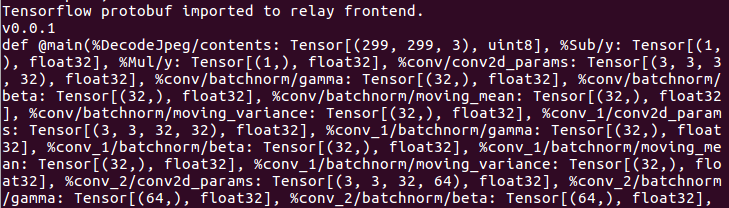
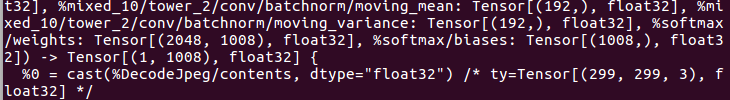
1. inceptionv4的relay层表示

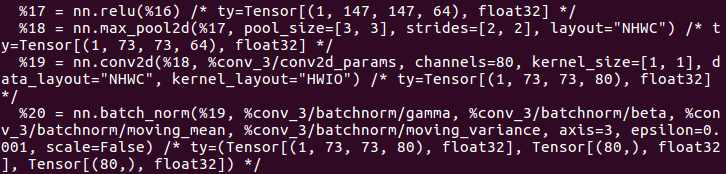
整个relay层类似于定义一个全局函数,基本形式为def @main( ) { body }。



（）中是输入和所有的输入参数。Relay严格区分局部变量和全局变量，在文本格式中，全局变量和局部变量由前缀或符号区分。 全局变量以@为前缀，局部变量以％为前缀。



->后面是输出的返回类型



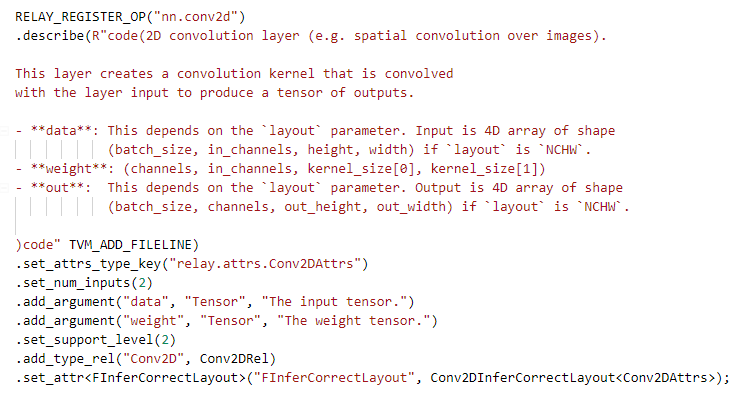
Conv2d , max\_pool2d等是relay中的op,op需要在Relay中注册，以确保它们将被集成到Relay的类型系统中。

Relay是一种静态类型（即语言的类型检查是在运行前的编译阶段）和类型推断的语言，Relay的类型系统具有一种形状依赖类型。也就是说，它的类型系统跟踪relay程序中张量的形状。类型系统允许我们静态地（即在编译时）区分不同类型的值。 这意味着我们知道表达式是否将计算为张量，函数（即（float32，float32） - > float32）或元组（float32，int32）。类型推断和检查取代了传统计算图风格IR中的形状推理。

1. 将op注册到relay(以conv2d为例）

分为以下3个步骤：

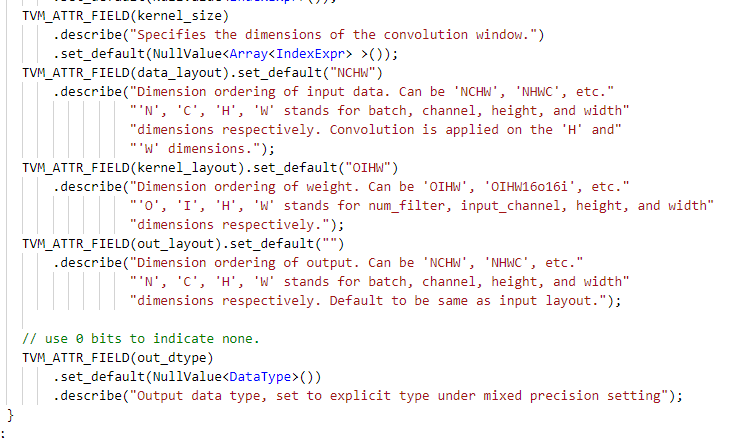
a.使用C ++中的RELAY\_REGISTER\_OP宏来注册op的参数数量和类型信息；



指明了参数类型（用type\_key”relay.attrs.conv2DAttrs”来标识），输入数量，位置参数的名称和描述(add.\_argument)，支持级别（1表示内部固有;较高的数字表示较少整体或外部支持的op）,算子的类型关系（type\_rel）,FInferCorrectLayout推断并纠正节点布局，返回一个包含两个元素的数组，两个元素为推断的输入布局和输出布局。

参数类型

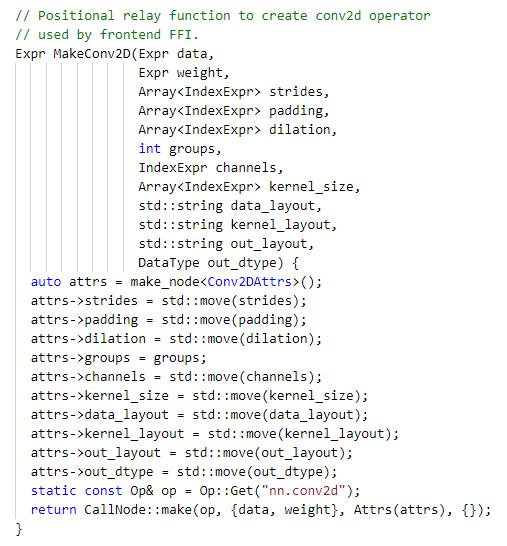


其中IndexExpr是张量形状的符号表达（using IndexExpr = ::tvm::Expr）

b.定义C ++函数来为op生成调用节点，并为该函数注册Python API挂钩；

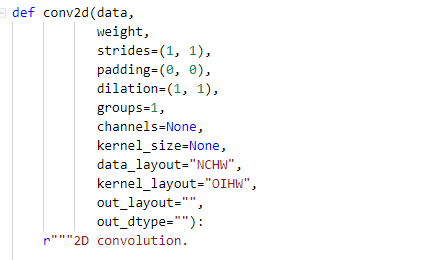
此步骤只需要编写一个函数，该函数将参数传递给op(以Relay表达式的形式）并将一个调用节点返回给op。目前不支持调用属性和类型参数（最后两个字段），因此使用Op :: Get从op注册表获取op信息并将参数传递给调用节点就足够了。

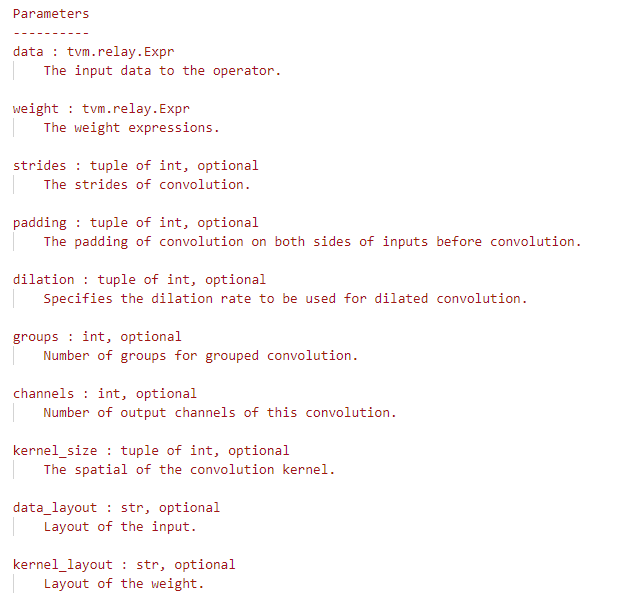


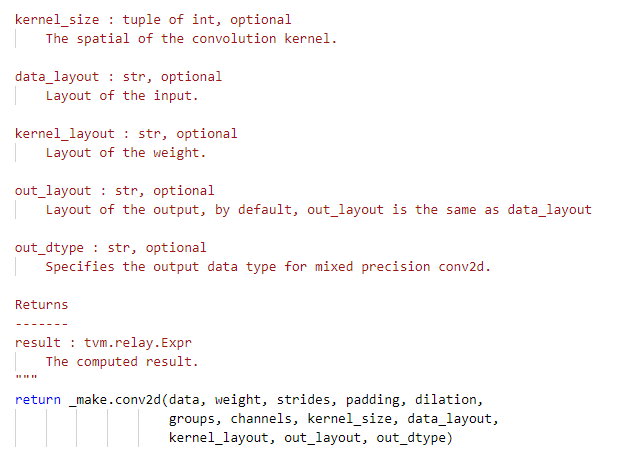


c.将上述Python API挂钩包含在更简洁的界面中。

通过TVM\_REGISTER\_API导出的函数包装在单独的Python函数中。

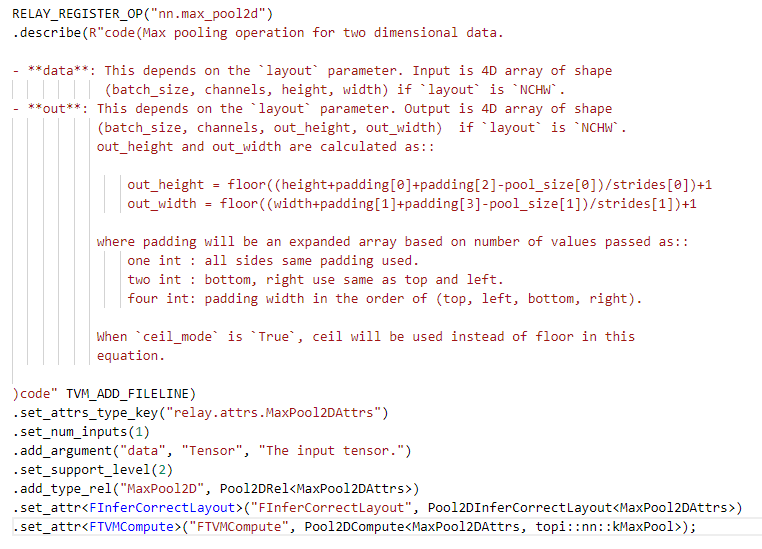




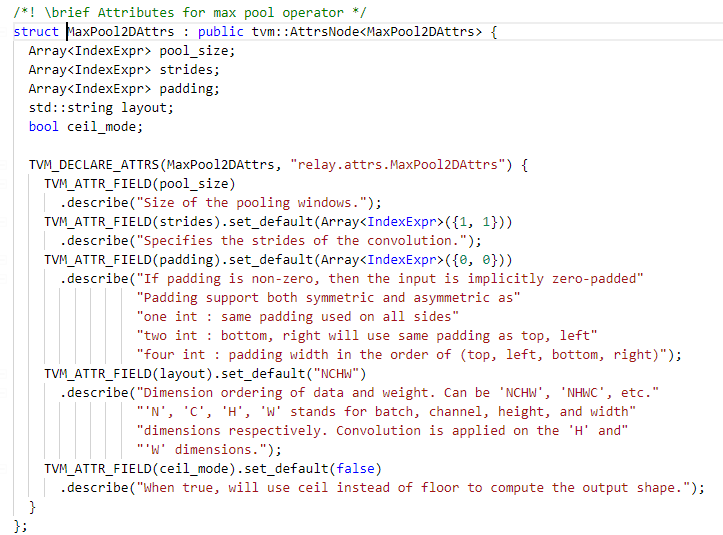


此外，maxpool2d的上述三个步骤对应代码如下，

算子注册

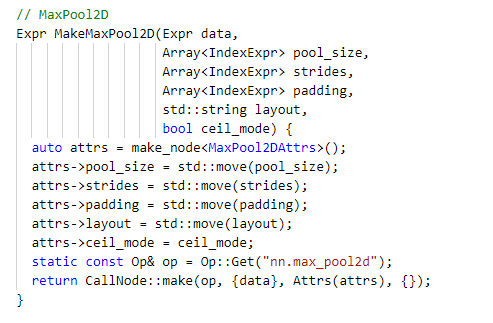


参数结构

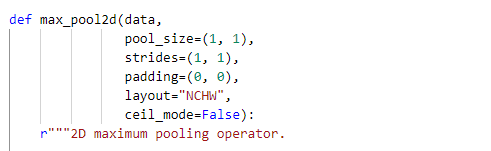


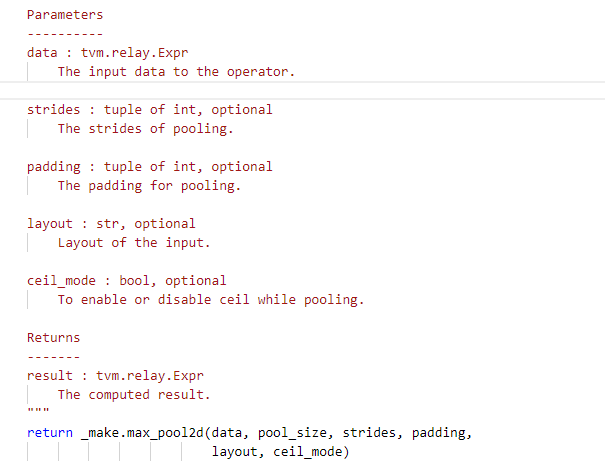
定义c++函数和注册python api



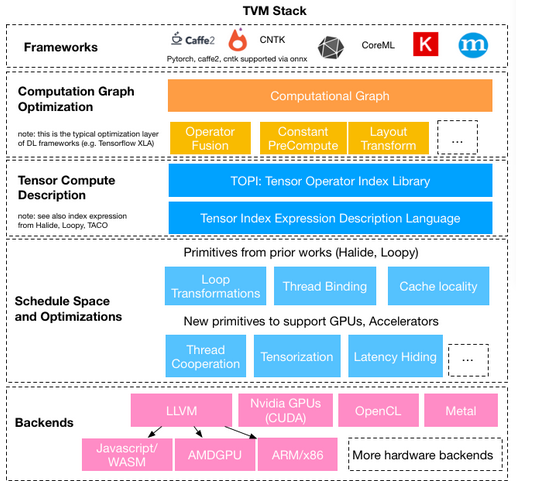


Python api





1. 数据结构



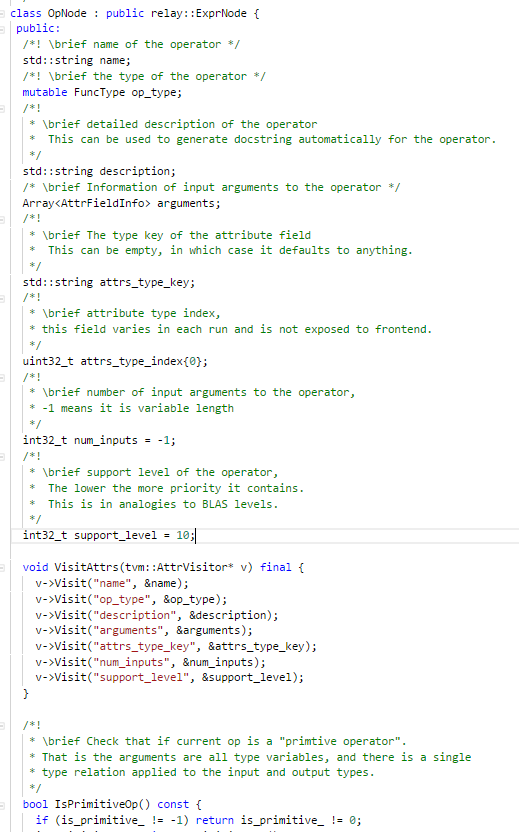
Relay的实施是根据TVM进行建模的，重用大部分已有基础，以便在TOPI和relay程序之间提供更好的兼容性。一个重大的设计决策是重用TVM节点系统，以TVM的方式将relay语言暴露给Python。此外还共享许多数据结构的表示，例如张量容器，即tvm :: runtime :: NDArray。

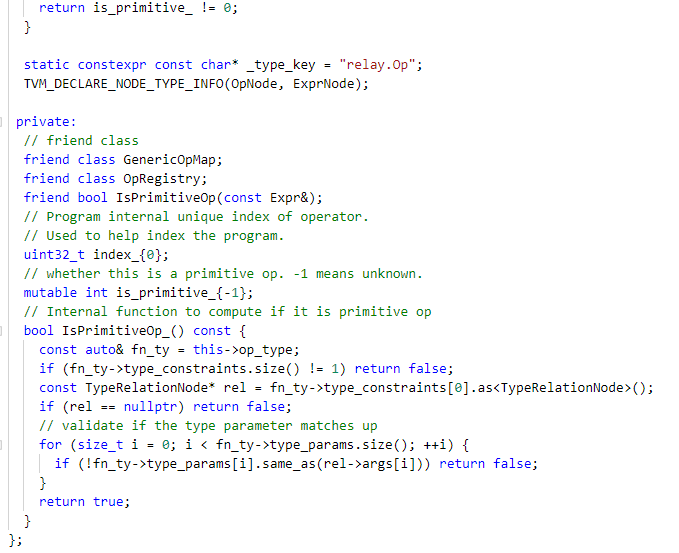
TOPI : TVM Operator Inventory,TOPI提供了比TVM更高抽象的numpy风格的通用算子和调度，描述在 DL 领域会用到的高层次 Operator 如 matmul, conv2d 等。

Relay可理解为NNVM2代，它的目标是将基于旧计算图的IR（如NNVM）替换为更具表现力的IR，可以针对许多目标进行有效优化。

NNVM有op,node,graph等数据结构，对照nnvm，找到如下定义

Op





其中ExprNode是RelayNode的子类，RelayNode是Node的子类。