ModelArts

模型训练

文档版本 01

发布日期 2026-03-13

华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2026。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 模型训练使用流程 1
2 准备模型训练代码 4
2.1 预置框架启动文件的启动流程说明 4
2.2 开发用于预置框架训练的代码 13
2.3 开发用于自定义镜像训练的代码 15
2.4 自定义镜像训练作业配置实例间 SSH 免密互信 18
3 准备模型训练镜像 20
4 创建调试训练作业 23
4.1 创建调试训练作业并使用 JupyterLab 调试 23
4.2 使用 PyCharm ToolKit 创建并调试训练作业 40
4.3 使用 VS Code 创建并调试训练作业 44
4.4 SSH 远程开发 50
5 创建算法 59
6 创建训练作业 68
6.1 创建训练自定义作业（推荐） 68
6.2 创建模型精调作业（推荐） 80
6.3 创建生产训练作业 85
7 分布式模型训练 105
7.1 分布式训练功能介绍 105
7.2 创建单机多卡的分布式训练（DataParallel） 106
7.3 创建多机多卡的分布式训练（DistributedDataParallel） 109
7.4 示例：创建 DDP 分布式训练（PyTorch+GPU） 118
7.5 示例：创建 DDP 分布式训练（PyTorch+NPU） 122
7.6 示例：创建 Ray 集群 124
8 模型训练存储加速 126
9 训练作业动态路由加速 128
10 增量模型训练 131
11 自动模型优化（AutoSearch） 135
11.1 自动模型优化介绍 135

11.2 创建自动模型优化的训练作业 137
12 模型训练高可靠性 140
12.1 设置断点续训练 140
12.2 训练作业容错检查 146
12.3 训练作业卡死检测 151
12.4 训练作业故障恢复 154
12.5 训练日志失败分析 162
12.6 配置算子重执行 163
13 超节点亲和组实例数配置 170
14 管理模型训练作业 173
14.1 训练大盘监控 173
14.2 查看训练作业列表和详情（新版） 175
14.3 查看训练作业列表和详情 183
14.4 训练作业流程可视化 189
14.5 查看训练作业事件 190
14.6 查看训练作业日志 192
14.7 使用 Cloud Shell 调试生产训练作业 198
14.8 查看训练作业资源占用情况 202
14.9 查看训练作业监控指标 205
14.10 使用 CTS 审计 ModelArts 服务 209
14.11 智能运维 211
14.12 查看模型评估结果 213
14.13 查看训练作业标签 217
14.14 修改训练作业优先级 217
14.15 保存调试模式训练作业镜像 219
14.16 复制、停止或删除训练作业 220
14.17 管理训练容器环境变量 221
14.18 管理训练实验 229
15 Moxing 233
15.1 MoXing 基础功能 233
16 待下线 239
16.1 创建生产训练作业（旧版） 239

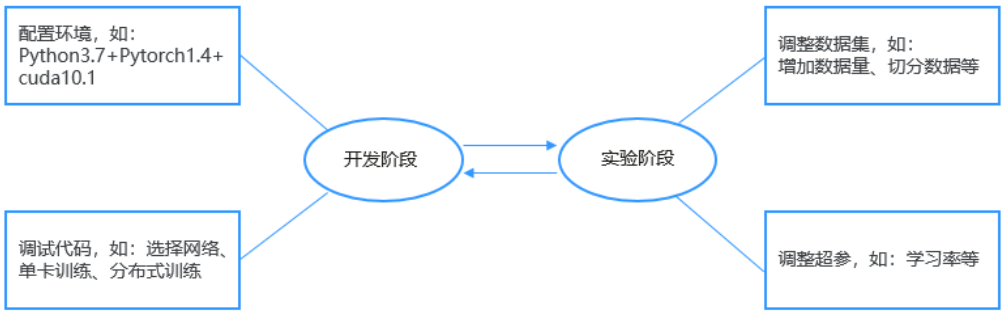
1 模型训练使用流程

AI模型开发的过程，称之为Modeling，一般包含两个阶段：

- 开发阶段：准备并配置环境，调试代码，使代码能够开始进行深度学习训练，推荐在ModelArts开发环境中调试。
- 实验阶段：调整数据集、调整超参等，通过多轮实验，训练出理想的模型，推荐在ModelArts训练中进行实验。

两个过程可以相互转换。如开发阶段代码稳定后，则会进入实验阶段，通过不断尝试调整超参来迭代模型；或在实验阶段，有一个可以优化训练的性能的想法，则会回到开发阶段，重新优化代码。

图 1-1 模型开发过程



ModelArts提供了模型训练的功能，方便您查看训练情况并不断调整您的模型参数。您还可以基于不同的数据，选择不同规格的资源池用于模型训练。

请参考以下指导在ModelArts上训练模型。

表 1-1 模型训练流程

操作任务	子任务	说明
准备工作	准备训练代码	模型训练必备要素包括训练代码、训练框架、训练数据。 训练代码包含训练作业的启动文件或启动命令、训练依赖包等内容。 ● 当使用预置框架创建训练作业时，训练代码的开发规范可以参考**开发用于预置框架训练的代码**。 ● 当使用自定义镜像创建训练作业时，训练代码的开发规范可以参考**开发用于自定义镜像训练的代码**。
	准备训练框架（即训练镜像）	模型训练有多种训练框架来源，具体可以参考**准备模型训练镜像**。 ● ModelArts平台提供了模型训练常用的预置框架，可以直接使用。 ● 当预置框架不满足训练要求时，支持用户构建自定义镜像用于训练。
	准备训练数据	训练数据除了训练数据集，也可以是预测模型。在创建训练作业前，需要先准备好训练数据。 ● 当训练数据可以直接使用，无需二次处理时，可以直接将数据上传至OBS桶。在创建训练作业时，训练的输入参数位置可以直接填写OBS桶路径。 ● 当训练数据集的数据未标注或者需要进一步的数据预处理，可以先将数据导入ModelArts数据管理模块进行数据预处理。在创建训练作业时，训练的输入参数位置可以选择数据管理模块的数据集。
创建调试训练作业	调试训练作业	模型训练前，一般会先对代码进行调试，ModelArts提供多种方式创建调试训练作业。 ● ModelArts提供了云化版本的JupyterLab，无需关注安装配置，即开即用。调试模式的训练作业支持连接JupyterLab，云上调试和运行代码，调试完成的代码可以直接创建生产训练作业。参考**使用JupyterLab创建并调试训练作业**。 ● ModelArts也提供了本地IDE的方式开发模型，通过开启SSH远程开发，本地IDE可以远程连接到调试训练作业中，进行调试和运行代码。本地IDE方式不影响用户的编码习惯，并且调试完成的代码可以零成本直接创建生产训练作业。
创建算法	创建算法	创建生产训练作业之前，需要先准备算法，可以是用户自己准备的算法，也可以使用从AI Gallery订阅的算法。
创建生产训练作业	训练作业基础功能	● ModelArts支持通过Console控制台的可视化界面创建训练作业，创建时基于算法来源和训练框架又区分多种创建方式，具体请参见**表1-2**。 ● ModelArts也支持通过调用API接口创建训练作业，请参见**以PyTorch框架创建训练作业**。

操作任务	子任务	说明
	训练作业进阶功能	ModelArts还支持以下训练进阶功能，例如： ● 增量训练 ● 分布式训练 ● 训练加速 ● 训练高可靠性
查看训练结果和日志	查看训练作业详情	训练作业运行中或运行结束后，可以在训练作业详情页面查看训练作业的参数设置，训练作业事件等。
	查看训练作业日志	训练日志用于记录训练作业运行过程和异常信息，可以通过查看训练作业日志定位作业运行中出现的问题。

表 1-2 训练作业的创建方式介绍

创建方式	适用场景
使用预置框架创建训练作业	如果您已在本地使用一些常用框架完成算法开发，您可以选择常用框架，创建训练作业来构建模型
使用自定义镜像创建训练作业	如果您开发算法时使用的框架并不是常用框架，您可以将算法构建为一个自定义镜像，通过自定义镜像创建训练作业。
使用已有算法创建训练作业	算法管理中，管理了用户自己创建的算法和AI Gallery订阅的算法，您可以使用算法管理中的算法，快速创建训练作业，构建模型。
使用订阅算法创建训练作业	AI Gallery中提供了现成的算法，供用户使用，您可以直接订阅AI Gallery中的算法，快速创建训练作业，构建模型。

2 准备模型训练代码

2.1 预置框架启动文件的启动流程说明

ModelArts训练服务为用户预置了多种主流AI框架，并针对不同框架的特点进行了专门的适配优化。在使用这些预置框架进行模型训练时，用户需要根据所选框架的特点，相应地调整训练启动命令，以确保训练任务能够顺利执行。

ModelArts提供4种预置框架，便于用户在创建训练作业时根据需求直接调整使用。4种框架如下：

表 2-1 预置框架对比

场景分类	预置框架名称	解释说明		优势与使用建议	
NPU场景	**Ascend-Powered-Engine框架启动**	是适配加速芯片Ascend的一组AI框架+运行环境+启动方式的集合。 有三种启动方式。	**启动方式一：使用RTF文件启动训练作业**	用户使用Ascend NPU时选择。 可提升模型训练的效率。	RTF：通用场景。
			启动方式二：使用torchrun命令启动训练作业		torchrun：使用PyTorch+Ascend Extension for PyTorch框架。
			启动方式三：使用msrun命令启动训练作业		msrun：安装MindSpore后即可使用。不依赖第三方库以及配置文件，具有容灾恢复功能，安全性较好。

场景分类	预置框架名称	解释说明	优势与使用建议
GPU场景	PyTorch-GPU框架启动	基于PyTorch-GPU的预置框架。	灵活性和易用性上占优，适合研究和快速开发。
	TensorFlow-GPU框架启动	基于TensorFlow-GPU的预置框架。	生产部署和性能优化上更好，适合企业应用。
	Horovod/MPI/MindSpore-GPU框架启动	使用Horovod/MPI/MindSpore-GPU预置框架来运行的启动文件，平台自动以mpirun命令启动。	此方式依赖开源库OpenMPI，推荐有OpenMPI使用经验的用户使用此种启动方式。

本章节详细介绍基于不同的预置框架启动原理，在创建训练作业时，用户如何修改训练的启动文件。

约束限制

使用torchrun命令启动训练作业时，要求PyTorch版本大于等于1.11.0。

使用msrun命令启动训练作业，要求MindSpore版本大于等于2.3.0。

Ascend-Powered-Engine 框架启动

在ModelArts创建训练作业界面选择AI框架时，有一个AI框架是“Ascend-Powered-Engine”，这一选项并不是传统意义上的AI框架（如PyTorch、TensorFlow），也不是并行执行框架（如MPI），而是为适配昇腾（Ascend）加速芯片而设计的一组AI框架、运行环境和启动方式的集合。

由于Snt9系列Ascend加速卡运行在ARM架构的CPU环境中，因此其对应的Docker镜像也均为ARM架构镜像。与GPU场景下镜像中安装的CUDA计算库（由英伟达推出）不同，“Ascend-Powered-Engine”的镜像中安装的是CANN计算库（华为针对AI场景推出的异构计算架构），以实现与昇腾驱动的适配。

在提交训练作业后，ModelArts平台会自动运行训练作业的启动文件。另外，使用“Ascend-Powered-Engine”框架时，其单机启动命令和分布式启动命令在形式上并无区别。

Ascend-Powered-Engine框架支持3种启动方式来启动“启动文件”，默认是基于“RANK_TABLE_FILE”启动，但用户也可以通过配置环境变量“MA_RUN_METHOD”选择其他方式启动。MA_RUN_METHOD环境变量支持“torchrun”和“msrun”两种启动方式，为用户提供灵活的选择。

- **启动方式一：使用RTF文件启动训练作业**

警告

MindSpore 2.7.2及以后的版本不支持ranktable启动方式，因此建议优先使用msrun方式进行启动。

在没有配置环境变量“MA_RUN_METHOD”时，ModelArts平台默认使用rank table file（RTF）文件启动训练作业的“启动文件”。

每个训练作业的启动文件的运行次数取决于任务卡数，即在训练作业运行时，有N个任务卡数训练作业内就会运行N次启动文件。例如，单机1卡，则worker-0任务的启动文件会被运行1次；单机8卡，则worker-0任务的启动文件会被运行8次。因此需要避免在启动文件中进行端口监听。

启动文件会被自动设置如下环境变量：

- RANK_TABLE_FILE：RTF文件路径。
- ASCEND_DEVICE_ID：逻辑device_id，例如单卡训练，该值始终为 0。
- RANK_ID：可以理解为训练作业级的device逻辑（顺序）编号。
- RANK_SIZE：根据RTF中device的数目设置该值，例如“4 * Snt9b”，则该值即为4。

当需要启动文件仍然在逻辑上仅运行1次时，则可以在启动文件中判断“ASCEND_DEVICE_ID”的值，当值为“0”则执行逻辑，当值为非0则直接退出。

当需要启用ranktable动态路由进行训练网络加速时，则可以添加环境变量“ROUTE_PLAN=true”。详细使用指导请参见**训练作业动态路由加速**。

Ascend-Powered-Engine框架对应的代码示例“mindspore-verification.py”如下：

```
import os
import numpy as np
from mindspore import Tensor
import mindspore.ops as ops
import mindspore.context as context

print('Ascend Envs')
print('------')
print('JOB_ID: ', os.environ['JOB_ID'])
print('RANK_TABLE_FILE: ', os.environ['RANK_TABLE_FILE'])
print('RANK_SIZE: ', os.environ['RANK_SIZE'])
print('ASCEND_DEVICE_ID: ', os.environ['ASCEND_DEVICE_ID'])
print('DEVICE_ID: ', os.environ['DEVICE_ID'])
print('RANK_ID: ', os.environ['RANK_ID'])
print('------')

context.set_context(device_target="Ascend")
x = Tensor(np.ones([1,3,3,4]).astype(np.float32))
y = Tensor(np.ones([1,3,3,4]).astype(np.float32))

print(ops.add(x, y))
```

- **启动方式二：使用torchrun命令启动训练作业**

  当环境变量“MA_RUN_METHOD=torchrun”时，表示ModelArts平台使用torchrun命令启动训练作业的“启动文件”。

  - 单机时，ModelArts平台使用如下命令启动训练作业的“启动文件”。
    ```
    torchrun --standalone --nnodes=${MA_NUM_HOSTS} --nproc_per_node=${MA_NUM_GPUS} $
    {MA_EXTRA_TORCHRUN_PARAMS} "启动文件" {arg1} {arg2} ...
    ```
  - 多机时，ModelArts平台使用如下命令启动训练作业的“启动文件”。
    ```
    torchrun --nnodes=${MA_NUM_HOSTS} --nproc_per_node=${MA_NUM_GPUS} --node_rank=$
    {VC_TASK_INDEX} --master_addr={master_addr} --master_port=$
    {MA_TORCHRUN_MASTER_PORT} --rdzv_id={ma_job_name} --rdzv_backend=static $
    {MA_EXTRA_TORCHRUN_PARAMS} "启动文件" {arg1} {arg2} ...
    ```

  参数说明如下：

表 2-2 使用 torchrun 命令启动训练作业参数说明

参数名	参数解释
standalone	标识为单任务实例作业。
nnodes	任务实例个数。
nproc_per_node	每个任务实例启动的主进程数，设置为任务分配的NPU数相同。
node_rank	任务rank，用于多任务分布式训练。
master_addr	主任务（rank 0）的地址，设置为任务worker-0的通信域名。
master_port	在主任务（rank 0）上，用于分布式训练期间通信的端口。默认设置为18888端口。当遇到master_port冲突问题时，可通过设置MA_TORCHRUN_MASTER_PORT环境变量值修改端口配置。
rdzv_id	Rendezvous标识，设置为带有训练作业ID的值。
rdzv_backend	Rendezvous后端，固定设置为static，即不使用Rendezvous，而是使用master_addr和master_port配置。

– 另外，可通过设置MA_EXTRA_TORCHRUN_PARAMS环境变量值，以增加额外的torchrun命令参数，或是覆盖预设的torchrun命令参数。例如配置torchrun命令中rdzv_conf参数的训练作业API环境变量的部分示例如下：

```
"environments": {
"MA_RUN_METHOD": "torchrun",
"MA_EXTRA_TORCHRUN_PARAMS": "--rdzv_conf=timeout=7200"
}
```

如果在torchrun初始化分布式一致性协商阶段出现“RuntimeError：Socket Timeout”错误时，可以查看常见问题1，进一步排查问题。

- **启动方式三：使用msrun命令启动训练作业**

  当环境变量“MA_RUN_METHOD=msrun”时，表示ModelArts平台使用msrun命令启动训练作业的“启动文件”。

  该方案支持动态组网和基于rank table file文件组网两种方式。当配置了环境变量MS_RANKTABLE_ENABLE="True"，则msrun会读取rank table file文件内容进行组网。否则默认使用动态组网。

  msrun使用如下命令启动训练作业的“启动文件”。

```
msrun --worker_num=${msrun_worker_num} --local_worker_num=${MA_NUM_GPUS} --master_addr=${msrun_master_addr} --node_rank=${VC_TASK_INDEX} --master_port=${msrun_master_port} --log_dir=${msrun_log_dir} --join=True --cluster_time_out=${MSRUN_CLUSTER_TIME_OUT} --rank_table_file=${msrun_rank_table_file} "启动文件" {arg1} {arg2} ...
```

  参数说明如下：

表 2-3 msrun 命令启动训练作业参数说明

参数名	参数解释
worker_num	所有进程个数。因为一个卡起一个进程，所以也表示使用总卡数。
local_worker_num	当前节点进程个数，即当前节点使用的卡数。
master_addr	msrun组网调度进程所在节点的IP地址，单机场景无需配置。
master_port	msrun组网调度进程的端口。
node_rank	msrun组网调度进程的端口。
log_dir	msrun组网和各个进程的日志输出地址。
join	训练进程拉起后，msrun进程是否仍存在，默认配置为“True”，等待所有进程退出后再退出。
cluster_time_out	集群组网超时时间，默认是“600s”，可通过环境变量“MSRUN_CLUSTER_TIME_OUT”控制。
rank_table_file	rank table file文件地址，如果配置了环境变量“MS_RANKTABLE_ENABLE="True"”，启动时会增加该参数。

PyTorch-GPU 框架启动

单机多卡场景下平台会为启动文件额外拼接 --init_method "tcp://<ip>:<port>" 参数。

多机多卡场景下平台会为启动文件额外拼接 --init_method "tcp://<ip>:<port>" --rank <rank_id> --world_size <node_num>参数。

启动文件需要解析上述参数。

PyTorch-GPU框架的代码示例，请参见**示例：创建DDP分布式训练（PyTorch+GPU）**中的方式一。

TensorFlow-GPU 框架启动

单机场景下（即选择的实例数为1），ModelArts只会在一个节点上启动一个训练容器，该训练容器独享节点规格的可使用资源。

多机场景下（即选择的实例数大于1），ModelArts会优先在相同节点上启动一个parameter server（以下简称ps）和一个worker，平台会自动一比一分配ps与worker任务。例如，双机场景会分配2个ps和2个worker任务，并为启动文件额外注入如下参数。

```
--task_index <VC_TASK_INDEX> --ps_hosts <TF_PS_HOSTS> --worker_hosts <TF_WORKER_HOSTS> --job_name <MA_TASK_NAME>
```

启动文件需要解析如下参数。

表 2-4 参数说明

参数名	参数解释
VC_TASK_INDEX	task序号，如0、1、2。
TF_PS_HOSTS	ps节点地址数组，如“[xx-ps-0.xx:TCP_PORT,xx-ps-1.xx:TCP_PORT]”，TCP_PORT是一个在5000~10000的随机端口。
TF_WORKER_HOSTS	worker节点地址数组，如“[xx-worker-0.xx:TCP_PORT,xx-worker-1.xx:TCP_PORT]”，TCP_PORT是一个在5000~10000的随机端口。
MA_TASK_NAME	任务名称，取值是ps或worker。

Horovod/MPI/MindSpore-GPU 框架启动

使用Horovod/MPI/MindSpore-GPU预置框架来运行的启动文件，平台自动以mpirun命令启动训练作业。使用ModelArts训练相应预置引擎，用户仅需关注启动文件（即训练脚本）的编写；mpirun命令和训练作业集群的构建都由平台自动完成。平台不会为启动文件额外拼接参数。

“pytorch_synthetic_benchmark.py”文件示例如下：

```
import argparse
import torch.backends.cudnn as cudnn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data.distributed
from torchvision import models
import horovod.torch as hvd
import timeit
import numpy as np

# Benchmark settings
parser = argparse.ArgumentParser(description='PyTorch Synthetic Benchmark',
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
                    help='use fp16 compression during allreduce')

parser.add_argument('--model', type=str, default='resnet50',
                    help='model to benchmark')
parser.add_argument('--batch-size', type=int, default=32,
                    help='input batch size')

parser.add_argument('--num-warmup-batches', type=int, default=10,
                    help='number of warm-up batches that don\'t count towards benchmark')
parser.add_argument('--num-batches-per-iter', type=int, default=10,
                    help='number of batches per benchmark iteration')
parser.add_argument('--num-iters', type=int, default=10,
                    help='number of benchmark iterations')

parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')

parser.add_argument('--use-adasum', action='store_true', default=False,
                    help='use adasum algorithm to do reduction')

args = parser.parse_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()

hvd.init()
```

```
if args.cuda:
    # Horovod: pin GPU to local rank.
    torch.cuda.set_device(hvd.local_rank())

cudnn.benchmark = True

# Set up standard model.
model = getattr(models, args.model)()

# By default, Adasum doesn't need scaling up learning rate.
lr_scaler = hvd.size() if not args.use_adasum else 1

if args.cuda:
    # Move model to GPU.
    model.cuda()
    # If using GPU Adasum allreduce, scale learning rate by local_size.
    if args.use_adasum and hvd.nccl_built():
        lr_scaler = hvd.local_size()

optimizer = optim.SGD(model.parameters(), lr=0.01 * lr_scaler)

# Horovod: (optional) compression algorithm.
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     compression=compression,
                                     op=hvd.Adasum if args.use_adasum else hvd.Average)

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

# Set up fixed fake data
data = torch.randn(args.batch_size, 3, 224, 224)
target = torch.LongTensor(args.batch_size).random_() % 1000
if args.cuda:
    data, target = data.cuda(), target.cuda()


def benchmark_step():
    optimizer.zero_grad()
    output = model(data)
    loss = F.cross_entropy(output, target)
    loss.backward()
    optimizer.step()


def log(s, nl=True):
    if hvd.rank() != 0:
        return
    print(s, end='\n' if nl else '')


log('Model: %s' % args.model)
log('Batch size: %d' % args.batch_size)
device = 'GPU' if args.cuda else 'CPU'
log('Number of %ss: %d' % (device, hvd.size()))

# Warm-up
log('Running warmup...')
timeit.timeit(benchmark_step, number=args.num_warmup_batches)

# Benchmark
log('Running benchmark...')
img_secs = []
for x in range(args.num_iters):
```

```
    time = timeit.timeit(benchmark_step, number=args.num_batches_per_iter)
    img_sec = args.batch_size * args.num_batches_per_iter / time
    log('Iter #%d: %.1f img/sec per %s' % (x, img_sec, device))
    img_secs.append(img_sec)

# Results
img_sec_mean = np.mean(img_secs)
img_sec_conf = 1.96 * np.std(img_secs)
log('Img/sec per %s: %.1f +-%.1f' % (device, img_sec_mean, img_sec_conf))
log('Total img/sec on %d %s(s): %.1f +-%.1f' %
    (hvd.size(), device, hvd.size() * img_sec_mean, hvd.size() * img_sec_conf))
```

run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done
```

```
        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

常见问题

1. 在torchrun初始化分布式一致性协商阶段出现“RuntimeError：Socket Timeout”错误时，如何解决？

   如果在torchrun初始化分布式一致性协商阶段出现RuntimeError：Socket Timeout错误时，可以通过增加如下环境变量再次创建训练作业以查看torchrun初始化阶段的详细信息，进一步排查问题。

   - LOGLEVEL=INFO
   - TORCH_CPP_LOG_LEVEL=INFO
   - TORCH_DISTRIBUTED_DEBUG=DETAIL

   出现RuntimeError: Socket Timeout错误，一般是因为不同任务执行torchrun命令的时机差距过大导致的。torchrun命令执行时机差距过大，又多是因为在torchrun命令被执行之前任务还有一些初始化动作，例如下载训练数据集、CKPT等。这些初始化动作执行耗时差距过大会直接导致出现Socket Timeout错误。所以遇到Socket Timeout问题时首先需要排查的是各个任务执行torchrun的时间点差距是否在合理范围内，如果时间点差距过大，需要优化执行torchrun命令之前的初始化动作，使其时间点差距在合理范围内。

2.2 开发用于预置框架训练的代码

在使用ModelArts平台进行算法开发时，开发者需要利用预置框架来构建和训练模型。然而，预置框架虽然提供了基础功能，但可能无法完全满足特定的业务需求，需要进行定制化开发。为了确保预置框架能够与具体的业务需求完美结合，开发者需要在使用预置框架创建算法之前，预先完成算法代码的开发工作。通过预先开发，可以确保预置框架能够高效地支持模型训练和优化，从而实现更高效的算法开发流程。

配置路径说明

创建算法时，您需要在创建页面提供**代码目录路径**、**代码目录路径中的启动文件**、**训练输入路径参数**和**训练输出路径参数**。这四种输入搭建了用户代码和ModelArts后台交互的桥梁。

- **代码目录路径**

  您需要在OBS桶中指定代码目录，并将训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。训练作业创建完成后，ModelArts会将代码目录及其子目录下载至后台容器中。

  例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。

  请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。建议训练代码目录大小小于或等于50MB。

- **代码目录路径中的启动文件**

  代码目录路径中的启动文件作为训练启动的入口，当前只支持python格式。预置框架启动文件的启动流程说明请参见**预置框架启动文件的启动流程说明**。

- **训练输入路径参数**

  训练数据需上传至OBS桶或者存储至**数据集**中。在训练代码中，用户需解析输入路径参数。系统后台会自动下载输入参数路径中的训练数据至训练容器的本地目

录。请保证您设置的桶路径有读取权限。在训练作业启动后，ModelArts会挂载硬盘至“/cache”目录，用户可以使用此目录来存储临时文件。“/cache”目录大小请参考**训练环境中不同规格资源“/cache”目录的大小**。

- **训练输出路径参数**

  建议设置一个空目录为训练输出路径。在训练代码中，您需要解析输出路径参数。系统后台会自动上传训练输出至指定的训练输出路径，请保证您设置的桶路径有写入权限和读取权限。

在ModelArts中，训练代码需包含**（可选）引入依赖**和**解析和设置输入路径参数、输出路径参数**步骤。

（可选）引入依赖

当您使用自定义脚本创建算法的时候，如果您的模型引用了其他依赖，您需要在“算法管理 > 创建算法”的“代码目录”下放置相应的文件或安装包。

- 安装python依赖包请参考**模型中引用依赖包时，如何创建训练作业?**
- 安装C++的依赖库请参考**如何安装C++的依赖库?**
- 在预训练模型中加载参数请参考**如何在训练中加载部分训练好的参数?**

解析和设置输入路径参数、输出路径参数

运行在ModelArts的训练作业会读取存储在OBS服务的数据，或者输出训练结果至OBS服务指定路径，输入和输出数据需要配置2个地方：

1. 训练代码中需解析输入路径参数和输出路径参数。ModelArts推荐以下方式实现参数解析。

```
import argparse
# 创建解析
parser = argparse.ArgumentParser(description='train mnist')

# 添加参数
parser.add_argument('--data_url', type=str, default="./Data/mnist.npz", help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='path where the model is saved')

# 解析参数
args = parser.parse_args()
```

   完成参数解析后，用户使用“data_url”、“train_url”代替算法中数据来源和数据输出所需的路径。

2. 在创建训练作业时，填写输入路径和输出路径。

   训练输入选择对应的OBS路径或者数据集路径；训练输出选择对应的OBS路径。

训练代码完整示例

训练代码示例中涉及的代码与您使用的AI引擎密切相关，以下案例以Tensorflow框架为例。案例中使用到的“mnist.npz”文件需要提前**下载**并上传至OBS桶中，训练输入为“mnist.npz”所在OBS路径。

以下训练代码样例中包含了保存模型代码。

```
import os
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='train mnist')
parser.add_argument('--data_url', type=str, default="./Data/mnist.npz", help='path where the dataset is
```

```
saved')
parser.add_argument('--train_url', type=str, default="./Model", help='path where the model is saved')
args = parser.parse_args()

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data(args.data_url)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)

model.save(os.path.join(args.train_url, 'model'))
```

2.3 开发用于自定义镜像训练的代码

在使用 ModelArts进行模型训练时，平台提供了多种预置框架和算法，能够满足大部分用户的需求。然而，当预置框架和算法确实无法满足需求时，ModelArts提供了自定义镜像训练的功能，为用户提供了一个灵活的解决方案。通过构建自定义镜像，用户可以完全定制训练环境，满足特定需求。

自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置框架无法满足需求，否则不推荐使用。自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts上训练。上传镜像到SWR服务中，具体操作请参见**如何登录并上传镜像到SWR**。

自定义镜像的启动命令规范

用户遵循ModelArts镜像的规范要求制作镜像，选择自己的镜像，并且通过指定代码目录（可选）和启动命令的方式来创建的训练作业。

图 2-1 创建训练作业选择自定义方式

警告

1. 当使用完全自定义镜像创建训练作业时，“启动命令”必须在“/home/ma-user”目录下执行，否则训练作业可能会运行异常。
2. 对于启动命令，当存在输入管道、输出管道、或是超参的情况下，请保证启动命令的最后一条命令是运行训练脚本，否则执行会报错。

1. 完全使用自定义镜像创建训练作业需要通过指定的“conda env”启动训练。

   由于训练作业运行时不是shell环境，因此无法直接使用“conda activate”命令激活指定的“conda env”，需要使用其他方式以达成使用指定“conda env”来启动训练的效果。假设您的自定义镜像中的“conda”安装于“/home/ma-user/anaconda3”目录“conda env”为“python-3.7.10”，训练脚本位于“/home/ma-user/modelarts/user-job-dir/code/train.py”。可通过以下方式使用指定的“conda env”启动训练：

   - 方式一：为镜像设置正确的“DEFAULT_CONDA_ENV_NAME”环境变量与“ANACONDA_DIR”环境变量。

     ```
     ANACONDA_DIR=/home/ma-user/anaconda3
     DEFAULT_CONDA_ENV_NAME=python-3.7.10
     ```

     您可以使用Python命令启动训练脚本。启动命令示例如下：

     ```
     python /home/ma-user/modelarts/user-job-dir/code/train.py
     ```

   - 方式二：使用“conda env python”的绝对路径。

     您可以使用“/home/ma-user/anaconda3/envs/python-3.7.10/bin/python”命令启动训练脚本。启动命令示例如下：

     ```
     /home/ma-user/anaconda3/envs/python-3.7.10/bin/python /home/ma-user/modelarts/user-job-dir/code/train.py
     ```

   - 方式三：设置PATH环境变量。

     您可以将指定的“conda env bin”目录配置到PATH环境变量中。您可以使用Python命令启动训练脚本。启动命令示例如下：

     ```
     export PATH=/home/ma-user/anaconda3/envs/python-3.7.10/bin:$PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
     ```

   - 方式四：使用“conda run -n”命令。

     您可以使用“/home/ma-user/anaconda3/bin/conda run -n python-3.7.10”命令来执行训练命令，启动命令示例如下：

     ```
     /home/ma-user/anaconda3/bin/conda run -n python-3.7.10 python /home/ma-user/modelarts/user-job-dir/code/train.py
     ```

2. 启动命令的最后一条命令必须为运行训练脚本命令。

   对于启动命令，当存在输入管道、输出管道、或是超参的情况下，请保证启动命令的最后一条命令是运行训练脚本。

   原因：系统会将输入管道、输出管道、以及超参添加到启动命令的末尾，如果最后一条命令不是运行训练脚本则会报错。

   例如：启动命令的最后一条是python train.py，且存在--data_url超参，系统正常运行会执行python train.py --data_url=/input。但是当启动命令python train.py后面有其他命令时，如下所示：

   ```
   python train.py
   pwd    #反例，启动命令的最后一条命令不是运行训练脚本，而是pwd
   ```

   此时，如果拼接了输入管道、输出管道、以及超参，系统运行实际执行的是python train.py pwd --data_url=/input，就会报错。

使用 Ascend 自定义镜像训练时的训练代码适配规范

使用NPU资源创建训练作业时，系统会在训练容器里自动生成Ascend HCCL RANK_TABLE_FILE文件。当使用预置框架创建训练作业时，在训练过程中预置框架会自动解析Ascend HCCL RANK_TABLE_FILE文件，当使用自定义镜像创建训练作业时，就要适配训练代码使得训练过程中在代码里读取解析Ascend HCCL RANK_TABLE_FILE文件。

Ascend HCCL RANK_TABLE_FILE文件说明

Ascend HCCL RANK_TABLE_FILE文件提供Ascend分布式训练作业的集群信息，用于Ascend芯片分布式通信，可以被HCCL集合通信库解析。该文件格式有模板一和模板二两个版本。

- ModelArts提供的是模板二格式。ModelArts训练环境的Ascend HCCL RANK_TABLE_FILE文件名为jobstart_hccl.json，获取方式可以通过预置的RANK_TABLE_FILE环境变量实现。

  表 2-5 RANK_TABLE_FILE 环境变量说明

环境变量	说明
RANK_TABLE_FILE	该环境变量指示Ascend HCCL RANK_TABLE_FILE文件所在目录，值为/user/config。 算法开发者可通过“${RANK_TABLE_FILE}/jobstart_hccl.json”，路径获取该文件。

  ModelArts训练环境jobstart_hccl.json文件内容（模板二）示例：

```
{
    "group_count": "1",
    "group_list": [{
        "device_count": "1",
        "group_name": "job-trainjob",
        "instance_count": "1",
        "instance_list": [{
            "devices": [{
                "device_id": "4",
                "device_ip": "192.1.10.254"
            }],
            "pod_name": "jobxxxxxxxx-job-trainjob-0",
            "server_id": "192.168.0.25"
        }]
    }],
    "status": "completed"
}
```

  jobstart_hccl.json文件中的status字段的值在训练脚本启动时，并不一定为completed状态。因此需要训练脚本等待status字段的值等于completed之后，再去读取文件的剩余内容。

- 通过训练脚本，可以使用模板一格式的jobstart_hccl.json文件，在等待status字段的值等于completed之后，将模板二格式jobstart_hccl.json文件转换为模板一格式的jobstart_hccl.json文件。

  转换后的jobstart_hccl.json文件格式（模板一）示例：

```
{
    "server_count": "1",
    "server_list": [{
        "device": [{
            "device_id": "4",
```

```
            "device_ip": "192.1.10.254",
            "rank_id": "0"
        }],
        "server_id": "192.168.0.25"
    }],
    "status": "completed",
    "version": "1.0"
}
```

训练作业在容器中的挂载点说明

使用自定义镜像训练模型时，训练作业在容器中的挂载点参考如表2-6所示。

表 2-6 训练作业挂载点介绍

挂载点	是否只读	备注
/xxx	否	专属池使用SFS盘挂载的目录，路径由客户自己指定。
/home/ma-user/modelarts	否	空文件夹，建议用户主要用这个目录。
/cache	否	裸机规格支持，挂载宿主机NVMe的硬盘。
/dev/shm	否	用于PyTorch引擎加速。
/usr/local/nvidia	是	宿主机的NV库。

常见问题

1. 在训练时发生找不到“$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib ”目录下“.so”文件的相关报错如何处理?

   如果在训练时发生找不到“$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib ”目录下“.so”文件的相关报错，可以尝试将该目录加入到“LD_LIBRARY_PATH”，将以下命令放在上述启动方式命令前：

   ```
   export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH;
   ```

   例如，方式一的启动命令示例此时变为：

   ```
   export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
   ```

2.4 自定义镜像训练作业配置实例间 SSH 免密互信

在使用基于MPI和Horovod框架的自定义镜像进行分布式训练时，如果未配置训练作业实例间的SSH免密互信，将会导致训练任务失败。为确保实例间的通信顺畅，必须预先配置实例间SSH免密互信，确保分布式训练任务顺利进行。

配置实例间SSH免密互信涉及代码适配和训练作业参数配置，本文提供了一个操作示例。

1. 准备一个预装OpenSSH的自定义镜像，使用的训练框架是MPI或Horovod。

2. 准备一个sshd启动脚本文件“start_sshd.sh”。
```
MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}
mkdir -p /home/ma-user/etc
ssh-keygen -f /home/ma-user/etc/ssh_host_rsa_key0 -N '' -t rsa > /dev/null
/usr/sbin/sshd -p $MY_SSHD_PORT -h /home/ma-user/etc/ssh_host_rsa_key0
```
3. 将准备好的sshd启动脚本文件上传至OBS的训练代码目录下。
4. 创建自定义镜像训练作业。
   - “代码目录”选择存有sshd启动脚本文件的OBS地址。
   - “启动命令”需要适配sshd启动脚本，如下所示：
     ```
     bash ${MA_JOB_DIR}/demo-code/start_sshd.sh && your custom command
     ```
     命令中的“your custom command”表示训练作业中需要执行的其他自定义命令。
   - “环境变量”增加“MY_SSHD_PORT = 38888”。
   - 开启（或勾选）“生成节点间SSH免密互信文件”，并设置“SSH免密互信文件目录”，一般保持默认值。该配置会在下发训练作业后，自动在训练容器的“/home/ma-user/.ssh”目录下生成SSH密钥文件和配置文件“authorized_keys config id_rsa id_rsa.pub”。
5. 提交创建训练作业后，训练过程中，训练作业的实例可通过域名+端口的方式SSH连接到其他实例，示例代码如下所示：
   ```
   ssh modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-1 -p $MY_SSHD_PORT
   ```

3 准备模型训练镜像

ModelArts平台提供了Tensorflow，PyTorch，MindSpore等常用深度学习任务的基础镜像，镜像里已经安装好运行任务所需软件。当基础镜像里的软件无法满足您的程序运行需求时，您还可以基于这些基础镜像制作一个新的镜像并进行训练。

训练作业的预置框架介绍

ModelArts中预置的训练基础镜像如下表所示。

表 3-1 ModelArts 训练基础镜像列表

引擎类型	版本名称
PyTorch	pytorch_2.7.0-cuda_12.8-py_3.11.10-ubuntu_22.04-x86_64
	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
	pytorch_2.1.0-cuda_12.1-py_3.10.6-ubuntu_22.04-x86_64
TensorFlow	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
Horovod	horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
	horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
MPI	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64
Ascend-Powered-Engine	tensorflow_1.15-cann_5.1.0-py_3.7-euler_2.8.3-aarch64
	mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64
	pytorch_2.1.0-cann_7.0.1.1-py_3.9-euler_2.10.7-aarch64-snt3p
	mindspore_2.2.12-cann_7.0.1.1-py_3.9-euler_2.10.7-aarch64-snt3p

构建自定义训练镜像

图 3-1 训练作业的自定义镜像制作流程

场景一：预置镜像满足ModelArts训练平台约束，但不满足代码依赖的要求，需要额外安装软件包。

具体案例参考**使用预置镜像制作自定义镜像用于训练模型**。

场景二：已有本地镜像满足代码依赖的要求，但是不满足ModelArts训练平台约束，需要适配。

具体案例参考**已有镜像迁移至ModelArts用于训练模型**。

场景三： 当前无可使用的镜像，需要从0制作镜像（既需要安装代码依赖，又需要制作出的镜像满足ModelArts平台约束）。具体案例参考：

从0制作自定义镜像用于创建训练作业（Pytorch+Ascend）

从0制作自定义镜像用于创建训练作业（PyTorch+CPU/GPU）

从0制作自定义镜像用于创建训练作业（MPI+CPU/GPU）

从0制作自定义镜像用于创建训练作业（Tensorflow+GPU）

在镜像中安装 pip 依赖

在创建分布式训练作业前，建议提前安装所有的pip依赖，否则，当节点数大于10时，系统会自动删除pip源配置，当训练过程中涉及pip install操作时可能会训练失败。

提前安装依赖，确保所有依赖包在训练前已安装完成，可以避免因节点数过多导致pip源配置被删除而引发训练失败，提升训练作业的稳定性和效率。

安装pip依赖的两种方式：

- 方式一：在Notebook中提前安装依赖并保存镜像
  a. 在Notebook环境中运行目标镜像。
  b. 执行**pip install**命令安装所有依赖包。
  c. 保存镜像。
  d. 在镜像详情页获取SWR地址以便后续训练使用。
- 方式二：在本地容器中安装依赖并导出镜像
  a. 在本地环境中运行容器，执行**pip install**命令安装所有依赖包。
  b. 使用以下命令将容器保存为镜像。
  ```
  docker commit <容器ID> <镜像名称>:<标签>
  ```
  示例：
  ```
  docker commit my_container my_image:v1
  ```
  c. 使用以下命令将镜像导出为tar文件。
  ```
  docker save -o <tar文件名>.tar <镜像名称>:<标签>
  ```

示例：

```
docker save -o my_image_v1.tar my_image:v1
```

d. 将镜像上传至容器镜像服务SWR，用于后续训练作业。

4 创建调试训练作业

4.1 创建调试训练作业并使用 JupyterLab 调试

说明

本章节内容适用除“西南-贵阳一”站点之外的所有华为云站点。控制台界面为旧版样式。

在AI模型开发过程中，高效的调试环境是确保模型质量的关键环节。传统开发模式中，开发人员需要在本地环境中完成模型调试，再进行云端部署，这一过程往往面临环境配置复杂、资源利用率低等问题。

ModelArts调试模式的训练作业提供了一套完整的云端开发解决方案，有效解决了上述问题。该方案主要包含以下核心功能：

1. 云化开发环境支持
   - 集成JupyterLab等主流开发工具链，提供开箱即用的云端开发环境
   - 支持本地IDE通过SSH远程连接，保持开发习惯的同时实现云端调试
2. 灵活的调试方式
   - 提供分布式算法调试能力，满足复杂模型开发需求
   - 调试完成的代码可无缝衔接创建生产训练作业，实现开发到生产的快速转换
3. 便捷的资源管理
   - 云上调试环境即开即用，无需关注安装配置
   - 本地IDE调试模式下，调试完成的代码可零成本迁移至生产环境

通过以上功能，ModelArts调试模式的训练作业为开发人员提供了高效、灵活的云端开发体验，显著提升了模型开发效率和质量。

约束限制

默认支持创建10000条训练作业，剩余额度可以在训练作业列表页查看。

图 4-1 查看训练作业剩余额度

前提条件

- 已经将用于训练作业的数据上传至OBS目录。
- 已经在OBS目录下创建了至少1个空的文件夹，用于存储训练输出的内容。

  说明

  ModelArts不支持加密的OBS桶，创建OBS桶时，请勿开启桶加密。
- 由于训练作业运行需消耗资源，为了避免训练失败请确保账户未欠费。
- 确保使用的OBS目录与ModelArts在同一区域。
- 检查是否配置了访问授权。如果未配置，请参见**快速配置ModelArts委托授权**完成操作。
- 已经上传镜像到SWR服务中，具体操作请参见**如何登录并上传镜像到SWR**。
- 已经准备好训练作业的专属资源池。如需新建资源池请参见**创建资源池**。

计费影响

在ModelArts进行模型训练时，会产生计算资源和存储资源的累计值计费。计算资源为训练作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

操作流程介绍

创建训练作业的操作步骤如下所示。

步骤一：进入创建训练作业页面。

步骤二：基础配置：配置基础信息，例如作业模式、作业名称、描述、实验等。

步骤三：环境配置：配置环境信息，例如创建方式（算法选择）、启动方式、引擎及版本等。

步骤四：训练配置：配置训练作业的输入、输出、超参、环境变量等参数。

步骤五：资源配置：配置资源池类型、实例规格、实例数、挂载存储等信息。

步骤六：高可用配置：配置自动重启（无条件自动重启、作业卡死重启）等信息。

步骤七：调度配置：调整作业优先级、是否允许被抢占、自动停止等。

步骤八：更多配置：配置日志、事件通知和标签等信息。

步骤九：提交训练作业并调试。

步骤一：进入创建训练作业页面

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型训练 > 训练作业”进入训练作业列表。
3. 单击“创建训练作业”，进入创建训练作业页面，默认进入新版页面，以下内容主要介绍新版创建训练页面操作。

步骤二：基础配置

在创建训练作业页面填写调试训练作业的基础配置。

表 4-1 调试训练作业的基础配置

参数名称	说明
作业模式	创建调试训练作业时，必须选择“调试模型”。 训练作业支持在云上和本地对训练代码进行调试，调试完成的代码可直接创建生产训练作业。 ● “生产模式”：在生产环境运行训练作业，初次上云训练建议先使用调试模式，再到生产环境运行，避免产生资源浪费。 ● “调试模式”：可以实时修改训练代码并迭代调试。 “生产模式”，则请参考**创建生产训练作业**创建生产训练作业。
作业名称	必填，训练作业的名称。 系统会自动生成一个名称，可以根据业务需求重新命名，命名规则如下： ● 支持1~64位字符。 ● 可以包含大小写字母、数字、中划线（-）或下划线（_）。
描述（可选）	训练作业的简介，便于在训练作业列表了解作业信息。
实验设置	勾选“实验设置”后，可以将训练作业放入实验中分类，并有序地进行管理，常用于多任务的版本管理等场景。 实验功能有助于管理和优化训练作业，如通过重建作业时微调超参得到的新作业，可在实验中有序归纳与比较作业结果，从而有效地寻找最佳训练配置。 ● “选择已有实验”：将训练作业纳入已有的实验中。需要选择“实验名称”。 ● “创建新实验”：将训练作业纳入新建的实验中。需要配置“新实验名称”和“新实验描述（可选）”。 不勾选实验设置，表示训练作业不在实验中进行统一管理。

参数名称	说明
训练应用程序	创建调试作业时，SSH远程开发和JupyterLab至少启用一个。 ● JupyterLab：通过云化版本的JupyterLab，在云上调试和运行代码，无需关注安装配置，即开即用。 ● SSH远程开发：支持通过本地IDE（例如VSCode）远程连接到调试训练作业中，对代码进行调试及运行。
密钥对	启用SSH远程开发时，必须配置密钥对。 可以选择已有密钥对，也可以单击“立即创建”新建密钥对，具体操作请参见创建密钥对。 **说明** 创建好的密钥对，请下载并妥善保管，使用本地IDE远程连接到调试训练作业时需要使用密钥对进行鉴权认证。

步骤三：环境配置

创建训练作业时，需要配置算法来源、启动方式、算法对应的引擎及版本、代码目录等信息。调试作业模式下只支持使用自定义镜像创建调试训练作业。

表 4-2 调试作业环境配置

参数名称	说明
创建方式	固定选择“自定义算法”。
启动方式	固定选择“自定义”。
镜像	必填，填写容器镜像的地址。 容器镜像地址的填写支持如下方式。 ● 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 ● 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息，系统会自动拼接域名地址。
代码来源	选择代码来源。默认是“对象存储服务OBS”。 ● “OBS对象存储”：如果训练代码存放在OBS中，则选择“对象存储服务OBS”。

参数名称	说明
代码目录	选择训练代码文件所在的OBS目录。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。 **说明** 如果您的OBS桶中有需要特别保护的敏感数据，请自行对数据加密后再存 放到OBS桶。

启动命令	必填，镜像的启动命令。
	运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。
	● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。
	`python ${MA_JOB_DIR}/demo-code/train.py`
	● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。
	说明
	为保证数据安全，请勿输入敏感信息，例如明文密码。

运行用户ID	容器运行时的用户ID，该参数为选填参数，建议使用默认值1000。 如果需要指定uid，则uid数值需要在规定范围内，专属资源池的uid范围是0~65535。同时，自定义镜像必须存在“/home/ma-user”目录，且“/home/ma-user”目录的属主为指定uid，属组为指定uid对应的gid。 如果运行用户ID配置为0，则训练容器中的运行用户是root。

参数名称	说明
本地代码目录	仅当“代码来源”选择“对象存储服务OBS”时才显示该参数。可选参数。 用于指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。 本地代码目录不支持在/home/ma-user/modelarts/*，/home/ma-user/modelarts-dev/*，/home/ma-user/infer/*目录下，也不支持配置成/home/ma-user。 单击页面右上角的“运行环境预览”，可以查看训练作业的工作目录。
容器运行目录	指定训练容器的本地目录，训练时系统会自动cd到此目录下执行启动文件。 容器运行目录是训练作业运行过程中执行启动命令的本地目录，可用于存储生成的一些临时文件。为确保该目录存在，容器运行目录必须是启动文件本地目录的父目录。

步骤四：训练配置

训练过程中可以从OBS桶或者数据集中获取输入数据进行模型训练，训练输出的结果也支持存储至OBS桶中。创建训练作业时可以参考**表4-3**配置输入、输出、超参、环境变量等参数。

表 4-3 训练配置

参数名称	说明
输入	单击“添加”，配置训练作业输入信息。 ● “参数名称” 算法代码需要通过“输入”的“参数名称”去读取训练的输入数据。 建议设置为“data_url”。训练输入参数要与所选算法的“输入”参数匹配。 ● “输入来源”：训练输入支持来源OBS存储和数据集。 – 单击“输入来源”参数右侧的 🗀，从OBS桶中选择训练输入数据的存储位置。文件总大小要小于或等于10GB，文件数要小于或等于1000个，单个文件大小要小于或等于1GB。 – 单击“数据集”，在ModelArts数据集列表中勾选目标数据集并选择对应的版本。 训练启动时，系统将自动下载输入路径中的数据到训练运行容器。 ● “容器内访问路径”：数据输入通道映射的容器本地路径。 – “输入”的本地路径不支持修改，此路径由系统自动设置。 – 值为参数名称、本地代码路径、索引和输入输出类型拼接而成，规则为：本地代码路径 + 输入输出（input / output）+ 参数名称 + ‘_’ + 索引。 **图 4-2** 示例 ● “获取方式” 以参数名称为“data_path”的训练输入为例，说明获取方式的作用。 – 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 import argparse parser = argparse.ArgumentParser() parser.add_argument('--data_path') args, unknown = parser.parse_known_args() data_path = args.data_path – 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 import os data_path = os.getenv("data_path", "")

版权所有 © 华为云计算技术有限公司

参数名称	说明
输出	单击“添加”，配置训练作业输出信息。 ● “参数名称” 算法代码需要通过“输出”的“参数名称”去读取训练的输出目录。 建议设置为“train_url”。训练输出参数要与所选算法的“输出”参数匹配。 ● “输出路径” 单击“输出路径”参数右侧的 🗀，从OBS桶中选择训练输出数据的存储位置。文件总大小必须小于或等于1GB，文件数必须小于或等于128个，单个文件大小必须小于或等于128MB。 训练过程中，系统将自动从训练容器的本地代码目录下同步文件到数据存储位置。 输出数据的存储位置仅支持OBS存储。为避免和训练输入数据存储冲突，建议选择一个空目录用作输出数据存储位置。 ● “获取方式” 以参数名称为“train_url”的训练输出为例，说明获取方式的作用。 – 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 import argparse parser = argparse.ArgumentParser() parser.add_argument('--train_url') args, unknown = parser.parse_known_args() train_url = args.train_url – 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 import os train_url = os.getenv("train_url", "") ● “容器内访问路径”：数据输入通道映射的容器本地路径。 – “输出”的本地路径不支持修改，此路径由系统自动设置。 – 值为参数名称、本地代码路径、索引和输入输出类型拼接而成，规则为：本地代码路径 + 输入输出（input / output）+ 参数名称 + ‘_’ + 索引。 ● “预下载至容器目录” 选择是否将输出目录下的文件预下载至训练容器中。 – 否：表示启动训练作业时不会将输出数据存储位置中的文件下载到训练容器的本地代码目录中。 – 是：表示系统会在启动训练作业时自动将输出数据存储位置中的所有文件下载到训练容器的本地代码目录中。下载时间会随着文件变大而变长，为了防止训练时间过长，请及时清理无用文件。增量模型训练或设置断点续训练场景下，必须选择“是”。

参数名称	说明
超参	超参用于训练调优。此参数由选择的算法决定，如果在算法中已经定义了超参，则此处会显示算法中所有的超参。 超参支持修改和删除，状态取决于算法中的超参“约束”设置，详情请参见表5-6。 ● 单击“添加”，添加超参信息，总数不超过100条。 ● 单击“本地上传”可以本地批量导入超参，需要按模板填写超参且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
环境变量	根据业务需求增加环境变量。训练容器中预置的环境变量请参见管理训练容器环境变量。 ● 单击“添加”，添加环境变量信息，总数不超过100条。 ● 单击“本地上传”可以本地批量导入环境变量，需要按模板填写环境变量且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。

步骤五：资源配置

参考表4-4配置训练作业的训练资源。

表 4-4 资源配置

参数名称	说明
资源池类型	仅支持“专属资源池”。 专属资源池需要提前单独创建，不与其他租户共享，具体创建操作请参见创建资源池。
资源池	仅选择“专属资源池”后出现此参数，在“资源池”中单击“选择资源池”，在右侧弹窗中选择要使用的专属资源池或者逻辑子池后，单击“确定”。 选择界面支持查看当前专属资源池的名称、节点池规格、可用节点数/最大节点数、可用NPU/GPU设备数、可用CPU（vCPUs）、可用内存（GiB）以及资源碎片。单击资源碎片列的“查看”可以查看碎片详情，确认资源池是否满足训练需求。 选择完成后，此处会展示所选资源池信息，也可单击“重新选择”再次选择资源池。

参数名称	说明
规格类型	当选择“专属资源池”时，会显示此参数。支持以下2种规格类型： ● “预置规格”：选择该专属资源池中预置的“实例规格”。请确保已选择的规格有足够的磁盘空间下载输入文件。 ● “自定义规格”：训练作业支持基于专属资源池的规格自定义资源规格，可自行选择CPU（核）、Memory（GB）、Ascend（卡）和实例数，进而提升资源池的利用率。自定义规格只能小于或等于专属资源池的“节点规格”。 ● CPU规格只支持自定义核数和内存，GPU和Ascend规格支持自定义核数、内存和卡数。 **图 4-3** 实例规格
实例规格	资源池类型为“专属资源池”时，需要先选定“资源池”且规格类型选择“预置规格”后方可选择，资源池类型为“公共资源池”时直接单击，然后在右侧弹窗中选择实例规格后，单击“确定”。 ● 最上方显示了CPU、NPU和GPU规格类型，下方表格可查看规格名称、内存、规格参考价等详情。 ● 规格名称中体现了节点规格的卡数、卡型号、CPU/GPU以及内存大小等信息。 ● 如果部分资源类型不可见或不可选，表示不支持。 ● 当配置了“输入”参数时，在选择资源池“实例规格”时可以单击下方的“获取实际可用规格预估”，检查输入数据的大小是否超出数据盘的容量限制，避免训练过程中出现内存不足的情况。 **说明** ● 实例规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。 ● 资源池类型为“公共资源池”时：使用自定义镜像（创建方式为自定义算法，启动方式为自定义）创建作业时，请确保您的“镜像”类型、“资源类型”和“实例规格”类型保持一致（例如：均为GPU类型）。否则会导致训练作业失败。

参数名称	说明
实例数	必填，根据需要选择实例数的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数 > 1”时，创建的是分布式训练作业，更多分布式训练配置请参见分布式训练功能介绍。 **如果用户开启了对应集群的高可用节点特性，资源池可供训练作业调度的总节点数为实际可用节点数与高可用节点数之和。高可用节点由系统预留，用户作业无法直接调度，仅在节点发生异常时用于故障恢复。** 在创建分布式训练作业前，建议提前安装所有的pip依赖（请参见在镜像中安装pip依赖），因为当节点数大于10时，系统会自动删除pip源配置，当训练过程中涉及pip install操作时可能会训练失败。
亲和调度策略	仅专属资源池支持配置。为训练作业配置超节点和节点亲和性。勾选后开启。 开启后，支持精细控制 Pod 的部署策略：严格部署（强亲和）、尽量部署（弱亲和）、禁止部署（强反亲和）、避免部署（弱反亲和）。 **亲和类型：** 节点亲和：要求训练作业的所有实例必须或者尽量调度到选择的节点上。 节点反亲和：要求训练作业的所有实例必须或者尽量不调度到选择的节点上。 **强度：**亲和强弱。 弱：尽量将 Pod 调度到指定节点，不保证成功。 强：严格将 Pod 调度到指定节点，否则不执行调度。 **超节点亲和方式：**仅超节点资源池支持。 超节点层级，当前仅支持训练作业的所有实例在1个亲和组内（即超节点亲和组实例数=实例数）的场景，适用于希望业务流量不跨超节点的训练。 随机子节点：系统随机调度至目标超节点内的子节点。 指定子节点：按指定节点调度。 **选择超节点：**选择需要配置的超节点。仅超节点资源池支持。 **选择节点：**选择需要配置的节点。

参数名称	说明
存储挂载	资源规格选择“专属资源池”时，支持添加多种存储挂载方式，以提高数据的访问效率。 **图 4-4** 存储挂载 ● “添加拓展存储SFS Turbo配置”：当ModelArts和SFS Turbo间网络直通时，训练作业支持挂载多个SFS Turbo存放训练数据。文件系统支持重复挂载，但是挂载路径不可重复。最多可以挂载5个盘。配置ModelArts和SFS Turbo间网络直通，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。 **图 4-5** SFS Turbo – “文件系统”：选择一个SFS Turbo。 – “云上挂载路径”：输入SFS Turbo对应在训练容器内的云上挂载路径。不能为“/”目录，不能为“/cache”、“/home/ma-user/modelarts”等系统已经默认挂载的路径。 – “存储位置”：选择SFS Turbo的存储位置。如果用户配置了文件夹控制权限，请选择存储位置；如果用户未配置文件夹控制权限，可以保持默认值“/”或者自定义位置。 – “挂载方式”：显示挂载SFS Turbo的权限。根据SFS Turbo存储位置的权限显示“读写”或“只读”，如果用户未配置文件夹控制权限，则该参数不可见。如果需要设置SFS Turbo的文件夹权限，请参考**权限管理**文档配置。 – “挂载参数”：支持配置SFS挂载参数实现训练加速优化，具体参数说明请参见**设置极速文件存储挂载参数**。不设置时，默认配置如下参数： mountOptions: - vers=3 - timeo=600 - nolock - hard

版权所有 © 华为云计算技术有限公司

参数名称	说明
	说明 当用户配置拓展存储SFS Turbo时，前端页面会自动从**SFS Turbo**服务获取最新的存储容量和配置信息，以确保数据有效性和准确性。 1. 查询文件系统详细信息。 2. 获取文件系统列表。 ● “添加拓展存储SFS 3.0容量型配置”：训练作业支持挂载一个通用文件系统SFS 3.0用于存放训练数据。需要配置如下参数。 **图 4-6** SFS 3.0 容量型 – “文件系统”：输入通用文件系统名称。需要和弹性文件服务SFS里面的名称一致，否则会挂载失败。 – “云上挂载路径”：输入对应在训练容器内的云上挂载路径。 **说明** “添加拓展存储SFS 3.0容量型配置”为受限使用阶段，需要提工单申请开通。 ● “添加拓展存储PFS配置”：训练作业支持挂载一个OBS的并行文件系统用于存放训练数据。需要配置如下参数。 **图 4-7** PFS – “存储位置”：选择一个并行文件系统。 – “云上挂载路径”：输入对应在训练容器内的云上挂载路径。 **说明** “添加拓展存储PFS配置”为受限使用阶段，需要提工单申请开通。

参数名称	说明
超节点亲和组实例数	● “资源池”选择“专属资源池”，且“节点规格”为Snt9b23系列情况下，支持配置超节点亲和组实例数。 ● 当选择超节点资源池时，支持配置超节点亲和组实例数。如果配置超节点亲和组实例数为n，则每n个Pod会被调度到同一个超节点内，从而实现亲和性作业调度。在分布式训练中亲和性作业调度可以保证计算资源的同构性。 ● 如果配置了“超节点亲和组实例数”，则“实例数”必须设置为超节点亲和组实例数的整数倍，否则无法创建训练作业。例如，超节点亲和组实例数为n，则实例数为n或2n。 ● 超节点亲和组实例配置更多内容请参见**超节点亲和组实例数配置**章节。

步骤六：高可用配置

创建训练作业时，支持设置训练作业自动重启。

表 4-5 高可用配置

参数名称	说明
自动重启	设置训练作业是否设置自动重启。 ● 不勾选（默认）：表示不启用自动重启，训练异常则直接停止作业。 ● 勾选：表示当由于环境问题、作业卡死等原因导致训练作业异常时，系统将自动检测故障问题并根据相应策略进行处理，以提高训练成功率。训练作业恢复策略已支持进程级、容器级、作业级自动重启，无需配置，自动匹配与升级恢复策略。 为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见**设置断点续训练**。 当训练过程中触发了自动重启，则系统会记录重启信息，在训练作业详情页可以查看故障恢复详情，具体请参见**训练作业容错检查**。
最大重启次数	当勾选“自动重启”时可配置。 当训练作业的自动重启次数达到最大重启次数，且训练作业依旧异常，则直接停止作业。 ● 缺省值：3 ● 取值范围：1~128 提交创建训练作业后不支持修改重启次数，请合理设置次数。
无条件自动重启	当勾选“自动重启”时可配置。勾选无条件自动重启后，只要系统检测到训练异常，就无条件重启训练作业。为了避免无效重启浪费算力资源，系统最多只支持连续无条件重启3次。

参数名称	说明
作业卡死重启	当勾选“自动重启”时可配置。系统支持自动监控作业进程的状态和资源利用率来判定作业是否卡死，开启此开关后，支持将标记为卡死的作业进行进程级自动重启，以提高资源使用率。 CPU规格不支持作业卡死重启。 因系统无法核实代码逻辑且检测存在周期性，卡死检测存在一定的误报概率，开启开关即表示接受误报率。为了避免无效重启浪费算力资源，系统最多只支持连续作业卡死重启3次。
算子重执行	默认不展示，需要符合一定的条件才会展示“算子重执行”开关。 开启算子重执行，可以解决当超节点网络链路故障时导致的训练作业通信异常的问题。 开启对训练任务的性能有较大影响。 详细介绍和约束条件请参见**配置算子重执行**章节。
性能监控与诊断	当训练作业选择自定义镜像时，可勾选该项并配置弹性文件服务 SFS Turbo，开启后，针对训练中耗时超预期、资源利用率失衡等性能劣化问题，可实时观测 step 耗时曲线；当性能劣化超限时，支持自动或手动采集 profiling 数据，采集完成后通过可视化工具 MindStudio Insight 提供可视化分析结果。

步骤七：调度配置

创建训练作业时，支持配置作业调度策略，例如提高作业调度优先级、允许被抢占、自动停止等，用于提高作业调度效率。

表 4-6 调度配置

参数名称	说明
作业优先级	• 使用专属资源池创建训练作业时，支持使用和修改训练作业的调度优先级。使用公共资源池时，不支持此参数配置。 • 作业默认按照优先级从高到低依次处理，对于相同优先级的任务，则根据提交时间的先后顺序进行调度，最终结合实际的资源情况决定调度顺序(保证在资源充足、相同优先级下，先提交的任务先调度)。 • 修改选择数字后可修改作业进入队列的优先级。优先级可以设置数字1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。 • 设置优先级为“3”时，还需要配置用户权限才会生效。用户权限设置请参见“**设置作业为高优先级权限**”。 • 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。

参数名称	说明
允许被抢占	● 使用专属资源池创建训练作业时，允许设置训练作业被抢占。使用公共资源池时，不支持此参数配置。 ● 开启后，当资源池剩余资源不足时，允许被抢占的作业可能会被终止并重新排队。为避免训练进度丢失，请在开启此功能前完成断点续训的设置，操作指导请参考设置断点续训练。
自动停止	选择是否启用训练作业自动停止。 ● 不勾选（默认）：表示训练作业将一直运行直至训练完成。 ● 勾选：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。

步骤八：更多配置

表 4-7 高级配置

参数名称	说明
永久保存日志	当实例规格选择Ascend资源时，默认开启“永久保存日志”。 当实例规格选择CPU或GPU资源时，支持选择是否打开“永久保存日志”开关。 ● 勾选（默认）：表示永久保存日志，此时必须配置“日志路径”，平台会将训练日志永久保存至指定的OBS路径。 ● 不勾选：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
日志路径	当开启“永久保存日志”时，必须配置“日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。
作业可见范围	分为“工作空间内可见”和“仅创建者可见”。 ● “工作空间内可见”：创建的训练作业在当前工作空间内，默认其他子用户均可看到该训练作业。 ● “仅创建者可见”：默认创建者可见，其他人希望查看该作业需要额外申请“modelarts:trainJob:listAll”权限（查看训练作业列表（包含仅创建者可见的作业）权限）。

参数名称	说明
事件通知	选择是否启用训练作业的事件通知。 ● 不勾选（默认）：表示不启用消息通知服务。 ● 勾选：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题，并为主题添加订阅。当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”“”、“作业终止”“”、“作业疑似卡死”“”、“作业重启”、“作业抢占”等。 **说明** ● 使用消息通知服务会产生相关服务费用，详细信息请参见**消息通知服务计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
生成实例间SSH免密互信文件	选择是否生成实例间SSH免密互信文件。 ● 不勾选（默认）：表示不生成实例间SSH免密互信文件。 ● 勾选：表示生成实例间SSH免密互信文件。在使用基于MPI和Horovod框架的自定义镜像进行分布式训练时，为确保实例间的通信顺畅，必须配置实例间SSH免密互信，确保分布式训练任务顺利进行。 同时还需配置“SSH免密互信文件目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。更多操作指导请参见**自定义镜像训练作业配置实例间SSH免密互信**。
标签	当需要使用同一标签标识多种云资源时，即所有服务均可在标签输入框下拉选择同一标签，则建议在TMS中创建预定义标签。标签详细用法请参见**使用TMS标签实现资源分组管理**。 最多支持添加20条标签。

步骤九：提交训练作业并调试

1. 当创建训练作业的参数配置完成后，单击“提交”，在信息确认页面单击“确定”，提交调试训练作业。

   在训练作业列表中，刚创建的训练作业状态为“等待中”。

2. 在训练作业列表中，当训练作业的状态变为“运行中”时，单击训练作业名称进入训练详情页，在训练详情页单击JupyterLab地址，弹出JupyterLab页面，即可进行训练代码调试。

   JupyterLab页面左侧会显示训练代码文件，JupyterLab的使用请参见**JupyterLab简介及常用操作**。

   当创建的调试训练作业开启了SSH远程开发时，在训练作业详情页获取“SSH远程开发”参数后的“密钥对”和SSH地址信息后，可以使用本地IDE远程连接到调试训练作业中进行代码调试。

4.2 使用 PyCharm ToolKit 创建并调试训练作业

由于AI开发者会使用PyCharm工具开发算法或模型，为方便快速将本地代码提交到ModelArts的训练环境，ModelArts提供了一个PyCharm插件工具PyCharm ToolKit，协助用户完成SSH远程连接Notebook、代码上传、提交训练作业、将训练日志获取到本地展示等，用户只需要专注于本地的代码开发即可。

说明

当前版本插件已停止维护，不建议使用。推荐直接通过控制台或者API调用此功能。

本章节介绍如何使用PyCharm ToolKit插件创建训练作业并调试。

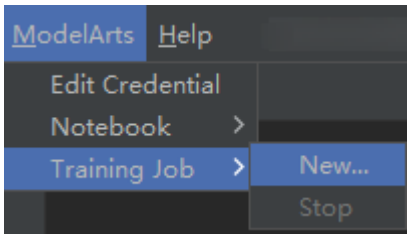
前提条件

- **Step1 下载并安装PyCharm ToolKit**。
- 在本地PyCharm中已有训练代码工程。
- 已在OBS中创建桶和文件夹，用于存放数据集和训练输出模型。 例如：创建命名为“test-modelarts2”的桶，创建文件夹“dataset-mnist”和“mnist-output”。训练作业使用的数据已上传至OBS，且OBS与ModelArts在同一区域。

配置训练作业参数

1. 在PyCharm中，打开训练代码工程和训练启动文件，然后在菜单栏中选择“ModelArts > Training Job > New...”。

   图 4-8 选择作业配置

2. 在弹出的对话框中，设置训练作业相关参数，详细参数说明请参见**表4-8**。

   表 4-8 训练作业配置参数说明

参数	说明
Job Name	训练作业的名称。 系统会自动生成一个名称，您可以根据业务需求重新命名，命名规则如下： ● 支持1~64位字符。 ● 并包含大小写字母、数字、中划线（-）或下划线（_）。
Job Description	训练作业的简要描述。

参数	说明
Algorithm Source	训练算法来源，分为“常用框架”和“自定义镜像”两种，二者选一项即可。 常用框架指使用ModelArts训练管理中支持的常用AI引擎，当前支持的引擎列表请参见**ModelArts支持的预置镜像列表**。 如果您使用的AI引擎为支持列表之外的，建议使用自定义镜像的方式创建训练作业。
AI Engine	选择代码使用的AI引擎及其版本。支持的AI引擎与ModelArts管理控制台里**ModelArts支持的预置镜像列表**一致。
Boot File Path	训练启动文件，所选启动文件必须是当前PyCharm训练工程中的文件。当“Algorithm source”选“Frequently-used”时，显示此参数。
Code Directory	训练代码目录，系统会自动填写为训练启动文件所在的目录，用户可根据需要修改，所选目录必须是当前工程中的目录且包含启动文件。 当算法来源为自定义镜像，训练代码已预置在镜像中时，该参数可以为空。
Image Path(optional)	SWR镜像的URL地址，例如swr.cn-north-4.myhuaweicloud.com/image-org/image-name:version。关于自定义镜像的说明，请参见**自定义镜像介绍**。
Boot Command	启动本次训练作业的运行命令。例如“bash /home/work/run_train.sh python {python启动文件及参数}”。当“Algorithm source”选“Custom”时，显示此参数。 当用户输入的命令中不包含“--data_url”和“--train_url”参数时，工具在提交训练作业时会在命令后面自动添加这两个参数，分别对应存储训练数据的OBS路径和存放训练输出的OBS路径。
Data OBS Path	设置为存储训练数据的OBS路径，例如“/test-modelarts2/mnist/dataset-mnist/”，其中“test-modelarts2”为桶名称。
Training OBS Path	设置OBS路径，该路径下会自动创建用于存放训练输出模型和训练日志的目录。
Running Parameters	运行参数。如果您的代码需要添加一些运行参数，可以在此处添加，多个运行参数使用英文分号隔开， 例如"key1=value1;key2=value2"。此参数也可以不设置，即保持为空。
Specifications	训练使用资源类型。目前支持公共资源池和专属资源池两种类型。 专属资源池规格以“Dedicated Resource Pool”标识。只有购买了专属资源池的用户才会显示专属资源池规格。

参数	说明
Compute Nodes	计算资源节点个数。数量设置为1时，表示单机运行；数量设置大于1时，表示后台的计算模式为分布式。
Available/Total Nodes	当“Specifications”选择专属资源池规格时，显示专属资源池的可用实例数和总实例数，用户选择“Compute Nodes”的个数不要超过可用实例数。

图 4-9 配置训练作业参数（公共资源池）

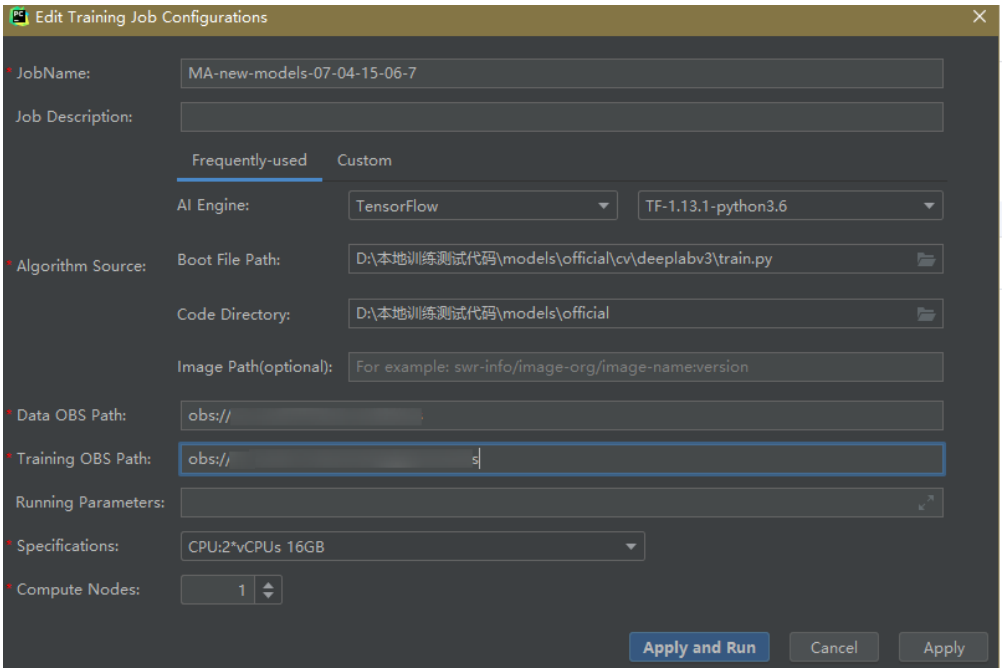
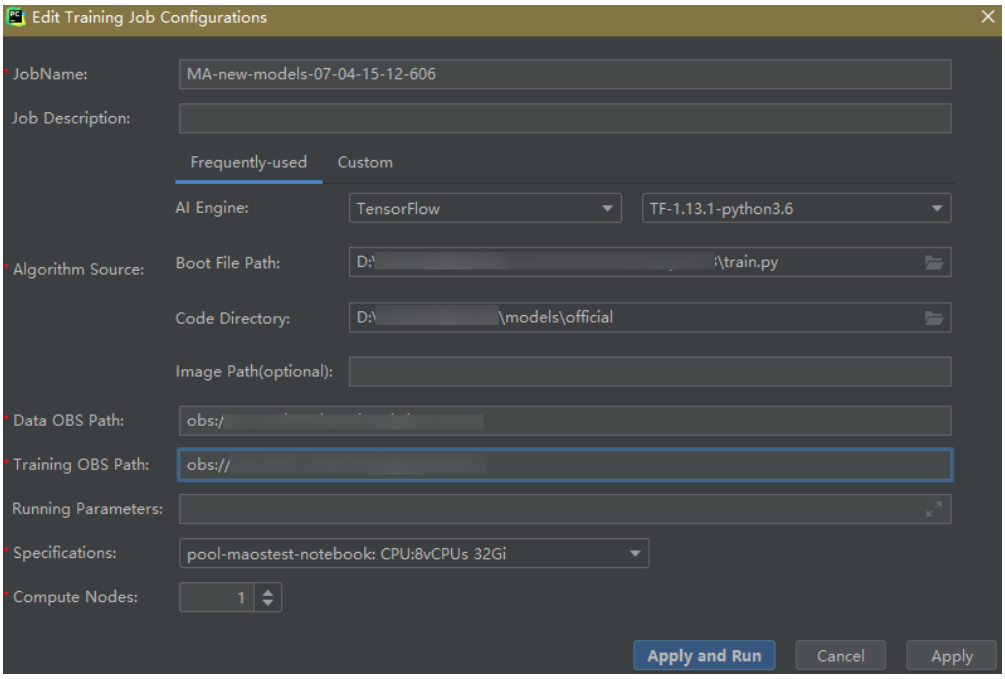
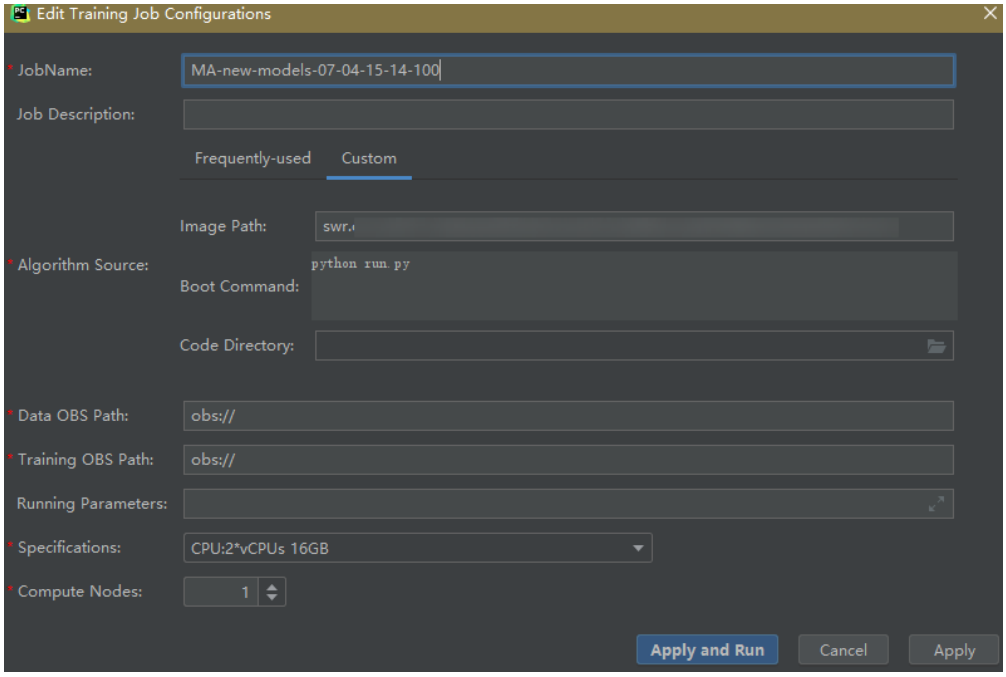


图 4-10 配置训练作业参数（专属资源池）



版权所有 © 华为云计算技术有限公司

图 4-11 配置训练作业参数（自定义镜像）

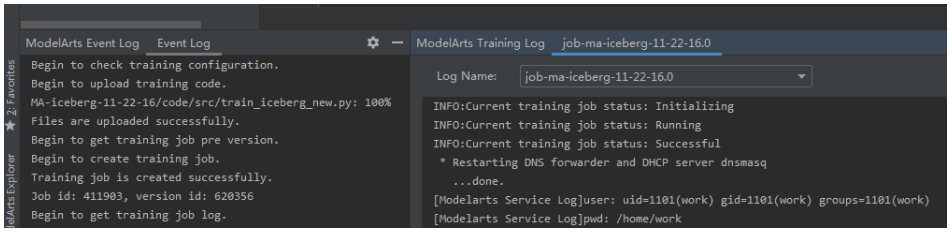


3. 参数填写完成后，单击“Apply and Run”，即自动上传本地代码至云端并启动训练，在工具下方的Training Log区域，会实时展示训练作业运行情况。当训练日志中出现“Current training job status: Successful”类似信息时，表示训练作业运行成功。

说明

- 在单击“Apply and Run”按钮后，系统将自动开始执行训练作业。如果您想停止此作业，可以选择菜单栏中的“ModelArts > Training Job >Stop”停止此作业。
- 如果单击“Apply”，不会直接启动运行，只是保存训练作业的设置，如果需要启动作业，可以单击“Apply and Run”。

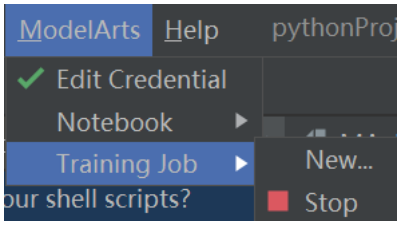
图 4-12 训练日志展示样例



停止作业

当训练作业在运行过程中时，您可以在PyCharm菜单栏中，选择“ModelArts >Training Job > Stop ”停止训练作业。

图 4-13 停止作业



查看训练日志

查看训练日志有2种方式，在OBS查看和在PyCharm ToolKit工具中查看。

- **在OBS查看训练日志**

  提交训练作业时，系统将自动在您配置的OBS Path中，使用作业名称创建一个新的文件夹，用于存储训练输出的模型、日志和代码。

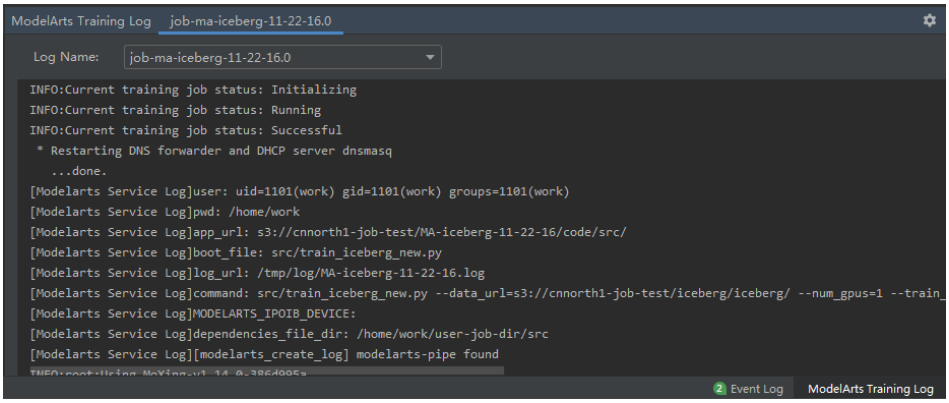
  例如“train-job-01”作业，提交作业时会在“test-modelarts2”桶下创建一个命名为“train-job-01”的文件夹，且此文件夹下分别新建了三个文件夹“output”、“log”、“code”，分别用于存储输出模型、日志和训练代码。“output”文件夹还会根据您的训练作业版本再创建子文件夹，结构示例如下。

```
test-modelarts2
  |---train-job-01
      |---output
      |---log
      |---code
```

- 在PyCharm ToolKit工具中查看训练日志

  在PyCharm ToolKit工具中，单击页面右下角的ModelArts Training Log，展示训练日志。

图 4-14 查看训练日志



4.3 使用 VS Code 创建并调试训练作业

由于AI开发者会使用VS Code工具开发算法或模型，为方便快速将本地代码提交到ModelArts的训练环境、贴近本地开发习惯地编写启动命令，ModelArts提供了一个训练作业场景下的IDE插件ModelArts-HuaweiCloud，用户通过简易的操作，实现在本地IDE中进行训练配置、资源监控、作业管理、代码管理等动作。

本章节介绍如何使用VS Code插件创建训练作业并调试。

说明

当前版本插件已停止维护，不建议使用。推荐直接通过控制台或者API调用此功能。

准备工作

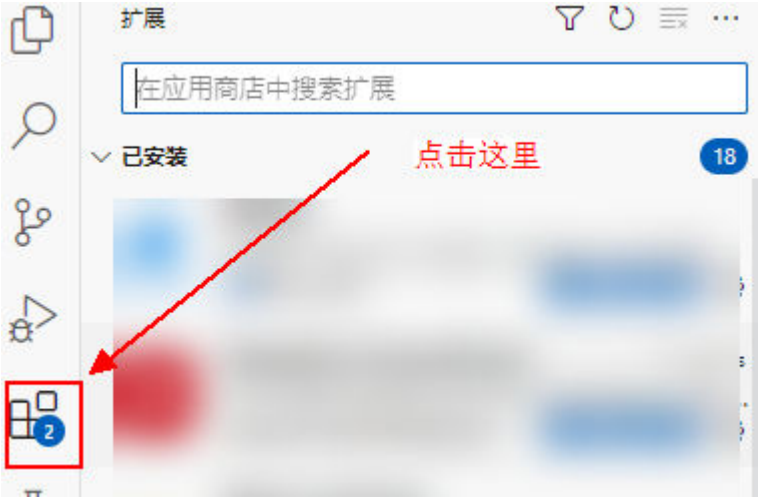
- 创建OBS桶，桶名称示例：“xxx-project”，在OBS桶里创建“data”文件夹。（建议下载OBS Browser+进行相关操作）。
- 上传本地数据至“data”文件夹。复制此OBS路径，此时输入路径应为“obs://xxx-project/data”，（此路径后面标记为输入路径）。

安装 VS Code 插件并登录

本地打开VS Code软件，单击左侧导航栏的 图标，将插件包拖入到扩展区空白处，即可自动安装。

待左侧导航出现 图标，表示插件安装完成。

图 4-15 安装 VS Code 插件



创建训练作业

1. 单击左侧导航的 图标，单击“登录”进入登录页面，输入对应的登录信息。

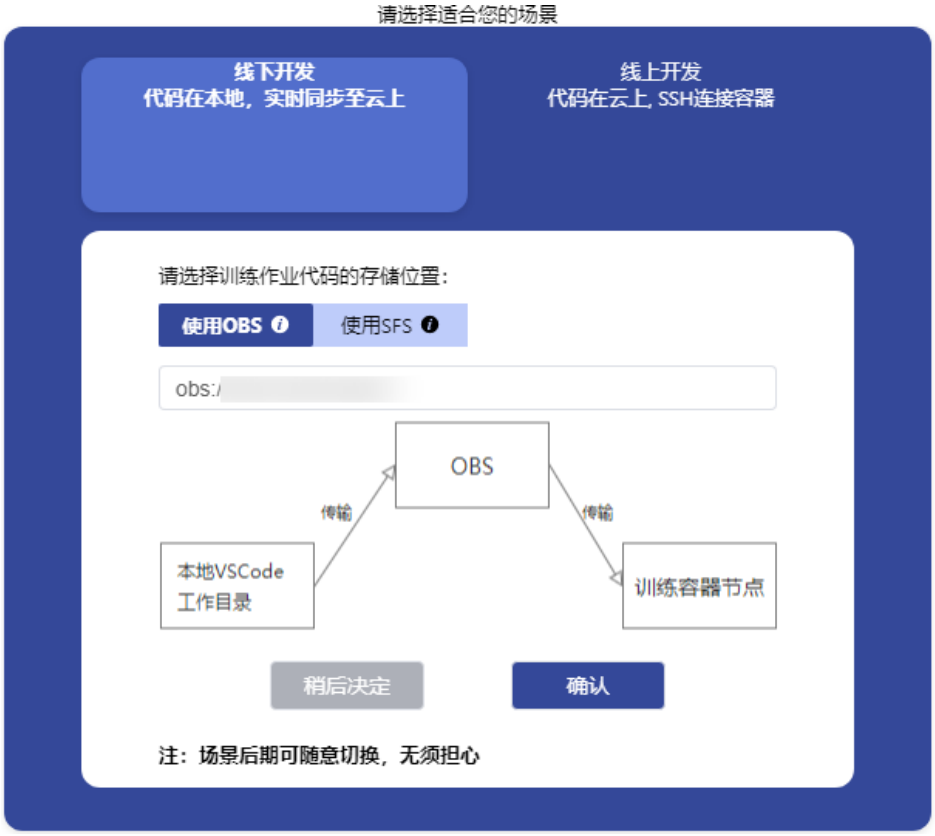
   单击“获取AccessKey”即可跳转到AK/SK获取教程页面。

   图 4-16 登录 VS Code 插件

2. 登录后，VS Code自动同步ModelArts中的训练作业列表，并进入开发场景选择页面，如图4-17所示。选择合适的开发场景，并选择工作目录后，进入训练作业代码存储路径选择界面，单击“使用OBS”或“使用SFS”，填写OBS或SFS路径，单击“确定”。选择此路径后，会自动同步在“高级配置>中转目录”，也可单击“稍后决定”直接跳过。
   - 线下开发：代码在本地，将本地代码传到OBS，然后通过OBS传至云上或直接使用SFS盘存储的代码。
   - 线上开发：代码在云上，通过SSH连接容器。

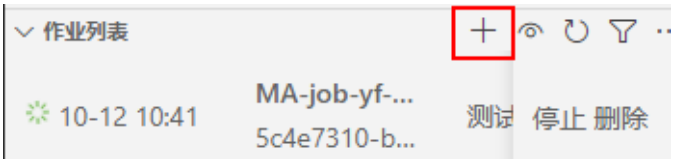
图 4-17 选择开发场景

图 4-18 选择训练作业代码存储位置



3. 鼠标放至作业列表处，单击作业列表旁边的 + 打开创建训练作业页面。

图 4-19 创建作业入口



4. （可选）设置中转目录：中转目录支持OBS路径和SFS盘挂载路径。如果已经在步骤3配置，此处会自动显示中转目录。

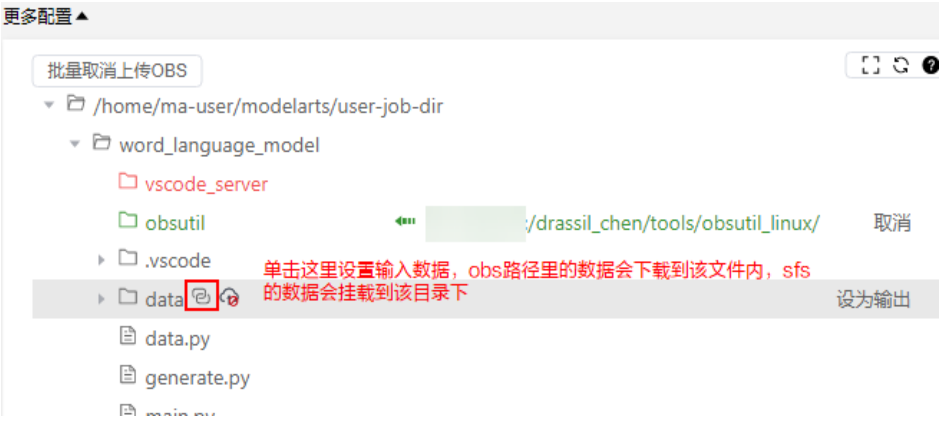
   如果设置的是OBS路径，单击“提交作业”后，插件会自动将当前打开的项目文件整体上传至OBS中转目录上。

图 4-20 设置中转目录

5. 设置输入路径：本地项目对应的输入文件设置为输入路径。

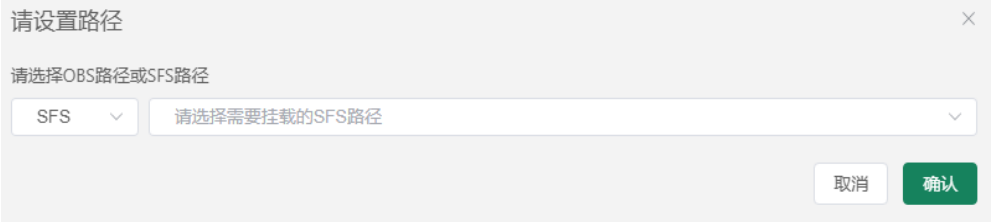
版权所有 © 华为云计算技术有限公司

图 4-21 设置输入路径



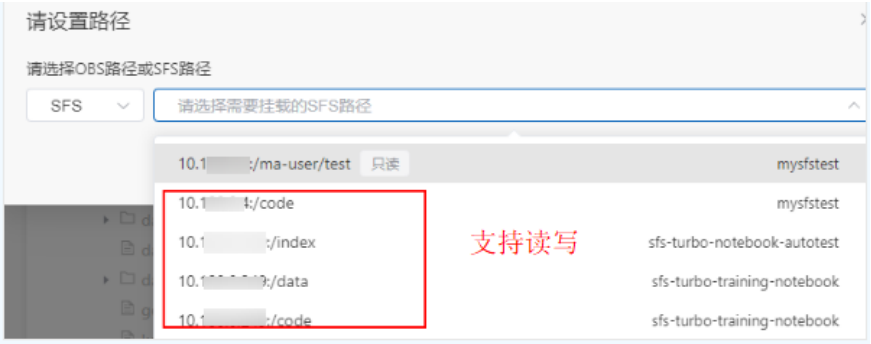
在输入框选择输入数据所在的路径，如图4-23所示。

图 4-22 填写训练输入路径



说明

选择挂载SFS路径时，下拉框只显示有权限的SFS路径，显示“只读”的文件夹只支持只读功能，不显示的读写均支持。可参考权限管理文档修改SFS Turbo权限。



版权所有 © 华为云计算技术有限公司

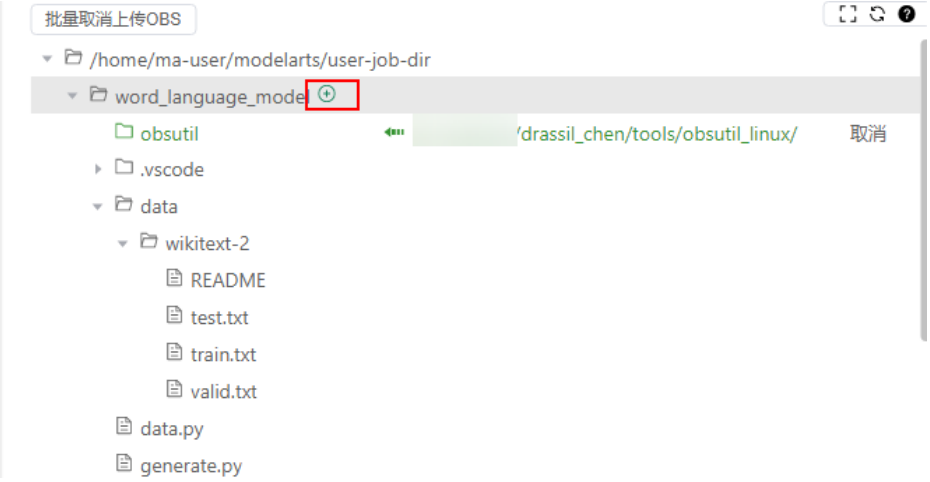
图 4-23 输入数据设置完成界面

6. 设置训练输出路径：新建“output”文件夹设置为输出。（如果本地有output文件夹，无需新建）

说明

只有文件夹下才支持新建文件夹，除项目根目录以外，其他文件夹需展开才能添加文件夹。

图 4-24 新建文件夹



这里用output作为举例，文件夹名需与项目真实输出文件名一致。

版权所有 © 华为云计算技术有限公司

图 4-25 设为输出路径

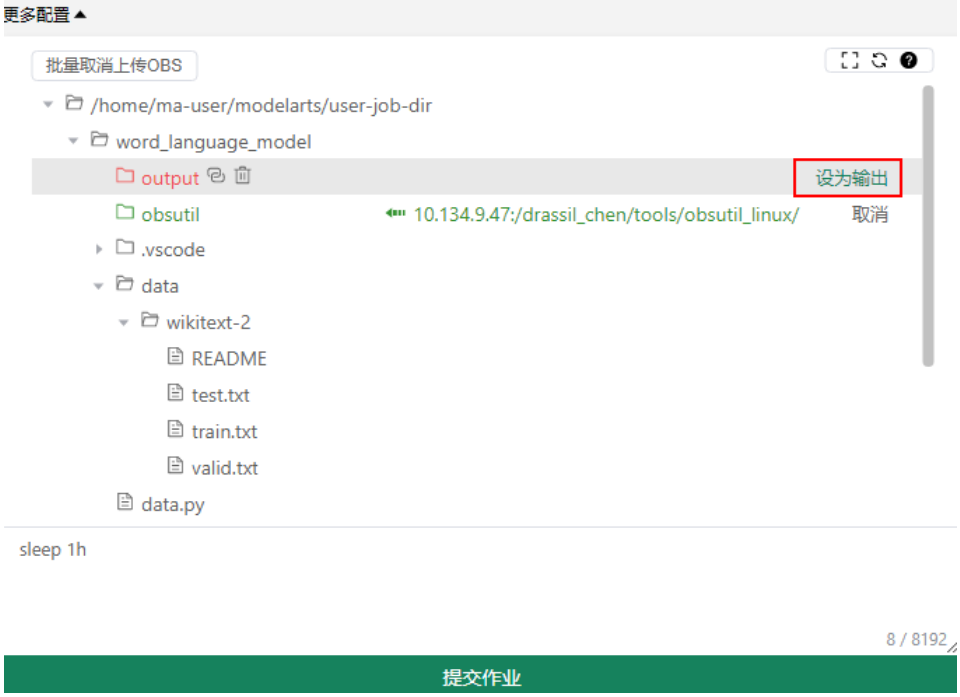


图 4-26 输出设置完成

7. 填写启动命令，提交作业。
8. 在作业列表单击作业名称进入作业详情。

4.4 SSH 远程开发

本节操作介绍在Windows环境中使用PuTTY SSH远程登录云上实例的操作步骤。

版权所有 © 华为云计算技术有限公司

约束限制

使用SSH连接训练作业实例的约束限制如下：

- 本地用户密钥和权限必须匹配。
- 本地用户密钥需要存放在指定目录下。
  - Windows：C:\Users\{{user}}
  - macOS/Linux：~

  说明

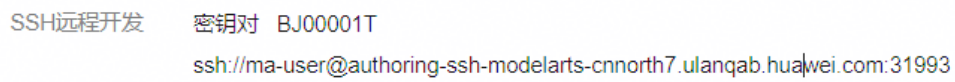
  在macOS和Linux系统中，~ 表示当前用户的主目录。
- 远端镜像中的ma-user不能被锁定。
- 远端的/home/ma-user/.ssh或/root/.ssh目录权限建议设置为750或755。
- 本地或远端的OpenSSH版本不能低于8.0。
- 建议同时建立的连接数不超过10个。

如果遇到SSH连接问题，请参考**本地通过SSH连接的故障排查**进行解决。

前提条件

- 已创建一个训练作业实例，并开启远程SSH开发。该实例状态必须处于“运行中”。
  - 关于如何新建训练作业实例，请参见新版训练创建页面。
  - 关于如何在已有训练作业实例开启远程SSH开发，请参见**修改SSH远程连接配置**。
- 在训练作业实例详情页面获取开发环境访问地址（例如：dev-modelarts-cnnorth4.huaweicloud.com）和端口号。

  图 4-27 训练作业实例详情页面

- 准备好密钥对文件。

  密钥对在用户第一次创建时，自动下载，之后使用相同的密钥时不会再有下载界面（用户一定要保存好），或者每次都使用新的密钥对。

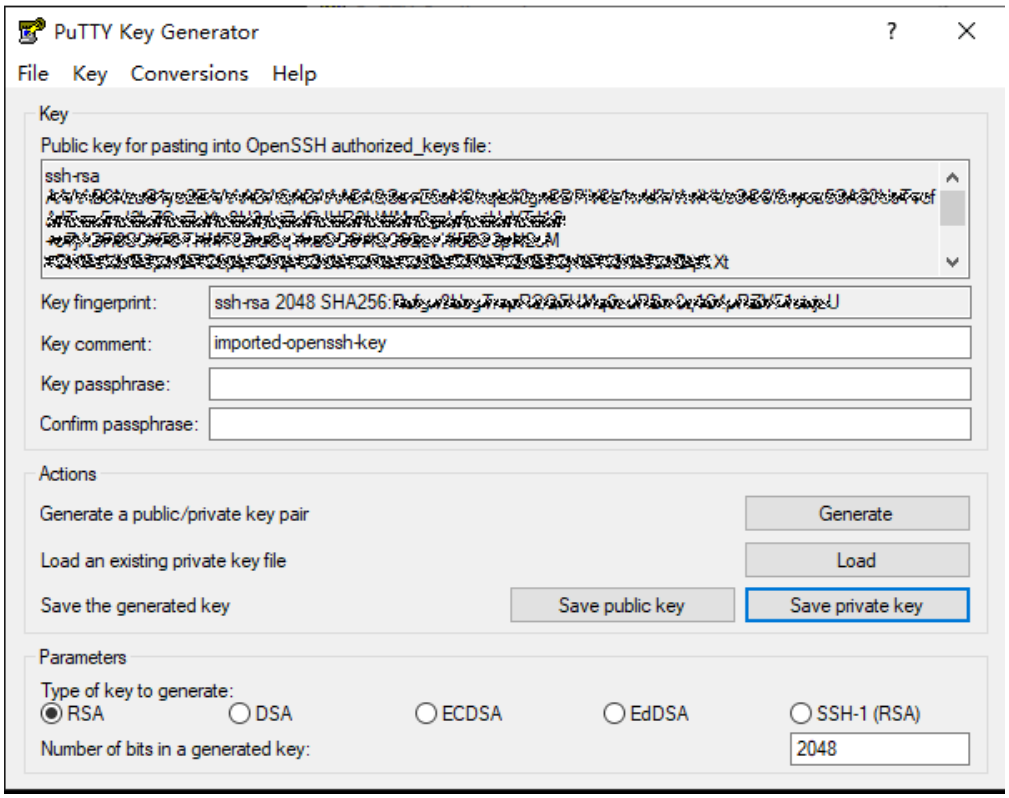
Step1 安装 SSH 工具

下载并安装SSH远程连接工具，以PuTTY为例，**下载链接**。

Step2 使用 puttygen 将密钥对.pem 文件转成.ppk 文件

1. **下载puttygen**，并双击运行puttygen。
2. 单击“Load”，上传.pem密钥（即在创建Notebook实例时创建并保存的密钥对文件）。
3. 单击“Save private key”，保存生成的.ppk文件。.ppk文件的名字可以自定义，例如key.ppk。

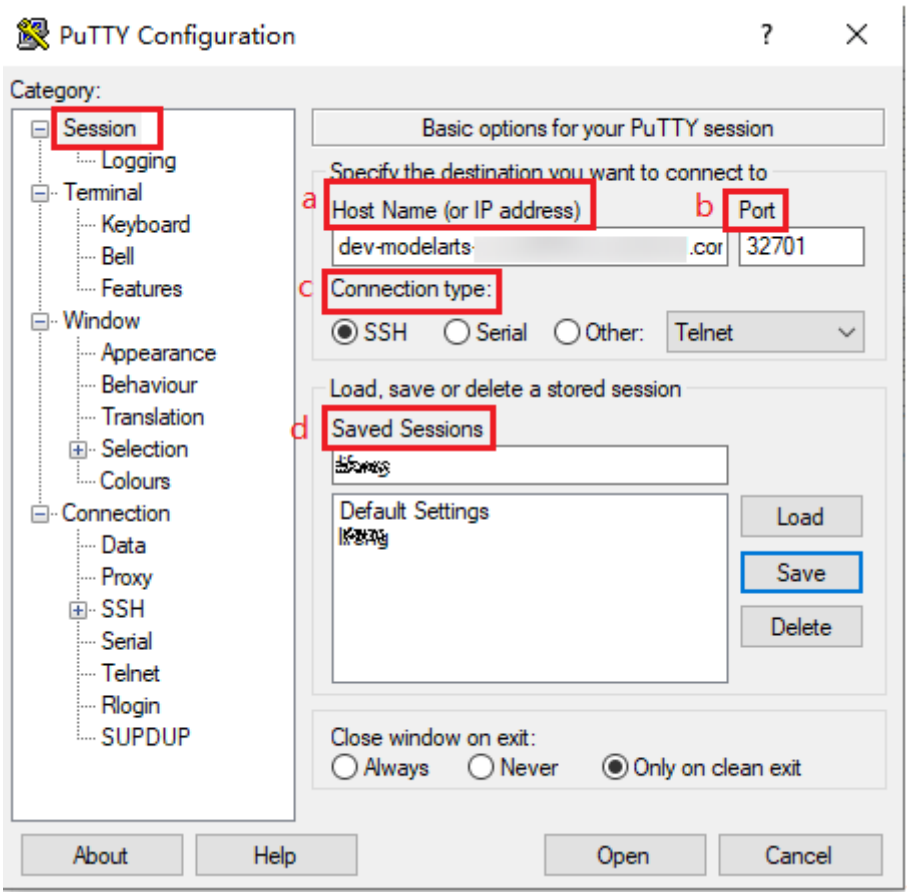
图 4-28 将密钥对.pem 文件转成.ppk 文件



Step3 使用 SSH 工具连接云上训练作业实例

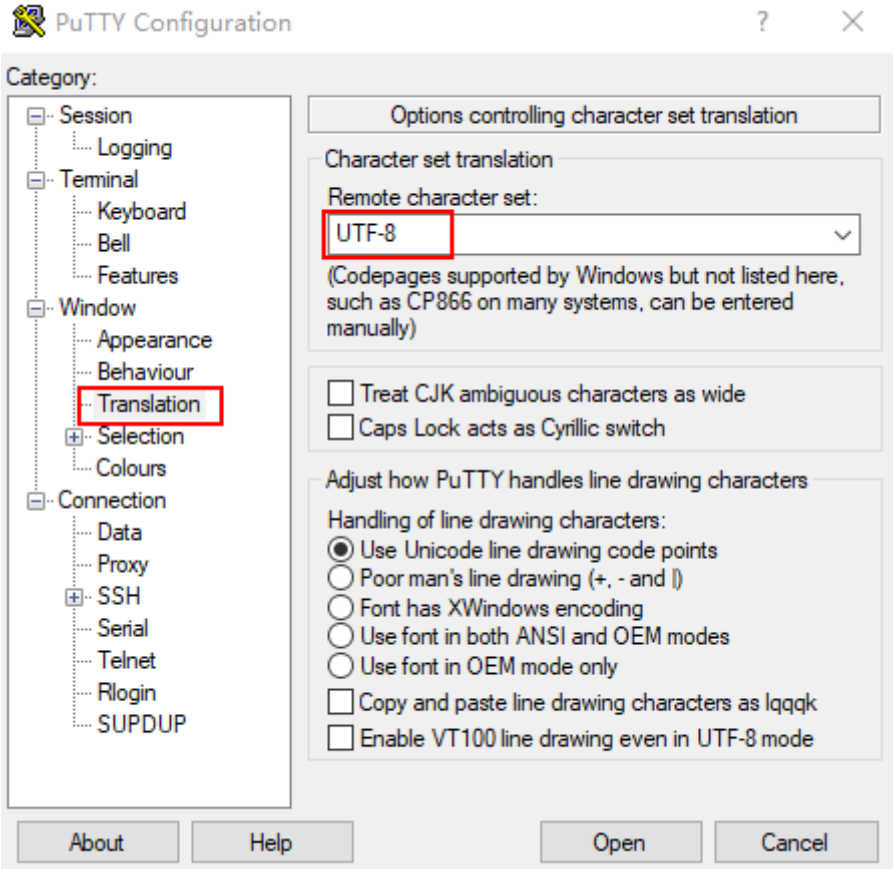
1. 运行PuTTY。
2. 单击“Session”，填写以下参数。
   a. Host Name (or IP address)：云上开发环境训练作业实例的访问地址，即在训练作业实例详情页获取的地址。例如：dev-modelarts-cnnorth4.huaweicloud.com。
   b. Port：云上训练作业实例的端口，即在训练作业实例详情页获取的端口号。例如：32701。
   c. Connection Type：选择SSH。
   d. Saved Sessions：任务名称，在下一次使用PuTTY时就可以单击保存的任务名称，即可打开远程连接。

图 4-29 设置 Session



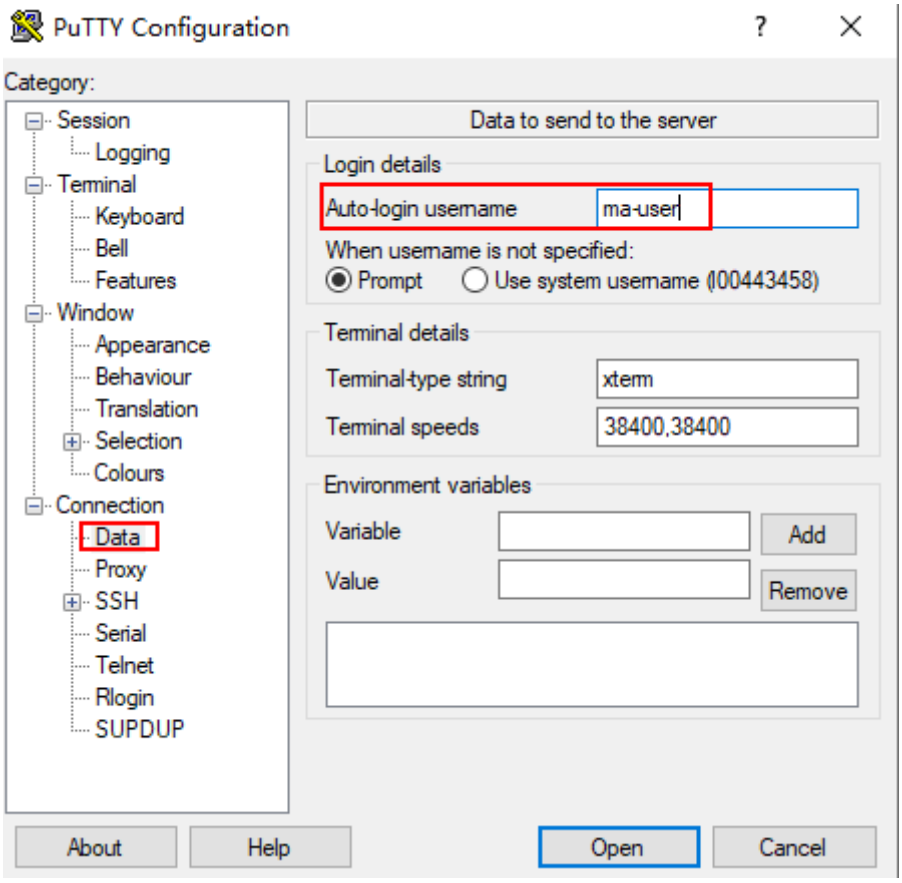
3. 选择“Window > Translation”，在“Remote character set:”中选择“UTF-8”。

图 4-30 设置字符格式



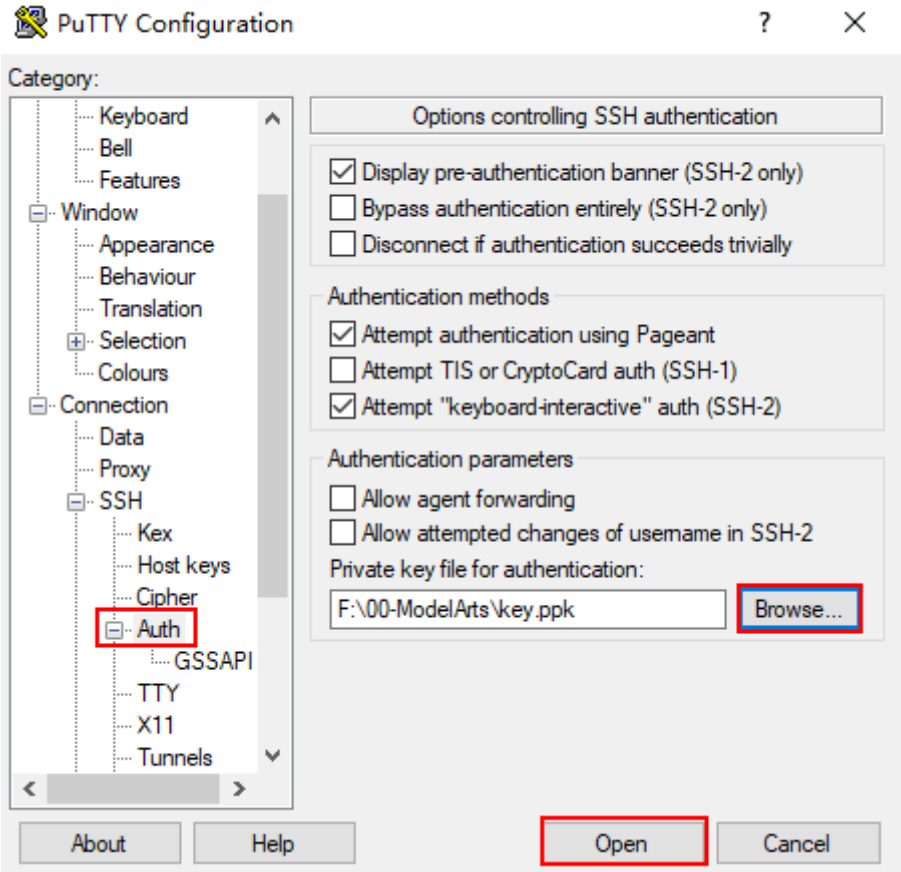
4. 选择“Connection > Data”， 在“Auto-login username” 中填写用户名“ma-user”。

图 4-31 填写用户名



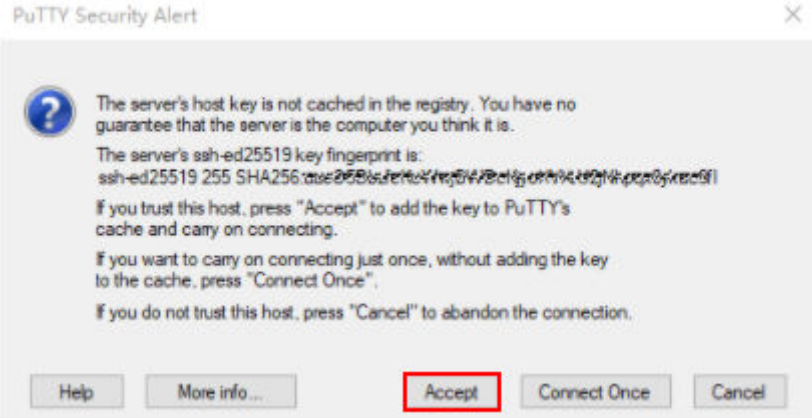
5. 选择“Connection > SSH > Auth”，单击“Browse”，选择“.ppk文件”（由**Step2**密钥对.pem文件生成）。

版权所有 © 华为云计算技术有限公司



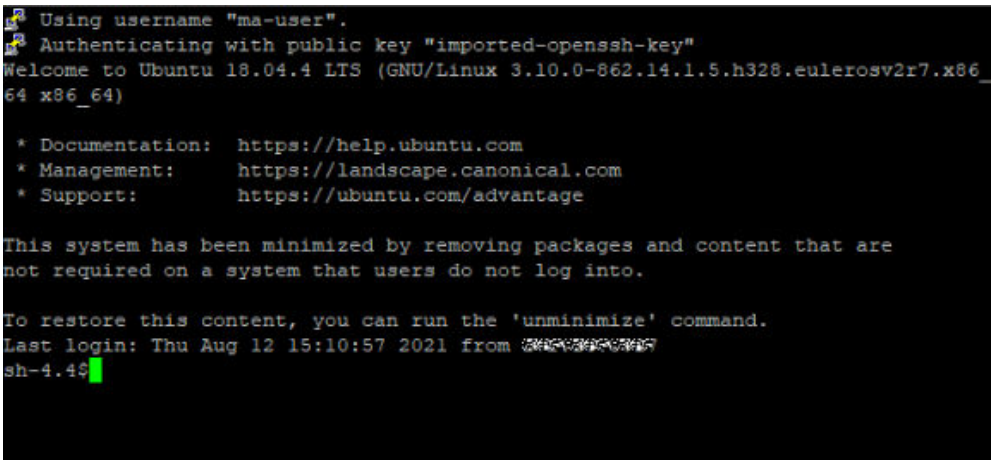
6. 单击“Open”。如果首次登录，PuTTY会显示安全警告对话框，询问是否接受服务器的安全证书。单击“Accept”将证书保存到本地注册表中。

图 4-32 询问是否接受服务器的安全证书



7. 成功连接到云上训练作业实例。

图 4-33 连接到云上训练作业实例



8.（可选）分布式作业跳转其他节点容器

在分布式训练作业（Volcano Job）场景下，一个作业通常包含多个 Task（如 worker-0、worker-1 等）。平台已自动为您配置了实例间的 SSH 免密登录，您可以在已登录的节点中直接跳转到其他节点，方便进行多节点集群调试。

操作步骤

1. 获取目标节点主机名 在当前已登录的容器终端中，通过平台自动注入的环境变量查看所有节点的主机名：

   env | grep VC_%s_HOSTS

   例如：

   env | grep VC_WORKER_HOSTS

   说明：Volcano 会根据 Task 名称注入变量，例如：

   VC_WORKER_HOSTS=ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f-worker-0.ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f,ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f-worker-1.ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f

2.直接执行 SSH 跳转 由于平台已预置免密互信，您无需输入密码，直接通过 ssh 命令即可跳转到作业内的其他节点：

跳转到 worker-1 节点

ssh ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f-worker-1.ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f

3.验证跳转结果 跳转成功后，终端提示符会显示新的主机名。您可以通过执行 hostname 命令确认：

hostname

输出示例：ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f-worker-1.ma-job-40a6df3b-d136-452e-9b8c-530baf9b777f

注意事项

- **用户名一致性**：跳转时默认使用当前登录的 ma-user 用户。

- **环境变量生效范围**：跳转后的新节点同样拥有平台注入的环境变量（如VC_WORKER_HOSTS），方便您在各节点间自由穿梭。
- **自动互信范围**：免密登录仅限于同一训练作业（Job）内的各节点容器之间。

5 创建算法

机器学习从有限的观测数据中学习一般性的规律，并利用这些规律对未知的数据进行预测。为了获取更准确的预测结果，用户需要选择一个合适的算法来训练模型。针对不同的场景，ModelArts提供大量的算法样例。以下章节提供了关于业务场景、算法学习方式、算法实现方式的指导。

选择算法的实现方式

ModelArts提供如下方式实现模型训练前的算法准备。

- 使用订阅算法

  ModelArts的AI Gallery提供了可以直接订阅的算法，不需要进行代码开发，即可使用现成的算法进行模型构建。

- 使用预置框架

  如果您需要使用自己开发的算法，可以选择使用ModelArts预置框架。ModelArts支持了大多数主流的AI引擎，详细请参见预置训练引擎。这些预置引擎预加载了一些额外的python包，例如numpy等；也支持您通过在代码目录中使用“requirements.txt”文件安装依赖包。使用预置框架创建训练作业请参考开发用于预置框架训练的代码指导。

- 使用预置框架 + 自定义镜像：

  如果先前基于预置框架且通过指定代码目录和启动文件的方式来创建的算法；但是随着业务逻辑的逐渐复杂，您期望可以基于预置框架修改或增加一些软件依赖的时候，此时您可以使用预置框架 + 自定义镜像的功能，即选择预置框架名称后，在预置框架版本下拉列表中选择“自定义”。

  此功能与直接基于预置框架创建算法的区别仅在于，镜像是由用户自行选择的。用户可以基于预置框架制作自定义镜像。基于预置框架制作自定义镜像代码可参考使用预置镜像制作自定义镜像用于训练模型章节。

- 完全自定义镜像：

  订阅算法和预置框架涵盖了大部分的训练场景。针对特殊场景，ModelArts支持用户构建自定义镜像用于模型训练。用户遵循ModelArts镜像的规范要求制作镜像，选择自己的镜像，并且通过指定代码目录（可选）和启动命令的方式来创建的训练作业。

  自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts上训练，请参考使用自定义镜像训练模型。由于自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置引擎无法满足需求，否则不推荐使用。

说明

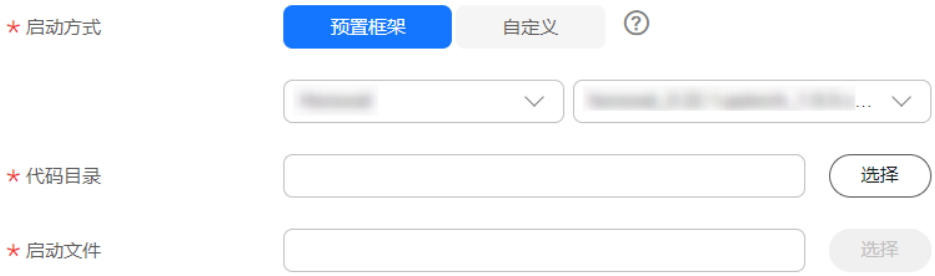
当使用完全自定义镜像创建训练作业时，“启动命令”必须在“/home/ma-user”目录下执行，否则训练作业可能会运行异常。

创建算法

您在本地或使用其他工具开发的算法，支持上传至ModelArts中统一管理。

1. 创建算法的准备工作。
   - 完成数据准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据集上传至OBS目录。
   - 准备训练脚本，并上传至OBS目录。训练脚本开发指导参见**开发用于预置框架训练的代码**或**开发用于自定义镜像训练的代码**。
   - 在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
   - 确保您使用的OBS目录与ModelArts在同一区域。
2. 进入算法创建页面。
   a. 登录**ModelArts管理控制台**，单击左侧菜单栏的“资产管理 > 算法管理”。
   b. 在“我的算法”管理页面，单击“创建”，进入“创建算法”页面。填写算法的基本信息，包含“名称”和“描述”。
3. 设置算法启动方式，有以下三种方式可以选择。
   - 设置算法启动方式（预置框架）

图 5-1 使用预置框架创建算法



需根据实际算法代码情况设置“代码目录”和“启动文件”。选择的预置框架和编写算法代码时选择的框架必须一致。例如编写算法代码使用的是TensorFlow，则在创建算法时也要选择TensorFlow。

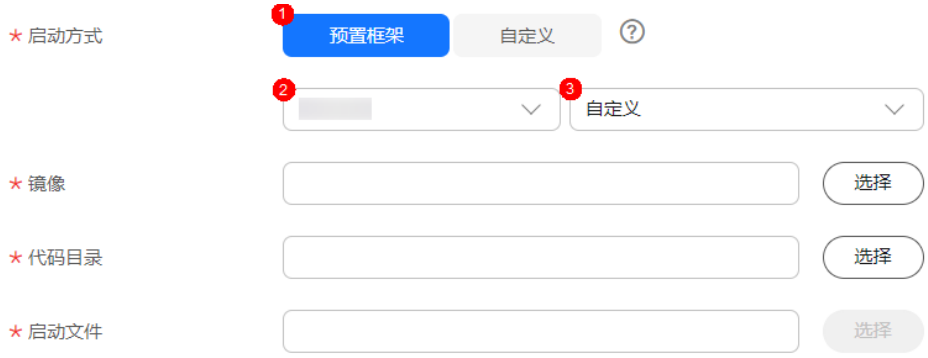
表 5-1 使用预置框架创建算法

参数	说明
“启动方式”	选择“预置框架”。 选择算法使用的预置框架引擎和引擎版本。

参数	说明
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业创建完成后，ModelArts会将代码目录及其子目录下载至训练后台容器中。 例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。 **说明** ● 编程语言不限。 ● 文件数（含文件、文件夹数量）小于或等于1000个。 ● 文件总大小小于或等于5GB。
“启动文件”	必须为“代码目录”下的文件，且以“.py”结尾，即ModelArts目前只支持使用Python语言编写的启动文件。 代码目录路径中的启动文件为训练启动的入口。

– 设置算法启动方式（预置框架+自定义）

图 5-2 使用预置框架+自定义镜像创建算法



需根据实际算法代码情况设置“镜像”、“代码目录”和“启动文件”。选择的预置框架和编写算法代码时选择的框架必须一致。例如编写算法代码使用的是TensorFlow，则在创建算法时也要选择TensorFlow。

表 5-2 使用预置框架+自定义镜像创建算法

参数	说明
“启动方式”	选择“预置框架”。 预置框架的引擎版本选择“自定义”。

参数	说明
“镜像”	用户制作的镜像需要提前上传到SWR，才可以在这里选择。制作镜像的方式请参见**训练作业的自定义镜像制作流程**。
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业启动时，ModelArts会将训练代码目录及其子目录下载至训练后台容器中。 例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。 **说明** ● 训练代码编程语言不限。训练启动文件必须为Python语言。 ● 文件数（含文件、文件夹数量）小于或等于1000个。 ● 文件总大小要小于或等于5GB。 ● 文件深度要小于或等于32
“启动文件”	必须为“代码目录”下的文件，且以“.py”结尾，即ModelArts目前只支持使用Python语言编写的启动文件。 代码目录路径中的启动文件为训练启动的入口。

选择预置框架+自定义时，该功能的后台行为与直接基于预置框架运行训练作业相同，例如：

- 系统将会自动注入一系列环境变量。
```
PATH=${MA_HOME}/anaconda/bin:${PATH}
LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
```
- 您选择的启动文件将会被系统自动以python命令直接启动，因此请确保镜像中的Python命令为您预期的Python环境。注意到系统自动注入的PATH环境变量，您可以参考下述命令确认训练作业最终使用的Python版本：
```
export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
docker run --rm {image} $(which python) -V
```
- 系统将会自动添加预置框架关联的超参。

– 设置算法启动方式（自定义）

图 5-3 完全使用自定义镜像创建算法

表 5-3 完全使用自定义镜像创建算法

参数	说明
“启动方式”	选择“自定义”。
“镜像”	用户制作的镜像需要提前上传到SWR，才可以在这里选择。制作镜像的方式请参见**训练作业的自定义镜像制作流程**。
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业启动时，ModelArts会将训练代码目录及其子目录下载至训练后台容器中。 例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。 **说明** ● 训练代码编程语言不限。训练启动文件必须为Python语言。 ● 文件数（含文件、文件夹数量）小于或等于1000个。 ● 文件总大小要小于或等于5GB。 ● 文件深度要小于或等于32

参数	说明
“启动命令”	必填，镜像的启动命令。 运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。 ● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。 python ${MA_JOB_DIR}/demo-code/train.py ● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。 bash ${MA_JOB_DIR}/demo-code/main.sh 启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。 当存在输入管道、输出管道、或是超参的情况下，请保证启动命令的最后一条命令是运行训练脚本。 原因：系统会将输入管道、输出管道、以及超参添加到启动命令的末尾，如果最后一条命令不是运行训练脚本则会报错。 例如：启动命令的最后一条是python train.py，且存在--data_url超参，系统正常运行会执行python train.py --data_url=/input。但是当启动命令python train.py后面有其他命令时，如下所示： python train.py pwd #反例，启动命令的最后一条命令不是运行训练脚本，而是pwd 此时，如果拼接了输入管道、输出管道、以及超参，系统运行实际执行的是python train.py pwd --data_url=/input，就会报错。

训练支持的自定义镜像使用说明请参考**自定义镜像的启动命令规范**。

4. 输入输出管道设置。

   训练过程中，算法需要从OBS桶或者数据集中获取数据进行模型训练，训练产生的输出结果也需要存储至OBS桶中。用户的算法代码中需解析输入输出参数实现ModelArts后台与OBS的数据交互，用户可以参考**准备模型训练代码**完成适配ModelArts训练的代码开发。

   – 输入配置

表 5-4 输入配置

参数	参数说明
参数名称	根据实际代码中的输入数据参数定义此处的名称。此处设置的代码路径参数必须与算法代码中解析的训练输入数据参数保持一致，否则您的算法代码无法获取正确的输入数据。 例如，算法代码中使用argparse解析的data_url作为输入数据的参数，那么创建算法时就需要配置输入数据的参数名称为“data_url”。

参数	参数说明
描述	输入参数的说明，用户可以自定义描述。
获取方式	输入参数的获取方式，默认使用“超参”，也可以选择“环境变量”。
输入约束	开启后，用户可以根据实际情况限制数据输入来源。输入来源可以选择“数据存储位置”或者“ModelArts数据集”。 如果用户选择数据来源为ModelArts数据集，还可以约束以下三种： ● 标注类型。数据类型请参考标注数据。 ● 数据格式。可选“Default”和“CarbonData”，支持多选。其中“Default”代表Manifest格式。 ● 数据切分。仅“图像分类”、“物体检测”、“文本分类”和“声音分类”类型数据集支持进行数据切分功能。可选“仅支持切分的数据集”、“仅支持未切分数据集”和“无限制”。数据切分详细内容可参考发布数据版本。
添加	用户可以根据实际算法添加多个输入数据来源。

– 输出配置

表 5-5 输出配置

参数	参数说明
参数名称	根据实际代码中的训练输出参数定义此处的名称。此处设置的代码路径参数必须与算法代码中解析的训练输出参数保持一致，否则您的算法代码无法获取正确的输出路径。 例如，算法代码中使用argparse解析的train_url作为训练输出数据的参数，那么创建算法时就需要配置输出数据的参数名称为“train_url”。
描述	输出参数的说明，用户可以自定义描述。
获取方式	输出参数的获取方式，默认使用“超参”，也可以选择“环境变量”。
添加	用户可以根据实际算法添加多个输出数据路径。

5. 定义超参。

   创建算法时，ModelArts支持用户自定义超参，方便用户查阅或修改。定义超参后会体现在启动命令中，以命令行参数的形式传入您的启动文件中。

   a. 单击“增加超参”手动添加超参。

   b. 编辑超参。

   说明

   为保证数据安全，请勿输入敏感信息，例如明文密码。

表 5-6 编辑超参数

参数	说明
名称	填入超参名称。 超参名称支持64个以内字符，仅支持大小写字母、数字、下划线和中划线。
类型	填入超参的数据类型。支持String、Integer、Float和Boolean。
默认值	填入超参的默认值。创建训练作业时，默认使用该值进行训练。
约束	单击“约束”。在弹出对话框中，支持用户设置默认值的取值范围或者枚举值范围。
必须	选择是或否。 ● 选择否，则在使用该算法创建训练作业时，支持在创建训练作业页面删除该超参。 ● 选择是，则在使用该算法创建训练作业时，不支持在创建训练作业页面删除该超参。
描述	填入超参的描述说明。 超参描述支持大小写字母、中文、数字、空格、中划线、下划线、中英文逗号和中英文句号。

6. 支持的策略。

   ModelArts支持用户使用自动化搜索功能。自动化搜索功能在零代码修改的前提下，自动找到最合适的超参，有助于提高模型精度和收敛速度。详细的参数配置请参考创建自动模型优化的训练作业。

   自动搜索目前仅支持“tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64”和“pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64”镜像

7. 添加训练约束。

   用户可以根据实际情况定义此算法的训练约束。

   – 资源类型：选择适用的资源类型，支持多选。
   – 多卡训练：选择是否支持多卡训练。
   – 分布式训练：选择是否支持分布式训练。

8. 当创建算法的参数配置完成后，单击“提交”，返回算法管理列表。

   在“我的算法”列表，单击算法名称进入详情页，可以查看算法详细信息。

   – 选择“基本信息”页签可以查看算法信息。

     “基本信息”页签，单击“编辑”，支持修改除名称和ID之外的算法信息。修改完成，单击“保存”即可完成修改。

   – 选择“训练列表”页签可以查看使用该算法的训练作业信息，例如训练作业名称、状态。

运行环境预览

创建算法时，可以打开创建页面右下方的运行环境预览窗口

，辅助您了解代码目录、启动文件、输入输出等数据配置在训练容器中的路径。

发布算法到 AI gallery

发布算法：创建完成的算法，支持发布到AI Gallery，并分享给其他用户使用。

登录**ModelArts管理控制台**，在“资产管理 > 算法管理 > 我的算法”页面，单击算法名称进入详情页，单击“发布”，在发布资产到AI Gallery页面，输入发布参数后，再次单击“发布”。

- 如果首次发布算法，则“发布方式”选择“创建新资产”，填写“资产标题”、选择发布区域等信息。
- 如果是为了更新已发布的算法版本，则“发布方式”选择“添加资产版本”，在“资产标题”下拉框中选择已有资产标题，填写“资产版本”。

说明

如果是首次在AI Gallery发布资产则此处会出现勾选“我已阅读并同意《华为云AI Gallery百模千态社区服务声明》和《华为云AI Gallery服务协议》”选项，需要阅读并勾选同意才能正常发布资产。

提交资产发布申请后，AI Gallery侧会自动托管上架，可以前往AI Gallery查看资产上架情况。

删除算法

须知

删除后，创建的算法资产会被删除，且无法恢复，请谨慎操作。

删除我的算法：在“资产管理 > 算法管理 > 我的算法”页面，“删除”运行结束的训练作业。您可以单击“操作”列的“删除”，在弹出的提示框中，输入**DELETE**，单击“确定”，删除对应的算法。

删除订阅算法：前往AI Gallery，在“我的资产 > 算法”中，单击我的订阅，对需要删除的算法单击“取消订阅”，在弹出的提示框中单击“确定”即可。

6 创建训练作业

6.1 创建训练自定义作业（推荐）

场景描述

说明

本章节内容仅适用于“西南-贵阳一”站点。控制台界面为新版样式。

在模型开发过程中，如何高效地优化模型性能是一个关键挑战。传统的模型训练方式往往需要反复尝试不同的模型结构、数据集和超参数组合，这不仅耗时耗力，还可能导致训练效果不理想。针对这一问题，ModelArts的训练模块提供了一套完整的解决方案。它支持创建训练作业、实时监控训练进展以及管理训练版本，帮助用户系统化地优化模型性能。通过ModelArts，用户可以轻松试验不同的模型结构、数据集和超参数组合，从而快速找到最优的模型配置和权重。

创建生产环境的训练作业有2种方式：

- 通过ModelArts控制台的方式创建生产环境的训练作业。本文提供了新版页面的操作指导，默认页面的操作指导请参见创建生产训练作业。
- 通过ModelArts提供的API接口创建生产环境的训练作业，详细操作请参见使用API创建训练作业。

约束限制

- 仅“西南-贵阳一”站点支持。
- 作业额度：默认支持创建10000条训练作业。
- 存储限制：ModelArts 不支持已启用桶加密功能的 OBS 桶，请在创建 OBS 桶时勿开启此选项。

前提条件

- 由于训练作业运行需消耗资源，为了避免训练失败请确保账户未欠费。
- 已经将用于训练作业的数据上传至OBS目录。
- 已经在OBS目录下创建了至少1个空的文件夹，用于存储训练输出的内容。

说明

ModelArts不支持加密的OBS桶，创建OBS桶时，请勿开启桶加密。

- 确保使用的OBS目录与ModelArts在同一区域。
- 检查是否配置了访问授权。如果未配置，请参见**快速配置ModelArts委托授权**完成操作。

计费影响

在ModelArts进行模型训练时，会产生计算资源和存储资源的累计值计费。计算资源为训练作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

创建作业操作流程一览

创建训练作业的操作步骤如下所示。

步骤一：进入创建训练作业页面：登录控制台,进入训练作业列表。

步骤二：训练模式：配置训练模式。

步骤三：基本信息：配置作业名称、描述等基本信息。

步骤四：训练配置：配置训练作业的镜像、启动命令和环境变量等参数。

步骤五：资源配置：配置资源池类型、实例规格、实例数、存储挂载、作业优先级和允许被抢占等信息。

步骤六：数据配置：配置数据集。

步骤七：发布到资产：配置训练的模型是否发布到资产。

步骤八：高可用配置：配置自动重启（无条件自动重启、作业卡死重启）等信息。

步骤九：访问配置：配置调试、SSH远程开发和生成节点间SSH免密互信文件等。

步骤十：可观测配置：配置Tensorboard、MindStudio Insight和采集Prometheus指标信息。

步骤十一：更多配置：配置日志、作业可见范围、自动停止、事件通知和标签等信息。

步骤十二：提交训练作业并查看作业状态。

步骤一：进入创建训练作业页面

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型开发与训练 > 模型训练”进入训练作业列表。
3. 单击“创建训练作业”，进入创建训练作业页面，默认进入新版页面，以下内容主要介绍新版创建训练页面操作。

步骤二：训练模式

表 6-1 训练模式

训练模式	解释说明
精调作业	适用于基于已有预训练模型（如盘古大模型、ResNet等）进行微调的场景。
自定义作业	适用于希望完全自定义训练流程，使用自研代码或特定框架镜像的场景。高自由度模型训练，允许用户使用自定义 Docker 镜像和算法对模型训练进行深度定制，完全开放训练流程控制权。 此处选择“自定义作业”模式。

步骤三：基本信息

在创建训练作业页面选择自定义作业，完成基础配置。

表 6-2 基础配置

参数名称	说明
作业名称	必填，训练作业的名称。 系统会自动生成一个名称，可以根据业务需求重新命名，命名规则如下： ● 支持1~64位字符。 ● 可以包含大小写字母、数字、中划线（-）或下划线（_）。
描述（可选）	训练作业的简介，便于在训练作业列表了解作业信息。 请输入 0-256 位字符，输入内容应只包含大小写字母、中文、数字、空格、中 划线、下划线、 逗号和句号。

步骤四：训练配置

表 6-3 训练配置

参数名称	说明
预置模板（可选）	单击“选择预置模板”根据需求过滤预置模板的类型（当前支持文本生成和图像理解）、品牌（当前支持Qwen），选择模板。 选择预置模板后，部分模板将自动填充当前作业的描述（可选）、运行镜像、启动命令代码目录、本地代码目录和环境变量等。请以实际情况为准。 用户也可在此基础上进行调整配置。

参数名称	说明
选择镜像	指定运行训练代码的容器镜像，有以下两种镜像： 预置镜像：选择平台提供的常用框架镜像（如 PyTorch 1.8、TensorFlow 2.1），适合大多数常见场景。 自定义镜像：使用您自己制作并推送至 SWR 镜像仓库的镜像，注意该镜像需要在ModelArts镜像管理内注册。 当平台预置的基础镜像无法满足您的依赖需求时，支持自定义镜像进行训练，所需镜像需提前注册到ModelArts镜像管理上。

启动命令	容器启动后执行的命令，用以启动您的训练脚本。
	● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。
	`python ${MA_JOB_DIR}/demo-code/train.py`
	● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。
	说明
	为保证数据安全，请勿输入敏感信息，例如明文密码。

代码目录（可选）	选择训练代码文件所在的OBS目录。当“选择镜像”选择“预置镜像”时，该参数必填；当“选择镜像”选择“自定义镜像”时，该参数可选。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小必须小于或等于5GB，文件数必须小于或等于1000个，文件深度必须小于或等于32。如果有预训练模型也请放置在代码目录下。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。
本地代码目录	用于指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。 本地代码目录不支持在/home/ma-user/modelarts/*，/home/ma-user/modelarts-dev/*，/home/ma-user/infer/*目录下，也不支持配置成/home/ma-user。 单击右侧的“运行环境预览”，可以查看训练作业的工作目录。
环境变量	根据业务需求增加环境变量。训练容器中预置的环境变量请参见**管理训练容器环境变量**。 ● 单击“添加”，添加环境变量信息，总数不超过100条。 ● 单击“本地上传”可以本地批量导入环境变量，需要按模板填写环境变量且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。

步骤五：资源配置

表 6-4 资源配置

参数名称	说明
资源池类型	● “公共资源池”：公共资源池由系统提供，供平台所有租户共享使用，不需要用户创建。 ● “专属资源池”：专属资源池需要提前单独创建，不与其他租户共享，具体创建操作请参见创建专属资源池。
资源池	仅选择“专属资源池”后出现此参数，在“资源池”中单击“选择资源池”，在右侧弹窗中选择要使用的专属资源池或者逻辑子池后，单击“确定”。 选择界面支持查看当前专属资源池的名称、节点池规格、可用节点数/最大节点数、可用NPU/GPU设备数、可用CPU（vCPUs）、可用内存（GiB）以及资源碎片。单击资源碎片列的“查看”可以查看碎片详情，确认资源池是否满足训练需求。 选择完成后，此处会展示所选资源池信息，也可单击“重新选择”再次选择资源池。
实例规格	资源池类型为“专属资源池”时，需要先选定“资源池”，资源池类型为“公共资源池”时直接单击，然后在右侧弹窗中选择实例规格后，单击“确定”。 ● 最上方显示了CPU、NPU和GPU规格类型，下方表格可查看规格名称、内存、规格参考价等详情。 ● 规格名称中体现了节点规格的卡数、卡型号、CPU/GPU以及内存大小等信息。 ● 如果部分资源类型不可见或不可选，表示不支持。 **说明** ● 实例规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。 ● 资源池类型为“公共资源池”时：使用自定义镜像（创建方式为自定义算法，启动方式为自定义）创建作业时，请确保您的“镜像”类型、“资源类型”和“实例规格”类型保持一致（例如：均为GPU类型）。否则会导致训练作业失败。
实例数	必填，根据需要选择实例数的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数 > 1”时，创建的是分布式训练作业，更多分布式训练配置请参见分布式训练功能介绍。 **如果用户开启了对应集群的高可用节点特性，资源池可供训练作业调度的总节点数为实际可用节点数与高可用节点数之和。高可用节点由系统预留，用户作业无法直接调度，仅在节点发生异常时用于故障恢复。** 在创建分布式训练作业前，建议提前安装所有的pip依赖（请参见在镜像中安装pip依赖），因为当节点数大于10时，系统会自动删除pip源配置，当训练过程中涉及pip install操作时可能会训练失败。

参数名称	说明
挂载存储	资源规格选择“专属资源池”时，支持添加多种存储挂载方式，以提高数据的访问效率。两种区别可参见**表12-1**。 ● “添加拓展存储SFS Turbo”： 当ModelArts和SFS Turbo间网络直通时，训练作业支持挂载多个SFS Turbo存放训练数据。文件系统支持重复挂载，但是挂载路径不可重复。最多可以挂载5个盘。配置ModelArts和SFS Turbo间网络直通，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。 **图 6-1** SFS Turbo – “文件系统”：选择一个SFS Turbo。 – “云上挂载路径”：输入SFS Turbo对应在训练容器内的云上挂载路径。不能为“/”目录，不能为“/cache”、“/home/ma-user/modelarts”等系统已经默认挂载的路径。 – “存储位置”：选择SFS Turbo的存储位置。如果用户配置了文件夹控制权限，请选择存储位置；如果用户未配置文件夹控制权限，可以保持默认值“/”或者自定义位置。 – “挂载方式”：显示挂载SFS Turbo的权限。根据SFS Turbo存储位置的权限显示“读写”或“只读”，如果用户未配置文件夹控制权限，则该参数不可见。如果需要设置SFS Turbo的文件夹权限，请参考**权限管理**文档配置。 – “挂载参数”：支持配置SFS挂载参数实现训练加速优化，具体参数说明请参见**设置极速文件存储挂载参数**。不设置时，默认配置如下参数： mountOptions: - vers=3 - timeo=600 - nolock - hard **说明** 当用户配置拓展存储SFS Turbo时，前端页面会自动从**SFS Turbo**服务获取最新的存储容量和配置信息，以确保数据有效性和准确性。 1. **查询文件系统详细信息**。 2. **获取文件系统列表**。 ● “添加拓展存储OBS”： 训练作业支持挂载一个 OBS 桶用于存放训练数据。

参数名称	说明
	警告 由于对象存储（OBS）的对象语义与POSIX文件系统不同，rclone通过FUSE方式提供“类文件系统”访问，存在以下限制： ● **写入语义限制**：不支持随机写入和追加写入，不支持对已有文件进行原地修改，包括随机写（seek write）和追加写（append）。例如，以 r+、w+、a 或 a+ 模式打开并修改同一文件，或不断追加写入日志文件（如 bash while true; do echo "$(date) line" >> app.log; sleep 1; done）。 ● **权限限制**：挂载显示的权限多为rclone伪造（或统一映射），不能作为真实的多用户权限系统使用，因此无法使用 chmod 或 chown 修改文件权限。 ● **并发写入限制**：并发写入同一路径的行为不可控，多个进程写入同一文件时无法确认最终内容。 ● **链接限制**：不支持硬链接，符号链接可能被当作普通文件存储（保存链接文本）。例如，ln -s target.txt link.txt 可能会导致直接失败，或者 link.txt 被当成普通文件上传，内容是字符串 target.txt。
作业调度优先级	● 使用专属资源池创建训练作业时，支持使用和修改训练作业的调度优先级。使用公共资源池时，不支持此参数配置。 ● 作业默认按照优先级从高到低依次处理，对于相同优先级的任务，则根据提交时间的先后顺序进行调度，最终结合实际的资源情况决定调度顺序(保证在资源充足、相同优先级下，先提交的任务先调度)。 ● 修改选择数字后可修改作业进入队列的优先级。优先级可以设置数字1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。 ● 设置优先级为“3”时，还需要配置用户权限才会生效。用户权限设置请参见“**设置作业为高优先级权限**”。 ● 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。
允许被抢占	● 使用专属资源池创建训练作业时，允许设置训练作业被抢占。使用公共资源池时，不支持此参数配置。 ● 开启后，当资源池剩余资源不足时，允许被抢占的作业可能会被终止并重新排队。为避免训练进度丢失，请在开启此功能前完成断点续训的设置，操作指导请参考**设置断点续训练**。

步骤六：数据配置

表 6-5 数据配置

参数名称	说明
训练集	训练集用于提升模型在特定任务上的表现。 目前支持最多选择三个数据集进行训练，支持混选“预置数据”或“我的数据”。 预置数据：平台官方提供的模板数据。 我的数据：用户上传的个人数据。如果无法选择到已有的数据资产，请前往“资产管理 > 数据”进行上线操作。

步骤七：发布到资产

参数名称	说明
发布到资产	开启后，系统自动将模型产物发布成模型资产，可以在平台进行推理、评测等操作。
模型输出路径	● 存储位置：模型训练完成后存储位置。 ● 云上挂载路径： 用户使用专属资源池时显示此参数。 系统将存储位置中的文件目录挂载到训练容器中指定路径。用户可自定义该路径，不支持配置为/home/、/home/ma-user/、/home/ma-user/modelarts/等系统目录。
自动发布到资产	勾选开关后，训练后的模型将自动上传至“资产管理-我的模型”。 默认不勾选。
选择发布方式	训练后的模型可以发布为**新模型**或**已有模型新版本**。 **新模型**：发布的模型为一个全新的模型，在“资产管理-我的模型”中是一个新的资产模型。 **已有模型新版本**：发布的模型是“资产管理-我的模型”中的同一个命名的模型，仅模型版本号有变化。
模型名称	设置新模型的名称。 支持2-128位，只包含中文、字母、数字、中划线、下划线，且以中文、字母开头，以中文、字母、数字结尾。
模型类型	根据发布的模型类型选择。
模型品牌	模型品牌。

参数名称	说明
模型版本号	发布为新模型时，版本号为V1。 发布为已有模型新版本时，版本号在该模型上一个版本号自动加1。 **注意：**模型版本号不支持修改，为系统自动生成。
描述（可选）	训练后模型的描述信息。该字段可选，最大长度256字符。

步骤八：高可用配置

表 6-6 高可用配置

参数名称	说明
容错与恢复	设置训练作业是否设置自动重启。 ● 不勾选（默认）：表示不启用自动重启，训练异常则直接停止作业。 ● 勾选：当训练作业因环境问题、作业卡死等原因异常时，系统将自动检测故障并根据相应策略进行处理，提高训练成功率。系统支持进程级、容器级、作业级自动重启恢复策略，无需额外配置，可自动匹配与升级恢复策略。 为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见**设置断点续训练**。 当训练过程中触发了自动重启，则系统会记录重启信息，在训练作业详情页可以查看故障恢复详情，具体请参见**训练作业容错检查**。
最大重启次数	当勾选“自动重启”时可配置。 当训练作业的自动重启次数达到最大重启次数，且训练作业依旧异常，则直接停止作业。 ● 缺省值：3 ● 取值范围：1~128 提交创建训练作业后不支持修改重启次数，请合理设置次数。
无条件自动重启	当勾选“自动重启”时可配置。勾选无条件自动重启后，只要系统检测到训练异常，就无条件重启训练作业。为了避免无效重启浪费算力资源，系统最多只支持连续无条件重启3次。
作业卡死重启	当勾选“自动重启”时可配置。系统支持自动监控作业进程的状态和资源利用率来判定作业是否卡死，开启此开关后，支持将标记为卡死的作业进行进程级自动重启，以提高资源使用率。 CPU规格不支持作业卡死重启。 因系统无法核实代码逻辑且检测存在周期性，卡死检测存在一定的误报概率，开启开关即表示接受误报率。为了避免无效重启浪费算力资源，系统最多只支持连续作业卡死重启3次。

步骤九：访问配置

表 6-7 访问配置

参数名称	说明
JupyterLab	开启后，可支持JupyterLab等开发工具进行在线调试。
SSH远程开发	勾选开启后，可以通过本地 IDE 远程连接到训练作业实例，进行代码的在线调试与运行。系统将自动为每个实例启动 SSHD 服务，并配置实例间的 SSH 免密登录，以便于跨节点协同操作。 需创建密钥对。 勾选后，“生成节点间SSH免密互信文件”功能不可用。
生成实例间SSH 免密互信文件	选择是否生成实例间SSH免密互信文件。 ● 不勾选（默认）：表示不生成实例间SSH免密互信文件。 ● 勾选：表示生成实例间SSH免密互信文件。在使用基于MPI和Horovod框架的自定义镜像进行分布式训练时，为确保实例间的通信顺畅，必须配置实例间SSH免密互信，确保分布式训练任务顺利进行。 同时还需配置“SSH免密互信文件目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。更多操作指导请参见**自定义镜像训练作业配置实例间SSH免密互信**。 勾选后，“SSH远程开发”功能不可用。

步骤十：可观测配置

表 6-8 可观测配置

参数名称	说明
Tensorboard	TensorBoard是TensorFlow的可视化工具包，提供机器学习实验所需的可视化功能和工具。TensorBoard能够有效地展示训练过程中的计算图、各种指标随时间的变化趋势以及训练中使用到的数据信息，相关概念请参考**TensorBoard官网**。 使用公共资源池时，不支持此参数配置。 用于存储可视化工具Tensorboard产生的结果文件。
MindStudio Insight	MindStudio Insight能可视化展现出训练过程中的标量、图像、计算图以及模型超参等信息，支持基于MindSpore引擎的训练作业。MindStudio Insight相关概念请参考**MindSpore官网**。 使用公共资源池时，不支持此参数配置。 用于存储可视化工具MindStudio Insight产生的结果文件。

参数名称	说明
监控指标对接AOM	选择是否启用采集Prometheus指标。通过在训练容器里配置采集Prometheus指标的参数，在训练过程中，系统会定时获取指标数据上传到AOM，您可以直接在AOM控制台查看自定义的Prometheus指标。 ModelArts提供了2种配置方式。 ● 方式一：用户提供一个可被平台调用的HTTP进程来定时获取指标。操作指导请参见**通过Prometheus指标采集到AOM（HTTP）**。 – “获取指标类型”选择“http获取指标”。 – 填写“采集URL”和“端口号”， 该方式要确保配置的URL和端口号与指标采集进程一致，否则可能导致指标无法上报，同时，训练作业所在的网络环境要允许访问配置的URL和端口号，避免因网络问题导致指标采集失败。 ● 方式二：用户提供一个可被平台执行的Linux命令来定时执行获取指标。操作指导请参见**通过Prometheus指标采集到AOM（Command）**。 – “获取指标类型”选择“命令行获取指标”。 – “执行命令”：读取指标的Linux命令，例如填写“cat”。 – “执行命令参数”：填写自定义指标文件的路径，例如“/XXX/a.prom”。 该方式要确保配置的执行命令和执行命令参数是可以稳定输出指标的，且响应要在秒级，否则指标可能会采集失败。

步骤十一：更多配置

参数名称	说明
永久保存日志	当实例规格选择Ascend资源时，默认开启“永久保存日志”。 当实例规格选择CPU或GPU资源时，支持选择是否打开“永久保存日志”开关。 ● 勾选（默认）：表示永久保存日志，此时必须配置“日志路径”，平台会将训练日志永久保存至指定的OBS路径。 ● 不勾选：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
日志路径	当开启“永久保存日志”时，必须配置“日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。

参数名称	说明
作业可见范围	分为“工作空间内可见”和“仅创建者可见”。 ● “工作空间内可见”：创建的训练作业在当前工作空间内，默认其他子用户均可看到该训练作业。 ● “仅创建者可见”：默认创建者可见，其他人希望查看该作业需要额外申请“modelarts:trainJob:listAll”权限（查看训练作业列表（包含仅创建者可见的作业）权限）。
自动停止	选择是否启用训练作业自动停止。 ● 不勾选（默认）：表示训练作业将一直运行直至训练完成。 ● 勾选：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。
事件通知	选择是否启用训练作业的事件通知。 ● 不勾选（默认）：表示不启用消息通知服务。 ● 勾选：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题，并为主题添加订阅。当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”“”、“作业终止”“”、“作业疑似卡死”“”、“作业重启”、“作业抢占”等。 **说明** ● 使用消息通知服务会产生相关服务费用，详细信息请参见**消息通知服务计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
标签	当需要使用同一标签标识多种云资源时，即所有服务均可在标签输入框下拉选择同一标签，则建议在TMS中创建预定义标签。标签详细用法请参见**使用TMS标签实现资源分组管理**。 最多支持添加20条标签。

步骤十二：提交训练作业并查看作业状态

当参数配置完成后，单击“提交”，创建训练作业任务。

训练作业一般需要运行一段时间，前往训练作业列表，可以查看训练作业的基本情况。

- 在训练作业列表中，刚创建的训练作业状态为“等待中”。
- 当训练作业的状态变为“已完成”时，表示训练作业运行结束，其生成模型将存放至模型资产供后续使用。

- 当训练作业的状态变为“运行失败”或“异常”时，可以单击训练作业的名称进入详情页面，通过查看日志等手段处理问题。

常见问题

1. 在ModelArts中训练好后的模型如何获取?
2. 在ModelArts上如何提升训练效率并减少与OBS的交互?
3. 如何查看ModelArts训练作业资源占用情况?

6.2 创建模型精调作业（推荐）

什么是模型精调

在大模型训练中，精调（或“微调”）（Fine-tuning）是指通过特定领域的数据集对已经做过全量预训练模型（Pre-trained Model, PT）进行二次训练的方法。通过精调能够更新模型权重，使模型能够更有效地应对具体的任务需求。这一阶段使模型能够精确执行如文案生成、代码生成和专业问答等特定场景中的任务。

模型精调使用场景

在模型开发过程中，精调的主要的使用场景如表6-9所示。

表 6-9 精调场景

精调场景	场景说明	精调目标
领域适配	基模缺乏垂直领域知识（如医学诊断标准、专用编程语言、企业内部术语）	注入领域Knowledge Graph或术语模式，降低模型幻觉，提升专业度。
指令/格式依从	需要模型严格输出JSON、SQL、XML，或遵循特定的CoT (Chain of Thought)逻辑，而提示词工程稳定性不够。	通过特定精调数据集固化输出结构，降低解析错误率。
风格对齐	需要模型按照特定风格回答问题。如角色扮演、拟人化客服、特定文风的公文写作。	调整logits分布，使其符合特定语料的分布特征。

约束限制

- 仅“西南-贵阳一”站点支持。
- 存储限制：ModelArts不支持已启用桶加密功能的OBS桶，请在创建OBS桶时勿开启此选项。

前提条件

- 由于精调作业运行需消耗资源，为了避免精调失败请确保账户未欠费。
- 已经将用于训练作业的数据上传至OBS目录。关于如何创建OBS桶和上传文件，请参见OBS控制台快速入门。

- 已经在OBS目录下创建了至少1个空的文件夹，用于存储训练输出的内容。
- 确保使用的OBS目录与ModelArts在同一区域。
- 检查是否配置了访问授权。如果未配置，请参见**快速配置ModelArts委托授权**完成操作。

计费影响

在ModelArts进行模型精调时，会产生计算资源和存储资源的累计值计费。计算资源为精调作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

模型精调流程

创建精调作业的操作步骤如下所示。

步骤一：进入创建训练作业页面：登录控制台，进入训练作业列表。

步骤二：配置精调参数：配置模型精调的各项配置参数。

步骤三：提交精调作业并查看作业状态。

步骤一：进入创建训练作业页面

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型开发与训练 > 模型训练”进入训练作业列表。
3. 单击“创建训练作业”，进入创建训练作业页面，默认进入新版页面，以下内容主要介绍新版创建训练页面操作。

步骤二：配置精调参数

表 6-10 创建调优作业参数说明

参数		说明
训练模式	精调作业	适用于基于已有预训练模型（如Qwen系列）进行微调的场景。 低门槛模型训练，使用平台预置的优质模型资产，用户无需关注镜像构建、环境依赖或代码调试，只需上传业务训练数据，调整关键参数。 此处选择“精调作业”。
	自定义作业	适用于希望完全自定义训练流程，使用自研代码或特定框架镜像的场景。
基本信息	任务名称	自定义调优作业名称。支持1~64位，只包含字母、数字、中划线、下划线的名称。
	描述（可选）	自定义调优作业简介。该字段为可选，支持最大输入256字符。

参数		说明
训练配置	选择模型	单击“请选择模型”，在“选择模型”对话框中选择“预制模型”或“我的模型”下面的模型。支持按照**来源**、**类型**、**品牌**过滤模型，也支持在搜索框搜索关键字查找模型。 **说明** 在“选择模型”对话框中，可以选择的模型代表该模型支持调优，未显示的模型不支持调优，请以实际环境为准。
	训练类型	当前支持对模型做“微调”。
	训练目标	ModelArts支持全量微调、LoRA微调两种调优类型。不同模型支持的调优类型不同。 ● **全量微调**：直接在模型上训练，影响模型全量参数的微调训练，效果较好，收敛速度较慢，训练时间较长。 ● **LoRA微调**：冻结原模型，通过往模型中加入额外的网络层，并只训练这些新增的网络层参数，效果接近或略差于全参训练，收敛速度快，训练时间短。
	模型输出路径	支持精调后的模型存储至OBS和SFS Turbo。当前仅支持精调后模型存储至OBS，后续将支持存储至SFS Turbo。 **说明** **注意**：精调后的模型存储至OBS，需要保证提前订阅OBS服务，并且保证OBS有充足空间。
资源配置	资源池类型	资源池分为公共资源池与专属资源池，目前仅支持使用公共资源池。 ● 公共资源池由所有租户共享使用。 ● 专属资源池需单独创建，不与其他租户共享。
	规格	选择规格，规格中描述了服务器类型、型号等信息，仅显示模型支持的资源。
	实例数	必填，根据需要选择实例数的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数 > 1”时，创建的是分布式训练作业，更多分布式训练配置请参见分布式训练功能介绍。
数据配置	训练集	在弹出对话框可以选择“预置数据”和“我的数据”。**预置数据**是平台内置的常用的数据集。**我的数据**是自己的原始或者处理过的数据集。请根据需要选择不同数据集。
训练参数	学习率/learning_rate	设置每个迭代步数（iteration）模型参数/权重更新的速率。学习率设置的过高会导致模型难以收敛，过低则会导致模型收敛速度过慢。
	学习率衰减比率/min_learning_rate	用于控制训练过程中学习率下降的幅度。 计算公式为：最低学习率 = 初始学习率 × 学习率衰减比率。

参数		说明
	迭代步数/Iterations	计算得出的模型参数/权重更新的次数。 当数据集是数百量级，则建议迭代4~8个epoch（epoch表示整个数据集被完整地用于一次训练的次数）；当数据集是数千量级，则建议迭代2~4个epoch；当数据集是更大数量，则建议迭代1~2个epoch。 总Iterations = 整个数据集完整训练需要的Iterations * epoch。例如，当一个数据集有3200条数据，完整训练一个数据集的Iterations为100，迭代2个epoch，总Iterations就是200。
	训练轮数/Epoch	表示完成全部训练数据集训练的次数。每个轮次都会遍历整个数据集一次。
	数据批量大小/batch_size	数据批量是指训练过程中将数据集分成小批次进行读取，并设定每个批次的数据大小。 通常，较大的批量能够使梯度更加稳定，有助于模型的收敛。然而，较大的批量也会占用更多显存，可能导致显存不足，并延长每次训练时间。
	序列长度/sequence_length	训练单条数据的最大长度，超过该长度的数据在训练时将被截断。
	热身比例/warmup_ratio	热身比例是指在模型训练初期逐渐增加学习率的过程。 由于训练初期模型的权重通常是随机初始化的，预测能力较弱，若直接使用较大的学习率，可能导致更新过快，进而影响收敛。为解决这一问题，通常在训练初期使用较小的学习率，并逐步增加，直到达到预设的最大学习率。通过这种方式，热身比例能够避免初期更新过快，从而帮助模型更好地收敛。
	数据条数	输入数据集中的总数据条数。
	数据类型/data_type	对于部分精炼模型，需要使用开源格式数据集。如Alpaca \ShareGPT格式。 可选参数：AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler，GeneralInstructionHandler
	权重衰减系数	通过在损失函数中加入与模型权重大小相关的惩罚项，鼓励模型保持较小的权重，防止过拟合或模型过于复杂。
	模型保存策略	save_checkpoint_steps/save_checkpoint_epoch，训练过程中是按迭代步数，还是训练轮数保存Checkpoint文件。
	Checkpoint保存间隔	save_checkpoint_steps，训练过程中每隔多少个训练步长保存一次模型Checkpoint文件。
	模型保存步数	save_checkpoint_epoch，训练过程中每个多少训练轮数保存一次模型Checkpoint文件。

参数		说明
发布模型	自动发布到资产	勾选开关后，训练后的模型将自动发布至控制台“资产管理-我的模型”列表。
	选择发布方式	精调后的模型可以发布为**新模型**或**已有模型新版本**，请根据需要选择不同的发布方式。 **新模型**：发布的精调模型为一个全新的模型，在控制台选择“资产管理-我的模型”列表中是一个新的模型。 **已有模型新版本**：发布的精调模型是“资产管理-我的模型”中的同一个命名的模型，仅模型版本号有变化，在**模型详情**中可以查看该模型的新版本号。
	模型名称	设置调优后产生的新模型的名称。 支持2-128位，只包含中文、字母、数字、中划线、下划线，且以中文、字母**开头**，以中文、字母、数字**结尾**。
	模型版本号	发布为新模型时，版本号为V1。 发布为已有模型新版本时，版本号在该模型上一个版本号自动加1。 **注意：**模型版本号不支持修改，为系统自动生成。
	描述	训练后模型的描述信息。该字段可选，最大长度256字符。
更多配置	checkpoints配置	checkpoints：在模型训练过程中，用于保存模型权重和状态的机制。 ● 关闭：关闭后不保存checkpoints，无法基于checkpoints执行续训操作。 ● 自定义：根据设置保存指定数量的checkpoints。

步骤三：提交精调作业并查看作业状态

当参数配置完成后，单击“提交”，创建精调作业任务。

精调作业一般需要运行一段时间，前往精调作业列表，可以查看精调作业的基本情况。

- 在精调作业列表中，刚创建的精调作业状态为“等待中”。
- 当精调作业的状态变为“已完成”时，表示精调作业运行结束，其生成模型将存放至模型资产供后续使用。
- 当精调作业的状态变为“运行失败”或“异常”时，可以单击训练作业的名称进入详情页面，通过查看日志等手段处理问题。

精调常见问题

1. 如何让训练产出的模型具备思考模式?

   如果想要训练带思考模式的模型，需要在Alpaca的output字段的头尾增加<think>和</think>标签，或者在output字段的头增加引导思考的prompt，比如“Let's think step by step”。当前预置数据是Alpaca格式数据，不带<think>标签，训练后的模型大概率只能回答不能思考。

示例一：

```
{
    "instruction": "不要问我从哪里来，我的故乡在青龙山 ",
    "input": "",
    "output": "<think>用户分享了一句歌词：“不要问我从哪里来，我的故乡在青龙山”。首先，我要确认这首歌的出处，xxx，最后xxx\n</think>\n\n“不要问我从哪里来，我的故乡在青龙山”这句话似乎是对经典歌曲《橄榄树》的歌词改编（原词为“不要问我从哪里来，我的故乡在远方”）。"
}
```

示例二：

```
{
    "instruction": "Karin 's science class weighed plastic rings for an experiment . They found that the orange ring weighed 0.08333333333333333 ounce , the purple ring weighed 0.3333333333333333 ounce , and the white ring weighed 0.4166666666666667 ounce . What was the total weight of the plastic rings ?",
    "input": "",
    "output": "Let's think step by step. \n\nThe orange ring weighed 0.08333333333333333 ounce. \nThe purple ring weighed 0.3333333333333333 ounce. \nThe white ring weighed 0.4166666666666667 ounce. \n\nTo find the total weight of the plastic rings, we need to add the weights of the orange ring, the purple ring, and the white ring. \n\n0.08333333333333333 + 0.3333333333333333 + 0.4166666666666667 = 0.8333333333333334 \nTherefore, the answer (arabic numerals) is 0.8333333333333334."
}
```

6.3 创建生产训练作业

说明

本章节内容适用除“西南-贵阳一”站点之外的所有华为云站点。控制台界面为旧版样式。

在模型开发过程中，如何高效地优化模型性能是一个关键挑战。传统的模型训练方式往往需要反复尝试不同的模型结构、数据集和超参数组合，这不仅耗时耗力，还可能导致训练效果不理想。针对这一问题，ModelArts的训练模块提供了一套完整的解决方案。它支持创建训练作业、实时监控训练进展以及管理训练版本，帮助用户系统化地优化模型性能。通过ModelArts，用户可以轻松试验不同的模型结构、数据集和超参数组合，从而快速找到最优的模型配置和权重。

创建生产环境的训练作业有2种方式：

- 通过ModelArts控制台的方式创建生产环境的训练作业。本文提供了新版页面的操作指导，默认页面（即旧版页面）的操作指导请参见创建生产训练作业（旧版）。
- 通过ModelArts提供的API接口创建生产环境的训练作业，详细操作请参见使用API创建训练作业。

约束限制

默认支持创建10000条训练作业，剩余额度可以在训练作业列表页查看。

图 6-2 查看训练作业剩余额度

前提条件

- 由于训练作业运行需消耗资源，为了避免训练失败请确保账户未欠费。
- 已经将用于训练作业的数据上传至OBS目录。
- 已经在OBS目录下创建了至少1个空的文件夹，用于存储训练输出的内容。

  说明

  ModelArts不支持加密的OBS桶，创建OBS桶时，请勿开启桶加密。
- 确保使用的OBS目录与ModelArts在同一区域。
- 检查是否配置了访问授权。如果未配置，请参见**快速配置ModelArts委托授权**完成操作。

计费影响

在ModelArts进行模型训练时，会产生计算资源和存储资源的累计值计费。计算资源为训练作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

操作流程介绍

创建训练作业的操作步骤如下所示。

步骤一：进入创建训练作业页面。

步骤二：基础配置：配置基础信息，例如作业模式、作业名称、描述、实验等。

步骤三：环境配置：配置环境信息，例如创建方式（算法选择）、启动方式、引擎及版本等。

步骤四：训练配置（可选）：配置训练作业的输入、输出、超参、环境变量等参数。

步骤五：资源配置：配置资源池类型、实例规格、实例数、挂载存储等信息。

步骤六：高可用配置：配置自动重启（无条件自动重启、作业卡死重启）等信息。

步骤七：调度配置：调整作业优先级、是否允许被抢占、自动停止等。

步骤八：更多配置：配置日志、事件通知和标签等信息。

步骤九：提交训练作业并查看作业状态。

步骤一：进入创建训练作业页面

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型训练 > 训练作业”进入训练作业列表。
3. 单击“创建训练作业”，进入创建训练作业页面，默认进入新版页面，以下内容主要介绍新版创建训练页面操作。

步骤二：基础配置

在创建训练作业页面，完成基础配置。

表 6-11 基础配置

参数名称	说明
作业模式	选择生产模式。 训练作业支持在云上和本地对训练代码进行调试，调试完成的代码可直接创建生产训练作业。 ● “生产模式”：在生产环境运行训练作业，初次上云训练建议先使用调试模式，再到生产环境运行，避免产生资源浪费。 ● “调试模式”：可以实时修改训练代码并迭代调试。 本文提供的是生产模式训练作业的操作指导，调试模式的操作指导请参见**创建调试训练作业并使用JupyterLab调试**。
作业名称	必填，训练作业的名称。 系统会自动生成一个名称，可以根据业务需求重新命名，命名规则如下： ● 支持1~64位字符。 ● 可以包含大小写字母、数字、中划线（-）或下划线（_）。
描述（可选）	训练作业的简介，便于在训练作业列表了解作业信息。 请输入 0-256 位字符，输入内容应只包含大小写字母、中文、数字、空格、中 划线、下划线、 逗号和句号。
实验设置	勾选“实验设置”后，可以将训练作业放入实验中分类，并有序地进行管理，常用于多任务的版本管理等场景。 实验功能有助于管理和优化训练作业，如通过重建作业时微调超参得到的新作业，可在实验中有序归纳与比较作业结果，从而有效地寻找最佳训练配置。 ● “选择已有实验”：将训练作业纳入已有的实验中。需要选择“实验名称”。 ● “创建新实验”：将训练作业纳入新建的实验中。需要配置“新实验名称”和“新实验描述（可选）”。 不勾选实验设置，表示训练作业不在实验中进行统一管理。

步骤三：环境配置

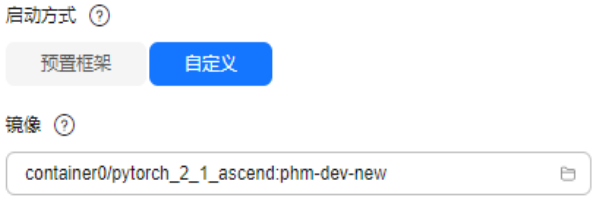
创建训练作业时，需要配置算法来源、启动方式、算法对应的引擎及版本、代码目录等信息。训练作业支持多种创建方式。

- **使用自定义算法创建**：使用平台预置的训练常用框架、镜像或自定义镜像的方式创建训练作业。

表 6-12 环境配置（自定义算法）

参数名称	说明
创建方式	必选，选择“自定义算法”。

参数名称	说明
启动方式	创建方式选择自定义算法时，启动方式参数必选。启动方式有以下两种： ● “预置框架”：通过平台预置的训练常用框架和镜像创建训练作业。 ● “自定义”：通过自定义镜像创建训练作业。 当平台预置的基础镜像中的软件无法满足实际程序运行需求时，支持自定义镜像进行训练，所需镜像需提前上传到SWR容器镜像服务上，详细镜像制作指导请参见**准备模型训练镜像**。
引擎及版本	当“启动方式”选择“预置框架”时，需要选择训练作业要使用的预置框架引擎和引擎版本。 选择的AI引擎和编写算法代码时使用的框架必须一致。例如编写算法代码使用的是PyTorch，则在创建作业时也要选择PyTorch。

参数名称	说明
镜像	选择训练需要的容器镜像，训练镜像制作要求请参见**准备模型训练镜像**。 ● 当“启动方式”选择“预置框架”，且引擎版本选择“自定义”时，需要从容器镜像中选择合适的镜像。 **图 6-3** 启动方式-预置框架 ● 当“启动方式”选择“自定义”时，需要从容器镜像中选择合适的镜像。 **图 6-4** 启动方式-自定义  容器镜像地址的填写支持如下方式。 ● 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 ● 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息（swr.<region>.myhuaweicloud.com），系统会自动拼接域名地址。例如：某公开镜像的SWR地址为“swr.<region>.myhuaweicloud.com/test-image/tensorflow2_1_1:1.1.1”，则此处填写“test-images/tensorflow2_1_1:1.1.1”。
代码来源	选择代码来源。默认是“对象存储服务OBS”。 ● “OBS对象存储”：如果训练代码存放在OBS中，则选择“对象存储服务OBS”。

参数名称	说明
代码目录	仅当“代码来源”选择“对象存储服务OBS”时显示该参数。 选择训练代码文件所在的OBS目录。当“启动方式”选择“预置框架”时，该参数必填；当“启动方式”选择“自定义”时，该参数可选。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小必须小于或等于5GB，文件数必须小于或等于1000个，文件深度必须小于或等于32。如果有预训练模型也请放置在代码目录下。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。 **说明** 如果您的OBS桶中有需要特别保护的敏感数据，请自行对数据加密后再存 放到OBS桶。
启动文件	选择或输入代码目录中训练作业的Python启动脚本。当“启动方式”选择“预置框架”时，该参数必填；当“启动方式”选择“自定义”时，不涉及该参数。 ModelArts只支持使用Python语言编写的启动文件，因此启动文件必须以“.py”结尾。

启动命令	镜像的启动命令。当“启动方式”选择“预置框架”时不涉及该参数；当“启动方式”选择“自定义”时，该参数必填。
	运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。
	● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。
	`python ${MA_JOB_DIR}/demo-code/train.py`
	● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。
	当存在输入管道、输出管道、或是超参的情况下，请保证启动命令的最后一条命令是运行训练脚本。
	原因：系统会将输入管道、输出管道、以及超参添加到启动命令的末尾，如果最后一条命令不是运行训练脚本则会报错。
	例如：启动命令的最后一条是python train.py，且存在--data_url超参，系统正常运行会执行python train.py --data_url=/input。但是当启动命令python train.py后面有其他命令时，如下所示：
	`python train.py`
	`pwd  #反例，启动命令的最后一条命令不是运行训练脚本，而是pwd`
	此时，如果拼接了输入管道、输出管道、以及超参，系统运行实际执行的是python train.py pwd --data_url=/input，就会报错。
	说明
	为保证数据安全，请勿输入敏感信息，例如明文密码。

参数名称	说明
运行用户ID	容器运行时的用户ID。当“启动方式”选择“预置框架”时不涉及该参数；当“启动方式”选择“自定义”时，该参数可选。 如果需要指定uid，则uid数值需要在规定范围内，不同资源池的uid范围如下： ● 公共资源池：1000-65535 ● 专属资源池：0-65535 建议使用默认值1000。 如果运行用户ID配置为0，则训练容器中的运行用户是**root**。
本地代码目录	仅当“代码来源”选择“对象存储服务OBS”时才显示该参数。可选参数。 用于指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。 本地代码目录不支持在/home/ma-user/modelarts/*，/home/ma-user/modelarts-dev/*，/home/ma-user/infer/*目录下，也不支持配置成/home/ma-user。 单击页面右上角的“运行环境预览”，可以查看训练作业的工作目录。
容器运行目录	指定训练容器的本地目录，训练时系统会自动cd到此目录下执行启动文件。 容器运行目录是训练作业运行过程中执行启动命令的本地目录，可用于存储生成的一些临时文件。为确保该目录存在，容器运行目录必须是启动文件本地目录的父目录。

- **使用我的算法创建**：使用算法管理中的算法创建训练作业。

  如果选择使用已有算法创建训练作业，则“创建方式”选择“我的算法”，在算法列表中选择算法。如果没有满足条件的算法，也可以新建算法，具体操作请参见创建算法。

- **使用订阅算法创建**：使用AI Gallery订阅的算法创建训练作业。

  如果选择使用订阅算法创建训练作业，则“创建方式”选择“我的订阅”，在算法列表中选择算法。如果没有满足条件的算法，也可以前往AI Gallery中订阅算法。

  a. 登录AI Gallery。
  b. 选择“模型”，进入模型列表页面，该页面展示了所有共享的模型，包括ModelArts算法。
  c. 单击目标算法进入详情页面。
  d. 在详情页面选择“训练 > ModelArts”，在弹出的“选择ModelArts的云服务区域”页面选择需要使用算法的区域，单击“确认”跳转至ModelArts控制台的“算法管理 > 订阅算法”页面。

     订阅后的算法，将自动展现在“算法管理 > 订阅算法”页面中。

步骤四：训练配置（可选）

创建训练作业时，需要配置训练作业的输入、输出、超参和环境变量。

表 6-13 训练配置

参数名称	说明
输入	单击“添加”，配置训练作业输入信息。 ● “参数名称” 算法代码需要通过“输入”的“参数名称”去读取训练的输入数据。 建议设置为“data_url”。训练输入参数要与所选算法的“输入”参数匹配。 ● “输入来源”：训练输入支持来源OBS存储和数据集。 – 单击“输入来源”参数右侧的 ，从OBS桶中选择训练输入数据的存储位置。文件总大小要小于或等于10GB，文件数要小于或等于1000个，单个文件大小要小于或等于1GB。 – 单击“数据集”，在ModelArts数据集列表中勾选目标数据集并选择对应的版本。 训练启动时，系统将自动下载输入路径中的数据到训练运行容器。 ● “获取方式” 以参数名称为“data_path”的训练输入为例，说明获取方式的作用。 – 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 import argparse parser = argparse.ArgumentParser() parser.add_argument('--data_path') args, unknown = parser.parse_known_args() data_path = args.data_path – 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 import os data_path = os.getenv("data_path", "") ● “容器内访问路径”：数据输入通道映射的容器本地路径。 – “输入”的本地路径不支持修改，此路径由系统自动设置。 – 值为参数名称、本地代码路径、索引和输入输出类型拼接而成，规则为：本地代码路径 + 输入输出（input / output） + 参数名称 + ‘_’ + 索引。 **图 6-5** 示例

参数名称	说明
输出	单击“添加”，配置训练作业输出信息。 ● “参数名称” 算法代码需要通过“输出”的“参数名称”去读取训练的输出目录。 建议设置为“train_url”。训练输出参数要与所选算法的“输出”参数匹配。 ● “输出路径” 单击“输出路径”参数右侧的 🗀，从OBS桶中选择训练输出数据的存储位置。文件总大小必须小于或等于1GB，文件数必须小于或等于128个，单个文件大小必须小于或等于128MB。 训练过程中，系统将自动从训练容器的本地代码目录下同步文件到数据存储位置。 输出数据的存储位置仅支持OBS存储。为避免和训练输入数据存储冲突，建议选择一个空目录用作输出数据存储位置。 ● “获取方式” 以参数名称为“train_url”的训练输出为例，说明获取方式的作用。 – 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 import argparse parser = argparse.ArgumentParser() parser.add_argument('--train_url') args, unknown = parser.parse_known_args() train_url = args.train_url – 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 import os train_url = os.getenv("train_url", "") ● “容器内访问路径”：数据输出通道映射的容器本地路径。 – “输出”的本地路径不支持修改，此路径由系统自动设置。 – 值为参数名称、本地代码路径、索引和输入输出类型拼接而成，规则为：本地代码路径 + 输入输出（input / output） + 参数名称 + ‘_’ + 索引。 ● “预下载至容器目录” 选择是否将输出目录下的文件预下载至训练容器中。 – 否：表示启动训练作业时不会将输出数据存储位置中的文件下载到训练容器的本地代码目录中。 – 是：表示系统会在启动训练作业时自动将输出数据存储位置中的所有文件下载到训练容器的本地代码目录中。下载时间会随着文件变大而变长，为了防止训练时间过长，请及时清理无用文件。**增量模型训练**或**设置断点续训练**场景下，必须选择“是”。

参数名称	说明
超参	超参用于训练调优。此参数由选择的算法决定，如果在算法中已经定义了超参，则此处会显示算法中所有的超参。 超参支持修改和删除，状态取决于算法中的超参“约束”设置，详情请参见表5-6。 ● 单击“添加”，添加超参信息，总数不超过100条。 ● 单击“本地上传”可以本地批量导入超参，需要按模板填写超参且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
环境变量	根据业务需求增加环境变量。训练容器中预置的环境变量请参见管理训练容器环境变量。 ● 单击“添加”，添加环境变量信息，总数不超过100条。 ● 单击“本地上传”可以本地批量导入环境变量，需要按模板填写环境变量且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
超参自动搜索	当训练作业的创建方式选择“我的算法”且所选择的算法“支持的策略”为“自动搜索（S）”时，单击“更多配置”会显示“超参自动搜索”参数。 勾选此参数后，训练作业运行时会开启自动超参搜索能力，可能会导致训练时长延长。 具体介绍请参见自动模型优化介绍。

步骤五：资源配置

创建训练作业时，需要选择训练资源。资源池分为公共资源池与专属资源池。根据需要选择不同的资源池用于训练作业，推荐使用专属资源池，两者的具体差异说明请参见专属资源池和公共资源池的能力差异。

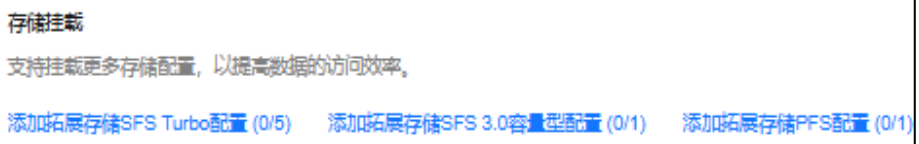
表 6-14 资源配置参数说明

参数名称	说明
资源池类型	● “公共资源池”：公共资源池由系统提供，供平台所有租户共享使用，不需要用户创建。 ● “专属资源池”：专属资源池需要提前单独创建，不与其他租户共享，具体创建操作请参见创建专属资源池。

参数名称	说明
资源池	仅选择“专属资源池”后出现此参数，在“资源池”中单击“选择资源池”，在右侧弹窗中选择要使用的专属资源池或者逻辑子池后，单击“确定”。 选择界面支持查看当前专属资源池的名称、节点池规格、可用节点数/最大节点数、可用NPU/GPU设备数、可用CPU（vCPUs）、可用内存（GiB）以及资源碎片。单击资源碎片列的“查看”可以查看碎片详情，确认资源池是否满足训练需求。 选择完成后，此处会展示所选资源池信息，也可单击“重新选择”再次选择资源池。
规格类型	当选择“专属资源池”时，会显示此参数。支持以下2种规格类型： ● “预置规格”：选择该专属资源池中预置的“实例规格”。请确保已选择的规格有足够的磁盘空间下载输入文件。 ● “自定义规格”：训练作业支持基于专属资源池的规格自定义资源规格，可自行选择CPU (核)、Memory (GB)、Ascend (卡)和实例数，进而提升资源池的利用率。自定义规格只能小于或等于专属资源池的“节点规格”。 ● CPU规格只支持自定义核数和内存，GPU和Ascend规格支持自定义核数、内存和卡数。 **图 6-6** 实例规格

参数名称	说明
实例规格	资源池类型为“专属资源池”时，需要先选定“资源池”且规格类型选择“预置规格”后方可选择，资源池类型为“公共资源池”时直接单击，然后在右侧弹窗中选择实例规格后，单击“确定”。 ● 最上方显示了CPU、NPU和GPU规格类型，下方表格可查看规格名称、内存、规格参考价等详情。 ● 规格名称中体现了节点规格的卡数、卡型号、CPU/GPU以及内存大小等信息。 ● 如果部分资源类型不可见或不可选，表示不支持。 ● 当配置了“输入”参数时，在选择资源池“实例规格”时可以单击下方的“获取实际可用规格预估”，检查输入数据的大小是否超出数据盘的容量限制，避免训练过程中出现内存不足的情况。 **说明** ● 实例规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。 ● 资源池类型为“公共资源池”时：使用自定义镜像（创建方式为自定义算法，启动方式为自定义）创建作业时，请确保您的“镜像”类型、“资源类型”和“实例规格”类型保持一致（例如：均为GPU类型）。否则会导致训练作业失败。
实例数	必填，根据需要选择实例数的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数 > 1”时，创建的是分布式训练作业，更多分布式训练配置请参见分布式训练功能介绍。 **如果用户开启了对应集群的高可用节点特性，资源池可供训练作业调度的总节点数为实际可用节点数与高可用节点数之和。高可用节点由系统预留，用户作业无法直接调度，仅在节点发生异常时用于故障恢复。** 在创建分布式训练作业前，建议提前安装所有的pip依赖（请参见在镜像中安装pip依赖），因为当节点数大于10时，系统会自动删除pip源配置，当训练过程中涉及pip install操作时可能会训练失败。

参数名称	说明
指定亲和节点	仅专属资源池支持配置。为训练作业配置超节点和节点亲和性。勾选后开启。 开启后，支持精细控制 Pod 的部署策略：严格部署（强亲和）、尽量部署（弱亲和）、禁止部署（强反亲和）、避免部署（弱反亲和）。 **亲和类型：** 节点亲和：要求训练作业的所有实例必须或者尽量调度到选择的节点上。 节点反亲和：要求训练作业的所有实例必须或者尽量不调度到选择的节点上。 **强度：**亲和强弱。 弱：尽量将 Pod 调度到指定节点，不保证成功。 强：严格将 Pod 调度到指定节点，否则不执行调度。 **超节点亲和方式：**仅超节点资源池支持。 超节点层级，当前仅支持训练作业的所有实例在1个亲和组内（即超节点亲和组实例数=实例数）的场景，适用于希望业务流量不跨超节点的训练。 随机子节点：系统随机调度至目标超节点内的子节点。 指定子节点：按指定节点调度。 **选择超节点：**选择需要配置的超节点。仅超节点资源池支持。 **选择节点：**选择需要配置的节点。

参数名称	说明
挂载存储	资源规格选择“专属资源池”时，支持添加多种存储挂载方式，以提高数据的访问效率。 **图 6-7** 存储挂载  ● “添加拓展存储SFS Turbo配置”： 当ModelArts和SFS Turbo间网络直通时，训练作业支持挂载多个SFS Turbo存放训练数据。文件系统支持重复挂载，但是挂载路径不可重复。最多可以挂载5个盘。配置ModelArts和SFS Turbo间网络直通，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。 **图 6-8** SFS Turbo – “文件系统”：选择一个SFS Turbo。 – “云上挂载路径”：输入SFS Turbo对应在训练容器内的云上挂载路径。不能为“/”目录，不能为“/cache”、“/home/ma-user/modelarts”等系统已经默认挂载的路径。 – “存储位置”：选择SFS Turbo的存储位置。如果用户配置了文件夹控制权限，请选择存储位置；如果用户未配置文件夹控制权限，可以保持默认值“/”或者自定义位置。 – “挂载方式”：显示挂载SFS Turbo的权限。根据SFS Turbo存储位置的权限显示“读写”或“只读”，如果用户未配置文件夹控制权限，则该参数不可见。如果需要设置SFS Turbo的文件夹权限，请参考**权限管理**文档配置。 – “挂载参数”：支持配置SFS挂载参数实现训练加速优化，具体参数说明请参见**设置极速文件存储挂载参数**。不设置时，默认配置如下参数： mountOptions: - vers=3 - timeo=600 - nolock - hard

参数名称	说明
	说明 当用户配置拓展存储SFS Turbo时，前端页面会自动从**SFS Turbo**服务获取最新的存储容量和配置信息，以确保数据有效性和准确性。 1. 查询文件系统详细信息。 2. 获取文件系统列表。 ● “添加拓展存储SFS 3.0容量型配置”：训练作业支持挂载一个通用文件系统SFS 3.0用于存放训练数据。需要配置如下参数。 **图 6-9** SFS 3.0 容量型 – “文件系统”：输入通用文件系统名称。需要和弹性文件服务SFS里面的名称一致，否则会挂载失败。 – “云上挂载路径”：输入对应在训练容器内的云上挂载路径。 **说明** “添加拓展存储SFS 3.0容量型配置”为受限使用阶段，需要提工单申请开通。 ● “添加拓展存储PFS配置”：训练作业支持挂载一个OBS的并行文件系统用于存放训练数据。需要配置如下参数。 **图 6-10** PFS – “存储位置”：选择一个并行文件系统。 – “云上挂载路径”：输入对应在训练容器内的云上挂载路径。 **说明** “添加拓展存储PFS配置”为受限使用阶段，需要提工单申请开通。
超节点亲和组实例数	● “资源池”选择“专属资源池”，且“节点规格”为Snt9b23系列情况下，支持配置超节点亲和组实例数。 ● 当选择超节点资源池时，支持配置超节点亲和组实例数。如果配置超节点亲和组实例数为n，则每n个Pod会被调度到同一个超节点内，从而实现亲和性作业调度。在分布式训练中亲和性作业调度可以保证计算资源的同构性。 ● 如果配置了“超节点亲和组实例数”，则“实例数”必须设置为超节点亲和组实例数的整数倍，否则无法创建训练作业。例如，超节点亲和组实例数为n，则实例数为n或2n。 ● 超节点亲和组实例配置更多内容请参见超节点亲和组实例数配置章节。

参数名称	说明
模式选择	当训练作业的“预置框架”选择的是MindSpore类镜像、“实例规格”选择的是Ascend资源时，ModelArts提供多种训练模式，支持根据实际训练场景选择模式。 ● 普通模式：默认训练场景。 ● 高性能模式：将关闭或调整部分运维功能，以最大程度提升运行速度，此模式下故障定位能力将会下降，适合于网络稳定并追求高性能的场景。 ● 故障诊断模式：将开启或调整部分运维功能，以收集更多的信息用于故障定位，适合于执行出现问题需要收集故障信息进行定位的场景。

步骤六：高可用配置

创建训练作业时，支持设置训练作业自动重启。

表 6-15 高可用配置

参数名称	说明
自动重启	设置训练作业是否设置自动重启。 ● 不勾选（默认）：表示不启用自动重启，训练异常则直接停止作业。 ● 勾选：表示当由于环境问题、作业卡死等原因导致训练作业异常时，系统将自动检测故障问题并根据相应策略进行处理，以提高训练成功率。训练作业恢复策略已支持进程级、容器级、作业级自动重启，无需配置，自动匹配与升级恢复策略。 为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见**设置断点续训练**。 当训练过程中触发了自动重启，则系统会记录重启信息，在训练作业详情页可以查看故障恢复详情，具体请参见**训练作业容错检查**。
最大重启次数	当勾选“自动重启”时可配置。 当训练作业的自动重启次数达到最大重启次数，且训练作业依旧异常，则直接停止作业。 ● 缺省值：3 ● 取值范围：1~128 提交创建训练作业后不支持修改重启次数，请合理设置次数。
无条件自动重启	当勾选“自动重启”时可配置。勾选无条件自动重启后，只要系统检测到训练异常，就无条件重启训练作业。为了避免无效重启浪费算力资源，系统最多只支持连续无条件重启3次。

参数名称	说明
作业卡死重启	当勾选“自动重启”时可配置。系统支持自动监控作业进程的状态和资源利用率来判定作业是否卡死，开启此开关后，支持将标记为卡死的作业进行进程级自动重启，以提高资源使用率。 CPU规格不支持作业卡死重启。 因系统无法核实代码逻辑且检测存在周期性，卡死检测存在一定的误报概率，开启开关即表示接受误报率。为了避免无效重启浪费算力资源，系统最多只支持连续作业卡死重启3次。
算子重执行	默认不展示，需要符合一定的条件才会展示“算子重执行”开关。 开启算子重执行，可以解决当超节点网络链路故障时导致的训练作业通信异常的问题。 开启对训练任务的性能有较大影响。 详细介绍和约束条件请参见**配置算子重执行**章节。
性能监控与诊断	当训练作业选择自定义镜像时，可勾选该项并配置弹性文件服务 SFS Turbo，开启后，针对训练中耗时超预期、资源利用率失衡等性能劣化问题，可实时观测 step 耗时曲线；当性能劣化超限时，支持自动或手动采集 profiling 数据，采集完成后通过可视化工具 MindStudio Insight 提供可视化分析结果。

步骤七：调度配置

创建训练作业时，支持配置作业调度策略，例如提高作业调度优先级、允许被抢占、自动停止等，用于提高作业调度效率。

表 6-16 调度配置

参数名称	说明
作业优先级	• 使用专属资源池创建训练作业时，支持使用和修改训练作业的调度优先级。使用公共资源池时，不支持此参数配置。 • 作业默认按照优先级从高到低依次处理，对于相同优先级的任务，则根据提交时间的先后顺序进行调度，最终结合实际的资源情况决定调度顺序(保证在资源充足、相同优先级下，先提交的任务先调度)。 • 修改选择数字后可修改作业进入队列的优先级。优先级可以设置数字1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。 • 设置优先级为“3”时，还需要配置用户权限才会生效。用户权限设置请参见“**设置作业为高优先级权限**”。 • 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。

参数名称	说明
允许被抢占	● 使用专属资源池创建训练作业时，允许设置训练作业被抢占。使用公共资源池时，不支持此参数配置。 ● 开启后，当资源池剩余资源不足时，允许被抢占的作业可能会被终止并重新排队。为避免训练进度丢失，请在开启此功能前完成断点续训的设置，操作指导请参考设置断点续训练。
自动停止	选择是否启用训练作业自动停止。 ● 不勾选（默认）：表示训练作业将一直运行直至训练完成。 ● 勾选：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。

步骤八：更多配置

表 6-17 高级配置

参数名称	说明
永久保存日志	当实例规格选择Ascend资源时，默认开启“永久保存日志”。 当实例规格选择CPU或GPU资源时，支持选择是否打开“永久保存日志”开关。 ● 勾选（默认）：表示永久保存日志，此时必须配置“日志路径”，平台会将训练日志永久保存至指定的OBS路径。 ● 不勾选：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
日志路径	当开启“永久保存日志”时，必须配置“日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。
作业可见范围	分为“工作空间内可见”和“仅创建者可见”。 ● “工作空间内可见”：创建的训练作业在当前工作空间内，默认其他子用户均可看到该训练作业。 ● “仅创建者可见”：默认创建者可见，其他人希望查看该作业需要额外申请“modelarts:trainJob:listAll”权限（查看训练作业列表（包含仅创建者可见的作业）权限）。

参数名称	说明
事件通知	选择是否启用训练作业的事件通知。 ● 不勾选（默认）：表示不启用消息通知服务。 ● 勾选：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题，并为主题添加订阅。当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”“”、“作业终止”“”、“作业疑似卡死”“”、“作业重启”、“作业抢占”等。 **说明** ● 使用消息通知服务会产生相关服务费用，详细信息请参见**消息通知服务计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
生成实例间SSH免密互信文件	选择是否生成实例间SSH免密互信文件。 ● 不勾选（默认）：表示不生成实例间SSH免密互信文件。 ● 勾选：表示生成实例间SSH免密互信文件。在使用基于MPI和Horovod框架的自定义镜像进行分布式训练时，为确保实例间的通信顺畅，必须配置实例间SSH免密互信，确保分布式训练任务顺利进行。 同时还需配置“SSH免密互信文件目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。更多操作指导请参见**自定义镜像训练作业配置实例间SSH免密互信**。
标签	当需要使用同一标签标识多种云资源时，即所有服务均可在标签输入框下拉选择同一标签，则建议在TMS中创建预定义标签。标签详细用法请参见**使用TMS标签实现资源分组管理**。 最多支持添加20条标签。

步骤九：提交训练作业并查看作业状态

当参数配置完成后，单击“提交”，创建训练作业任务。

训练作业一般需要运行一段时间，前往训练作业列表，可以查看训练作业的基本情况。

- 在训练作业列表中，刚创建的训练作业状态为“等待中”。
- 当训练作业的状态变为“已完成”时，表示训练作业运行结束，其生成的模型将存储至对应的“输出”目录中。
- 当训练作业的状态变为“运行失败”或“异常”时，可以单击训练作业的名称进入详情页面，通过查看日志等手段处理问题。

常见问题

1. 在ModelArts中训练好后的模型如何获取?
2. 在ModelArts上训练模型如何配置输入输出数据?
3. 在ModelArts上如何提升训练效率并减少与OBS的交互?
4. 如何查看ModelArts训练作业资源占用情况?

7 分布式模型训练

7.1 分布式训练功能介绍

分布式训练

分布式训练是指在多个计算节点（如多台服务器或GPU设备）上并行执行深度学习任务，以加快模型训练速度或处理更大规模的数据。通过将训练任务分配到多个节点上，每个节点负责计算模型的一部分，然后通过通信机制将计算结果同步，最终完成整个模型的训练。这种方式可以显著提高训练效率，尤其适用于复杂模型和大规模数据集的场景。

ModelArts 提供了对分布式训练的支持，能够自动配置和管理多节点间的通信与资源分配，从而实现高效的并行计算。

ModelArts提供了如下能力：

- 丰富的官方预置镜像，满足用户的需求。
- 支持基于预置镜像自定义制作专属开发环境，并保存使用。
- 丰富的教程，帮助用户快速适配分布式训练，使用分布式训练极大减少训练时间。
- 分布式训练调测的能力，可在PyCharm/VSCode/JupyterLab等开发工具中调试分布式训练。

它支持两种主要模式：

1. 单机多卡数据并行（DataParallel, DP）：在同一台服务器上使用多张GPU卡，通过数据并行的方式加速训练。适用于需要充分利用单台服务器多GPU资源的场景。
2. 多机多卡数据并行（DistributedDataParallel, DDP）：通过多台服务器协同工作，每台服务器使用多张GPU卡，进一步扩展训练规模，适用于需要处理大规模数据或复杂模型的场景。

约束限制

- 总览页面打开的CodeLab不支持此项功能，但是如果用户在AI Gallery中打开了可用的案例，会自动跳转到CodeLab中，此时是可以使用这项功能的。

- 如果切换了Notebook的规格，那么只能在Notebook进行单机调测，不能进行分布式调测，也不能提交远程训练作业。
- 当前仅支持PyTorch和MindSpore AI框架，如果MindSpore要进行多机分布式训练调试，则每台机器上都必须有8张卡。
- 本文档提供的调测代码中涉及到的OBS路径，请用户替换为自己的实际OBS路径。
- 本文档提供的调测代码是以PyTorch为例编写的，不同的AI框架之间，整体流程是完全相同的，只需要修改个别的参数即可。

计费影响

在ModelArts进行模型训练时，会产生计算资源和存储资源的累计值计费。计算资源为训练作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

DataParallel 进行单机多卡训练的优缺点

- 代码简单：仅需修改一行代码。
- 通信瓶颈 ： 负责reducer的GPU更新模型参数后分发到不同的GPU，因此有较大的通信开销。
- GPU负载不均衡：负责reducer的GPU需要负责汇总输出、计算损失和更新权重，因此显存和使用率相比其他GPU都会更高。

DistributedDataParallel 进行多机多卡训练的优缺点

- **通信更快**：相比于DP，通信速度更快
- **负载相对均衡**：相比于DP，GPU负载相对更均衡
- **运行速度快**：因为通信时间更短，效率更高，能更快速地完成训练作业

相关章节

- **创建单机多卡的分布式训练（DataParallel）**：介绍单机多卡数据并行分布式训练原理和代码改造点。
- **创建多机多卡的分布式训练（DistributedDataParallel）**：介绍多机多卡数据并行分布式训练原理和代码改造点。
- **示例：创建DDP分布式训练（PyTorch+GPU）**：提供了分布式训练调测具体的代码适配操作过程和代码示例。
- **示例：创建DDP分布式训练（PyTorch+NPU）**：针对Resnet18在cifar10数据集上的分类任务，给出了分布式训练改造(DDP)的完整代码示例，供用户学习参考。
- **基于开发环境使用SDK调测训练作业**：介绍如何在ModelArts的开发环境中，使用SDK调测单机和多机分布式训练作业。

7.2 创建单机多卡的分布式训练（DataParallel）

在深度学习领域，随着模型规模的不断扩大，训练时间也随之增加。为了提高训练效率，需要采用高效的并行计算方法。在单机环境下，如何充分利用多块GPU卡的计算能力成为一个关键问题。本章节将介绍基于PyTorch引擎的单机多卡数据并行训练方

法，通过合理的数据划分和模型同步策略，充分发挥多GPU的计算能力，显著提升训练效率。

MindSpore引擎的分布式训练参见**MindSpore官网**，可在左上角自行选择相应的版本进行查看。

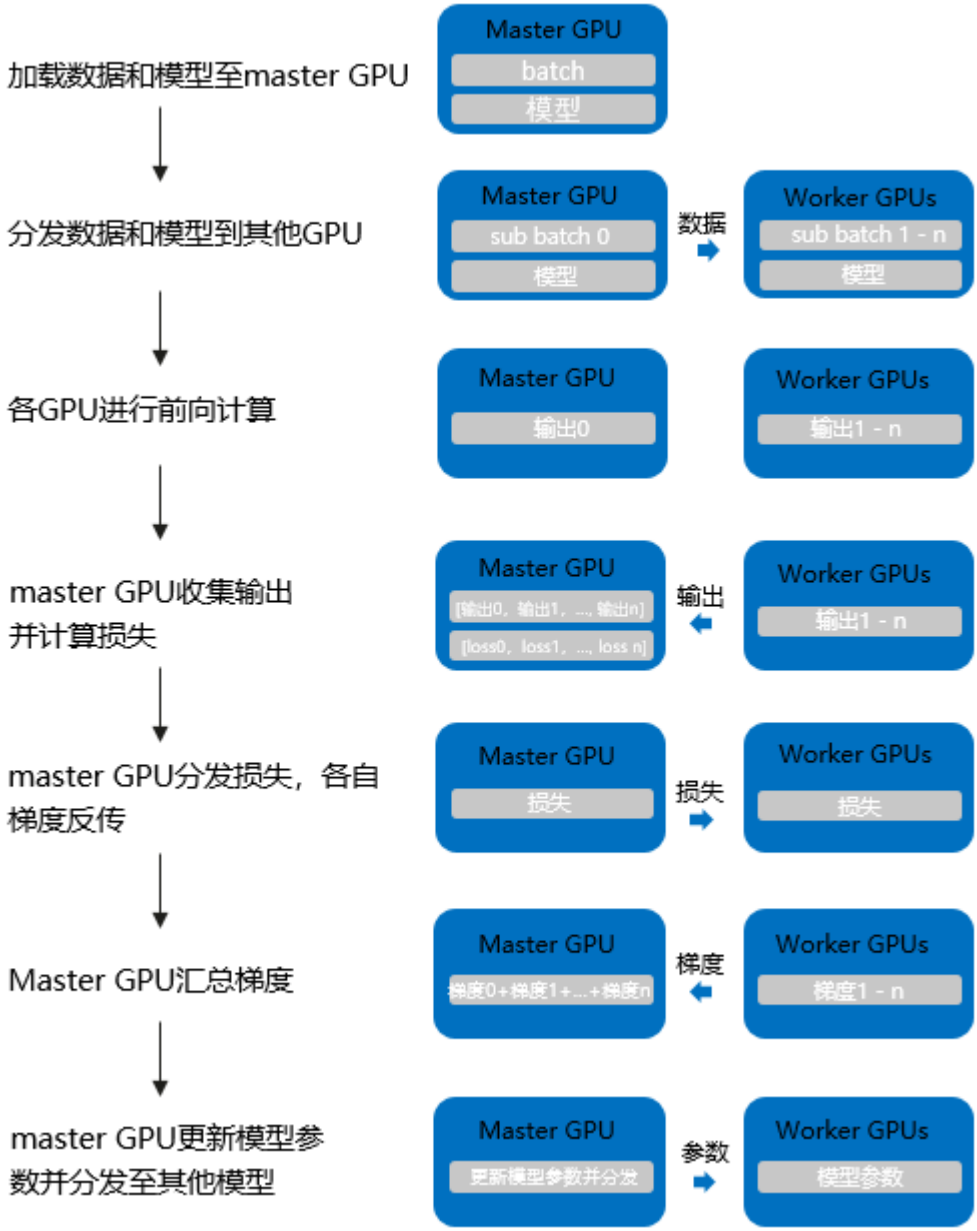
训练流程简述

单机多卡数据并行训练流程介绍如下：

1. 将模型复制到多个GPU上
2. 将一个Batch的数据均分到每一个GPU上
3. 各GPU上的模型进行前向传播，得到输出
4. 主GPU（逻辑序号为0）收集各GPU的输出，汇总后计算损失
5. 分发损失，各GPU各自反向传播梯度
6. 主GPU收集梯度并更新参数，将更新后的模型参数分发到各GPU

具体流程图如下：

图 7-1 单机多卡数据并行训练



代码改造点

模型分发：DataParallel(model)

完整代码由于代码变动较少，此处进行简略介绍。

```
import torch
class Net(torch.nn.Module):
    pass

model = Net().cuda()

### DataParallel Begin ###
model = torch.nn.DataParallel(Net().cuda())
### DataParallel End ###
```

7.3 创建多机多卡的分布式训练（DistributedDataParallel）

在深度学习领域，随着模型复杂度的不断提升和训练规模的不断扩大，传统的单机训练方式已难以满足实际需求。在这一背景下，如何在多机多卡环境下实现高效的分布式训练成为了亟待解决的关键问题。

针对这一问题，本章节将详细介绍基于PyTorch引擎的多机多卡数据并行训练方法。本文不仅提供具体的代码适配操作过程，还通过完整的代码示例帮助读者更好地理解和实践。特别地，本文以ResNet18在CIFAR10数据集上的图像分类任务为例，展示了如何进行分布式训练改造（DDP），为读者提供可直接参考和复现的实践方案。

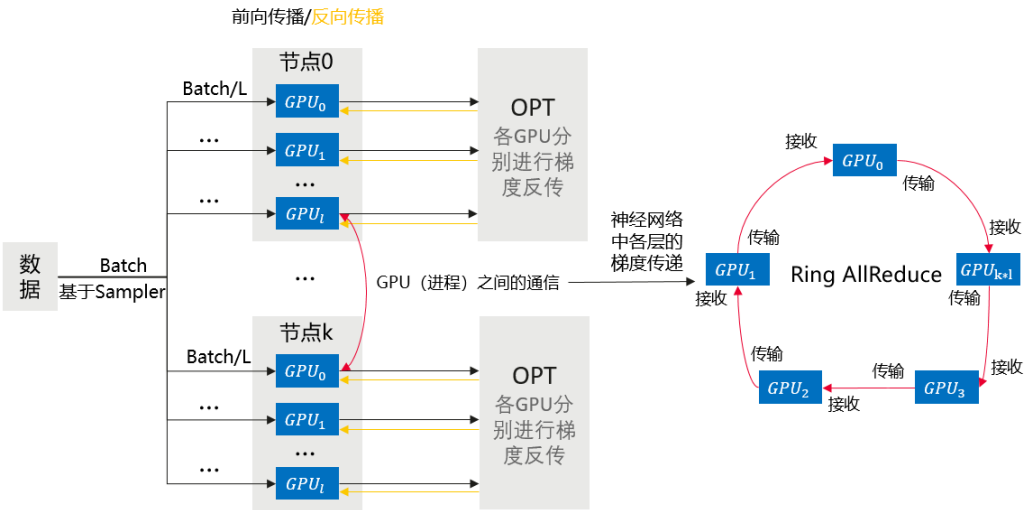
训练流程简述

相比于DP，DDP能够启动多进程进行运算，从而大幅度提升计算资源的利用率。可以基于torch.distributed实现真正的分布式计算，具体的原理此处不再赘述。大致的流程如下：

1. 初始化进程组。
2. 创建分布式并行模型，每个进程都会有相同的模型和参数。
3. 创建数据分发Sampler，使每个进程加载一个mini batch中不同部分的数据。
4. 网络中相邻参数分桶，一般为神经网络模型中需要进行参数更新的每一层网络。
5. 每个进程前向传播并各自计算梯度。
6. 模型某一层的参数得到梯度后会马上进行通讯并进行梯度平均。
7. 各GPU更新模型参数。

具体流程图如下：

图 7-2 多机多卡数据并行训练



代码改造点

- 引入多进程启动机制：初始化进程
- 引入几个变量：tcp协议，rank进程序号，worldsize开启的进程数量
- 分发数据：DataLoader中多了一个Sampler参数，避免不同进程数据重复
- 模型分发：DistributedDataParallel(model)
- 模型保存：在序号为0的进程下保存模型

```
import torch
class Net(torch.nn.Module):
    pass

model = Net().cuda()

### DistributedDataParallel Begin ###
model = torch.nn.parallel.DistributedDataParallel(Net().cuda())
### DistributedDataParallel End ###
```

多节点分布式调测适配及代码示例

在分布式训练中，DistributedDataParallel（DDP）是一种常用的数据并行训练方法。在这一机制下，不同进程会分别从原始数据中加载batch数据，最终通过将各个进程计算得到的梯度进行平均，得到最终的梯度值。由于DDP能够利用更多的样本量进行计算，因此其计算出的梯度更加可靠，这也意味着在分布式训练中可以适当增大学习率以加快收敛速度。

为了帮助读者更好地理解和实践，我们以ResNet18在CIFAR10数据集上的图像分类任务为例，提供了完整的单机训练和分布式训练改造（DDP）代码示例。该代码支持多节点分布式训练，同时兼容CPU和GPU分布式训练环境。值得注意的是，用户可以通过注释掉代码中的分布式改造点，轻松切换为单节点单卡训练模式。

在训练代码中，我们设计了三个主要参数输入模块：训练基础参数、分布式参数和数据相关参数。其中，分布式参数由平台自动注入，无需用户手动定义。在数据相关参数中，我们特别设计了一个custom_data开关，用于控制是否使用自定义数据进行训练。当custom_data设置为"true"时，系统将基于PyTorch生成随机数据进行训练和验证，这为用户提供了灵活的实验环境。

cifar10数据集

在Notebook中，无法直接使用默认版本的torchvision获取数据集，因此示例代码中提供了三种训练数据加载方式。

cifar-10数据集下载链接，单击“CIFAR-10 python version”。

- 尝试基于torchvision获取cifar10数据集。
- 基于数据链接下载数据并解压，放置在指定目录下，训练集和测试集的大小分别为(50000，3，32，32)和(10000，3，32，32)。
- 考虑到下载cifar10数据集较慢，基于torch生成类似cifar10的随机数据集，训练集和测试集的大小分别为(5000，3，32，32)和(1000，3，32，32)，标签仍为10类，指定custom_data = 'true'后可直接进行训练作业，无需加载数据。

训练代码

以下代码中以“### 分布式改造，... ###”注释的代码即为多节点分布式训练需要适配的代码改造点。

不对示例代码进行任何修改，适配数据路径后即可在ModelArts上完成多节点分布式训练。

注释掉分布式代码改造点，即可完成单节点单卡训练。完整代码见**分布式训练完整代码示例**。

- **导入依赖包**

```
import datetime
import inspect
import os
import pickle
import random

import argparse
import numpy as np
import torch
import torch.distributed as dist
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler
from sklearn.metrics import accuracy_score
```

- **定义加载数据的方法和随机数**，由于加载数据部分代码较多，此处省略

```
def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True

def get_data(path):
    pass
```

- **定义网络结构**

```
class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)


class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
```

```
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out
```

- **进行训练和验证**

```
def main():
    file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)

    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()

    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        print('[warning] you are training on custom random dataset, '
              'validation accuracy may range from 0.4 to 0.6.')

    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
rank=args.rank)
    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch, shuffle=False)

    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
```

```
    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###

    val_loader = DataLoader(val_set, batch_size=batch, shuffle=False)

    lr = args.lr * gpus_per_node
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### 分布式改造，构建DDP分布式模型 ###
    model = nn.parallel.DistributedDataParallel(model)
    ### 分布式改造，构建DDP分布式模型 ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###
        tr_sampler.set_epoch(epoch)
        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

        val_loss = 0
        pred_record = []
        real_record = []
        model.eval()
        with torch.no_grad():
            for step, (val_x, val_y) in enumerate(val_loader):
                if args.enable_gpu:
                    val_x, val_y = val_x.cuda(), val_y.cuda()
                out = model(val_x)
                pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
                real_record += list(val_y.cpu().numpy())
                val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
        print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu),
'\n')

        if args.rank == 0:
            # save ckpt every epoch
            torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))


if __name__ == '__main__':
    main()
```

- **结果对比**

  分别以单机单卡和两节点16卡两种资源类型完成100epoch的cifar-10数据集训练，训练时长和测试集准确率如下。

表 7-1 训练结果对比

资源类型	单机单卡	两节点16卡
耗时	60分钟	20分钟
准确率	80+	80+

分布式训练完整代码示例

以下对resnet18在cifar10数据集上的分类任务，给出了分布式训练改造(DDP)的完整代码示例。

训练启动文件main.py内容如下（如果需要执行单机单卡训练作业，则将分布式改造的代码删除）：

```
import datetime
import inspect
import os
import pickle
import random
import logging

import argparse
import numpy as np
from sklearn.metrics import accuracy_score
import torch
from torch import nn, optim
import torch.distributed as dist
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler

file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)


def load_pickle_data(path):
    with open(path, 'rb') as file:
        data = pickle.load(file, encoding='bytes')
    return data


def _load_data(file_path):
    raw_data = load_pickle_data(file_path)
    labels = raw_data[b'labels']
    data = raw_data[b'data']
    filenames = raw_data[b'filenames']

    data = data.reshape(10000, 3, 32, 32) / 255
    return data, labels, filenames


def load_cifar_data(root_path):
    train_root_path = os.path.join(root_path, 'cifar-10-batches-py/data_batch_')
    train_data_record = []
    train_labels = []
    train_filenames = []
    for i in range(1, 6):
        train_file_path = train_root_path + str(i)
        data, labels, filenames = _load_data(train_file_path)
        train_data_record.append(data)
        train_labels += labels
        train_filenames += filenames
    train_data = np.concatenate(train_data_record, axis=0)
    train_labels = np.array(train_labels)
```

```
    val_file_path = os.path.join(root_path, 'cifar-10-batches-py/test_batch')
    val_data, val_labels, val_filenames = _load_data(val_file_path)
    val_labels = np.array(val_labels)

    tr_data = torch.from_numpy(train_data).float()
    tr_labels = torch.from_numpy(train_labels).long()
    val_data = torch.from_numpy(val_data).float()
    val_labels = torch.from_numpy(val_labels).long()
    return tr_data, tr_labels, val_data, val_labels


def get_data(root_path, custom_data=False):
    if custom_data:
        train_samples, test_samples, img_size = 5000, 1000, 32
        tr_label = [1] * int(train_samples / 2) + [0] * int(train_samples / 2)
        val_label = [1] * int(test_samples / 2) + [0] * int(test_samples / 2)
        random.seed(2021)
        random.shuffle(tr_label)
        random.shuffle(val_label)
        tr_data, tr_labels = torch.randn((train_samples, 3, img_size, img_size)).float(),
torch.tensor(tr_label).long()
        val_data, val_labels = torch.randn((test_samples, 3, img_size, img_size)).float(), torch.tensor(
            val_label).long()
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    elif os.path.exists(os.path.join(root_path, 'cifar-10-batches-py')):
        tr_data, tr_labels, val_data, val_labels = load_cifar_data(root_path)
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    else:
        try:
            import torchvision
            from torchvision import transforms
            tr_set = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                  download=True, transform=transforms)
            val_set = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                   download=True, transform=transforms)
            return tr_set, val_set
        except Exception as e:
            raise Exception(
                f"{e}, you can download and unzip cifar-10 dataset manually, "
                "the data url is http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz")


class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)
```

```
class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out


def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True


def obs_transfer(src_path, dst_path):
    import moxing as mox
    mox.file.copy_parallel(src_path, dst_path)
    logging.info(f"end copy data from {src_path} to {dst_path}")


def main():
    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()
```

```
    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        logging.warning('you are training on custom random dataset, '
                'validation accuracy may range from 0.4 to 0.6.')

    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
rank=args.rank)
    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch, shuffle=False)

    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###

    val_loader = DataLoader(val_set, batch_size=batch, shuffle=False)

    lr = args.lr * gpus_per_node * args.world_size
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### 分布式改造，构建DDP分布式模型 ###
    model = nn.parallel.DistributedDataParallel(model)
    ### 分布式改造，构建DDP分布式模型 ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###
        tr_sampler.set_epoch(epoch)
        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

        val_loss = 0
        pred_record = []
        real_record = []
        model.eval()
        with torch.no_grad():
            for step, (val_x, val_y) in enumerate(val_loader):
                if args.enable_gpu:
                    val_x, val_y = val_x.cuda(), val_y.cuda()
```

```
            out = model(val_x)
            pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
            real_record += list(val_y.cpu().numpy())
            val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
        print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu), '\n')

        if args.rank == 0:
            # save ckpt every epoch
            torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))


if __name__ == '__main__':
    main()
```

常见问题

1. **示例代码中如何使用不同的数据集？**
   - 上述代码如果使用cifar10数据集，则将数据集下载并解压后，上传至OBS桶中，文件目录结构如下：

     ```
     DDP
     |--- main.py
     |--- input_dir
     |------ cifar-10-batches-py
     |-------- data_batch_1
     |-------- data_batch_2
     |-------- ...
     ```

     其中“DDP”为创建训练作业时的“代码目录”，“main.py”为上文代码示例（即创建训练作业时的“启动文件”），“cifar-10-batches-py”为解压后的数据集文件夹（放在input_dir文件夹下）。
   - 如果使用自定义的随机数据，则将代码示例中的参数“custom_data”改为“true”，修改后内容如下：

     ```
     parser.add_argument('--custom_data', default='true')
     ```

     然后直接运行代码示例“main.py”即可，创建训练作业的参数与上图相同。
2. **为什么DDP可以不输入主节点ip？**

   “parser.add_argument('--init_method', default=None, help='tcp_port')”中的init method参数值会包含主节点的ip和端口，由平台自动入参，不需要用户输入主节点的ip和端口。

7.4 示例：创建 DDP 分布式训练（PyTorch+GPU）

在分布式训练场景中，使用PyTorch的DistributedDataParallel（DDP）功能是实现高效训练的重要方式。为了帮助用户更好地理解和应用这一功能，本文将详细介绍三种通过训练作业启动PyTorchDDP训练的方法，并提供对应的代码示例。

- 使用PyTorch预置框架功能，通过mp.spawn命令启动
- 使用自定义镜像功能
  - 通过torch.distributed.launch命令启动
  - 通过torch.distributed.run命令启动

  这些方法为用户提供了灵活的选择，能够根据实际需求和环境配置进行适配。

  通过本文的介绍，用户可以根据自身场景选择合适的启动方式，快速上手PyTorch的分布式训练功能。

创建训练作业

- 方式一：使用PyTorch预置框架功能，通过mp.spawn命令启动训练作业。
  创建训练作业的关键参数如表7-2所示。

  表 7-2 创建训练作业（预置框架）

参数名称	说明
创建方式	选择“自定义算法”。
启动方式	选择“预置框架”，引擎选择“PyTorch”，PyTorch版本根据训练要求选择。
代码目录	选择OBS桶中训练code文件夹所在路径，例如“obs://test-modelarts/code/”。
启动文件	选择代码目录中训练作业的Python启动脚本。例如“obs://test-modelarts/code/main.py”。
超参	当资源规格为单机多卡时，需要指定超参world_size和rank。 当资源规格为多机时（即实例数大于 1），无需设置超参world_size和rank，超参会由平台自动注入。

- 方式二：使用自定义镜像功能，通过torch.distributed.launch命令启动训练作业。
  创建训练作业的关键参数如表7-3所示。

  表 7-3 创建训练作业（自定义镜像+torch.distributed.launch 命令）

参数名称	说明
创建方式	选择“自定义算法”。
启动方式	选择“自定义”。
镜像	选择用于训练的PyTorch镜像。
代码目录	选择OBS桶中训练code文件夹所在路径，例如“obs://test-modelarts/code/”。
启动命令	输入镜像的Python启动命令，例如： `bash ${MA_JOB_DIR}/code/torchlaunch.sh`

- 方式三：使用自定义镜像功能，通过torch.distributed.run命令启动训练作业。
  创建训练作业的关键参数如表7-4所示。

表 7-4 创建训练作业（自定义镜像+torch.distributed.run 命令）

参数名称	说明
创建方式	选择“自定义算法”。
启动方式	选择“自定义”。
镜像	选择用于训练的PyTorch镜像。
代码目录	选择OBS桶中训练code文件夹所在路径，例如“obs://test-modelarts/code/”。
启动命令	输入镜像的Python启动命令，例如： `bash ${MA_JOB_DIR}/code/torchrun.sh`

代码示例

文件目录结构如下所示，将以下文件上传至OBS桶中：

```
code                        # 代码根目录
 └─torch_ddp.py               # PyTorch DDP训练代码文件
 └─main.py                   # 使用PyTorch预置框架功能，通过mp.spawn命令启动训练的启动文件
 └─torchlaunch.sh             # 使用自定义镜像功能，通过torch.distributed.launch命令启动训练的启动文件
 └─torchrun.sh               # 使用自定义镜像功能，通过torch.distributed.run命令启动训练的启动文件
```

torch_ddp.py内容如下：

```
import os
import torch
import torch.distributed as dist
import torch.nn as nn
import torch.optim as optim
from torch.nn.parallel import DistributedDataParallel as DDP

# 用于通过 mp.spawn 启动
def init_from_arg(local_rank, base_rank, world_size, init_method):
    rank = base_rank + local_rank
    dist.init_process_group("nccl", rank=rank, init_method=init_method, world_size=world_size)
    ddp_train(local_rank)

# 用于通过 torch.distributed.launch 或 torch.distributed.run 启动
def init_from_env():
    dist.init_process_group(backend='nccl', init_method='env://')
    local_rank=int(os.environ["LOCAL_RANK"])
    ddp_train(local_rank)

def cleanup():
    dist.destroy_process_group()

class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = nn.Linear(10, 10)
        self.relu = nn.ReLU()
        self.net2 = nn.Linear(10, 5)
    def forward(self, x):
        return self.net2(self.relu(self.net1(x)))

def ddp_train(device_id):
```

```
    # create model and move it to GPU with id rank
    model = ToyModel().to(device_id)
    ddp_model = DDP(model, device_ids=[device_id])
    loss_fn = nn.MSELoss()
    optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)
    optimizer.zero_grad()
    outputs = ddp_model(torch.randn(20, 10))
    labels = torch.randn(20, 5).to(device_id)
    loss_fn(outputs, labels).backward()
    optimizer.step()
    cleanup()

if __name__ == "__main__":
    init_from_env()
```

main.py内容如下：

```
import argparse
import torch
import torch.multiprocessing as mp

parser = argparse.ArgumentParser(description='ddp demo args')
parser.add_argument('--world_size', type=int, required=True)
parser.add_argument('--rank', type=int, required=True)
parser.add_argument('--init_method', type=str, required=True)
args, unknown = parser.parse_known_args()

if __name__ == "__main__":
    n_gpus = torch.cuda.device_count()
    world_size = n_gpus * args.world_size
    base_rank = n_gpus * args.rank
    # 调用 DDP 示例代码中的启动函数
    from torch_ddp import init_from_arg
    mp.spawn(init_from_arg,
        args=(base_rank, world_size, args.init_method),
        nprocs=n_gpus,
        join=True)
```

torchlaunch.sh内容如下：

```
#!/bin/bash
# 系统默认环境变量，不建议修改
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
MASTER_PORT="6060"
JOB_ID="1234"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

# 自定义环境变量，指定python脚本和参数
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

CMD="python -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr $MASTER_ADDR \
    --master_port=$MASTER_PORT \
    --use_env \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

torchrun.sh内容如下：

```
#!/bin/bash
# 系统默认环境变量，不建议修改
```

```
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
MASTER_PORT="6060"
JOB_ID="1234"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"
# 自定义环境变量，指定python脚本和参数
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""
CMD="python -m torch.distributed.run \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    $EXT_ARGS \
    --nproc_per_node=$NGPUS_PER_NODE \
    --rdzv_id=$JOB_ID \
    --rdzv_backend=static \  # PyTorch 2.1版本需要将“rdzv_backend”参数设置为“static：--
rdzv_backend=static”。
    --rdzv_endpoint=$MASTER_ADDR:$MASTER_PORT \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
    "
echo $CMD
```

7.5 示例：创建 DDP 分布式训练（PyTorch+NPU）

在深度学习领域，使用PyTorch的DistributedDataParallel（DDP）功能在Ascend加速卡上进行分布式训练是一种高效的方式。然而，如何通过自定义镜像和自定义启动命令来实现这一目标，是用户在实际操作中可能遇到的挑战。

针对这一问题，本文提供了一种解决方案：通过配置训练作业的自定义镜像，并结合自定义启动命令，用户可以轻松实现PyTorch DDP在Ascend加速卡上的训练任务。这种方法不仅能够满足用户的个性化需求，还能够灵活适配不同的训练场景。

通过本文的介绍，用户可以掌握如何利用自定义镜像和启动命令来优化PyTorch DDP训练流程，从而在Ascend加速卡上实现高效的分布式训练。

前提条件

需要有Ascend加速卡资源池。

创建训练作业

本案例创建训练作业时，需要配置如下参数。

表 7-5 创建训练作业的配置说明

参数名称	说明
“创建方式”	选择“自定义算法”。
“启动方式”	选择“自定义”。
“镜像”	选择用于训练的自定义镜像。

参数名称	说明
“代码目录”	执行本次训练作业所需的代码目录。本文示例的代码目录为“obs://test-modelarts/ascend/code/”。
“启动命令”	镜像的Python启动命令。本文示例的启动命令为“bash ${MA_JOB_DIR}/code/run_torch_ddp_npu.sh”。其中，启动脚本的完整代码请参见**代码示例**。

（可选）启用 ranktable 动态路由

如果训练作业需要使用ranktable动态路由算法进行网络加速，则可以联系技术支持开启集群的cabinet调度权限。详细使用指导请参见**训练作业动态路由加速**。

代码示例

训练作业的启动脚本示例如下。

说明

启动脚本中设置plog生成后存放在“/home/ma-user/modelarts/log/modelarts-job-{id}/worker-{index}/”目录，而“/home/ma-user/modelarts/log/”目录下的“*.log”文件将会被自动上传至ModelArts训练作业的日志目录（OBS）。如果本地相应目录没有生成大小>0的日志文件，则对应的父级目录也不会上传。因此，PyTorch NPU的plog日志是按worker存储的，而不是按rank id存储的（这是区别于MindSpore的）。目前，PyTorch NPU并不依赖rank table file。

```
#!/bin/bash

# MA preset envs
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
# also indicates NPU per node
NGPUS_PER_NODE="$MA_NUM_GPUS"

# self-define, it can be changed to >=10000 port
MASTER_PORT="38888"

# replace ${MA_JOB_DIR}/code/torch_ddp.py to the actutal training script
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

export HCCL_WHITELIST_DISABLE=1

# set npu plog env
ma_vj_name=`echo ${MA_VJ_NAME} | sed 's:ma-job:modelarts-job:g'`
task_name="worker-${VC_TASK_INDEX}"
task_plog_path=${MA_LOG_DIR}/${ma_vj_name}/${task_name}

mkdir -p ${task_plog_path}
export ASCEND_PROCESS_LOG_PATH=${task_plog_path}

echo "plog path: ${ASCEND_PROCESS_LOG_PATH}"

# set hccl timeout time in seconds
export HCCL_CONNECT_TIMEOUT=1800

# replace ${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python to the actual python
CMD="${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python -m torch.distributed.launch \
    --nnodes=$NNODES \
```

```
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr=$MASTER_ADDR \
    --master_port=$MASTER_PORT \
    --use_env \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

7.6 示例：创建 Ray 集群

用户在执行与强化学习相关的训练任务时，通常需要使用Ray集群来确保任务的顺利进行。

ModelArts环境支持运行Ray分布式作业，只需在正式训练作业前启动Ray集群即可。启动Ray集群时，需在主节点执行 **ray start --head**命令，从节点则通过执行 **ray start --address="master_ip:6379"**加入集群。ModelArts容器中提供了环境变量，用于判断当前节点是主节点还是从节点，从而执行相应的命令。以下是多节点启动Ray集群的脚本示例。

```
#!/bin/bash
pkill -9 python
ray stop --force

# 共使用多少节点训练
NNODES=${VC_WORKER_NUM:-1}
# 每个节点有多少张卡
NPUS_PER_NODE=$(lspci | grep d80 | wc -l)
MASTER_ADDR=$(python -c "import os; print(os.getenv('VC_WORKER_HOSTS','127.0.0.1').split(',')[0])")

if [ "$VC_TASK_INDEX" == "0" ]; then
# 主节点启动
    ray start --head --resources='{"NPU": '$NPUS_PER_NODE'}'

    while true; do
        ray_status_output=$(ray status)
        npu_count=$(echo "$ray_status_output" | grep -oP '(?<=/)\d+\.\d+(?=\s*NPU)' | head -n 1)
        npu_count_int=$(echo "$npu_count" | awk '{print int($1)}')
        device_count=$((npu_count_int / $NPUS_PER_NODE))

        # 判断 device_count 是否与 NNODES 相等
        if [ "$device_count" -eq "$NNODES" ]; then
            echo "Ray cluster is ready with $device_count devices (from $npu_count NPU resources), starting
Python script."
            ray status
            break
        else
            echo "Waiting for Ray to allocate $NNODES devices. Current device count: $device_count"
            sleep 5
        fi
    done
else
    # 子节点尝试往主节点注册ray直到成功
    while true; do
        # 尝试连接 Ray 集群
        ray start --address="$MASTER_ADDR:6379" --resources='{"NPU": '$NPUS_PER_NODE'}'
        # 检查连接是否成功
        ray status
        if [ $? -eq 0 ]; then
            echo "Successfully connected to the Ray cluster!"
            break
        else
            echo "Failed to connect to the Ray cluster. Retrying in 5 seconds..."
            sleep 5
```

```
        fi
    done
fi
```

将以上脚本写入文件，例如start_ray_cluster.sh，并上传至OBS。

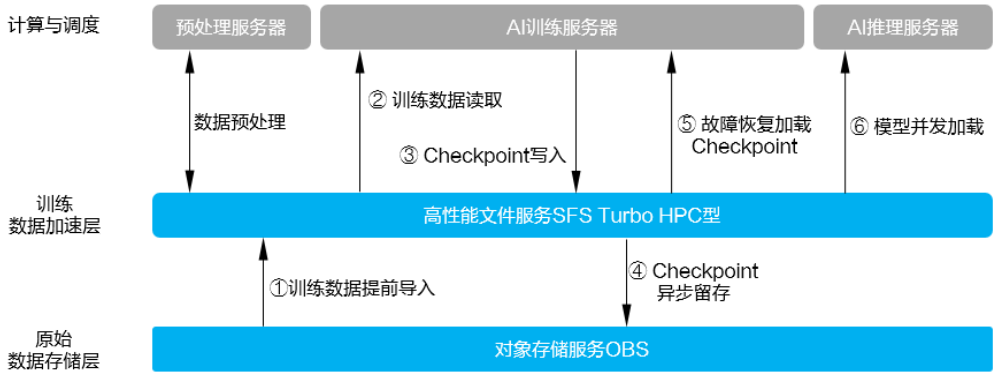
要在ModelArts中通过自定义方式启动训练任务并创建Ray集群，只需执行**bash start_ray_cluster.sh**命令，执行后可通过**ray status**命令查看Ray集群状态。

8 模型训练存储加速

针对AI训练场景中大模型Checkpoint保存和加载带来的I/O挑战，华为云提供了基于对象存储服务OBS+高性能弹性文件服务SFS Turbo的AI云存储解决方案，如下图所示。

SFS Turbo HPC型支持和OBS数据联动，您可以通过SFS Turbo HPC型文件系统来加速对OBS对象存储中的数据访问，并将生成的结果数据异步持久化到OBS对象存储中长期低成本保存。

图 8-1 基于 OBS+SFS Turbo 的存储解决方案



OBS + SFS Turbo存储加速的具体方案请参见**面向AI场景使用OBS+SFS Turbo的存储加速实践**。

当训练程序从已有Checkpoint恢复时，每张卡都需要从持久化存储中加载，在训练集群规模较大，存储带宽较低的场景下，加载耗时可能会达到小时级，严重影响训练恢复。因此，通过AITurbo SDK的快速保存和加载Checkpoint的功能，可以有效提升训练恢复速度。具体方案请参见**通过AITurbo加速保存与加载checkpoint**。

- 在保存Checkpoint的时候，利用两阶段写、内存副本、异步持久化等技术保证Checkpoint的快速、高可靠存储。
- 在加载Checkpoint的时候，利用内存快恢、Checkpoint广播等技术，大大减少后端存储的带宽压力，提升加载效率。

前提条件

- 已有可用的OBS桶。

- 已开通SFS Turbo服务。
- 由于训练作业运行需消耗资源，为了避免训练失败请确保账户未欠费。
- 确保使用的OBS目录与ModelArts在同一区域。

计费影响

在ModelArts进行模型训练时，会产生计算资源和存储资源的累计值计费。计算资源为训练作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

设置训练存储加速

当完成**上传数据至OBS并预热到SFS Turbo中**步骤后，在ModelArts中创建训练作业时，设置训练“SFS Turbo”，在“文件系统”中选择SFS Turbo实例名称，并指定“存储位置”和“云上挂载路径”。系统会在训练作业启动前，自动将存储位置中的文件目录挂载到训练容器中指定路径。

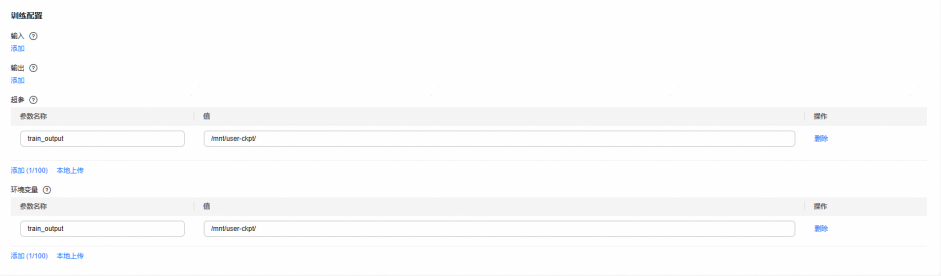
图 8-2 设置训练“SFS Turbo”

注意

当前训练作业支持挂载多个弹性文件服务SFS Turbo，文件系统支持重复挂载，但挂载路径不可重复。文件系统目录需指定已存在的目录，否则会导致训练作业异常。

然后在超参或者环境变量中设置checkpoint和数据的挂载路径。

图 8-3 在超参或者环境变量中设置 checkpoint 和数据的挂载路径



训练存储加速的代码样例

详情参见**设置断点续训练**。

9 训练作业动态路由加速

在分布式训练场景下，由于多节点之间的数据交互频繁，网络通信效率往往成为性能瓶颈。传统方式下，训练作业的网络通信存在带宽利用率低、数据传输延迟高等问题，这会直接影响训练效率和资源利用率。针对这一技术挑战，用户可能会问：如何有效优化分布式训练中的网络通信性能？ModelArts提供了一项动态路由加速功能，通过智能优化训练作业的网络通信路径，显著提升分布式训练的性能表现。本文档将详细介绍如何在ModelArts平台上启用这一功能，涵盖使用预置训练框架和自定义镜像两种典型场景，并提供详细的配置要求和实用的操作建议，帮助用户充分释放分布式训练的性能潜力。

约束限制

- 只有以下2种训练场景支持启用动态路由加速。
  - **场景一：使用Ascend-Powered-Engine预置框架+ Mindspore+NPU训练**
  - **场景二：使用自定义镜像+Pytorch+NPU训练**
- 训练作业使用的Python版本必须是3.7、3.8、3.9或3.10。
- 在启用动态路由加速之前，请联系ModelArts技术支持，确保集群的cabinet插件和调度权限已开启。

场景一：使用 Ascend-Powered-Engine 预置框架+ Mindspore+NPU 训练

当使用Ascend-Powered-Engine预置框架创建训练作业时，可以参考**表9-1**创建训练作业实现动态路由加速。表中仅提供了关键参数的配置说明，其他参数请根据业务需要配置。

表 9-1 使用预置框架创建训练作业

配置阶段	参数名称	说明
环境配置	创建方式	选择“自定义算法”。 训练作业支持使用用户自定义的算法或者AI Gallery订阅的算法创建训练作业。
	启动方式	选择“预置框架”。 创建训练作业时的环境的启动方式。

配置阶段	参数名称	说明
	引擎及版本	引擎选择“Ascend-Powered-Engine”，引擎版本选择Mindspore相关的版本。
	代码目录	选择训练代码文件所在的OBS目录。 动态路由加速通过改变rank编号来优化网络通信，因此代码中对rank的使用必须统一，避免因rank编号不一致导致通信异常。
	启动文件	选择代码目录中训练作业的Python启动脚本。
训练配置	环境变量	添加如下环境变量。 ROUTE_PLAN = true 不能配置环境变量“MA_RUN_METHOD”，确保是使用rank table file（RTF）文件启动训练作业的“启动文件”。
资源配置	资源池	选择专属资源池。
	实例规格	选择的实例规格必须满足以下条件： ● 必须是节点满卡，否则可能会影响动态路由加速的效果。例如，在节点8卡场景下，每个节点必须是8卡，不能出现2卡或4卡的情况。 ● 必须是Snt9b或Snt9b23的Ascend资源。 规格中描述了服务器类型、型号等信息。
	实例数	选择的实例数必须大于或等于3。

场景二：使用自定义镜像+Pytorch+NPU 训练

当使用自定义镜像和Ascend资源池创建训练作业时，可以参考表9-2创建训练作业实现动态路由加速。表中仅提供了关键参数的配置说明，其他参数请根据业务需要配置。

表 9-2 使用自定义镜像创建训练作业

配置阶段	参数名称	说明
环境配置	创建方式	选择“自定义算法”。
	启动方式	选择“自定义”。
	镜像	选择用于训练的自定义镜像。必须是使用Pytorch框架的训练镜像。
	代码目录（可选）	选择训练代码文件所在的OBS目录。 动态路由加速通过改变rank编号来优化网络通信，因此代码中对rank的使用必须统一，避免因rank编号不一致导致通信异常。

配置阶段	参数名称	说明
	启动命令	输入镜像的Python启动命令。 其中，要修改训练启动脚本中的如下代码，不同NPU硬件的修改值有差异。 ● Snt9b场景 MASTER_ADDR="${MA_VJ_NAME}-${MA_TASK_NAME}-${MA_MASTER_INDEX:-0}.${MA_VJ_NAME}" NODE_RANK="${RANK_AFTER_ACC:-$VC_TASK_INDEX}" ● Snt9b23场景 MASTER_ADDR="${VC_WORKER_HOSTS%%,*}" NODE_RANK="${RANK_AFTER_ACC:-$VC_TASK_INDEX}"
训练配置	环境变量	添加如下环境变量。 ROUTE_PLAN = true
资源配置	资源池	选择专属资源池。
	实例规格	选择的实例规格必须满足以下条件： ● 必须是节点满卡，否则可能会影响动态路由加速的效果。例如，在节点8卡场景下，每个节点必须是8卡，不能出现2卡或4卡的情况。 ● 必须是Snt9b或Snt9b23的Ascend资源。
	实例数	选择的实例数必须大于或等于3。

查看动态路由加速的训练日志

当使用Ascend资源池训练时，训练过程中可以在训练作业详情页的“日志”页签中查看路由状态是否启用，以及查询各个Rank的日志信息。

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型开发与训练 > 模型训练”（旧版控制台：“模型训练 > 训练作业”）。
3. 在训练作业列表中，单击目标作业，进入训练作业详情页。
4. 选择“日志”页签。

   如**图9-1**所示，可以查看训练作业已启用动态路由，还可以根据Rank-ID搜索日志。

图 9-1 查看动态路由加速日志

10 增量模型训练

什么是增量训练

增量训练（Incremental Learning）是机器学习领域中的一种训练方法，它允许人工智能（AI）模型在已经学习了一定知识的基础上，增加新的训练数据到当前训练流程中，扩展当前模型的知识和能力，而不需要从头开始。

增量训练不需要一次性存储所有的训练数据，缓解了存储资源有限的问题；另一方面，增量训练节约了重新训练中需要消耗大量算力、时间以及经济成本。

增量训练特别适用于以下情况：

- 数据流更新：在实际应用中，数据可能会持续更新，增量训练允许模型适应新的数据而不必重新训练。
- 资源限制：如果重新训练一个大型模型成本过高，增量训练可以是一个更经济的选择。
- 避免灾难性遗忘：在传统训练中，新数据可能会覆盖旧数据的知识，导致模型忘记之前学到的内容。增量训练通过保留旧知识的同时学习新知识来避免这个问题。

增量训练在很多领域都有应用，比如自然语言处理、计算机视觉和推荐系统等。它使得AI系统能够更加灵活和适应性强，更好地应对现实世界中不断变化的数据环境。

ModelArts 中如何实现增量训练

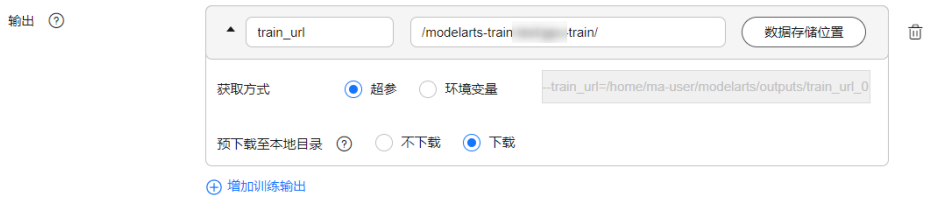
增量训练是通过Checkpoint机制实现。

Checkpoint的机制是：在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。当需要增加新的数据继续训练时，只需要加载Checkpoint，并用Checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，使能读取前一次训练保存的预训练模型。

在ModelArts训练中实现增量训练，建议使用“训练输出”功能。

在创建训练作业时，设置训练“输出”参数为“train_url”，在指定的训练输出的数据存储位置中保存**Checkpoint**，且**“预下载至本地目录”**选择“下载”。选择预下载至本地目录时，系统在训练作业启动前，自动将数据存储位置中的**Checkpoint**文件下载到训练容器的本地目录。

图 10-1 训练输出设置



PyTorch 版 reload ckpt

- PyTorch模型保存有两种方式。
  - 仅保存模型参数
    ```
    state_dict = model.state_dict()
    torch.save(state_dict, path)
    ```
  - 保存整个Model（不推荐）
    ```
    torch.save(model, path)
    ```
- 可根据step步数、时间等周期性保存模型的训练过程的产物。

  将模型训练过程中的网络权重、优化器权重、以及epoch进行保存，便于中断后继续训练恢复。

  ```
  checkpoint = {
          "net": model.state_dict(),
          "optimizer": optimizer.state_dict(),
          "epoch": epoch
  }
  if not os.path.isdir('model_save_dir'):
      os.makedirs('model_save_dir')
  torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
  ```
- 完整代码示例。

  ```
  import os
  import argparse
  parser = argparse.ArgumentParser()
  parser.add_argument("--train_output", type=str)
  args, unparsed = parser.parse_known_args()
  args = parser.parse_known_args()
  # train_output 将被赋值为"/home/ma-user/modelarts/outputs/train_output_0"
  train_output = args.train_output

  # 判断输出路径中是否有模型文件。如果无文件则默认从头训练，如果有模型文件，则加载epoch值最大的ckpt文件当做预训练模型。
  if os.listdir(train_output):
      print('> load last ckpt and continue training!!')
      last_ckpt = sorted([file for file in os.listdir(train_output) if file.endswith(".pth")])[-1]
      local_ckpt_file = os.path.join(train_output, last_ckpt)
      print('last_ckpt:', last_ckpt)
      # 加载断点
      checkpoint = torch.load(local_ckpt_file)
      # 加载模型可学习参数
      model.load_state_dict(checkpoint['net'])
      # 加载优化器参数
      optimizer.load_state_dict(checkpoint['optimizer'])
      # 获取保存的epoch，模型会在此epoch的基础上继续训练
      start_epoch = checkpoint['epoch']
  start = datetime.now()
  total_step = len(train_loader)
  for epoch in range(start_epoch + 1, args.epochs):
      for i, (images, labels) in enumerate(train_loader):
          images = images.cuda(non_blocking=True)
          labels = labels.cuda(non_blocking=True)
          # Forward pass
          outputs = model(images)
          loss = criterion(outputs, labels)
  ```

```
    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    ...

# 保存模型训练过程中的网络权重、优化器权重、以及epoch
checkpoint = {
      "net": model.state_dict(),
      "optimizer": optimizer.state_dict(),
      "epoch": epoch
    }
if not os.path.isdir(train_output):
    os.makedirs(train_output)
    torch.save(checkpoint, os.path.join(train_output, 'ckpt_best_{}.pth'.format(epoch)))
```

MindSpore 版 reload ckpt

```
import os
import argparse
from resnet import resnet50
from mindspore.nn.optim.momentum import Momentum
from mindspore.nn.loss import SoftmaxCrossEntropyWithLogits
from mindspore import load_checkpoint, load_param_into_net
from mindspore.train import Model, CheckpointConfig, ModelCheckpoint
from mindspore.train.callback import LossMonitor

parser = argparse.ArgumentParser()
parser.add_argument("--train_output", type=str)
parser.add_argument("--batch_size", type=int, default=32, help="Batch size.")
parser.add_argument("--num_classes", type=int, default=10, help="Num classes.")
parser.add_argument("--do_train", type=bool, default=True, help="Do train or not.")
args_opt, unparsed = parser.parse_known_args()
# train_output 将被赋值为"/home/ma-user/modelarts/outputs/train_output_0" 。
train_output = args_opt.train_output

# 初始定义的网络、损失函数及优化器，详细请参见MindSpore保存与加载。
# 1.初始定义的网络,以“ResNet50”为例。详细请参见ResNet50。
net = resnet50(args_opt.batch_size, args_opt.num_classes)
# 2.定义损失函数，详细请参见MindSpore自定义损失函数。
ls = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
# 3.定义优化器，详细请参见MindSpore自定义优化器。
opt = Momentum(filter(lambda x: x.requires_grad, net.get_parameters()), 0.01, 0.9)
# 首次训练的epoch初始值，mindspore1.3及以后版本会支持定义epoch_size初始值。
# cur_epoch_num = 0
# 判断输出obs路径中是否有模型文件。如果无文件则默认从头训练，如果有模型文件，则加载epoch值最大的
ckpt文件当做预训练模型。
if os.listdir(train_output):
    last_ckpt = sorted([file for file in os.listdir(train_output) if file.endswith(".ckpt")])[-1]
    print('last_ckpt:', last_ckpt)
    last_ckpt_file = os.path.join(train_output, last_ckpt)
     # 加载断点，详细请参见mindspore.load_checkpoint。
    param_dict = load_checkpoint(last_ckpt_file)
    print('> load last ckpt and continue training!!')
    # 加载模型参数到net。
    load_param_into_net(net, param_dict)
    # 加载模型参数到opt。
    load_param_into_net(opt, param_dict)

    # 获取保存的epoch值，模型会在此epoch的基础上继续训练,此参数在mindspore1.3及以后版本会支持。
    # if param_dict.get("epoch_num"):
    #     cur_epoch_num = int(param_dict["epoch_num"].data.asnumpy())
model = Model(net, loss_fn=ls, optimizer=opt, metrics={'acc'})
# as for train, users could use model.train
if args_opt.do_train:
    dataset = create_dataset()
    batch_num = dataset.get_dataset_size()
    config_ck = CheckpointConfig(save_checkpoint_steps=batch_num,
                                 keep_checkpoint_max=35)
```

```
    # append_info=[{"epoch_num": cur_epoch_num}],mindspore1.3及以后版本会支持append_info参数，保存当前时刻的epoch值。
    # 保存网络参数，详细请参见mindspore.train.ModelCheckpoint。
    ckpoint_cb = ModelCheckpoint(prefix="train_resnet_cifar10",
                                 directory=args_opt.train_output,
                                 config=config_ck)
    loss_cb = LossMonitor()
    model.train(epoch_size, dataset, callbacks=[ckpoint_cb, loss_cb])
    # model.train(epoch_size-cur_epoch_num, dataset, callbacks=[ckpoint_cb, loss_cb])，mindspore1.3及以后版本支持从断点恢复训练。
```

11 自动模型优化（AutoSearch）

11.1 自动模型优化介绍

ModelArts训练支持超参搜索功能，自动实现模型超参搜索，为您的模型匹配最合适的超参。

在模型训练过程中，有很多超参需要根据任务进行调整，比如learning_rate、weight_decay等，这一工作往往需要一个有经验的算法工程师花费一定精力和大量时间进行手动调优。ModelArts支持的超参搜索功能，在无需算法工程师介入的情况下，即可自动进行超参的调优，在速度和精度上超过人工调优。

ModelArts支持以下三种超参搜索算法：

- 贝叶斯优化（SMAC）
- TPE算法
- 模拟退火算法（Anneal）

贝叶斯优化（SMAC）

贝叶斯优化假设超参和目标函数存在一个函数关系。基于已搜索超参的评估值，通过高斯过程回归来估计其他搜索点处目标函数值的均值和方差。根据均值和方差构造采集函数（Acquisition Function），下一个搜索点为采集函数的极大值点。相比网格搜索，贝叶斯优化会利用之前的评估结果，从而降低迭代次数、缩短搜索时间；缺点是不容易找到全局最优解。

表 11-1 贝叶斯优化的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
kind	采集函数类型	string，默认为'ucb'，可能取值还有'ei'、'poi'，一般不建议用户修改
kappa	采集函数ucb的调节参数，可理解为上置信边界	float，一般不建议用户修改

参数	说明	取值参考
xi	采集函数poi和ei的调节参数	float，一般不建议用户修改

TPE 算法

TPE算法全称Tree-structured Parzen Estimator，是一种利用高斯混合模型来学习超参模型的算法。在每次试验中，对于每个超参，TPE为与最佳目标值相关的超参维护一个高斯混合模型l(x)，为剩余的超参维护另一个高斯混合模型g(x)，选择l(x)/g(x)最大化时对应的超参作为下一组搜索值。

表 11-2 TPE 算法的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
n_initial_points	采用TPE接近目标函数之前，对目标函数的随机评估数	int，一般不建议用户修改
gamma	TPE算法的一定分位数，用于划分l(x)和g(x)	float，范围(0,1)，一般不建议用户修改

模拟退火算法（Anneal）

模拟退火算法即Anneal算法，是随机搜索中一个简单但有效的变体，它利用了响应曲面中的平滑度。退火速率不自适应。Anneal算法从先前采样的一个试验点作为起点，然后从与先验分布相似的分布中采样每组超参数，但其密度更集中在选择的试验点周围。随着时间推移，算法会倾向于从越来越接近最佳点处采样。在采样过程中，算法可能绘制一个次佳试验作为最佳试验，以一定概率跳出局部最优解。

表 11-3 模拟退火算法的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
avg_best_idx	要探索试验的几何分布平均，从按照分数排序的试验中选择	float，一般不建议用户修改
shrink_coef	随着更多的点被探索，邻域采样大小的减少率	float，一般不建议用户修改

11.2 创建自动模型优化的训练作业

背景信息

如果用户使用的AI引擎为pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64和tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64，并且优化的超参类型为float类型，ModelArts支持用户使用超参搜索功能。

在0代码修改的基础下，实现算法模型的超参搜索。需要完成以下步骤：

1. 准备工作
2. 创建算法
3. 创建训练作业
4. 查看超参搜索作业详情

准备工作

- 数据已完成准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据集上传至OBS目录。
- 请准备好训练脚本，并上传至OBS目录。训练脚本开发指导参见开发用于预置框架训练的代码。
- 在训练代码中，用户需打印搜索指标参数。
- 已在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
- 由于训练作业运行需消耗资源，确保账户未欠费。
- 确保您使用的OBS目录与ModelArts在同一区域。

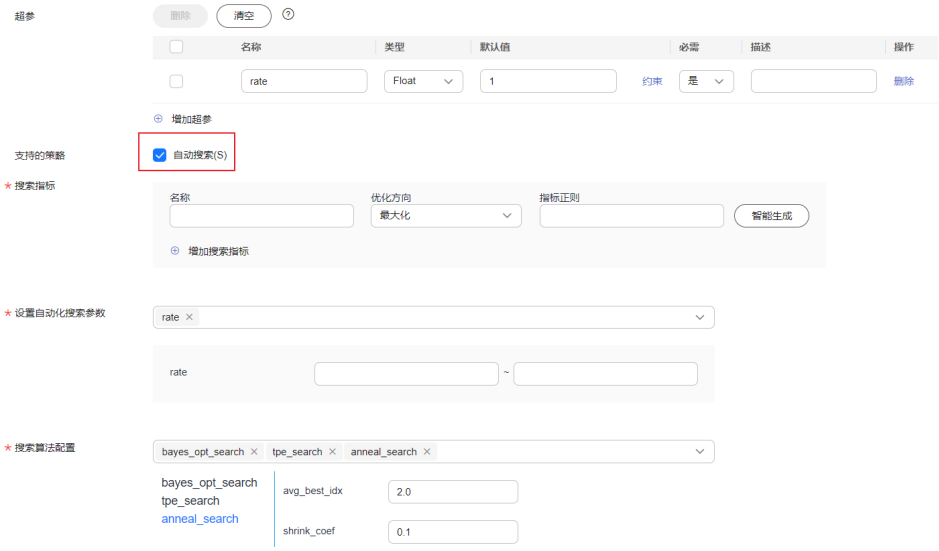
创建算法

进入登录**ModelArts管理控制台**，参考创建算法操作指导，创建自定义算法。镜像应该满足pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64或tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64引擎。

对于用户希望优化的超参，需在“超参”设置中定义，可以给定名称、类型、默认值、约束等，具体设置方法可以参考**表5-6**。

单击勾选“自动搜索”，为算法设置算法搜索功能。自动搜索作业运行过程中，ModelArts后台通过指标正则表达式获取搜索指标参数，朝指定的优化方向进行超参优化。用户需要在代码中打印搜索参数并在控制台配置以下参数。

图 11-1 设置算法搜索功能



- 搜索指标

  搜索指标为目标函数的值，通常可以设置为loss、accuracy等。通过优化搜索指标的目标值超优化方向收敛，找到最契合的超参，提高模型精度和收敛速度。

  表 11-4 搜索指标参数

参数	说明
名称	搜索指标的名称。需要与您在代码中打印的搜索指标参数保持一致。
优化方向	可选“最大化”或者“最小化”。
指标正则	填入正则表达式。您可以单击智能生成功能自动获取正则表达式。

- 设置自动化搜索参数

  从已设置的“超参”中选择可用于搜索优化的超参。优化的超参仅支持float类型，选中自动化搜索参数后，需设置取值范围。

- 搜索算法配置

  ModelArts内置三种超参搜索算法，用户可以根据实际情况选择对应的算法，支持多选。对应的算法和参数解析请参考以下：

  – bayes_opt_search：**贝叶斯优化（SMAC）**
  – tpe_search：**TPE算法**
  – anneal_search：**模拟退火算法（Anneal）**

提交创建算法完成后即可执行下一步，创建训练作业。

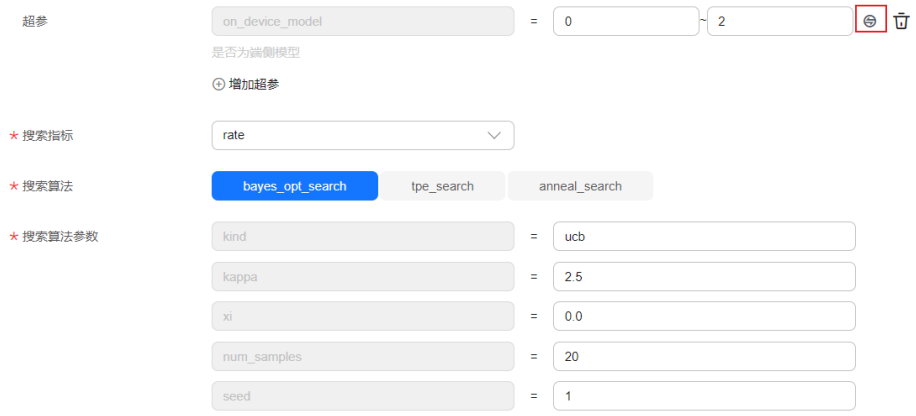
创建训练作业

登录**ModelArts管理控制台**，参考**创建生产训练作业**操作指导，创建训练作业。用户需关注以下操作才能开启超参搜索。

版权所有 © 华为云计算技术有限公司

当您选择支持超参搜索的算法，需单击超参的范围设置按钮才能开启超参搜索功能。

图 11-2 开启超参搜索功能



开启超参搜索功能后，用户可以设置搜索指标、搜索算法和搜索算法参数。三个参数显示的支持值与算法管理模块的超参设置对应。

完成超参搜索作业的创建后，训练作业需要运行一段时间。

查看超参搜索作业详情

训练作业运行结束后，可以查看自动超参搜索结果判断此训练作业是否满意。

如果训练作业是超参搜索作业，进入训练作业详情页，选择“自动超参搜索结果”页签查看超参搜索结果。

图 11-3 超参搜索结果

版权所有 © 华为云计算技术有限公司

12 模型训练高可靠性

12.1 设置断点续训练

什么是断点续训练

断点续训练是指因为某些原因（例如容错重启、资源抢占、作业卡死等）导致训练作业还未完成就被中断，下一次训练可以在上一次的训练基础上继续进行。这种方式对于需要长时间训练的模型而言比较友好。

断点续训练是通过checkpoint机制实现。

checkpoint的机制是：在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当需要从训练中断的位置接续训练，只需要加载checkpoint，并用checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，使能读取前一次训练保存的预训练模型。

ModelArts 中设置训练输出实现断点续训练

新版：

在ModelArts训练中实现断点续训练或增量训练，建议使用存储挂载功能。

在创建训练作业时，通过挂载存储路径来保存和读取Checkpoint文件。具体操作如下：

1. 在训练作业中配置存储路径挂载，将存储Checkpoint的目录挂载到训练容器的本地目录。
2. 训练过程中，将Checkpoint文件保存到挂载的本地目录中，数据会自动同步到挂载的存储位置。
3. 对于断点续训练，确保挂载的存储目录中包含之前的Checkpoint文件，训练脚本会自动加载最新的Checkpoint继续训练。

通过存储挂载方式，可以实现训练数据的持久化存储和跨作业的模型复用。

在ModelArts中创建训练作业时，可以选择以下两种存储挂载选项。以下是它们的详细对比，帮助您根据需求选择合适的存储方案。

表 12-1 两种存储挂载

存储类型	性能	容量	适用场景	价格	备注
SFS Turbo	高	大	适用于多种应用场景，包括AI训练、AIGC、自动驾驶、渲染、EDA仿真、企业NAS应用等。	较高	通用
OBS	中	大	基于对象存储服务作为统一数据存储的大数据场景。	适中	高频读取，低频写入。

旧版：

在ModelArts训练中实现断点续训练或增量训练，建议使用“训练输出”功能。

在创建训练作业时，设置训练“输出”参数名称为“train_output”，用户可通过环境变量或超参方式获取该参数。设置成功后可在指定的训练输出的数据存储位置中保存**Checkpoint**，且**“预下载至本地目录”**选择“下载”。选择预下载至本地目录时，系统在训练作业启动前，自动将数据存储位置中的**Checkpoint**文件下载到训练容器的本地目录。

图 12-1 训练输出设置

断点续训练建议和训练容错检查（即自动重启）功能同时使用。在创建训练作业页面，开启“自动重启”开关。训练环境预检测失败、或者训练容器硬件检测故障、或者训练作业失败时会自动重新下发并运行训练作业。

VeRL 框架 reload ckpt

VeRL是一个灵活、高效且被广泛使用的强化学习（RL）训练库，后训练的事实标准框架。VeRL是论文 **HybridFlow: A Flexible and Efficient RLHF Framework** 的开源实现。

1. VeRL的训练yaml中配置**trainer.save_freq**参数和**trainer.default_local_dir**参数

   VeRL通过trainer.default_local_dir参数配置输出目录，该目录下会存在多个global_steps_xx权重目录，通过trainer.save_freq参数配置权重保存的频率，每隔一定的步数保存权重。

2. VeRL的训练yaml中配置**trainer.resume_mode**参数

当trainer.resume_mode设置为auto时，VeRL会自动遍历trainer.default_local_dir路径，加载最新且有效的ckpt。以**ModelArts中设置训练输出实现断点续训练**中的参数名称为train_output为例，参数设置如下所示：

```
trainer.default_local_dir="${train_output}"
trainer.resume_mode=auto
```

MindSpeed-LLM 框架 reload ckpt

MindSpeed LLM是基于昇腾生态的大语言模型分布式训练框架，旨在为华为 **昇腾芯片** 生态合作伙伴提供端到端的大语言模型训练方案，包含分布式预训练、分布式指令微调以及对应的开发工具链，如：数据预处理、权重转换、在线推理、基线评估。作为昇腾计算主打的训练框架，在性能上做了极致的优化，特别在大参数、大集群和MOE类型模型的训练性能突出，且兼容Megatron-LM框架，对于Megatron客户可以平滑迁移。

1. MindSpeed-LLM中配置**--save**参数和**--save-interval**参数

   MindSpeed-LLM训练启动脚本中通过--save参数配置输出目录，该目录下会存在多个iter_xx权重目录和一个记录最新权重保存步数的文件latest_checkpointed_iteration.txt，每次save会更新latest_checkpointed_iteration.txt。通过--save-interval参数配置权重保存的频率，每隔一定的步数保存权重。

2. MindSpeed-LLM中配置**--load**参数，并且和--save参数路径保持一致

   MindSpeed-LLM训练启动脚本中通过--load参数配置输入目录，当--save参数和--load参数设置的参数保持一致时，每次重启训练业务即可加载最新的权重，以**ModelArts中设置训练输出实现断点续训练**中的参数名称为train_output为例，参数设置如下所示：

   ```
   --save-interval 1000
   --save ${train_output}
   --load ${train_output}
   ```

LLaMa-Factory 框架 reload ckpt

LLaMa-Factory是开源社区中一个非常活跃的大模型训练框架，它的主打特点是简单易用，通过命令行或者WebUI界面可以轻松微调数百种大模型，包括大语言和多模态模型。LLaMa-Factory的底层是基于Transformers+DeepSpeed构建，对开源模型就有非常好的兼容性。

1. LLaMa-Factory的训练yaml中配置**output_dir**参数和**save_steps**参数

   LLaMa-Factory通过output_dir参数配置输出目录，该目录下会存在多个checkpoint-xxx权重目录，通过save_steps参数配置权重保存频率。

2. LLaMa-Factory的训练yaml中配置**resume_from_checkpoint**参数，并且和**output_dir**参数路径保持一致

   LLaMa-Factory通过resume_from_checkpoint显式指定本次训练使用的ckpt，如果指定了有效的ckpt则从指定的ckpt中恢复训练。当resume_from_checkpoint参数和output_dir参数保持一致时，但output_dir参数不是一个有效ckpt目录，因此还需要步骤3和步骤4。以**ModelArts中设置训练输出实现断点续训练**中的参数名称为train_output为例，参数设置如下所示：

   ```
   ### output
   output_dir: ${train_output}
   save_steps: 500

   ### train
   resume_from_checkpoint: ${train_output}
   ```

3. 首先需要创建一个resume.py脚本，该脚本需要传入训练配置yaml的绝对路径，具体代码如下所示：

```
import os
import re
import sys


def update_resume_config(config_file):  # 接收传入的配置文件路径
    # 读取配置文件内容
    with open(config_file, 'r', encoding='utf-8') as f:
        lines = f.readlines()

    resume_line_num = None
    resume_path = None

    # 查找 resume_from_checkpoint 行
    for i, line in enumerate(lines):
        if line.strip().startswith('resume_from_checkpoint:'):
            resume_line_num = i
            # 提取值
            parts = line.split(':', 1)
            if len(parts) > 1:
                resume_path = parts[1].strip().strip('"\'')  # 去除引号
            break

    # 如果没找到或值为null，不处理
    if resume_line_num is None or resume_path in (None, 'null', ''):
        return

    # 检查路径并找到最新的checkpoint
    new_resume_path = None
    if os.path.isdir(resume_path):
        # 查找所有 checkpoint-数字 文件夹
        checkpoint_pattern = re.compile(r'^checkpoint-(\d+)$')
        checkpoints = []

        for item in os.listdir(resume_path):
            item_path = os.path.join(resume_path, item)
            if os.path.isdir(item_path):
                match = checkpoint_pattern.match(item)
                if match:
                    step = int(match.group(1))
                    checkpoints.append((step, item_path))

        # 如果找到checkpoint，使用最新的那个
        if checkpoints:
            checkpoints.sort(key=lambda x: x[0])
            new_resume_path = checkpoints[-1][1]

    # 修改配置行
    indent = len(line) - len(line.lstrip())  # 保留原缩进
    if new_resume_path:
        lines[resume_line_num] = f'{" " * indent}resume_from_checkpoint: {new_resume_path}\n'
    else:
        lines[resume_line_num] = f'{" " * indent}resume_from_checkpoint: null\n'

    # 写回文件
    with open(config_file, 'w', encoding='utf-8') as f:
        f.writelines(lines)


if __name__ == "__main__":
    # 从命令行参数中获取配置文件路径
    if len(sys.argv) < 2:
        print("用法：python resume.py <配置文件路径>")
        sys.exit(1)
    config_file = sys.argv[1]  # 接收命令行传入的 abc.yaml
    update_resume_config(config_file)  # 传入函数执行
```

4. 修改训练启动脚本，在执行llamafactory-cli train命令之前，通过resume.py脚本修改训练配置中yaml中resume_from_checkpoint参数。resume.py脚本会获取resume_from_checkpoint参数中的路径，去该路径下查找步数最大的checkpoint-xxx权重目录，然后将resume_from_checkpoint参数修改为该权重目录的绝对路径。以train_lora/deepseek3_lora_sft_kt.yaml为例，其中WORK_DIR为工作路径，shell脚本修改如下所示：

```
#!/bin/bash
...
...

python $WORK_DIR/resume.py $WORK_DIR/LLaMA-Factory/examples/train_lora/
deepseek3_lora_sft_kt.yaml
llamafactory-cli train $WORK_DIR/LLaMA-Factory/examples/train_lora/deepseek3_lora_sft_kt.yaml
```

PyTorch 版 reload ckpt

- PyTorch模型保存有两种方式。
  - 仅保存模型参数

    ```
    state_dict = model.state_dict()
    torch.save(state_dict, path)
    ```

  - 保存整个Model（不推荐）

    ```
    torch.save(model, path)
    ```

- 可根据step步数、时间等周期性保存模型的训练过程的产物。

  将模型训练过程中的网络权重、优化器权重、以及epoch进行保存，便于中断后继续训练恢复。

  ```
  checkpoint = {
          "net": model.state_dict(),
          "optimizer": optimizer.state_dict(),
          "epoch": epoch
  }
  if not os.path.isdir('model_save_dir'):
      os.makedirs('model_save_dir')
  torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
  ```

- 完整代码示例。

  ```
  import os
  import argparse
  parser = argparse.ArgumentParser()
  parser.add_argument("--train_output", type=str)
  args, unparsed = parser.parse_known_args()
  args = parser.parse_known_args()
  # train_output 将被赋值为"/home/ma-user/modelarts/outputs/train_output_0"
  train_output = args.train_output

  # 判断输出路径中是否有模型文件。如果无文件则默认从头训练，如果有模型文件，则加载epoch值最大的
  ckpt文件当做预训练模型。
  if os.listdir(train_output):
      print('> load last ckpt and continue training!!')
      last_ckpt = sorted([file for file in os.listdir(train_output) if file.endswith(".pth")])[-1]
      local_ckpt_file = os.path.join(train_output, last_ckpt)
      print('last_ckpt:', last_ckpt)
      # 加载断点
      checkpoint = torch.load(local_ckpt_file)
      # 加载模型可学习参数
      model.load_state_dict(checkpoint['net'])
      # 加载优化器参数
      optimizer.load_state_dict(checkpoint['optimizer'])
      # 获取保存的epoch，模型会在此epoch的基础上继续训练
      start_epoch = checkpoint['epoch']
  start = datetime.now()
  total_step = len(train_loader)
  for epoch in range(start_epoch + 1, args.epochs):
      for i, (images, labels) in enumerate(train_loader):
          images = images.cuda(non_blocking=True)
  ```

```
    labels = labels.cuda(non_blocking=True)
    # Forward pass
    outputs = model(images)
    loss = criterion(outputs, labels)
    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    ...

# 保存模型训练过程中的网络权重、优化器权重、以及epoch
checkpoint = {
      "net": model.state_dict(),
      "optimizer": optimizer.state_dict(),
      "epoch": epoch
    }
if not os.path.isdir(train_output):
    os.makedirs(train_output)
    torch.save(checkpoint, os.path.join(train_output, 'ckpt_best_{}.pth'.format(epoch)))
```

MindSpore 版 reload ckpt

```
import os
import argparse
from resnet import resnet50
from mindspore.nn.optim.momentum import Momentum
from mindspore.nn.loss import SoftmaxCrossEntropyWithLogits
from mindspore import load_checkpoint, load_param_into_net
from mindspore.train import Model, CheckpointConfig, ModelCheckpoint
from mindspore.train.callback import LossMonitor

parser = argparse.ArgumentParser()
parser.add_argument("--train_output", type=str)
parser.add_argument("--batch_size", type=int, default=32, help="Batch size.")
parser.add_argument("--num_classes", type=int, default=10, help="Num classes.")
parser.add_argument("--do_train", type=bool, default=True, help="Do train or not.")
args_opt, unparsed = parser.parse_known_args()
# train_output 将被赋值为"/home/ma-user/modelarts/outputs/train_output_0" 。
train_output = args_opt.train_output

# 初始定义的网络、损失函数及优化器，详细请参见MindSpore保存与加载。
# 1.初始定义的网络,以“ResNet50”为例。详细请参见ResNet50。
net = resnet50(args_opt.batch_size, args_opt.num_classes)
# 2.定义损失函数，详细请参见MindSpore自定义损失函数。
ls = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
# 3.定义优化器，详细请参见MindSpore自定义优化器。
opt = Momentum(filter(lambda x: x.requires_grad, net.get_parameters()), 0.01, 0.9)
# 首次训练的epoch初始值，mindspore1.3及以后版本会支持定义epoch_size初始值。
# cur_epoch_num = 0
# 判断输出obs路径中是否有模型文件。如果无文件则默认从头训练，如果有模型文件，则加载epoch值最大的
ckpt文件当做预训练模型。
if os.listdir(train_output):
    last_ckpt = sorted([file for file in os.listdir(train_output) if file.endswith(".ckpt")])[-1]
    print('last_ckpt:', last_ckpt)
    last_ckpt_file = os.path.join(train_output, last_ckpt)
     # 加载断点，详细请参见mindspore.load_checkpoint。
    param_dict = load_checkpoint(last_ckpt_file)
    print('> load last ckpt and continue training!!')
    # 加载模型参数到net。
    load_param_into_net(net, param_dict)
    # 加载模型参数到opt。
    load_param_into_net(opt, param_dict)

    # 获取保存的epoch值，模型会在此epoch的基础上继续训练,此参数在mindspore1.3及以后版本会支持。
    # if param_dict.get("epoch_num"):
    #     cur_epoch_num = int(param_dict["epoch_num"].data.asnumpy())
model = Model(net, loss_fn=ls, optimizer=opt, metrics={'acc'})
# as for train, users could use model.train
if args_opt.do_train:
```

```
    dataset = create_dataset()
    batch_num = dataset.get_dataset_size()
    config_ck = CheckpointConfig(save_checkpoint_steps=batch_num,
                                 keep_checkpoint_max=35)
    # append_info=[{"epoch_num": cur_epoch_num}],mindspore1.3及以后版本会支持append_info参数，保存当前时刻的epoch值。
    # 保存网络参数，详细请参见mindspore.train.ModelCheckpoint。
    ckpoint_cb = ModelCheckpoint(prefix="train_resnet_cifar10",
                                 directory=args_opt.train_output,
                                 config=config_ck)
    loss_cb = LossMonitor()
    model.train(epoch_size, dataset, callbacks=[ckpoint_cb, loss_cb])
    # model.train(epoch_size-cur_epoch_num, dataset, callbacks=[ckpoint_cb, loss_cb])，mindspore1.3及以后版本支持从断点恢复训练。
```

12.2 训练作业容错检查

视频介绍

场景描述

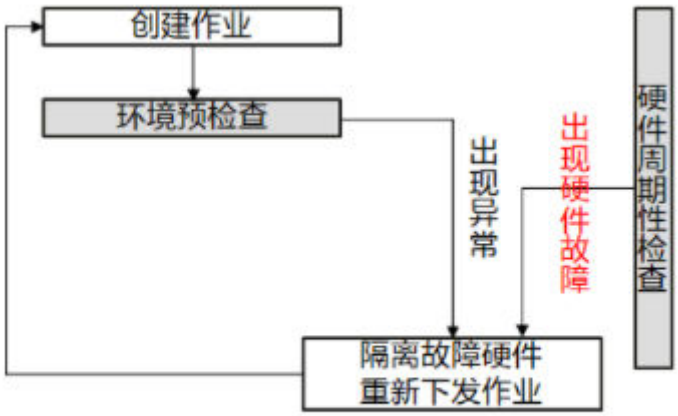
用户在训练模型过程中，存在因硬件故障而产生的训练失败场景。针对硬件故障场景，ModelArts提供容错检查功能，帮助用户隔离故障节点，优化用户训练体验。

容错检查包括两个检查项：环境预检测与硬件周期性检查。当环境预检查或者硬件周期性检查任一检查项出现故障时，隔离故障硬件并重新下发训练作业。针对于分布式场景，容错检查会检查本次训练作业的全部计算节点。

下图中有四个场景，其中场景四为正常训练作业失败场景，其他三个场景下可开启容错功能进行训练作业自动恢复。

- 场景一：环境预检测失败、硬件检测出现故障，系统隔离所有故障节点并重新下发训练作业。

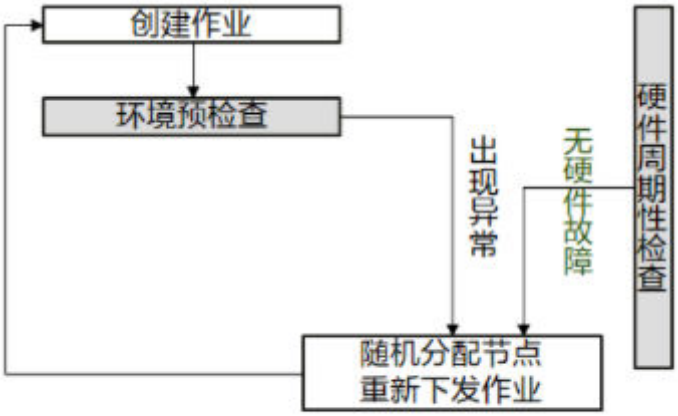
  图 12-2 预检失败&硬件故障



1. 预检失败&硬件故障

- 场景二：环境预检测失败、硬件无故障，系统随机再分配节点并重新下发训练作业。

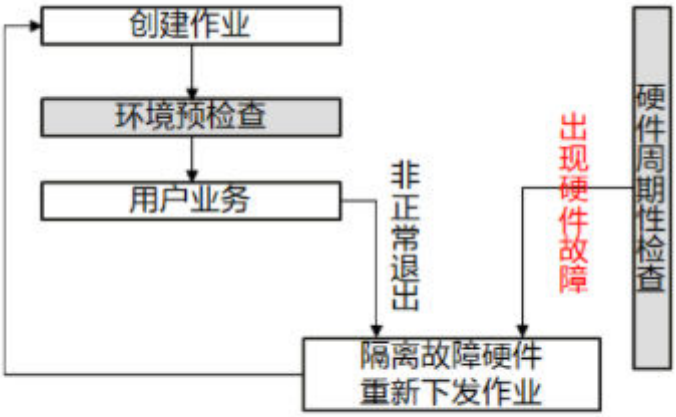
图 12-3 预检失败&硬件正常



2. 预检失败&硬件正常

- 场景三：环境预检测成功并进入用户业务阶段，硬件检测出现故障并且用户业务非正常退出，系统隔离所有故障节点并重新下发训练作业。

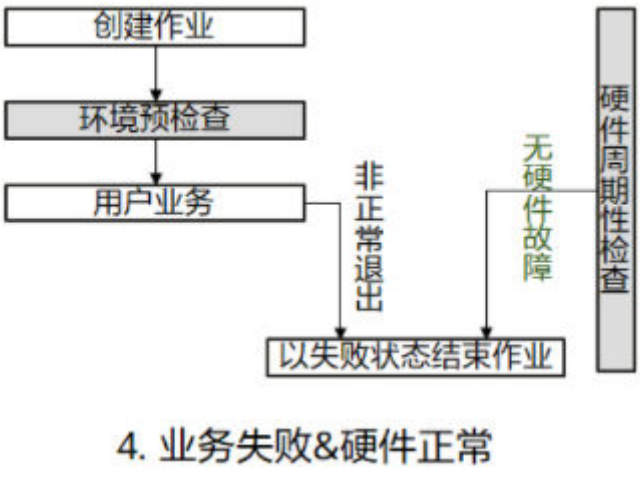
图 12-4 业务失败&硬件故障



3. 业务失败&硬件故障

- 场景四：环境预检测成功并进入用户业务阶段，硬件无故障，当用户业务异常时系统以失败状态结束作业。

图 12-5 业务失败&硬件正常



隔离故障节点后，系统会在新的计算节点上重新创建训练作业。如果资源池规格紧张，重新下发的训练作业会以第一优先级进行排队。如果排队时间超过30分钟，训练作业会自动退出。该现象表明资源池规格任务紧张，训练作业无法正常启动，推荐您购买专属资源池补充计算节点。

如果您使用专属资源池创建训练作业，容错检查识别的故障节点会被剔除。系统自动补充健康的计算节点至专属资源池。（该功能即将上线）

容错检查详细介绍请参考：

1. 开启容错检查
2. 检测项目与执行条件
3. 触发容错环境检测达到的效果
4. 环境预检查通过后，如果发生硬件故障会导致用户业务中断。您可以在训练中补充reload ckpt的代码逻辑，使能读取训练中断前保存的预训练模型。指导请参考设置断点续训练。

开启容错检查

用户可以在创建训练作业时通过设置自动重启的方式开启容错检查。

- 使用ModelArts控制台的创建训练作业页面设置自动重启：

  用户可以在控制台页面通过开关的方式开启自动重启。“自动重启”开关默认不开启，表示不做重新下发作业，也不会启用环境检测。打开开关后，允许设置重启次数为1~128次。

  图 12-6 自动重启设置

- 使用API接口设置容错检查：

  用户可以通过API接口的方式开启自动重启。创建训练作业时，在“metadata”字段的“annotations”中传入“fault-tolerance/job-retry-num”字段。

  添加“fault-tolerance/job-retry-num”字段，视为开启自动重启，value的范围可以设置为1~128的整数。value值表示最大允许重新下发作业的次数。如果不传入则默认为0，表示不做重新下发作业，也不会启用环境检测。

  图 12-7 设置 API

```
{
    "kind": "job",
    "metadata": {
        "annotations": {
            "fault-tolerance/job-retry-num": "3"
        },
```

检测项目与执行条件

检测项目	item（日志关键字）	执行条件	检测成功要求
域名检测	dns	无	volcano容器的域名都解析成功（/etc/volcano下的“.host”文件中的域名解析成功）
磁盘空间-容器根目录	disk-size root	无	大于32GB
磁盘空间-/dev/shm目录	disk-size shm	无	大于1GB
磁盘空间-/cache目录	disk-size cache	无	大于32GB
ulimit检查	ulimit	使用IB网络时	• max locked memory > 16000 • open files > 1000000 • stack size > 8000 • max user processes > 1000000
gpu检查	gpu-check	使用gpu，且使用v2训练引擎时（北京四暂无）	检测到gpu

触发容错环境检测达到的效果

- 容错检查正常通过时，会打印检测项目的日志，表示具体涉及的检查项目成功。您可以通过在日志中搜索“item”关键字查看。当容错检查正常通过时，可以减少运行故障上报问题。

- 容错检查失败时，会打印检查失败的日志。您可以通过在日志中搜索“item”关键字查看失败信息。

  如果作业重启次数没有达到设定的次数，则会自动做重新下发作业。您可以通过搜索“error,exiting”关键字查找作业重启失败结束的日志。

使用 reload ckpt 恢复中断的训练

在容错机制下，如果因为硬件问题导致训练作业重启，用户可以在代码中读取预训练模型，恢复至重启前的训练状态。用户需要在代码里加上reload ckpt的代码，使能读取训练中断前保存的预训练模型。具体请参见**断点续训练**。

查看容错与恢复详情

当训练作业发生故障恢复时（例如NPU原地恢复、JOB级重调度等），作业详情页面中会出现“故障恢复详情”页签，里面记录了训练作业的启停情况。

用户创建训练作业时开启自动重启后，可以在训练作业详情页面查看“重启次数”。“重启次数”会展示当前重启次数/最大重启次数。在训练作业详情页的“容错与恢复”页签，可以查看训练作业重启详情。

图 12-8 重启次数

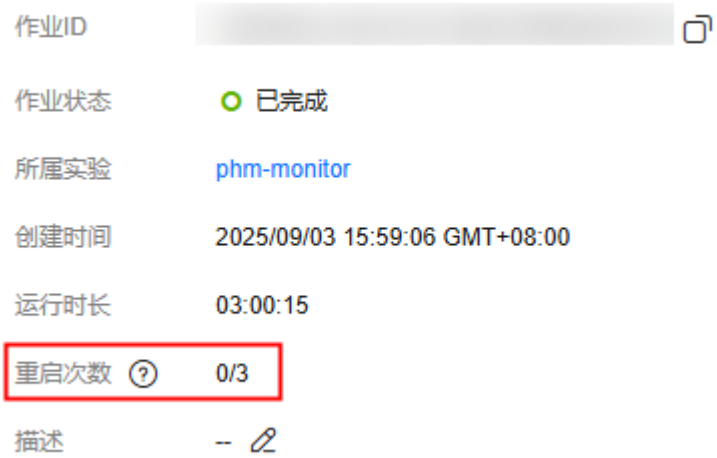
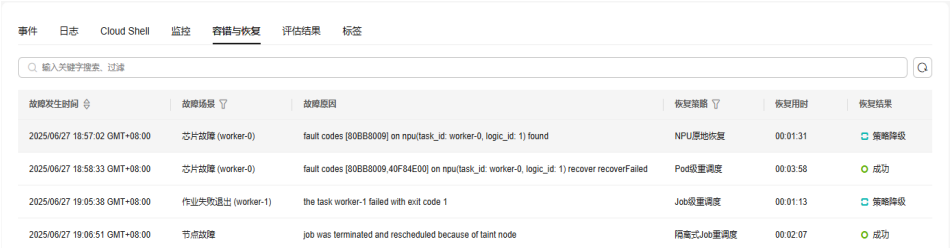


图 12-9 容错与恢复



12.3 训练作业卡死检测

什么是训练作业卡死检测

训练作业在运行中可能会因为某些未知原因导致作业卡死，如果不能及时发现，就会导致无法及时释放资源，从而造成极大的资源浪费。为了节省训练资源成本，提高使用体验，ModelArts提供了卡死检测功能，能自动识别作业是否卡死，并在日志详情界面上展示，同时能配置通知及时提醒用户作业卡死。

检测规则

卡死检测主要是通过监控作业进程的状态和资源利用率来判定作业是否卡死，会启动一个协程来周期性地监控上述两个指标的变化情况。卡死检测有单实例和全实例两种检测规则，是同时生效的。

- **单实例检测**
  - 进程状态：只要训练作业单实例中的进程IO存在变化，就进入下一个检测周期。如果在多个检测周期内，所有进程IO都没有变化，则进入资源利用率检测阶段。
  - 资源利用率：在作业单实例进程IO没有变化的情况下，采集一定时间段内的GPU利用率或NPU利用率，并根据这段时间内的GPU利用率或NPU利用率的方差和中位数来判断资源使用率是否有变化。如果没有变化，则判定作业卡死。
- **全实例检测**

  资源利用率：当作业在一段时间内所有运行中的实例的GPU利用率或者NPU利用率没有变化，同时每个实例的CPU使用也低于1核，则判定作业卡死。

系统预置了卡死检测的环境变量“MA_HANG_DETECT_TIME=30”，表示检测到指标异常并持续30分钟则判定作业卡死。如果需要修改卡死检测时间，则可以修改环境变量“MA_HANG_DETECT_TIME”的值，具体操作指导请参见**管理训练容器环境变量**。

注意

- 由于检测规则的局限性，当前卡死检测存在一定的误检率。如果是作业代码本身逻辑（如长时间sleep）导致的卡死，请忽略。
- 如果对于误检有疑问或者卡死问题无法自行解决，您可以前往**ModelArts开发者论坛**进行提问或者搜索问题。

约束限制

卡死检测仅支持资源类型为GPU和NPU的训练作业。

操作步骤

卡死检测无需额外配置，作业运行中会自动执行检测。检测到作业卡死后会在训练作业详情页提示作业疑似卡死。如需检测到卡死后发送通知（短信、邮件等）请在作业创建页面配置事件通知。

常见案例

1. **复制数据卡死**

   问题现象

   调用mox.file.copy_parallel复制数据时卡死。

   解决方案

   – 复制文件和文件夹均可采用：
   ```
   import moxing as mox
   mox.file.set_auth(is_secure=False)
   ```
   – 复制单个大文件5G以上时可采用：
   ```
   from moxing.framework.file import file_io
   ```
   查看当前moxing调用的接口版本：file_io._LARGE_FILE_METHOD，如果输出值为1则为V1版本，如果输出值为2，则为V2版本。

   V1版本修改：file_io._NUMBER_OF_PROCESSES=1

   V2版本修改：file_io._LARGE_FILE_METHOD = 1，将模式设置成V1然后用V1的方式修改规避，也可以直接file_io._LARGE_FILE_TASK_NUM=1。

   – 复制文件夹时可采用：
   ```
   mox.file.copy_parallel(threads=0,is_processing=False)
   ```

2. **训练前卡死**

   作业为多节点训练，且还未开始训练时发生卡死，可以在代码中加入os.environ["NCCL_DEBUG"] = "INFO"，查看NCCL DEBUG信息。

   – **问题现象**1

   日志中还未出现NCCL DEBUG信息时已卡死。

   解决方案1

   检查代码，检查是否有参数中未传入“master_ip”和“rank”参数等问题。

   – **问题现象2**

   分布式训练的日志中，发现有的节点含有GDR信息，而有的节点无GDR信息，导致卡死的原因可能为GDR。
   ```
   # 节点A日志
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1136:1191 [2] NCCL INFO
   Channel 00 : 3[5f000] -> 10[5b000] [receive] via NET/IB/0/GDRDMA
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1140:1196 [6] NCCL INFO
   Channel 00 : 14[e1000] -> 15[e9000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1141:1187 [7] NCCL INFO
   Channel 00 : 15[e9000] -> 11[5f000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1138:1189 [4] NCCL INFO
   Channel 00 : 12[b5000] -> 14[e1000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1137:1197 [3] NCCL INFO
   Channel 00 : 11[5f000] -> 16[2d000] [send] via NET/IB/0/GDRDMA

   # 节点B日志
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1139:1198 [2] NCCL INFO
   Channel 00 : 18[5b000] -> 19[5f000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1144:1200 [7] NCCL INFO
   Channel 00 : 23[e9000] -> 20[b5000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1142:1196 [5] NCCL INFO
   Channel 00 : 21[be000] -> 17[32000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1143:1194 [6] NCCL INFO
   Channel 00 : 22[e1000] -> 21[be000] via P2P/IPC
   modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1141:1191 [4] NCCL INFO
   Channel 00 : 20[b5000] -> 22[e1000] via P2P/IPC
   ```

   解决方案2

   在程序开头设置“os.environ["NCCL_NET_GDR_LEVEL"] = '0'”关闭使用GDR，或者寻找运维人员将机器添加GDR。

– **问题现象3**

NCCL信息中报出Got completion with error 12, opcode 1, len 32478, vendor err 129等通信信息时，说明当前网络不是很稳定。

解决方案3

可加入3个环境变量。

- NCCL_IB_GID_INDEX=3： 使用RoCE v2协议，默认使用RoCE v1，但是v1在交换机上没有拥塞控制，可能丢包，而且后面的交换机不会支持v1，就无法启动。
- NCCL_IB_TC=128：数据包走交换机的队列4通道，这是RoCE协议标准。
- NCCL_IB_TIMEOUT=22：把超时时间设置长一点，正常情况下网络不稳定会有5秒钟左右的间断，超过5秒就返回timeout了，改成22预计有二十秒左右，算法为4.096 μs * 2 ^ timeout。

3. **训练中途卡死**

– **问题现象1**

检测每个节点日志是否有报错信息，某个节点报错但作业未退出导致整个训练作业卡死。

解决方案1

查看报错原因，解决报错。

– **问题现象2**

作业卡在sync-batch-norm中或者训练速度变慢。pytorch如果开了sync-batch-norm，多机会慢，因开了sync-batch-norm以后，每一个iter里面每个batch-norm层都要做同步，通信量很大，而且要所有节点同步。

解决方案2

关掉sync-batch-norm，或者升pytorch版本，升级pytorch到1.10。

– **问题现象3**

作业卡在tensorboard中，出现报错：

```
writer = SummaryWriter('./path/to/log')
```

解决方案3

存储路径设为本地路径，如cache/tensorboard，不要使用OBS路径。

– **问题现象4**

使用pytorch中的dataloader读数据时，作业卡在读数据过程中，日志停在训练的过程中并不再更新日志。

解决方案4

用dataloader读数据时，适当减小num_worker。

4. **训练最后一个epoch卡死**

问题现象

通过日志查看数据切分是否对齐，如果未对齐，容易导致部分进程完成训练退出，而部分训练进程因未收到其他进程反馈卡死，如下图同一时间有的进程在epoch48，而有的进程在epoch49。

```
loss exit lane:0.12314446270465851
step loss is 0.29470521211624146
[2022-04-26 13:57:20,757][INFO][train_epoch]:Rank:2 Epoch:[48][20384/all] Data Time 0.000(0.000)
Net Time 0.705(0.890) Loss 0.3403(0.3792)LR 0.00021887
[2022-04-26 13:57:20,757][INFO][train_epoch]:Rank:1 Epoch:[48][20384/all] Data Time 0.000(0.000)
Net Time 0.705(0.891) Loss 0.3028(0.3466) LR 0.00021887
```

```
[2022-04-26 13:57:20,757][INFO][train_epoch]:Rank:4 Epoch:[49][20384/all] Data Time 0.000(0.147)
Net Time 0.705(0.709) Loss 0.3364(0.3414)LR 0.00021887
[2022-04-26 13:57:20,758][INFO][train_epoch]:Rank:3 Epoch:[49][20384/all] Data Time 0.000 (0.115)
Net Time 0.706(0.814) Loss 0.3345(0.3418) LR 0.00021887
[2022-04-26 13:57:20,758][INFO][train_epoch]:Rank:0 Epoch:[49][20384/all] Data Time 0.000(0.006)
Net Time 0.704(0.885) Loss 0.2947(0.3566) LR 0.00021887
[2022-04-26 13:57:20,758][INFO][train_epoch]:Rank:7 Epoch:[49][20384/all] Data Time 0.001 (0.000)
Net Time 0.706 (0.891) Loss 0.3782(0.3614) LR 0.00021887
[2022-04-26 13:57:20,759][INFO][train_epoch]:Rank:5 Epoch:[48][20384/all] Data Time 0.000(0.000)
Net Time 0.706(0.891) Loss 0.5471(0.3642) LR 0.00021887
[2022-04-26 13:57:20,763][INFO][train_epoch]:Rank:6 Epoch:[49][20384/all] Data Time 0.000(0.000)
Net Time 0.704(0.891) Loss 0.2643(0.3390)LR 0.00021887
stage 1 loss 0.4600560665130615 mul_cls_loss loss:0.01245919056236744 mul_offset_loss
0.44759687781333923 origin stage2_loss 0.048592399805784225
stage 1 loss:0.4600560665130615 stage 2 loss:0.048592399805784225 loss exit
lane:0.10233864188194275
```

解决方案

使用tensor的切分操作对齐数据。

12.4 训练作业故障恢复

场景描述

用户在创建训练作业时可以通过设置自动重启的方式开启容错检查，当训练作业所在节点故障或训练作业检查到异常时，系统会自动触发训练作业故障恢复机制，通过重启进程或重建作业等手段尝试恢复训练业务。不同恢复策略的及差异如表1 不同恢复策略及差异所示。

约束限制

1. 容错恢复主要是通过重启进程或重建作业恢复训练业务，为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见设置断点续训练。
2. 不同恢复策略约束情况如下：

表 12-2 不同恢复策略及差异

恢复策略	策略说明	应对的故障场景	约束限制
原地恢复	过程中不涉及资源调度，仅在原容器中重启训练作业。	• NPU芯片可自愈故障，以HBM多比特ECC为典型代表	• 作业定期保存CKPT • 作业支持断点续训 • 原地恢复保留故障时的容器环境，要求作业脚本可重入 • 无需备用机
无条件Job级重调度	在作业异常结束退出码非0时，将用户作业涉及到的所有Pod终止，并将Job实例完全重建。	• 偶现的软件故障或网络波动	• 作业定期保存CKPT • 作业支持断点续训 • 作业异常时业务能够中断退出，且作业退出码非0 • 无需备用机

恢复策略	策略说明	应对的故障场景	约束限制
隔离式Job重调度	在无条件Job重调度策略基础上，如果作业异常时发现节点故障，则在调度前先隔离故障节点。	• 所有NPU芯片故障 • 节点故障	• 作业定期保存CKPT • 作业支持断点续训 • 需要健康备用机
Pod重调度	保留现有的Job实例不变，在调度前先隔离故障节点，且仅重新创建故障Pod。	• 所有NPU芯片故障 • 节点故障	• 作业定期保存CKPT • 作业支持断点续训 • Pod重调度部分实例保留故障时的容器环境，要求作业脚本可重入 • 需要健康备用机
卡死重启	ModelArts会在作业运行过程中检测到卡死状态时，强制停止容器中的用户进程，容器本身不销毁保留运行环境，并在进程停止后重新运行训练作业启动命令。作业卡死重启的过程中不涉及资源调度，仅在原容器中重启训练作业。	• ModelArts平台检测到训练作业卡死	• 作业定期保存CKPT • 作业支持断点续训 • 卡死重启保留故障时的容器环境，要求作业脚本可重入 • 无需备用机

故障恢复开启方式

- 控制台设置

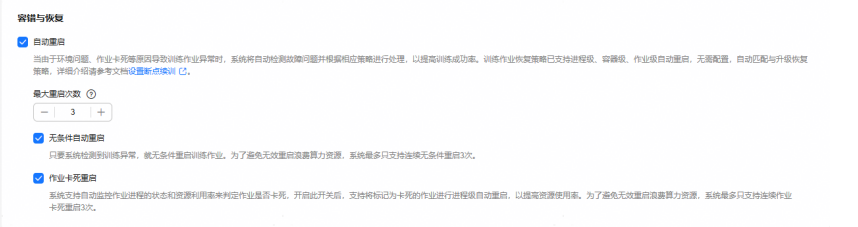
  在创建训练作业时，用户可根据**步骤六：高可用配置**配置开启。

  自动重启：包含原地恢复、隔离式Job级重调度，其中原地恢复不占用“最大重启次数”。

  无条件自动重启：包含无条件Job级重调度。

  作业卡死重启：包含卡死重启，卡死重启不占用“最大重启次数”。

  图 12-10 容错与恢复

- API接口设置

  “自动重启”：通过API接口创建训练作业时，在“metadata”字段的“annotations”中传入“fault-tolerance/job-retry-num”，“fault-tolerance/

job-retry-num”赋值为1~128之间的任意整数，表示开启自动重启并设置自动重启次数。

“无条件自动重启”：赋值“fault-tolerance/job-retry-num”，并将“fault-tolerance/job-unconditional-retry”赋值为“true”。

“作业卡死重启”：赋值“fault-tolerance/job-retry-num”，并将“fault-tolerance/hang-retry”赋值为“true”。

“Pod重调度”：赋值“fault-tolerance/job-retry-num”和“fault-tolerance/pod-retry-num”。

参数	是否必选	参数类型	描述
annotations	否	Map<String,String>	**参数解释**：训练作业高级功能配置。 **约束限制**：可选取值如下： ● "fault-tolerance/job-retry-num": "3"（故障自动重启次数） ● "fault-tolerance/job-unconditional-retry": "true"（无条件重启） ● "fault-tolerance/hang-retry": "true"（卡死重启） ● "fault-tolerance/pod-retry-num": "3"（Pod重调度次数）

```
{
    "kind": "job",
    "metadata": {
        "annotations": {
            "fault-tolerance/job-retry-num": "8",
            "fault-tolerance/job-unconditional-retry": "true",
           "fault-tolerance/hang-retry": "true",
            "fault-tolerance/pod-retry-num": "3"
        }
    }
}
```

故障恢复环境变量

以下环境变量可判断作业是否发生过断点续训，主要用于训练脚本的可重入。

变量名	说明
MA_SCHEDULE_CNT	Task所在Pod被调度的次数，新下发的作业初始值1，每次重调度恢复后，MA_SCHEDULE_CNT会在原值基础上加1。 因此当MA_SCHEDULE_CNT>1时，说明当前容器已经执行过一次重调度恢复。

变量名	说明
MA_PROC_START_CNT	当前Pod内用户脚本执行过的次数，新发下作业或重调度恢复后作业脚本启动时MA_PROC_START_CNT会重置到1。当每次发生**原地恢复**或**卡死重启**后，再次执行用户脚本时会在MA_PROC_START_CNT原值基础上加1。 因此当MA_PROC_START_CNT>1 时，说明当前容器已经执行过一次用户脚本，如果业务流程中存在共享内存、数据加载等动作，可用MA_PROC_START_CNT来判断业务是否需要执行重新打开共享内存、跳过数据加载等流程。

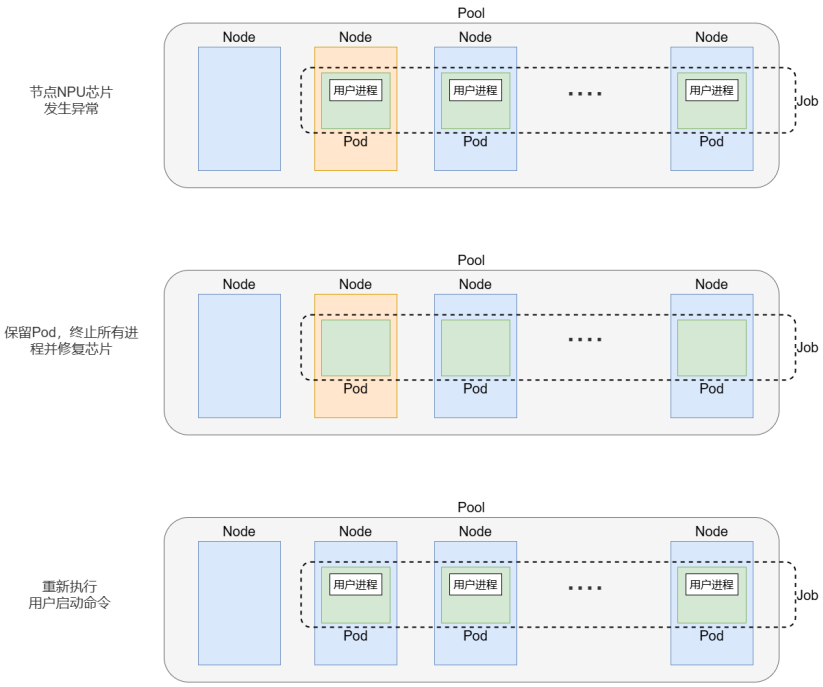
以下环境变量可加速重调度执行的速度。

MA_FAILOVER_TERMINATION_GRACE_PERIOD_SECONDS	当设置为正整数N时，可以在N秒后将容器结束阶段的卷卸载步骤设置为异步执行，从而有效降低大规模作业的重调度恢复时间，建议在SFSTurbo存储为主的场景配置为10。 默认值：-1。

原地恢复

NPU训练作业在运行过程中可能会发生芯片故障，部分芯片故障可通过系统修复或复位进行自愈。对此类可自愈的芯片故障，系统强制停止容器中的用户进程，容器本身不销毁保留运行环境，在所有业务进程终止后尝试NPU芯片自愈，如果芯片故障清除恢复正常则所有容器重新运行训练作业启动命令。原地恢复的过程中不涉及资源调度，仅在原容器中重启训练作业。原地恢复过程如下图所示：

图 12-11 原地恢复



触发场景：

NPU芯片发生可自愈故障。

约束：

- 作业定期保存CKPT。
- 作业支持断点续训。
- 原地恢复保留故障时的容器环境，要求作业脚本可重入，通常需要用户跳过**数据下载、数据预处理步骤，删除和重建同名共享内存**等。可通过**容错恢复环境变量**中的MA_PROC_START_CNT判断是否发生过原地恢复。

降级策略

- 芯片自愈存在自愈失败的场景，当自愈失败时原地恢复同时失败，此时会将该节点隔离，降级到**隔离式Job重调度**
- 一次训练过程中，同一节点同一芯片24小时内连续3次出现相同故障码，此时会将该节点隔离，降级到**隔离式Job重调度**
- 一次训练过程中，同一节点同一芯片24小时内连续3次出现相同故障码，且是80C98002或80CB8002这类由用户输入引发的芯片故障，此时不隔离节点，只执行重调度。

说明

NPU芯片故障码说明：

- **《Atlas A3 中心推理和训练硬件 25.3.RC1 健康管理故障定义》**
- **《Atlas A2 中心推理和训练硬件 25.3.RC1 健康管理故障定义》**
- **《Atlas 中心训练服务器 25.3.RC1 健康管理故障定义》**

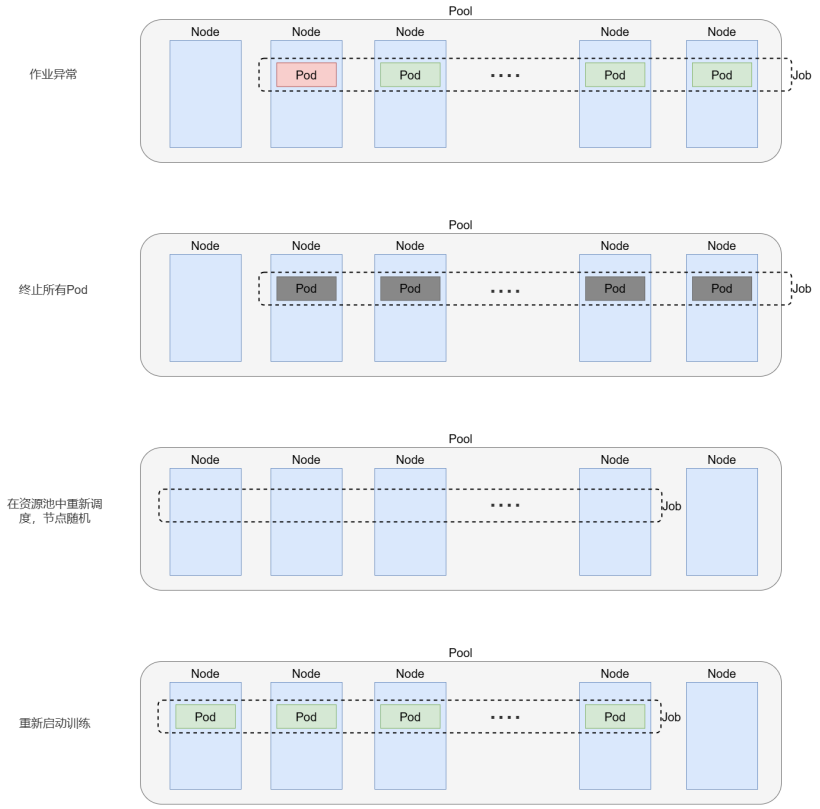
可自愈故障：通常是故障定义中故障级别“次要”“重要”的故障码，“提示”级别不需要处理，“紧急”级别无法自愈。

用户侧引发故障：算子异常或输入的数据导致NPU芯片发生故障，通常需要排查CANN版本、算子实现以及数据正确性。

无条件 Job 级重调度

训练过程中可能会碰到预期外的情况导致训练失败，且无法及时重启训练作业，导致训练周期长，而无条件Job重调度可以避免这类问题，提高训练成功率和提升作业的稳定性。无条件Job重调度是在作业异常结束退出码非0时，将用户作业涉及到的所有Pod终止，并将Job实例完全重建，其过程如下图所示：

图 12-12 无条件 Job 级重调度

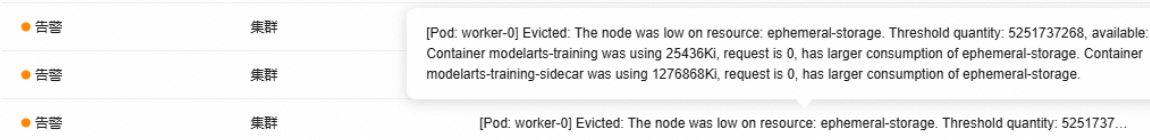


触发场景：

- **原地恢复**中用户输入引发的降级。
- 作业失败中断，且退出码非0。
- **Pod重调度**失败降级。

警告

在作业被驱逐的场景下，目前不支持触发"无条件Job级重调度"，作业状态将变为失败，建议优先排查软件代码相关问题。



约束：

- 作业定期保存CKPT。
- 作业支持断点续训。
- 作业异常时业务能够中断退出，且作业退出码非0。如果作业异常时业务无法中断一直处于运行状态，则无法触发Job重调度。

版权所有 © 华为云计算技术有限公司

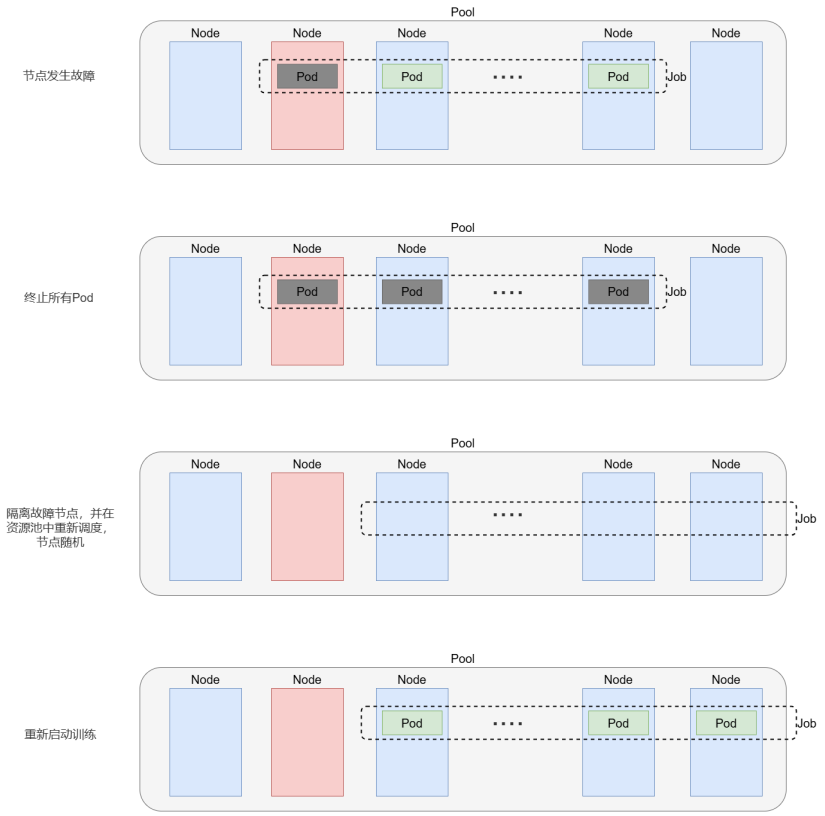
降级策略

连续3次触发无条件Job重调度后，第4次再次出现同样的问题，且中间未发生显式的节点故障和芯片故障，系统默认用户侧代码存在异常，作业将被终止并设置为失败状态。

隔离式 Job 重调度

隔离式Job重调度是在无条件Job重调度策略基础上，如果作业异常时发现节点故障，则在调度前先隔离故障节点，其过程如下图所示：

图 12-13 隔离式 Job 重调度



触发场景：

- 原地恢复中非用户输入引发降级。
- 节点发生故障。

约束：

- 作业定期保存CKPT。
- 作业支持断点续训。

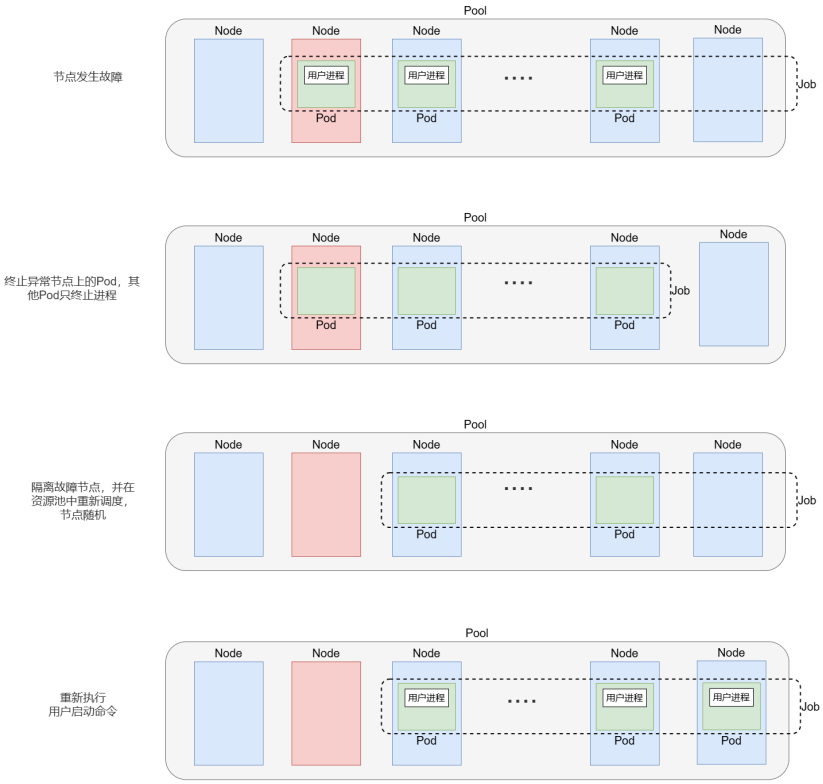
降级策略

无。

Pod 重调度

相比于**隔离式Job重调度**直接删除重建整个Job，Pod重调度中会保留现有的Job实例不变，在调度前先隔离故障节点，且仅重新创建故障Pod。

图 12-14 Pod 重调度



触发场景：

- **原地恢复**中非用户输入引发的降级。
- 节点发生故障。

约束：

- 作业定期保存CKPT。
- 作业支持断点续训。
- Pod重调度保留正常实例的容器环境，要求作业脚本可重入，通常需要用户**跳过数据下载或预处理步骤，删除并重建同名共享内存**等。

降级策略

Pod重调度失败时，由于已经隔离过节点，降级到**无条件Job级重调度**。

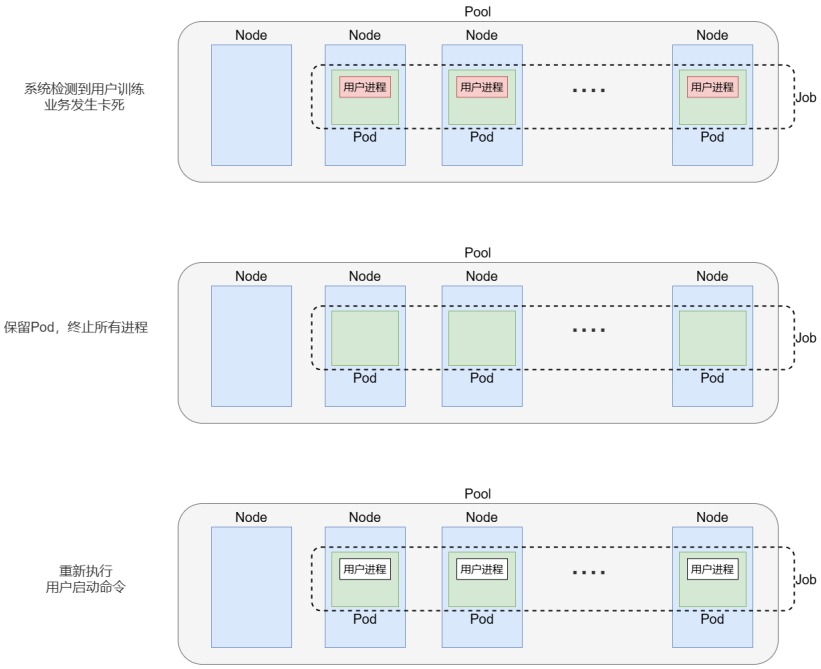
卡死重启

当长稳的训练作业正常运行一段时间后，如果训练作业没有硬件故障，出现卡死时，通常重启训练作业即可恢复正常。但由于训练作业卡死时无法自动结束容器，因此无

法直接触发**Job级重调度**，只能设置作业卡死重启。当训练作业设置为作业卡死重启时，ModelArts会在作业运行过程中检测到卡死状态时，强制停止容器中的用户进程，容器本身不销毁保留运行环境，并在进程停止后重新运行训练作业启动命令。作业卡死重启的过程中不涉及资源调度，仅在原容器中重启训练作业。

卡死检测的规则请参见**训练作业卡死检测**，卡死重启过程如下图所示：

图 12-15 卡死重启



触发场景：

- 系统检测到NPU作业或GPU作业发生卡死。

约束：

- 作业定期保存CKPT。
- 作业支持断点续训。
- 卡死重启保留故障时的容器环境，要求作业脚本可重入，通常涉及**数据下载、数据预处理、创建同名共享内存**等业务逻辑。

降级策略

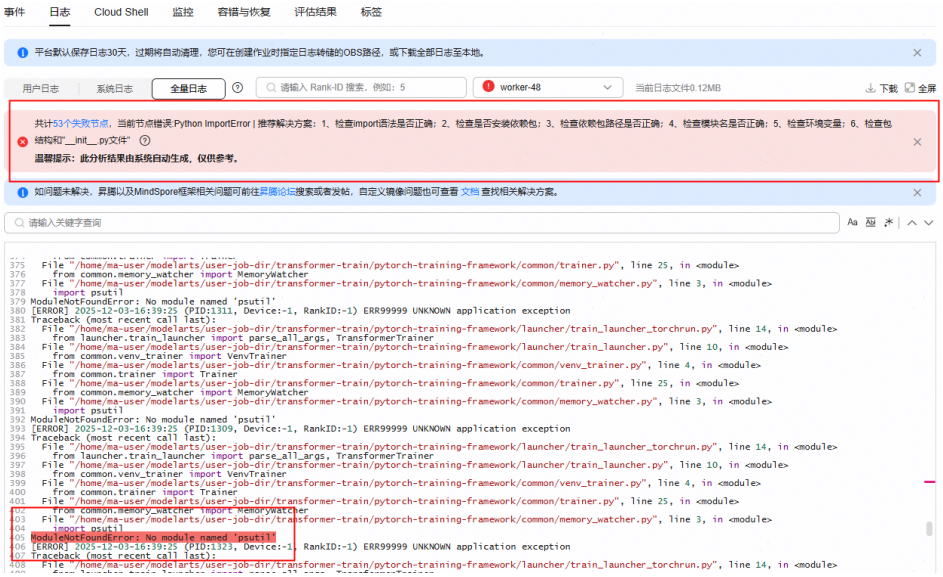
连续3次触发卡死重启后，系统自动将作业终止并置为失败。

12.5 训练日志失败分析

在ModelArts中训练作业遇到问题时，可首先查看日志，多数场景下的问题可以通过日志报错信息直接定位。

ModelArts提供了训练作业失败定位与分析功能，如果训练作业运行失败，ModelArts会自动识别导致作业失败的原因，在训练“全量日志”界面上给出提示。提示包括三部分：失败的可能原因、推荐的解决方案以及对应的日志（底色标红部分）。

图 12-16 训练故障识别



ModelArts会对部分常见训练错误给出分析建议，目前还不能识别所有错误，提供的失败可能原因仅供参考。针对分布式作业，只会显示当前节点的一个分析结果，作业的失败需要综合各个节点的失败原因做一个综合判断。

常见训练问题定位思路如下：

1. 根据日志界面提示中提供的分析建议解决。
   - 参考案例解决：会提供当前故障对应的指导文档链接，请参照文档中的解决方案修复问题。
   - 复制作业：建议复制作业进行重试，大概率能修复问题。
2. 上一步不能解决问题时，可以尝试分析日志中提示的错误信息，定位并解决问题。
3. 也可以前往**ModelArts论坛**中查看是否有同类问题。Ascend训练场景可以前往**昇腾论坛**查看或提问。
4. 最后，如果以上均不能解决问题，可以提工单进行人工咨询。

12.6 配置算子重执行

什么是算子重执行

算子重执行是指当因为通信链路故障导致训练作业中通信算子报错，借助HCCL能力实现失败算子重新执行，保障训练作业不失败。在超节点大模型训练场景，集群通信链路存在闪断的情况，此时通信算子会执行报错，算子重执行可以很好的避免由于链路闪断故障造成的整个作业失败，提升训练作业的通信稳定性。

场景描述

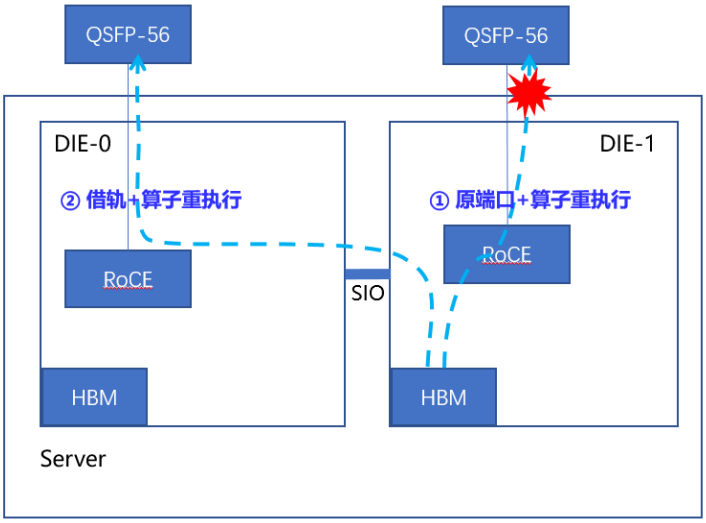
在昇腾新一代硬件Snt9b23构建的超节点集群中，每个超节点内部的服务器的NPU通过灵衢总线实现HCCS协议互联，形成一个高效的“超平面”网络。同时，两个超节点之间的服务器通过“参数面”网络实现互联。然而，运行在超节点集群中的作业需要依赖这两个网络进行分布式通信，而任何一个网络的闪断都会导致通信失败，进而引发业务失败的问题。为了解决这一问题，ModelArts推出了算子重执行的可配置能力，为用户提供了一个可靠的解决方案。

主要涉及“超平面”和“参数面”两种场景网络故障的算子重执行能力：

场景一：“参数面”网络故障

硬件Snt9b32是一张卡两个DIE的硬件结构，而且两个DIE之间是通过SIO快速通道连接，如果DIE-1设备“参数面”链路彻底故障，DIE-1可以通过SIO链路借用DIE-0的“参数面”链路进行通信，开启算子重执行，可以借轨进行通信。如果DIE-1设备“参数面”链路出现闪断，开启算子重执行可以基于现有链路进行通信。

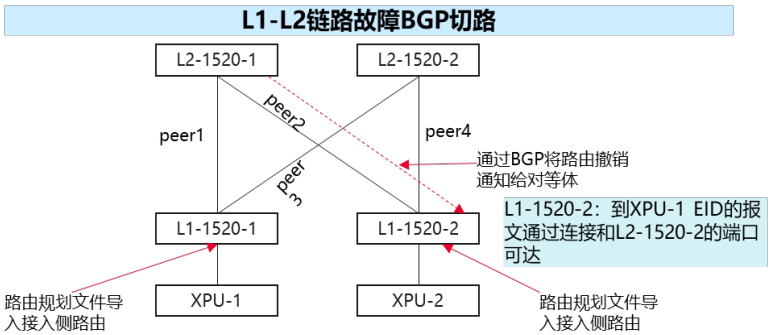
图 12-17 借轨通信



场景二：“超平面”网络故障

昇腾硬件Snt9b23组成的超节点集群中“超平面”网络是通过两层交换机实现NPU互联，如果L1-L2链路故障，可以借助交换机的BGP切路算法进行通信链路重新规划，从而保证算子重执行成功，业务不中断。

图 12-18 BGP 路径切换



约束限制

- 仅支持昇腾Snt9b32超节点的专属资源池。
- 仅支持使用PyTorch框架的预置镜像环境。
- 仅支持使用满卡配置的训练作业。
- 依赖昇腾组件，版本约束为：HDK版本大于等于24.1.RC3.5，CANN版本大于等于8.1.RC1。
- MindSpore框架暂不支持算子重执行。

性能影响说明

警告

只要开启算子重执行，就会带来性能影响且无法关闭！您可以先开启算子重执行评估业务模型性能影响后，再选择是否开启。

开启算子重执行对业务的性能有影响，基于部分开源大模型测试的性能影响如下表所示，具体的影响跟训练模型通信算子有关，如果业务模型性能影响较大，请谨慎开启。

表 12-3 开启算子重执行对业务的性能影响参考（两个超节点测试）

模型类型	用例名	未开启算子重执行时FPS	开启算子重执行时FPS	性能下降
LLama_7B_FastChat	LLama_7B_FastChat_ID4324_for_PyTorch_bs2_2X16P_FA_bf16_RT2_PT21_perf	3629.17	3627.27	-0.05%
BLOOM_ID3985	BLOOM_ID3985_for_PyTorch_bs2048_2x16p_RT2_176B_bf16_FA_perf	7699.445	7659.4295	-0.52%
BLOOM_ID3985（带mc2）	BLOOM_ID3985_for_PyTorch_bs2048_2x16p_RT2_176B_bf16_FA_mc2_perf	7692.59876	7663.907871	-0.37%
LLAMA2_ID4421	LLAMA2_ID4421_for_PyTorch_bs16_2x16p_RT2_7B_bf16_FA_NOMC2_perf	1586	1591.3	0.33%
LLama31_70B	train_llama3_405b_server2_tp8_pp2_perf	6.29	6.29	0.00%
GPT_Moe_8K	gpt_moe_server2_tp4_pp2_perf	20.92	20.89	-0.14%
LLAMA2_70B	LLAMA2_ID0039_for_PyTorch_bs1_2x16p_RT2_70B_bf16_fa_tp2_pp1_perf	2229.5	2219.86	-0.43%

模型类型	用例名	未开启算子重执行时FPS	开启算子重执行时FPS	性能下降
	LLAMA2_ID0039_for_PyTorch_bs1_2x16p_RT2_70B_bf16_fa_tp2_pp2_perf	1218.54	1214.84	-0.30%
	LLAMA2_ID0039_for_PyTorch_bs1_2x16p_RT2_70B_bf16_fa_tp8_pp2_perf	360.59	359.02	-0.44%
	LLAMA2_ID0039_for_PyTorch_bs1_2x16p_RT2_70B_bf16_fa_tp2_pp4_perf	591.95	589.94	-0.34%
Open-Sora_ID4574	Open-Sora_ID4574_for_PyTorch_2x16p_bf16_bs1_120x256x256_perf	9.77	10.02	2.56%
beit2	Beit2_ID3448_for_Pytorch_False_bs120_2x16p_perf	929858.63	935647.44	0.62%
Megablocks	Megablocks_ID4363_for_PyTorch_bs40_32p_mt03_perf	1235.41	1220.49	-1.21%
Qwen_ID4341_for_PyTorch	Qwen_ID4341_for_PyTorch_bs64_32p_tp8_pp1_14b_perf	783.89644	784.993139	0.14%
Baichuan2_13B_ID4446_for_PyTorch	Baichuan2_13B_ID4446_for_PyTorch_RT2_bs8_32p_tp8_pp1_fp16_perf	801.84	802.51	0.08%

开启算子重执行

开启算子重执行有2种方式：控制台设置或API接口设置。

- 控制台设置

  在创建训练作业界面，当训练作业配置满足约束限制的条件时，才能开启（勾选）“算子重执行”。

  开启算子重执行时，在创建训练作业的确认界面和训练作业详情页显示“已开启”。未配置或未开启时不展示此项。

警告

请勿使用配置环境变量HCCL_OP_RETRY_ENABLE的方式开启算子重执行。因为平台组件无法感知是否开启，会导致算子重执行开启失败。

只要开启算子重执行，就会带来性能影响且无法关闭！您可以先开启算子重执行评估业务模型性能影响后，再选择是否开启。

图 12-19 算子重执行-旧版创建训练作业

图 12-20 算子重执行-新版创建训练作业

- API接口设置

  通过API接口创建训练作业时，在“metadata”字段的“annotations”中传入fault-tolerance/hccl_op_retry字段。fault-tolerance/hccl_op_retry可取值为"true"或"false"，当赋值为"true"时，表示开启算子重执行。参数值填写参考表12-4。

表 12-4 参数说明

参数	是否必选	参数类型	描述
kind	是	String	**参数解释**：训练作业类型。 **约束限制**：不涉及。 **取值范围**： ● job：普通作业。 ● edge_job：边缘作业。 ● hetero_job：异构作业。 ● mrs_job：MRS作业。 ● autosearch_job：自动化搜索作业。 ● diag_job：诊断作业。 ● visualization_job：可视化作业。 **默认取值**：job。
annotations	否	Map<String,String>	**参数解释**：训练作业高级功能配置。 **约束限制**：可选取值如下： ● "job_template": "Template RL"（异构作业）。 ● "fault-tolerance/job-retry-num": "3"（故障自动重启次数）。 ● "fault-tolerance/job-unconditional-retry": "true"（无条件重启）。 ● "fault-tolerance/hang-retry": "true"（卡死重启）。 ● "jupyter-lab/enable": "true"（JupyterLab训练应用程序）。 ● "tensorboard/enable": "true"（TensorBoard训练应用程序）。 ● "mindstudio-insight/enable": "true"（MindStudio Insight训练应用程序）。 ● "fault-tolerance/hccl_op_retry":"true"（算子重执行）。

请求参数填写参考如下样例。已开启算子重执行的训练作业，在查询训练作业详情时，也会返回**"fault-tolerance/hccl_op_retry":"true"**的字段。

```
{
    "kind": "job",
    "metadata": {
        "annotations": {
            "fault-tolerance/hccl_op_retry": "true"
        }
```

```
  }
}
```

13 超节点亲和组实例数配置

场景描述

大模型训练中通常会采取多种并行策略，通过分布式计算来提高模型的训练效率和扩展性。其中模型并行中的allreduce通信以及MoE专家并行中的alltoall通信对卡间的互联带宽要求非常高，受限于硬件的组网，这些通信开销较大的阶段往往成为限制训练性能的瓶颈。

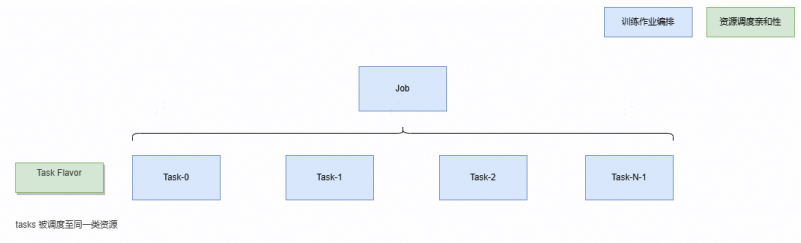
昇腾新一代硬件Snt9b23，使用HCCS总线将多个计算节点的NPU互联进行大带宽互联组成超节点，超节点内的这种全互联的组网形态称为“超平面”，可以极大地提升AI任务的通信效率。基于超节点硬件，模型可以更大范围地调整模型并行或MoE专家并行等并行策略的参数，以利于更大的带宽加速训练过程。

ModelArts在这个基础上，推出了**超节点亲和特性**，它是一种调度策略，通过对AI训练任务进行编排分组来匹配计算资源的硬件组网形态，以充分利用超节点的高带宽和低延迟特性，提升模型训练执行效率。算法工程师只需通过简单配置即可轻松使用超节点硬件。

原理介绍

如图所示，在训练作业中，我们将一个task称为一个实例。为了将训练作业合理地调度到超节点上，并能够充分利用超节点的超高带宽，ModelArts引入了“亲和组”的概念，**亲和组**是训练任务中期望通过超平面连接的一组实例，同一个亲和组的所有实例会被确保调度到同一个超节点内。

图 13-1 训练作业关系



如何配置亲和组来提高训练的效率？核心思想是将训练过程中对带宽要求高的卡放到同一个亲和组内。

版权所有 © 华为云计算技术有限公司

以TP（Tensor Parallel，张量并行）为例，在上一代硬件中，TP通常被设置为8，对应单机8卡的HCCS高速互联。而在超节点中，HCCS互联的设备数增多了，显然，对应的TP域也可扩大。

前提：分布式训练需要每个实例占满节点的所有卡。

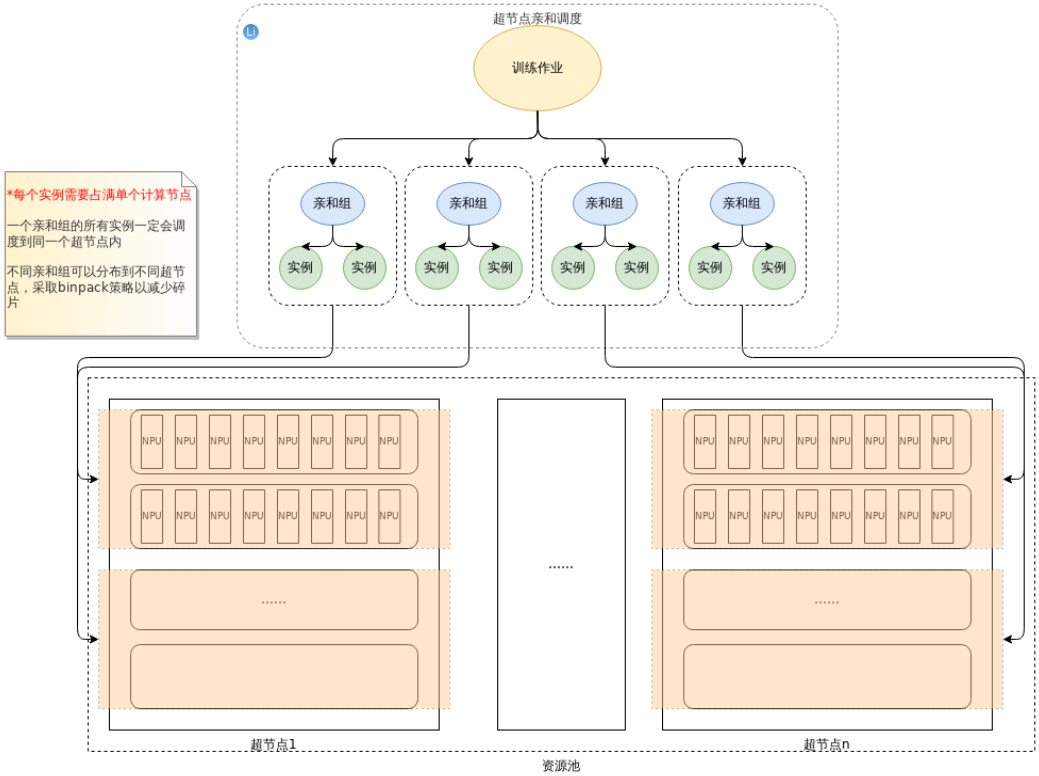
假设TP=32，即训练过程中，每32张卡组成一个TP域，而每个节点有16张卡，那么每2个实例需要确保调度到同一个超节点，我们称这样的2个实例为一个亲和组。

而用户只需在创建训练作业时，填写新增参数“超节点亲和组实例数”=2。

上述的这个训练作业要实际能够调度执行，还需资源池的可用超节点同时满足如下条件：

1. 可用节点数>=训练作业实例数。
2. 所有可用节点可分配的亲和组数>=作业所需亲和组数。

图 13-2 训练作业超节点亲和调度示意图



假设该训练作业总共需要8个实例，超节点亲和组实例数设置为2，当前可选的资源情况如下：

- 资源池A
  超节点1：最大节点数8，可用节点数6
  超节点2：最大节点数8，可用节点数2
- 资源池B
  超节点3：最大节点数8，可用节点数7
  超节点4：最大节点数8，可用节点数1

资源池A和B可用节点数都为8，条件1都满足，但资源池B的超节点3可分配3个亲和组，超节点4只有1个可用节点，可分配0个亲和组，不满足条件2，因此只有资源池A可以调度执行该训练作业。如果该训练作业下发到资源池B，会由于当前可用资源不足排队。

约束限制

- 亲和组特性仅支持昇腾Snt9b23超节点专属资源池。
- 分布式训练需要每个实例占满节点的所有卡。
- 亲和组特性仅支持单步长规格的资源池，不支持多步长规格的资源池。

配置方法

在**ModelArts管理控制台**界面**创建训练作业**时配置，“资源池”选择“专属资源池”，且“节点规格”为Snt9b23系列情况下，支持在**步骤五：资源配置**的阶段配置超节点亲和组实例数。

超节点亲和组实例数参数设置要求如下：

- 超节点亲和组实例数默认为1，最大不超过单个超节点的最大节点数。
- 如果配置了“超节点亲和组实例数”，则“实例数”必须设置为超节点亲和组实例数的整数倍，否则无法创建训练作业。例如，超节点亲和组实例数为n，则实例数为n或2n。
- 在选择专属资源池时，可以查看超节点规格，包括总节点数、总卡数、可用节点数和可用卡数，从而确定超节点亲和组实例数设置。

14 管理模型训练作业

14.1 训练大盘监控

场景描述

在使用ModelArts进行模型训练时，用户需要实时监控训练任务的运行状态，以确保训练过程的顺利进行。然而，有时用户可能需要更详细的数据分析和管理，而不仅仅是实时监控。为此，**ModelArts管理控制台**提供了全面的监控和数据导出功能。通过导航至“模型训练 > 训练作业 > 大盘监控”（旧版控制台），用户可以一站式查看训练作业的概览、健康监测和综合监控情况，实时掌握训练任务的运行状态。此外，用户还可以单击页面上的“导出”按钮，选择需要的监控数据，将其下载到本地，以便进行进一步的分析和管理。通过这些功能，用户不仅能够实时监控训练任务的运行状态，还能获取详细的监控数据，从而更好地管理和优化训练过程。

约束限制

训练大盘监控最多支持查看近1年的监控数据。

训练作业概览

训练作业概览模块提供作业总数、当前资源请求量及各状态作业数量的全局视图，便于快速掌握训练整体状况与资源负荷。

指标名称	说明
作业总数	账号在当前工作空间下所有训练作业的总数，展示整体作业规模。
当前资源请求量（卡）	当前运行中的所有训练作业申请的加速卡总数，体现实时资源需求。
各个作业状态对应的作业数量	分别展示处于不同状态（如“等待中”、“运行中”、“已完成”、“异常/失败”等）的作业数量，用于监控作业健康度与分布。

健康监测

健康检测模块专注于训练作业的稳定性与可靠性管理，通过量化评估作业执行结果和系统容错能力，为运维决策提供关键依据。

指标名称	说明
作业运行成功率	在统计周期内，成功完成的作业数量占全部作业数量的比例。
故障恢复开启率	在统计周期内，容错与恢复功能开启的作业数量占全部作业数量的比例。
作业恢复成功率	在统计周期内，在作业运行出现异常或中断后，作业成功拉起并使用加速卡的事件数量占成功和失败恢复事件数量的比例。

作业综合监控

作业综合监控分为作业故障监控和作业资源消耗两部分。用户可以选择最近7天、最近30天和自定义时间查看对应的监控数据。

作业故障监控

表 14-1 作业故障监控

指标名称	说明
作业失败率	在统计周期内，运行失败的作业数量占全部作业数量的比例。
作业恢复成功率	在统计周期内，在作业运行出现异常或中断后，作业成功拉起并使用加速卡的事件数量占成功和失败恢复事件数量的比例。
作业恢复时长	在统计周期内，在作业运行出现异常或中断后，从发现问题到将作业成功拉起并使用加速卡所经历的时间长度。

作业资源消耗

表 14-2 作业资源消耗

指标名称	说明
资源消耗趋势	在统计周期内，训练作业对各类计算资源的请求量，支持按NPU、GPU筛选。
资源消耗TOP作业	在统计周期内，对各类计算资源请求量最高的训练作业，支持按NPU、GPU、CPU筛选。

14.2 查看训练作业列表和详情（新版）

场景描述

说明

本章节内容仅适用于“西南-贵阳一”站点。控制台界面为新版样式。

在使用ModelArts平台的模型训练功能时，您可能需要查看当前正在进行的训练作业列表及其详细信息，以确保训练过程顺利进行。

通过作业列表页面，您可以轻松掌握每个作业的状态，调整列表展示内容，根据需要的属性类型进行筛选过滤，快速查找训练作业。

单击作业名称，进入查看精调作业详情页或查看自定义作业详情页，查看详细信息。

查看训练作业列表

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“模型开发与训练 > 模型训练”，进入“训练作业”列表。
   - 在列表界面可以查看作业名称/ID、状态、优先级、训练模式、进度、运行时长、标签、创建时间、创建者和描述等信息。部分列支持筛选或排序查看。

     单击作业搜索框右侧的 可对作业列表展示内容进行设置和调整。
   - 用户可以通过选择“全部”或者“我创建的”和最近天数（或自定义）快速过滤可见的作业列表范围。
   - 在训练作业列表上方的搜索框中，根据您需要的属性类型，例如名称、ID、状态、训练模式等进行筛选过滤，快速查找训练作业。
   - 可以在列表页最右侧的“操作”列，对作业进行克隆、终止或删除等操作。
3. 在“训练作业”列表中，根据“训练模式”列展示的“精调-微调”或“自定义”进行作业筛选。单击作业名称，进入查看精调作业详情页或查看自定义作业详情页，查看详细信息。

查看精调作业详情

在“模型训练”列表中，筛选“训练模式”为“**精调-微调**”单击作业名称，进入精调训练作业详情页。模型训练详情包含**模型产出**、**任务详情**、**事件**、**日志**、**资源占用**标签页。模型详情页面右上角支持根据模型不同状态对精调任务做“**克隆**”、“**删除**”、“**重试**”等操作。

精调作业详情页展示的分类和功能介绍如下：

- **模型产出**

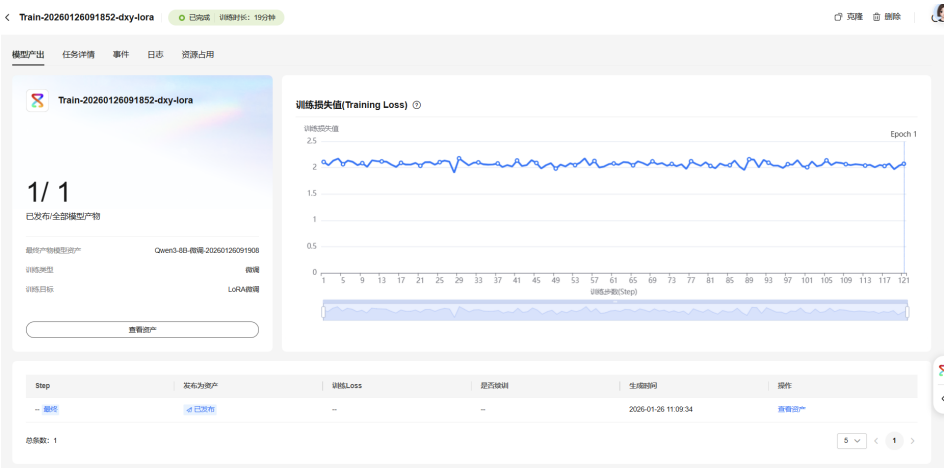
  模型产出支持查看模型如下参数：

  训练类型及训练目标：查询当前精调任务训练类型及训练目标。

  训练损失值：展示训练损失值坐标，通过坐标可观测训练模型是否达到要求。

  模型资产信息：模型精调完成后，需要发布为资产才能在后续使用该模型。发布后的模型可以通过“查看资产”可跳转至精调后的模型详情页面。对于相同模型的不同版本，支持在**模型详情**页面查看发布为资产的不同版本。

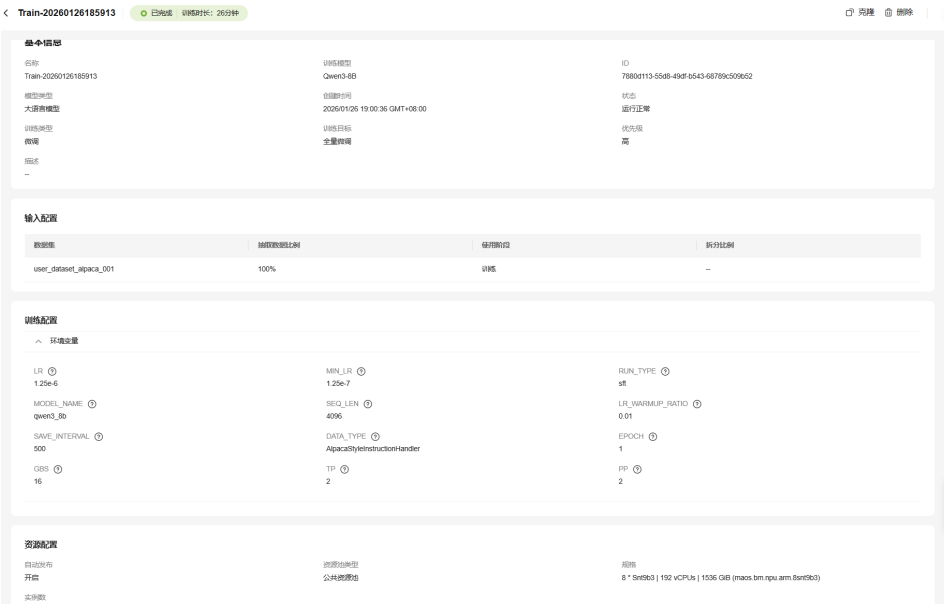
图 14-1 模型产出



- **任务详情**

  任务详情支持查看模型精调任务时的配置信息。包含基本信息、数据集信息、训练配置、资源配置、以及工作流 节点详情，具体配置信息可以参考**步骤二：配置精调参数**章节的参数说明。

图 14-2 任务详情

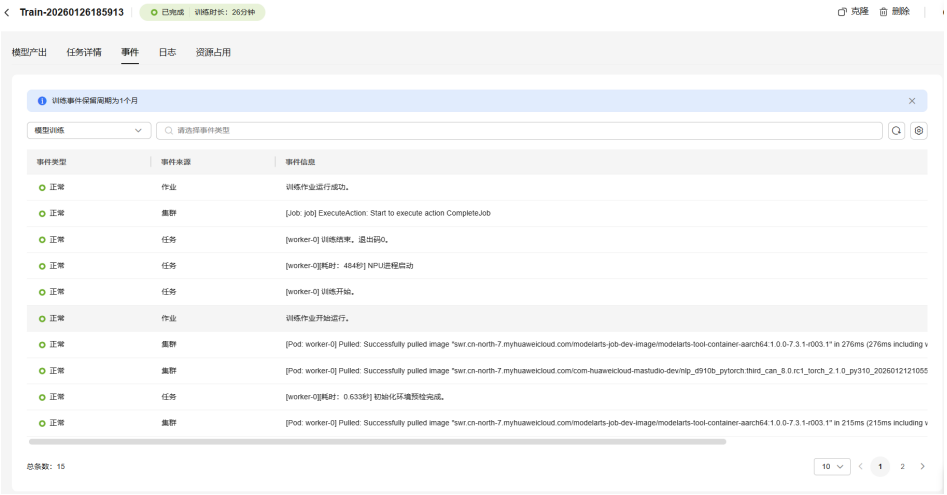


- **事件**

  事件包含精调任务运行时的各种事件上报，便于用户了解模型精调的各个阶段是否有异常。

版权所有 © 华为云计算技术有限公司

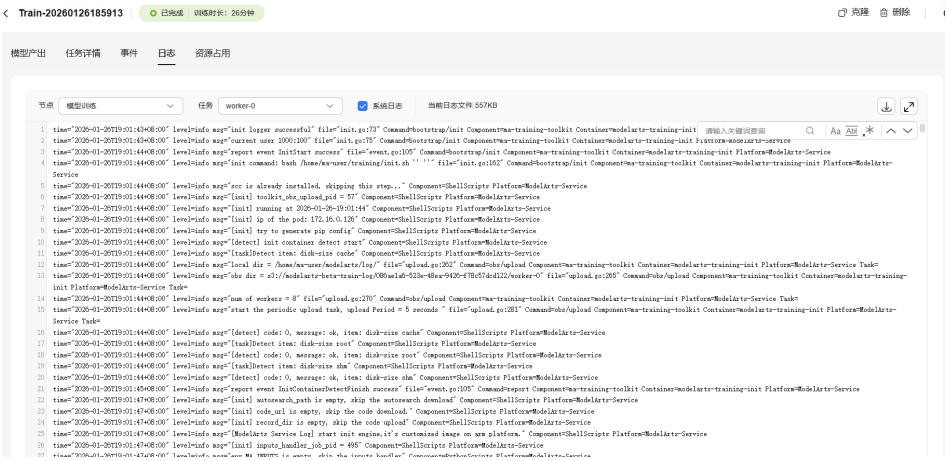
图 14-3 事件



- **日志**

  日志包含精调任务运行时的关键，便于用户了解模型精调的各个阶段是否有异常，以及增强问题定位的手段。

图 14-4 日志



- **资源占用**

  用于统计任务运行期各项指标，包含CPU、NPU、文件系统各项的使用情况。

图 14-5 资源占用

版权所有 © 华为云计算技术有限公司

查看自定义作业详情

在“训练作业”列表中，筛选“训练模式”为“自定义”单击作业名称，进入自定义训练作业详情页。

自定义作业详情页展示的分类和功能介绍如下：

表 14-3 自定义作业详情页展示的分类和功能

分类	说明
任务详情	作业的基本信息。
事件	• 在训练详情页，支持管理训练作业的事件通知。 • 查看训练作业事件 在训练作业的整个生命周期中，从用户可见的开始阶段起，系统后台会记录每一个关键事件点，用户可以随时在对应训练作业的详情页面查看这些记录。这样，用户能够清晰地了解训练作业的进展和状态，确保信息的透明度和可追溯性。
日志	训练日志记录了训练作业的运行过程和异常信息，帮助用户快速定位作业运行中出现的问题。用户代码中的标准输出和标准错误信息都会在训练日志中显示。
Cloud Shell	ModelArts提供了Cloud Shell，可以登录运行中的容器，用于调试生产环境的训练作业。
监控	用户可以通过监控功能查看训练作业资源占用情况和查看训练作业监控指标，快速掌握训练作业运行情况。 • 查看训练作业资源占用情况 当前ModelArts训练作业模块在训练作业详情页的“监控”页签中提供了训练作业的资源使用情况监控，可以查看当前训练作业整体或单个节点占用的CPU、GPU或NPU资源使用情况。 • 查看训练作业监控指标 在训练作业运行过程中，如果用户能在模型训练出问题（例如loss值异常）的情况下能收到告警并及时处理，可以节省大量时间和资源，避免无效运行作业导致的浪费。同时通过指标监控可以实时掌握训练作业的进度，了解模型在不同阶段的训练状态。
智能运维	训练作业运行过程中，ModelArts平台会对训练作业进行全方位的实时监测，确保作业的正常运行。训练作业详情中提供智能运维功能，便于用户对作业的监测和运维。
评估结果	训练作业运行结束后，ModelArts可为您的模型进行评估，并且给出调优诊断和建议。
标签	通过给训练作业添加标签，可以标识云资源，便于快速搜索训练作业。

训练作业任务详情信息

自定义作业详情页的“任务详情”展示作业的基本信息。

表 14-4 任务详情信息

分类	参数	说明
基本信息	任务名称	训练任务的名称。
	任务ID	训练作业唯一标识。
	作业状态	● 训练作业状态。 ● 分为：已完成、等待中、运行中、创建中、终止中、已终止、运行失败、异常、删除中。 **说明** ● 如果昇腾以及MindSpore框架的训练作业运行失败，您可以在作业状态的提示信息中，单击**昇腾论坛**进行发帖提问或者搜索问题。 ● 非昇腾规格的作业运行失败后，您可以在作业状态的提示信息中，单击**ModelArts开发者论坛**进行发帖或者搜索已有问题。
	创建时间	记录训练作业创建时间。
	运行时长	记录训练作业运行时长，是训练作业全生命周期中多次的k8s资源运行的时长总和。
	描述	训练作业的描述。 未配置时显示“--”，您可以单击 图标，更新训练作业的描述。
训练配置	镜像类型	训练作业所选择的镜像类型。 预置镜像或自定义镜像。
	镜像	训练作业所选择的镜像名称。
	镜像地址	镜像所在SWR地址。
	代码目录	训练作业代码目录所在的OBS路径。 未配置时显示“--”。 您可以单击 图标，更新代码。
	本地代码目录	训练代码在训练容器中的存放路径。
	启动命令	镜像的启动命令。
	环境变量	训练作业设置的环境变量。
资源配置	资源类型	训练作业所选择资源池的类型。专属资源池或公共资源池。
	资源池	训练作业所选择的资源池名称。

分类	参数	说明
	实例规格	训练作业使用的实例规格信息。 此处显示的是本次训练作业实际分配给训练容器的实例规格，以及创建训练作业时选择的“实例规格”。 ● 目标规格：是指用户在创建训练作业时配置或选择的资源规格，表示作业期望使用的计算资源总量，包括 CPU、内存等。 ● 实际规格：是指训练作业运行时平台实际分配给训练容器的资源配置。在目标规格基础上，平台会为操作系统、Kubernetes 系统组件及资源池插件预留必要资源，因此用户实际可用资源通常小于目标规格。
	实例数	训练作业设置的实例数。
	计算节点 ID	默认展示当前计算节点个数。 单击展示训练作业使用的计算节点名称及其IP地址，仅当训练作业使用专属资源池时可见。
	作业调度优先级	● 使用专属资源池创建的训练作业，展示作业优先级。使用公共资源池创建的训练作业，不展示此参数。 ● 作业默认按照优先级从高到低依次处理，对于相同优先级的任务，则根据提交时间的先后顺序进行调度，最终结合实际的资源情况决定调度顺序(保证在资源充足、相同优先级下，先提交的任务先调度)。 ● 作业优先级数字为1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。 ● 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。
	允许被抢占	● 使用专属资源池创建训练作业时，允许设置训练作业被抢占。使用公共资源池时，不展示此参数配置。 ● 开启后，当资源池剩余资源不足时，允许被抢占的作业可能会被终止并重新排队。为避免训练进度丢失，请在开启此功能前完成断点续训的设置，操作指导请参考**设置断点续训练**。 ● 未设置时显示“已关闭”。
数据配置	训练集	本次训练作业所选用的数据集名称。 未配置或未开启时显示“未开启”。
高可用配置	容错与恢复	● 记录训练作业的出现故障后自动重启的次数，仅当创建训练作业时开启“容错与恢复”功能时可见。 ● 此处显示最大重启次数和已重启次数。 ● 未配置或未开启时显示“未开启”。
	无条件自动重启	“容错与恢复”开启后显示。 创建作业时开启无条件自动重启时显示“已开启”。 未配置或未开启时显示“未开启”。

分类	参数	说明
	作业卡死重启	“容错与恢复”开启后显示。 创建作业时开启作业卡死重启时显示“已开启”。 未配置或未开启时显示“未开启”。
发布到资产	发布到资产	开启后显示“已开启”。 系统自动将模型产物发布成模型资产，可以在平台进行推理、评测等操作。 未配置或未开启时显示“未开启”。
	模型输出路径	用“|”分隔显示“存储路径”和“云上挂载路径”。 ● 存储位置：模型训练完成后存储位置。 ● 云上挂载路径： 用户使用专属资源池时显示此参数。 系统将存储位置中的文件目录挂载到训练容器中指定路径。用户可自定义该路径，不支持配置为/home/、/home/ma-user/、/home/ma-user/modelarts/等系统目录。 未配置显示“--”。
	自动发布到资产	开启后显示“已开启”。 训练后的模型将自动上传至“资产管理-我的模型”。 未配置或未开启时显示“未开启”。
	模型名称	“自动发布到资产”开启时显示。 设置新模型的名称。 支持2-128位，只包含中文、字母、数字、中划线、下划线，且以中文、字母开头，以中文、字母、数字结尾。
	模型类型	“自动发布到资产”开启时显示。 根据发布的模型类型选择。
	模型品牌	“自动发布到资产”开启时显示。 模型品牌。
	模型版本号	“自动发布到资产”开启时显示。 发布为新模型时，版本号为V1。 发布为已有模型新版本时，版本号在该模型上一个版本号自动加1。 **注意：**模型版本号不支持修改，为系统自动生成。
	描述	“自动发布到资产”开启时显示。 训练后模型的描述信息。该字段可选，最大长度256字符。
访问配置	JupyterLab	显示训练作业的JupyterLab地址，仅勾选JupyterLab训练应用程序的调试训练作业才有该参数。 未配置时显示“未开启”。

分类	参数	说明
	SSH远程开发	显示训练作业SSH远程开发的密钥对和SSH地址，仅启用SSH远程开发的调试训练作业才有该参数。 未配置时显示“未开启”。
	生成实例间SSH免密互信文件	显示训练作业配置的生成实例间 SSH 免密互信文件信息。 未配置时显示“未开启”。
可观测配置	Tensorboard	TensorBoard是TensorFlow的可视化工具包，提供机器学习实验所需的可视化功能和工具。TensorBoard能够有效地展示训练过程中的计算图、各种指标随时间的变化趋势以及训练中使用到的数据信息，相关概念请参考**TensorBoard官网**。 使用公共资源池时，不展示此参数配置。 显示“已开启”，同时展示配置的存储路径。 未配置时显示“未开启”。
	MindStudio Insight	MindStudio Insight能可视化展现出训练过程中的标量、图像、计算图以及模型超参等信息，支持基于MindSpore引擎的训练作业。MindStudio Insight相关概念请参考**MindSpore官网**。 使用公共资源池时，不展示此参数配置。 显示“已开启”，同时展示配置的存储路径。 未配置时显示“未开启”。
	监控指标对接AOM	● http获取指标：已配置此参数的训练作业展示已配置的采集URL和采集端口。 ● 命令行获取指标：已配置此参数的训练作业展示已配置的执行命令和执行命令参数。 未配置时显示“未开启”。
更多配置	永久保存日志	配置“日志路径”后展示配置的路径。 未配置时显示“未开启”。
	作业可见范围	分为“工作空间内可见”和“仅创建者可见”。 ● “工作空间内可见”：创建的训练作业在当前工作空间内，默认其他子用户均可看到该训练作业。 ● “仅创建者可见”：默认创建者可见，其他人希望查看该作业需要额外申请“modelarts:trainJob:listAll”权限（查看训练作业列表（包含仅创建者可见的作业）权限）。
	自动停止	显示本次训练作业配置的自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。 启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。 未配置时显示“未开启”。

分类	参数	说明
	事件通知	展示创建训练作业时配置的事件通知主题和通知事件。 未配置时显示“未开启”。

- 在训练详情页，支持管理训练作业的事件通知。

  说明

  - 作业状态为“已完成”“运行失败”“异常”和“已终止”的训练作业不支持在详情中配置事件通知。
  - 拥有查看列表中作业的权限才能配置事件通知。
  - 如果是修改事件类型，只会通知修改成功之后的训练状态变化。

  配置事件通知后，在训练作业发生特定事件（如作业状态变化或者疑似卡死）后会发送通知（短信邮件等），发送通知涉及少量费用，详情查看消息通知服务计费说明。

  – 如果训练作业已经启用事件通知，则可以单击“已配置”右侧的，修改或关闭事件通知。

  – 如果训练作业未启用事件通知，则可以单击“未配置”右侧的，启用并配置事件通知。

  表 14-5 事件通知的参数说明

参数名称	说明
主题名	事件通知的主题名称。可以在下拉框中选择合适的主题名称，也可以单击“立即创建”，前往消息通知服务中新建主题。 **说明** 在消息通知服务中创建主题，并为该主题添加订阅，确认该订阅状态为“已确认”后，方可收到事件消息提醒。
事件	选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”、“作业终止”、“作业疑似卡死”等。 **说明** 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。

14.3 查看训练作业列表和详情

场景描述

说明

本章节内容适用除“西南-贵阳一”站点之外的所有华为云站点。控制台界面为旧版样式。

在使用ModelArts平台的模型训练功能时，您可能需要查看当前正在进行的训练作业列表及其详细信息，以确保训练过程顺利进行。

通过作业列表页面，您可以轻松掌握每个作业的状态，调整列表展示内容，根据需要的属性类型进行筛选过滤，快速查找训练作业，并将训练作业导出为Excel表到本地。

单击具体作业，进入详情页后，您可以查看作业流程、具体配置信息，以及该作业的事件、日志监控、评估结果和标签等信息。此外，您还可以在详情页进行日志的查看和导出，通过Cloud Shell登录查看作业，或通过智能运维功能查看和运维作业。

查看方式

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型训练 > 训练作业”（旧版控制台），进入“训练作业”列表。
   - 在作业列表，单击“导出”，可以将训练作业根据时间周期导出Excel表到本地。最多只支持导出前200行数据。
   - 单击作业搜索框右侧的 可对作业列表展示内容进行设置和调整。
   - 在训练作业列表上方的搜索框中，根据您需要的属性类型，例如名称、ID、状态、作业模式、作业类型、算法、资源池ID、作业优先级、描述、创建者等进行筛选过滤，快速查找训练作业。
3. 在“训练作业”列表中，单击作业名称，进入训练作业详情页。
4. 在训练作业详情页的左侧，可以查看此次训练作业的基本信息和算法配置的相关信息。
   - **训练作业基本信息**

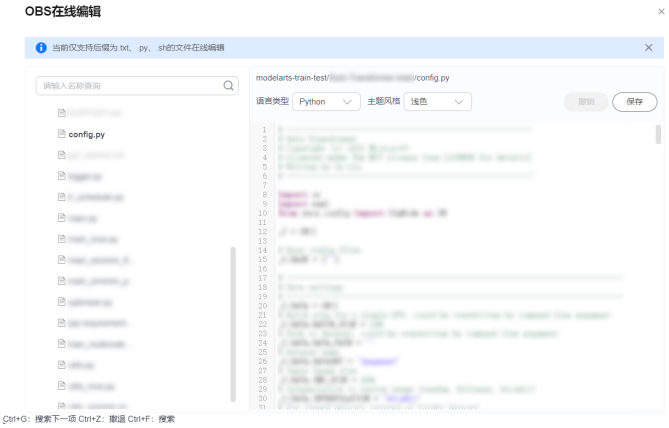
表 14-6 训练作业基本信息

参数	说明
“作业ID”	训练作业唯一标识。
“作业状态”	● 训练作业状态。 ● 分为：已完成、等待中、运行中、创建中、终止中、已终止、运行失败、异常、删除中。 **说明** ● 如果昇腾以及MindSpore框架的训练作业运行失败，您可以在作业状态的提示信息中，单击**昇腾论坛**进行发帖提问或者搜索问题。 ● 非昇腾规格的作业运行失败后，您可以在作业状态的提示信息中，单击**ModelArts开发者论坛**进行发帖或者搜索已有问题。
“所属实验”	● 创建训练作业时配置“实验设置”的作业显示所属实验的名称。 ● 单击可进入对应的实验作业列表。
“创建时间”	记录训练作业创建时间。
“运行时长”	记录训练作业运行时长，是训练作业全生命周期中多次的k8s资源运行的时长总和。

参数	说明
“重启次数”	● 记录训练作业的出现故障后自动重启的次数，仅当创建训练作业时开启“自动重启”功能时可见。 ● 此处显示“当前重启次数/最大重启次数”。
“无条件自动重启”	“自动重启”开启后显示。 创建作业时开启无条件自动重启时显示“已开启”。 未配置或未开启时显示“未配置”。
“作业卡死重启”	“自动重启”开启后显示。 创建作业时开启作业卡死重启时显示“已开启”。 未配置或未开启时显示“未配置”。
“描述”	训练作业的描述。 未配置时显示“--”，您可以单击 图标，更新训练作业的描述。
“作业优先级”	● 使用专属资源池创建的训练作业，展示作业优先级。使用公共资源池创建的训练作业，不展示此参数。 ● 作业默认按照优先级从高到低依次处理，对于相同优先级的任务，则根据提交时间的先后顺序进行调度，最终结合实际的资源情况决定调度顺序(保证在资源充足、相同优先级下，先提交的任务先调度)。 ● 作业优先级数字为1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。 ● 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。
“算子重执行”	开启算子重执行时显示“已开启”。 未配置或未开启时不展示此项。
“允许被抢占”	● 使用专属资源池创建训练作业时，允许设置训练作业被抢占。使用公共资源池时，不展示此参数配置。 ● 开启后，当资源池剩余资源不足时，允许被抢占的作业可能会被终止并重新排队。为避免训练进度丢失，请在开启此功能前完成断点续训的设置，操作指导请参考**设置断点续训练**。 ● 未设置时显示“已关闭”。
“作业可见范围”	分为“工作空间内可见”和“仅创建者可见”。 ● “工作空间内可见”：创建的训练作业在当前工作空间内，默认其他子用户均可看到该训练作业。 ● “仅创建者可见”：默认创建者可见，其他人希望查看该作业需要额外申请“modelarts:trainJob:listAll”权限（查看训练作业列表（包含仅创建者可见的作业）权限）。

– **训练作业参数**

表 14-7 训练作业参数

参数	说明
“作业模式”	显示本次训练作业的模式，是调试模式还是生产模式。
“预置镜像”	本次训练作业使用的预置镜像框架。仅使用预置框架创建的训练作业才有该参数。
“自定义镜像”	本次训练作业使用的自定义镜像。仅使用自定义镜像创建的训练作业才有该参数。
“代码目录”	训练作业代码目录所在的OBS路径。 您可以单击代码目录后的“编辑代码”，在“OBS在线编辑”对话框中实时编辑训练脚本代码。当训练作业状态为“等待中”、“创建中”和“运行中”时，不支持“OBS在线编辑”功能。  **说明** 当您使用订阅算法创建训练作业时，不支持该参数。
“启动文件”	训练作业启动文件位置。 **说明** 当您使用订阅算法创建训练作业时，不支持该参数。
“启动命令”	镜像的启动命令。创建训练作业时，当“启动方式”选择“预置框架”时不涉及该参数；当“启动方式”选择“自定义”时，展示该参数，训练作业详情页面展示该参数和对应的值。
“运行用户ID”	容器运行时的用户ID。
“算法名称”	● 本次训练作业使用的算法。单击算法名称，可以跳转至算法详情页面。 ● 未配置时显示“--”。
“本地代码目录”	训练代码在训练容器中的存放路径。

参数	说明
“工作目录”	训练启动文件在训练容器中的路径。
“实例数”	本次训练作业设置的实例数。
“专属资源池”	专属资源池信息，仅当训练作业使用专属资源池时可见。
“计算节点信息”	展示训练作业使用的计算节点名称及其IP地址，仅当训练作业使用专属资源池时可见。
“实例规格”	● 本次训练作业使用的实例规格信息，仅当训练作业未配置专属资源池的自定义规格时可见。 ● 此处显示的是本次训练作业实际分配给训练容器的实例规格，以及创建训练作业时选择的“实例规格”。实际分配的资源一般小于创建时选择的资源，因为作业内部容器会占用一部分资源，这些容器用于支持训练作业的正常运行。
“自定义规格”	● 本次训练作业使用的实例规格信息，仅当训练作业配置了专属资源池的自定义规格时可见。 ● 此处显示的是创建训练作业时自定义的资源规格，以及创建训练作业时选择的“实例规格”。
“作业日志路径”	● 创建训练作业时，勾选开启“永久保存日志”，配置“日志路径”后展示配置的路径。未勾选开启“永久保存日志”时不展示该参数。 ● 单击路径跳转配置的路径所在目录。
“事件通知”	● 展示创建训练作业时配置的事件通知主题和通知事件。 ● 未配置时显示“未配置”。
“输入-输入路径”	本次训练中，输入数据的OBS路径。
“输入-参数名称”	算法代码中，输入路径指代的参数。
“输入-获取方式”	本次训练作业的输入采用的获取方式。
“输入-容器内访问路径”	训练启动后，ModelArts将OBS路径中的数据下载至后台容器，本地路径指ModelArts后台容器中存储输入数据的路径。
“输出-输出路径”	本次训练中，输出数据的OBS路径。
“输出-参数名称”	算法代码中，输出路径指代的参数。
“输出-获取方式”	本次训练作业的输出采用的获取方式。

参数	说明
“输出-容器内访问路径”	ModelArts后台容器中存储训练输出的路径。
“超参”	本次训练作业使用的超参。
“环境变量”	本次训练作业设置的环境变量。

5. 在训练详情页，支持管理训练作业的事件通知。

说明

- 作业状态为“已完成”“运行失败”“异常”和“已终止”的训练作业不支持在详情中配置事件通知。
- 拥有查看列表中作业的权限才能配置事件通知。
- 如果是修改事件类型，只会通知修改成功之后的训练状态变化。

配置事件通知后，在训练作业发生特定事件（如作业状态变化或者疑似卡死）后会发送通知（短信邮件等），发送通知涉及少量费用，详情查看消息通知服务计费说明。

– 如果训练作业已经启用事件通知，则可以单击“已配置”右侧的，修改或关闭事件通知。

图 14-6 修改事件通知

– 如果训练作业未启用事件通知，则可以单击“未配置”右侧的，启用并配置事件通知。

图 14-7 配置事件通知

表 14-8 事件通知的参数说明

参数名称	说明
主题名	事件通知的主题名称。可以在下拉框中选择合适的主题名称，也可以单击“立即创建”，前往消息通知服务中新建主题。 **说明** 在消息通知服务中创建主题，并为该主题添加订阅，确认该订阅状态为“已确认”后，方可收到事件消息提醒。
事件	选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”、“作业终止”、“作业疑似卡死”等。 **说明** 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。

14.4 训练作业流程可视化

场景描述

在机器学习训练过程中，用户通常需要监控训练作业的执行状态。然而，传统的监控方式往往难以提供全面的作业运行信息，导致用户无法及时掌握训练进度。通过作业流程可视化功能模块，用户可以实时查看训练作业的详细运行状态。该功能支持多维度的信息展示，包括作业调度情况、环境准备状态以及作业执行进度等关键环节。用户只需进入可视化界面，即可一目了然地掌握训练作业的全生命周期信息，从而更精准地进行进度跟踪和参数调整，有效提升训练效率和用户体验。

约束限制

作业流程主阶段信息固定，但子阶段信息不固定，例如“环境准备”中的子阶段“训练输入下载中”，若训练作业无输入，则该子阶段不存在；例如若“初始化环境预检中”事件丢失，则“作业运行”中的子阶段“初始化环境预检中”子阶段不存在。

版权所有 © 华为云计算技术有限公司

训练作业流程可视化查看

当前ModelArts训练作业模块在训练作业详情页的顶部，展示了作业流程详情，共有四个主阶段信息：作业调度 > 环境准备 > 作业运行 > 作业结束。

每个主阶段，可以查看具体子阶段流程信息。

- 作业调度：记录作业创建情况（成功与失败），作业调度情况，发生的日期与时间等信息。
- 环境准备：记录环境准备的相关内容，例如作业环境开始准备、初始化环境预检、训练代码下载中、作业环境准备完成等关键步骤的发生的日期与时间等信息。
- 作业运行：记录作业运行相关内容，例如训练作业开始启动、训练作业执行完成等关键步骤的发生的日期与时间等信息。
- 作业结束：记录作业运行结果等关键步骤的发生的日期与时间等信息。

作业流程右上部，单击“历史记录”，可以查看该作业的所有流程记录，例如发生多次抢占，作业重新调度，作业流程多次重新开始，则可以查看每次作业运行的具体流程信息。

图 14-8 作业流程

14.5 查看训练作业事件

场景描述

在训练作业的整个生命周期中，从用户可见的开始阶段起，系统后台会记录每一个关键事件点，用户可以随时在对应训练作业的详情页面查看这些记录。这样，用户能够清晰地了解训练作业的进展和状态，确保信息的透明度和可追溯性。

事件列表

方便用户更清楚地了解训练作业运行过程，遇到任务异常时，更加准确地排查定位问题。当前支持的作业事件如下所示：

- 训练作业创建成功
- 训练作业创建失败报错
- 准备阶段超时。可能原因是跨区域算法同步或者创建共享存储超时
- 训练作业已排队，正在等待资源分配
- 训练作业排队失败

- 训练作业开始运行
- 训练作业运行成功
- 训练作业运行失败
- 训练作业被抢占
- 系统检测到您的作业疑似卡死，请及时前往作业详情界面查看并处理
- 训练作业已重启
- 训练作业已被手动终止
- 训练作业已被终止（最大运行时长：x h）
- 训练作业已被手动删除
- 计费信息同步结束
- [worker-0] [耗时： 秒] 初始化环境预检完成
- [worker-0] [耗时： 秒] 检测项预检未通过，异常：
- [worker-0] [耗时： 秒] 检测项预检未通过，错误：
- [worker-0] [耗时： 秒] 训练代码下载完成
- [worker-0] [耗时： 秒] 训练代码下载失败，失败原因：
- [worker-0] [耗时： 秒] 训练输入（参数名称： ）下载完成
- [worker-0] [耗时： 秒] 训练输入（参数名称： ）下载失败，失败原因：
- [worker-0] [耗时： 秒] 训练输出（参数名称： ）上传完成
- [worker-0] [耗时： 秒] 训练输出（ ）预取成功
- [worker-0] [耗时： 秒] 训练预启动脚本执行完成
- [worker-0] [耗时： 秒] Python依赖包安装完成
- [worker-0] rankTable训练加速库安装结束
- [worker-0] modelarts-turbo 训练加速库安装结束
- [worker-0] turbo训练加速库安装结束
- [worker-0] 训练discover库安装结束
- [worker-0] 训练输出和日志上传进程退出，文件将不再同步
- [worker-0] 训练容器心跳检测超时
- [worker-0] 训练作业开始运行
- [worker-0] 训练开始
- [worker-0] 训练结束，退出码
- [worker-0] [耗时： 秒] 训练输出（参数名称： ）上传完成

训练运行到结束的过程中，关键事件支持手动/自动刷新。

约束限制

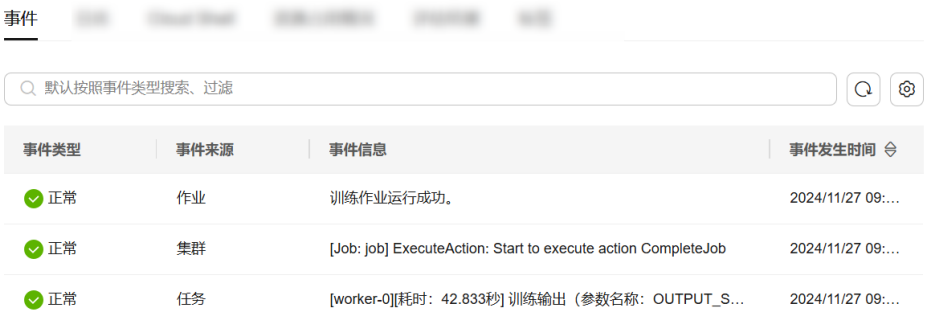
训练作业的事件信息系统会自动保存30天，过期会被清除。

查看操作

1. 在**ModelArts管理控制台**的左侧导航栏中选择“模型开发与训练 > 模型训练”（旧版控制台：“模型训练 > 训练作业”）。

2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“事件”页签查看事件类型、事件来源、事件信息、事件发生时间等信息。

图 14-9 查看事件信息



14.6 查看训练作业日志

场景描述

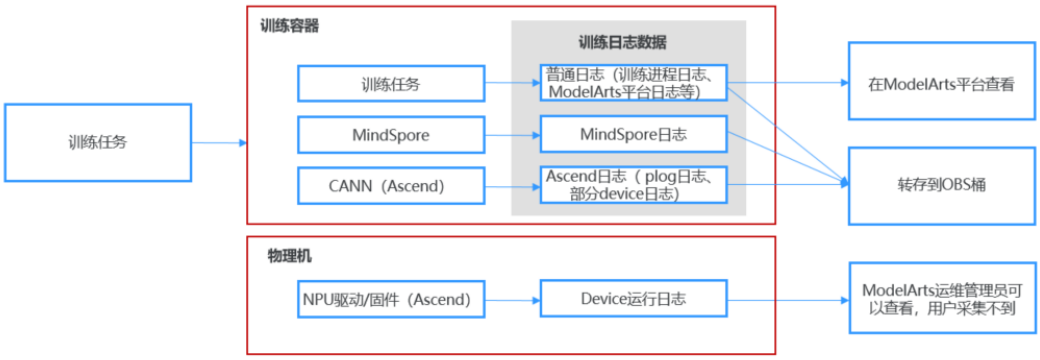
训练日志记录了训练作业的运行过程和异常信息，帮助用户快速定位作业运行中出现的问题。用户代码中的标准输出和标准错误信息都会在训练日志中显示。

当在 ModelArts 中遇到训练作业问题时，用户可以首先查看日志，大多数情况下，问题可以通过日志中的报错信息直接定位。

训练日志包括普通训练日志和Ascend相关日志。

- **普通日志**：当使用Ascend之外的资源训练时仅产生普通训练日志，普通日志中包含训练进程日志、pip-requirement.txt安装日志和ModelArts平台日志。
- **Ascend场景日志**：使用Ascend资源训练时会产生device日志、plog日志、proc log单卡训练日志、MindSpore日志、普通日志。

图 14-10 ModelArts 训练日志



说明

只有MindSpore+Ascend训练场景下会产生单独的MindSpore日志。其他AI引擎的日志都包含在普通日志中，无法区分。

训练日志的时效性

从日志产生的时效性上可以分为以下3种情况：

- 实时日志：训练作业实时运行时产生，在ModelArts训练作业详情页面上可以查看。
- 历史日志：训练作业结束后，可以在ModelArts训练作业详情页面上查看历史日志，ModelArts系统自动保存30天。
- 永久日志：转存到OBS桶中的训练日志，在创建训练作业时，打开永久保存日志开关设置作业日志路径即可将日志转存至OBS路径。

图 14-11 开启永久保存日志开关

实时日志和历史日志都是标准日志输出，内容上没有区别。Ascend训练场景下，永久日志中会包含Ascend日志，这部分日志内容在ModelArts界面上看不到。

普通日志

普通日志中包含训练进程日志、pip-requirement.txt安装日志和ModelArts平台日志。

表 14-9 普通日志类型

日志类型	说明
训练进程日志	用户训练代码的标准输出。
pip-requirement.txt安装日志	如果用户有定义pip-requirement.txt文件，会产生pip包安装日志。
ModelArts平台日志	ModelArts平台产生的系统日志，主要用于运维人员定位平台问题。

普通日志的文件格式如下，其中task id为训练作业中的节点id。

```
统一日志格式：modelarts-job-[job id]-[task id].log
样例：log/modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log
```

- 单机训练作业只会生成一个日志文件，单机作业的task id默认为worker-0。
- 分布式场景下有多个节点日志文件并存，通过task id区分不同节点，例如：worker-0，worker-1等。

训练进程日志、“pip-requirement.txt”安装日志和ModelArts平台日志都包含在普通日志文件“modelarts-job-[job id]-[task id].log”中。

ModelArts平台日志可以通过关键字在训练的普通日志文件“modelarts-job-[job id]-[task id].log”中筛查，筛查关键字有：“[ModelArts Service Log]”或“Platform=ModelArts-Service”。

- 类型一：[ModelArts Service Log] xxx
```
[ModelArts Service Log][init] download code_url: s3://dgg-test-user/snt9-test-cases/mindspore/lenet/
```
- 类型二：time=“xxx” level=“xxx” msg=“xxx” file=“xxx” Command=xxx Component=xxx Platform=xxx
```
time="2021-07-26T19:24:11+08:00" level=info msg="start the periodic upload task, upload period = 5 seconds " file="upload.go:46" Command=obs/upload Component=ma-training-toolkit Platform=ModelArts-Service
```

Ascend 场景日志

使用Ascend资源运行训练作业时，会产生Ascend相关日志。Ascend训练场景下会生成device日志、plog日志、proc log单卡训练日志、MindSpore日志、普通日志。

其中，Ascend训练场景下的普通日志包括训练进程日志、pip-requirement.txt安装日志、ModelArts平台日志、ma-pre-start日志和davincirun日志。

Ascend日志结构举例说明如下：
```
obs://dgg-test-user/snt9-test-cases/log-out/                                    # 作业日志路径
├──modelarts-job-9ccf15f2-6610-42f9-ab99-059ba049a41e
   ├── ascend
      ├── process_log
            ├── rank_0
         ├── plog                                  # plog日志
                        ...
         ├── device-0                               # device日志
                        ...
   ├── mindspore                                                  # MindSpore日志
├──modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log              # 普通日志
├──modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-proc-rank-0-device-0.txt    # proc log单卡训练日志
```

表 14-10 Ascend 场景下日志说明

日志类型	日志说明	日志文件名
device日志	HOST侧用户进程，在DEVICE侧产生的AICPU、HCCP的日志，回传到HOST侧（训练容器）。 如果出现如下情况，则device日志会获取不到。 ● 节点异常重启 ● 被主动停止的节点 在训练进程结束后，该日志会生成到训练容器中。	“~/ascend/log/device-{device-id}/device-{pid}_{timestamp}.log” 其中，pid是HOST侧用户进程号。 样例： device-166_20220718191853764.log
plog日志	HOST侧用户进程，在HOST侧产生的日志（例如:ACL / GE）。 plog日志会生成到训练容器中。	“~/ascend/log/plog/plog-{pid}_{timestamp}.log” 其中，pid是HOST侧用户进程号。 样例： plog-166_20220718191843620.log

日志类型	日志说明	日志文件名
proc log	proc log是单卡训练日志重定向文件，方便用户快速定位对应计算节点的日志。使用MindSpore预置框架，ranktable启动的训练的proc log日志会生成到训练容器中，且自动保存到OBS。 使用MindSpore自定义镜像或其他框架镜像训练的作业不涉及proc log。	“[modelarts-job-uuid]-proc-rank-[rank id]-device-[device logic id].txt” ● device id为本次训练作业的NPU卡编号，取值单卡为0，8卡为0~7。 例如：Ascend规格为 8*Snt9时，device id取值为0~7；Ascend规格为 1*Snt9时，device id取值为0。 ● rank id为本次训练作业的全局NPU卡编号，取值为0~实例数*卡数-1，单个实例下，rank id与device id取值相同。 样例： modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-proc-rank-0-device-0.txt
MindSpore日志	使用MindSpore+Ascend训练时会产生单独的MindSpore日志。 MindSpore日志会生成到训练容器中。	MindSpore的日志介绍请参见**MindSpore官网**。

日志类型	日志说明	日志文件名
普通训练日志	普通训练日志会生成到训练容器的“/home/ma-user/modelarts/log”目录中，且自动上传到OBS。普通训练日志的类型如下所示。 ● ma-pre-start日志（Ascend场景特有）：如果用户有定义ma-pre-start脚本，会产生该脚本执行日志。 ● davincirun日志（Ascend场景特有）：Ascend训练进程通过davincirun.py文件启动，该启动文件产生的日志。 ● 训练进程日志：用户训练代码的标准输出。 ● pip-requirement.txt安装日志：如果用户有定义pip-requirement.txt文件，会产生pip包安装日志。 ● ModelArts平台日志：ModelArts平台产生的系统日志，主要用于运维人员定位平台问题。	合并输出在日志文件modelarts-job-[job id]-[task id].log中。 task id表示实例ID，单节点时取值为worker-0，多节点时取值为worker-0、worker-1、...worker-{n-1}，n为实例数。 样例： modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log

Ascend训练场景下，当训练进程退出后，ModelArts会上传训练容器中的日志文件至“作业日志路径” 参数设置的OBS目录中。在作业详情页可以获取“作业日志路径”，单击OBS地址可以直接跳转到OBS控制台查看日志。

图 14-12 日志存放路径

您可以通过ma-pre-start脚本修改默认环境变量配置。

```
ASCEND_GLOBAL_LOG_LEVEL=3      # 设置日志级别  debug级别为0;info级别为1;warning级别为2;error级别为3。
ASCEND_SLOG_PRINT_TO_STDOUT=1  # 设置plog日志是否在屏幕上显示，1表示默认设置在屏幕上显示日志。
ASCEND_GLOBAL_EVENT_ENABLE=1   # 设置事件级别  不开启Event日志级别为0；开启Event日志级别为1。
```

ma-pre-start脚本在与训练启动文件同级的目录下放置，命名为ma-pre-start.sh or ma-pre-start.py脚本。

在训练启动文件被执行前，系统会在 /home/work/user-job-dir/ 目录下执行上述ma-pre-start脚本，使用该机制可以更新容器镜像内安装的Ascend RUN包，或者设置一些训练运行时额外需要的全局环境变量。

查看训练作业日志

在训练作业详情页，训练日志窗口提供日志预览、日志下载、日志中搜索关键字、系统日志过滤能力。

- 预览

  界面默认显示“用户日志”，可以根据需要选择“系统日志”和“全量日志”查看日志详情。

  a. 用户日志：用户训练代码的标准输出。
  b. 系统日志： ModelArts 平台产生的系统日志，主要用于运维人员定位平台问题。
  c. 全量日志：包括用户日志和系统日志。当选择全量日志时，会显示当前全量日志文件大小。

  系统日志窗口提供训练日志预览功能，如果训练作业有多个计算节点，则支持查看不同计算节点的日志，通过右侧下拉框可以选择目标节点预览。

  当日志文件过大时，日志窗口仅加载最新的部分日志，并在日志窗口上方提供全量日志访问链接。打开该链接可在新页面查看全部日志。

  图 14-13 日志预览

  说明

  - 如果全部日志超过500M，可能会引起浏览页面卡顿，建议您直接下载日志查看。
  - 预览链接在生成后的一小时内，支持任何人打开并查看。您可以分享链接至他人。
  - **请注意日志中不能包含隐私内容，否则会造成信息泄露。**

- 下载

  训练日志仅保留30天，超过30天会被清理。如果用户需要永久保存日志，请单击日志窗口右上角“下载”按钮下载日志至本地保存，支持批量下载多节点日志。用户也可以在创建训练作业时打开永久保存日志按钮，保存训练日志至指定OBS路径。

  下载日志时默认下载全量日志。

  针对使用Ascend规格创建的训练作业，部分系统日志暂不支持直接在训练日志窗口下载，请在创建训练作业时指定OBS路径用于保存训练日志。

- 搜索关键字

  用户可以在日志搜索框搜索关键字，查询指定日志内容。

  系统支持高亮关键字并实现搜索结果间的跳转。搜索功能仅支持搜索当前页面加载的日志，如果日志加载不全（请关注页面提示）则需要下载或者通过打开全量日志访问链接进行搜索。全量日志访问链接打开的新页面可以通过Ctrl+F进行搜索。

- 查看动态路由加速日志

  当训练作业的实例数大于或等于3，环境变量配置了“ROUTE_PLAN=true”，且使用的是Ascend资源时，则可以根据Rank-ID查看动态路由加速的日志，还可以查看训练作业是否启用了动态路由加速。

  图 14-14 查看动态路由加速日志

14.7 使用 Cloud Shell 调试生产训练作业

ModelArts提供了Cloud Shell，可以登录运行中的容器，用于调试生产环境的训练作业。

约束限制

仅专属资源池支持使用Cloud Shell登录训练容器，且训练作业必须处于“运行中”状态。

前提条件：给子账号配置允许使用 Cloud Shell 的权限

1. 使用主用户账号登录华为云管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：允许通过Cloud Shell访问运行中的训练作业。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:trainJob:exec并选中，所有资源选择默认值。
3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限通过Cloud Shell登录运行中的训练作业容器。

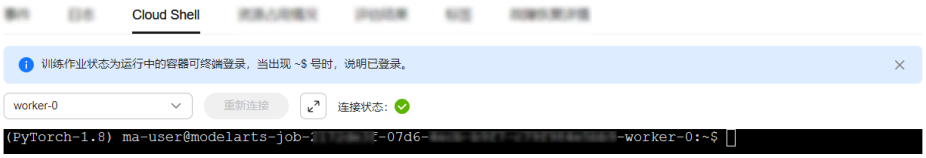
   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

使用 Cloud Shell

1. 参考**前提条件：给子账号配置允许使用Cloud Shell的权限**，完成配置。
2. 在**ModelArts管理控制台**的左侧导航栏中选择“模型开发与训练 > 模型训练”（旧版控制台：“模型训练 > 训练作业”）。
3. 在训练作业列表中，单击作业名称进入训练作业详情页面。
4. 在训练作业详情页面，单击“Cloud Shell”页签，登录训练容器。

   连接成功后，Cloud Shell界面提示如下。

   图 14-15 Cloud Shell 界面

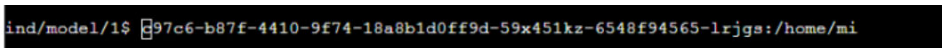
   当作业处于非运行状态或权限不足时会导致无法使用Cloud Shell，请根据提示定位原因即可。

   图 14-16 报错提示

   说明

   部分用户登录Cloud Shell界面时，可能会出现路径显示异常情况，此时在Cloud Shell中单击回车键即可恢复正常。

   图 14-17 路径异常

如何使训练作业保持运行中状态

由于需要训练作业处于“运行中”状态才能登录Cloud Shell，因此本文介绍如何使训练作业保持运行中状态，方便您快速通过Cloud Shell登录运行中的训练容器。

通过Sleep命令使训练作业保持运行

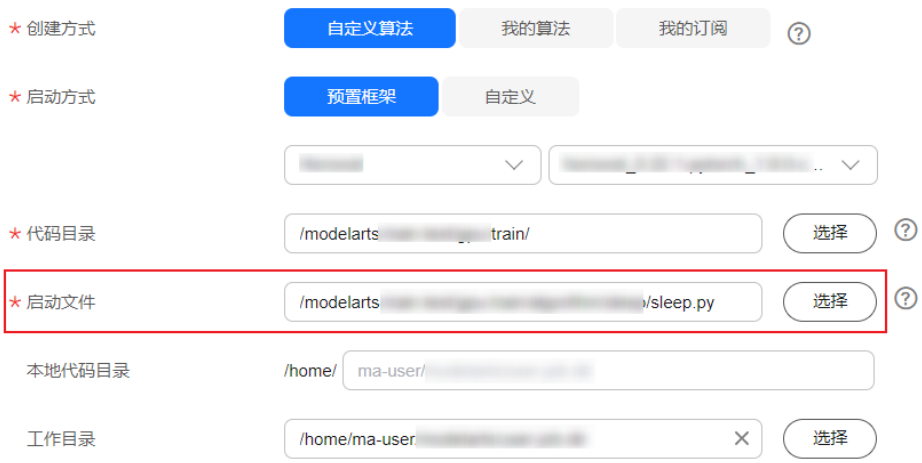
- 如果训练作业使用的是预置框架：

在创建训练作业时，“创建方式”选择“自定义算法”，“启动方式”选择“预置框架”，代码目录中新增sleep.py并将此脚本作为“启动文件”。这样启动的作业将会持续运行60分钟。您可通过Cloud Shell进入容器进行调试。

sleep.py示例：

```
import os
os.system('sleep 60m')
```

图 14-18 预置框架启动方式



- 如果训练作业使用的是自定义镜像

  在创建训练作业时，“创建方式”选择“自定义算法”，“启动方式”选择“自定义”，“启动命令”输入“sleep 60m”。这样启动的作业将会持续运行60分钟。您可通过Cloud Shell进入容器进行调试。

图 14-19 自定义启动方式

出错的任务如何卡在运行中状态

创建训练作业时，启动命令末尾新增"|| sleep 5h"，并启动训练作业，例如下方的cmd为您的启动命令：

```
cmd || sleep 5h
```

如果训练失败，则会执行sleep命令，此时可通过Cloud Shell登录容器镜像中调试。

说明

在Cloud Shell中调试多节点训练作业时，需要在Cloud Shell中切换work0、work1来实现对不同节点下发启动命令，否则任务会处于等待其他节点的状态。

如何防止 Cloud Shell 的 Session 断开

如果需要长时间运行某一个任务，为避免在期间连接断开导致任务失败，可通过使用screen命令使得任务在远程终端窗口运行。

1. 如果镜像中未安装screen，则执行“apt-get install screen”安装。
2. 创建screen终端。
```
# 使用 -S 创建一个叫name的screen终端
screen -S name
```
3. 显示已创建的screen终端。
```
screen -ls
There are screens on:
2433.pts-3.linux    (2013年10月20日 16时48分59秒)    (Detached)
2428.pts-3.linux    (2013年10月20日 16时48分05秒)    (Detached)
2284.pts-3.linux    (2013年10月20日 16时14分55秒)    (Detached)
2276.pts-3.linux    (2013年10月20日 16时13分18秒)    (Detached)
4 Sockets in /var/run/screen/S-root.
```
4. 连接“screen_id”为“2276”的screen终端。
```
screen -r 2276
```
5. 按下“Ctrl”+“a”+“d”键离开screen终端。离开后，screen会话仍将是活跃的，之后可以随时重新连接。

更多Screen使用说明可参考**Screen User's Manual**。

通过 py-spy 工具分析卡死进程的调用栈并结合代码分析定位卡死问题

本文指导用户通过py-spy工具分析卡死进程的调用栈并结合代码分析定位卡死问题。

步骤1 在**ModelArts管理控制台**，选择“模型开发与训练 > 模型训练”（旧版控制台：“模型训练 > 训练作业”）。

步骤2 在训练作业详情页面，选择Cloud Shell页签，登录训练容器（训练作业需处于运行中）。

步骤3 安装py-spy工具。
```
# 通过utils.sh脚本自动配置python环境
source /home/ma-user/modelarts/run/utils.sh

# 安装py-spy
pip install py-spy

# 如果超时提示connection broken by 'ProxyError('Cannot connect to proxy.')则表示用户设置了proxy，需要先关掉
export no_proxy=$no_proxy,repo.myhuaweicloud.com（此处需要替换成对应局点的pip源地址）
pip install py-spy
```

步骤4 查看堆栈。py-spy工具的具体使用方法可参考**py-spy官方文档**。
```
# 找到训练进程的PID
ps -ef

# 查看进程12345的进程堆栈
```

```
# 如果是8卡的训练作业，一般用此命令依次去查看主进程起的对应的8个进程的堆栈情况
py-spy dump --pid 12345
```

----结束

14.8 查看训练作业资源占用情况

场景描述

当前ModelArts训练作业模块在训练作业详情页的“监控”页签中提供了训练作业的资源使用情况监控，可以查看当前训练作业整体或单个节点占用的CPU、GPU或NPU资源使用情况。

约束限制

支持查看整个训练周期内的监控数据，具体数据参见**表14-11**。

训练作业的资源占用情况系统会自动保存30天，过期会被清除。

如何查看训练作业资源使用详情

1. 在**ModelArts管理控制台**的左侧导航栏中选择“模型开发与训练 > 模型训练”（旧版控制台：“模型训练 > 训练作业”）。
2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“监控”页签查看训练作业的资源使用情况，支持以下维度的资源监控，具体的监控指标如**表14-11**所示。
   - 作业监控：监控当前训练作业整体的资源使用情况，监控周期可以选择最近15分钟、最近30分钟、最近1小时、最近6小时、最近12小时、最近24小时，也可以自定义时段。
   - 任务监控：监控训练作业运行的具体节点的资源使用情况，监控周期可以选择最近15分钟、最近30分钟、最近1小时、最近6小时、最近12小时、最近24小时，也可以自定义时段。
   - 作业状态为创建中、等待中和运行中的作业默认展示最近15分钟的监控数据。作业状态为异常、终止中、已终止和已完成的作业默认展示作业运行开始到结束的时间段的监控数据。

表 14-11 训练作业监控指标

分类	指标名称	指标参数	指标含义	单位	取值范围	呈现形式
CPU	CPU使用率	ma_container_cpu_util	该指标用于统计测量对象的CPU使用率。	百分比（Percent）	0～100%	折线图
	CPU使用核数	ma_container_cpu_used_core	该指标用于统计测量对象已经使用的CPU核个数	核（Core）	≥0	柱状图

分类	指标名称	指标参数	指标含义	单位	取值范围	呈现形式
内存使用	物理内存使用率	ma_container_memory_util	该指标用于统计测量对象已使用内存占申请物理内存总量的百分比。	百分比（Percent）	0～100%	折线图
	物理内存使用量	ma_container_memory_used_meg	该指标用于统计测量对象实际已经使用的物理内存，对应container_memory_working_set_bytes当前内存工作集（working set）使用量。 工作区内存使用量=活跃的匿名页和缓存，以及file-baked页<=container_memory_usage_bytes。	兆字节（Megabytes）	≥0	柱状图
GPU	GPU使用率	ma_container_gpu_util	该指标用于统计测量对象的GPU使用率。	百分比（Percent）	0～100%	折线图
	GPU显存使用率	ma_container_gpu_mem_util	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0～100%	折线图
	GPU显存使用量	ma_container_gpu_mem_used_megabytes	该指标用于统计测量对象已使用的显存。	兆字节（Megabytes）	≥0	柱状图
NPU	NPU使用率	ma_container_npu_ai_core_util	昇腾系列AI处理器AI Core利用率	百分比（Percent）	0～100%	折线图
	NPU显存使用率	ma_container_npu_memory_util	该指标用于统计测量对象已使用的NPU显存占NPU存储容量的百分比。（即将废弃， Snt3系列替代指标为ma_container_npu_ddr_memory_util，Snt9系列替代指标为ma_container_npu_hbm_util）。	百分比（Percent）	0～100%	折线图

分类	指标名称	指标参数	指标含义	单位	取值范围	呈现形式
	NPU显存使用量	ma_container_npu_memory_used_megabytes	该指标用于统计测量对象已使用的NPU显存。（即将废弃，Snt3系列替代指标为ma_container_npu_ddr_memory_usage_bytes，Snt9系列替代指标为ma_container_npu_hbm_usage_bytes）。	兆字节（Megabytes）	≥0	柱状图
网络	网络IO-下行速率	ma_container_network_receive_bytes	网络接收数据速率，该指标用于统计测试对象的入方向网络流速。	字节/秒（Bytes/Second）	≥0	折线图
	网络IO-上行速率	ma_container_network_transmit_bytes	网络发送数据速率，该指标用于统计测试对象的出方向网络流速。	字节/秒（Bytes/Second）	≥0	折线图
磁盘	磁盘读取速率	ma_container_disk_read_kilobytes	该指标用于统计每秒从磁盘读出的数据量。	千字节/秒（Kilobytes/Second）	≥0	折线图
	磁盘写入速率	ma_container_disk_write_kilobytes	该指标用于统计每秒写入磁盘的数据量。	千字节/秒（Kilobytes/Second）	≥0	折线图

更多指标查看请参见文档**查看训练作业监控指标**章节实现。

如何判断训练作业资源利用率高低

在模型训练的训练作业列表页可以查看作业资源利用率情况。当作业worker-0实例的GPU/NPU的平均利用率低于50%时，在训练作业列表中会进行告警提示。

图 14-20 作业列表显示作业资源利用率情况

此处的作业资源利用率只涉及GPU和NPU资源。作业worker-0实例的GPU/NPU平均利用率计算方法：将作业worker-0实例的各个GPU/NPU加速卡每个时间点的利用率汇总取平均值。

如何提高训练作业资源利用率

- 适当增大batch_size：较大的batch_size可以让GPU/NPU计算单元获得更高的利用率，但是也要根据实际情况来选择batch_size，防止batch_size过大导致内存溢出。
- 提升数据读取的效率：如果读取一个batch数据的时间要长于GPU/NPU计算一个batch的时间，就有可能出现GPU/NPU利用率上下浮动的情况。建议优化数据读取和数据增强的性能，例如将数据读取并行化，或者使用NVIDIA Data Loading Library（DALI）等工具提高数据增强的速度。
- 模型保存不要太频繁：模型保存操作一般会阻塞训练，如果模型较大，并且较频繁地进行保存，就会影响GPU/NPU利用率。同理，其他非GPU/NPU操作尽量不要阻塞训练主进程太多的时间，如日志打印，保存训练指标信息等。

14.9 查看训练作业监控指标

在训练作业运行过程中，如果用户能在模型训练出问题（例如loss值异常）的情况下能收到告警并及时处理，可以节省大量时间和资源，避免无效运行作业导致的浪费。同时通过指标监控可以实时掌握训练作业的进度，了解模型在不同阶段的训练状态。

当前ModelArts训练作业模块在训练作业详情页的“监控”页签中提供了训练作业占用的CPU、GPU或NPU资源使用情况，具体参见**支持在ModelArts控制台上直接查看的监控指标**。

除了在**ModelArts管理控制台**训练作业详情页可以查看训练作业的指标外，更多指标可以登录到**AOM控制台**查看，具体参见**在AOM控制台查看ModelArts所有监控指标**。

除此外，ModelArts还支持自定义一些指标信息采集上报到AOM系统，比如训练日志常见打印loss值、step耗时、gpu throughput等指标信息，方便观察指标变化趋势，或对比不同训练作业指标情况。

通过Prometheus指标采集到AOM（HTTP）：通过配置采集进程和HTTP接口，由平台自动拉取指标数据。

- 适合希望简化操作，减少代码改动的用户。
- 适合已经熟悉Prometheus生态，有现成采集工具的用户。

通过Prometheus指标采集到AOM（Command）：通过配置执行命令和执行命令参数，由平台自动拉取指标数据。

- 适合不想暴露指标解析的用户。
- 适合已经熟悉Prometheus生态，有现成采集工具的用户。

使用SDK上报自定义监控指标到AOM：通过在代码中集成SDK，手动上传指标数据到AOM。

- 适合需要高度定制化指标，例如自定义维度或复杂计算的用户。
- 适用于复杂的业务逻辑，需要在代码层面控制指标上报的场景。
- 适合已经使用华为云SDK，熟悉云服务的用户。

支持在 ModelArts 控制台上直接查看的监控指标

支持在**ModelArts管理控制台**训练作业详情页中直接查看的监控指标请参见**表1 训练作业监控指标**。

支持在**ModelArts管理控制台**总览页中查看的训练相关监控指标请参见**表1 训练作业任务级的指标说明**。

表 14-12 训练作业任务级的指标说明

指标	说明	查看方式
训练作业资源利用率	每个训练作业的CPU、GPU或NPU资源利用率。	在**ModelArts管理控制台**总览页的“训练作业资源利用情况”版块查看。
卡时	每个训练作业运行时长和占用卡数。	在**ModelArts管理控制台**总览页的“训练作业资源利用情况”版块查看。

通过 Prometheus 指标采集到 AOM（HTTP）

通过配置采集进程和HTTP接口，由平台自动拉取Prometheus指标数据上报到**AOM控制台**。该方案用户只需配置指标的采集和暴露，无需手动上传，简化了操作流程，特别适合需要实时监控训练作业性能的场景。

注意

在使用本方案时，请注意以下约束：

- 指标格式要求：用户自定义的指标必须符合云原生exporter的标准上报格式，否则可能导致数据无法解析。

  指标格式示例：
  ```
  mspti_marker_range_cost_time{name="Step: 0", source_kind="host", process_id="1887317", thread_id="1887317"} 261981960 1732357956859823920
  mspti_marker_range_cost_time{name="Step matmul: 0", source_kind="host", process_id="1887317", thread_id="1887757"} 2366640 1732357956863054960
  ```
- 数据量限制：如果在10秒内上报的指标数据过多（超过32KB），可能会导致数据丢失。
- 打点频率限制：如果指标的打点频率低于毫秒级，可能会因为时间戳重复而导致部分指标丢失。

步骤1 在训练容器中启动指标采集进程。要在训练进程之外，启动一个单独的进程来采集自定义指标。

例如，可以使用Flask框架来创建一个简单的HTTP服务器。

```
from flask import Flask
app = Flask(__name__)

@app.route('/metrics')
def get_metrics():
    # 生成或获取自定义指标数据
    metrics = """
    mspti_marker_range_cost_time{name="Step: 0", source_kind="host", process_id="1887317",
thread_id="1887317"} 261981960 1732357956859823920
    mspti_marker_range_cost_time{name="Step matmul: 0", source_kind="host", process_id="1887317",
thread_id="1887757"} 2366640 1732357956863054960
    """
    return metrics

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

步骤2 配置HTTP接口，确保指标采集进程提供一个HTTP接口（如/metrics），以便训练平台可以定时拉取指标数据。

每次HTTP接口被调用后，及时清理已采集的指标，避免内存泄漏或性能问题。

步骤3 启用Prometheus指标采集。

在创建训练作业时，开启“采集Prometheus指标”，“获取指标类型”选择“http获取指标”，并配置“采集URL”和“采集端口”。参数配置要满足以下要求：

- 确保配置的URL和端口号与指标采集进程一致，否则可能导致指标无法上报。
- 确保训练作业所在的网络环境允许访问配置的URL和端口号，避免因网络问题导致指标采集失败。

步骤4 在AOM查看自定义监控指标。

训练作业运行后，用户可以登录**AOM控制台**，在“指标浏览”页面，通过指定“指标”查看上报的自定义Prometheus指标。

----结束

通过 Prometheus 指标采集到 AOM（Command）

通过配置执行命令和执行命令参数，由平台自动拉取Prometheus指标数据上报到**AOM控制台**。该方案用户只需配置指标的采集，无需手动上传，指标解析过程完全由用户自主定制，特别适合需要实时监控训练作业性能的场景。

注意

在使用本方案时，请注意以下约束：

- 指标格式要求：自定义指标数据的格式必须符合特定格式的文本，即每个指标的格式应为“<指标名称>{<标签名称>=<标签值>,...} <采样值> [毫秒时戳]”。

  指标数据示例：

  ```
  # HELP http_requests_total The total number of HTTP requests.
  # TYPE http_requests_total gauge
  html_http_requests_total{method="post",code="200"} 1656 1686660980680
  html_http_requests_total{method="post",code="400"} 2 1686660980681
  ```

- 数据量限制：如果在10秒内上报的指标数据过多（超过32KB），可能会导致数据丢失。
- 打点频率限制：如果指标的打点频率低于毫秒级，可能会因为时间戳重复而导致部分指标丢失。

步骤1 提供自定义指标文件，比如“test.prom”，该文件可以是存储在训练镜像或者是训练代码中，但必须保证是训练容器可获取到的。

步骤2 启用Prometheus指标采集。

在创建训练作业时，开启“采集Prometheus指标”，“获取指标类型”选择“命令行获取指标”，并配置如下参数：

- “执行命令”：读取指标的Linux命令，例如填写“cat”。
- “执行命令参数”：填写自定义指标文件的路径，例如“/XXX/a.prom”。

要确保配置的执行命令和执行命令参数是可以稳定输出指标的，且响应要在秒级，否则指标可能会采集失败。

步骤3 在AOM查看自定义监控指标。

训练作业运行后，用户可以登录**AOM控制台**，在“指标浏览”页面，通过指定“指标”查看上报的自定义Prometheus指标。

----结束

使用 SDK 上报自定义监控指标到 AOM

通过在代码中集成SDK，手动上传指标数据到AOM。

该方案适用于监控高度定制化的指标，例如自定义维度或复杂计算，需要在代码层面直接控制指标的上报，适合复杂的业务逻辑。

1. 准备训练代码，在训练代码中增加指标监控的代码。具体代码示例如下。准备训练代码的其它要求请参见**准备模型训练代码**。

   代码中倒数第2行的region = "cn-southwest-2"参数值请替换成实际region值，region取值可以参考**终端节点**。

   代码中增加监控指标数据，具体参数解释参考**AOM文档**。

```
# coding: utf-8
import os

from huaweicloudsdkaom.v2 import *
from huaweicloudsdkaom.v2.region.aom_region import AomRegion
from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkcore.exceptions import exceptions
from moxing.framework import cloud_utils


def report2Aom(request,region):
    auth = cloud_utils.get_auth() #AK、SK和临时TOKEN值，系统会自动获取无需在代码中填写。

    ak = auth.AK
    sk = auth.SK
    securityToken = auth.TOKEN

    projectId = os.environ.get("MA_IAM_PROJECT_ID")
    credentials = BasicCredentials(ak, sk, projectId).with_security_token(securityToken)
    client = AomClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(AomRegion.value_of(region)) \
        .build()
    try:
        response = client.add_metric_data(request)
        print(response)
    except Exception as e:
        print(e)
if __name__ == "__main__":
```

```
request = AddMetricDataRequest()
listValuesBody = [
    #下面填上对应的指标名称、类型、单位、数值，比如step_time、loss值等
    ValueData(
        metric_name="step_time", #监控指标名称，例如step_time
        type="float", #指标的数据类型，取值范围只能是"int"或"float"。
        unit="ms",    #数据的单位。长度不超过32个字符，此处举例为ms
        value=135.572  #指标数据的值。取值范围有效的数值类型。最小值0
    ),
    ValueData(
        metric_name="loss",
        type="float",
        value=0.6932
    )
]
listDimensionsMetric = [
    #下面填上想查看的指标维度，比如线程、host等等
    Dimension2(
        name="cluster_name",#此处仅为举例示意，请替换为实际需要查看的指标维度
        value="fab2c5cf438b4f0c851fdcdf"# 此处仅为举例示意，请替换为实际参数值
    ),
    Dimension2(
        name="user_name",
        value="modelarts_02" # 此处仅为举例示意，请替换为实际参数值
    ),
    Dimension2(
        name="user_id",
        value="04f258c8fb00d42a1f6xxx" # 此处仅为举例示意，请替换为实际参数值
    )
]
metricBody = MetricItemInfo(
    dimensions=listDimensionsMetric,
    namespace="NOPAAS.ESC"  #保持默认值即可，无需修改
)
listBodybody = [
    MetricDataItem(
        collect_time=int(round(time.time()*1000)), #监控指标数据收集时间，为最新的时间戳，ms为单
位
        metric=metricBody,
        values=listValuesBody
    )
]
request.body = listBodybody
region = "cn-southwest-2"  #请根据实际region替换
response = report2Aom(request,region)
```

2. 在训练代码中加入命令，用于加载对应的依赖包。如果使用的是自定义镜像，也可以在制作镜像时安装以下依赖，具体参见**开发用于自定义镜像训练的代码**。

```
pip install huaweicloudsdkaom
pip install huaweicloudsdkcore
```

3. 创建训练作业并运行，具体参考**创建生产训练作业**章节。
4. 登录**AOM控制台**，在“指标浏览”页面，通过指定“指标”查看上报的指标数据。
5. 参考**告警上报配置方法**章节设置AOM告警和通知机制。

14.10 使用 CTS 审计 ModelArts 服务

ModelArts支持对接云审计CTS服务，通过云审计服务，您可以记录与ModelArts相关的操作事件，便于日后的查询、审计和回溯。

在您开启了云审计服务后，系统会记录ModelArts的相关操作，且控制台保存最近7天的操作记录。本节介绍如何在云审计服务管理控制台查看最近7天的操作记录。

前提条件

已开通云审计服务。详细操作请参见《云审计服务用户指南》。

训练作业支持审计的关键操作列表

表 14-13 训练作业支持审计的关键操作列表

操作名称	资源类型	事件名称
创建训练作业	JobV2	CreateJobV2
更新训练作业	JobV2	UpdateJobV2
删除训练作业	JobV2	DeleteJobV2
终止训练作业	JobV2	StopJobV2
创建算法	AlgorithmV2	CreateAlgorithmV2
更新算法	AlgorithmV2	UpdateAlgorithmV2
删除算法	AlgorithmV2	DeleteAlgorithmV2
创建训练作业镜像保存任务	SaveImageTask	CreateSaveImageTask
创建训练作业标签	JobV2	CreateJobTags
删除训练作业标签	JobV2	DeleteJobTags
提交诊断任务	DiagnosisTask	CreateDiagnosisTask
创建训练实验	Experiment	CreateExperiment
更新训练实验	Experiment	UpdateExperiment
删除训练实验	Experiment	DeleteExperiment

操作步骤

1. 登录云审计服务管理控制台。
2. 在管理控制台左上角单击 图标，选择区域。
3. 在左侧导航栏中，单击“事件列表”，进入“事件列表”页面。
4. 事件列表支持通过筛选来查询对应的操作事件。当前事件列表支持四个维度的组合查询，详细信息如下：
   - 事件来源、资源类型和筛选类型。

     在下拉框中选择查询条件。

     其中筛选类型选择事件名称时，还需选择某个具体的事件名称。

     选择资源ID时，还需输入某个具体的资源ID。

     选择资源名称时，还需选择或手动输入某个具体的资源名称。

- 操作用户：在下拉框中选择某一具体的操作用户，此操作用户指用户级别，而非租户级别。
- 事件级别：可选项为“所有事件级别”、“normal”、“warning”、“incident”，只可选择其中一项。
- 时间范围：可选择查询最近七天内任意时间段的操作事件。

5. 在需要查看的事件左侧，单击 ⌄ 展开该事件的详细信息。
6. 单击需要查看的事件“操作”列的“查看事件”，可以在弹窗中查看该操作事件结构的详细信息。

   更多关于云审计服务事件结构的信息，请参见《云审计服务用户指南》。

14.11 智能运维

场景描述

训练作业运行过程中，ModelArts平台会对训练作业进行全方位的实时监测，确保作业的正常运行。训练作业详情中提供智能运维功能，便于用户对作业的监测和运维。

当训练作业运行结束后，如果作业状态是“失败“或者“已终止”， 可以在“智能运维”中根据需要选择诊断工具对作业进行诊断。不同工具的诊断范围和诊断时间存在差异，请根据需要选择诊断类型。

前提条件

故障监测需要用户创建训练作业时开启“自动重启”功能。

代码异常监测需要用户创建训练作业时开启“自动重启”功能。

性能监测依赖需要用户创建训练作业时开启“性能监控与诊断”功能。

实时监测

智能运维的实时监测分为故障监测、代码异常监测和性能监测。

当用户创建的训练作业开启了前提条件中对应的功能时，智能运维系统会实时监测训练作业，在界面能看到监测未防护、无风险、低风险、中风险、高风险和监测中等状态提示。监测异常时会有相应的风险等级和监测报告，便于用户及时处理异常作业。

表 14-14 实时监测对比

实时监测类型	说明	监测异常风险等级
故障监测	当作业出现故障时，系统将自动进行重启恢复，以保障训练作业高可用。	中风险 高风险
代码异常监测	对当前训练作业代码进行实时监测，当出现异常时将自动生成诊断报告协助排障。	低风险 中风险 高风险

实时监测类型	说明	监测异常风险等级
性能监测	实时监测训练作业的性能指标，当性能指标异常时，自动生成监测报告协助排障。	中风险

诊断工具

诊断工具暂时支持两种，性能分析和标准诊断。

表 14-15

诊断工具	工具说明
性能分析	针对训练中耗时超预期、资源利用率失衡等性能劣化问题，可实时观测 step 耗时曲线；支持手动采集 profiling 数据，采集完成后生成可视化分析结果。
标准诊断	主要是对训练作业环境信息、作业事件、作业日志、设备日志等进行检测，可识别作业运行环境、代码异常、硬件等故障。诊断时间与作业集群、作业日志大小正相关。

单击对应工具右侧的“立即诊断”，可对训练作业进行诊断检测。

诊断完成后可查看诊断报告。

查看监测报告

当实时监测到异常时，会有异常检测报告。单击“查看报告”可以查看详细诊断结果。

- 代码异常监测报告

  诊断结果中包含故障事件涉及的故障设备、关键日志、故障类别、故障组件、故障模块以及处理建议方案等信息。

查看诊断报告

诊断完成后，可在训练作业详情页的“智能运维”页签单击“诊断报告”查看详情。也可以在“运维管理 > 日志诊断”的列表中查找对应的作业，查看诊断详情。

诊断报告可以详细查看诊断作业的基础信息和诊断结果。

- 基础信息

  包括作业ID、诊断耗时、诊断创建时间、更新时间、创建者、描述、资源类型、训练作业ID等内容。

- 诊断结果

  包括现象描述、故障描述和故障处理建议方案等内容。

14.12 查看模型评估结果

训练作业运行结束后，ModelArts可为您的模型进行评估，并且给出调优诊断和建议。

- 针对使用预置算法创建训练作业，无需任何配置，即可查看此评估结果（由于每个模型情况不同，系统将自动根据您的模型指标情况，给出一些调优建议，请仔细阅读界面中的建议和指导，对您的模型进行进一步的调优）。
- 针对用户自己编写训练脚本或自定义镜像方式创建的训练作业，则需要在您的训练代码中添加评估代码，才可以在训练作业结束后查看相应的评估诊断建议。

说明

- 只支持验证集的数据格式为图片。
- 目前，仅如下常用框架的训练脚本支持添加评估代码。
  - TF-1.13.1-python3.6
  - TF-2.1.0-python3.6
  - PyTorch-1.4.0-python3.6

下文将介绍如何在训练中使用评估代码。对训练代码做一定的适配和修正，分为三个方面：**添加输出目录**、**复制数据集到本地**、**映射数据集路径到OBS**。

添加输出目录

添加输出目录的代码比较简单，即在代码中添加一个输出评估结果文件的目录，被称为train_url，也就是页面上的训练输出位置。并把train_url添加到使用的函数analysis中，使用save_path来获取train_url。示例代码如下所示：

```
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
tf.app.flags.DEFINE_string('adv_param_json',
                           '{"attack_method":"FGSM","eps":40}',
                           'params for adversarial attacks')
FLAGS(sys.argv, known_only=True)

...

# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
    name_list=file_name_list,
    label_map_dict=label_dict,
    save_path=FLAGS.train_url)
```

复制数据集到本地

复制数据集到本地主要是为了防止长时间访问OBS容易导致OBS连接中断使得作业卡住，所以一般先将数据复制到本地再进行操作。

数据集复制有两种方式，推荐使用OBS路径复制。

- OBS路径（推荐）

  直接使用moxing的copy_parallel接口，复制对应的OBS路径。

- ModelArts数据管理中的数据集（即Manifest文件格式）

  使用moxing的copy_manifest接口将文件复制到本地并获取新的Manifest文件路径，然后使用SDK解析新的Manifest文件。

说明

ModelArts数据管理模块在重构升级中，对未使用过数据管理的用户不可见。建议新用户将训练数据存放至OBS桶中使用。

```
if data_path.startswith('obs://'):
    if '.manifest' in data_path:
        new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
        data_path = new_manifest_path
    else:
        mox.file.copy_parallel(data_path, '/cache/data/')
        data_path = '/cache/data/'
    print('------------- download dataset success -------------')
```

映射数据集路径到 OBS

由于最终JSON体中需要填写的是图片文件的真实路径，也就是OBS对应的路径，所以在复制到本地做完分析和评估操作后，需要将原来的本地数据集路径映射到OBS路径，然后将新的list送入analysis接口。

如果使用的是OBS路径作为输入的data_url，则只需要替换本地路径的字符串即可。

```
if FLAGS.data_url.startswith('obs://'):
    for idx, item in enumerate(file_name_list):
        file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
```

如果使用Manifest文件，需要再解析一遍原版的Manifest文件获取list，然后再送入analysis接口。

```
if or FLAGS.data_url.startswith('obs://'):
    if 'manifest' in FLAGS.data_url:
            file_name_list = []
            manifest, _ = get_sample_list(
                manifest_path=FLAGS.data_url, task_type='image_classification')
            for item in manifest:
                if len(item[1]) != 0:
                    file_name_list.append(item[0])
```

完整的适配了训练作业创建的图像分类样例代码如下：

```
import json
import logging
import os
import sys
import tempfile

import h5py
import numpy as np
from PIL import Image

import moxing as mox
import tensorflow as tf
from deep_moxing.framework.manifest_api.manifest_api import get_sample_list
from deep_moxing.model_analysis.api import analyse, tmp_save
from deep_moxing.model_analysis.common.constant import TMP_FILE_NAME

logging.basicConfig(level=logging.DEBUG)

FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
```

```
tf.app.flags.DEFINE_string('adv_param_json',
                           '{"attack_method":"FGSM","eps":40}',
                           'params for adversarial attacks')
FLAGS(sys.argv, known_only=True)


def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[np.newaxis, :, :, :]
    return img


def softmax(x):
    x = np.array(x)
    orig_shape = x.shape
    if len(x.shape) > 1:
        # Matrix
        x = np.apply_along_axis(lambda x: np.exp(x - np.max(x)), 1, x)
        denominator = np.apply_along_axis(lambda x: 1.0 / np.sum(x), 1, x)
        if len(denominator.shape) == 1:
            denominator = denominator.reshape((denominator.shape[0], 1))
        x = x * denominator
    else:
        # Vector
        x_max = np.max(x)
        x = x - x_max
        numerator = np.exp(x)
        denominator = 1.0 / np.sum(numerator)
        x = numerator.dot(denominator)
    assert x.shape == orig_shape
    return x


def get_dataset(data_path, label_map_dict):
    label_list = []
    img_name_list = []
    if 'manifest' in data_path:
        manifest, _ = get_sample_list(
            manifest_path=data_path, task_type='image_classification')
        for item in manifest:
            if len(item[1]) != 0:
                label_list.append(label_map_dict.get(item[1][0]))
                img_name_list.append(item[0])
            else:
                continue
    else:
        label_name_list = os.listdir(data_path)
        label_dict = {}
        for idx, item in enumerate(label_name_list):
            label_dict[str(idx)] = item
            sub_img_list = os.listdir(os.path.join(data_path, item))
            img_name_list += [
                os.path.join(data_path, item, img_name) for img_name in sub_img_list
            ]
            label_list += [label_map_dict.get(item)] * len(sub_img_list)
    return img_name_list, label_list


def deal_ckpt_and_data_with_obs():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url

    if pb_dir.startswith('obs://'):
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/')
        pb_dir = '/cache/ckpt'
        print('------------- download success -------------')
    if data_path.startswith('obs://'):
```

```
        if '.manifest' in data_path:
            new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
            data_path = new_manifest_path
        else:
            mox.file.copy_parallel(data_path, '/cache/data/')
            data_path = '/cache/data/'
        print('------------ download dataset success ------------')
    assert os.path.isdir(pb_dir), 'Error, pb_dir must be a directory'
    return pb_dir, data_path


def evalution():
    pb_dir, data_path = deal_ckpt_and_data_with_obs()
    index_file = os.path.join(pb_dir, 'index')
    try:
        label_file = h5py.File(index_file, 'r')
        label_array = label_file['labels_list'][:].tolist()
        label_array = [item.decode('utf-8') for item in label_array]
    except Exception as e:
        logging.warning(e)
        logging.warning('index file is not a h5 file, try json.')
        with open(index_file, 'r') as load_f:
            label_file = json.load(load_f)
        label_array = label_file['labels_list'][:]
    label_map_dict = {}
    label_dict = {}
    for idx, item in enumerate(label_array):
        label_map_dict[item] = idx
        label_dict[idx] = item
    print(label_map_dict)
    print(label_dict)

    data_file_list, label_list = get_dataset(data_path, label_map_dict)

    assert len(label_list) > 0, 'missing valid data'
    assert None not in label_list, 'dataset and model not match'

    pred_list = []
    file_name_list = []
    img_list = []

    for img_path in data_file_list:
        img = _preprocess(img_path)
        img_list.append(img)
        file_name_list.append(img_path)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.visible_device_list = '0'
    with tf.Session(graph=tf.Graph(), config=config) as sess:
        meta_graph_def = tf.saved_model.loader.load(
            sess, [tf.saved_model.tag_constants.SERVING], pb_dir)
        signature = meta_graph_def.signature_def
        signature_key = 'predict_object'
        input_key = 'images'
        output_key = 'logits'
        x_tensor_name = signature[signature_key].inputs[input_key].name
        y_tensor_name = signature[signature_key].outputs[output_key].name
        x = sess.graph.get_tensor_by_name(x_tensor_name)
        y = sess.graph.get_tensor_by_name(y_tensor_name)
        for img in img_list:
            pred_output = sess.run([y], {x: img})
            pred_output = softmax(pred_output[0])
            pred_list.append(pred_output[0].tolist())

    label_dict = json.dumps(label_dict)
    task_type = 'image_classification'

    if FLAGS.data_url.startswith('obs://'):
```

```
        if 'manifest' in FLAGS.data_url:
            file_name_list = []
            manifest, _ = get_sample_list(
                manifest_path=FLAGS.data_url, task_type='image_classification')
            for item in manifest:
                if len(item[1]) != 0:
                    file_name_list.append(item[0])
        for idx, item in enumerate(file_name_list):
            file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
    # analyse
    res = analyse(
        task_type=task_type,
        pred_list=pred_list,
        label_list=label_list,
        name_list=file_name_list,
        label_map_dict=label_dict,
        save_path=FLAGS.train_url)

if __name__ == "__main__":
    evalution()
```

14.13 查看训练作业标签

通过给训练作业添加标签，可以标识云资源，便于快速搜索训练作业。

1. 在**ModelArts管理控制台**的左侧导航栏中选择“模型开发与训练 > 模型训练”（旧版控制台：“模型训练 > 训练作业”）。
2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“标签”页签查看标签信息。

   支持添加、修改、删除标签。标签详细用法请参见**使用TMS标签实现资源分组管理**。

   图 14-21 查看训练标签

说明

最多支持添加20个标签。

14.14 修改训练作业优先级

使用专属资源池训练作业时，支持在创建训练作业时设置任务优先级，也支持作业在长时间处于“等待中”的状态时调整优先级。如通过调整作业优先级可以减少作业的排队时长。

什么是训练作业优先级

在用户运行训练作业过程中，需要对训练作业做优先级划分。比如有一些任务是低优先级，可能是跑一些测试、也可能是跑一些简单的不重要的实验。在这类场景下，当有高优先级作业的时候，需要能比低优先级作业更快进入排队队列。

在资源使用高峰期，用户可以通过提供或降低训练作业的优先级，来动态调节作业的执行顺序，保障关键业务的及时运行。

约束限制

- 仅使用专属资源池训练时才支持设置训练作业优先级。公共资源池不支持设置训练作业优先级。
- 作业优先级可以设置的取值为数字1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。当选择“3”优先级时，还需要配置用户权限才可以生效，权限配置参考本章节内容**给子账号配置"设置作业调度最高优先级"权限**。
- 只有训练作业长时间在专属资源池的排队列表中导致的“等待中”状态才支持修改作业优先级，如果是其他原因导致的长时间等待则不支持修改作业优先级。

如何设置训练作业优先级

在创建训练作业页面可以勾选“提高作业调度优先级”。优先级可以设置数字1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。

如何修改训练作业优先级

1. 在训练作业列表页面，选择“状态”为“等待中”的训练作业。
2. 单击训练作业名称，进入详情页面。
3. 在“作业优先级”单击，在弹窗中修改优先级后单击“确定”。

在ModelArts中，训练作业的排队和优先级下发逻辑如下:

1. 相同优先级作业的下发逻辑：
   - 如果资源充足，作业按照提交时间的顺序进行调度，遵循先进先出（FIFO）原则依次执行。
   - 如果资源不足，可能会出现不同逻辑池的作业被优先调度的情况，导致先进先出的顺序不完全保证。
2. 排队中调整优先级后的下发逻辑：

   当前训练作业下发到资源池后，不支持修改优先级。

给子账号配置"设置作业调度最高优先级"权限

默认用户权限可选择优先级“1”和“2”，配置了"设置作业为高优先级"权限的用户可选择优先级1~3。当选择优先级为“3”时还需要为用户配置“modelarts:trainJob:setHighPriority”权限，才能生效。

1. 使用主用户账号登录**华为云管理控制台**，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：允许用户设置训练作业最高优先级。

- “策略配置方式”：选择可视化视图。
- “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词“modelarts:trainJob:setHighPriority”并选中，所有资源选择默认值。

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限设置训练作业优先级1~3。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

14.15 保存调试模式训练作业镜像

在使用镜像进行开发和调试的过程中，用户可能会对镜像环境进行修改和优化。如果用户在完成镜像修改后，希望保留当前的镜像环境以备后续开发使用，此时可以在控制台使用"保存镜像"功能，将当前的镜像环境进行保存，用于后续的开发和调试工作。

在镜像保存时，需要注意以下几点：

1. 安装的依赖包将被保留，但训练数据、代码等需要持久化存储的内容不会被保存在最终生成的容器镜像中。
2. 在VS Code远程开发场景下，用户在Server端安装的插件将被保留。

这样可以确保关键配置和依赖得到保存，而动态数据和代码则需要通过其他方式进行管理。

前提条件

- 调试作业创建在专属资源池上。
- 作业状态为“运行中”。

约束限制

- 在Console上仅支持保存训练作业worker-0节点的镜像。
- 建议保存的镜像大小不要超过35G，镜像层数不要超过125层，因为节点容器存储Rootfs差异，可能会导致镜像保存失败。
  - 若想保存更大的镜像，可尝试在“资源管理 > 专属算力资源 > 资源池”（旧版控制台：“资源管理 > 专属资源池”）页面按需调整容器引擎空间大小，具体步骤请参考扩缩容专属资源池的“修改容器引擎空间大小”。
  - 如果问题仍未解决，请联系技术支持。

操作步骤

1. 单击训练作业名进入作业详情，单击页面右上方“保存镜像”，弹出保存镜像对话框。
2. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。单击“确定”保存镜像。

   在“组织”下拉框中选择一个组织。如果没有组织，可以单击右侧的“立即创建”，创建一个组织。

同一个组织内的用户可以共享使用该组织内的所有镜像。

镜像会以快照的形式保存，保存需要一段时间，请耐心等待。此时镜像状态为保存中。

说明

快照中耗费的时间仍占用作业的总运行时长，如果在快照中时，作业因运行时间到期停止，将导致镜像保存失败。

3. 镜像保存成功后，用户可在创建训练作业时选择已保存的镜像。

常见问题

1. 当镜像保存失败时如何处理?
   a. 当镜像保存失败时，请在训练作业详情页查看事件，事件描述请参考查看训练作业事件。
   b. 如使用的是专属资源池，可尝试在“资源管理 > 轻量算力资源 > 轻量算力集群”（旧版控制台：“资源管理 > 轻量算力集群”）页面按需调整容器引擎空间大小，具体步骤请参考集群扩缩容。
   c. 如果问题仍未解决，请联系技术支持。
2. 镜像保存过程中无法查看实时日志?

   镜像保存中可能会影响查看实时日志，Cloud Shell连接可能中断，镜像保存后自动恢复。

14.16 复制、停止或删除训练作业

另存为算法

当您需要修改训练作业的算法时，可以在训练作业详情页面右上角，单击“另存为算法”。

在“创建算法”页面中，会自动填充上一次训练作业的算法参数配置，您可以根据业务需求在原来算法配置基础上进行修改。

说明

1. 贵阳一新版控制台不支持此功能。
2. 订阅算法不支持另存为算法。

复制训练作业

当对创建的训练作业不满意时，您可以单击操作列的“复制”（或“克隆”），重新创建训练作业。在重新创建训练作业页面，会自动填入上一次训练作业设置的参数，您仅需在原来的基础上进行修改即可重新创建训练作业。

停止训练作业

在训练作业列表中，针对“创建中”、“等待中”、“运行中”的训练作业，您可以单击“操作”列的“终止”，在对话框中输入“YES”，然后单击“确定”，停止正在运行中的训练作业。

训练作业停止后，ModelArts将停止计费。

运行结束的训练作业，如“已完成”、“运行失败”、“已终止”、“异常”的作业，不涉及“终止”操作。

删除训练作业

如果不再需要使用此训练作业，建议清除相关资源，避免产生不必要的费用。

说明

请注意，删除训练作业后无法恢复，请谨慎操作。

- 在“训练作业”（新版为“模型训练”）页面，删除运行结束的训练作业。您可以单击“操作”列的“删除”，在弹出的提示框中，输入**DELETE**，单击“确定”，删除对应的训练作业。
- 进入OBS，删除本训练作业使用的OBS桶及文件。

14.17 管理训练容器环境变量

什么是环境变量

本章节展示了训练容器环境中预置的环境变量，方便用户查看，主要包括以下类型。

- 路径相关环境变量
- 分布式训练作业环境变量
- NCCL（Nvidia Collective multi-GPU Communication Library）环境变量
- OBS环境变量
- PIP源环境变量
- API网关地址环境变量
- 作业元信息环境变量

约束限制

为了避免新设置的环境变量与系统环境变量冲突，而引起作业运行异常或失败，请在定义自定义环境变量时，不要使用“MA_”开头的名称。

如何修改环境变量

用户可以在创建训练作业页面增加新的环境变量，也可以设置新的取值覆盖当前训练容器中预置的环境变量值。

说明

为保证数据安全，请勿输入敏感信息，例如明文密码。

训练容器中预置的环境变量

训练容器中预置的环境变量如下面表格所示，包括**表14-16**、**表14-17**、**表14-18**、**表14-19**、**表14-20**、**表14-21**、**表14-22**。

此处的环境变量取值仅为示例，涉及不同规格、引擎、Region可能取值不一样，此处仅供参考。

表 14-16 路径相关环境变量

变量名	说明	示例	默认值
PATH	可执行文件路径，已包含常用的可执行文件路径。	“PATH=/usr/local/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin”	可执行文件路径，不固定
LD_LIBRARY_PATH	动态链接库路径，已包含常用的动态库路径。	“LD_LIBRARY_PATH=/usr/local/seccomponent/lib:/usr/local/cuda/lib64:/usr/local/cuda/compat:/root/miniconda3/lib:/usr/local/lib:/usr/local/nvidia/lib64”	动态链接库路径，不固定
LIBRARY_PATH	静态库路径，已包含常用的静态库路径。	“LIBRARY_PATH=/usr/local/cuda/lib64/stubs”	静态库路径，不固定
MA_HOME	训练作业的主目录。	“MA_HOME=/home/ma-user”	/home/ma-user
MA_JOB_DIR	训练算法文件夹所在的父目录。	“MA_JOB_DIR=/home/ma-user/modelarts/user-job-dir”	/home/ma-user/modelarts/user-job-dir
MA_MOUNT_PATH	ModelArts挂载至训练容器内的路径，用于临时存放训练算法、算法输入、算法输出、日志等文件。	“MA_MOUNT_PATH=/home/ma-user/modelarts”	/home/ma-user/modelarts
MA_LOG_DIR	训练日志目录。	“MA_LOG_DIR=/home/ma-user/modelarts/log”	/home/ma-user/modelarts/log
MA_SCRIPT_INTERPRETER	训练脚本解释器。	“MA_SCRIPT_INTERPRETER=”	未设置

表 14-17 分布式训练作业环境变量

变量名	说明	示例	默认值
MA_CURRENT_IP	作业容器IP。	“MA_CURRENT_IP=192.168.23.38”	作业容器IP，不固定
MA_NUM_GPUS	作业容器的加速卡数量。	“MA_NUM_GPUS=8”	0
MA_TASK_NAME	作业容器的角色名，例如： ● MindSpore、PyTorch为worker ● 强化学习引擎为learner，worker ● TensorFlow为ps，worker	“MA_TASK_NAME=worker”	worker
MA_NUM_HOSTS	实例数。系统自动从资源参数的“实例数”中读取。	“MA_NUM_HOSTS=4”	1
VC_TASK_INDEX	当前容器索引，容器从0开始编号。单机训练的时候，该字段无意义。在多机作业中，用户可以根据这个值来确定当前容器运行的算法逻辑。	“VC_TASK_INDEX=0”	0
VC_WORKER_NUM	训练作业使用的实例数量。	“VC_WORKER_NUM=4”	1

变量名	说明	示例	默认值
VC_WORKER_HOSTS	多节点训练时，每个节点的域名地址，按顺序以英文逗号分隔，可以通过域名解析获取IP地址。	“VC_WORKER_HOSTS=modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-0.modelarts-job-a0978141-1712-4f9b-8a83-000000000000,modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-1.ob-a0978141-1712-4f9b-8a83-000000000000,modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-2.modelarts-job-a0978141-1712-4f9b-8a83-000000000000,ob-a0978141-1712-4f9b-8a83-000000000000-worker-3.modelarts-job-a0978141-1712-4f9b-8a83-000000000000”	节点的域名地址，不固定
${MA_VJ_NAME}-${MA_TASK_NAME}-N.${MA_VJ_NAME}	表示不同节点的通信域名，例如0号节点的通信域名为“${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}”。 N表示实例数。 **警告** 此方法拼接的通信域名不适用于超节点资源池。 所有资源池（包含超节点资源池）均可直接通过VC_WORKER_HOSTS获取所有节点的通信域名。	例如，实例数为4时，此环境变量分别为 “${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}”、 “${MA_VJ_NAME}-${MA_TASK_NAME}-1.${MA_VJ_NAME}”、 “${MA_VJ_NAME}-${MA_TASK_NAME}-2.${MA_VJ_NAME}”、 “${MA_VJ_NAME}-${MA_TASK_NAME}-3.${MA_VJ_NAME}”。	节点的通信域名，不固定

表 14-18 NCCL 环境变量

变量名	说明	示例	默认值
NCCL_VERSION	NCCL版本。	“NCCL_VERSION=2.7.8”	NCCL库版本，不固定
NCCL_DEBUG	NCCL日志等级。	“NCCL_DEBUG=INFO”	INFO
NCCL_IB_HCA	指定NCCL使用的IB网卡。	“NCCL_IB_HCA=^mlx5_bond_0”	IB网卡，不固定
NCCL_IB_TIMEOUT	InfiniBand传输超时时间。	“NCCL_IB_TIMEOUT=18”	18
NCCL_IB_RETRY_CNT	InfiniBand传输最大重试次数。	“NCCL_IB_RETRY_CNT=15”	15
NCCL_IB_GID_INDEX	InfiniBand在RoCE模式下使用的全局ID索引。	“NCCL_IB_GID_INDEX=3”	3
NCCL_IB_TC	InfiniBand的流量类型。	“NCCL_IB_TC=128”	128
NCCL_SOCKET_IFNAME	指定NCCL使用的SOCKET网卡。	“NCCL_SOCKET_IFNAME=bond0,eth0”	SOCKET网卡，不固定
NCCL_NET_PLUGIN	指定NCCL使用的网络插件。	“NCCL_NET_PLUGIN=none”	none

表 14-19 OBS 环境变量

变量名	说明	示例	默认值
MA_S3_ENDPOINT	OBS地址，确保应用程序能够正确地连接到指定的OBS。	“MA_S3_ENDPOINT=https://obs.region.myhuaweicloud.com”	OBS地址，不固定
S3_VERIFY_SSL	访问OBS是否使用SSL。如果设置为0，则不验证SSL证书；如果设置为1，则验证SSL证书。	“S3_VERIFY_SSL=0”	0
S3_USE_HTTPS	访问OBS是否使用HTTPS。如果设置为1，则使用HTTPS；如果设置为0，则使用HTTP。	“S3_USE_HTTPS=1”	1

表 14-20 PIP 源和 API 网关地址环境变量

变量名	说明	示例	默认值
MA_PIP_HOST	PIP源域名。	“MA_PIP_HOST=repo.example.com”	华为内部PIP源域名
MA_PIP_URL	PIP源地址。	“MA_PIP_URL=http://repo.example.com/repository/pypi/simple/”	华为内部PIP源地址

变量名	说明	示例	默认值
MA_APIGW_ENDPOINT	ModelArts API网关地址。	“MA_APIGW_ENDPOINT=https://modelarts.region.xxx.example.com”	ModelArts API网关地址，不固定

表 14-21 作业元信息环境变量

变量名	说明	示例	默认值
MA_CURRENT_INSTANCE_NAME	多节点训练时，当前节点的名称。	“MA_CURRENT_INSTANCE_NAME=modelarts-job-a0978141-1712-4f9b-8a83-0000000000000-worker-1”	当前节点的名称，不固定

表 14-22 预检相关环境变量

变量名	说明	示例	默认值
MA_SKIP_IMAGE_DETECT	ModelArts预检是否开启。1和不设置表示开启预检，0表示关闭预检。 推荐开启预检，预检可提前发现节点故障、驱动故障。默认不设置（开启预检）。	“1”	未设置

表 14-23 卡死检测相关环境变量

变量名	说明	示例	默认值
MA_HANG_DETECT_TIME	卡死检测时间。在这段时间内IO无变化则判定为任务卡死。 取值范围：10~720 单位：分钟 默认值：30	“30”	30

如何查看训练环境变量

训练环境变量可能在容器中注入，也可能在进程中注入，取决于具体的业务。进程中注入的环境变量在CloudShell中不可见。

推荐使用方式1查看全量的环境变量。

1. 通过启动命令查看训练环境变量

   在创建训练作业时，创建方式选择“自定义算法”，启动方式选择“自定义”，“启动命令”输入为“env”，其他参数保持不变。

   图 14-22 启动命令

   当训练作业执行完成后，在训练作业详情页面中查看“日志”。日志中即为所有的环境变量信息。

图 14-23 查看日志

```
NV_LIBCUBLAS_DEV_VERSION=11.3.1.68-1
NV_CUDA_COMPAT_PACKAGE=cuda-compat-11-2
NV_CUDNN_PACKAGE_DEV=libcudnn8-dev=8.1.1.33-1+cuda11.2
LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64
NV_LIBNCCL_DEV_PACKAGE=libnccl-dev=2.8.4-1+cuda11.2
MA_ENGINE_VERSION=
MA_NUM_HOSTS=1
VC_WORKER_HOSTS=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0.modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
VK_TASK_INDEX=0
_=/usr/bin/env
MA_SCRIPT_INTERPRETER=
NV_LIBNPP_DEV_PACKAGE=libnpp-dev-11-2=11.2.1.68-1
MA_MAX_BACKOFF=0
HOSTNAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0
MA_IAM_USER_ID=79098163dd814fd986eca7ef0325d086
MA_CURRENT_IP=10.0.0.62
NV_LIBNPP_VERSION=11.2.1.68-1
NV_NVPROF_DEV_PACKAGE=cuda-nvprof-11-2=11.2.67-1
MA_MOUNT_PATH=/home/ma-user/modelarts
NVIDIA_VISIBLE_DEVICES=all
MA_ENGINE_TYPE=
NV_NVPROF_VERSION=11.2.67-1
NV_LIBCUSPARSE_VERSION=11.3.1.68-1
MODELARTS_SCC_SERVICE_PORT=60687
MA_HOME=/home/ma-user
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.247.0.1
NV_LIBCUBLAS_DEV_PACKAGE=libcublas-dev-11-2=11.3.1.68-1
MA_VJ_NAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/
NCCL_VERSION=2.8.4-1
KUBERNETES_PORT=tcp://10.247.0.1:443
PWD=/
NVARCH=x86_64
HOME=/home/ma-user
```

2. 通过CloudShell查看训练环境变量

   在CloudShell中，使用命令env获取环境变量。

   这种方式无法获取训练平台在进程中注入的环境变量（如超节点场景的VC_TASK_INDEX、VC_WORKER_NUM、VC_WORKER_HOSTS等），不推荐使用。

14.18 管理训练实验

训练实验

当训练作业的数量较多时，可能会遇到难以快速定位作业或不便跟踪的情况。为了便于管理训练作业，我们引入了训练实验的概念，类似于对训练作业进行分组管理。用户可以根据需求将作业归类到不同的训练实验中，实现分类管理。每个训练实验可以包含多个同类型的作业。

在管理训练实验时，支持以下操作：将单个作业纳入实验、在已有的实验中创建作业、查看实验列表、查看实验详情或者直接删除整个实验。

说明

本章节只在旧版控制台界面存在。新版控制台已无实验内容。因此需要切换至旧版控制台进行操作。

如何将训练作业纳入实验

训练作业纳入实验的方式是在“创建训练作业”页面指定作业所属实验。有以下几个选项：

- 创建新实验：实验不支持单独创建，只支持随着训练作业创建。此处填写新建的实验名称，在提交作业成功后，会同时创建该实验，并将该作业纳入新建的实验中。此处会校验实验名称，如果重名则无法提交作业。
- 选择已有实验：您可以在下拉框中选择已有实验，将作业纳入已存在的实验。
- 不勾选实验设置：如果您不想通过实验管理您的作业，可以选择此选项。不纳入实验的作业无法在训练作业首页的“实验”中查看到。

新建纳入实验的作业

登录**ModelArts管理控制台**，选择“模型训练 > 训练作业”，单击右上角“创建训练作业”，进入“训练作业”页面。

勾选“实验设置”，并选中“创建新实验”，同时填写“新实验名称”，即可在新建训练作业时新建实验。

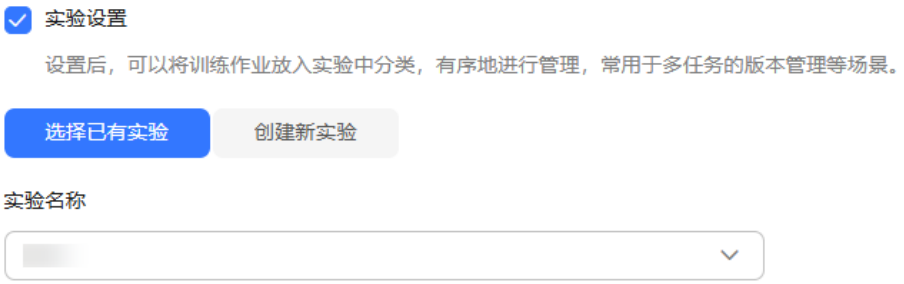
图 14-24 新建训练作业

已创建作业纳入实验

登录**ModelArts管理控制台**，选择“模型训练 > 训练作业”，选择“操作 > 复制”，进入“训练作业”页面。也可在作业列表中，单击作业“名称/ID”，进入作业详情页，单击右上角“复制”，进入“训练作业”页面。

- 未纳入实验的作业：在“实验设置”中选择“创建新实验”，填写“新实验名称”，即可在复制训练作业时新建实验。
- 已纳入实验的作业：勾选“实验设置”，并选中“选择已有实验”，选择要纳入的实验名称。

图 14-25 复制训练作业



查看实验列表

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型训练 > 训练作业”，进入“训练作业”列表。
2. 单击“实验管理”，进入实验列表。实验列表展示了实验的一些基本信息。

表 14-24 实验基本信息

参数	说明
实验名称	实验名，该名称支持在实验详情页面修改。
训练作业总数	该实验下所包含的训练作业数量。
创建时间	实验创建时间。
修改时间	实验的修改时间，指实验产生以下变化的时间： ● 修改实验名称 ● 修改实验描述 ● 实验下新增/删除了训练作业
描述	实验描述，可修改。
操作	当前仅支持删除实验。

查看实验详情

在实验列表单击某个实验名称进入实验详情页面。实验详情上方会显示实验的基本信息，下方会显示该实验下的作业列表。

图 14-26 查看实验详情

- 单击支持编辑实验名称/描述。
- 打开“只显示自己”按钮，可查看自己创建的纳入实验的作业。

说明

账号下有多个IAM用户时，默认只显示当前登录的IAM用户的作业列表。

- 按照名称、ID、算法、状态、创建时间、作业类型、资源池等条件筛选的高级搜索。
- 支持作业列表刷新。单击作业列表右上角“刷新”图标，刷新作业列表。
- 支持自定义列表选项。单击作业列表右上角“设置”图标，自定义选择需要在作业列表中显示的选项。

删除实验

警告

删除实验为高危操作，会同时删除实验下的所有作业，且不可恢复。

实验支持删除操作，您可以在实验列表页单击“删除”，或者在实验详情页右上角单击“删除实验”，进入“删除实验”页面。“删除实验”页面会展示实验下所有的作业。如确需执行该操作，请输入“DELETE”，再单击“确定”进行删除。

15 Moxing

15.1 MoXing 基础功能

场景描述

在使用ModelArts服务时，您可能会遇到需要访问对象存储服务（OBS）的情况。OBS是一个基于对象的海量存储服务，与传统的本地文件系统不同，无法直接通过文件路径访问OBS上的文件。

直接访问OBS文件时会遇到以下问题：

- 无法像访问本地文件那样使用open()方法。
- 需要通过网络请求进行文件读写操作。
- 使用OBS Python SDK接口可能较为复杂。

如何在ModelArts服务中更便捷地访问和操作OBS文件?

ModelArts提供了mox.file接口，为OBS操作提供了更简便的解决方案。以下是使用mox.file接口访问OBS文件的示例：

```
# 访问OBS文件示例
import moxing as mox

# 打开OBS文件
with mox.file.File('obs://bucket_name/a.txt', 'r') as f:
    print(f.read())

# 列举OBS目录
mox.file.list_directory('obs://bucket_name/my_dir/')
```

使用mox.file接口注意事项

1. mox.file接口主要适用于提升从OBS读取和下载数据的易用性。
2. 对于OBS并行文件系统部分接口可能存在兼容性问题，建议直接使用OBS Python SDK进行生产业务开发。
3. 详细接口说明请参见《对象存储服务 OBS》的**Python SDK接口概览**。

通过mox.file接口，您可以像操作本地文件一样便捷地访问和管理OBS资源，从而提高开发效率。

约束限制

- 需要访问的OBS桶需要满足当前训练作业，可以正常访问。
- 当前账户有读写OBS桶权限。
- 使用ModelArts标准镜像或基于自定义镜像安装**moxing**库。

文件复制

- 复制一个文件。**mox.file.copy**仅支持对文件操作，如果要对文件夹进行操作，请使用**mox.file.copy_parallel**。
  - 从OBS复制到OBS，例如将“obs://bucket_name/obs_file.txt”复制到“obs://bucket_name/obs_file_2.txt”。
    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
    ```
  - 将OBS文件复制到本地，也就是下载一个OBS文件。例如下载“obs://bucket_name/obs_file.txt”到本地“/tmp/obs_file.txt”。
    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```
  - 将本地文件复制到OBS，也就是上传一个OBS文件，例如上传“/tmp/obs_file.txt”到“obs://bucket_name/obs_file.txt”。
    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
    # 支持复制到OBS后，再从OBS预加载到缓存中：
    mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt', is_preload=True)
    ```
  - 将本地文件复制到本地，操作等价于**shutil.copyfile**，例如将“/tmp/obs_file.txt”复制到“/tmp/obs_file_2.txt”。
    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
    ```
- 对于体积较大的文件，**mox.file.copy**会默认使用分片并发下载方式进行加速，相关参数可以通过环境变量控制。
  a. 判断文件是否为大文件，超过该阈值会启用分片并发下载。
     ```
     MOX_FILE_PARTIAL_MAXIMUM_SIZE
     ```
     默认大小为5GB，设置单位为Byte，例如设置5GB需要写为5368709120。
  b. 管理文件分片大小。
     ```
     MOX_FILE_LARGE_FILE_PART_SIZE
     ```
     默认大小为10MB，设置单位为Byte。由于OBS限制分片数量最多为10000片，当文件大小超过100G时需要上调分片大小。
  c. 管理下载并发多进程数量。下载并发多进程数，表示大文件下载时会启动几个线程进行下载，新版本中默认值为8，节点较少或者大文件数量较少时，可以适当扩大并发数，提升下载性能。
     ```
     MOX_FILE_LARGE_FILE_TASK_NUM
     ```
     默认大小为32，当并发数过高导致OBS桶流控时需要调小。
- 复制一个文件夹。**mox.file.copy_parallel**仅支持对文件夹操作，如果要对文件进行操作，请使用**mox.file.copy**。
  - 从OBS复制到OBS，例如将obs://bucket_name/sub_dir_0复制到obs://bucket_name/sub_dir_1
    ```
    import moxing as mox
    mox.file.copy_parallel('obs://bucket_name/sub_dir_0', 'obs://bucket_name/sub_dir_1')
    ```
  - 将OBS文件夹复制到本地，也就是下载一个OBS文件夹。例如下载“obs://bucket_name/sub_dir_0”到本地“/tmp/sub_dir_0”。
    ```
    import moxing as mox
    mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
    ```

- 将本地文件夹复制到OBS，也就是上传一个OBS文件夹，例如上传“/tmp/sub_dir_0”到“obs://bucket_name/sub_dir_0”。
```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0')
```
- 将本地文件夹复制到本地，操作等价于**shutil.copytree**，例如将“/tmp/sub_dir_0”复制到“/tmp/sub_dir_1”。
```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', '/tmp/sub_dir_1')
```

mox.file.copy_parallel可以通过threads参数控制复制操作的并发数，该并发数默认为16。

file_list参数可以选择需要拷贝的文件。例如上传“/tmp/sub_dir_0”中的“/tmp/sub_dir_0/train/1.jpg”和“/tmp/sub_dir_0/eval/2.jpg”到“obs://bucket_name/sub_dir_0”。

```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0', file_list=['train/1.jpg', 'eval/2.jpg'])
```

读写操作

- 读取一个OBS文件。

  例如读取“obs://bucket_name/obs_file.txt”文件内容，返回string（字符串类型）。
```
import moxing as mox
file_str = mox.file.read('obs://bucket_name/obs_file.txt')
```
  也可以使用打开文件对象并读取的方式来实现，两者是等价的。
```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  file_str = f.read()
```
- 从文件中读取一行，返回string，以换行符结尾。同样可以打开OBS的文件对象。
```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  file_line = f.readline()
```
- 从文件中读取所有行，返回一个list，每个元素是一行，以换行符结尾。
```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  file_line_list = f.readlines()
```
- 以二进制模式读取一个OBS文件。

  例如读取“obs://bucket_name/obs_file.bin”文件内容，返回bytes（字节类型）。
```
import moxing as mox
file_bytes = mox.file.read('obs://bucket_name/obs_file.bin',binary=True)
```
  也可以使用打开文件对象并读取的方式来实现，两者是等价的。
```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.bin', 'rb') as f:
  file_bytes = f.read()
```
  以二进制模式打开的文件也支持读取一行或者读取所有行，用法不变。
- 将字符串写入一个文件（不支持写入超过2G的内容）。

  例如将字符串“Hello World!”写入OBS文件“obs://bucket_name/obs_file.txt”中。
```
import moxing as mox
mox.file.write('obs://bucket_name/obs_file.txt', 'Hello World!')
# 支持写入OBS后，再从OBS预加载到缓存中：
mox.file.write('obs://bucket_name/obs_file.txt', 'Hello World!', is_preload=True)
```
  也可以使用打开文件对象并写入的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'w') as f:
  f.write('Hello World!')
```

说明

用写入模式打开文件或者调用mox.file.write时，如果被写入文件不存在，则会创建，如果已经存在，则会覆盖。

- 追加一个OBS文件。

  例如将字符串“Hello World!”追加到“obs://bucket_name/obs_file.txt”文件中。

```
import moxing as mox
mox.file.append('obs://bucket_name/obs_file.txt', 'Hello World!')
```

  也可以使用打开文件对象并追加的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'a') as f:
  f.write('Hello World!')
```

  用追加模式打开文件或者调用**mox.file.append**时，如果被写入文件不存在，则会创建，如果已经存在，则直接追加。

  当被追加的源文件比较大时，例如“obs://bucket_name/obs_file.txt”文件大小超过5MB时，追加一个OBS文件的性能比较低。

说明

如果以写入模式或追加模式打开文件，当调用write方法时，待写入内容只是暂时的被存在的缓冲区，直到关闭文件对象（退出with语句时会自动关闭文件对象）或者主动调用文件对象的close()方法或flush()方法时，文件内容才会被写入。

列举操作

- 列举一个OBS目录，只返回顶层结果（相对路径），不做递归列举。

  例如列举“obs://bucket_name/object_dir”，返回该目录下所有的文件和文件夹，不会递归查询。

  假设“obs://bucket_name/object_dir”中有如下结构

```
bucket_name
    |- object_dir
      |- dir0
        |- file00
      |- file1
```

  调用如下代码：

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/object_dir')
```

  返回一个list：

```
['dir0', 'file1']
```

- 递归列举一个OBS目录，返回目录中所有的文件和文件夹（相对路径），并且会做递归查询。

  假设obs://bucket_name/object_dir中有如下结构。

```
bucket_name
    |- object_dir
      |- dir0
        |- file00
      |- file1
```

  调用如下代码：

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/object_dir', recursive=True)
```

返回一个list：

```
['dir0', 'dir0/file00', 'file1']
```

创建文件夹

创建一个OBS目录（即OBS文件夹）。支持递归创建，即如果“sub_dir_0”文件夹不存在，也会自动被创建，如果已经存在，则不起任何作用。

```
import moxing as mox
mox.file.make_dirs('obs://bucket_name/sub_dir_0/sub_dir_1')
```

查询操作

- 判断一个OBS文件是否存在，如果存在则返回**True**，如果不存在则返回**False**。
```
import moxing as mox
mox.file.exists('obs://bucket_name/sub_dir_0/file.txt')
```
- 判断一个OBS文件夹是否存在，如果存在则返回**True**，如果不存在则返回**False**。
```
import moxing as mox
mox.file.exists('obs://bucket_name/sub_dir_0/sub_dir_1')
```

说明

由于OBS允许同名的文件和文件夹（Unix操作系统不允许），如果存在同名的文件和文件夹，例如“obs://bucket_name/sub_dir_0/abc”，当调用mox.file.exists时，不论abc是文件还是文件夹，都会返回True。

- 判断一个OBS路径是否为文件夹，如果是则返回**True**，否则返回**False**。
```
import moxing as mox
mox.file.is_directory('obs://bucket_name/sub_dir_0/sub_dir_1')
```

说明

由于OBS允许同名的文件和文件夹（Unix操作系统不允许），如果存在同名的文件和文件夹，例如obs://bucket_name/sub_dir_0/abc，当调用mox.file.is_directory时，会返回True。

- 获取一个OBS文件的大小，单位为bytes。

  例如获取“obs://bucket_name/obs_file.txt”的文件大小。
```
import moxing as mox
mox.file.get_size('obs://bucket_name/obs_file.txt')
```
- 递归获取一个OBS文件夹下所有文件的大小，单位为bytes。

  例如获取“obs://bucket_name/object_dir”目录下所有文件大小的总和。
```
import moxing as mox
mox.file.get_size('obs://bucket_name/object_dir', recursive=True)
```
- 获取一个OBS文件或文件夹的stat信息，stat信息中包含如下信息。
  - length：文件大小。
  - mtime_nsec：创建时间戳。
  - is_directory：是否为目录。

  例如查询一个OBS文件“obs://bucket_name/obs_file.txt”，此文件地址也可以替换成一个文件夹地址。
```
import moxing as mox
stat = mox.file.stat('obs://bucket_name/obs_file.txt')
print(stat.length)
print(stat.mtime_nsec)
print(stat.is_directory)
```

删除操作

- 删除一个OBS文件。

  例如删除“obs://bucket_name/obs_file.txt”。

```
import moxing as mox
mox.file.remove('obs://bucket_name/obs_file.txt')
```

- 删除一个OBS目录，并且递归的删除这个目录下的所有内容。如果这个目录不存在，则会报错。

  例如删除“obs://bucket_name/sub_dir_0”下的所有内容。

```
import moxing as mox
mox.file.remove('obs://bucket_name/sub_dir_0', recursive=True)
```

设置参数

- 使用mox.file.set_auth函数设置所有可配置的参数。

```
#配置moxing读取的参数，具体如下，对于不需要修改配置项不用显示指定即可，该API全局执行一次，不能多次执行
#ak - Access Key ， 字符串类型，在ModelArts使用时使用时，系统默认配置，其他环境参考统一身份认证
#sk - Secret Access Key，字符串类型，在ModelArts使用时使用时，系统默认配置，其他环境参考统一身份认证
#server - OBS 服务端， 在ModelArts使用时，系统默认配置，其他环境配置参考https://support.huaweicloud.com/productdesc-obs/obs_03_0152.html
#port - OBS服务器端口， 在ModelArts使用时，系统默认配置，其他环境配置参考server，一般不会给定特殊端口号，不需要配置
#is_secure - 是否使用https, 布尔类型，默认值为True
#ssl_verify - 是否使用ssl验证，布尔类型，默认值为False
#long_conn_mode - 是否使用长连接模式，布尔类型，默认配置为True
#path_style - 是否使用路径风格. 布尔类型，HEC则默认设置为True，当为公有云则默认设置为False，默认不建议修改
#retry - 总共尝试的次数，类型为整型，默认值为10，单位为次
#retry_wait - 在每次尝试时所等待的时间，类型为浮点型，默认值为0.1，单位秒，若不配置，则第一次等待0.1秒，第二次失败等待0.2秒，第三次为0.4秒，指数递增#client_timeout - obs客户端超时时间， 类型为整型，默认值为30， 单位秒#list_max_keys - 每页列出的最大对象数，用于list_directory api，类型为整型，默认值为1000， 单位为个
#timeout_config - 设置moxing各个函数的默认超时时间列表，单位为秒，默认为
{
  'read': 60,  # read函数超时时间
  'listObjects': 10,  # list_directory函数列举1000个对象的超时时间
  'deleteObject': 60,  # rename，remove函数删除对象的超时时间
  'getObject': 60,  # copy函数本地获取obs对象的超时时间
  'getObjectMetadata': 30,  #copy，read等函数获取obs对象元数据的超时时间
  'putFile': 120,  # copy函数本地上传obs小于5G文件的超时时间
  'putContent': 30,  # copy_parallel函数创建obs目录超时时间
  'copyPart': 120,  # copy函数obs之间拷贝文件分片超时时间
  'copyObject': 120,  # copy函数obs之间拷贝超时时间
  'uploadPart': 120,  # copy函数本地上传obs文件分片的超时时间
  'initiateMultipartUpload': 30,  # copy函数本地上传obs大于5G文件的初始化超时时间
  'completeMultipartUpload': 120,  # copy函数本地上传obs小于5G文件的完成超时时间
} # 当遇到对应场景超时时，可修改对应函数的超时时间
import moxing as mox
mox.file.set_auth(ak='xxx', sk='xxx')
```

- 删除一个OBS目录，并且递归的删除这个目录下的所有内容。如果这个目录不存在，则会报错。

  例如删除“obs://bucket_name/sub_dir_0”下的所有内容。

```
import moxing as mox
mox.file.remove('obs://bucket_name/sub_dir_0', recursive=True)
```

16 待下线

16.1 创建生产训练作业（旧版）

在模型开发过程中，如何高效地优化模型性能是一个关键挑战。传统的模型训练方式往往需要反复尝试不同的模型结构、数据集和超参数组合，这不仅耗时耗力，还可能导致训练效果不理想。针对这一问题，ModelArts的训练模块提供了一套完整的解决方案。它支持创建训练作业、实时监控训练进展以及管理训练版本，帮助用户系统化地优化模型性能。通过ModelArts，用户可以轻松试验不同的模型结构、数据集和超参数组合，从而快速找到最优的模型配置和权重。

创建生产环境的训练作业有2种方式：

- 通过ModelArts控制台的方式创建生产环境的训练作业，详细操作请参考本章节以下内容。本文提供了默认页面（即旧版页面）的操作指导，新版页面的操作指导请参见创建生产训练作业。
- 通过ModelArts提供的API接口创建生产环境的训练作业，详细操作请参见使用API创建训练作业。

约束限制

默认支持创建10000条训练作业，剩余额度可以在训练作业列表页查看。

图 16-1 查看训练作业剩余额度

前提条件

- 已经将用于训练作业的数据上传至OBS目录。
- 已经在OBS目录下创建了至少1个空的文件夹，用于存储训练输出的内容。

说明

ModelArts不支持加密的OBS桶，创建OBS桶时，请勿开启桶加密。

- 由于训练作业运行需消耗资源，为了避免训练失败请确保账户未欠费。
- 确保使用的OBS目录与ModelArts在同一区域。
- 检查是否配置了访问授权。如果未配置，请参见**快速配置ModelArts委托授权**完成操作。
- 已经准备好训练算法，具体操作请参见**创建算法**。

计费影响

在ModelArts进行模型训练时，会产生计算资源和存储资源的累计值计费。计算资源为训练作业运行的费用。存储资源包括数据存储到OBS或SFS的费用。详见**模型训练计费项**。

操作流程介绍

创建训练作业的操作步骤如下所示。

步骤1 **进入创建训练作业页面**。

步骤2 **配置训练作业基本信息**。

步骤3 根据不同的算法来源，选择不同的训练作业创建方式。

- 使用已有算法创建训练作业：**选择创建方式（使用我的算法）**
- 使用订阅算法创建训练作业：**选择创建方式（使用订阅算法）**
- 使用预置镜像创建训练作业：**选择创建方式（自定义算法）**
- 使用自定义镜像创建训练作业：**选择创建方式（使用自定义镜像）**

步骤4 **配置训练参数**：配置训练作业的输入、输出、超参、环境变量等参数。

步骤5 根据需要选择不同的资源池用于训练作业，推荐使用专属资源池。

- **配置资源池（公共资源池）**
- **配置资源池（专属资源池）**

步骤6 **（可选）选择训练模式**：当训练作业的算法框架选用的是预置框架的MindSpore类引擎、资源池类型选用的是Ascend资源时，则支持选择训练模式。

步骤7 **（可选）设置标签**：如果需要对训练作业进行资源分组管理，可以设置标签。

步骤8 **后续操作**。

----结束

进入创建训练作业页面

1. 登录**ModelArts管理控制台**。
2. 在左侧导航栏中，选择“模型训练 > 训练作业”进入训练作业列表。
3. 单击“创建训练作业”，右上角单击“返回旧版”，进入旧版创建训练作业页面。

配置训练作业基本信息

在创建训练作业页面填写训练作业基本信息。

表 16-1 创建训练作业的基本信息

参数名称	说明
名称	必填，训练作业的名称。 系统会自动生成一个名称，可以根据业务需求重新命名，命名规则如下： ● 支持1~64位字符。 ● 可以包含大小写字母、数字、中划线（-）或下划线（_）。
描述	训练作业的简介，便于在训练作业列表了解作业信息。
设置实验	将训练作业分类有序地放入实验中进行管理。 ● 如果选择“纳入新实验”，需要配置“新建实验名称”和“新建实验描述”。 ● 如果选择“纳入已有实验”，需要选择“实验名称”。 ● 如果选择“不纳入实验”，则不在实验中进行统一管理。

选择创建方式（使用我的算法）

如果选择使用已有算法创建训练作业，则“创建方式”选择“我的算法”，在算法列表中选择算法。如果没有满足条件的算法，也可以新建算法，具体操作请参见创建算法。

选择创建方式（使用订阅算法）

如果选择使用订阅算法创建训练作业，则“创建方式”选择“订阅算法”，在算法列表中选择算法。如果没有满足条件的算法，也可以前往AI Gallery中订阅算法。

1. 登录AI Gallery。
2. 选择“模型”，进入模型列表页面，该页面展示了所有共享的模型，包括ModelArts算法。
3. 单击目标算法进入详情页面。
4. 在详情页面选择“训练 > ModelArts”，在弹出的“选择ModelArts的云服务区域”页面选择需要使用算法的区域，单击“确认”跳转至ModelArts控制台的“算法管理 > 订阅算法”页面。

   订阅后的算法，将自动展现在“算法管理 > 订阅算法”页面中。

选择创建方式（自定义算法）

如果在算法管理中已经创建算法，此处建议选择“我的算法”页签中已经准备好的算法。如果没有已经准备好的算法，此处可以选择“自定义算法”创建方式。如果选择使用自定义算法创建训练作业，则参考表16-2选择训练作业的创建方式。

表 16-2 创建训练作业的创建方式（使用自定义算法）

参数名称	说明
创建方式	必选，选择“自定义算法”。
启动方式	必选，选择“预置框架”，并选择训练作业要使用的预置框架引擎和引擎版本。 如果引擎版本选择“自定义”，则需要配置“镜像”参数，选择自定义镜像用于训练作业。
镜像	仅当预置框架的引擎版本选择“自定义”时才显示该参数，且是必填参数。 容器镜像地址的填写支持如下方式。 ● 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 ● 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息，系统会自动拼接域名地址。
代码来源	选择训练代码来源。 ● 对象OBS存储：如果训练代码存放在OBS中，则选择“对象OBS存储”。 ● 文件存储：如果训练代码存放在文件存储中，则选择“文件存储”。
代码目录	仅当“代码来源”选择“对象OBS存储”时才显示该参数。 必填，选择训练代码文件所在的OBS目录。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。 **说明** 如果您的OBS桶中有需要特别保护的敏感数据，请自行对数据加密后再存 放到OBS桶。
启动文件	必填，选择代码目录中训练作业的Python启动脚本。 ModelArts只支持使用Python语言编写的启动文件，因此启动文件必须以“.py”结尾。

参数名称	说明
本地代码目录	仅当“代码来源”选择“对象存储服务OBS”时才显示该参数。可选参数。 用于指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。 本地代码目录不支持在/home/ma-user/modelarts/*，/home/ma-user/modelarts-dev/*，/home/ma-user/infer/*目录下，也不支持配置成/home/ma-user。 单击页面右上角的“运行环境预览”，可以查看训练作业的工作目录。
工作目录	训练时，系统会自动cd到此目录下执行启动文件。

选择预置框架+自定义时，该功能的后台行为与直接基于预置框架运行训练作业相同，例如：

- 系统将会自动注入一系列环境变量。
```
PATH=${MA_HOME}/anaconda/bin:${PATH}
LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
```
- 您选择的启动文件将会被系统自动以python命令直接启动，因此请确保镜像中的Python命令为您预期的Python环境。注意到系统自动注入的PATH环境变量，您可以参考下述命令确认训练作业最终使用的Python版本：
```
export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
docker run --rm {image} $(which python) -V
```
- 系统将会自动添加预置框架关联的超参。

选择创建方式（使用自定义镜像）

如果选择使用自定义镜像创建训练作业，则参考**表16-3**选择训练作业的创建方式。

表 16-3 创建训练作业的创建方式（使用自定义镜像）

参数名称	说明
创建方式	必选，选择“自定义算法”。
启动方式	必选，选择“自定义”。
镜像	必填，填写容器镜像的地址。 容器镜像地址的填写支持如下方式。 ● 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 ● 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息，系统会自动拼接域名地址。

参数名称	说明
代码目录	选择训练代码文件所在的OBS目录。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。
运行用户ID	容器运行时的用户ID，该参数为选填参数，建议使用默认值1000。 如果需要指定uid，则uid数值需要在规定范围内，不同资源池的uid范围如下： ● 公共资源池：1000-65535 ● 专属资源池：0-65535 如果运行用户ID配置为0，则训练容器中的运行用户是**root**。

启动命令	必填，镜像的启动命令。
	运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。
	● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。
	`python ${MA_JOB_DIR}/demo-code/train.py`
	● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。
	当存在输入管道、输出管道、或是超参的情况下，请保证启动命令的最后一条命令是运行训练脚本。
	原因：系统会将输入管道、输出管道、以及超参添加到启动命令的末尾，如果最后一条命令不是运行训练脚本则会报错。
	例如：启动命令的最后一条是python train.py，且存在--data_url超参，系统正常运行会执行python train.py --data_url=/input。但是当启动命令python train.py后面有其他命令时，如下所示：
	`python train.py`
	`pwd  #反例，启动命令的最后一条命令不是运行训练脚本，而是pwd`
	此时，如果拼接了输入管道、输出管道、以及超参，系统运行实际执行的是python train.py pwd --data_url=/input，就会报错。
	说明
	为保证数据安全，请勿输入敏感信息，例如明文密码。

参数名称	说明
本地代码目录	仅当“代码来源”选择“对象存储服务OBS”时才显示该参数。可选参数。 用于指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。 本地代码目录不支持在/home/ma-user/modelarts/*，/home/ma-user/modelarts-dev/*，/home/ma-user/infer/*目录下，也不支持配置成/home/ma-user。 单击页面右上角的“运行环境预览”，可以查看训练作业的工作目录。
工作目录	训练时，系统会自动cd到此目录下执行启动文件。

训练支持的自定义镜像使用说明请参考自定义镜像的启动命令规范。

配置训练参数

训练过程中可以从OBS桶或者数据集中获取输入数据进行模型训练，训练输出的结果也支持存储至OBS桶中。创建训练作业时可以参考表16-4配置输入、输出、超参、环境变量等参数。

说明

创建训练作业时选择的创建方式不同，训练作业的输入、输出和超参显示不同。如果参数值置灰，即表示该参数已经在算法代码中配置了且不支持修改。

表 16-4 配置训练参数

参数名称	子参数	说明
输入	参数名称	算法代码需要通过“输入”的“参数名称”去读取训练的输入数据。 建议设置为“data_url”。训练输入参数要与所选算法的“输入”参数匹配，请参见创建算法时的表5-4。
	数据集	单击“数据集”，在ModelArts数据集列表中勾选目标数据集并选择对应的版本。 训练启动时，系统将自动下载输入路径中的数据到训练运行容器。
	数据存储位置	单击“数据存储位置”，从OBS桶中选择训练输入数据的存储位置。文件总大小要小于或等于10GB，文件数要小于或等于1000个，单个文件大小要小于或等于1GB。 训练启动时，系统将自动下载输入路径中的数据到训练运行容器。

参数名称	子参数	说明
	获取方式	以参数名称为“data_path”的训练输入为例，说明获取方式的作用。 ● 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 import argparse parser = argparse.ArgumentParser() parser.add_argument('--data_path') args, unknown = parser.parse_known_args() data_path = args.data_path ● 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 import os data_path = os.getenv("data_path", "")
输出	参数名称	算法代码需要通过“输出”的“参数名称”去读取训练的输出目录。 建议设置为“train_url”。训练输出参数要与所选算法的“输出”参数匹配，请参见创建算法时的表5-5。
	数据存储位置	单击“数据存储位置”，从OBS桶中选择训练输出数据的存储位置。文件总大小要小于或等于1GB，文件数要小于或等于128个，单个文件大小要小于或等于128MB。 训练过程中，系统将自动从训练容器的本地代码目录下同步文件到数据存储位置。 **说明** 数据存储位置仅支持OBS路径。为避免数据存储冲突，建议选择一个空目录用作“数据存储位置”。
	获取方式	以参数名称为“train_url”的训练输出为例，说明获取方式的作用。 ● 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 import argparse parser = argparse.ArgumentParser() parser.add_argument('--train_url') args, unknown = parser.parse_known_args() train_url = args.train_url ● 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 import os train_url = os.getenv("train_url", "")
	预下载至本地目录	选择是否将输出目录下的文件预下载至本地目录。 ● 不下载：表示启动训练作业时不会将输出数据的存储位置中的文件下载到训练容器的本地代码目录中。 ● 下载：表示系统会在启动训练作业时自动将输出数据的存储位置中的所有文件下载到训练容器的本地代码目录中。下载时间会随着文件变大而变长，为了防止训练时间过长，请及时清理训练容器的本地代码目录中的无用文件。如果要使用设置断点续训练，则必须选择“下载”。

参数名称	子参数	说明
超参	-	超参用于训练调优。此参数由选择的算法决定，如果在算法中已经定义了超参，则此处会显示算法中所有的超参。 超参支持修改和删除，状态取决于算法中的超参“约束”设置，详情请参见表5-6。 单击“本地上传”可以本地批量导入超参，需要按模板填写超参且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
环境变量	-	根据业务需求增加环境变量。训练容器中预置的环境变量请参见管理训练容器环境变量。 单击“本地上传”可以本地批量导入环境变量，需要按模板填写环境变量且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
自动重启	-	打开开关后，可以设置重启次数和是否启用无条件自动重启、作业卡死重启。 打开自动重启开关后，当由于环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见设置断点续训练。 “重启次数”的取值范围是1~128，缺省值为3。创建训练后不支持修改重启次数，请合理设置次数。 勾选“无条件自动重启”后，只要系统检测到训练异常，就无条件重启训练作业。为了避免无效重启浪费算力资源，系统最多只支持连续无条件重启3次。 系统支持自动监控作业进程的状态和资源利用率来判定作业是否卡死，开启“作业卡死重启”开关后，支持将标记为卡死的作业进行进程级自动重启，以提高资源使用率。CPU规格不支持作业卡死重启。因系统无法核实代码逻辑且检测存在周期性，卡死检测存在一定的误报概率，开启开关即表示接受误报率。为了避免无效重启浪费算力资源，系统最多只支持连续作业卡死重启3次。 当训练过程中触发了自动重启，则系统会记录重启信息，在训练作业详情页可以查看故障恢复详情，具体请参见训练作业容错检查。
算子重执行	-	默认不展示，需要符合一定的条件才会展示“算子重执行”开关。 开启算子重执行，可以解决当超节点网络链路故障时导致的训练作业通信异常的问题。 开启对训练任务的性能有较大影响。 详细介绍和约束条件请参见配置算子重执行章节。

配置资源池（公共资源池）

如果使用公共资源池创建训练作业，则参考表16-5配置公共资源池。

表 16-5 创建训练作业的公共资源池

参数名称	说明
资源池	必选，选择“公共资源池”。
资源类型	必选，选择训练需要的资源类型。 当训练代码中已定义资源类型时，则根据算法的约束条件选择合适的资源类型。例如，训练代码中定义的资源类型为CPU，这里选择其他类型时会导致训练作业失败。 如果部分资源类型不可见或不可选，表示不支持。
实例规格	必选，根据不同的资源类型，选择所需的资源规格。 当“输入”参数选择“数据存储位置”时，在选择资源池规格时可以单击右侧的“获取输入数据大小”，检查输入数据的大小是否超出数据盘的容量限制，避免训练过程中出现内存不足的情况。 **说明** • 实例规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。 • 使用自定义镜像（创建方式为自定义算法，启动方式为自定义）创建作业时，请确保您的“镜像”类型、“资源类型”和“实例规格”类型保持一致（例如：均为GPU类型）。否则会导致训练作业失败。
实例数	必填，根据需要选择实例数的个数。默认值为“1”。 • 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 • 当“实例数 > 1”时，创建的是分布式训练作业，更多分布式训练配置请参见分布式训练功能介绍。 在创建分布式训练作业前，建议提前安装所有的pip依赖（请参见在镜像中安装pip依赖），否则，当节点数大于10时，系统会自动删除pip源配置，当训练过程中涉及pip install操作时可能会训练失败。
永久保存日志	选择CPU或者GPU资源时，支持选择是否关闭“永久保存日志”开关。 • 开关打开（默认打开）：表示永久保存日志，此时必须配置“作业日志路径”，系统会将训练日志永久保存至指定的OBS路径。 • 开关关闭：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
作业日志路径	选择Ascend资源时或者打开“永久保存日志”开关时，必须配置“作业日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。

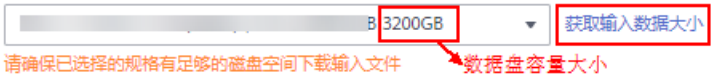
参数名称	说明
事件通知	选择是否打开“事件通知”开关。 ● 开关关闭（默认关闭）：表示不启用消息通知服务。 ● 开关打开：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”、“作业终止”、“作业疑似卡死”等。 **说明** ● 需要为消息通知服务中创建的主题添加订阅，当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 ● 使用消息通知服务会产生相关服务费用，详细信息请参见**计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
自动停止	当使用付费资源时，可以选择是否打开“自动停止”开关。 ● 开关关闭（默认关闭）：表示训练作业将一直运行直至训练完成。 ● 开关打开：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。
配置节点间SSH免密互信	选择是否打开“配置节点间SSH免密互信”开关。 ● 开关关闭（默认关闭）：表示不配置训练作业节点间SSH免密互信。 ● 开关打开：表示配置训练作业节点间SSH免密互信，此时必须配置“SSH密钥目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。

配置资源池（专属资源池）

如果使用专属资源池创建训练作业，则参考**表16-6**配置专属资源池。

表 16-6 创建训练作业的专属资源池

参数名称	说明
资源池	必选，选择“专属资源池”并选择要使用的资源池。 选择专属资源池时，支持查看当前资源池的名称、状态、节点规格、空闲/碎片节点数、可用节点/总节点数以及卡数信息。单击“空闲/碎片节点数”列的“查看”可以查看碎片详情，确认资源池是否满足训练需求。

参数名称	说明
实例规格	必选，根据不同的资源类型，选择所需的资源规格。 当“输入”参数选择“数据存储位置”时，在选择资源池规格时可以单击右侧的“获取输入数据大小”，检查输入数据的大小是否超出数据盘的容量限制，避免训练过程中出现内存不足的情况。  **说明** 资源规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。
实例数	必填，根据需要选择实例的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数> 1”时，创建的是分布式训练作业，更多分布式训练配置请参见**分布式训练功能介绍**。 在创建分布式训练作业前，建议提前安装所有的pip依赖（请参见**在镜像中安装pip依赖**），否则，当节点数大于10时，系统会自动删除pip源配置，当训练过程中涉及pip install操作时可能会训练失败。
超节点亲和组实例数	● “资源池”选择“专属资源池”，且“节点规格”为Snt9b23系列情况下，支持配置超节点亲和组实例数。 ● 当选择超节点资源池时，支持配置超节点亲和组实例数。如果配置超节点亲和组实例数为n，则每n个Pod会被调度到同一个超节点内，从而实现亲和性作业调度。在分布式训练中亲和性作业调度可以保证计算资源的同构性。 ● 如果配置了“超节点亲和组实例数”，则“实例数”必须设置为超节点亲和组实例数的整数倍，否则无法创建训练作业。例如，超节点亲和组实例数为n，则实例数为n或2n。 ● 超节点亲和组实例配置更多内容请参见**超节点亲和组实例数配置**章节。
作业优先级	● 使用专属资源池创建训练作业时，支持使用和修改训练作业的调度优先级。使用公共资源池时，不支持此参数配置。 ● 作业默认按照优先级从高到低依次处理，对于相同优先级的任务，则根据提交时间的先后顺序进行调度，最终结合实际的资源情况决定调度顺序(保证在资源充足、相同优先级下，先提交的任务先调度)。 ● 修改选择数字后可修改作业进入队列的优先级。优先级可以设置数字1-3，数字越大，优先级越高，默认优先级为“1”，最高优先级为“3”。 ● 设置优先级为“3”时，还需要配置用户权限才会生效。用户权限设置请参见“**设置作业为高优先级权限**”。 ● 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。

参数名称	说明
允许被抢占	开启后，当资源池剩余资源不足时，允许被抢占的作业可能会被终止并重新排队。为避免训练进度丢失，请在开启此功能前完成断点续训的设置。操作指导请参考设置断点续训练。
SFS Turbo	当ModelArts和SFS Turbo间网络直通时，训练作业支持挂载多个SFS Turbo存放训练数据。单击“增加挂载配置”，填写如下参数。 ● “文件系统”：选择一个SFS Turbo。 ● “云上挂载路径”：输入SFS Turbo对应在训练容器内的云上挂载路径。 ● “存储位置”：选择SFS Turbo的存储位置。如果用户配置了文件夹控制权限，请选择存储位置；如果用户未配置文件夹控制权限，可以保持默认值“/”或者自定义位置。 ● “挂载方式”：显示挂载SFS Turbo的权限。根据SFS Turbo存储位置的权限显示“读写”或“只读”，如果用户未配置文件夹控制权限，则该参数不可见。 ● “挂载参数”：支持配置SFS挂载参数实现训练加速优化，具体参数说明请参见设置极速文件存储挂载参数。不设置时，默认配置如下参数： mountOptions: - vers=3 - timeo=600 - nolock - hard **说明** ● 文件系统支持重复挂载，但是挂载路径不可重复。最多可以挂载5个盘。 ● 如果要使用训练作业挂载SFS Turbo功能，需要配置ModelArts和SFS Turbo间网络直通，具体操作请参见配置ModelArts和SFS Turbo间网络直通。 ● 云上挂载路径有如下限制：不能为 / 目录，不能为 /cache、 /home/ma-user/modelarts等系统已经默认挂载的路径。 ● 如果需要设置SFS Turbo的文件夹权限，请参考权限管理文档配置。
永久保存日志	选择CPU或者GPU资源时，支持选择是否关闭“永久保存日志”开关。 ● 开关打开（默认打开）：表示永久保存日志，此时必须配置“作业日志路径”，系统会将训练日志永久保存至指定的OBS路径。 ● 开关关闭：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
作业日志路径	选择Ascend资源时或者打开“永久保存日志”开关时，必须配置“作业日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。

参数名称	说明
事件通知	选择是否打开“事件通知”开关。 ● 开关关闭（默认关闭）：表示不启用消息通知服务。 ● 开关打开：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”、“作业终止”、“作业疑似卡死”等。 **说明** ● 需要为消息通知服务中创建的主题添加订阅，当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 ● 使用消息通知服务会产生相关服务费用，详细信息请参见**计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
自动停止	当使用付费资源时，可以选择是否打开“自动停止”开关。 ● 开关关闭（默认关闭）：表示训练作业将一直运行直至训练完成。 ● 开关打开：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。
配置节点间SSH免密互信	选择是否打开“配置节点间SSH免密互信”开关。 ● 开关关闭（默认关闭）：表示不配置训练作业节点间SSH免密互信。 ● 开关打开：表示配置训练作业节点间SSH免密互信，此时必须配置“SSH密钥目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。

（可选）选择训练模式

当训练作业的算法框架选用的是预置框架的MindSpore类引擎、资源池类型选用的是Ascend资源时，则支持选择训练模式。ModelArts提供了3种训练模式供用户选择，支持根据实际场景获取不同的诊断信息。

- 普通模式：默认训练场景。
- 高性能模式：最小化调测信息，可以提升运行速度，适合于网络稳定并追求高性能的场景。
- 故障诊断模式：收集更多的信息用于定位，适合于执行出现问题需要收集故障信息进行定位的场景。此模式提供故障诊断，用户可以根据实际需求选择诊断类别。

（可选）设置标签

如果需要通过标签实现资源分组管理，可以在“高级选项”处勾选“现在配置”，可以设置训练作业的“标签”。标签详细用法请参见**使用TMS标签实现资源分组管理**。

后续操作

当创建训练作业的参数配置完成后，单击“提交”，在信息确认页面单击“确定”，提交创建训练作业任务。

训练作业一般需要运行一段时间，前往训练作业列表，可以查看训练作业的基本情况。

- 在训练作业列表中，刚创建的训练作业状态为“等待中”。
- 当训练作业的状态变为“已完成”时，表示训练作业运行结束，其生成的模型将存储至对应的“输出”目录中。
- 当训练作业的状态变为“运行失败”或“异常”时，可以单击训练作业的名称进入详情页面，通过查看日志等手段处理问题。

常见问题

1. **在ModelArts中训练好后的模型如何获取?**
2. **在ModelArts上训练模型如何配置输入输出数据?**
3. **在ModelArts上如何提升训练效率并减少与OBS的交互?**
4. **如何查看ModelArts训练作业资源占用情况?**