class paddle.fluid.incubate.fleet.collective.Collective(Fleet)

Fleet 中 Collective 训练对应的模块，包含创建、执行 Collective 训练的方法。

方法

- distributed_optimizer(optimizer, strategy=dist_strategy)

  - **参数**:
  - `optimizer(Optimizer)` – 训练使用的优化器
  - `strategy(DistributedStrategy) – 分布式策略的配置

  封装常规的 paddle optimizer 为 Distributed Optimizer ， 提供对分布式优化的支持。

class paddle.fluid.incubate.fleet.collective.CollectiveOptimizer(DistributedOptimizer)

Collective 分布式训练优化器，用于包装普通的优化器对象，将普通优化器转换为分布式训练的优化器，并将单机训练程序转换为分布式训练程序。

方法

- minimize(self, loss, startup_program=None, parameter_list=None, no_grad_set=None)

  - **参数**:
  - `loss(Variable|List[Variable])` – 需要优化的 loss 变量
  - `start_up_program(Program)` – 初始化 `parameter_list` 对应的初始化程序
  - `parameter_list(list)` – 需要求解的参数列表
  - `no_grad_set(set|None)` – 需要忽略的变量

  fleet api 对 paddle minimize 的封装，对于使用 FleetAPI 的训练，需要使用该 API 替换 paddle minimize。

  该方法除了调用 optmizer.minimize 构建普通的优化程序之外，内部调用 _try_to_compile 方法，对 loss 对应的 program 进行转换，将普通的单机 program 分布式化。

- _try_to_compile(self, startup_program, main_program):

  - **参数**:
  - `loss(Variable|List[Variable])` – 需要优化的 loss 变量
  - `main_program(Program)` – 初始化 `parameter_list` 对应的初始化程序

- `parameter_list(list)` – 需要求解的参数列表
- `no_grad_set(set|None)` – 需要忽略的变量

内部方法，根据用户配置生成编译配置，调用 `_transpile` 方法将 `startup_program` 和 `main_program` 转换为分布式程序，调用编译后程序的 `CompiledProgram.with_data_parallel` 方法构建数据并行。

- _transpile(self, startup_program, main_program)

  - **参数**:
  - `startup_program(Program)` – 需要被转换的程序的初始化部分
  - `main_program(Program)` – 需要被程序的主要部分

  该方法调用 `DistributeTranspiler` 将程序转换为分布式程序，向程序中添加分布式执行相关的变量。

class paddle.fluid.incubate.fleet.dist_transpiler.DistributeTranspiler

将 `fluid` 程序转换为分布式数据并行程序。支持两种模式：parameter server (pserver) 模式和 NCCL2 模式。

在 pserver 模式， `main_program` 将被转换为与远程的 parameter server 交互进行参数优化，程序的优化图部分将被置于 parameter server 程序中。

在 NCCL2 模式，转换过程中会将 `NCCL_ID broadcasting` 算子追加到 `startup_program` 初始化程序中，实现不同的工作节点共享 `NCCL_ID` 。在调用 `transpile_nccl2` 之后，用户**必须**向 `ParallelExecutor` 传递 `trainer_id` 和 `num_trainersw` 参数以使 NCCL2 训练模式生效。

方法

- transpile(self, trainer_id, program=None, pservers="127.0.0.1:6174", trainers=1, sync_mode=True, startup_program=None, current_endpoint="127.0.0.1:6174")

  - **参数**
  - `trainer_id(int)` – 当前 `trainer` 的 id，从 0~n-1 编号|
  - `program(Program|None)` – 需要转换的程序
  - `pservers(str)` – 参数服务器的地址，ip:port 格式，使用逗号分隔
  - `trainers(int|str)` – pserver模式中的trainer数，NCCL2 模式中的节点数
  - `sync_mode(bool)` – 是否使用同步训练
  - `startup_program(Program|None)` – 需要转换的 startup_program 初始化程序
  - `current_endpoint(str)` – NCCL2 模式中的当前节点名。

  将 program 转换为分布式程序。如果是 NCCL2 模式或 collective 则调用 _transpile_nccl2、

_transpile_collective 进行转换，否则将程序转换为 pserver 程序。

在 pserver 模式下，向程序中追加与 pserver 通信的 op，将 parameters 连接后发送至 pserver，并从 pserver 接收更新的参数。

- _transpile_nccl2(self, trainer_id, trainers, current_endpoint, startup_program=None, wait_port=True)

该方法将普通训练程序转换为基于 NCCL2 的分布式训练程序。该方法向训练程序中添加 NCCL2 endpoint 注册算子和 NCCL2 同步算子，由算子调用 NCCL2 实现多机多卡间的参数同步。

- _transpile_collective(self, collective_mode, trainer_id, trainers, current_endpoint, startup_program=None, main_program=None, wait_port=True):

  - **参数**
  - `collective_mode(str|collective)` – 训练模式。`grad_allreduce|local_sgd|single_process_multi_thread`
  - `trainer_id(int)` – 当前 `trainer` 的 id，从 0~n-1 编号
  - `trainers(int|str)` – 训练节点数
  - `current_endpoint(str)` – 当前节点名。
  - `startup_program(Program|None)` – 需要转换的 startup_program 初始化程序
  - `main_program(Program|None)` – 需要转换的程序
  - `wait_port(bool)`

  该算子根据 `collective_mode` 的选择，调用 `collective.GradAllReduce`、`collective.LocalSGD`、`collective.SingleProcessMultiThread` 将程序转换为 collective 分布式训练程序。三种方式均为 `collective.Collective` 的子类，调用 `collective.Collective.transpile` 进行转换。

- compiler.CompiledProgram(main_program)

  - **参数**
  - `program_or_graph(Graph|Program)` – 需要编译的程序或计算图
  - `build_strategy(BuildStrategy)` – 编译配置项

  `CompiledProgram` 类根据 `build_strategy` 传递的编译设置将程序进行多种优化处理。创建对象时只进行配置处理，实际编译过程发生在 `Executor.run` 调用中。

- with_data_parallel(self, loss_name=None, build_strategy=None, exec_strategy=None, share_vars_from=None, places=None)

  该方法将 `CompiledProgram._is_data_parallel` 属性置为 `True`，以将该程序标记为数据并行编译模式。

class paddle.fluid.incubate.fleet.base.role_maker.RoleMakerBase

方法

- RoleMakerBase() –> RoleMakerBase
  - **参数:** 无
  - **返回**: `RoleMakerBase` 对象， 用作 `fleet.init(role_maker = None)` 的初始化
- is_worker() –> bool
  - **参数**: 无
  - **返回**: 判断当前进程是否是worker节点, 仅限Parameter Server训练模式中使用
- is_server() –> bool
  - **参数**: 无
  - **返回**: 判断当前进程是否是server节点, 仅限Parameter Server训练模式中使用
- is_first_worker() –> bool
  - **参数**: 无
  - **返回**: 判断当前进程是否是woker节点的第一个实例, 仅限Parameter Server训练模式中使用
- woker_num() –> int
  - **参数**: 无
  - **返回**: 得到当前woker节点的数量
- woker_index() –> int
  - **参数**: 无
  - **返回**: 得到当前woker的woker id
- server_index() –> int
  - **参数**: 无
  - **返回**: 得到当前server的server id, 仅限Parameter Server训练模式中使用
- get_trainer_endpoints() –> list[str]
  - **参数**: 无
  - **返回**: 得到所有trainer节点的地址信息, 即IP地址与端口号信息
- get_pserver_endpoints() –> list[str]
  - **参数**: 无
  - **返回**: 得到所有server节点的地址信息, 即IP地址与端口号信息, 仅限Parameter Server训练模式中使用

class paddle.fluid.incubate.fleet.base.role_maker.MPIRoleMaker

> - *继承自 `paddle.fluid.incubate.fleet.base.role_maker.RoleMakerBase` , 使用 `MPI` 接口点对点RPC作为分布式通信的方法, 对立的是 `K8SRoleMaker` 采用分布式注册中心的分布式通信方法*
> - *内部依赖 `mpi4py` 包*

class paddle.fluid.incubate.fleet.base.role_maker.MPISymetricRoleMaker

> - *继承自 `paddle.fluid.incubate.fleet.base.role_maker.MPIRoleMaker`，会对每个物理节点启动两个进程1woker + 1pserver*

class paddle.fluid.incubate.fleet.base.role_maker.PaddleCloudRoleMaker

> - *继承自 `paddle.fluid.incubate.fleet.base.role_maker.RoleMakerBase`，是一个高级封装，会从环境变量中读取分布式训练的节点信息，如*
>
> *PADDLE_PSERVERS_IP_PORT_LIST*
>
> *PADDLE_TRAINER_ENDPOINTS*
>
> *PADDLE_TRAINERS_NUM*
>
> *TRAINING_ROLE*
>
> *PADDLE_TRAINER_ID*
>
> *PADDLE_PSERVER_ID*
>
> *POD_IP*
>
> *PADDLE_PORT*
>
> - *支持使用paddle.distributed.launch 或者 paddle.distributed.launch_ps启动脚本*

方法

- PaddleCloudRoleMaker(is_collective=False) -> PaddleCloudRoleMaker
  - **参数**:
    - is_collective(bool): 是否是collective训练模式还是Parameter Server训练模式
  - **返回**:
    - `PaddleCloudRoleMaker` 对象， 用作 `fleet.init(role_maker = None)` 的初始化

class paddle.fluid.incubate.fleet.base.role_maker.GeneralRoleMaker

> - *继承自 `paddle.fluid.incubate.fleet.base.role_maker.RoleMakerBase`，为了通用的RoleMaker的使用，如分布式文件存储 `HDFS`*

方法

- GeneralRoleMaker(**kwargs) –> GeneralRoleMaker
  - **参数**:
  - kwargs(dict) – 收集关键字参数, 配置自定义的RoleMaker
  - **返回**:
  - `GeneralRoleMaker` 对象， 用作 `fleet.init(role_maker = None)` 的初始化

class paddle.fluid.incubate.fleet.base.role_maker.UserDefinedRoleMaker

> - *人为手工指定trainer和woker在物理节点上的分配*

方法

- UserDefinedRoleMaker(current_id=0, role=Role.WORKER, worker_num=0, server_endpoints=None) –> UserDefinedRoleMaker
  - **参数**:
  - current_id(int) – 当前节点的id(trainer id 或 pserver id)
  - role(paddle.fluid.incubate.fleet.base.role_maker.Role) – WORKER 或 SERVER
  - worker_num(int) – 需要指定的worker的个数
  - server_endpoints(list of str) – 每个server的地址信息, 即IP地址与端口号信息
  - **返回**:
  - `UserDefinedRoleMaker` 对象， 用作 `fleet.init(role_maker = None)` 的初始化

class paddle.fluid.Executor(place=None)

- Executor支持单GPU、多GPU以及CPU运行。
  - **参数**:
  - place (fluid.CPUPlace()，fluid.CUDAPlace(N)，None) — 该参数表示Executor执行所在的设备，这里的N为GPU对应的ID。当该参数为 None 时，PaddlePaddle会根据其安装版本设置默认的运行设备。当安装的Paddle为CPU版时，默认运行设置会设置成 CPUPlace() ，而当Paddle为GPU版时，默认运行设备会设置成 CUDAPlace(0) 。默认值为None。
  - **返回**:
  - 初始化后的 Executor 对象。

方法

- close()

- 关闭执行器。该接口主要用于对于分布式训练，调用该接口后不可以再使用该执行器。该接口会释

放在PServers上和目前Trainer有关联的资源。

- **返回**:无。

方法

- run(program=None, feed=None, fetch_list=None, feed_var_name='feed', fetch_var_name='fetch', scope=None, return_numpy=True, use_program_cache=False, return_merged=True)

- 执行指定的Program或者CompiledProgram。需要注意的是，执行器会执行Program或CompiledProgram中的所有算子，而不会根据fetch_list对Program或CompiledProgram中的算子进行裁剪。同时，需要传入运行该模型用到的scope，如果没有指定scope，执行器将使用全局scope，即fluid.global_scope()。

  - **参数**:
  - program (Program，CompiledProgram) – 该参数为被执行的Program或CompiledProgram，如果未提供该参数，即该参数为None，在该接口内，main_program将被设置为fluid.
  - default_main_program()。默认为：None。
  - feed (list|dict) – 该参数表示模型的输入变量。如果是单卡训练，feed 为 dict 类型，如果是多卡训练，参数 feed 可以是 dict 或者 list 类型变量，如果该参数类型为 dict ，feed中的数据将会被分割(split)并分送给多个设备（CPU/GPU），即输入数据被均匀分配到不同设备上；如果该参数类型为 list ，则列表中的各个元素都会直接分别被拷贝到各设备中。默认为：None。
  - fetch_list (list) – 该参数表示模型运行之后需要返回的变量。默认为：None。
  - feed_var_name (str) – 该参数表示数据输入算子(feed operator)的输入变量名称。默认为："feed"。
  - fetch_var_name (str) – 该参数表示结果获取算子(fetch operator)的输出变量名称。默认为："fetch"。
  - scope (Scope) – 该参数表示执行当前program所使用的作用域，用户可以为不同的program指定不同的作用域。默认值：fluid.global_scope()。
  - return_numpy (bool) – 该参数表示是否将返回的计算结果（fetch list中指定的变量）转化为numpy；如果为False，则每个变量返回的类型为LoDTensor，否则返回变量的类型为numpy.ndarray。默认为：True。
  - use_program_cache (bool) – 该参数表示是否对输入的Program进行缓存。如果该参数为True，在以下情况时，模型运行速度可能会更快：输入的program为 fluid.Program ，并且模型运行过程中，调用该接口的参数（program、 feed变量名和fetch_list变量）名始终不变。默认为：False。

  - return_merged (bool) – 该参数表示是否按照执行设备维度将返回的计算结果（fetch list中指定的变量）进行合并。如果 return_merged 设为False，返回值类型是一个Tensor的二维

列表（ return_numpy 设为Fasle时）或者一个numpy.ndarray的二维列表（ return_numpy 设为True时）。如果 return_merged 设为True，返回值类型是一个Tensor的一维列表（ return_numpy 设为Fasle时）或者一个numpy.ndarray的一维列表（ return_numpy 设为True时）。更多细节请参考示例代码2。如果返回的计算结果是变长的，请设置 return_merged 为False，即不按照执行设备维度合并返回的计算结果。该参数的默认值为True，但这仅是为了兼容性考虑，在未来的版本中默认值可能会更改为False。

- **返回**:
- fetch_list中指定的变量值

注意：如果是多卡训练，并且feed参数为dict类型，输入数据将被均匀分配到不同的卡上，例如：使用2块GPU训练，输入样本数为3，即[0, 1, 2]，经过拆分之后，GPU0上的样本数为1，即[0]，GPU1上的样本数为2，即[1, 2]。如果样本数少于设备数，程序会报错，因此运行模型时，应额外注意数据集的最后一个batch的样本数是否少于当前可用的CPU核数或GPU卡数，如果是少于，建议丢弃该batch。
如果可用的CPU核数或GPU卡数大于1，则fetch出来的结果为不同设备上的相同变量值（fetch_list中的变量）在第0维拼接在一起。

方法

- _run_impl(self, program, feed, fetch_list, feed_var_name,fetch_var_name, scope, return_numpy, use_program_cache, return_merged, use_prune):

run()方法通过调用_run_impl()方法，进而调用compiler._compile()根据用户提供内容编译文件；调用_prune_program()实现对程序对剪枝；如果程序未编译，则调用 _run_program()对程序进行编译；通过调用_run_inference()判断是否为测试模型，若是调用_run_inference()方法，实现对程序对测试，若否则调用_run_parallel()方法，进而调用调用feed_and_split_tensor_into_local_scope()将输入分配到不同的GPU或CPU得到程序输出结果并返回。

- **参数**:
  - program (Program，CompiledProgram) – 该参数为被执行的Program或CompiledProgram，如果未提供该参数，即该参数为None，在该接口内，main_program将被设置为fluid.
  - default_main_program()。默认为：None。
  - feed (list|dict) – 该参数表示模型的输入变量。如果是单卡训练，feed 为 dict 类型，如果是多卡训练，参数 feed 可以是 dict 或者 list 类型变量，如果该参数类型为 dict ，feed中的数据将会被分割(split)并分送给多个设备（CPU/GPU），即输入数据被均匀分配到不同设备上；如果该参数类型为 list ，则列表中的各个元素都会直接分别被拷贝到各设备中。默认为：None。
  - fetch_list (list) – 该参数表示模型运行之后需要返回的变量。默认为：None。
  - feed_var_name (str) – 该参数表示数据输入算子(feed operator)的输入变量名称。默认

为："feed"。
- fetch_var_name (str) – 该参数表示结果获取算子(fetch operator)的输出变量名称。默认为："fetch"。
- scope (Scope) – 该参数表示执行当前program所使用的作用域，用户可以为不同的program指定不同的作用域。默认值：fluid.global_scope()。
- return_numpy (bool) – 该参数表示是否将返回的计算结果（fetch list中指定的变量）转化为numpy；如果为False，则每个变量返回的类型为LoDTensor，否则返回变量的类型为numpy.ndarray。默认为：True。
- use_program_cache (bool) – 该参数表示是否对输入的Program进行缓存。如果该参数为True，在以下情况时，模型运行速度可能会更快：输入的program为 fluid.Program ，并且模型运行过程中，调用该接口的参数（program、 feed变量名和fetch_list变量）名始终不变。默认为：False。
- return_merged (bool) – 该参数表示是否按照执行设备维度将返回的计算结果（fetch list中指定的变量）进行合并。如果 return_merged 设为False，返回值类型是一个Tensor的二维列表（ return_numpy 设为Fasle时）或者一个numpy.ndarray的二维列表（ return_numpy 设为True时）。如果 return_merged 设为True，返回值类型是一个Tensor的一维列表（ return_numpy 设为Fasle时）或者一个numpy.ndarray的一维列表（ return_numpy 设为True时）。更多细节请参考示例代码2。如果返回的计算结果是变长的，请设置 return_merged 为False，即不按照执行设备维度合并返回的计算结果。该参数的默认值为True，但这仅是为了兼容性考虑，在未来的版本中默认值可能会更改为False。

- **返回**:
  - fetch_list中指定的变量值

方法

- _run_program(self, program, feed, fetch_list, feed_var_name, fetch_var_name, scope, return_numpy, use_program_cache)

对程序进行编译。

- **参数**:
  - program (Program，CompiledProgram) – 该参数为被执行的Program或CompiledProgram，如果未提供该参数，即该参数为None，在该接口内，main_program将被设置为fluid.
  - feed (list|dict) – 该参数表示模型的输入变量。如果是单卡训练，feed 为 dict 类型，如果是多卡训练，参数 feed 可以是 dict 或者 list 类型变量，如果该参数类型为 dict ，feed中的数据将会被分割(split)并分送给多个设备（CPU/GPU），即输入数据被均匀分配到不同设备上；如果该参数类型为 list ，则列表中的各个元素都会直接分别被拷贝到各设备中。默认为：None。

- fetch_list (list) – 该参数表示模型运行之后需要返回的变量。默认为：None。
- feed_var_name (str) – 该参数表示数据输入算子(feed operator)的输入变量名称。默认为："feed"。
- fetch_var_name (str) – 该参数表示结果获取算子(fetch operator)的输出变量名称。默认为："fetch"。
- scope (Scope) – 该参数表示执行当前program所使用的作用域，用户可以为不同的program指定不同的作用域。默认值：fluid.global_scope()。
- return_numpy (bool) – 该参数表示是否将返回的计算结果（fetch list中指定的变量）转化为numpy；如果为False，则每个变量返回的类型为LoDTensor，否则返回变量的类型为numpy.ndarray。默认为：True。
- use_program_cache (bool) – 该参数表示是否对输入的Program进行缓存。如果该参数为True，在以下情况时，模型运行速度可能会更快：输入的program为 fluid.Program ，并且模型运行过程中，调用该接口的参数（program、 feed变量名和fetch_list变量）名始终不变。默认为：False。

- **返回**:

  - fetch_list中指定的变量值

方法

- _run_inference(self, exe, feed)

对程序进行测试。

- **参数**:

  - exe – 该参数表示程序对应的执行器。
  - feed – 该参数表示模型的输入变量。如果是单卡训练，feed 为 dict 类型，如果是多卡训练，参数 feed 可以是 dict 或者 list 类型变量，如果该参数类型为 dict ，feed中的数据将会被分割(split)并分送给多个设备（CPU/GPU），即输入数据被均匀分配到不同设备上；如果该参数类型为 list ，则列表中的各个元素都会直接分别被拷贝到各设备中。默认为：None。

- _split_optimize_ops_in_fetch_list(self, fetch_list):

将fetch_list中的optimize_ops拆分出来，用于指定程序的修剪。

- **参数**:

- fetch_list(list) – 原本的fetch_list。

  - fetch_list可能对类型为：
    - fetch_list = ['loss']
    - fetch_list = [[sgd, sgd], 'loss']

        - fetch_list = [([sgd, sgd], [(param, grad)]), 'loss']

- **返回**:
    - 不包含优化算子的fetch_list和由优化算子构成的列表。