

PyLayer功能支持动转静

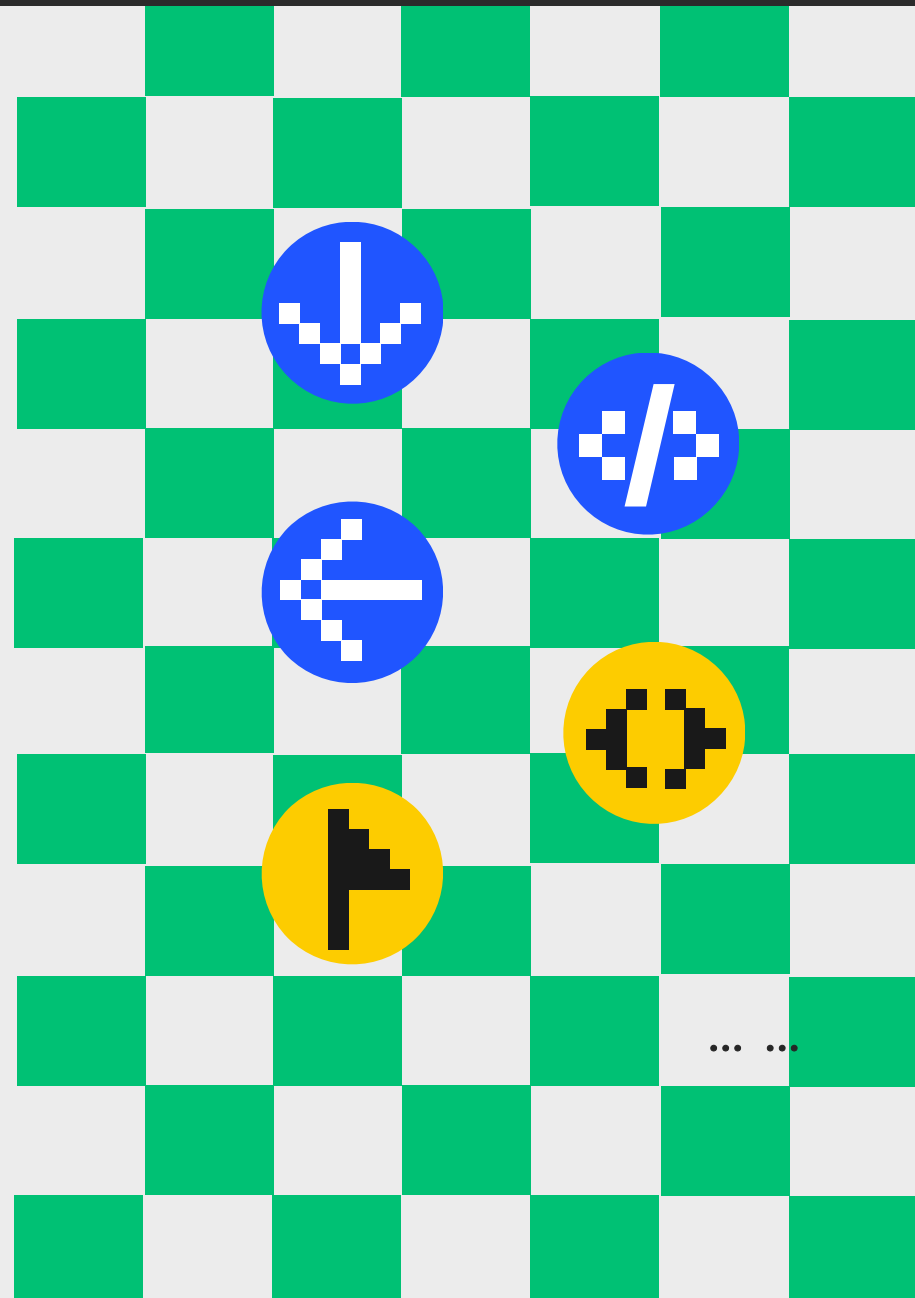


学生：67

导师：dl

学校：三本

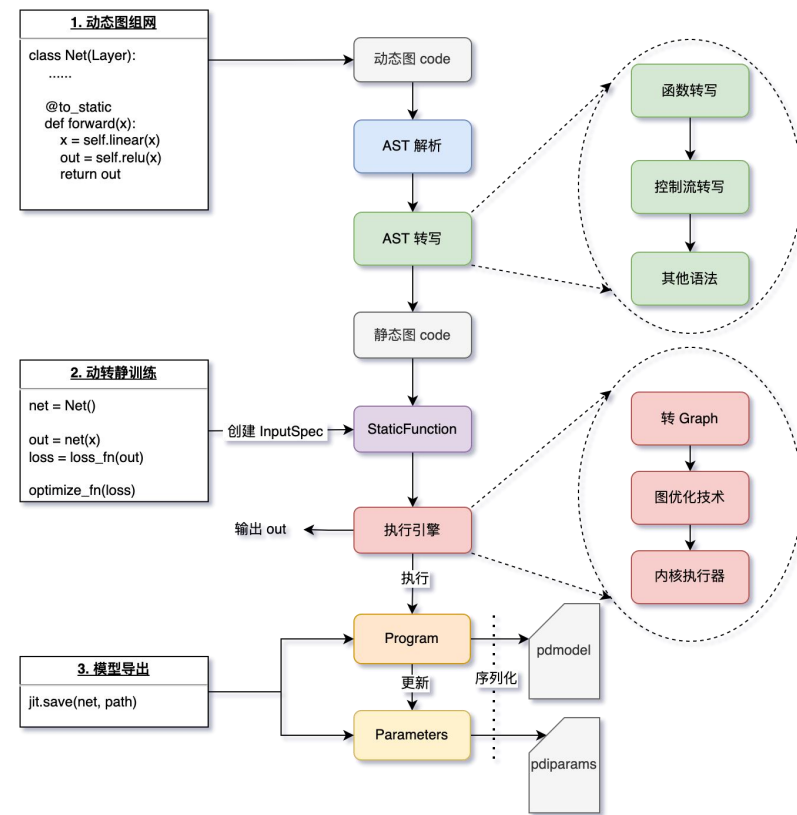
时间：2023/08/15



- 在深度学习模型构建上，飞桨框架支持动态图编程和静态图编程两种方式，其代码编写和执行方式均存在差异。Paddle 目前提供 `to_static` 来完成动态图编程的模型到静态图模型的转换。

- 飞桨的动态图 `PyLayer` 向用户提供了一种高度灵活且便利的自定义网络层前、反向计算的机制。但是目前当动态图模型中包含 `PyLayer` 接口功能的使用时，暂不支持 `@to_static` 装饰以生成对应的静态图 Program。

- 项目需求：**要求飞桨的动态图中的 `PyLayer` 机制能够与飞桨的动转静 (`to_static`) 互通，支持模型中 `PyLayer` 的自定义层能够被 `@to_static` 感知并正确地生成静态图 Program，进而支撑转静训练和导出推理。



动转静原理

PyLayer 功能支持动
转静

静态图 PyLayer 算子

前向 OP

反向 OP

使 @to_static 动转静能感知 PyLayer 模块

1. ast 模块能感知到 PyLayer, 并使用 convert_call 对 PyLayer 进行转写

2. 对 PyLayer 的 forward 和 backward 方法进行静态图转写

3. 根据 forward 和 backward 的逻辑生成前向 block 和反向 block

使 @to_static 动转静能感知并转写和处理 PyLayerContext 的相关逻辑

1. 感知 PyLayerContext

2. 处理 PyLayerContext 的逻辑

- 根据前期对 Paddle 中 PyLayer 实现机制和 动转静@to_static 模块的调研分析, 可以将课题的解决思路划分为以下3个步骤:
- 1. 构建静态图的 PyLayer 算子, 支持运行用户自定义的前向 block 和反向 block
- 2. @to_static 动转静能感知 PyLayer 模块, 并对 forward 逻辑和 backward 逻辑进行转写, 分别生成前向 block 和 反向 block
- 3. @to_static 动转静能感知并转写和处理 PyLayerContext 的相关逻辑, 完成前向和反向的变量传递: 保存前向过程产生的中间变量, 在反向阶段取出相关的中间变量进行运算

静态图 PyLayer 算子

静态图 PyLayer 算子的期望形态：**无 kernel op**，其接受用户自定义的前向 block 和反向 block，并在前向过程中运行用户自定义的前向 block；在反向 op 构建时根据 op 保存的“blocks”信息指定反向 op 运行的 block 为用户自定义的反向 block，从而实现正反向 block 的关联

前向 op 的 op maker

```
class StaticPyLayerForwardOpProtoMaker
: public framework::OpProtoAndCheckerMaker {
public:
    void Make() override {
        AddInput(StaticPyLayerOp::kInputs, "The input variables of the sub-block.")
            .AsDuplicable();
        AddOutput(StaticPyLayerOp::kOutputs,
            "The output variables of the sub-block.")
            .AsDuplicable();
        AddOutput(StaticPyLayerOp::kScope,
            "(std::vector<Scope*>) The scope of static pylayer block.");
        AddAttr<std::vector<framework::BlockDesc *>>(
            "blocks", "The blocks of PyLayer operator");
    }
};
```

静态图的 PyLayer op 的执行过程概括为以下三个步骤：

1. 从 blocks 或 backward_block 属性中获取执行的 block 的 BlockDesc
2. 设置 ExecutionConfig 等参数，创建执行器
3. 运行

反向 op 的 op maker

```
template <typename T>
class StaticPyLayerBackwardMaker : public framework::SingleGradOpMaker<T> {
public:
    using framework::SingleGradOpMaker<T>::SingleGradOpMaker;

protected:
    void Apply(GradOpPtr<T> grad_op) const override {
        grad_op->SetType("static_pylayer_grad");
        grad_op->SetInput(StaticPyLayerOp::kInputs,
            this->Input(StaticPyLayerOp::kInputs));
        grad_op->SetInput(framework::GradVarName(StaticPyLayerOp::kOutputs),
            this->OutputGrad(StaticPyLayerOp::kOutputs));
        grad_op->SetInput(StaticPyLayerOp::kScope,
            this->Output(StaticPyLayerOp::kScope));
        grad_op->SetOutput(framework::GradVarName(StaticPyLayerOp::kInputs),
            this->InputGrad(StaticPyLayerOp::kInputs, false));

        const std::vector<framework::BlockDesc *> &blocks =
            PADDLE_GET_CONST(std::vector<framework::BlockDesc *>,
                this->GetAttr(StaticPyLayerOp::kBlocks));
        grad_op->SetBlockAttr("backward_block",
            blocks[static_cast<size_t>(PyLayerBlockIndex::kBACKWARD)]);
    }
};
```

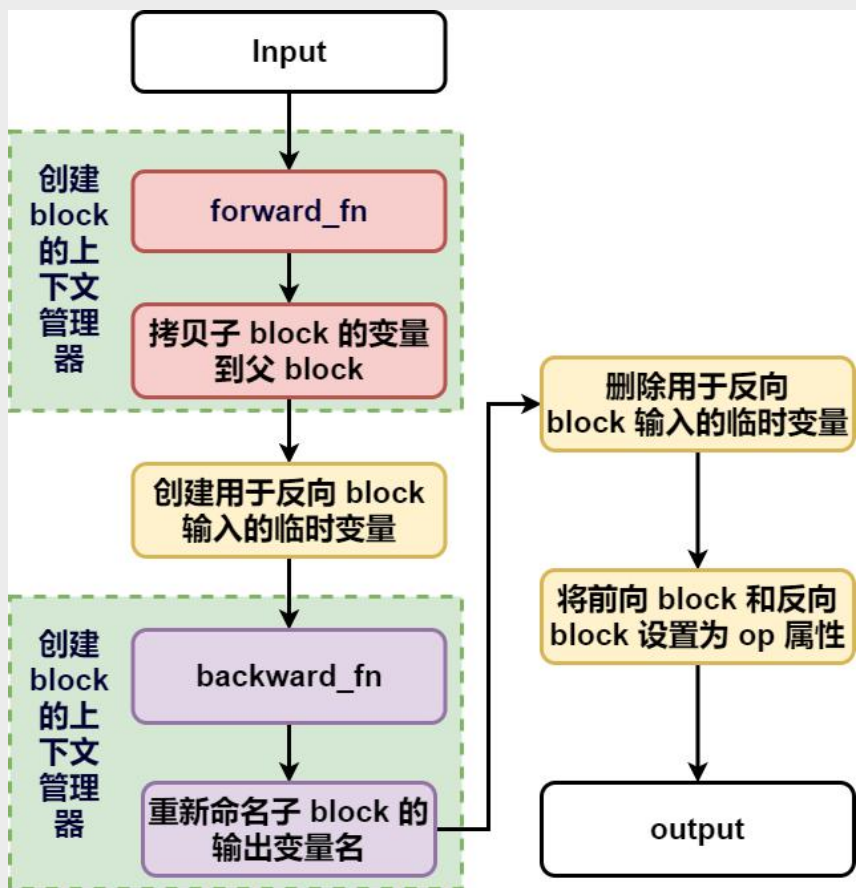
静态图 PyLayer OP
的 Python API

```
def static_pylayer(forward_fn, inputs, backward_fn):
```

静态图前向函数

静态图前向函数的
输入

静态图反向函数



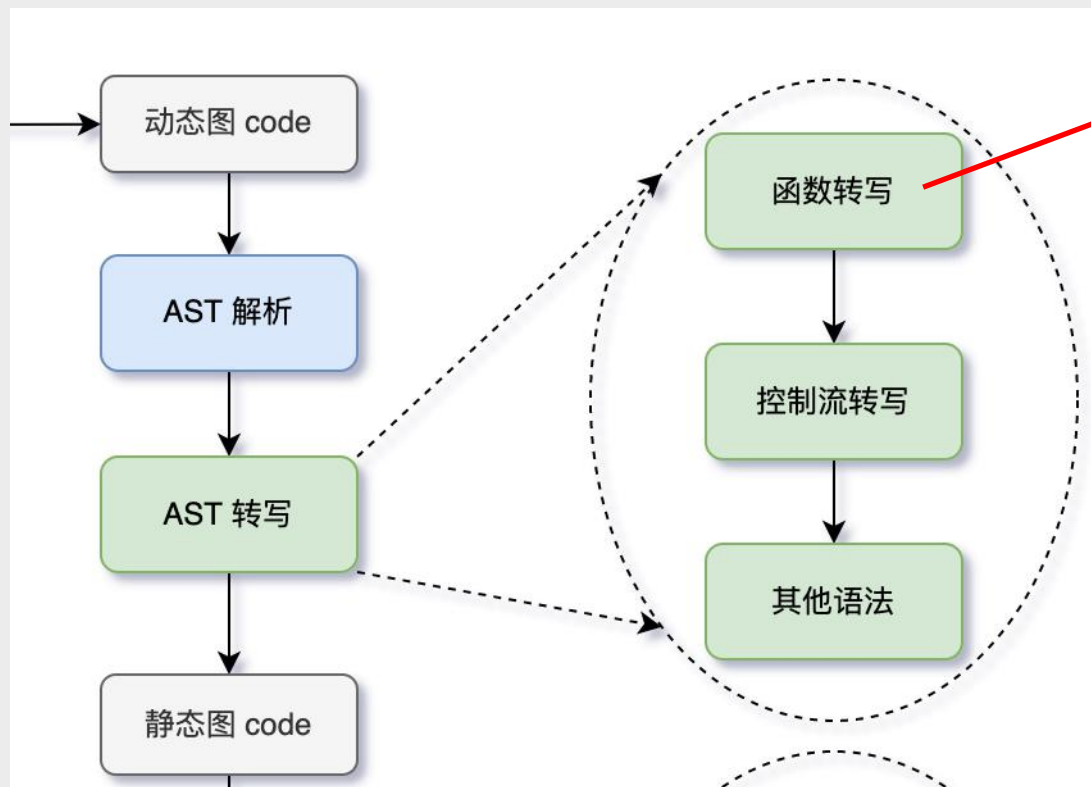
- 静态图 PyLayer op 的 python API 使用 `StaticPyLayerBlock` 作为 block 创建的上下文管理器 (其作用类似于现有的 `ConditionalBlockGuard`), 分别创建正反向 block
- 需要注意的是, 临时输入变量的名称, 以及类型、形状推导
- 还需要注意的是, 构建反向 block 会得到反向 block 的输出名称, 需要将其名字重命名为符合“前向 block 输入的梯度”的命名规范, 以便反向 block 在执行器运行时对父 block 的变量进行赋值。

静态图 PyLayer OP 的 Python API 执行流程图

动转静感知 PyLayer
模块

此处的“感知”有三层含义：

1. AST 模块能感知到 **PyLayer**，并使用 **convert_call** 对 **PyLayer** 进行转写
2. 对 **PyLayer** 的 **forward** 和 **backward** 方法进行静态图转写
3. 根据 **forward** 和 **backward** 的逻辑生成前向 **block** 和反向 **block**



在函数转写对函数的 `__self__` 属性进行类型判断，判断其是否为 `PyLayerMeta`

可改进的：AST 模块应能直接感知到 **PyLayer**，而不是现阶段的必须感知到 **PyLayer.apply** 调用。在使用 **convert_call** 进行转写后的 **PyLayer.apply** 最终应该会调用 **StaticPyLayer.apply**。为了达到这个效果，可能需要修改语法树转换器 **Transformer**

动转静感知和处理 PyLayerContext

PyLayerContext 能够协助 **PyLayer** 完成特定的功能，比如 **save_for_backward** 函数可以保存 backward 需要的中间变量；**saved_tensor** 函数可以获取被 **save_for_backward** 保存的中间变量。对于本次项目，我们只需要考虑 **PyLayerContext** 上述的两个功能。

如何感知 PyLayerContext ?

- 新建 **StaticPyLayerContext** 类，与转写的 forward 和 backward 方法绑定，进而使得 **ctx.save_for_backward** 和 **ctx.saved_tensor** 的函数调用，ctx 指向的是 **StaticPyLayerContext**

如何处理 PyLayerContext 的逻辑?

- save_for_backward** 可以记录下保存的中间变量的变量名字，在创建静态图 pylayer 算子时将变量名字作为属性传递给静态图 pylayer op，使其运行期执行器执行前向 block 后不删除 scope 里的对应变量。前向的 scope 会作为输出被静态图 pylayer op 传递出去。
- saved_tensor** 可以在反向 block 内根据保存的中间变量的变量属性创建对应的 var，然后在运行期根据保存的中间变量的名字从前向的 scope 中获取并赋值给当前 block 的相关变量。
- 总而言之，可以通过 scope 和算子属性，来实现前向信息和反向信息的传递。

7.1 ~ 7.14
阅读 Paddle 源码，弄清
to_static 和 PyLayer 实现细节



7.23 ~ 8.1
编写静态图的 PyLayer 算子及其
Python api



8.4 ~ 8.11
code review 和代码改进



7.15 ~ 7.22
完成 Paddle 源码编译，并使
to_static 能感知 PyLayer




8.1 ~ 8.3
排查编译链接时出现的静态库链
接裁剪的问题



8.12 ~ 8.15
中期答辩




code, debug, and
discussion ...


code, debug, and
discussion ...



成果展示

PR 链接: <https://github.com/PaddlePaddle/Paddle/pull/56108>项目过程与文档记录: <https://github.com/MarioLulab/GLCC2023-Paddle-Record>

```
paddle.enable_static()

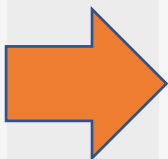
def forward_fn(x):
    y = paddle.tanh(x)
    return y

def backward_fn(dy):
    dx = dy * 2
    return dx

train_program = static.Program()
start_program = static.Program()

place = paddle.CPUPlace()
exe = paddle.static.Executor(place)
with static.program_guard(train_program, start_program):
    data = paddle.static.data(name="X", shape=[None, 5], dtype="float32")
    data.stop_gradient = False
    ret = static_pylayer.static_pylayer(forward_fn, [data], backward_fn)
    loss = paddle.mean(ret)
    sgd_opt = paddle.optimizer.SGD(learning_rate=0.01).minimize(loss)
    print(static.default_main_program())

exe = paddle.static.Executor(place)
exe.run(start_program)
x = np.random.randn(10, 5).astype(np.float32)
loss, loss_g, x_g, y, y_g = exe.run(
    train_program,
    feed={"X": x},
    fetch_list=[
        loss.name,
        loss.name + '@GRAD',
        data.name + '@GRAD',
        ret.name,
        ret.name + '@GRAD',
    ],
)
```



```
{ // block 0
    var X : LOD_TENSOR.shape(-1, 5).dtype(float32).stop_gradient(False)
    var _generated_var_0 : LOD_TENSOR.shape(-1, 5).dtype(float32).stop_gradient(False)
    var _generated_var_1 : STEP_SCOPES
    var mean_0.tmp_0 : LOD_TENSOR.shape().dtype(float32).stop_gradient(False)
    var X@GRAD : LOD_TENSOR.shape(-1, 5).dtype(float32).stop_gradient(False)
    var _generated_var_0@GRAD : LOD_TENSOR.shape(-1, 5).dtype(float32).stop_gradient(False)
    var mean_0.tmp_0@GRAD : LOD_TENSOR.shape().dtype(float32).stop_gradient(False)
    persist var learning_rate_0 : LOD_TENSOR.shape().dtype(float32).stop_gradient(True)

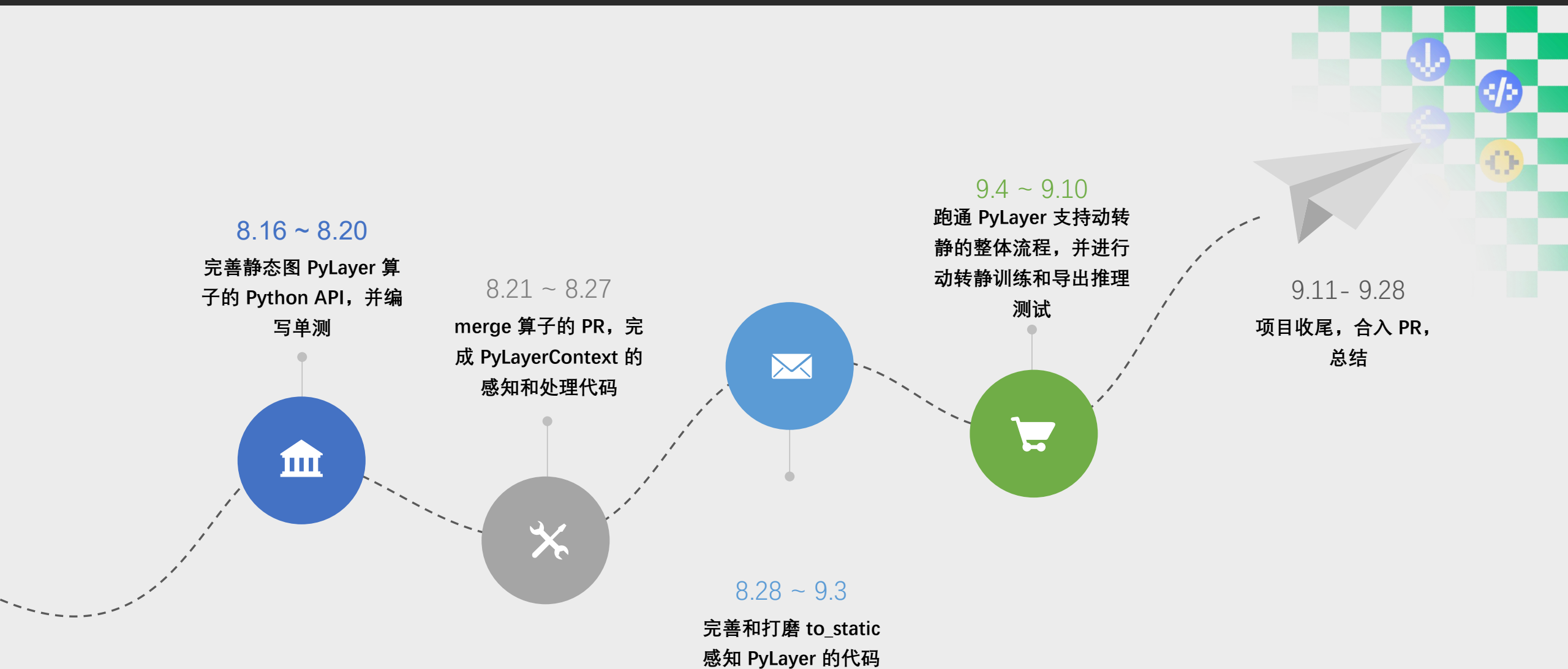
    {Out=['_generated_var_0'], Scope=['_generated_var_1']} = static_pyplayer(inputs={Input=['X']}, blocks = blocks[1, 2], op_device = ,
        op_namescope = /, op_role = 0, op_role_var = [], with_quant_attr = False)
    {Out=['mean_0.tmp_0']} = reduce_mean(inputs={X=['_generated_var_0']}, dim = [], in_dtype = -1, keep_dim = False, op_device = ,
        op_namescope = /, op_role = 256, op_role_var = [], out_dtype = -1, reduce_all = True, with_quant_attr = False)
    {Out=['mean_0.tmp_0@GRAD']} = fill_constant(inputs={}, dtype = 5, force_cpu = False, op_device = , op_namescope = ,
        op_role = 257, op_role_var = [], place_type = -1, shape = [], str_value = , value = 1.0, with_quant_attr = False)
    {X@GRAD=['_generated_var_0@GRAD']} = reduce_mean_grad(inputs={Out@GRAD=['mean_0.tmp_0@GRAD'], X=['_generated_var_0']}, dim = [],
        in_dtype = -1, keep_dim = False, op_device = , op_namescope = /, op_role = 1, op_role_var = [], out_dtype = -1, reduce_all = True, with_quant_attr = False)
    {Input@GRAD=['X@GRAD']} = static_pyplayer_grad(inputs={Input=['X'], Out@GRAD=['_generated_var_0@GRAD'], Scope=['_generated_var_1']},
        backward_block = block[2], op_device = , op_role = 1)
}

{ // block 1
    var tanh_0.tmp_0 : LOD_TENSOR.shape(-1, 5).dtype(float32).stop_gradient(False)

    {Out=['tanh_0.tmp_0']} = tanh(inputs={X=['X']}, op_device = , op_namescope = /, op_role = 0, op_role_var = [], with_quant_attr = False)
    {Out=['_generated_var_0']} = assign(inputs={X=['tanh_0.tmp_0']}, op_device = , op_namescope = /, op_role = 0, op_role_var = [], with_quant_attr = False)
}

{ // block 2
    var X@GRAD : LOD_TENSOR.shape(-1, 5).dtype(float32).stop_gradient(False)

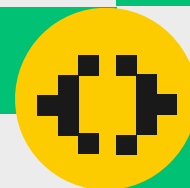
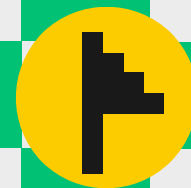
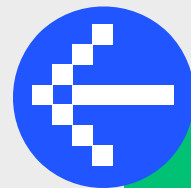
    {Out=['X@GRAD']} = scale(inputs={ScaleTensor=[], X=['_generated_var_0@GRAD']}, bias = 0.0, bias_after_scale = True, op_device = , op_namescope = /,
        op_role = 1, op_role_var = [], scale = 2.0, with_quant_attr = False)
}
```



- **了解学习 Paddle 与深度学习框架技术：**通过参与代码贡献，我更加深入地了解了飞桨框架的设计和原理，特别是动态图和静态图的构建和运行机制。这让我对深度学习框架有了更加全面和深入的认识，也让我更加有信心和动力去学习和探索新的技术。
- **开源协作：**在社区中，我结识了很多有才华和热心的开发者，我们一起探讨问题、解决问题，共同为项目的进展贡献力量。这种团队协作的精神和氛围也让我更加有归属感和责任感，更加珍视团队中每一个人的贡献。
- **发现问题、分析问题和解决问题：**在实现这个需求的过程中，我遇到了一些困难和挑战。首先，我需要深入了解飞桨框架的内部实现和PyLayer机制的细节，这对我来说是一个挑战。其次，我需要理解@to_static装饰器内部的细节，这也是一个具有挑战性的任务。然而，通过不断地阅读文档、调试代码和向社区求助，我最终成功地解决了这些问题。

谢谢大家

THANKS



...