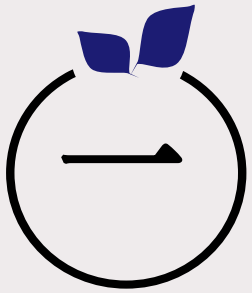
动转静乱讲

Nyakku Shigure

目录

CONTENT



什么是动转静？

动态图长啥样子呢？

- 即时执行，随时可以获得结果
- 调试方便

```
import paddle

class Net(paddle.nn.Layer):
    def __init__(self):
        super().__init__()
        self.linear = paddle.nn.Linear(10, 3)

    def forward(self, x):
        return self.linear(x)


net = Net()

x = paddle.randn([4, 10])
out = net(x)
print(out)
```

静态图又是个啥呢？

- 需先构建 Program，之后由 Executor 来执行
- 组网代码可以只执行一次，之后将由 Executor 来调度执行，效率更高

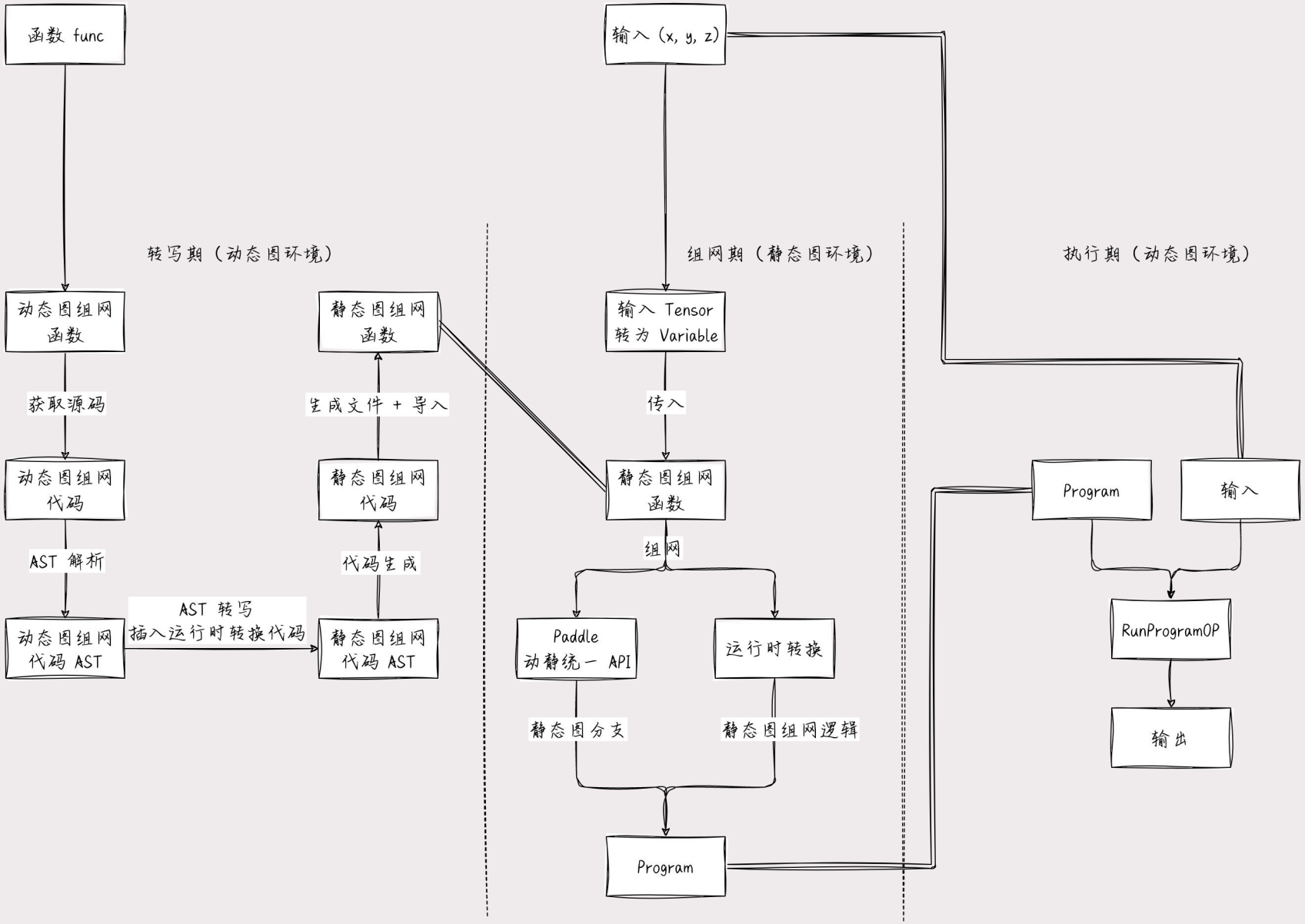
```
import paddle
import numpy as np

paddle.enable_static()

class Net(paddle.nn.Layer):
    def __init__(self):
        super().__init__()
        self.linear = paddle.nn.Linear(10, 3)

    def forward(self, x):
        return self.linear(x)


net = Net()

x = paddle.static.data(name="x", shape=[None, 10], dtype=
"float32")
out = net(x)
place = paddle.CPUPlace()

exe = paddle.static.Executor(place)
exe.run(paddle.static.default_startup_program())

out = exe.run(
    paddle.static.default_main_program(),
    feed={x.name: np.random.randn(4, 10).astype("float32")},
    fetch_list=[out]
)
print(out)
```

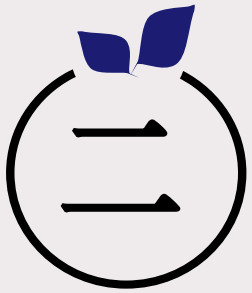
动转静会发生啥呢？

- 仍然写动态图代码即可，只需加上一个装饰器就可以使其在静态图机制运行，运行效率提高

```
import paddle

class Net(paddle.nn.Layer):
    def __init__(self):
        super().__init__()
        self.linear = paddle.nn.Linear(10, 3
)

    @paddle.jit.to_static
    def forward(self, x):
        return self.linear(x)


net = Net()

x = paddle.randn([4, 10])
out = net(x)
print(out)
```

动转静的几个阶段


函数 func
输入（x, y, z）
转写期（动态图环境）
组网期（静态图环境）
执行期（动态图环境）
动态图组网函数
获取源码
动态图组网代码
AST 解析
动态图组网代码 AST
AST 转写
插入运行时转换代码
静态图组网代码 AST
代码生成
静态图组网代码
生成文件 + 导入
静态图组网函数
输入 Tensor 转为 Variable
传入
静态图组网函数
组网
Paddle 动静统一 API
运行时转换
静态图分支
静态图组网逻辑
Program
Program
输入
RunProgramOP
输出



动转静转写期

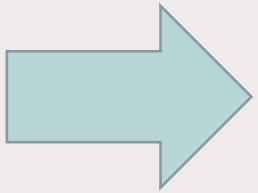
我们会将代码转写成什么样子捏？

- to_static 装饰器会将原来的函数转换为 Callable 的 StaticFunction，我们可以通过其 code 属性拿到其转换后的源码
- 这里 _jst.Call 是一个运行时转换函数，会在组网阶段根据情况来转换

```python
import paddle

@paddle.jit.to_static
def foo(x):
    len(x)

print(foo.code)

# def foo(x):
#     _jst.Call(len)(x)
```

控制流会转成啥样子呢？

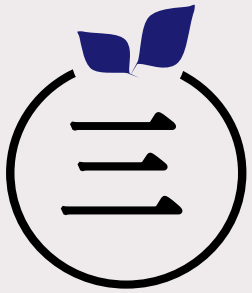
```python
def foo(x):
    if x > 3:
        y = x + 1
    else:
        y = x - 1
    return y
```



```python
def foo(x):
    y = _jst.UndefinedVar('y')
    __return_value_5 = None

    def get_args_2():
        nonlocal y
        return y,

    def set_args_2(__args):
        nonlocal y
        y, = __args

    def true_fn_1():
        nonlocal y
        y = x + 1
        return

    def false_fn_1():
        nonlocal y
        y = x - 1
        return
    _jst.IfElse(
        x > 3,
        true_fn_1, false_fn_1,
        get_args_2, set_args_2,
        ('y',),
        push_pop_names=None
    )
    __return_value_5 = y
    return __return_value_5
```

代码是如何转写的呢？

- 首先通过 inspect.getsource 获取函数源码
- 之后通过 ast.parse 解析成为抽象语法树
- （演示一下～）



动转静组网期

组网期发生了什么呢？

- Tensor 转 Variable
- 动静统一 API
- 运行时转换
- （演示一下～）

动转静执行期

执行期又发生了什么呢？

- Tensor 和 Program 传入 RunProgram OP
- Tensor share 进 Program 的 Variable
- 静态图执行器调度执行
- Variable share 回 Program 的 Tensor
- 返回结果

ก感谢观看ก