libnvonnxparser.so或者https://github.com/onnx/onnx-tensorrt（源代码）。使用源代码的目的，是为了更好的进行自定义封装，简化插件开发或者模型编译的过程，更加具有定制化，遇到问题可以调试

# 插件实现

## 插件实现-重点：

1. 如何在pytorch里面导出一个插件
2. 插件解析时如何对应，在onnx parser中如何处理
3. 插件的creator实现
4. 插件的具体实现，继承自IPluginV2DynamicExt
5. 插件的序列化与反序列化

# Int8量化

**int8量化是利用int8乘法替换float32乘法实现性能加速的一种方法**

1. 对于常规模型有：y = kx + b，此时x、k、b都是float32, 对于kx的计算使用float32的乘法
2. 对于int8模型有：y = tofp32(toint8(k) * toint8(x)) + b，其中int8 * int8结果为int16
3. 因此int8模型解决的问题是如何将float32合理的转换为int8，使得精度损失最小
4. **也因此，经过int8量化的精度会受到影响**

# Int8量化

**Int8量化步骤：**

1. 配置setFlag nvinfer1::BuilderFlag::kINT8

2. 实现Int8EntropyCalibrator类并继承自IInt8EntropyCalibrator2

3. 实例化Int8EntropyCalibrator并且设置到config.setInt8Calibrator

4. Int8EntropyCalibrator的作用，是读取并预处理图像数据作为输入

    - 标定过程的理解：对于输入图像A，使用FP32推理后得到P1再用INT8推理得到P2，调整int8权重使得P1与P2足够的接近

    - 因此标定时需要使用一些图像，正常发布时，使用100张图左右即可

# Int8量化

**Int8EntropyCalibrator类主要关注：**

1. getBatchSize，告诉引擎，这次标定的batch是多少
2. getBatch，告诉引擎，这次标定的输入数据是什么，把指针赋值给bindings即可，返回false表示没有数据了
3. readCalibrationCache，若从缓存文件加载标定信息，则可避免读取文件和预处理，若该函数返回空指针则表示没有缓存，程序会重新通过getBatch重新计算
4. writeCalibrationCache，当标定结束后，会调用该函数，我们可以储存标定后的缓存结果，多次标定可以使用该缓存实现加速

谢谢!