盘古大模型

用户指南

文档版本 01

发布日期 2024-08-31

华为云计算技术有限公司

商标声明


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 盘古大模型套件使用流程

盘古大模型套件平台是一款功能强大、集成度高的大模型开发与应用平台。该平台全面支持大模型的数据管理、清洗与配比，涵盖预训练与微调功能。此外，平台还提供了强大的模型部署、评估与调用功能，确保模型能够在生产环境中高效应用。平台支持提示词工程、AI助手及SDK开发，满足多样化业务需求，助力企业在大模型领域取得卓越成果。无论是开发、训练还是部署，盘古大模型套件平台均为用户提供一站式解决方案。

通过使用盘古大模型套件平台，您将体验从数据准备到模型应用的全流程一站式服务，将模型高效集成至您的业务流程中。接下来，将详细介绍该平台的使用流程，帮助您充分发挥盘古大模型套件平台的潜力。

图 1-1 盘古大模型套件使用流程

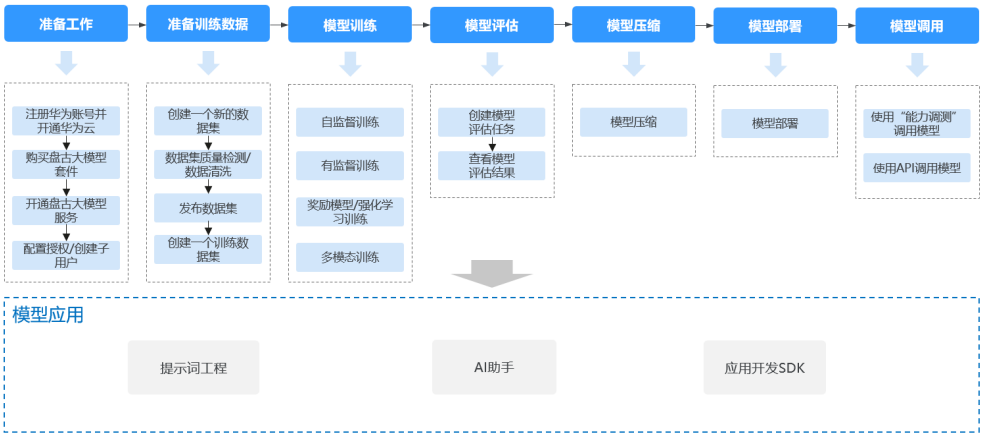


表 1-1 使用流程说明

流程	子流程	说明	操作指导
准备工作	注册华为账号并开通华为云	在使用华为云服务之前您需要注册华为账号并开通华为云。	注册华为账号并开通华为云
	购买盘古大模型套件	购买盘古系列大模型及推理资产。	购买盘古大模型套件

流程	子流程	说明	操作指导
	开通盘古大模型服务	开通大模型的文本补全、多轮对话能力。	开通盘古大模型服务
	配置授权/创建子用户	配置盘古访问OBS服务权限，多用户使用平台情况下需要创建子用户。	配置盘古访问授权 创建子用户并授权使用盘古
准备训练数据	创建一个新的数据集	创建一个新的数据集，用来管理上传至平台的训练或者评测数据。	创建一个新的数据集
	数据集质量检测/数据清洗	对上传的数据进行质量检测，若质量有问题可以进行数据清洗。	检测数据集质量 清洗数据集（可选）
	发布数据集	对无质量问题的数据集执行发布操作。	发布数据集
	创建一个训练数据集	通过数据配比组合多个数据集，创建出用于模型训练的数据集。	创建一个训练数据集
模型训练	自监督训练	使用不含有标记的数据进行模型训练。	创建自监督微调训练任务
	有监督训练	使用含有标记的数据进行模型训练，以学习输入和输出之间的映射关系。	创建有监督训练任务
模型评估	创建模型评估任务	训练完成后评估模型的回答效果。	创建模型评估任务
	查看模型评估结果	查看模型评估指标和评估结果。	查看评估任务详情
模型压缩	-	通过模型压缩技术实现同等QPS目标下，降低推理显存占用。	压缩盘古大模型
模型部署	-	对模型执行部署操作。	部署盘古大模型
模型调用	使用“能力调测”调用模型	使用可视化的“能力调测”页面调用模型。	使用“能力调测”调用模型
	使用API调用模型	通过API编写代码方式调用模型。	使用API调用模型
提示词工程	-	利用精心设计的提示词优化和引导大模型生成更加准确和相关的输出，提高模型在特定任务中的表现。	提示词工程

流程	子流程	说明	操作指导
AI助手	-	通过大模型搭建Agent应用，并结合多种工具，实现对话问答、规划推理和逻辑判断功能。	**AI助手**
应用开发SDK	-	通过应用开发SDK提供的大模型调用、提示词模板、记忆、技能、智能代理等功能模块，快速开发大模型应用。	**盘古应用开发SDK**

2 准备工作

2.1 注册华为账号并开通华为云

注册华为账号并开通华为云

在使用华为云服务之前，您需要先注册华为账号并开通华为云。通过此账号，您可以按需付费，灵活使用所有华为云提供的服务。

进入**华为云**官网，参考**账号注册**指导及界面提示信息，完成账号注册。

注册成功后即可自动登录华为云，您需要完成“实名认证”才可以正常使用服务。具体认证方式请参见**实名认证**。

获取账号信息

在调用服务API、SDK时，需要将账号相关的信息作为API凭证传入代码。

API凭证主要包括：IAM用户名、IAM用户名ID、账号名、账号ID、项目ID、项目、所属区域。可登录控制台在**“我的凭证 > API凭证”**页面获取。

图 2-1 获取账号信息

2.2 购买盘古大模型套件

在购买盘古大模型套件之前，您可以通过“能力调测”功能体验平台预置的模型，请参见**体验盘古预置模型能力**。

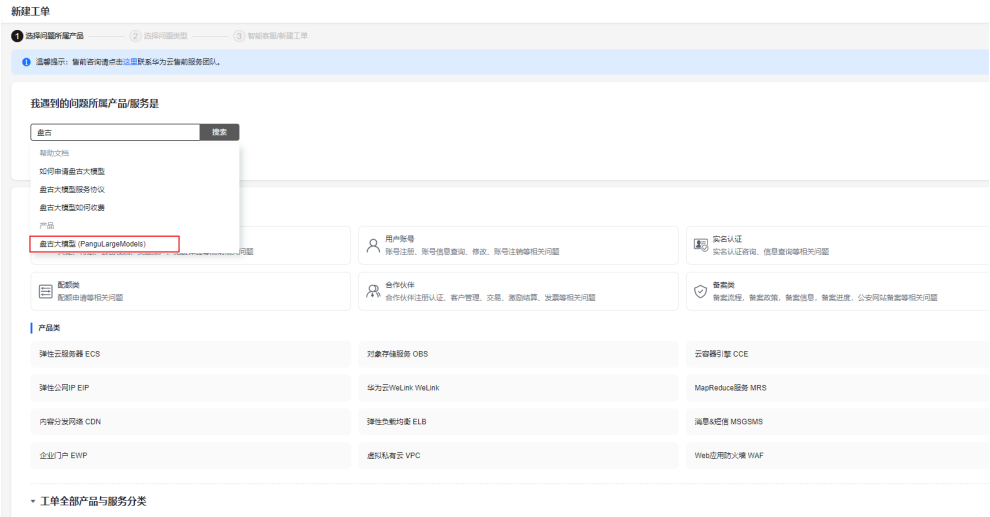
盘古大模型套件在订购时分为模型资产和模型推理资产。

- 模型资产即盘古系列大模型，用户可以订购盘古基模型、功能模型、专业大模型。
  - **基模型：**基模型经过大规模数据的预训练，能够学习并理解多种复杂特征和模式。这些模型可作为各种任务的基础，包括但不限于阅读理解、文本生成和情感分析等，但不具备对话问答能力。
  - **功能模型：**功能模型是在基模型的基础上经过微调，专门适应特定任务，并具备对话问答的能力。经过特定场景优化的功能模型能够更有效地处理文案生成、阅读理解、代码生成等任务。
  - **专业大模型：**针对特定场景优化的大模型。例如，与非专业大模型相比，BI专业大模型更适合执行数据分析、报告生成和业务洞察等任务。
- 模型推理资产即部署模型所需的cpu、gpu资源（专属资源池）。如果不订购推理资产，可以使用订购的盘古模型进行训练，但无法部署训练后的模型。

1. 登录**盘古大模型套件平台**。
2. 在服务“总览”页面，单击“立即购买”，平台将为您提交购买权限申请。如您有加急购买需求，可在页面右上角单击“工单 > 新建工单”，搜索“盘古大模型”产品，选择问题类型并提交工单。

图 2-2 立即购买

图 2-3 新建工单



3. 获取购买权限后，您可在购买页面选择合适的模型和推理资产，购买盘古大模型套件。

图 2-4 购买盘古大模型套件

说明

- 对于前期邀测用户，如果未购买模型推理资产，仍可以使用公共资源池部署模型；对于购买推理资产的邀测用户，仅可以使用专属资源池部署模型。
- 对于新购买平台的用户，仅可购买并使用专属资源池。

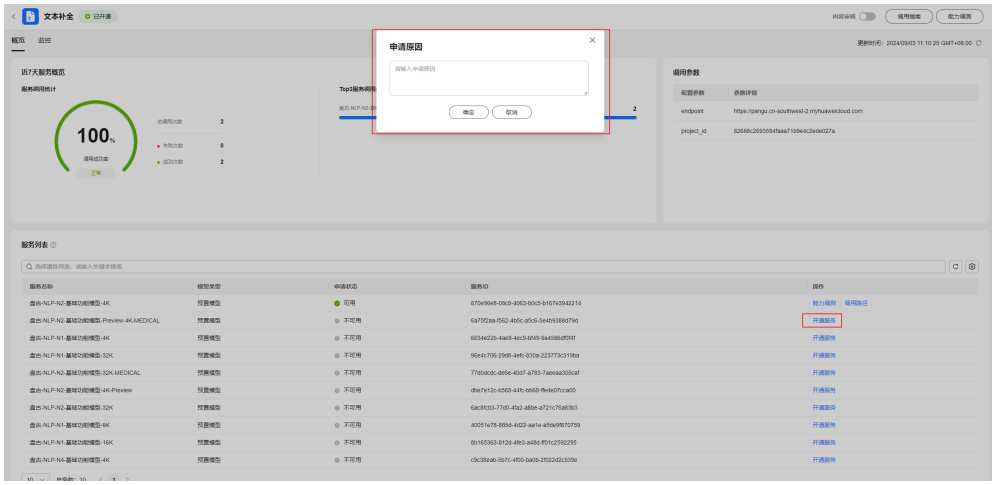
2.3 开通盘古大模型服务

盘古大模型具备文本补全和多轮对话能力，用户在完成盘古大模型套件的订购操作后，需要开通大模型服务，才可以调用模型，实现与模型对话问答。

1. 登录**盘古大模型套件平台**。
2. 在左侧导航栏中选择“服务管理”，在相应服务的操作列单击“查看详情”，可在服务列表中申请需要开通的服务。
   - 文本补全：提供单轮文本能力，常用于文本生成、文本摘要、闭卷问答等任务。
   - 多轮对话：提供多轮文本能力，常用于多轮对话、聊天任务。

图 2-5 服务管理

图 2-6 申请开通服务



3. 您可按照需要选择是否开启内容审核。

   开启内容审核后，可以有效拦截大模型输入输出的有害信息，保障模型调用安全，推荐进行开启。

图 2-7 大模型内容审核

说明

购买内容审核套餐包时，如果使用“文本补全”和“多轮对话”功能，需要选择“文本内容审核”套餐。

2.4 配置盘古访问授权

盘古大模型服务使用对象存储服务（Object Storage Service，简称OBS）进行数据存储，实现安全、高可靠和低成本的存储需求。因此，为了能够正常的存储数据、训练模型，需要用户配置盘古访问OBS的权限。

1. 使用主账号登录盘古大模型套件平台。
2. 在左侧菜单选择“平台管理 > 授权管理”，单击右上角“一键授权”进行授权。

图 2-8 一键授权

2.5 创建子用户并授权使用盘古

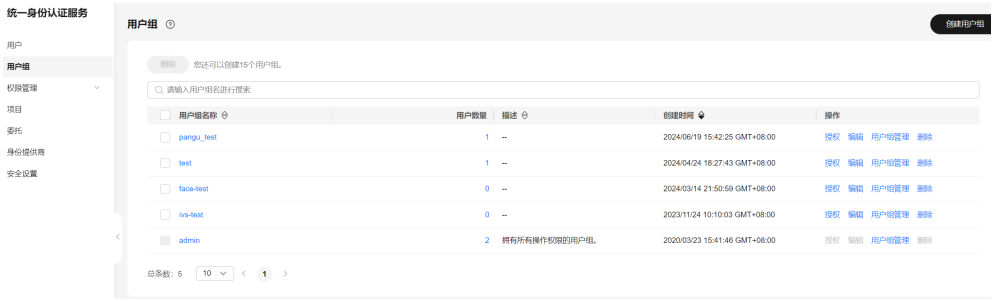
如果您需要对华为云上购买的盘古资源，为企业中的员工设置不同的访问权限，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（IAM）并结合盘古大模型套件平台提供的“角色管理”功能实现精细的权限管理。

如果华为云账号已经能满足您的要求，不需要创建独立的IAM用户（子用户）进行权限管理，您可以跳过本章节，不影响您使用盘古的其他功能。

创建用户组

1. 使用主账号登录**IAM服务控制台**。
2. 左侧导航窗格中，选择“用户组”页签，单击右上方的“创建用户组”。

图 2-9 创建用户组



3. 在“创建用户组”界面，输入“用户组名称”，创建用户组。
4. 返回用户组列表，单击列表中的“授权”。

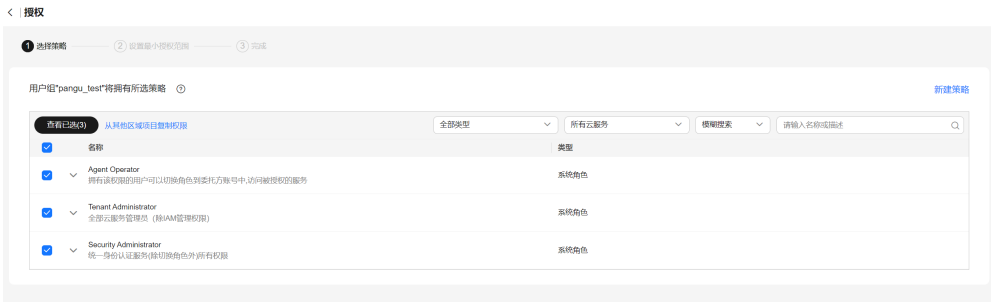
图 2-10 用户组授权

5. 参考**表2-1**，为用户组设置权限。

表 2-1 授权项

授权项	说明
Agent Operator	拥有该权限的用户可以切换角色到委托方账号中，访问被授权的服务。
Tenant Administrator	全部云服务管理员（除IAM管理权限）。
Security Administrator	统一身份认证服务(除切换角色外)所有权限。

图 2-11 添加用户组权限



6. 设置最小授权范围。

   根据授权项策略，系统会自动推荐授权范围方案。例如，可以选择“所有资源”，即用户组内的IAM用户可以基于设置的授权项限使用账号中所有的企业项目、区域项目、全局服务资源。也可以选择“指定区域项目资源”，如指定“西南-贵阳一”区域，即用户组内的IAM用户仅可使用该区域项目中的资源。

图 2-12 设置最小授权范围

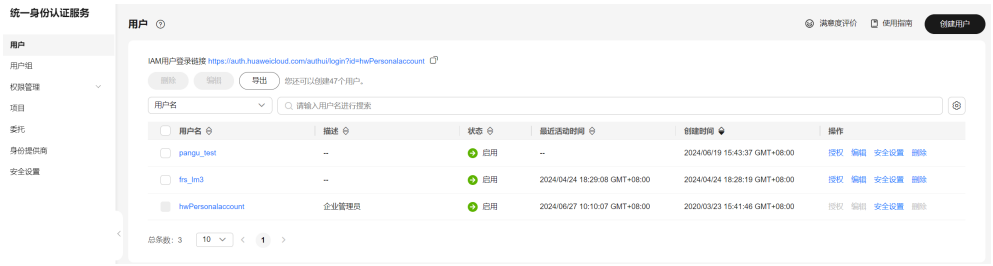
7. 完成用户组授权。

图 2-13 完成授权

创建 IAM 用户，并加入用户组

1. 使用主账号登录IAM服务控制台。
2. 左侧导航窗格中，选择“用户”页签，单击右上方的“创建用户”。

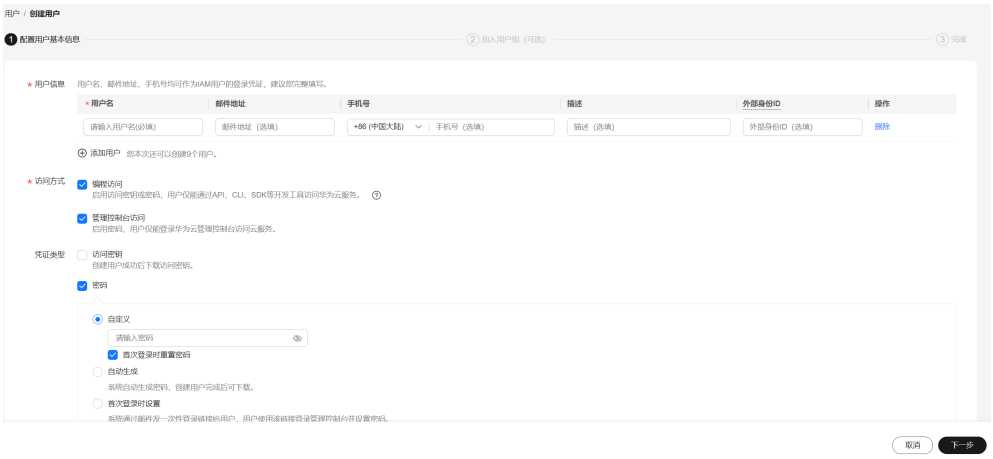
图 2-14 创建用户



3. 配置用户基本信息。

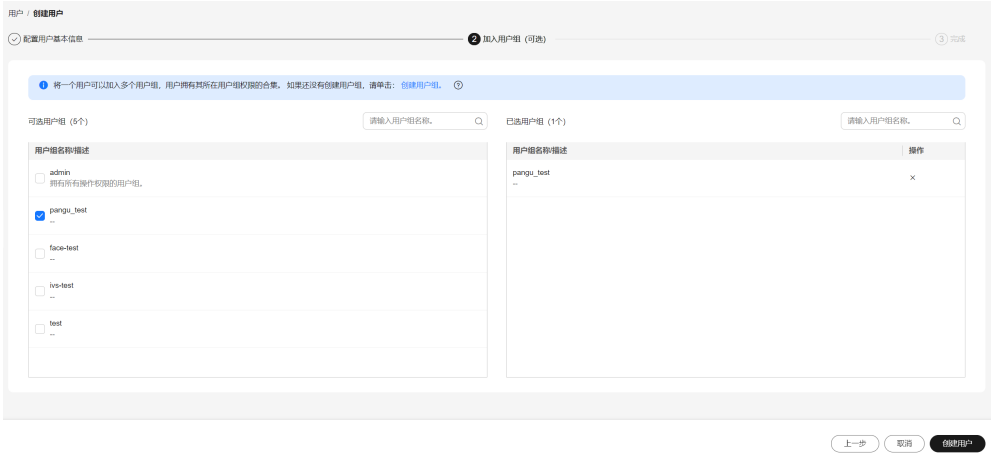
   配置用户信息时，需要勾选“编程访问”，如果未勾选此项，会导致IAM用户无法使用盘古服务API、SDK。

图 2-15 配置用户基本信息



4. 单击“下一步”，将用户添加至创建用户组步骤创建的用户组中，完成IAM用户的创建。

图 2-16 加入用户组



设置用户角色

主账号登录盘古大模型套件平台页面，在页面左侧导航栏“平台管理 > 权限管理”功能中为不同子用户设置角色。

- 主账号（最终租户）默认拥有系统管理员权限，支持给子账号分配角色。
- 角色包括：系统管理员、运营用户、模型开发用户、推理服务调用用户、提示词工程用户、应用开发用户、运维用户。

图 2-17 设置用户角色

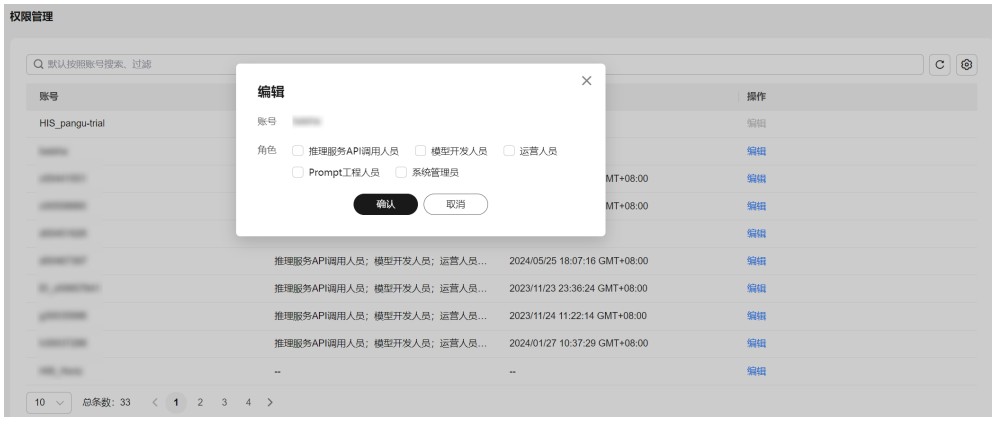


表 2-2 角色和功能关系

功能	系统管理员	运营人员	模型开发人员	推理服务API调用人员	Prompt工程人员
总览	√	√	√	√	√
服务管理	√	-	√	√	√
能力调测	√	-	√	√	√
数据工程-数据集管理	√	-	√	-	-
数据工程-提示用例管理	√	-	-	-	√
模型开发	√	-	√	-	-
应用开发-提示词工程&提示词管理	√	-	-	-	√
应用开发-应用开发SDK（link）	√	-	√	√	√

功能	系统管理员	运营人员	模型开发人员	推理服务API调用人员	Prompt工程人员
平台管理-资产管理	√	√	-	-	-
平台管理-权限管理	√	-	-	-	-
平台管理-授权管理	√	-	-	-	-

3 体验盘古大模型功能

3.1 体验盘古预置模型能力

登录**盘古大模型套件平台**，在左侧导航栏中单击“能力调测”。

如图所示，能力调测页面提供了文本补全和多轮对话功能，且每种功能都提供了预置的盘古大模型供用户体验。用户可以在页面右侧进行参数设置，然后在输入框中输入问题，模型就会返回对应的答案内容，具体参数信息如下表。

图 3-1 体验预置模型功能

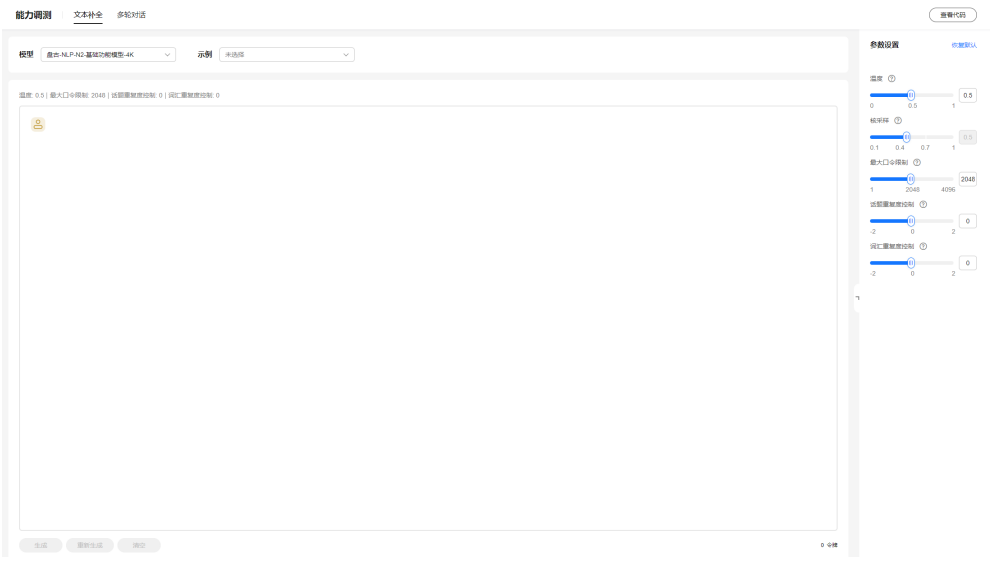


表 3-1 能力调测参数信息表

参数名称	描述
温度	控制语言模型输出的随机性与创造性。温度设置越低，输出更可预测；温度设置越高，输出种类更多，更不可预测。
核采样	控制生成文本多样性和质量。

参数名称	描述
最大口令限制	用于控制聊天回复的长度和质量。一般来说，设置较大的参数值可以生成较长和较完整的回复，但也可能增加生成无关或重复内容的风险。较小的参数值可以生成较短和较简洁的回复，但也可能导致生成不完整或不连贯的内容，请避免该值小于10，否则可能生成空值或极差的效果。因此，需要根据不同的场景和需求来选择合适的参数值。
话题重复度控制	用于调整模型对新令牌（Token）的处理方式。即如果一个Token已经在之前的文本出现过，那么模型在生成这个Token时会受到一定的惩罚。当值为正数时，模型会更倾向于生成新的Token，即更倾向于谈论新的话题。
词汇重复度控制	用于调整模型对频繁出现的Token的处理方式。即如果一个Token在训练集中出现的频率较高，那么模型在生成这个Token时会受到一定的惩罚。当的值为正数时，模型会更倾向于生成出现频率较低的Token，即模型会更倾向于使用不常见的词汇。
历史对话保留轮数	选择要包含在每个新API请求中的过去消息数。这有助于为新用户查询提供模型上下文。参数设置为10，表示包括5个用户查询和5个系统响应。该参数只涉及多轮对话功能。

- 体验预置模型文本补全能力
  a. 进入“文本补全”页签，选择模型与示例，参数设置为默认参数，在输入框输入问题，单击“生成”，模型将基于问题进行回答。

  图 3-2 体验预置模型文本补全能力

  b. 修改参数以查看模型效果，示例如下：
     i. **将“核采样”参数调小，如改为0.1**，保持其他参数不变，单击“重新生成”，再单击“重新生成”，可以看到模型前后两次回复内容的多样性降低。

图 3-3 “核采样”参数调小后生成结果 1

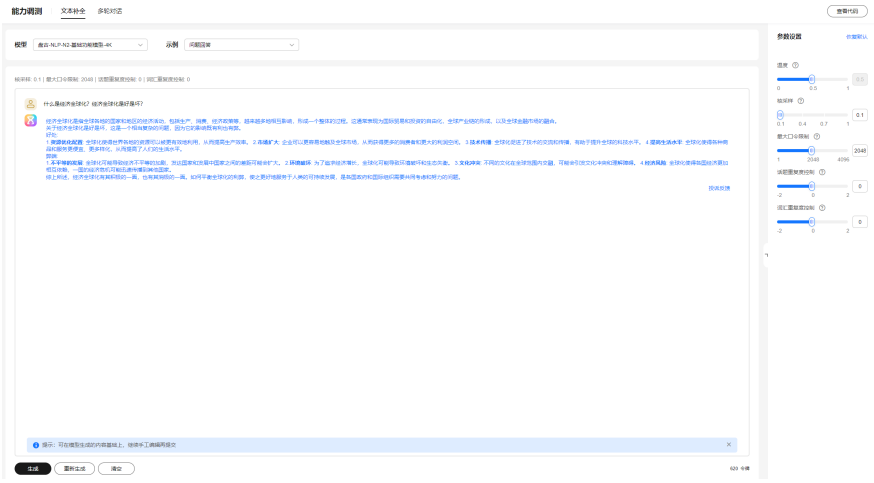
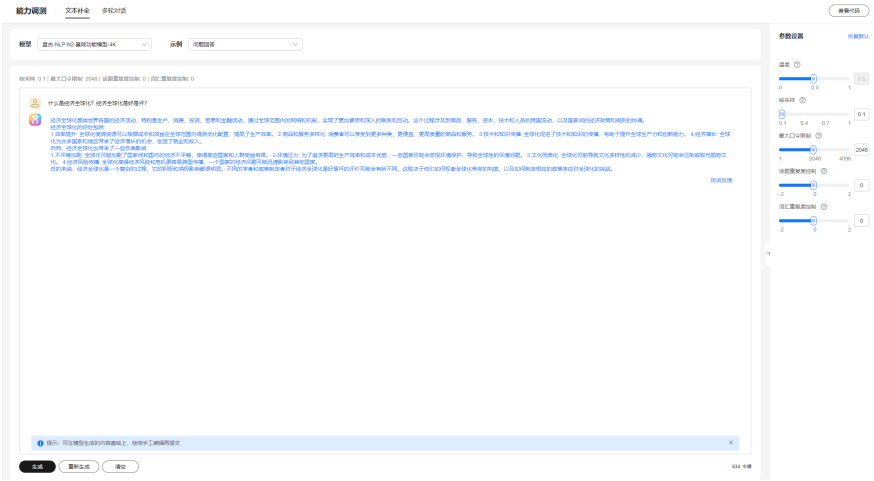


图 3-4 “核采样”参数调小后生成结果 2



ii. **将“核采样”参数调大，如改为1**，保持其他参数不变，单击“重新生成”，再单击“重新生成”，可以看到模型前后两次回复内容的多样性提高。

图 3-5 “核采样”参数调大后生成结果 1

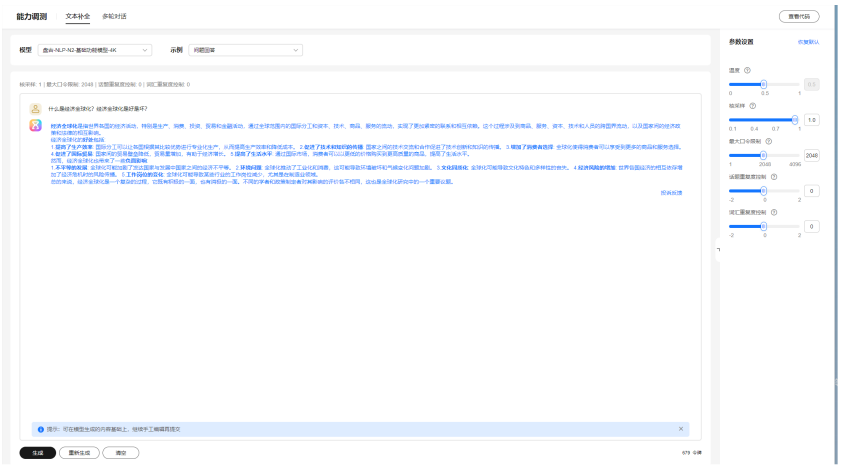
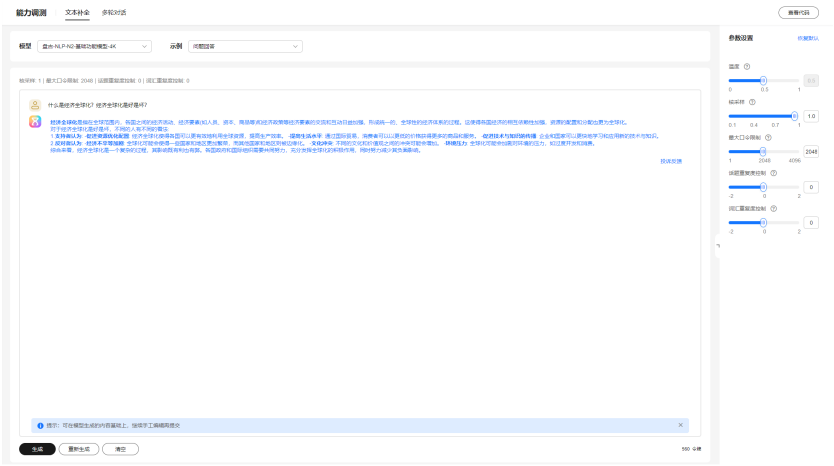


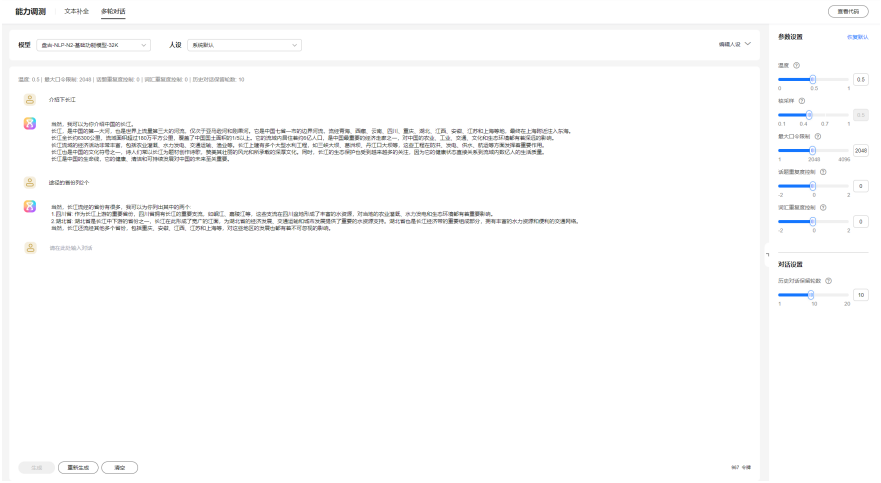
图 3-6 “核采样”参数调大后生成结果 2



- 体验预置模型的多轮对话能力

  a. 进入“多轮对话”页签，选择模型与人设，参数设置为默认参数，在输入框输入问题，单击“生成”，模型将基于问题进行回答。

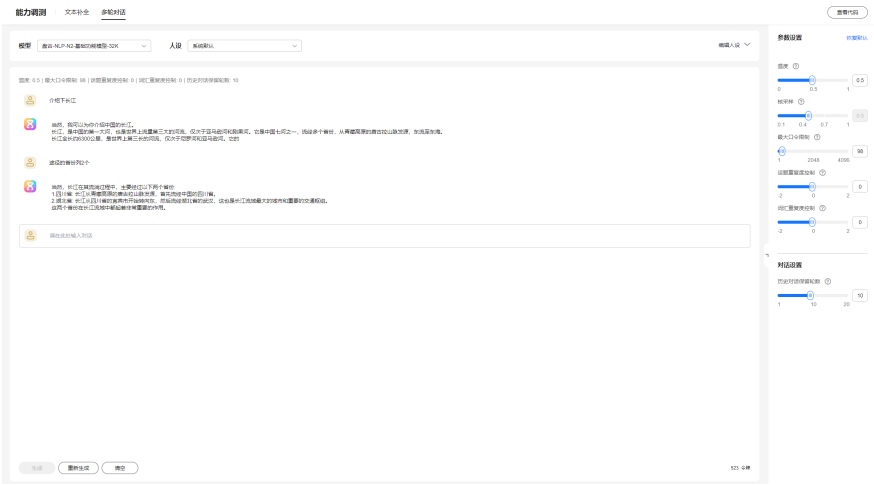
  图 3-7 体验预置模型多轮对话能力

  b. 修改参数以查看模型效果，示例如下：

  将“最大口令限制”参数调小，如改为98，保持其他参数不变，单击“重新生成”，可以看到模型回复内容长度减小。

图 3-8 修改“最大口令限制”参数



3.2 体验盘古驱动的应用百宝箱

应用百宝箱是盘古大模型为用户提供的便捷AI应用集，用户可在其中使用盘古大模型预置的场景应用和外部应用，轻松体验大模型开箱即用的强大能力。

1. 登录**盘古大模型套件平台**，在左侧导航栏中选择“应用百宝箱”，进入“应用百宝箱”页面。
2. 在“应用市场”页签中，选择场景应用，立即体验应用能力。

图 3-9 应用市场

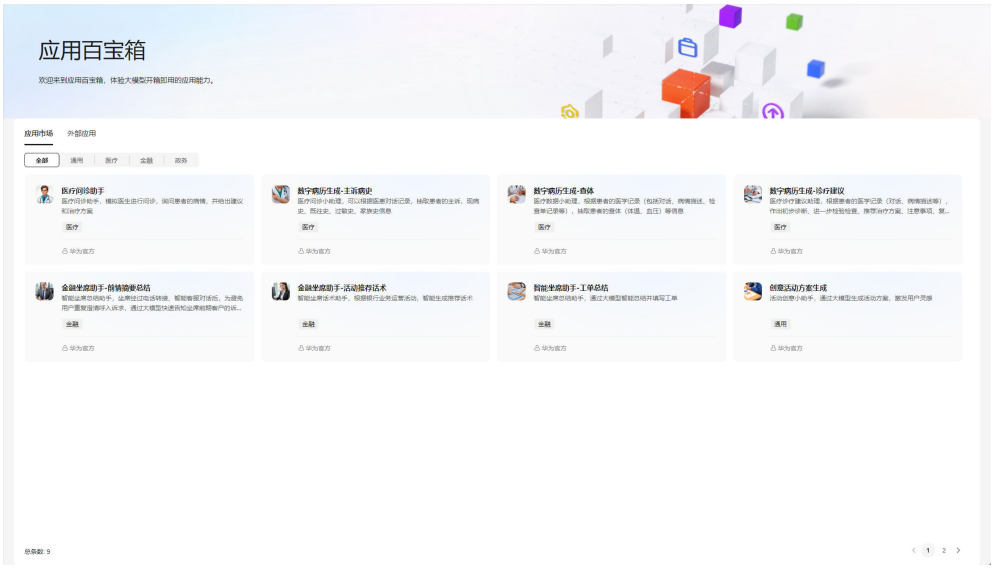
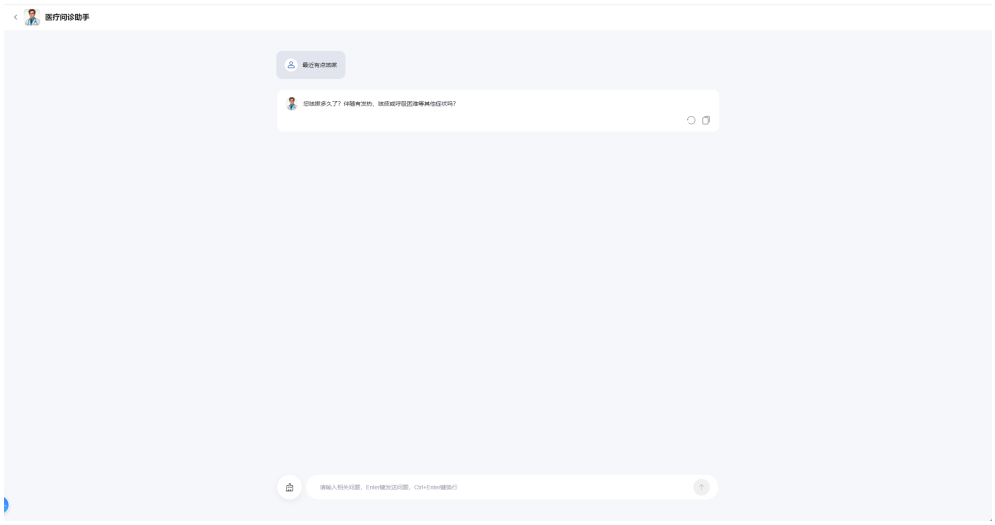


图 3-10 应用试用



3. 在“外部应用”页签中，选择外部应用，单击“继续前往”，页面将跳转至外部应用页面供用户体验。

图 3-11 外部应用

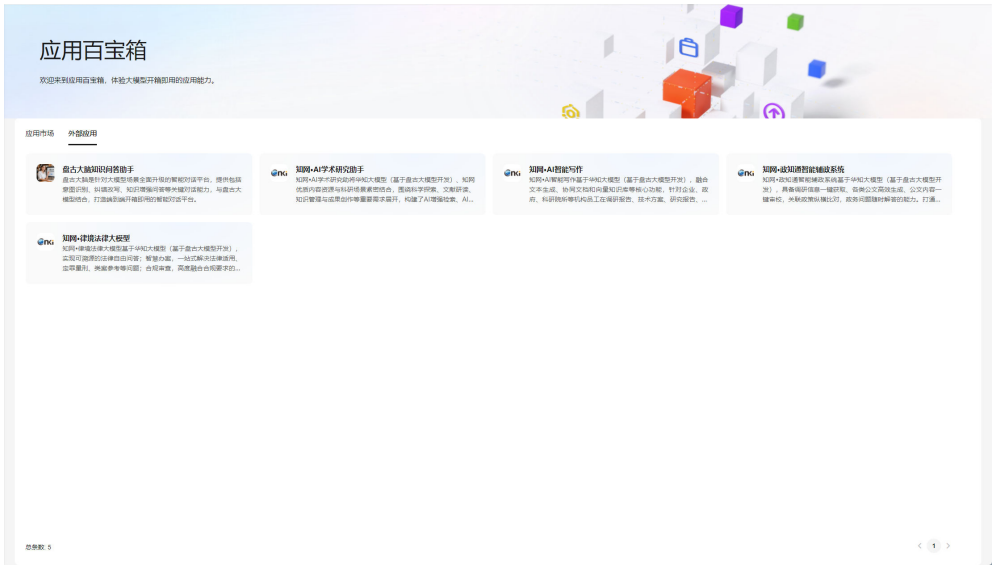
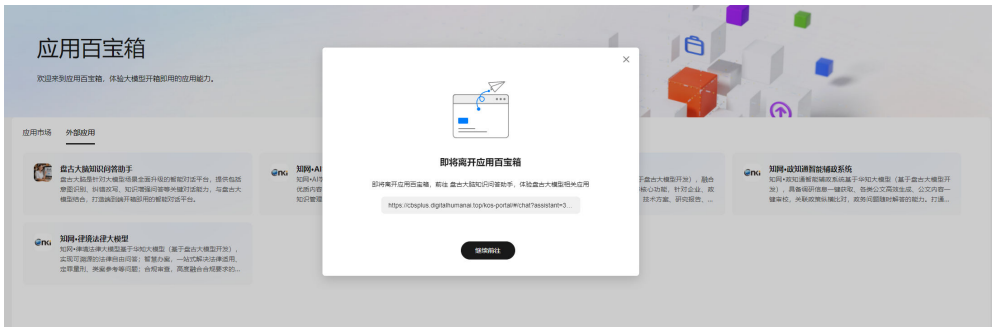


图 3-12 外部应用试用



4 准备盘古大模型训练数据集

4.1 训练数据集创建流程

数据是大模型训练的基础，提供了模型学习所需的知识和信息。大模型通过对大量数据的学习，能够理解并抽象出其中的复杂模式，从而进行精准的预测和决策。在训练过程中，数据的质量和多样性至关重要。高质量的数据能够提升模型对任务的理解，而多样化的数据则帮助模型更好地应对各种情况。因此，数据的收集和处理是大模型训练中的关键环节。

盘古大模型套件平台通过提供数据获取、清洗、配比与管理等功能，确保构建高质量的训练数据。

4.2 模型训练所需数据量与数据格式要求

盘古大模型套件平台支持NLP大模型、多模态大模型的训练。不同模型训练所需的数据量和数据格式有所差异，请基于数据要求提前准备训练数据。

数据量要求

- **自监督训练**

  在单次训练任务中，一个自监督训练数据集内，上传的数据文件数量不得超过**1000个**，单文件大小不得超过**1GB**，所有文件的总大小不得超过**200GB**。

  表 4-1 自监督训练数据大小说明

模型规格	最小数据量（数据条数）	推荐数据量	单条数据token长度限制
N4-4K版本	1万条/每场景	4GB（等价10亿Tokens）	4096

- **有监督训练**

  在单次训练任务中，一个有监督数据集内，上传的数据文件数量不得超过**100个**，单文件大小不得超过**1GB**，所有文件的总大小不得超过**1GB**。

表 4-2 有监督微调数据大小说明

模型规格	最小数据量（数据条数）	单场景推荐训练数据量	单条数据token长度限制	训练集：验证集推荐比例
N1-4K版本	1000条/每场景	≥ 1万条/每场景	4096	10：1
N1-32K版本	1000条/每场景	≥ 1万条/每场景	32768	10：1
N2-4K版本	1000条/每场景	≥ 1万条/每场景	4096	10：1
N2-32K版本	1000条/每场景	≥ 1万条/每场景	32768	10：1
N4-4K版本	1000条/每场景	≥ 1万条/每场景	4096	10：1

训练数据需要依据不同任务场景进行构造。例如，当训练阅读理解任务时，需要选择一些包含大量阅读材料的数据进行训练。当训练广告文案生成任务时，训练数据则需要包含一定量的广告文案数据。

- **模型评估**

  一个评估数据集内，上传的数据文件数量不得超过**100个**，单文件大小不得超过**1GB**，所有文件的总大小不得超过**1GB**。

- **多模态训练**
  - 预训练数据

    tar包存储原始的图片，单个tar包的大小不得超过**500MB**，图片描述JSONL文件只需一份。
  - 指令微调数据

    tar包存储原始的图片，单个tar包的大小不得超过**500MB**，图片描述JSONL文件只需一份。
  - 图片数据仅支持jpg格式，图片大小不得低于**5kb**，图片最小边不得低于**200px**，长边：短边比例不得大于**3：1**。

    表 4-3 多模态数据大小说明

训练类型	最小数据量（图文对数据）	推荐数据量（图文对数据）
预训练	1000万对	5000万对起
指令微调	20~30万对	100万对

数据格式要求

盘古大模型服务支持如下数据，格式要求请参见表4-4。

表 4-4 盘古数据文件格式要求

数据类型	支持格式	数据样例	是否支持拆分（划分训练集/验证集）
自监督训练数据	TXT、JSONL、PDF、WORD、HTML	编码格式为UTF-8。 #TXT格式，一行对应1条JSON #PDF、WORD、HTML只需上传对应的文档，文档内容为文本 #JSONL {"text":"《活着》，是中国著名作家余华所写的一部长篇小说。《活着》讲述了一个普通农民徐福贵的人生历程。他的人生充满了苦难和挫折，但他在面对这些困难时，始终保持着坚强和乐观的态度。"}	否
有监督微调数据	单轮：CSV、JSONL 多轮：JSONL	编码格式为UTF-8。 **#单轮问答示例** #CSV 第一列对应context 第二列对应target #content、target分别表示问题、答案 #JSONL {"context":"非深户在职人员长期在异地居住的是否可以办理异地就医备案手续","target":"可以。本市用人单位长期派驻异地（国内市外）工作的在职参保人员，可以按照常驻异地工作人员申请办理备案。"} 详细有监督数据格式性参见**表 4-5**。	是
评测数据	CSV、JSONL	同有监督单轮不带system prompt数据。	否

表 4-5 有监督数据格式

数据类型	格式说明
有监督单轮，JSONL格式	编码格式为UTF-8。 每一行表示一段文本，形式为{"context":"context内容","target":"target内容"} content、target分别表示问题、答案 #示例 {"context":"非深户在职人员长期在异地居住的是否可以办理异地就医备案手续","target":"可以。本市用人单位长期派驻异地（国内市外）工作的在职参保人员，可以按照常驻异地工作人员申请办理备案。"}
有监督单轮，CSV格式	编码格式为UTF-8。 每一行代表一个问答对，第一列对应context 第二列对应target，确保每个问题和答案的数据都以逗号分隔，每行的数据完整且格式正确
有监督单轮，带人设，JSONL格式	编码格式为UTF-8。 每一行表示一段文本，system不能为空，形式为{"system":"system内容","context":"context内容","target":"target内容"} system、content、target分别表示人设、问题、答案 #示例 {"system":"你是一个知识问答助手","context":"诗仙指的是哪位诗人？","target": "唐代诗人李白为诗仙。"}

数据类型	格式说明
有监督多轮，JSONL格式	编码格式为UTF-8。 每一行表示一段文本，为数组格式，至少一组问答对，形式为[{"context":"context内容1","target":"target内容1"},{"context":"context内容2","target":"target内容2"}] content、target分别表示问题、答案 #示例 [{"context":"诗仙指的是哪位诗人","target":"唐代诗人李白为诗仙"},{"context":"他都有哪些代表作？","target":"李白的代表作有《望庐山瀑布》、《行路难》、《蜀道难》等"}]
有监督多轮，带人设，JSONL格式	编码格式为UTF-8。 每一行表示一段文本，为数组格式，至少一组问答对，system不能为空，形式为[{"system":"system内容"},{"context":"context内容1","target":"target内容1"},{"context":"context内容2","target":"target内容2"}] system、content、target分别表示人设、问题、答案 #示例 [{"system":"你是一个知识问答助手"},{"context":"诗仙指的是哪位诗人","target":"唐代诗人李白为诗仙"},{"context":"他都有哪些代表作？","target":"李白的代表作有《望庐山瀑布》、《行路难》、《蜀道难》等"}]

4.3 创建一个新的数据集

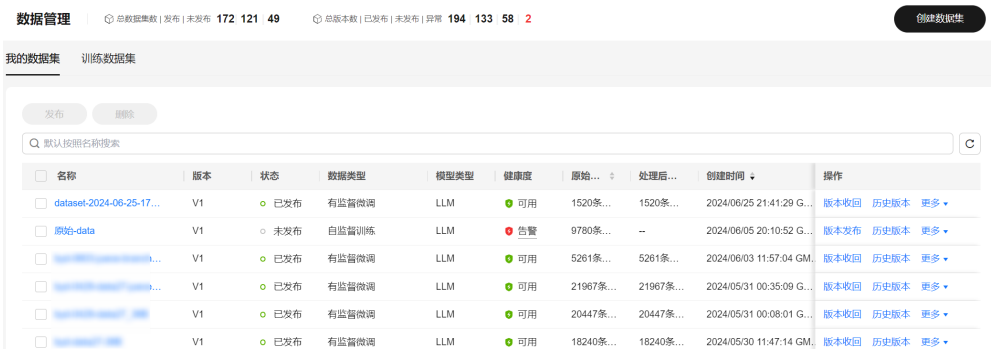
数据集是指用于训练模型或评估的一组相关数据样本。存储在OBS中的数据可以通过数据集的形式放置在到盘古平台中，便于管理。

在创建数据集之前，请先将数据上传至OBS平台。

上传数据至 OBS

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“数据工程 > 数据管理”，单击界面右上角“创建数据集”。

图 4-1 数据管理



3. 在创建数据集弹框中选择“创建一个新的数据集”，单击“创建”。

图 4-2 创建数据集

4. 在创建数据集页面，单击“前往OBS”，进入OBS服务页面。

图 4-3 前往 OBS

5. 在OBS控制台页面，单击界面右上角“创建桶”。

图 4-4 OBS 页面

6. 创建OBS桶时，**桶区域需要与盘古大模型区域保持一致**。其余配置参数可以使用默认值，详细OBS桶参数说明请参见**OBS用户指南**。

图 4-5 创建 OBS 桶

7. 参数填选完成后，单击“立即创建”。创建好的OBS桶将显示在桶列表中。

图 4-6 OBS 页面

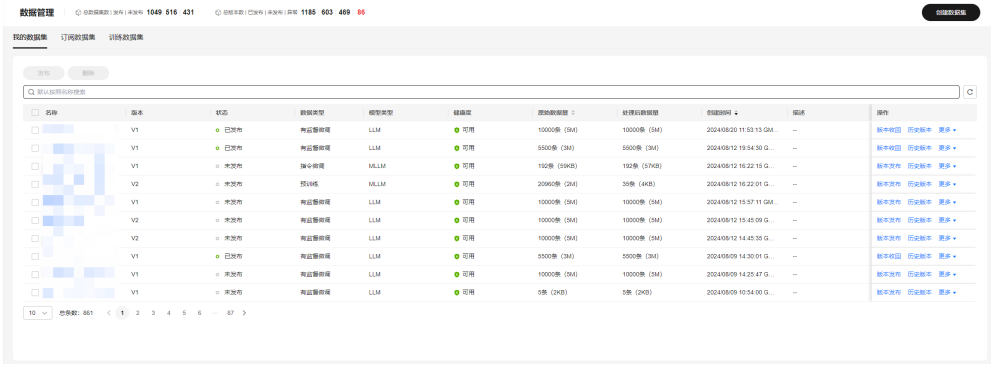
8. 在OBS中创建文件目录，并在目录中上传数据文件。

创建一个新的数据集

创建数据集前，需按要求将数据文件上传至OBS。

1. 登录盘古大模型套件平台，左侧导航栏中选择“数据工程 > 数据管理”，单击界面右上角“创建数据集”。

图 4-7 数据管理



2. 在创建数据集弹出框中选择“创建一个新的数据集”，单击“创建”。

图 4-8 创建数据集

3. 在新建数据集页面，依据需要进行的训练任务，选择导入数据，填写基本信息。
   - 导入数据

     选择模型类型、训练类型、数据类型、导入格式以及数据来源。

表 4-6 数据集路径说明

数据集训练类型	数据集所在OBS路径
自监督训练数据集	创建数据集时，需要指定数据文件所在的文件夹。
有监督微调数据集	创建数据集时，可以指定数据文件或者数据文件所在的文件夹。

数据集训练类型	数据集所在OBS路径
评测数据集	创建数据集时，可以指定数据文件或者数据文件所在的文件夹。

- 基本信息
  填写数据集名称与描述，选择行业、语言和数据标签。

  图 4-9 填写基本信息

4. 参数填选完成后，单击“立即创建”。
   创建好的数据集将显示在数据集列表中。

4.4 检测数据集质量

数据集创建成功后，平台将对数据集中的数据进行质量校验，并给出健康度评分、合规度评分与数据长度分布。

检测数据集质量

1. 在“数据工程 > 数据管理”页面，选择“我的数据集”或者“训练数据集”页签。
2. 单击数据集名称，进入数据集详情页，查看详细的数据质量。
   其中，数据长度按照token长度划分为2K以下、2K-4K、4K-8K等多个区间，用户可以参考**模型训练所需数据量与数据格式要求**，调整训练数据。

图 4-10 校验数据集质量

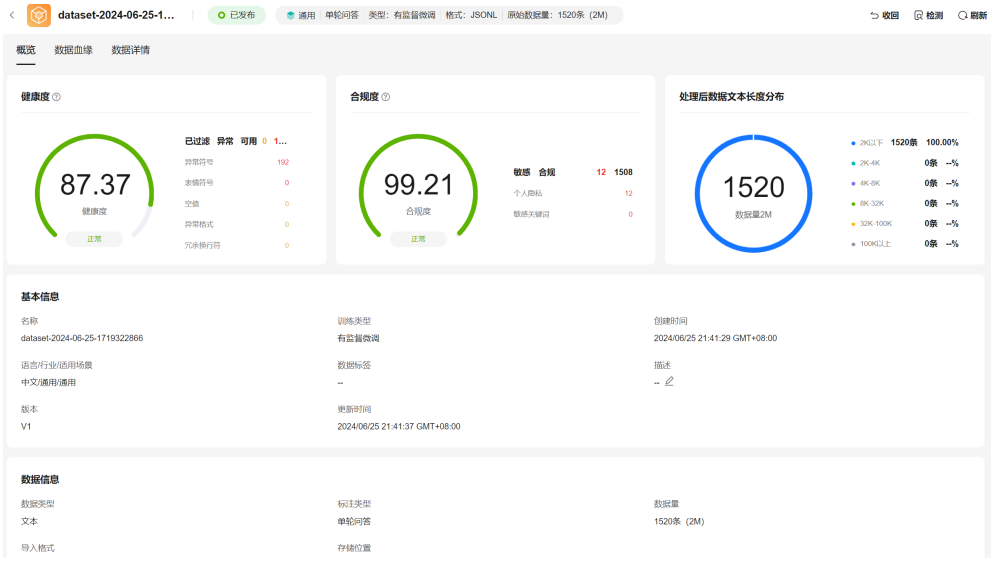


表 4-7 健康度校验规则说明

校验项	说明
异常符号校验	数据中不能存在异常字符，异常字符示例如下。 \u0000 \u0001 \u0002
表情符号校验	校验数据中是否存在表情符号，如□ □ □等，常见表情符清单请参见**Full Emoji List**。
空值校验	校验数据中是否存在空字符串。
异常格式校验	检查数据是否满足**数据格式要求**。
冗余换行符校验	检查数据中是否存在连续两个及以上的换行符。

表 4-8 健康度状态说明

正常数据量	健康度颜色	是否可用于训练
> = 80%	绿色	可用
> = 40%	黄色	预警，需要优化数据
< 40%	红色	告警，需要优化数据

说明

- 正常数据量：数据集中，有效数据占总体数据的比例。
- 预警：数据集中，有效数据占总体数据的比例在40%-80%之间，表示数据质量较差，提示需要进行优化。
- 告警：数据集中，有效数据占总体数据的比例低于40%，表示数据质量极差，提示需要进行优化。

表 4-9 合规度校验规则说明

校验项	说明
个人隐私	校验数据中是否存在个人隐私信息，例如，身份证号、手机号、固定电话、Email地址、护照号、车牌号、军官证、车架号、GPS地址、IP地址、MAC地址和IMEI码等。
敏感关键词	校验数据中是否存在敏感关键字，如涉政信息。

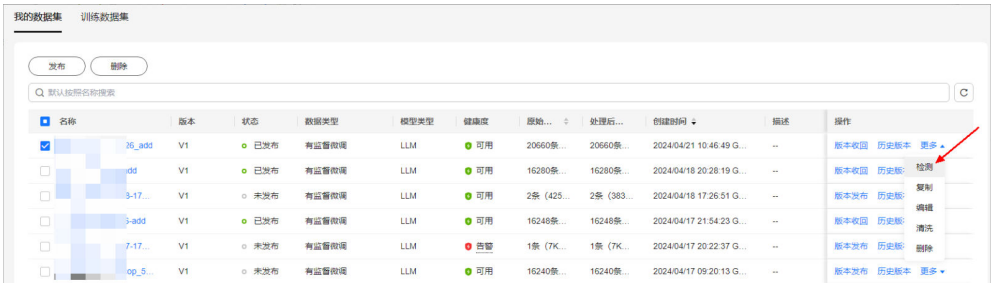
表 4-10 合规度状态说明

合规数据量	合规度颜色	是否可用于训练
> = 80%	绿色	可用
> = 40%	黄色	预警，需要优化数据
< 40%	红色	告警，需要优化数据

3. （可选）当“我的数据集”的OBS数据发生变更时，可以单击右上角“检测”按钮重新校验数据集，也可以在“我的数据集”页签中，单击操作栏中的“更多 > 检测”，重新校验数据集。历史存量未校验过的数据集也可以进行重新校验。

图 4-11 重新校验数据集质量 1

图 4-12 重新校验数据集质量 2



4.5 清洗数据集（可选）

4.5.1 清洗算子功能介绍

数据清洗是提高数据质量的重要环节，包括去除异常的字符、去除表情符号和去除个人敏感内容等，经过清洗的数据可以提升训练阶段的稳定性。

平台支持通过以下清洗能力：

表 4-11 清洗算子说明

算子类型	功能	说明
数据转换	全角转半角	将文本中的所有全角字符转换成半角字符。
	中文繁简体互转	简体转换成繁体或者繁体转换成简体。
	去除不可见字符	移除文本中不可见字符，如U+0000-U+001F。
	去除表情符	移除文本中表情符，如□□。
	去除网页标签	移除文本中网页标签符号。
	去除特殊字符	移除文本中特殊符号，如● ■ ◆ ◪ ► ® ©。
	统一空格	将文本中不同的unicode空格比如U+00A0、U+200A，统一替换成通用空格。
	去除乱码	移除去除乱码和无意义的字符。
	html转义符反转	将文本中html转义符进行反转，如>|>替换为>|>。
	冗余说明去除	移除文本中冗余的说明。
	去除冗余尾部信息	移除文本尾部冗余的信息。
	冗余段落过滤	移除文本中的冗余段落。
	字符归一化	将文本中不同的字符风格统一显示，如①,⑴,㈠,1.,⓵,❶,❶,①统一显示为1.,1.,1.,1.,1.,1.,1.,1.。
数据过滤	符号比率过滤	如果文本中符号比例大于指定阈值时，则过滤文本，符号包括特殊符号、标点符号、大中小。
	文本长度过滤	过滤文本长度超出指定范围的内容。
	乱码文本	过滤乱码字符占比超过阈值的文本。

算子类型	功能	说明
	汉字比率过滤	基于文档中汉字占比过滤数据。
	目录\封面过滤	移除文本的目录和封面。
	图注标注过滤	移除文本中的图标和标注信息。
	参考文献过滤	移除文本中参考文献的信息。
数据去重	去重	移除文本中重复内容。
数据安全	数据脱敏	识别并对文本中电话号码、邮箱、身份证等信息进行脱敏。
	敏感词过滤	识别并过滤文本中包含的涉黄、涉暴、涉政等敏感词。
通用清洗	正则替换	基于给定的正则表达式，进行文本替换。
	正则过滤	基于给定的正则表达式，进行文本过滤。
数据读取	单栏文字版PDF文档读取	解析PDF文档。数据集文件类型为PDF时显示。
	word文本读取	解析WORD文档，支持doc和docx格式。
	html格式读取	解析HTML文件。

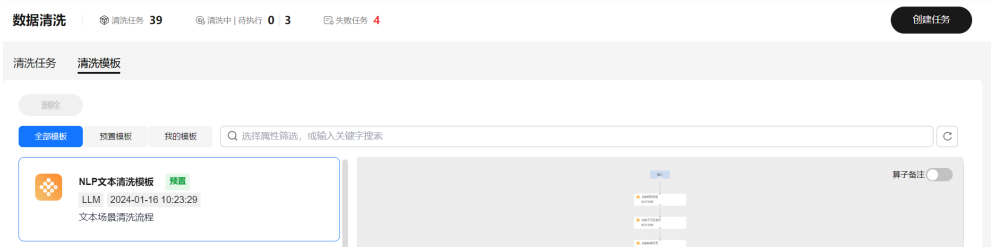
4.5.2 获取数据清洗模板

在清洗数据时，用户可以通过组合不同的数据清洗算子来实现数据清洗功能。此外，平台还提供多种数据清洗模板，用户可以直接套用这些模板进行数据清洗。

数据清洗模板获取方式如下：

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“数据工程 > 数据清洗”，进入“清洗模板”页面，在该页面查看预置的数据清洗模板。

图 4-13 获取数据清洗模板



4.5.3 创建数据集清洗任务

数据集创建完成后，可以使用数据清洗功能，对异常数据进行清理，或进行数据转换、过滤和去重等操作。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“数据工程 > 数据清洗”，单击界面右上角“创建任务”。

图 4-14 数据清洗

3. 依据需要清洗的数据类型，选择对应的数据集和数据集版本，输出路径，设置名称、描述等信息为可选项。

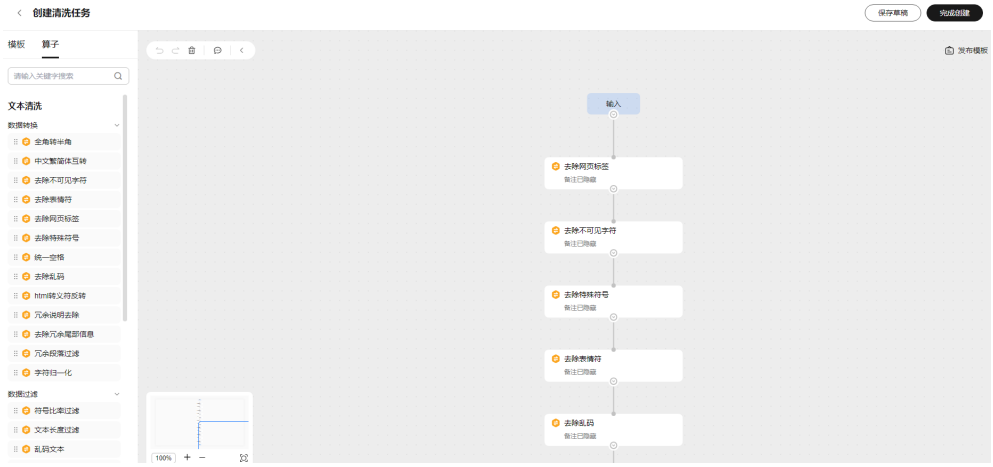
   输出路径默认为系统生成，您也可以自定义输出路径，当前支持覆盖和追加两种方式。

   - 覆盖：清洗后数据覆盖和替换原有数据集内容。
   - 追加：清洗后数据增加到原有数据集路径下。

4. 任务信息填写完成后，单击“下一步”，搭建数据清洗流程。

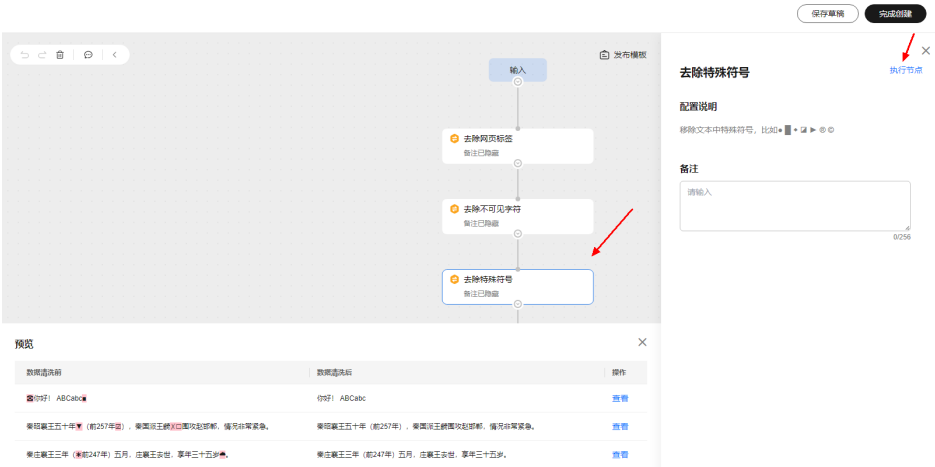
   可以使用预置的清洗模板完成对数据集的清洗，也可以基于算子搭建清洗流程。

图 4-15 搭建数据清洗流程



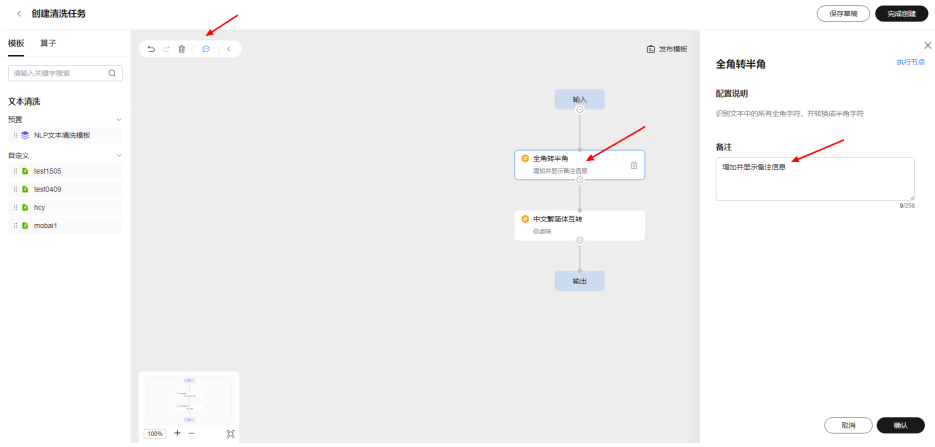
   - 将算子拖拽至“输入”、“输出”之间，即可完成清洗流程的搭建，搭建过程中可以通过“执行节点”功能查看算子对数据的清洗效果。算子功能的详细介绍请参见**清洗算子功能介绍**。

图 4-16 执行节点



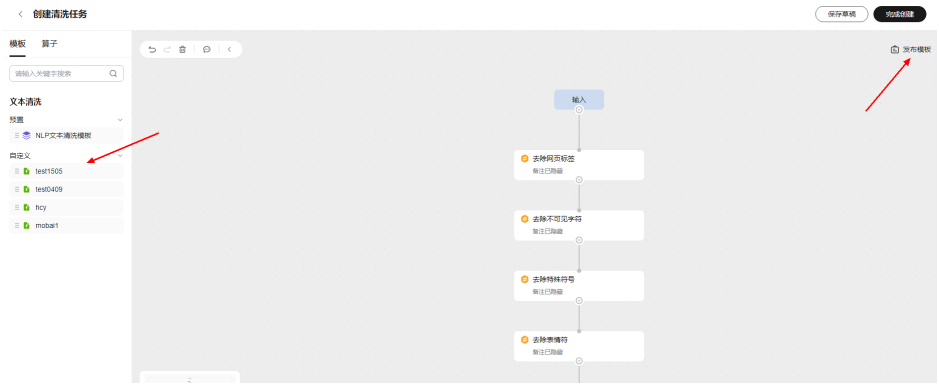
- 用户配置算子后推荐增加、显示备注信息，用于团队其他成员快速了解算子编排。

图 4-17 增加并显示备注信息



- 对于搭建满意的清洗流程，可以“发布模板”，后续重复使用。发布后的模板，可以在“模板”页签查看，也可以返回数据清洗列表，在“清洗模板 > 我的模板”中查看。

图 4-18 发布模板



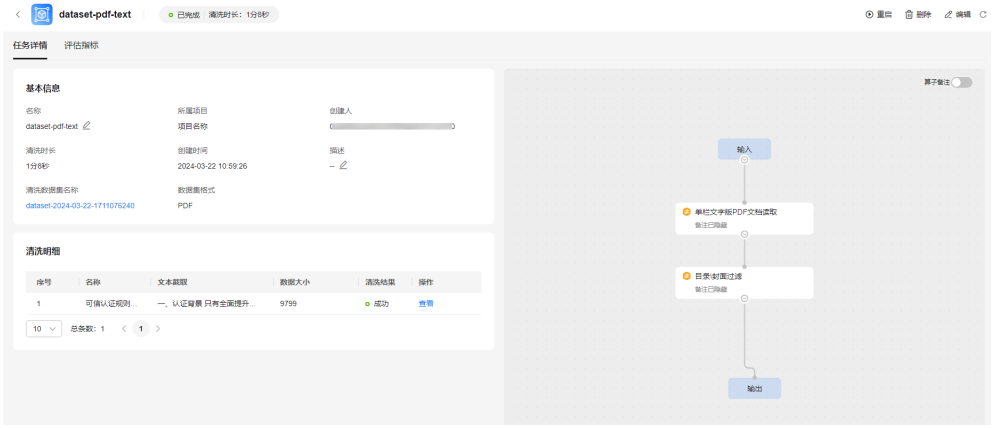
5. 清洗流程搭建完成后，单击界面右上角“完成创建”。

6. 单击清洗任务列表操作栏中的“启动”，启动清洗任务。

   清洗任务完成后，可以单击“任务名称”，在任务详情页面，查看任务详情、评估指标、清洗明细及清洗流程图。

图 4-19 启动清洗任务

图 4-20 查看清洗任务详情



4.6 发布数据集

刚创建的数据集在未发布状态下，无法应用于模型训练，数据集创建、清洗完成后需要执行“发布”操作才可以将该数据集用于后续的任务中。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“数据工程 > 数据管理”，在“我的数据集”页签找到未发布的数据集，单击操作列“版本发布”执行发布数据集操作。

   对不再使用的数据集可以单击“版本收回”撤销当前版本。

图 4-21 发布数据集

4.7 创建一个训练数据集

训练数据集是用于模型训练的实际数据集。通常，通过**创建一个新的数据集**步骤，可以生成包含某个特定场景数据的数据集。例如，这个数据集可能只包含用于训练摘要提取功能的数据。然而，在实际模型训练中，通常需要结合多种任务类型的数据，而不仅限于单一场景的数据。因此，实际的训练会混合不同类型的数据。例如，为防止模型在训练后出现通用问答能力下降，会混入一定的通用数据。

训练数据集常见业务场景如下：

- 当创建训练数据集时，可以将不同数据集合并成一个训练集，并且可以控制各数据集的数据比例。这对于用户数据集较小的情况非常有用，因为可以通过组合多个数据集来进行训练。
- 当需要对模型进行综合训练时，会组合多种数据集，以提高模型处理不同数据类型的能力。

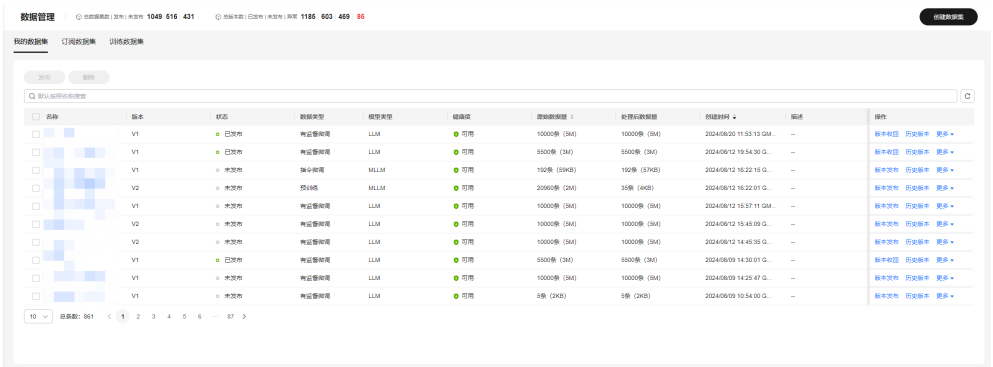
说明

- 在准备自监督训练数据和有监督微调数据时，除行业数据外，建议混入一定比例的通用数据，防止模型在经过训练后出现通用问答能力下降的情况。
- 行业数据：通用数据的比例按业内经验有1：1、1：5。实际训练过程中，行业数据和通用数据和的配比需要根据具体情况进行权衡，需要通过多次训练进行调整，既要考虑模型的通用能力，也要考虑模型在特定领域的性能。

创建一个训练数据集

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“数据工程 > 数据管理”，单击界面右上角“创建数据集”。

图 4-22 数据管理



3. 在创建数据集弹出框中选择“创建一个训练数据集”，单击“创建”。

图 4-23 创建训练数据集

说明

创建训练数据集的常见业务场景包括：

- 当用户的数据集较小时，可以将多个数据集组合起来进行训练。
- 需要进行模型的综合训练时，会组合多样的数据集，以提升模型处理不同类型数据的能力。例如，通过组合数据集，NLP模型在训练后可以同时具备文本生成、情感分析等多种能力。

4. 进入训练数据集页面后，需要进行训练配置、数据配置和基本配置。
   - 训练配置

     选择模型类型、训练类型以及基础模型。
   - 数据配置

     选择训练数据集和配比类型，设置训练数据集配比，详情请参考**数据配比功能介绍**。

     说明

     在训练数据集配比完成后，在单击“创建”或后续修改保存时，会对数据集的有效数据进行统计，确保满足模型训练的要求。

图 4-24 数据配置

- 基本配置
  填写训练数据集名称和描述，选择数据标签。

图 4-25 基本配置

5. 参数填选完成后，单击“立即创建”。

数据配比功能介绍

用户针对业务场景，可以通过数据配比功能，自由组合多个数据集，并控制数据占比。

- 数据集来源：用户自己创建并且已经发布的数据集。
- 数据集组合：选择多个数据集，并且可以指定数据之间的配比和条数，最大支持20个。
  - 配比的作用：支持用户灵活调整数据集的比例。

比例：用户自己创建的数据集，默认1:1:1的方式。例如，3个数据集D1（100GB）、D2（50GB）、D3（200GB），配比按照最大比例去配比，即为D1（50GB）、D2（50GB）、D3（50GB），则3*50=150GB，此时用户可以控制最大的数据量，限制数据量大小，如100GB。

表 4-12 配置比例

配置比例	**数据集大小上限500GB**	**第一阶段**	**第二阶段**	**-**
数据集	原始大小	默认值	手动修改	实际大小
D1	100GB	1	1	100GB
D2	50GB	1	2	50GB
D3	200GB	1	1	200GB
训练数据集PD1	/	15	15	750GB

- 条数：用户指定每个数据集需要提供的条数；如果某个数据集的条数不满足用户需求，则提示用户重新输入，避免用户无感配置失败。

  条数：不提供配比，默认全都选上。

表 4-13 配置条数

配置条数	**数据集大小上限500GB**	**第一阶段**	**第二阶段**	**-**
数据集	原始大小	默认值	手动修改	实际条数
D1	100	100	100	53
D2	50	50	50	27
D3	200	200	100	53
训练数据集PD1	/	/	1250	667

5 训练盘古大模型

5.1 选择模型与训练方法

NLP 大模型

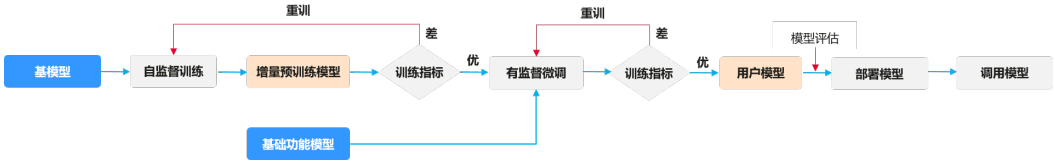
NLP大模型主要用于处理和理解人类语言，能够实现对话问答、文案生成和阅读理解等任务，并具备逻辑推理、代码生成以及插件调用等高阶能力。

NLP大模型提供了基模型和功能模型两种类型：

- **基模型**：已经在大量数据上进行了预训练，学习并理解了各种复杂特征和模式。这些模型可以作为其他任务的基础，例如阅读理解、文本生成和情感分析等。基模型本身不具备对话问答能力。
- **功能模型**：在基模型的基础上进行微调，以适应特定任务。功能模型具备对话问答能力，并经过特定场景的优化，能够更好地处理文案生成、阅读理解和代码生成等任务。

  功能模型无需额外训练即可直接用于客户任务，而基模型则需要经过微调训练才能应用。NLP大模型不仅支持预训练和微调，还可以通过如下训练途径来构建满足客户需求的“用户模型”。

图 5-1 NLP 大模型训练方式与流程



除基模型、功能模型这两种模型划分途径外，NLP大模型还提供了多种系列的模型，不同系列模型在能力上有所差异，可执行的训练操作也有所不同。

表 5-1 不同系列模型对训练的支撑情况

训练任务	N1	N2	N4
预训练	×	×	√

训练任务	N1	N2	N4
微调	√	√	√

不同系列的模型，对文本长度的处理也各有差异，选择合适的模型能够处理特定长度的文本，从而提高模型的整理效果。

表 5-2 NLP 大模型清单

模型类别	模型	token	简介
NLP大模型	盘古-NLP-N1-基础功能模型-32K	部署可选4096、32768	基于NLP-N1-基模型训练的基础功能模型，具备文案生成、多轮对话、实体抽取、翻译、知识问答等大模型通用能力，具有32K上下文能力。
	盘古-NLP-N1-基础功能模型-8K	8192 可外推：16384	基于NLP-N1-基模型训练的基础功能模型，具备文案生成、多轮对话、实体抽取、翻译、知识问答等大模型通用能力，具有8K上下文能力，可外推至16K。
	盘古-NLP-N2-基模型	-	预训练模型，擅长通用任务，擅长文本理解，可以高效进行文案生成与文本解析，高性能、时延低。
	盘古-NLP-N2-基础功能模型-4K	4098	基于NLP-N2-基模型训练的基础功能模型，具备文案生成、多轮对话、实体抽取、翻译、知识问答等大模型通用能力。
	盘古-NLP-N2-基础功能模型-32K	32768	基于NLP-N2-基模型训练的基础功能模型，具备文案生成、多轮对话、实体抽取、翻译、知识问答等大模型通用能力。
	盘古-NLP-N2-应用增强模型-4K	4096	基于NLP-N2-基模型训练的应用增强模型，支持插件调用，支持多种开发套件，可部署集成至业务系统。
	盘古-NLP-N4-基模型	-	预训练模型，擅长逻辑推理，支持工具调用、自然语言生成SQL，可执行复杂任务，质量更高。
	盘古-NLP-N4-基础功能模型-4K	4096	基于NLP-N4-基模型训练的基础功能模型，具备文案生成、多轮对话、实体抽取、翻译、知识问答等大模型通用能力，具有4K上下文能力。
	盘古-NLP-BI专业大模型-4K	4096	基于NLP-N2-基础功能模型运用特定专业代码数据训练后的BI专业大模型，具有4K上下文能力。

模型类别	模型	token	简介
	盘古-NLP-BI专业大模型-32K	32768	基于NLP-N2-基础功能模型运用特定专业代码数据训练后的BI专业大模型，具有32K上下文能力。
	盘古-NLP-N2单场景模型-4K	4096	基于NLP-N2-基模型训练的单场景模型，可支持选择一个场景进行推理，如：搜索RAG方案等，具有4K上下文能力。
	盘古-NLP-N2单场景模型-32K	32768	基于NLP-N2-基模型训练的单场景模型，可支持选择一个场景进行推理，如：搜索RAG方案等，具有32K上下文能力。

NLP大模型训练过程中，一般使用token来描述模型可以处理的文本长度。token（令牌）是指模型处理和生成文本的基本单位。token可以是词或者字符的片段。模型的输入和输出的文本都会被转换成token，然后根据模型的概率分布进行采样或计算。不同系列模型在读取中文和英文内容时，字符长度转换为token长度的转换比如下。以N1为例，盘古模型1token≈0.75个英文单词，1token≈1.5汉字。

表 5-3 token 比

模型规格	token比（token/英文单词）	token比（token/汉字）
N1系列模型	0.75	1.5
N2系列模型（不包含盘古-NLP-N2-基础功能模型-4K-Preview）	0.88	1.24
盘古-NLP-N2-基础功能模型-4K-Preview	0.86	1.69
N3系列模型	0.77	1
N4系列模型	0.75	1.5

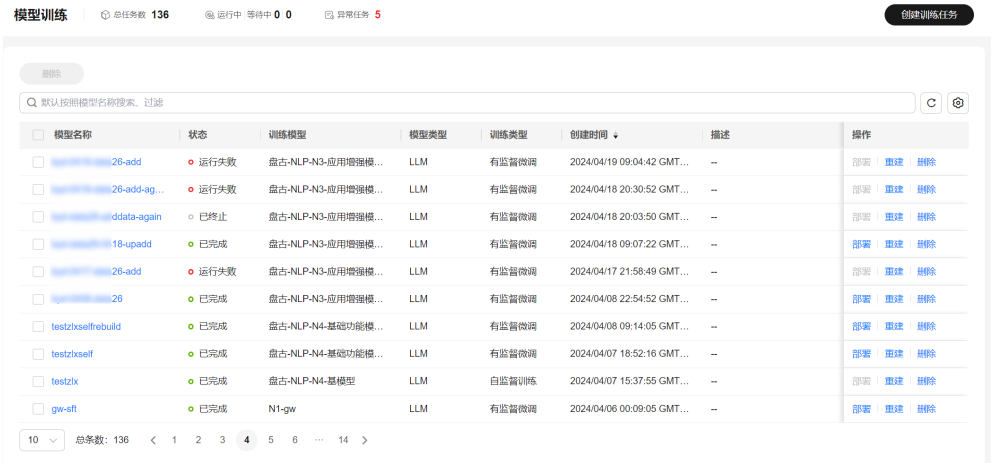
5.2 创建训练任务

5.2.1 创建自监督微调训练任务

创建自监督微调训练任务

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“模型开发 > 模型训练”，单击界面右上角“创建训练任务”。

图 5-2 模型训练列表



3. 在训练配置中，设置模型类型、训练类型、训练模型、训练参数和checkpoints等参数。

   其中，训练配置选择LLM（大语言模型），训练类型选择自监督训练，根据所选模型配置训练参数。

表 5-4 自监督训练参数说明

参数名称	说明
模型类型	选择“LLM”。
训练类型	选择“自监督训练”。
训练模型	选择训练所需要的模型，模型详细介绍请参见**选择模型与训练方法**。
训练参数	指定用于训练模型的超参数。 训练参数说明和调参策略请参见**自监督微调训练参数说明**。
checkpoints	模型训练任务过程中，checkpoints用于保存模型权重和状态的机制，以便故障场景及用户主动终止训练任务后，能够基于中间checkpoints继续训练。

4. 在数据配置中，选择训练模型所需的数据集。

图 5-3 数据配置

5. 完成训练任务基本信息。设置模型的名称、描述以及订阅提醒。

   设置订阅提醒后，模型训练和部署过程产生的事件可以通过手机或邮箱发送给用户。

   图 5-4 基本信息

6. 单击“立即创建”，创建自监督训练任务。

自监督微调训练参数说明

不同模型训练参数默认值存在一定差异，请以前端页面展示的默认值为准。

表 5-5 自监督训练参数说明

训练参数	默认值	范围	说明
数据批量大小	8	>=1	数据集进行分批读取训练，设定每个批次数据的大小。 一般来说，批大小越大，训练速度越快，但会占用更多的内存资源，且可能导致收敛困难或过拟合。批大小越小，训练速度越慢，但会减少内存消耗，且可能提高泛化能力。因此，批大小需要根据数据集的规模和特点，以及模型的复杂度和性能进行调整。同时，批大小还与学习率相关。学习率是指每次更新参数时，沿着梯度方向移动的步长。一般来说，批大小和学习率成正比。如果批大小增大，学习率也相应增大；如果批大小减小，那么学习率也应减小。
训练轮数	1	1~50	完成全部训练数据集训练的次数。

训练参数	默认值	范围	说明
学习率	0.0001	0~1	学习率用于控制每个训练步数（step）参数更新的幅度。需要选择一个合适的学习，因为学习率过大会导致模型难以收敛，学习率过小会导致收敛速度过慢。
模型保存步数	500	10的倍数	每训练一定数量的步骤（或批次）后，模型的状态就会被保存下来。 可以通过token_num = step * batch_size * sequence公式进行预估。其中： ● token_num：已训练的数据量。 ● step：已完成的训练步数。 ● batch_size：每个训练步骤中使用的样本数据量。 ● sequence：每个数据样本中的token数量。 数据量以token为单位。
优化器	adamw	adamw	优化器参数指的是用于更新模型权重的优化算法的相关参数，可以选择adamw。 ● adamw是一种改进的Adam优化器，它在原有的基础上加入了权重衰减（weight decay）的机制，可以有效地防止过拟合（overfitting）的问题。
学习率衰减比率	0.00001	0~1	学习率衰减后，最小不会低于的学习率，计算公式为：学习率*学习率衰减比率。
热身比例	0.1	0~1	热身阶段占整体训练的比例。 模型刚开始训练时，如果选择一个较大的学习率，可能导致模型训练不稳定。选择使用warmup热身的方式，可以使开始训练的热身阶段内学习率较小，模型可以慢慢趋于稳定，待模型相对稳定后再逐渐提升至预设的最大学习率进行训练。使用热身可以使得模型收敛速度更快，效果更佳。

说明

当前盘古-NLP-N1-基模型与盘古-NLP-N4-基模型支持自监督训练。

表 5-6 盘古-NLP-N4-基模型训练参数推荐

训练参数	推荐值
数据批量大小	4
训练轮数	1

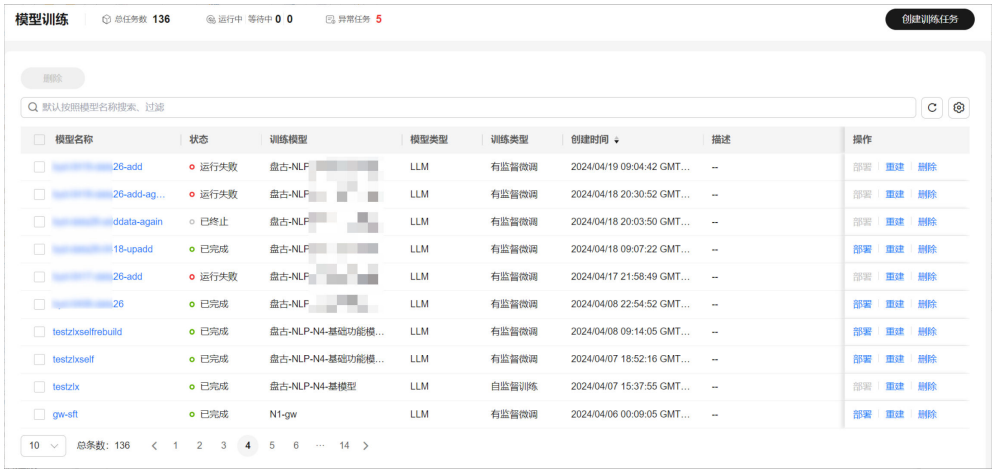
训练参数	推荐值
学习率	0.00001
模型保存步数	5000
优化器	adamw
学习率衰减比率	0.01
热身比例	0.1

5.2.2 创建有监督训练任务

创建有监督微调训练任务

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“模型开发 > 模型训练”，单击界面右上角“创建训练任务”。

图 5-5 模型训练列表



3. 在训练配置中，选择模型类型、训练类型、训练方式、训练模型与训练参数。

   其中，训练配置选择LLM（大语言模型），训练类型选择有监督训练，根据所选模型配置训练参数。

表 5-7 有监督微调参数说明

参数名称	说明
模型类型	选择“LLM”。
训练类型	选择“有监督微调”。

参数名称	说明
训练方式	• 全量微调：在模型有监督微调过程中，对大模型的全部参数进行更新。这种方法通常会带来最优的模型性能，但需要大量的计算资源和时间，计算开销较高。 • 局部微调（LoRA）：在模型微调过程中，只对特定的层或模块的参数进行更新，而其余参数保持冻结状态。这种方法在很多情况下可以显著减少计算资源和时间消耗，且依旧可以保持较好的模型性能。
训练模型	选择训练所需要的模型。支持选择“预置模型”或者“我的模型”。 • 预置模型：系统提供的LLM（大语言）预置模型。 • 我的模型：经过用户预训练或者微调训练后的模型。 模型详细介绍请参见**选择模型与训练方法**。
训练参数	指定用于训练模型的超参数。 训练参数说明和调参策略请参见**有监督微调（全量微调）训练参数说明**、**表5-13**。

4. 在数据配置中，选择训练数据集、验证数据等参数。

   验证数据可选择“从训练数据拆分”和“从已有数据导入”。

   - 从训练数据拆分：取值范围[1%-50%]。设置1%即从训练数据中随机拆分出1%的数据作为验证集，验证集中最多使用100条数据用于模型训练效果评估。数据按比例拆分后，如果超过100条，会**随机取100条**数据。
   - 从已有数据导入：从已有的数据集中选择数据用于模型训练效果评估，如果数据超过100条，会取**前100条**数据。

   图 5-6 从训练数据拆分

5. 完成训练任务基本信息。设置模型的名称、描述以及订阅提醒。

   设置订阅提醒后，模型训练和部署过程产生的事件可以通过手机或邮箱发送给用户。

   图 5-7 基本信息

6. 单击“立即创建”，创建有监督微调训练任务。

有监督微调（全量微调）训练参数说明

不同模型训练参数默认值存在一定差异，请以前端页面展示的默认值为准。

表 5-8 有监督微调（全量微调）参数说明

训练参数	默认值	范围	说明
数据批量大小	8	>=1	数据集进行分批读取训练，设定每个批次数据的大小。 一般来说，批大小越大，训练速度越快，但会占用更多的内存资源，且可能导致收敛困难或过拟合。批大小越小，训练速度越慢，但会减少内存消耗，且可能提高泛化能力。因此，批大小需要根据数据集的规模和特点，以及模型的复杂度和性能进行调整。同时，批大小还与学习率相关。学习率是指每次更新参数时，沿着梯度方向移动的步长。一般来说，批大小和学习率成正比。如果批大小增大，学习率也相应增大；如果批大小减小，那么学习率也应减小。
训练轮数	1	1~50	完成全部训练数据集训练的次数。

训练参数	默认值	范围	说明
学习率	0.0001	0~1	学习率用于控制每个训练步数（step）参数更新的幅度。需要选择一个合适的学习，因为学习率过大会导致模型难以收敛，学习率过小会导致收敛速度过慢。
优化器	adamw	adamw	用于更新模型权重的优化算法参数，可以选择adamw。 ● adamw是一种改进的Adam优化器，它在原有的基础上加入了权重衰减（weight decay）的机制，可以有效地防止过拟合（overfitting）的问题。
学习率衰减比率	0.00001	0~1	学习率衰减后，最小不会低于的学习率。计算公式为：学习率*学习率衰减比率。
热身比例	0.1	0~1	热身阶段占整体训练的比例。 模型刚开始训练时，如果选择一个较大的学习率，可能导致模型训练不稳定。选择使用warmup热身的方式，可以使开始训练的热身阶段内学习率较小，模型可以慢慢趋于稳定，待模型相对稳定后再逐渐提升至预设的最大学习率进行训练。使用热身可以使得模型收敛速度更快，效果更佳。
模型保存步数	1000	1000~2000中10的倍数	每训练一定数量的步骤（或批次）后，模型的状态就会被保存下来。 可以通过token_num = step * batch_size * sequence公式进行预估。其中： ● token_num：已训练的数据量。 ● step：已完成的训练步数。 ● batch_size：每个训练步骤中使用的样本数据量。 ● sequence：每个数据样本中的token数量。 数据量以token为单位。
流水线并行微批次大小	4	4、8、12、64	在流水线并行处理中，通过合理设置并行程度，可以减少各阶段之间的空闲等待时间，从而提升整个流水线的效率。
每个数据并行下的批处理大小	1	1、2	设置在并行训练中，每个微批次包含的数据批量大小。适当的数据批量大小能够确保训练各个阶段都能充分利用计算资源，提升并行效率。

说明

当前盘古-NLP-N2-基模型与盘古-NLP-N4-基模型支持有监督微调。

表 5-9 盘古-NLP-N1-基础功能模型-2K 训练参数推荐

应用场景	参数	推荐值
基础场景（文本分析、文本生成、文本翻译、query生成、开放问答、知识问答、改写、总结聚合、聊天）	数据批量大小	8
	训练轮数	4
	学习率	0.000005
	优化器	adamw
	学习率衰减比率	0.01
	热身比例	0.05

表 5-10 盘古-NLP-N1-基础功能模型-8K 训练参数推荐

应用场景	参数	推荐值
基础场景（文本分析、文本生成、文本翻译、query生成、开放问答、知识问答、改写、总结聚合、聊天）	流水线并行微批次大小	4
	训练轮数	2~11
	学习率	0.00005
	模型保存步数	1000
	优化器	adamw
	学习率衰减比率	0.01
	热身比例	0.05

表 5-11 盘古-NLP-N2-基模型训练参数推荐

应用场景	参数	推荐值
基础场景（文本分析、文本生成、文本翻译、query生成、开放问答、知识问答、改写、总结聚合、聊天）	数据批量大小	8
	训练轮数	4
	学习率	0.000075
	优化器	adamw
	学习率衰减比率	0.067
	热身比例	0.01
NL2SQL场景	数据批量大小	8

应用场景	参数	推荐值
	训练轮数	4
	学习率	0.00001
	优化器	adamw
	学习率衰减比率	0.067
	热身比例	0.013
NL2JSON场景	数据批量大小	8
	训练轮数	3
	学习率	0.000075
	优化器	adamw
	学习率衰减比率	0.01
	热身比例	0.01
NL2CODE场景	数据批量大小	8
	训练轮数	4
	学习率	0.00001
	优化器	adamw
	学习率衰减比率	0.067
	热身比例	0.013

表 5-12 盘古-NLP-N4-基础功能模型-4K 训练参数推荐

应用场景	参数	推荐值
基础场景（文本分析、文本生成、文本翻译、query生成、开放问答、知识问答、改写、总结聚合、聊天）	数据批量大小	8
	训练轮数	6
	学习率	0.000003
	模型保存步数	1000
	优化器	adamw
	学习率衰减比率	0.01
	热身比例	0.05

有监督微调（局部微调）训练参数说明

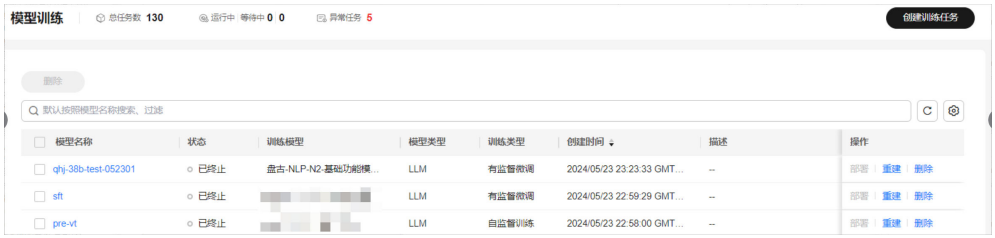
表 5-13 有监督微调（局部微调）参数说明

训练参数	默认值	范围	说明
LoRA秩值	/	8、16、32、64	较高的取值意味着更多的参数被更新，模型具有更大的灵活性，但需要更多的计算资源和内存。较低的取值则意味着更少的参数更新，资源消耗更少，但模型的表达能力可能受到限制。
训练轮数	4	1~50	完成全部训练数据集训练的次数。
学习率	0.0001	0~1	学习率用于控制每个训练步数（step）参数更新的幅度。需要选择一个合适的学习，因为学习率过大会导致模型难以收敛，学习率过小会导致收敛速度过慢。
优化器	adamw	adamw	优化器参数指的是用于更新模型权重的优化算法的相关参数，可以选择adamw。 ● adamw是一种改进的Adam优化器，它在原有的基础上加入了权重衰减（weight decay）的机制，可以有效地防止过拟合（overfitting）的问题。
学习率衰减比率	0.1	0~1	学习率衰减后，最小不会低于的学习率，计算公式为：学习率*学习率衰减比率。
热身比例	0.01	0~1	热身阶段占整体训练的比例。 模型刚开始训练时，如果选择一个较大的学习率，可能导致模型训练不稳定。选择使用warmup热身的方式，可以使开始训练的热身阶段内学习率较小，模型可以慢慢趋于稳定，待模型相对稳定后再逐渐提升至预设的最大学习率进行训练。使用热身可以使得模型收敛速度更快，效果更佳。

5.3 查看训练任务详情与训练指标

模型启动训练后，可以在模型训练列表中查看训练任务的状态，单击任务名称可以进入详情页查看训练指标、训练任务详情和训练日志。

图 5-8 模型训练列表



不同类型的训练方法可支持查看的训练指标有所差异，训练指标和训练方法的关系如下：

表 5-14 训练指标和训练方法对应关系

训练指标\模型类型	自监督训练	有监督训练
训练损失值	√	√
模型准确率	×	√
指标看板	×	√
困惑度	×	√

训练损失值指标介绍

训练损失值（Training Loss）是一种衡量模型预测结果和真实结果差距的指标，通常情况下越小越好。

一般来说，一个正常的Loss曲线是单调递减的，即随着训练的进行，Loss值不断减小，直到收敛到一个较小的值。以下给出了几种正常的Loss曲线形式：

图 5-9 正常的 Loss 曲线：平滑下降

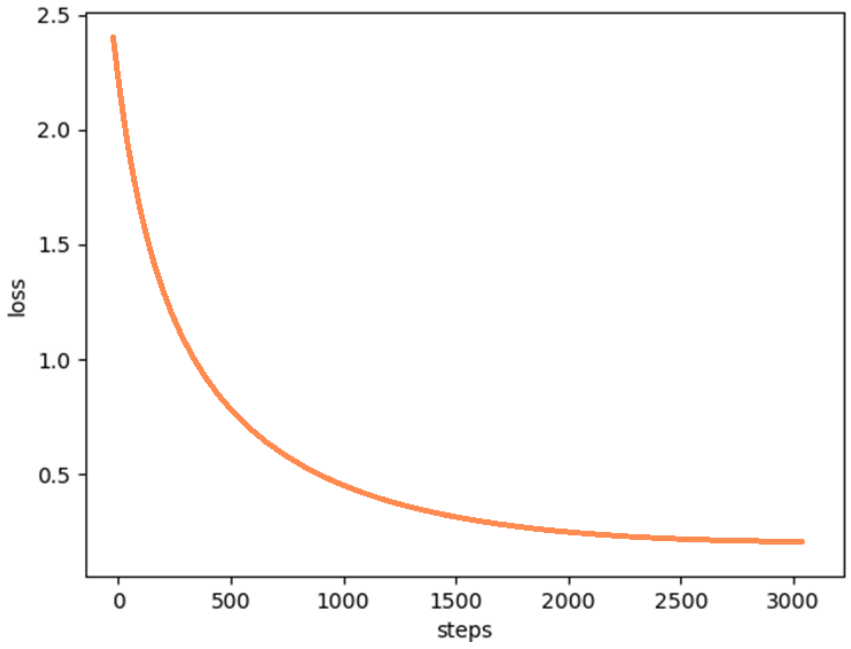
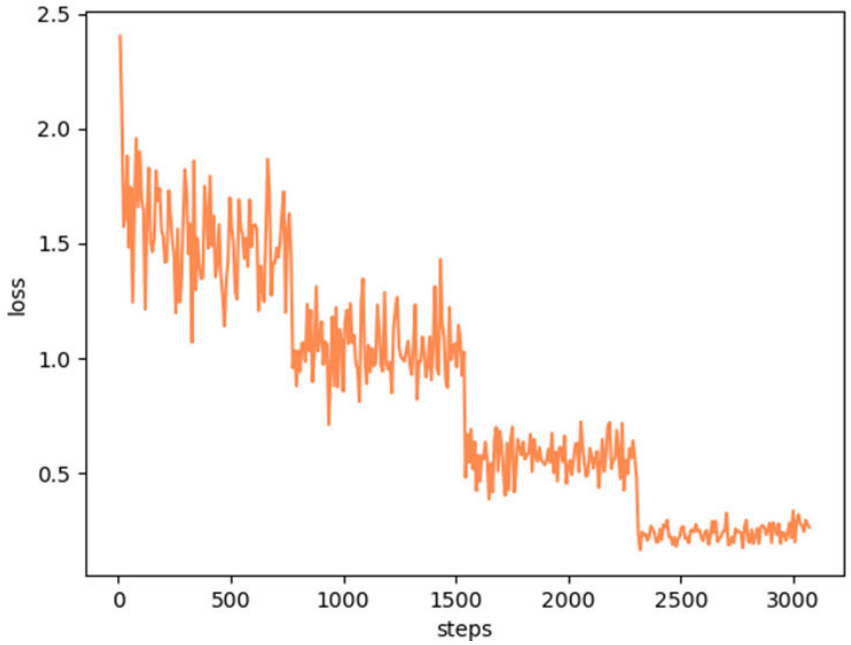


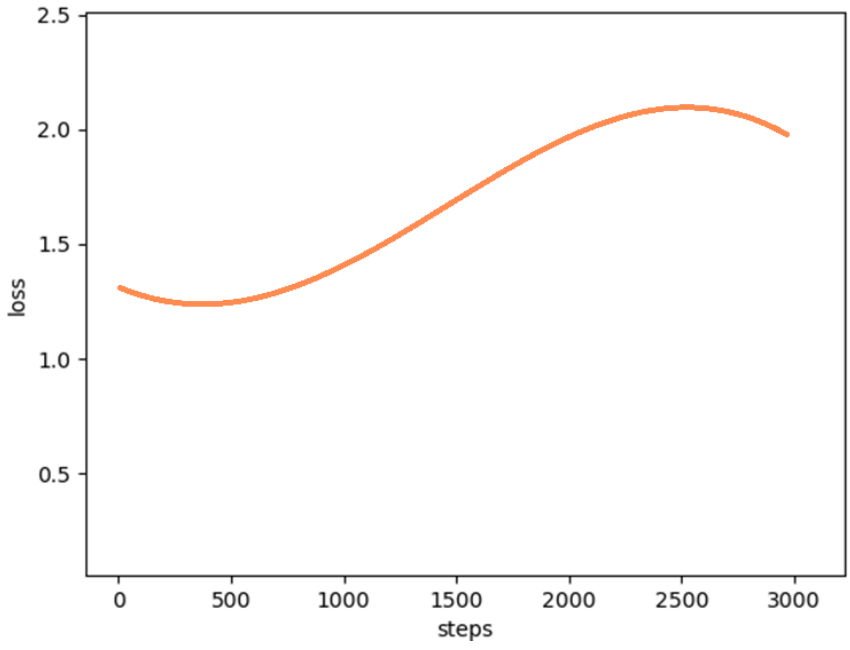
图 5-10 正常的 Loss 曲线：阶梯下降



如果您发现Loss曲线出现了以下几种情况，可能意味着模型训练状态不正常：

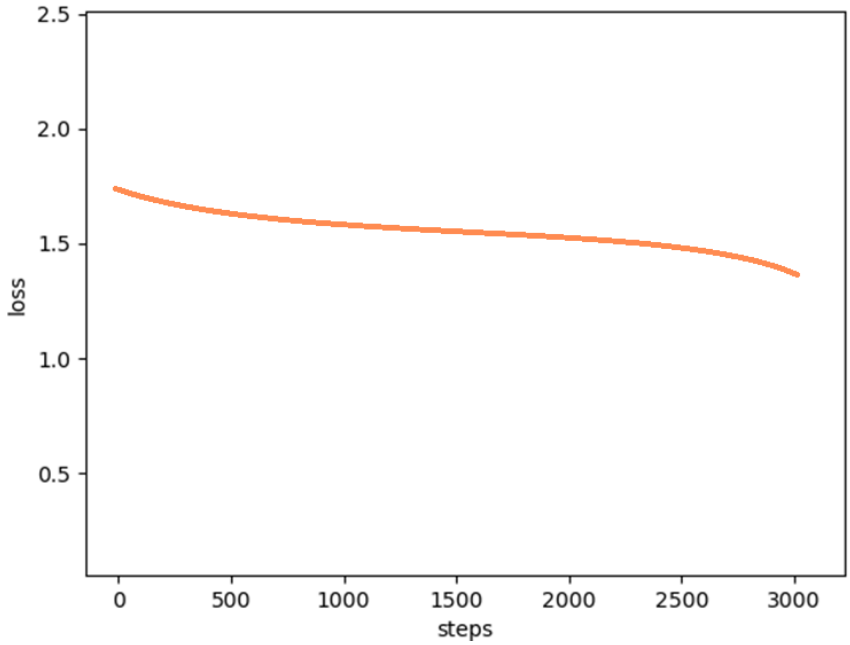
- **Loss曲线上升：**Loss曲线上升的原因可能是数据质量差，或学习率设置过大，使得模型在最优解附近震荡，甚至跳过最优解，导致无法收敛。您可以尝试提升数据质量或减小学习率来解决。

  图 5-11 异常的 Loss 曲线：上升



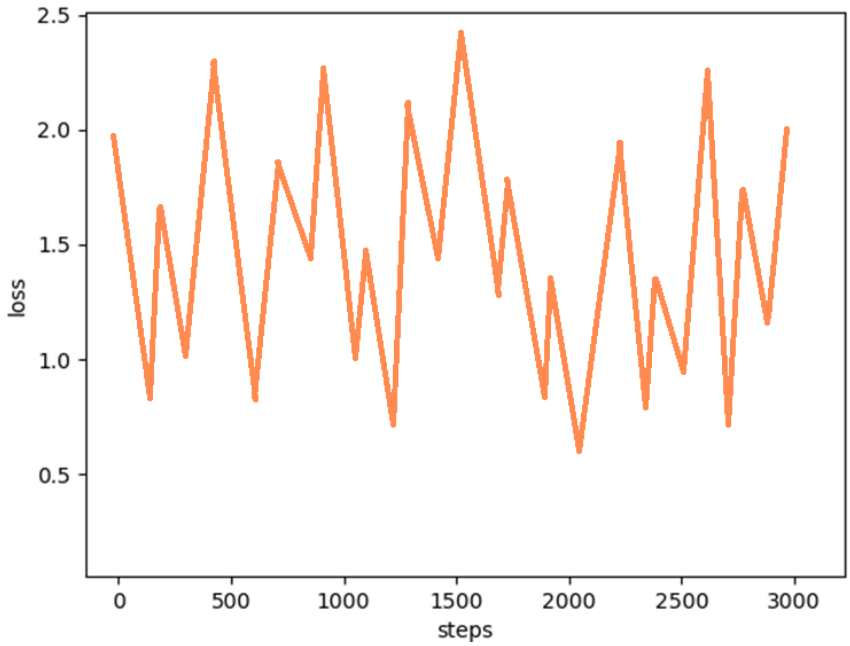
- **Loss曲线平缓，保持高位：**Loss曲线平缓且保持高位不下降的原因可能是目标任务的难度较大，或模型的学习率设置过小，导致模型的收敛速度太慢，无法达到最优解。您可以尝试增大训练轮数或者增大学习率来解决。

图 5-12 异常的 Loss 曲线：平缓且保持高位



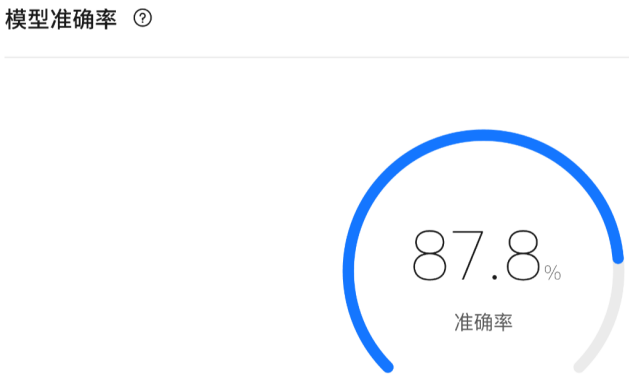
- **Loss曲线异常抖动：**Loss曲线异常抖动的原因可能是训练数据质量差，比如数据存在噪声或分布不均衡，导致训练不稳定。您可以尝试提升数据质量来解决。

图 5-13 异常的 Loss 曲线：异常抖动



模型准确率指标介绍

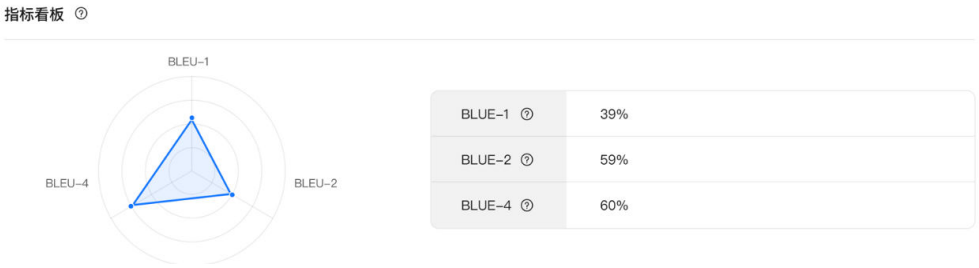
模型准确率：正确预测（标注与预测完全匹配）的样本数与总样本数的比例。模型准确率越高，表明模型性能越好。



指标看板介绍

指标看板使用BLEU指标评价模型，其核心思想是计算准确率。例如，给定一个标准译文（reference）和一个算法生成的句子（candidate），BLEU-1的计算公式为候选句中出现于标准译文中的单词数（m）与候选句总单词数（n）的比值，即m/n。指标看板通过BLEU-1、BLEU-2与BLEU-4评价模型性能。

- BLEU-1：机器翻译、文本摘要等生成类任务常用的评价指标。用于评估模型生成句子与实际句子在单字层面的匹配度，数值越高，表明模型性能越好。
- BLEU-2：机器翻译、文本摘要等生成类任务常用的评价指标。用于评估模型生成句子与实际句子在中词组层面的匹配度，数值越高，表明模型性能越好。
- BLEU-4：机器翻译、文本摘要等生成类任务常用的评价指标。它通过将模型生成结果和标注结果分别按1-gram、2-gram、3-gram和4-gram拆分后，然后计算加权平均精确率。其中，n-gram指的是一个句子中连续的n个单词片段。BLEU-4 的数值越高，表明模型性能越好。



困惑度指标介绍

困惑度用来衡量大语言模型预测一个语言样本的能力。数值越低，准确率越高，表明模型性能越好。

5.4 常见训练报错与解决方案

read example failed 报错

- 报错原因：模型训练过程中，训练日志出现“read example failed”报错，表示当前数据集格式不满足训练要求。
- 解决方案：请参考**数据格式要求**校验数据集格式。

图 5-14 read example failed 报错

```
[ERROR] [2023-08-25 10:40:20] rank_1: read example failed, example len is 0
[ERROR] [2023-08-25 10:40:20] rank_1: Traceback (most recent call last):
  File "/home/ma-user/code/pangu_nlp/src/obs.py", line 441, in wrapped_func
    run_func(cloud_adapter) # main function
  File "/home/ma-user/code/pangu_nlp/train.py", line 127, in run_pangu_train
    args, cloud_adapter, dev=False)
  File "/home/ma-user/code/pangu_nlp/src/dataset/multitask_dataset.py", line 244, in create_multitask_dataset
    lm_task_configs)
  File "/home/ma-user/code/pangu_nlp/src/dataset/multitask_dataset.py", line 202, in _multitask_unified_datasets_provider
    test=args.do_sup_test)
  File "/home/ma-user/code/pangu_nlp/src/dataset/task_dataset.py", line 554, in create_downstream_dataset
    sup_test_ds_dict, train, dev, test, lm=lm)
  File "/home/ma-user/code/pangu_nlp/src/dataset/task_dataset.py", line 516, in build_multipattern_unified_dataset
    examples, task_name, task_configs[task_name], pattern, lm, processor)
  File "/home/ma-user/code/pangu_nlp/src/dataset/task_dataset.py", line 457, in build_singlepattern_unified_dataset
    verbalizer, is_tokenized, processor=processor) if train_examples is not None else None
  File "/home/ma-user/code/pangu_nlp/src/dataset/task_dataset.py", line 42, in __init__
    raise ValueError('read example failed, example len is 0')
ValueError: read example failed, example len is 0
```

no such file or directory 报错

- 报错原因：模型训练过程中，训练日志出现“no such file or directory”报错，表示当前数据集格式、数据命名、数据存储路径不满足训练要求。
- 解决方案：请参考**数据格式要求**校验数据集格式。

请检查数据集路径是否设置正确。

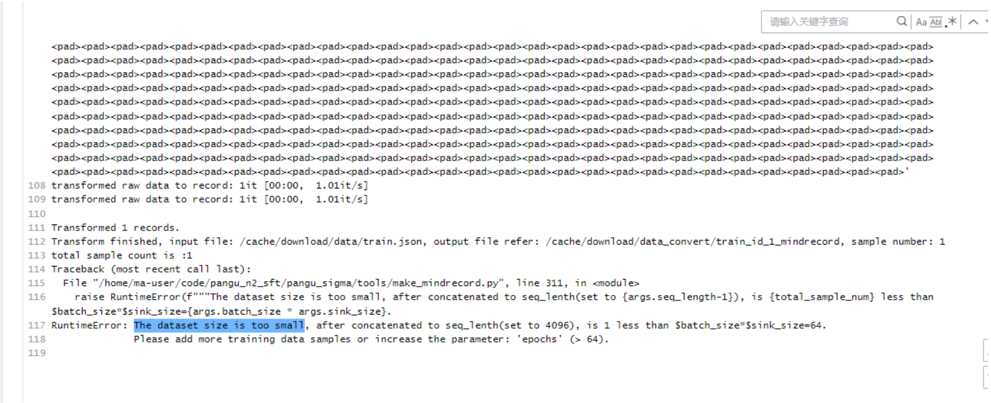
图 5-15 no such file or directory 报错

```
  File "/home/ma-user/code/pangu_nlp/src/dataset/task_processor.py", line 85, in get_train_examples
    origin_examples = self._create_examples(self._read_json(train_path), "train")
  File "/home/ma-user/code/pangu_nlp/src/dataset/task_processor.py", line 113, in _read_json
    with open(input_file, "r", encoding='utf-8', errors='ignore') as f:
  File "/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages/moxing/framework/file/file_io_utils.py", line 306
    return _origin_open(url, mode, *args, **kwargs)
FileNotFoundError: [Errno 2] No such file or directory: '/cache/download/data/question_answer/train.json'
+ echo 'finish training'
finish training
+ echo 'finish training'
finish training
time="2023-09-02T14:19:03+08:00" level=info msg="npu-smi sampler routine is exiting" file="sample_routine.go:107" Command=bo
time="2023-09-02T14:19:03+08:00" level=info msg="zombie process cleaner is exiting" file="cleaner_unix.go:53" Command=bootst
time="2023-09-02T14:19:03+08:00" level=info msg="hccn sampler routine is exiting" file="sample_routine.go:104" Command=boots
time="2023-09-02T14:19:03+08:00" level=info msg="stop run_command_pid" file="run_train.go:455" Command=bootstrap/run Compone
time="2023-09-02T14:19:03+08:00" level=error msg="start run_train script failed, error: failed to get process exit code, exi
```

The dataset size is too small

- 报错原因：模型训练过程中，训练日志出现“The dataset size is too small”报错，表示数据量太少，拼接到模型要求长度后，条数不满足一次训练下沉。
- 解决方案：请增大数据集大小或者把epochs设大，保证日志中的Sink_num > 0。

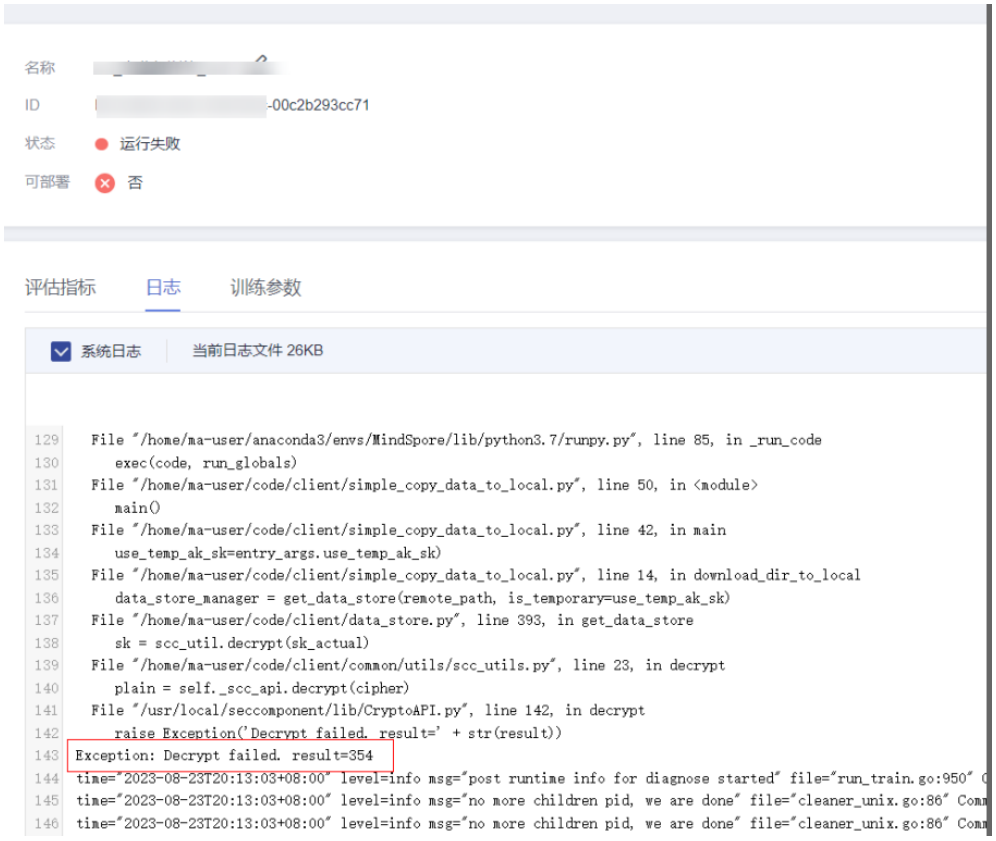
图 5-16 The dataset size is too small



Decrypt failed

- 报错原因：模型训练过程中，训练日志出现“Decrypt failed”报错，表示解密失败。
- 解决方案：请联系华为云排查环境变量ak、sk。

图 5-17 Decrypt failed 报错



6 评估盘古大模型

6.1 创建模型评估数据集

在收集评估数据集时，应确保数据集的独立性和随机性，并使其能够代表现实世界的样本数据，以避免对评估结果产生偏差。对评估数据集进行分析，可以帮助了解模型在不同情境下的表现，从而得到模型的优化方向。

在“数据工程 > 数据管理”中创建“评测”类型的数据集作为评估数据集，数据集创建完成后需要执行发布操作。

数据量建议3-1000条。当前数据集数据保存与上传的文件类型有以下两种，大小均不可超过1024MB。

- **文件类型为JSONL：**每一行表示一段文本，形式为{"context":"context内容","target":"target内容"}，每一段需要准确完整的语义，符合主流价值观，并且文本中不能存在异常字符、分行异常等影响模型训练的问题。问题和答案需要匹配，且不能有空值。
- **文件类型为CSV：**每一行代表一个问答对，确保每个问题和答案的数据都以逗号分隔，每行的数据完整且格式正确，文件中每个字段或列都应有适当的数据类型，例如文本、数值、日期等。每一段需要准确完整的语义，符合主流价值观，并且文本中不能存在异常字符、分行异常等影响模型训练的问题。问题和答案需要匹配，且不能有空值。

说明

当前仅支持对NLP大模型进行模型评估操作。

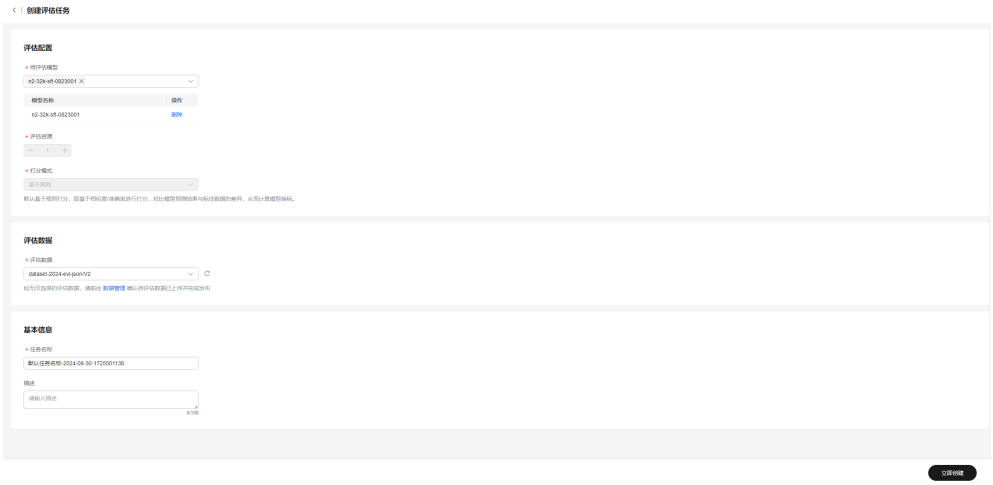
6.2 创建模型评估任务

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“模型开发 > 模型评估”。
3. 单击界面右上角“创建评估任务”，进入评估任务创建页面。

图 6-1 模型评估列表页面

4. 填写评估任务所需的评估配置、评估数据和基本信息。

图 6-2 创建评估任务



- 评估配置：
  - **待评估模型：**支持选择多个模型版本同时评估，最多选择5个。待评估模型必须符合**前提条件。**
  - **评估资源**：依据选择的模型数据自动给出所需的评估资源。
  - **打分模式：**当前版本打分模式仅支持**基于规则**，用户不可选，且暂无人工打分。基于规则打分：使用预置的相似度或准确率打分规则对比模型生成结果与真实标注的差异，从而计算模型指标。
- 评估数据：

  选择已创建并发布的评估数据集。
- 基本信息：

  输入任务的名称和描述。

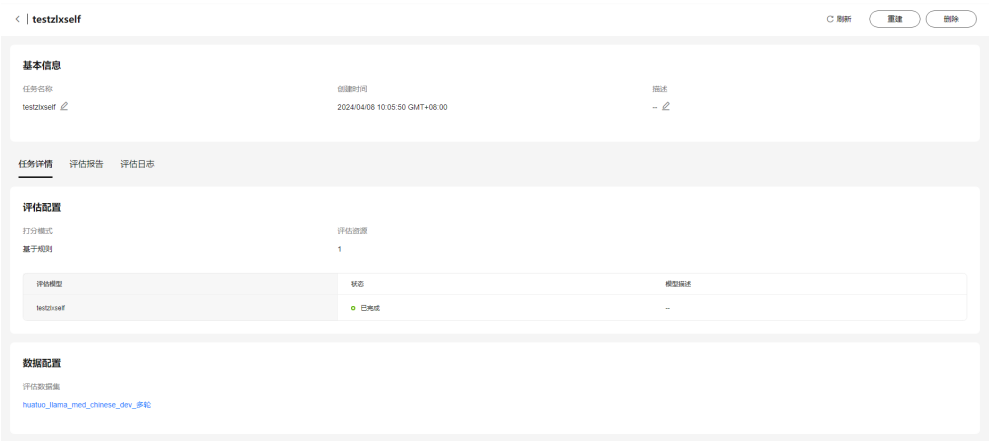
5. 单击“立即创建”，创建一个模型评估任务。

6.3 查看评估任务详情

查看评估任务详情

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“模型开发 > 模型评估”。
3. 单击任务名称查看模型评估任务详情。包含基本信息、评估详情、评估报告、评估日志以及数据配置。

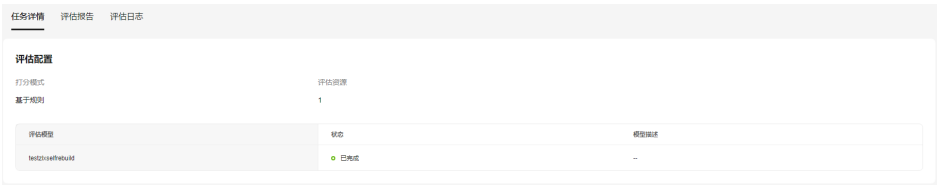
图 6-3 任务详情界面



- **任务详情：**

  任务详情中包含打分模式、评估资源、评估模型、任务状态以及模型描述。

  图 6-4 任务详情

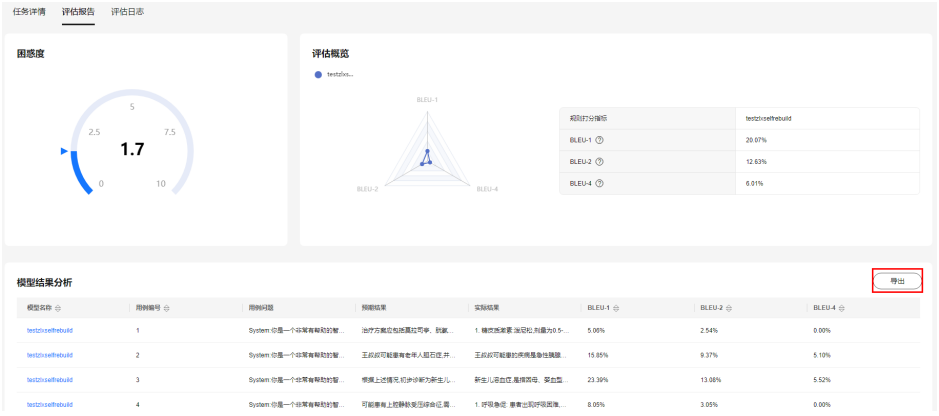
- **评估报告：**

  任务状态为“已完成”时，查看评估报告。评估报告中包含困惑度、评估概览以及模型结果分析。

  - 困惑度：分数越低，评估结果越好。
  - 评估概览：查看此次评估任务的各个规则指标评分情况。
  - 模型结果分析：查看各个模型此次评估任务的基于各个指标的评分情况，以及具体到某条数据的打分情况。

    用户可以将此次的评估报告通过导出按钮全部导出至本地存储，文件导出格式为CSV。

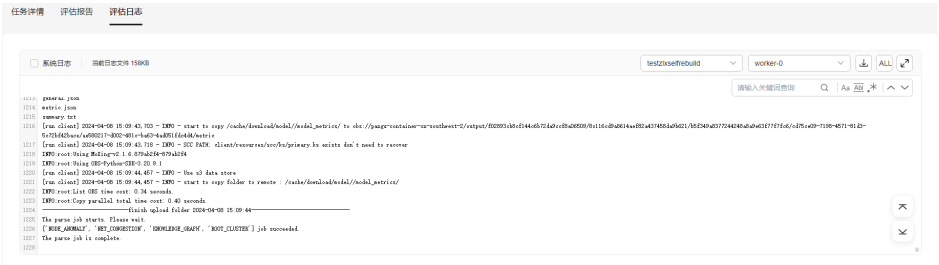
图 6-5 评估报告页面



- **评估日志：**

  平台支持查看本次模型评估任务的详细日志。选择评估的模型后，可以查看其从创建开始到任务结束的日志内容，支持下载并保存到本地（log格式）。可通过日志查看报错，调整任务配置重新发起。

图 6-6 评估日志页面



评估指标说明

模型训练完成后，可以通过一系列的评估方法来衡量模型的性能。当前支持基于规则打分，即基于相似度/准确率进行打分，对比模型预测结果与标注数据的差异，从而计算模型指标。支持的模型指标请参见下表。

表1 规则打分指标

指标名称	说明
BLEU-1	模型生成句子与实际句子在单字层面的匹配度，数值越高，表明模型性能越好。
BLEU-2	模型生成句子与实际句子在中词组层面的匹配度，数值越高，表明模型性能越好。
BLEU-4	模型生成结果和实际句子的加权平均精确率，数值越高，表明模型性能越好。
困惑度	用来衡量大语言模型预测一个语言样本的能力，数值越低，准确率也就越高，表明模型性能越好。

- **指标适用的任务场景**

  任务答案是相对比较确定的，例如固定答案的问答任务、NL2SQL、NL2JSON、文本分类等。

- **指标不适用的任务场景**

  文案创作、聊天等符合要求即可的场景，该类场景的创作型更强，不存在唯一答案。

- **指标与模型能力的关系**

  BLEU指标用于评估模型生成句子（candidate）与实际句子（reference）差异的指标。取值范围在0.0到1.0之间，值越高说明模型生成和实际答案匹配度越高。

  可以作为模型能力的参考指标，当两个模型进行比较时，BLEU指标越大的模型效果一般更好。但是模型的能力还是需要通过人工评测来评判，BLEU指标只能作为参考。

- **指标的缺陷**

  BLEU指标只考虑n-gram词的重叠度，不考虑句子的结构和语义。

模型优化建议

如何基于指标的分值对训练任务进行调整：一般横向比较两个模型时，可以参考该指标。然而，指标没有一个明确的阈值来指示何时模型效果差。因此，单靠该指标无法直接决定任务的调整策略。

如果指标低是由于提示词（prompt）设置不合理，可以通过在模型训练阶段扩大训练集和验证集来优化模型，从而改善评估结果。另外，还可以将评估数据集设计得更接近训练集的数据，以提升评估结果的准确性。

7 压缩盘古大模型

N2基础功能模型、N4基础功能模型、经有监督微调训练以及RLHF训练后的N2、N4模型可以通过模型压缩技术在保持相同QPS目标的情况下，降低推理时的显存占用。

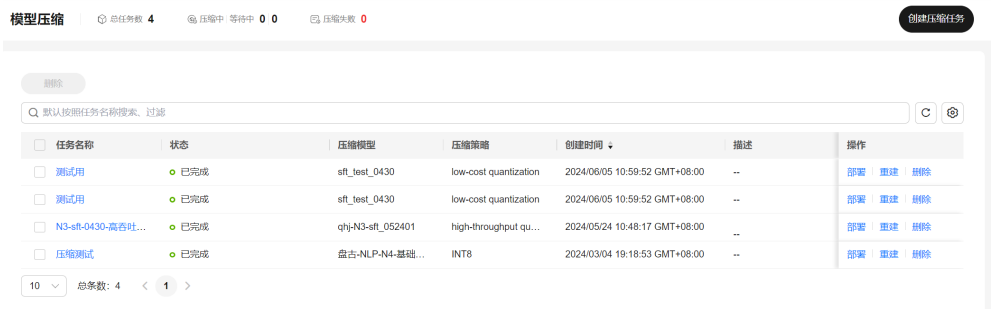
采用INT8的压缩方式，INT8量化可以显著减小模型的存储大小与降低功耗，并提高计算速度。

模型经过量化压缩后，不支持评估操作，但可以进行部署操作。

创建模型压缩任务

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“模型开发 > 模型压缩”。
3. 单击界面右上角“创建压缩任务”，进入创建压缩任务页面。

图 7-1 模型压缩



4. 选择需要进行压缩的模型执行模型压缩，压缩策略为“INT8”。当压缩模型为N2基础功能模型，或是经有监督微调训练和RLHF训练后的N2模型，支持选择“低消耗模式”，减少推理资源的消耗。

图 7-2 创建压缩任务

5. 输入任务名称和描述，单击“立即创建”，即可下发压缩模型任务。模型压缩任务完成后，可以使用压缩后的模型进行部署操作。

8 部署盘古大模型

8.1 部署为在线服务

模型训练完成后，即模型处于“已完成”状态时，可以启动模型的部署操作。

说明

基于盘古大模型打造的专业大模型包括BI专业大模型与单场景大模型支持模型推理，但不支持模型训练。

部署为在线服务

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“模型开发 > 模型部署”，单击界面右上角“部署”。
3. 在创建部署页面，完成部署配置，填写基本信息。

表 8-1 部署配置参数

参数名称	说明
选择模型	选择需要部署的模型。
推理资源	选择非限时免费的模型时显示。选择盘古大模型服务提供的在线推理资产。
部署方式	选择“在线部署”，即将算法部署至盘古大模型服务提供的资源池中。
推理资产	选择“已购资产”。 ● 限时免费：使用免费的推理资源，仅支持部署一个实例。 ● 已购资产：由用户购买的推理资源，实际可用推理单元由购买时的数量决定。
实例数	实例数越大，能够同时处理的请求数量越多。

参数名称	说明
高级配置	选择盘古-NLP-N4系列模型时显示，配置最大Token长度。
服务名称	在线服务的名称。
描述	在线服务的简要描述。
订阅提醒	勾选订阅提醒，并添加手机号/邮箱，系统将在训练任务完成或重要事件发生时，发送提醒。

表 8-2 部署实例量与推理单元数关系

模型类型	推理资源
盘古-NLP-N1系列模型	2K版：部署1实例占用0.125个推理单元。 8K版：最大Token长度为8192，部署1实例占用0.125个推理单元；最大Token长度为16384，部署1实例占用0.25个推理单元。
盘古-NLP-N2系列模型	部署1实例占用0.5个推理单元。
盘古-NLP-N4系列模型	部署1实例占用1个推理单元。

4. 单击“立即创建”，下发模型部署任务。

使用外推扩展模型上下文处理长度

在部署模型、部署后修改模型规格时，可以通过外推功能调整模型的输入输出长度。修改部署时**扩缩容**和外推场景互斥，每次只能修改一个。

当前仅盘古-NLP-N4系列模型、盘古-NLP-N1系列模型（8K版）以及基于它们训练的模型支持外推。

图 8-1 模型部署外推升级

扩缩容部署实例数量

扩缩容是指运行中的模型支持增加或减少模型部署的实例数。

修改部署时扩缩容和**外推**场景互斥，每次只能修改一个。

图 8-2 修改部署

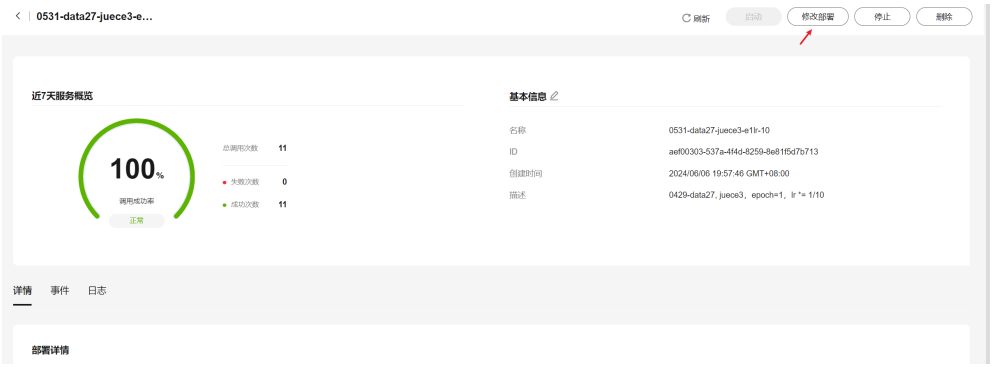


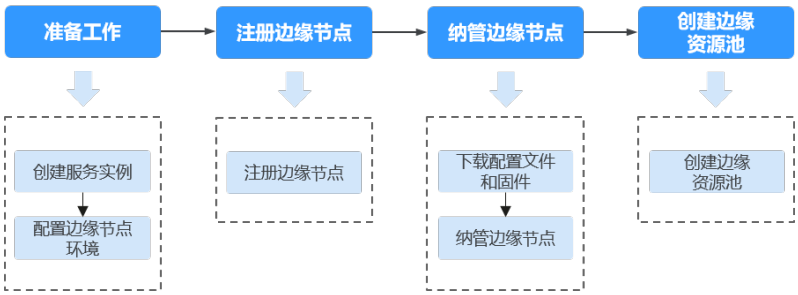
图 8-3 模型部署扩缩容

8.2 部署为边缘服务

8.2.1 边缘服务部署流程

边缘部署是指将模型部署到用户的边缘设备上。这些设备通常是用户自行采购的服务器，通过ModelArts服务纳管为边缘资源池。然后利用盘古大模型服务将算法部署到这些边缘资源池中。

图 8-4 边缘资源池创建步骤



说明

- 当前仅支持**预置模型**（盘古-NLP-N2-基础功能模型）和**基于N2的模型**（盘古-NLP-N2-基模型、盘古-NLP-N2-基础功能模型、盘古-NLP-N2-SQL模型、盘古-NLP-N2-Agent模型、盘古-NLP-N2-Code模型）经有监督微调（SFT）训练后的用户模型进行边缘部署。
- 使用边缘部署功能需要在ModelArts服务中开通“边缘资源池”功能，该功能为**白名单特性**，需要联系ModelArts服务技术支持人员进行开通。
- **创建边缘资源池操作较为复杂，建议联系盘古服务技术支持人员进行协助。**

8.2.2 边缘部署准备工作

本指南的边缘部署操作以largemodel集群为例，示例集群信息如下表。

表 8-3 示例集群信息

集群名	节点类型	节点名	规格	备注
largemodel	controller	ecs-edge-30037210	鲲鹏通用计算型|8vCPUs|29GiB|rc3.2xlarge.4镜像 EulerOS 2.9 64bit with ARM for Tenant 20230728 base 2.9.15	公网IP：100.85.220.207 root密码： CPU架构：aarch64（登录设备，执行arch命令查看）
	worker	bms-pangu30037210	CPU:Kunpeng 920(4*48Core@2.6GHz) 内存：24*64GB DDR4 RAM(GB) 本地磁盘：3*7.68TB NVMe SSD 扩展配置：2*100GE +8*200GE 类型：physical.kat2e.48xlarge.8.313t.ei.pod101 euler2.10_arm_sdi3_1980b_hc_sdi5_b080_20230831v2	公网IP：100.85.216.151 root密码： CPU架构：aarch64（登录设备，执行arch命令查看）

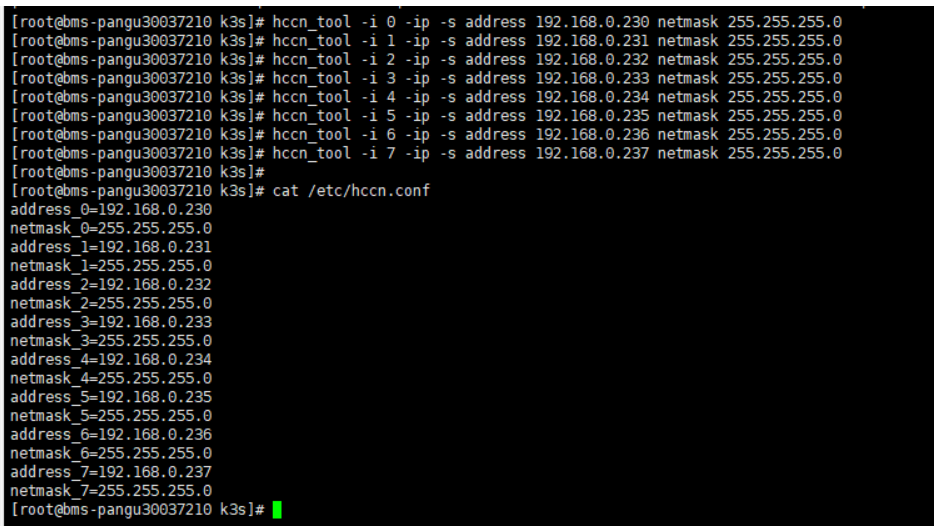
1. 依赖包下载。
   - docker下载：**https://download.docker.com/linux/static/stable**

   选择对应cpu架构下载，docker版本选在19.0.3+。
   - K3S下载：**https://github.com/k3s-io/k3s/releases/tag/v1.21.12%2Bk3s1**

   按照对应cpu架构下载二进制文件以及air-gap镜像。
2. npu驱动和固件安装。

   执行命令**npu-smi info**查看驱动是否已安装。如果有回显npu卡信息，说明驱动已安装。

   详情请参见**昇腾官方文档**。
3. hccn too网卡配置。
   a. 执行如下命令，查看是否有回显网卡信息。如果有，则说明网卡已经配置，否则继续操作下面步骤。
      ```
      cat /etc/hccn.conf
      ```
   b. 执行如下命令，查看npu卡数。
      ```
      npu-smi info
      ```
   c. 执行如下命令（地址自行配置）：
      ```
      hccn_tool -i 0 -ip -s address 192.168.0.230 netmask 255.255.255.0
      hccn_tool -i 1 -ip -s address 192.168.0.231 netmask 255.255.255.0
      hccn_tool -i 2 -ip -s address 192.168.0.232 netmask 255.255.255.0
      hccn_tool -i 3 -ip -s address 192.168.0.233 netmask 255.255.255.0
      hccn_tool -i 4 -ip -s address 192.168.0.234 netmask 255.255.255.0
      hccn_tool -i 5 -ip -s address 192.168.0.235 netmask 255.255.255.0
      hccn_tool -i 6 -ip -s address 192.168.0.236 netmask 255.255.255.0
      hccn_tool -i 7 -ip -s address 192.168.0.237 netmask 255.255.255.0
      ```
   d. 执行命令**cat /etc/hccn.conf**，确保有如下回显网卡信息，则配置完成。

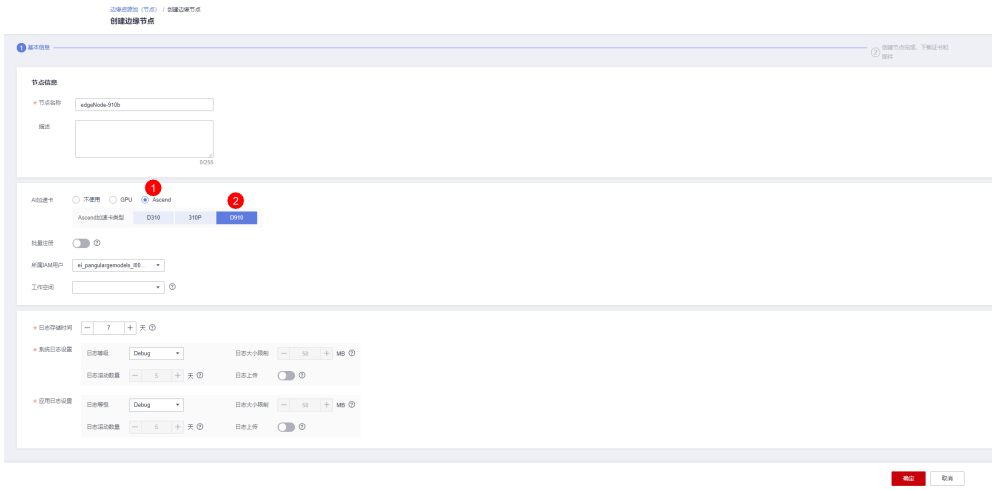
4. 配置NFS网盘服务。

   大模型采用镜像+模型分开的方式部署时，需要有一个节点来提供NFS网盘服务，创建部署时通过NFS挂载的方式访问模型。

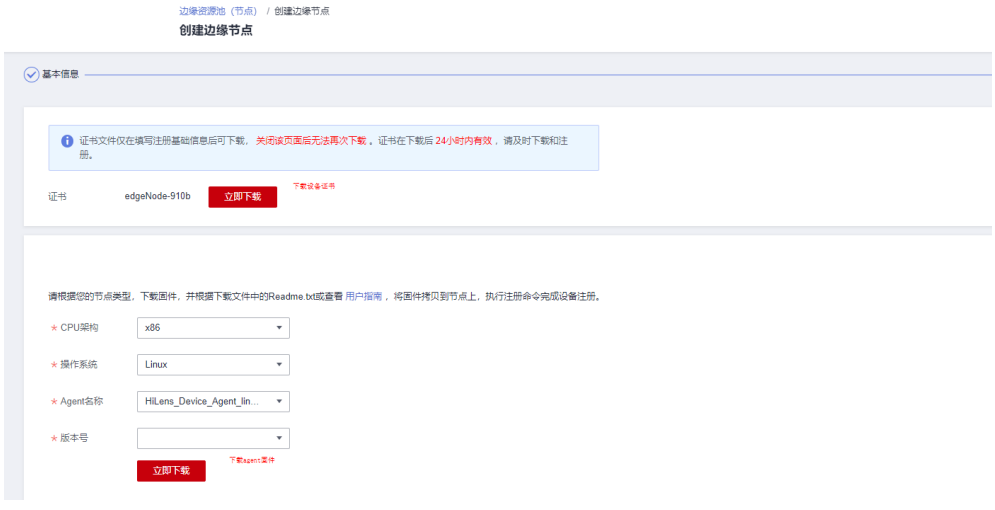
8.2.3 注册边缘资源池节点

1. 进入ModelArts服务，选择所需空间。
2. 在左侧列表中单击“边缘资源池”，在“节点”页签中，单击“创建”。

3. 在“创建边缘节点”页面中，填写节点名称，配置AI加速卡与日志信息，单击“确定”。
   - 如果节点有npu设备需选择“AI加速卡 > Ascend”，并选择加速卡类型，如D910。
   - 如果节点没有加速卡，则选择“AI加速卡 > 不使用”。



4. 单击“立即下载”，下载设备证书和Agent固件，并将设备证书与Agent固件分别重命名为**license.tgz**、**hilens-agent.tgz**。

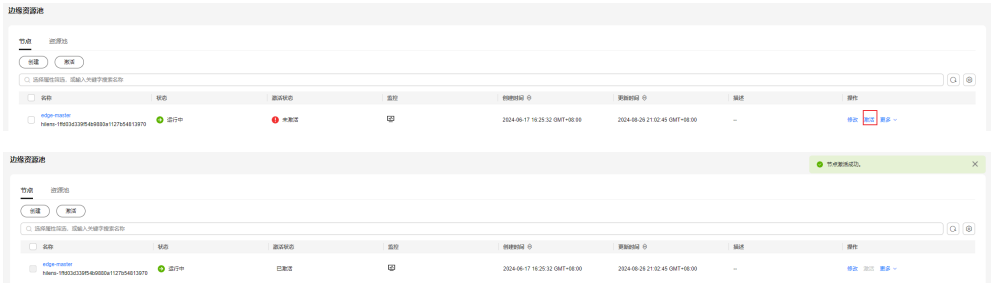


8.2.4 搭建边缘服务器集群

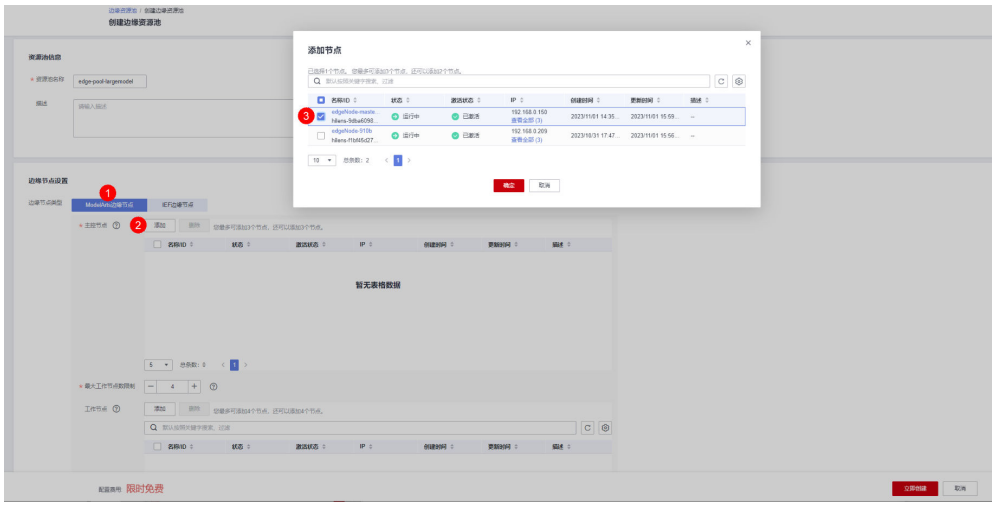
1. 执行如下命令，生成docker证书。注意该命令只需执行一次，如果已有相关证书，请跳过该步骤。
```
bash cluster_install-ascend.sh generate_docker_cert --pkg-path=/home/hilens/pkgs
```
2. 基于**边缘部署准备工作**与**注册边缘资源池节点**，按照以下目录结构存放下载文件，注意修改下载文件的命名。其中，docker下的certs证书会自动生成，一般无需修改。
```
pkgs // 包目录，用户自行命名
    docker
        docker.tgz    // docker 二进制文件，要求版本>19.0.3
        certs         // 使用generate命令生成的证书，指定--pkg-path后会自动创建到certs目录
            ca.crt
            server.crt
            server.key
    k3s
        k3s // k3s可执行文件
        agent
            images
                k3s-airgap-images-[arm64|amd64].tar.gz //k3s离线镜像
    hilens-agent
        hilens-agent.tgz // hilens agent固件包
        license.tgz // hilens 设备license
```
3. 工作节点执行命令如下：
```
bash -x cluster_install-ascend.sh --pkg-path=/home/hilens/pkgs --node-type=worker --host-ip=192.168.0.209
```
主控节点执行命令如下：
```
bash -x cluster_install-ascend.sh --pkg-path=/home/hilens/pkgs --node-type=controller --host-ip=192.168.0.150
```
   - **cluster_install-ascend.sh**脚本主要用于安装docker、hdad和k3s，请联系华为工程师获取。
   - **pkg-path**是步骤2中整合的安装包文件目录。
   - **host-ip**是设备在集群中的ip，一般为内网ip。
   - **node-type**是集群节点类型。其中，worker表示工作节点，controller表示主控节点。
4. 在服务器执行如下命令，判断docker是否安装成功。
```
systemctl status docker
```
```
[root@bms-pangu30037210 hilens-agent]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2023-11-01 14:23:17 CST; 2h 32min ago
     Docs: https://docs.docker.com
 Main PID: 1629388 (dockerd)
    Tasks: 65
   Memory: 31.0M
   CGroup: /system.slice/docker.service
           ├─1629388 /usr/bin/dockerd -H fd://
           └─1629412 containerd --config /var/run/docker/containerd/containerd.toml --log-level info
```
5. 在服务器执行如下命令，判断edge agent是否安装成功。
```
hdactl info
```

```
[root@bms-pangu30037210 hilens-agent]# hdactl info
{
 "device_info": {
  "device_id": "hilens-f1bf45d27c464413b6b53fa40d9a1c0d",
  "device_ip": "192.168.0.209",
  "device_name": "edgeNode-910b",
  "device_type": "General ARM Device",
  "domain": "",
  "is_active": true,
  "obs": "",
  "projectId": "51fae1a6ef804316abeaadb80e302339"
 },
 "dm": {
  "state": "success",
  "connect_err": "",
  "last_message": {
   "topic": "pong",
   "timestamp": 1698829033
  },
  "channel_available": true,
  "address": "wss://modelarts-dev.cn-north-7.myhuaweicloud.com/v3/login"
 }
}
```

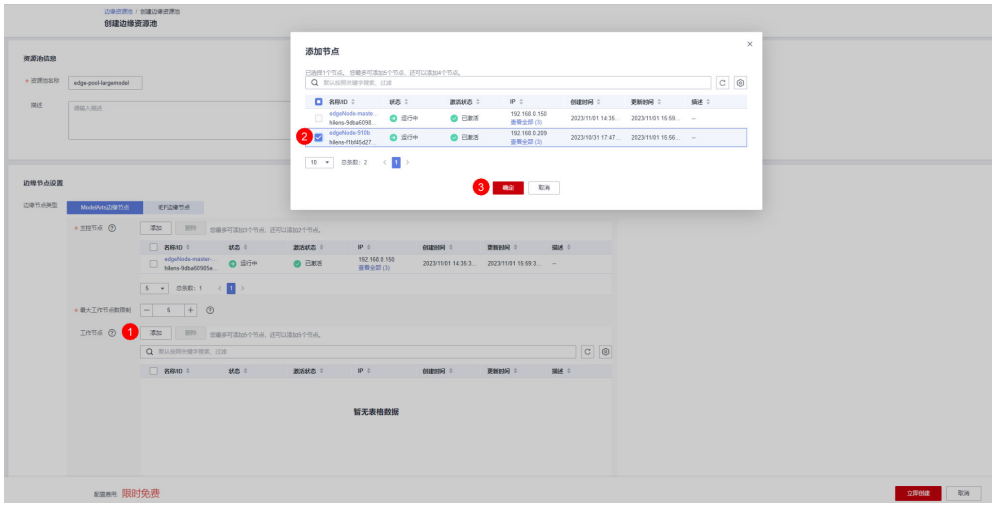
6. 配置hda.conf配置文件信息（可选）
   a. 登录nfs服务节点，执行如下命令：
      ```
      vi /etc/hilens/hda.conf
      ```
   b. 增加如下配置：
      ```
      hilens.nfs.server.ip=192.168.0.150
      hilens.nfs.mount.dir=/home/mind/model
      hilens.nfs.source.dir=/var/docker/hilens
      ```
      其中，**server.ip**是nfs存储节点内网ip，**mount.dir**是大模型默认挂载路径，**source.dir**是大模型下载路径。
   c. 配置完成后，执行如下命令重启固件：
      ```
      systemctl restart hdad
      ```
7. 进入ModelArts服务，选择所需空间。进入“边缘资源池 > 节点”，在当前设备节点操作列单击“激活”，节点状态将从“未激活”转为“已激活”。



8. 进入“边缘资源池 > 资源池”，单击“创建”。填写资源池名称，选择“ModelArts边缘节点”，在“主控节点”处单击“添加”，选择要添加的主控节点，单击“确定”。



9. 在“工作节点”处单击“添加”，选择要添加的工作节点，单击“确定”。



10. 单击“立即创建”，可在资源池列表中查看节点的状态。如果状态为“运行中”，则创建成功。
11. 在主控节点执行如下k8s命令，验证边缘池创建结果：
    a. 执行如下命令建立软连接。
    ```
    ln -s /home/k3s/k3s /usr/bin/kubectl
    ```
    b. 执行如下命令查看节点状态。
    ```
    kubectl get node -o wide
    ```
    c. 如果所有节点状态STATUS为“Ready”，则说明集群创建成功。

```
[root@ecs-edge-30037210 ~]# kubectl get node -o wide
NAME                      STATUS   ROLES                  AGE    VERSION         INTERNAL-IP      EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION
                CONTAINER-RUNTIME
hilens-9dba60985e5444629958f4a5cc07c1b2   Ready    control-plane,master   7m58s   v1.21.12+k3s1   192.168.0.150   <none>        EulerOS 2.0 (SP9)       4.19.90-vhulk2103.1.0.h1060.eul
erosv2r9.aarch64   docker://20.10.9
hilens-f1bf45d27c464413b6b53fa40d9a1c0d   Ready    <none>                 7m46s   v1.21.12+k3s1   192.168.0.209   <none>        EulerOS 2.0 (SP10)      4.19.90-vhulk2211.3.0.h1543.eul
erosv2r10.aarch64   docker://20.10.9
[root@ecs-edge-30037210 ~]#
```

8.2.5 安装 Ascend 插件

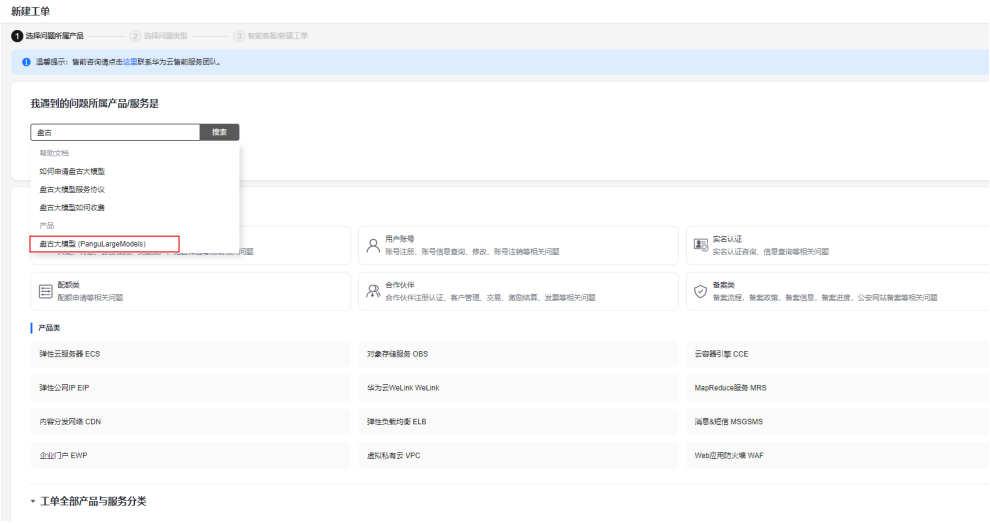
详情请参考官方文档：**https://www.hiascend.com/document/detail/zh/mindx-dl/50rc1/dluserguide/clusterscheduling/dlug_scheduling_02_000001.html**

8.2.6 订购盘古边缘部署服务

1. 登录盘古大模型套件平台，在服务“总览”页面，单击“立即购买”，平台将为您提交购买权限申请。如您有加急购买需求，可在页面右上角单击“工单 > 新建工单”，搜索“盘古大模型”产品，选择问题类型并提交工单。

图 8-5 立即购买

图 8-6 新建工单



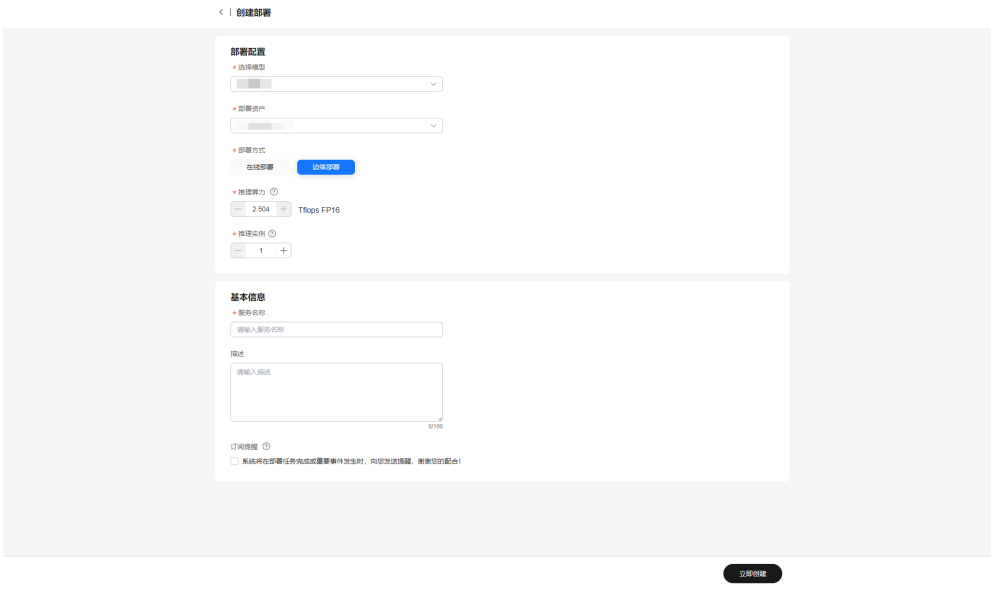
2. 获取购买权限后，根据需要选择计费模式，基模型需选择“N2 - 基础模型功能 & 应用增强功能”。用户可根据需求自行选择功能模型，输入资源名称，类型选择“边缘部署”，输入需要订购的推理算力，单击“确认订单”。

3. 订购完成后，进入“平台管理 > 资产管理 > 模型推理资产”，可查看订购的边缘部署资产。

8.2.7 部署边缘模型

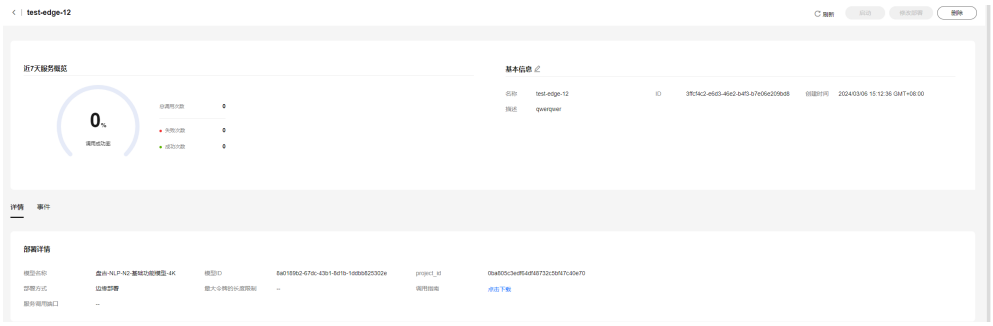
1. 进入盘古大模型套件平台，进入“模型开发 > 模型部署 > 边缘部署”，单击右上角“部署”按钮。

2. 在创建部署页面选择模型与部署资产，选择部署方式为边缘部署，输入推理实例数（根据边缘资源池的实际资源选择），输入服务名称，单击“立即创建”。

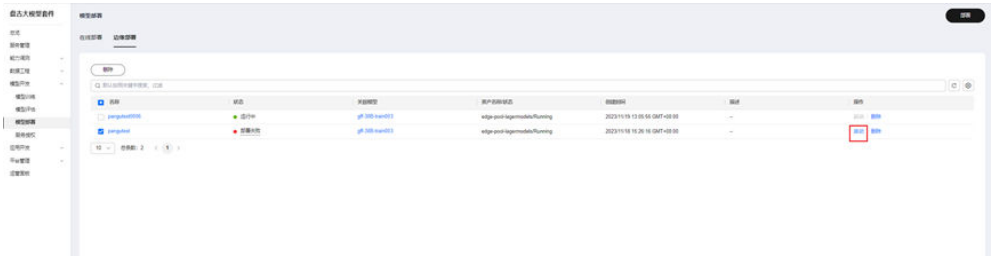


3. 创建成功后，可在“模型部署 > 边缘部署”，查看边缘部署列表。

4. 单击“服务名称”可进入服务详情界面。



5. 如果服务部署状态为“部署失败”，可单击服务操作列的“启动”按钮，重新部署。



8.2.8 调用边缘模型

调用边缘模型的步骤与使用“在线部署”调用模型的步骤相同，具体步骤请参考**使用API调用模型**。

9 调用盘古大模型

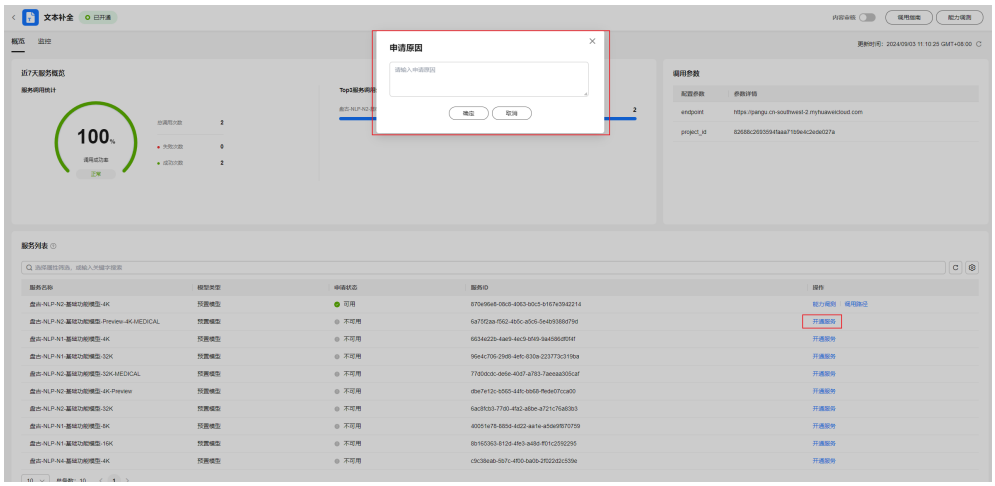
9.1 开通盘古大模型服务

调用模型之前，需要先开通盘古大模型服务。

1. 登录**盘古大模型套件平台**。
2. 在左侧导航栏中选择“服务管理”，在相应服务的操作列单击“查看详情”，可在服务列表中申请需要开通的服务。
   - 文本补全：提供单轮文本能力，常用于文本生成、文本摘要、闭卷问答等任务。
   - 多轮对话：提供多轮文本能力，常用于多轮对话、聊天任务。

图 9-1 服务管理

图 9-2 申请开通服务



3. 您可按照需要选择是否开启内容审核。

   开启内容审核后，可以有效拦截大模型输入输出的有害信息，保障模型调用安全，推荐进行开启。

图 9-3 大模型内容审核

说明

- 盘古大模型支持通过对接内容审核，实现拦截大模型输入、输出的有害信息，保障模型调用安全。用户可依据需求选择是否开通、启用内容审核。
- 推荐用户购买内容审核套餐包，购买内容审核套餐包时，需要选择“文本内容审核”套餐。
- 如果未启用内容审核服务，可以在开通服务之后，查看服务详情，在详情界面右上角开通内容审核。

9.2 使用“能力调测”调用模型

前提条件

使用能力调测调用模型之前，需要先**开通盘古大模型服务**。

使用“能力调测”调用模型

能力调测通过图形化问答界面，提供了快速访问盘古大模型能力的入口。用户可以通过能力调测调用基模型与训练后的模型。

说明

训练后的模型需要“在线部署”且状态为“运行中”时，才可以使用本章节提供的方法进行调测。

- 文本补全：给定一个提示和一些参数，模型会根据这些信息生成一个或多个预测的补全。例如，让模型依据要求写邮件、做摘要总结、生成观点见解等。
- 多轮对话：基于对话问答功能，用户可以与模型进行自然而流畅的对话和交流。

图 9-4 使用能力调测

表 9-1 能力调测参数说明

参数	说明
温度	用于控制生成文本的多样性和创造力。
核采样	控制生成文本多样性和质量。
最大口令限制	用于控制聊天回复的长度和质量。
话题重复度配置	用于控制生成文本中的重复程度。
词汇重复度控制	用于调整模型对频繁出现的Token的处理方式。
历史对话保留轮数	选择“多轮对话”功能时具备此参数，表示系统能够记忆的历史对话数。

9.3 使用 API 调用模型

用户可以通过API调用盘古大模型服务提供的基模型以及用户训练后的模型。训练后的模型需使用“在线部署”，才可以使用本章节提供的方法进行调用。本章节分别介绍使用Postman调用API和多语言（Java/Python/Go）调用API的方法，仅供测试使用。

前提条件

使用API调用模型前，需要先**开通盘古大模型服务**。

使用 Postman 调用 API

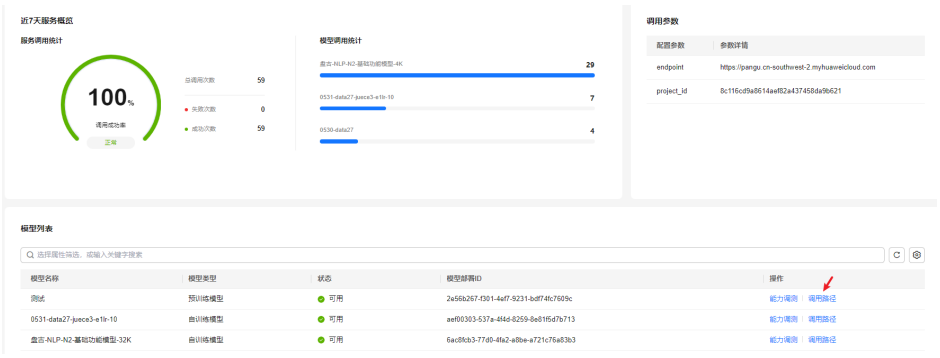
1. 获取API请求地址。
   a. 在“服务管理”页面，单击所需API的“查看详情”按钮。

   图 9-5 服务管理

   b. 在“模型列表”中选择需要调用的模型，单击操作栏中的“调用路径”，复制对应模型的API请求地址。

   图 9-6 获取 API 请求地址

2. 获取Token。

   在调用盘古API过程中，Token起到了身份验证和权限管理的作用。

   在调用盘古API前，需要先使用“获取Token”接口，获取Token值，再将Token值传入盘古API的请求header参数中，实现盘古服务在接收到用户的API请求时进行身份验证。

   说明

   关于Token有效期的详细说明请参见**获取IAM用户Token（使用密码）**。

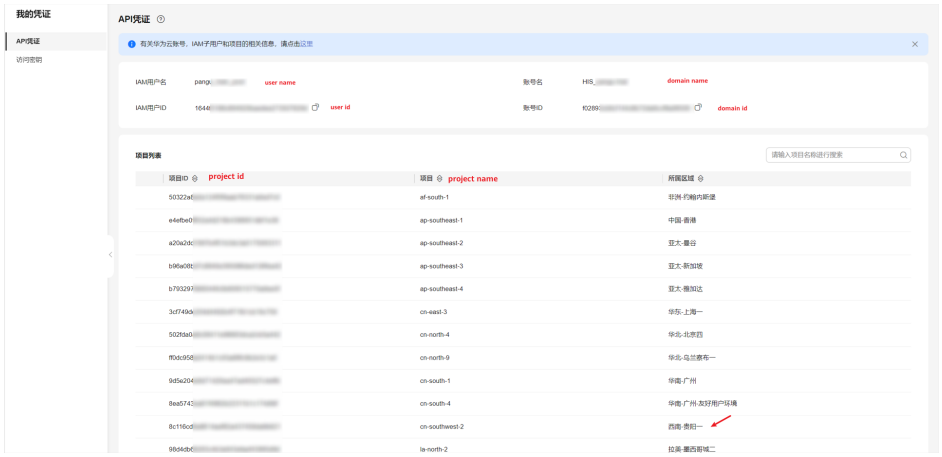
   获取token步骤如下：

   a. 登录**“我的凭证 > API凭证”**页面，获取user name、domain name、project id。

   project id参数需要与盘古服务部署区域一致。例如，盘古大模型部署在“西南-贵阳一”区域，需要获取与“西南-贵阳一”区域对应的project id。

图 9-7 查看盘古服务区域

图 9-8 获取 user name、domain name、project id

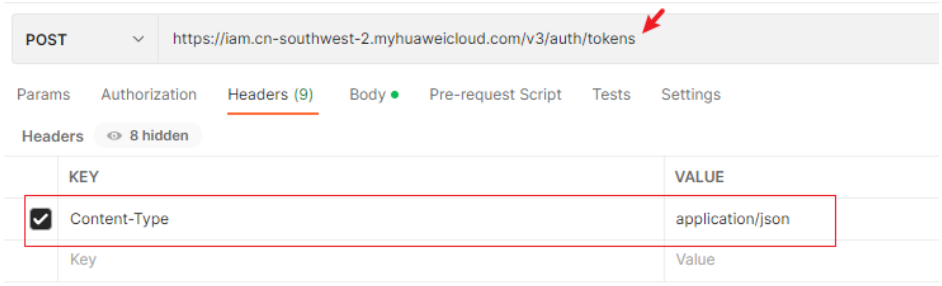


b. 下载并安装**Postman**调测工具。

c. 打开Postman，新建一个POST请求，并输入“西南-贵阳一”区域的“获取Token”接口。并填写请求Header参数。

- 接口地址为：https://iam.cn-southwest-2.myhuaweicloud.com/v3/auth/tokens
- 请求Header参数名为Content-Type，参数值为application/json

图 9-9 填写获取 Token 接口

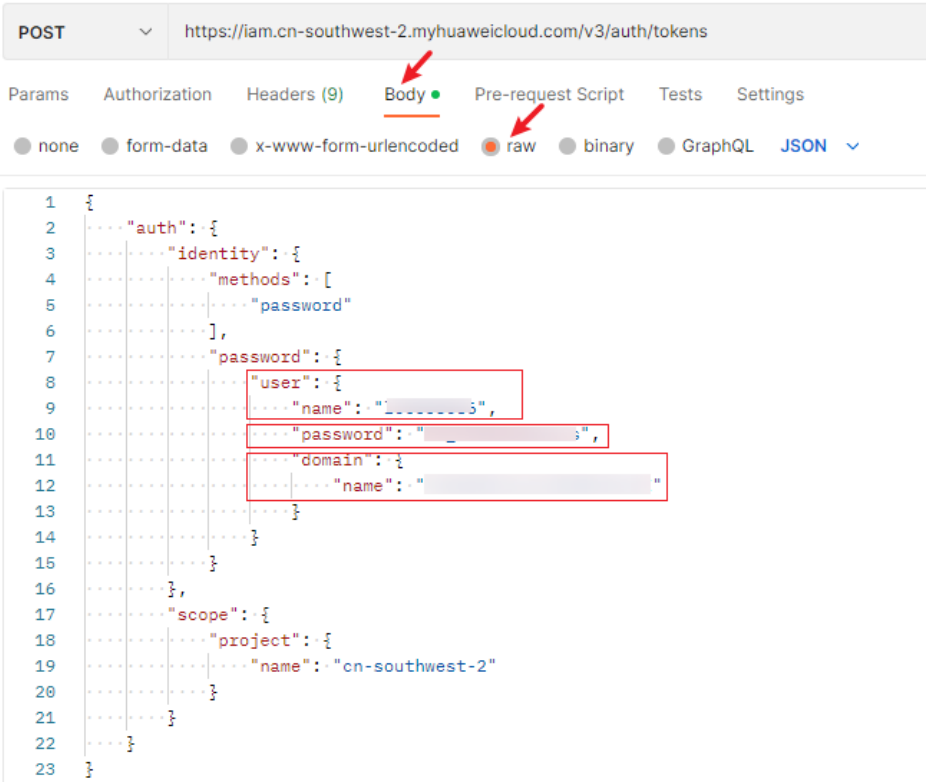


d. 填写“获取token”接口的请求体。在Postman中选择“Body > raw”选项，参考**图9-10**复制并填入以下代码，并填写user name、domain name、password。

```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
```

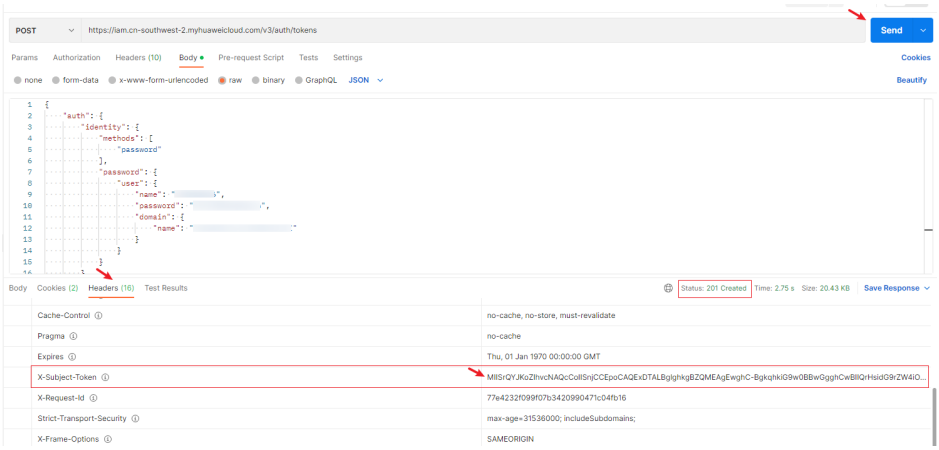
```
        "password": {
            "user": {
                "name": "username", //IAM用户名
                "password": "********", //华为云账号密码
                "domain": {
                    "name": "domainname" //账号名
                }
            }
        }
    },
    "scope": {
        "project": {
            "name": "cn-southwest-2" //盘古大模型当前部署在“西南-贵阳一”区域，取值为cn-southwest-2
        }
    }
  }
}
```

图 9-10 填写请求 Body



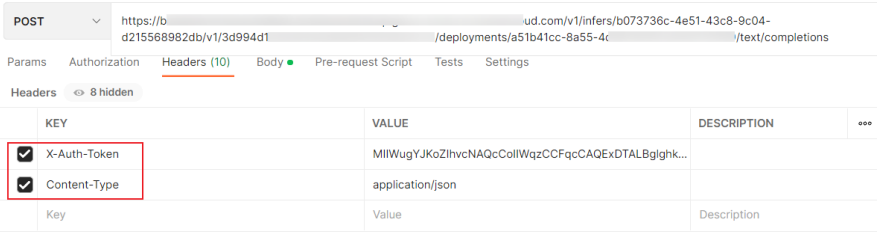
e. 单击Postman界面“Send”按钮，发送请求。当接口返回状态为201时，表示Token接口调用成功，此时单击“Headers”选项，找到并复制“X-Subject-Token”参数对应的值，该值即为需要获取的Token。

图 9-11 获取 Token



3. 调用盘古API。
   a. 在Postman中新建POST请求，并填入盘古API请求地址。
   b. 参考图9-12填写2个请求Header参数。
      - 参数名为Content-Type，参数值为application/json。
      - 参数名为X-Auth-Token，参数值为**获取Token**中获取的Token值。

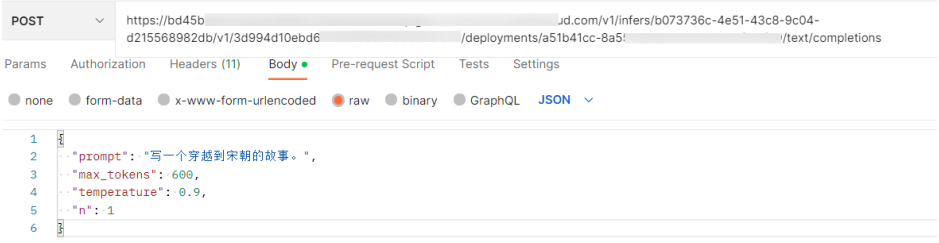
      图 9-12 填写盘古 API



   c. 在Postman中选择“Body > raw”选项，参考图9-13复制并填入以下代码，填写请求Body。

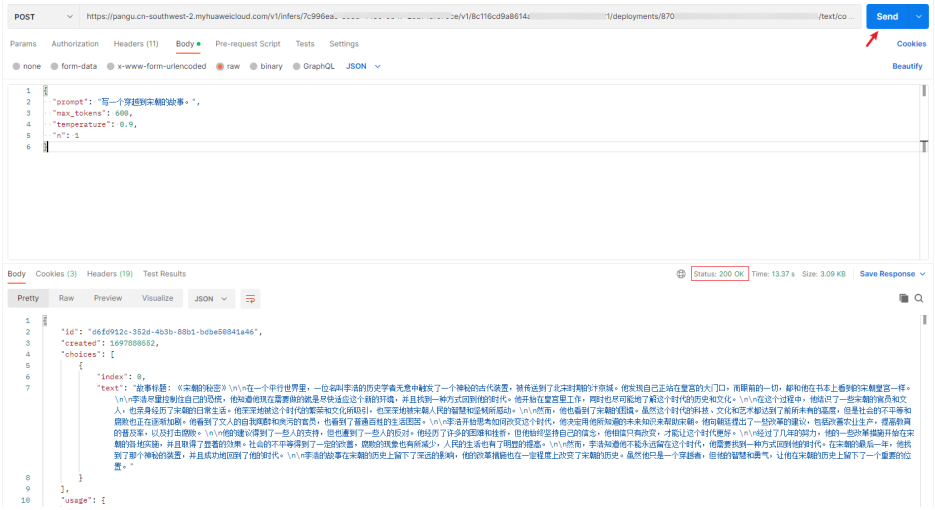
```
{
 "prompt": "写一个穿越到宋朝的故事。",
 "max_tokens": 600,
 "temperature": 0.9,
 "n": 1
}
```

图 9-13 填写盘古请求 Body



   d. 单击Postman界面的“Send”按钮，发送请求。当接口返回状态为200时，表示盘古API调用成功，并可在Postman中看到接口的返回信息。

图 9-14 获取盘古 API 调用结果



说明

- 使用Postman调用API时，如果出现SSL证书无效相关的报错，如“self signed certificate”（自签名证书）、“certificate has expired”（证书已过期）或“unable to verify the first certificate”（无法验证第一个证书）等。可以在Postman的设置中关闭“SSL certificate verification”选项。
- 关于盘古大模型API的详细请求参数、响应参数介绍请参见《API参考》文档。

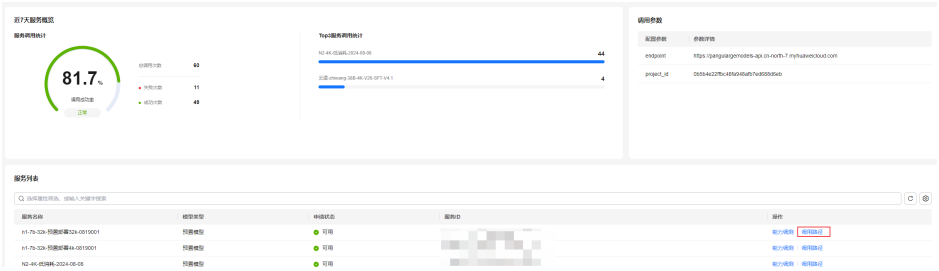
使用多语言（Java/Python/Go）调用 API

1. 获取API请求地址。
   a. 在“服务管理”页面，单击所需API的“查看详情”按钮。

图 9-15 服务管理

   b. 在“服务列表”中选择需要调用的模型，单击操作栏中的“调用路径”，复制对应模型的API请求地址。

图 9-16 获取 API 请求地址



2. 获取AK/SK。

   在使用盘古SDK时，需要使用AK/SK进行身份验证。

   a. 登录“**我的凭证 > 访问密钥**”页面，单击“新增访问密钥”。

   图 9-17 新增访问密钥

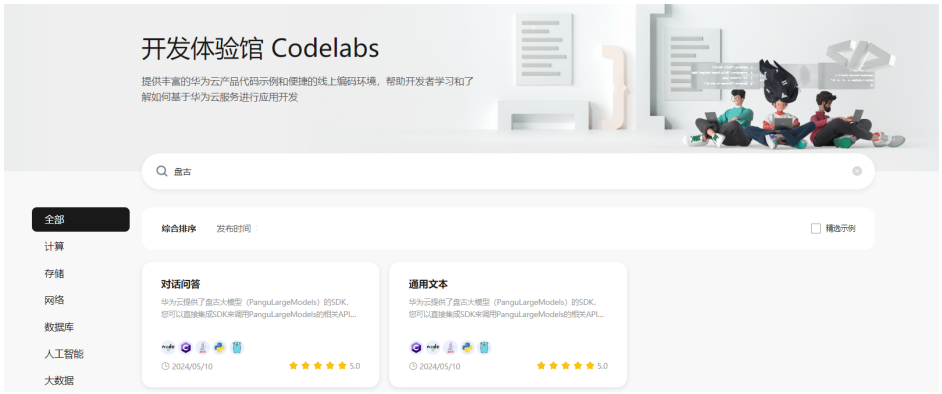
   

   b. 依据页面提示完成AK、SK的创建，并下载credentials.csv文件，Access Key Id即为AK，Secret Access Key即为SK，AK和SK需要妥善保存，避免泄露导致安全风险，如果不慎丢失，需要及时删除，并重新生成。

3. 使用SDK调用盘古API。

   a. 进入“**开发体验馆 Codelabs**”页面，查看盘古API场景示例如下。

   图 9-18 开发体验馆

   b. 选择要调用的场景示例，按照使用参考完成前置条件，可选择不同语言SDK，**图9-19**以Java为例。

   参数说明如下：

   ak，sk即为**2**中获取到的AK/SK，建议加密使用，避免引起安全问题。

   其他参数可从**获取API请求地址**中获取。获取到的请求地址结构如下：

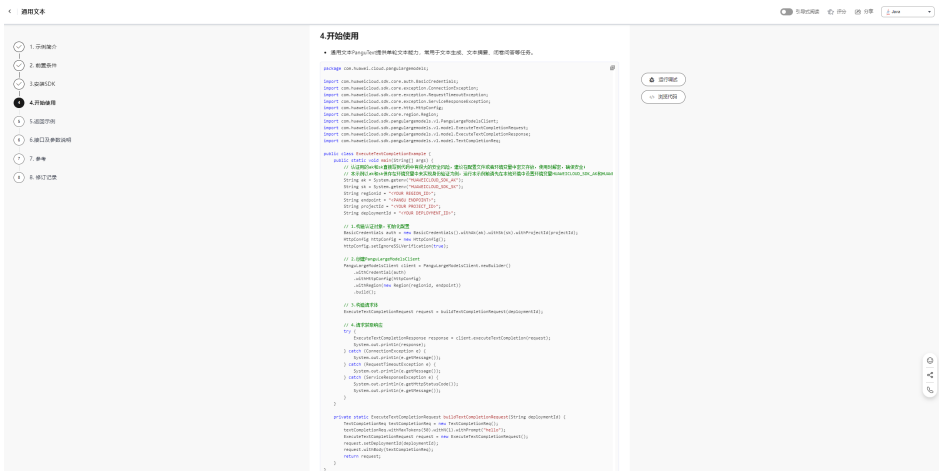
   https://pangu.{region_id}.myhuaweicloud.com/v1/{project_id}/deployments/{deployment_id}/chat/completions

   - regionId为{region_id}
   - endpoint为https://pangu.cn-east-3.myhuaweicloud.com
   - projectId为{project_id}
   - deploymentId为{deployment_id}

图 9-19 JavaSDK

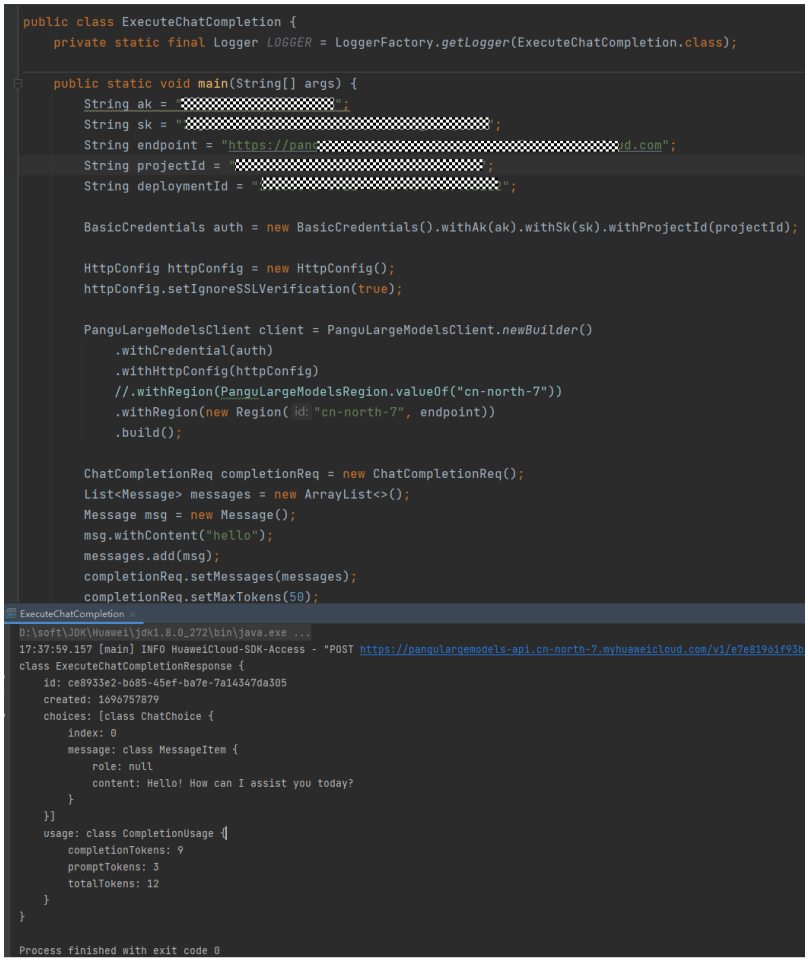
c. 利用SDK编写代码调用API。

图 9-20 调用 API



d. 结果响应如图9-21所示。

图 9-21 结果响应



9.4 启用模型内容审核

- 内容审核是文本的检测技术，可自动检测涉黄、涉暴、违规等内容，对用户向模型输入的内容、模型输出内容进行内容审核，帮助客户降低业务违规风险。
- 授权使用华为云内容审核，有效拦截大模型输入输出的有害信息，保障模型调用安全。
- 授权后，在调用盘古大模型能力时，模型的输入和输出将分别调用一次内容审核服务，该服务为付费项，用户可**按需购买**。
- 若不使用，您也可以自行对接第三方内容审核服务。关于大模型生成内容的责任主体，请参考**《盘古大模型服务协议》**。

启用内容审核服务

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“服务管理”，单击“开启内容审核”，进行授权。

图 9-22 内容审核授权

3. 购买内容审核套餐包，使用“文本补全”、“多轮对话”功能时需要购买**“文本内容审核”**套餐包。

图 9-23 购买内容审核套餐包

9.5 统计模型调用量

模型调用成功后，有两种方式可以查看模型的调用量。

- 通过“服务管理”功能查看调用量：查看具体某个模型的调用总量、调用成功量、调用失败量，且可按时间进行筛选。
- 通过“运营面板”功能查看调用量：查看全部模型访问总数、模型回复时的响应时长、兜底回复比例以及输入/输出token信息。

通过“服务管理”功能查看调用量

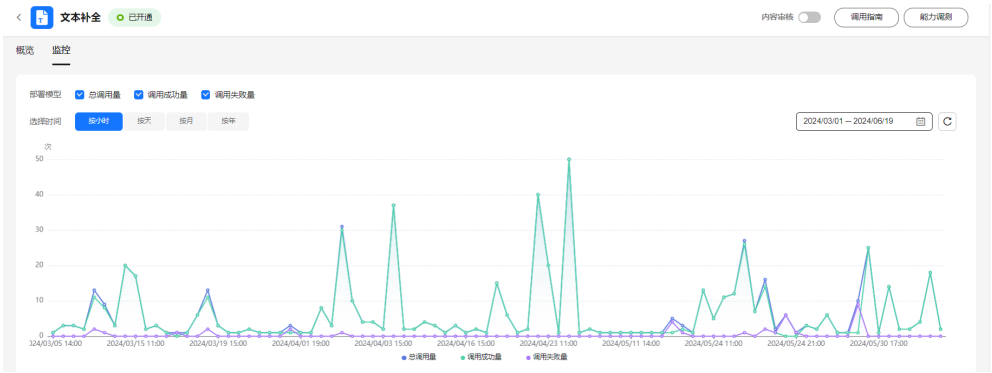
1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“服务管理”，选择所需要查看的服务，单击操作列“查看详情”。

图 9-24 查看详情

3. 在服务详情页面，在“概览”页签，可以查看调用量的概览信息，在“监控”页签，可以查看下详细的调用总量、调用成功量与调用失败量。

图 9-25 查看调用量概览信息

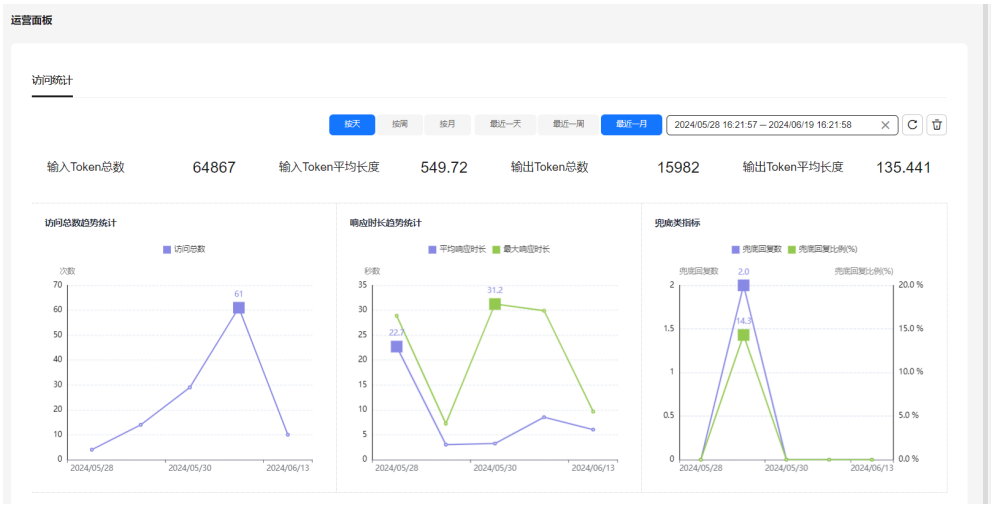
图 9-26 查看调用量详情



通过“运营面板”功能查看调用量

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“运营面板”，通过运营面板查看模型访问总数、模型回复时的响应时长、兜底回复比例与输入/输出token信息。

图 9-27 运营面板

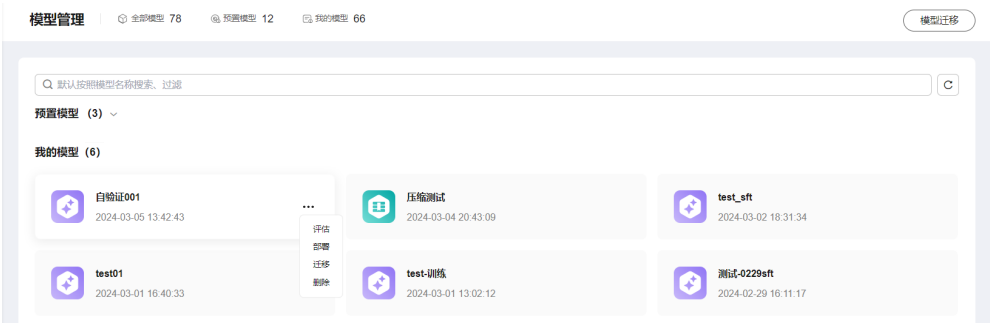


10 迁移盘古大模型

模型训练完成后，可以通过迁移（导入模型、导出模型）功能将本局点训练的模型导出，或将其他局点训练的模型导入本局点进行使用。

支持迁移操作的模型可以在“模型开发 > 模型管理 > 我的模型”中查看。

图 10-1 模型管理



导入/导出模型

以从环境A迁移模型到环境B为例：

1. 登录环境B的盘古大模型套件平台，在“模型开发 > 模型管理”页面，单击右上角的“模型迁移”。
2. 在“模型迁移”页面，下载用户证书。

图 10-2 下载用户证书

3. 登录环境A的盘古大模型套件平台，在“模型迁移”页面，选择“导出模型”，在导入证书的地方上传环境B下载的证书文件，并选择需要导出的模型和模型导出的obs路径。

图 10-3 导出模型

4. 单击“确定”，导出模型。

   模型导出成功后，可以在obs中查看导出后的模型文件。下载该obs文件，上传到环境B对应的obs桶中。

5. 登录环境B的盘古大模型套件平台，在“模型迁移”页面，选择“导入模型”，输入模型对应的obs地址和模型名称后，单击“确定”，启动导入模型任务。

图 10-4 导入模型

11 平台资源管理

11.1 管理模型资产、推理资产

查看模型资产与模型推理资产

用户购买盘古大模型套件后，可以在“平台管理 > 资产管理”中查看购买的模型资产和模型推理资产。

图 11-1 查看模型资产

图 11-2 查看模型推理资产

续订模型推理资产

模型推理资产到期后，可以进行续订操作。

在“平台管理 > 资产管理 > 模型推理资产”中单击操作列“续订”执行续订操作。

图 11-3 续订模型推理资产

扩容模型推理资产

推理资产不足，现有资源无法满足同时部署多个模型时，可以扩容模型推理资产。

在“平台管理 > 资产管理 > 模型推理资产”中，单击操作列“扩容”执行扩容操作。

图 11-4 扩容模型推理资产

不同类型的模型在部署时，做占用的推理资产数量存在差异，部署模型时所占的推理资产数量与模型类型关系如下。

表 11-1 部署模型

模型类型	推理资产占有数量
盘古-NLP-N1 系列模型	部署1实例占用0.125个推理单元。
盘古-NLP-N2 系列模型	部署1实例占用0.5个推理单元。
盘古-NLP-N4 系列模型	部署1实例占用1个推理单元。

11.2 获取 token 消耗规则

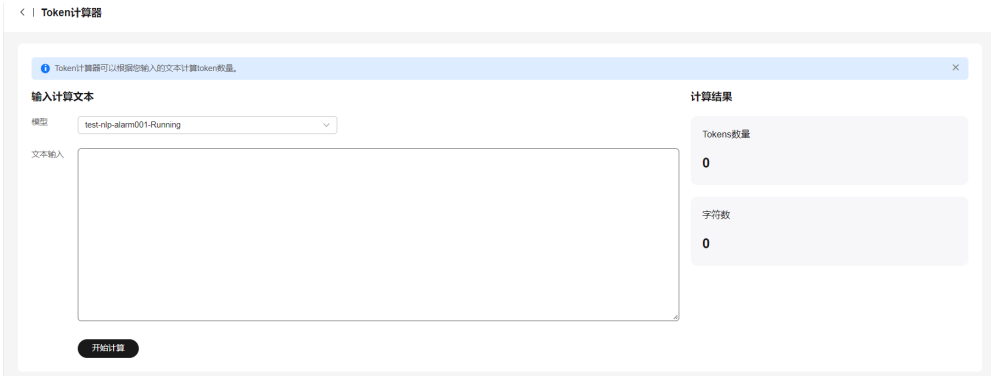
了解Token消耗规则对于模型训练至关重要。掌握从字符到Token的转换规律可以优化数据预处理过程并有效控制训练成本。每个Token代表模型处理和生成文本的基本单位，它可以是一个单词、字符或字符的片段。模型的输入和输出都会被转换成Tokens，并根据模型的概率分布进行采样或计算。了解Token的消耗和费用计算规则，用户可以更准确地预测和控制训练成本，提高预算管理效率。训练服务的费用按实际消耗的Token数量计算，即实际消耗的Token数量乘以Token的单价。

为了帮助用户更好地管理和优化Token消耗，平台提供了Token计算器工具。Token计算器可以帮助用户在模型训练前评估文本的Token数量，提供费用预估，并优化数据预处理策略。

使用Token计算器的步骤如下：

1. 登录**盘古大模型套件平台**。
2. 在“服务管理”页面，单击页面右上角“Token计算器”。
3. 在Token计算器中选择所需的模型，并输入文本内容后，单击“开始计算”即可统计输入文本的Token数量。

图 11-5 Token 计算器



说明

预置模型和已经部署的模型可以使用Token计算器。

12 提示词工程

12.1 什么是提示词工程

什么是提示词工程

提示工程是一个较新的学科，应用于开发和优化提示词（Prompt），帮助用户有效地将语言模型用于各种应用场景和研究领域。掌握提示词工程相关技能将有助于用户了解大型语言模型的能力和局限性。

提示工程不仅涉及设计和研发提示词，还包括与大型语言模型的交互和研发中的各种技能和技术。它在实现和对接大型语言模型、理解其能力方面扮演着关键角色。用户可以通过提示工程提高语言模型的安全性，也可以通过专业领域知识和外部工具赋能语言模型，增强其能力。

Prompt 基本要素

您可以通过简单的提示词获得大量结果，但结果的质量与您提供的信息数量和完善度有关。一个提示词可以包含您传递到模型的指令或问题等信息，也可以包含其他种类的信息，如上下文、输入或示例等。您可以通过这些元素来更好地指导模型，并因此获得更好的结果。提示词主要包含以下要素：

- **指令**：想要模型执行的特定任务或指令。如总结、提取、生成等。
- **上下文**：包含外部信息或额外的上下文信息，引导语言模型更好地响应。
- **输入数据**：用户输入的内容或问题。
- **输出指示**：指定输出的类型或格式。

提示词所需的格式取决于您想要语言模型完成的任务类型，以上要素并非都是必须的。

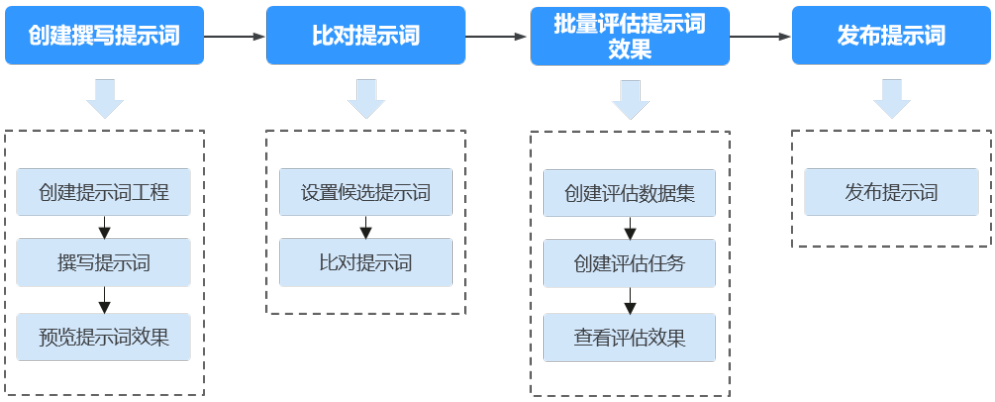
提示词工程使用流程

盘古大模型套件平台可以辅助用户进行提示词设计、调优、比较和对提示词通用性进行自动评估等功能，并对调优得到的提示词进行保存和管理。

表 12-1 功能说明

功能	说明
提示用例管理	提示用例集用于维护多组提示词变量的信息，可以用于提示词的调优、比较和评估。 支持对用例集的创建、查询、修改、删除。
提示词工程任务管理	提示词工程平台以提示词工程任务为管理维度，一个任务代表一个场景或一个调优需求，在提示词工程任务下可以进行提示词的调优、比较和评估。 提示词工程任务管理支持工程任务的创建、查询、修改、删除。
提示词调优	提示词调优支持对提示词文本的编辑、提示词变量设置、提示词结果生成和调优历史记录管理。
提示词候选	提示词候选支持用户对调优后初步筛选的提示词进行候选管理，每个工程任务下可以保存上限9个候选提示词，进一步基于候选提示词进行比较和评估。
提示词比较	提示词比较支持选择两个候选提示词对其文本和参数进行比较，支持对选择的候选提示词设置相同变量值查看效果。
提示词评估	提示词评估以任务维度管理，支持评估任务的创建、查询、修改、删除。支持创建评估任务，选择候选提示词和需要使用的变量数据集，设置评估算法，执行任务自动化对候选提示词生成结果和结果评估。
提示词管理	提示词管理支持用户对满意的候选提示词进行保存管理，同时支持提示词的查询、删除。

图 12-1 提示词工程使用流程



12.2 获取提示词模板

平台提供了多种任务场景的提示词模板，可以帮助用户更好地利用大模型的能力，引导模型生成更准确且更具针对性的输出，从而提高模型在特定任务上的性能。在创建提示词工程前，可以先使用预置的提示词模板，或基于提示词模板进行改造，如果提示词模板满足不了使用需求，可再单独创建。

提示词模板可以在平台“应用开发 > 提示词管理 > 预置提示词”中获取。

图 12-2 获取提示词模板

12.3 撰写提示词

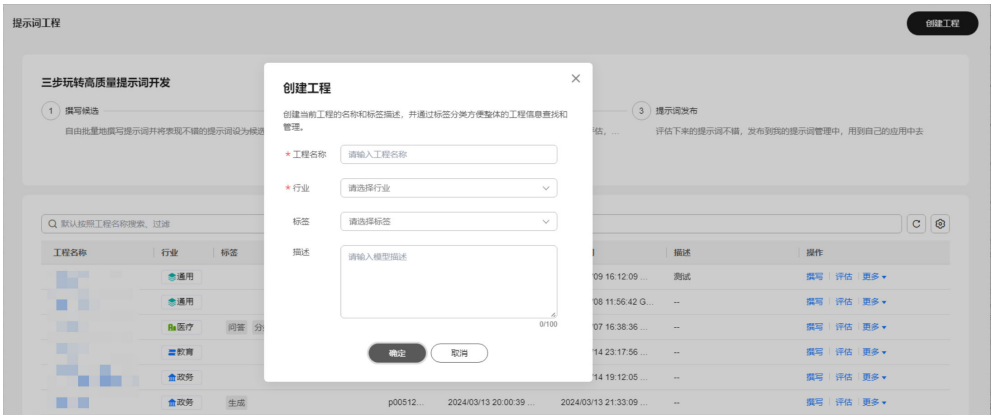
12.3.1 创建提示词工程

通过精心设计和优化提示词，可以引导大模型生成用户期望的输出，提示词工程任务的目标是通过设计和实施一系列的实验，来探索如何利用提示词来提高大模型在各种任务上的表现。

撰写提示词前需要先创建提示词工程，用于对提示词的统一管理。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 提示词工程”，进入提示词工程页面。
3. 单击页面右上角“创建工程”，进入工程任务创建弹窗。输入工程名称、描述，选择行业、标签，工程任务下的所有提示词会同步继承该标签。

图 12-3 创建提示词工程



4. 单击“确定”完成工程创建。

12.3.2 撰写提示词

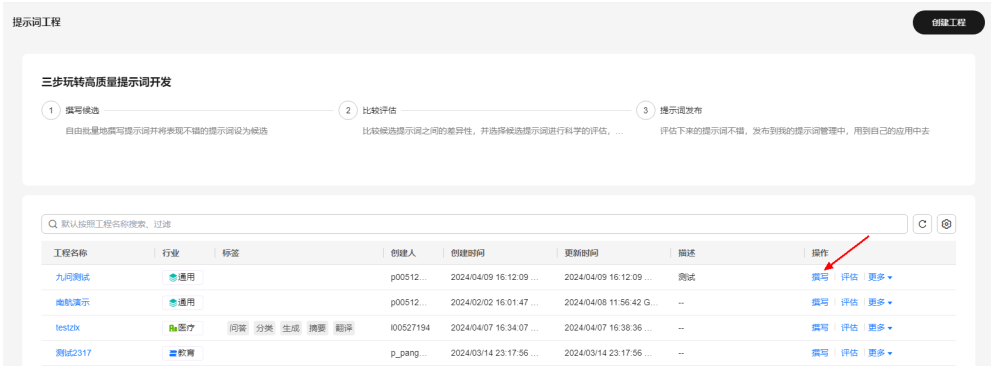
提示词是用来引导模型生成的一段文本。撰写的提示词应该包含任务或领域的关键信息，如主题、风格和格式等。

撰写提示词时，可以设置提示词变量，即在提示词中通过添加占位符{{ }}标识，表示一些动态的信息，让模型根据不同的情况生成不同的文本，增加模型的灵活性和适应性。例如，将提示词设置为“你是一个旅游助手，需要给用户介绍旅行地的风土人情。请介绍一下{{location}}的风土人情。”在评估提示词效果时，可以通过批量替换{{location}}的值，来获得模型回答，提升评测效率。

同时，撰写提示词过程中，可以通过设置模型参数控制模型生成行为，如调整温度、核采样和最大口令限制等。模型参数的设置会影响模型的生成质量和多样性，因此需要根据不同的场景进行选择。提示词的撰写步骤如下：

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 提示词工程”，进入提示词工程页面。
3. 在工程任务列表页面，找到所需要操作的工程任务，单击该工程任务操作栏中的“撰写”。

图 12-4 提示词工程



4. 在撰写提示词区域输入提示词文本，可以插入若干个变量，通过占位符{{ }}标识，单击“确认”按钮，平台会自动识别插入的变量。

图 12-5 撰写提示词

图 12-6 确认提示词内容

5. 识别的变量展示在变量定义区域，可以编辑变量名称便于理解。

图 12-7 查看提示词变量

说明

变量定义区域展示的是整个工程任务下定义的变量信息，候选提示词中关联的变量也会进行展示，候选词相关操作请参见**设置候选提示词**。

6. 在模型区域单击“设置”，设置提示词输入的模型和模型参数。

图 12-8 设置模型

说明

同一个提示词工程中，定义的变量不能超过20个。

12.3.3 预览提示词效果

提示词撰写完成后，可以通过输入具体的变量值，组成完整的提示词，查看不同提示词在模型中的使用效果。

1. 在撰写提示词页面，找到页面右侧变量输入区域，在输入框中输入具体的变量值信息。

   输入变量值后预览区域会自动组装展示提示词。用户也可以直接选择已创建的变量集填入变量值信息，变量集是一个excel文件，每行数据是需要输入的变量值信息，可以通过“导入”功能进行上传。

图 12-9 预览提示词效果

2. 单击“查看效果”按钮，输出模型回复结果，用户可以根据预览效果调整提示词的文本和变量。

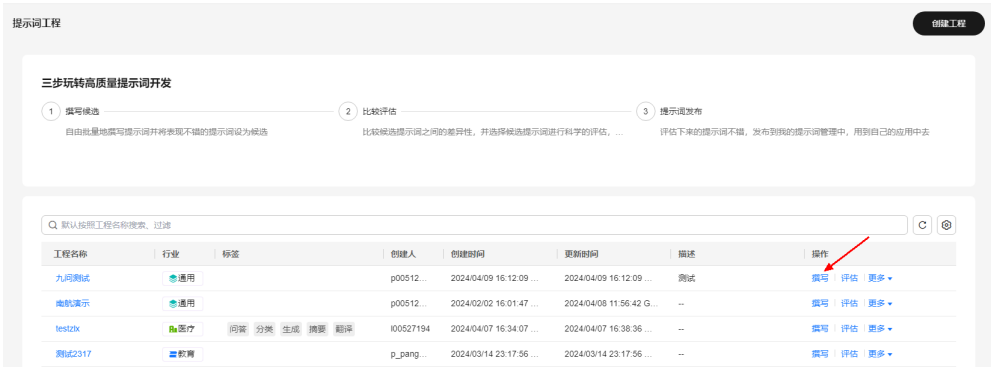
12.4 两两比对提示词效果

12.4.1 设置候选提示词

用户可以将效果较好的提示词设为候选提示词，并对提示词进行比对查看效果。

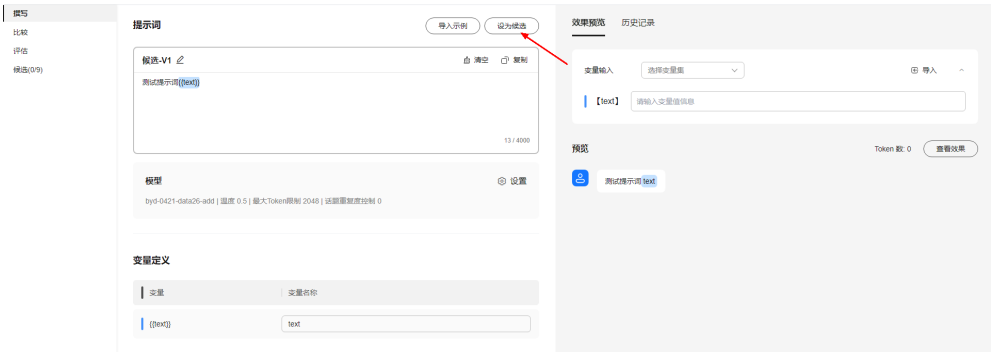
1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 提示词工程”，进入提示词工程页面。
3. 在工程任务列表页面，找到所需要操作的工程任务，单击该工程任务操作栏中的“撰写”。

图 12-10 撰写提示词



4. 在撰写提示词区域单击“设为候选”按钮，将当前撰写的提示词设置为候选提示词。

图 12-11 设为候选



说明

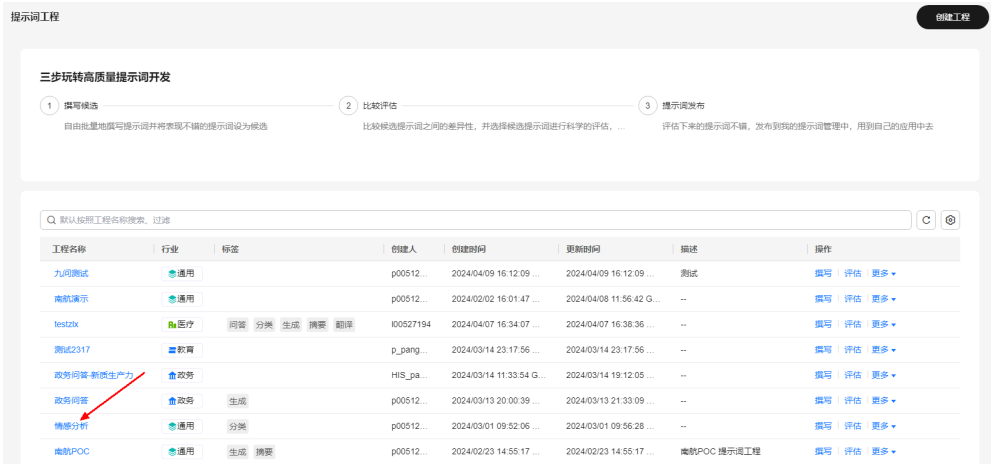
每个工程任务下候选提示词上限9个，达到上限9个时需要删除其他候选提示词才能继续添加。

12.4.2 两两比对提示词效果

将设置为候选的提示词两两比较，获取提示词的差异性和效果。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 提示词工程”，进入提示词工程页面。
3. 在工程任务列表页面，找到所需要操作的工程任务，单击该工程任务名称，跳转工程任务下候选提示词页面。

图 12-12 提示词工程



4. 选中两个候选提示词，单击左上角“横向比较”按钮，跳转提示词比较页面。

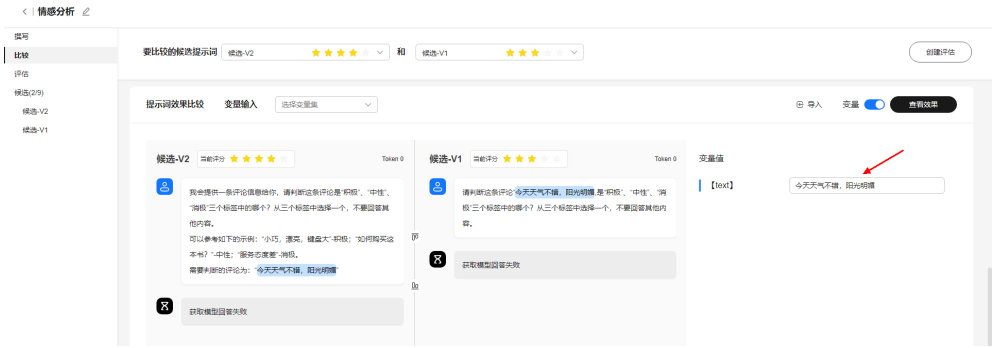
图 12-13 横向比较

5. 比较候选提示词信息的差异性，可以单击开启“高亮展示差异点”。

图 12-14 高亮差异点

6. 下拉页面至“提示词效果比较”模块，比较提示词的效果，输入相同的变量值，查看两个提示词生成的结果。

图 12-15 比较提示词的效果



12.5 批量评估提示词效果

12.5.1 创建提示词评估数据集

批量评估提示词效果前，需要先上传提示词变量数据文件用于创建对应的评估数据集。

提示词变量是一种可以在文本生成中动态替换的占位符，用于根据不同的场景或用户输入生成不同的内容。其中，变量名称可以是任意的文字，用于描述变量的含义或作用。

提示词评估数据集约束限制

- 上传文件限xlsx格式。
- 数据行数不小于10行，不大于50行。
- 数据不允许相同表头，表头数量小于20个。
- 数据单条文本长度不超过1000。

说明

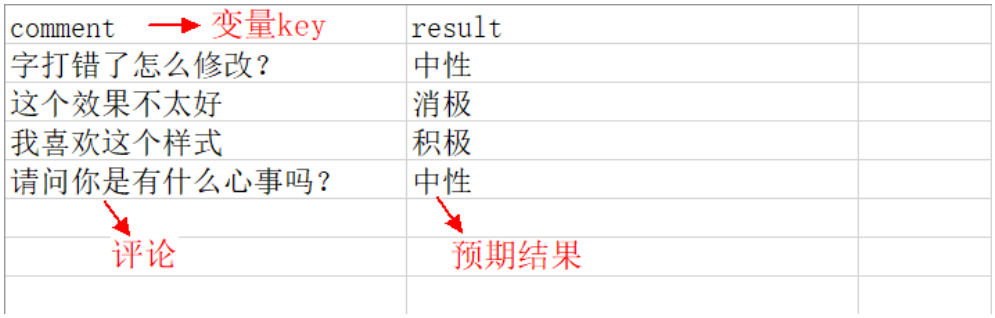
创建数据集时会对相关限制条件进行校验。

数据参考格式

图 12-16 数据参考格式

	A	B	C	D	E	F	G
1	key1	key2	key3	result	→ 表头为提示词变量key		
2	组1-v1	组1-v2	组1-v3	组1-r			
3	组2-v1	组2-v2	组2-v3	组2-r	→ 每行为1组变量值		
4	组3-v1	组3-v2	组3-v3	组3-r			
5							

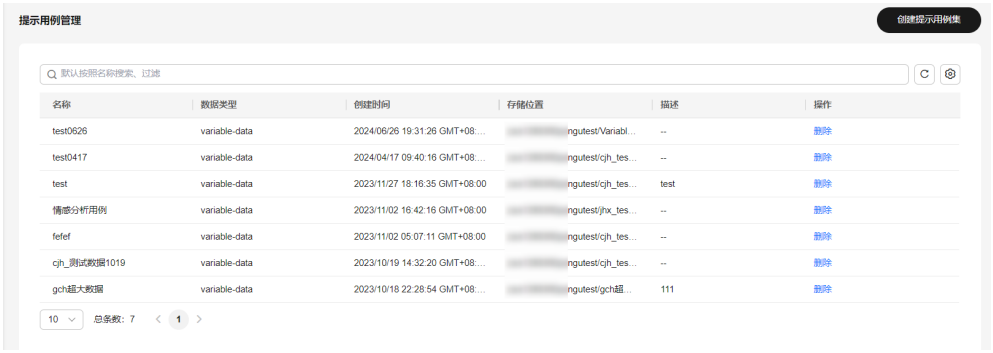
图 12-17 数据示例



创建提示词评估数据集

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“数据工程 > 提示用例管理”。

图 12-18 提示用例管理



3. 单击页面右上角“创建提示用例集”，进入创建弹窗。
4. 单击存储位置最右侧的图标，选择数据集文件所对应的obs路径，然后输入数据集名称、描述，创建数据集。

   创建数据集前，请先将数据上传至OBS。

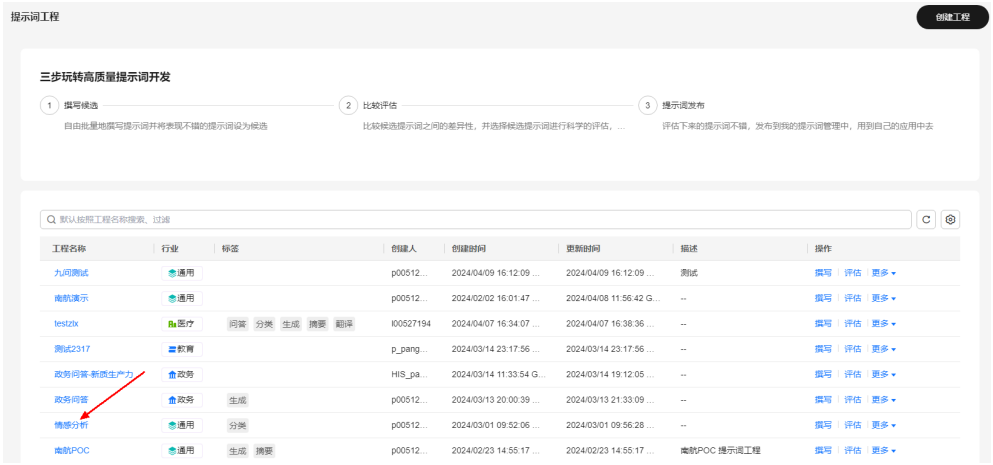
图 12-19 创建数据集

12.5.2 创建提示词评估任务

选择候选提示词进行批量自动化评估。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 提示词工程”，进入提示词工程页面。
3. 在工程任务列表页面，找到所需要操作的工程任务，单击该工程名称，跳转工程任务下候选提示词页面。

图 12-20 提示词工程

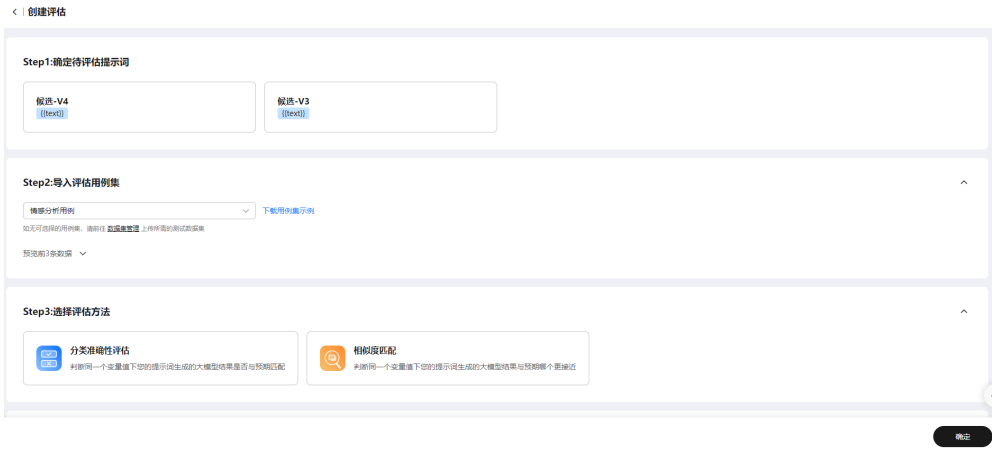


4. 选中需要评估的候选提示词，单击左上角“创建评估”按钮，跳转评估任务创建页面。

图 12-21 创建评估

5. 选择评估使用的变量数据集和评估方法。
   - 数据集：根据选择的数据集，将待评估的提示词和数据集中的变量自动组装成完整的提示词，输入模型生成结果。
   - 评估方法：根据选择的评估方法，对模型生成结果和预期结果进行比较，并根据算法给出相应的得分。

图 12-22 创建评估



6. 输入评估名称和描述。

图 12-23 输入评估名称

7. 单击右下角“确定”按钮，评估任务自动进入执行状态。

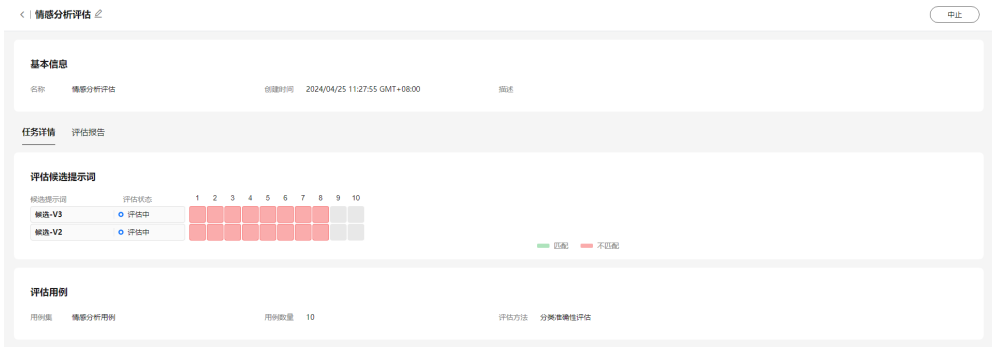
12.5.3 查看提示词评估结果

1. 评估任务创建完成后，会跳转至“评估”页面，在该页面可以查看评估状态。

图 12-24 查看评估状态

2. 单击评估名称，进入评估任务详情页，可以查看详细的评估进度。例如，在图12-25中有10条评估用例，当前已经评估了8条，剩余2条待评估。

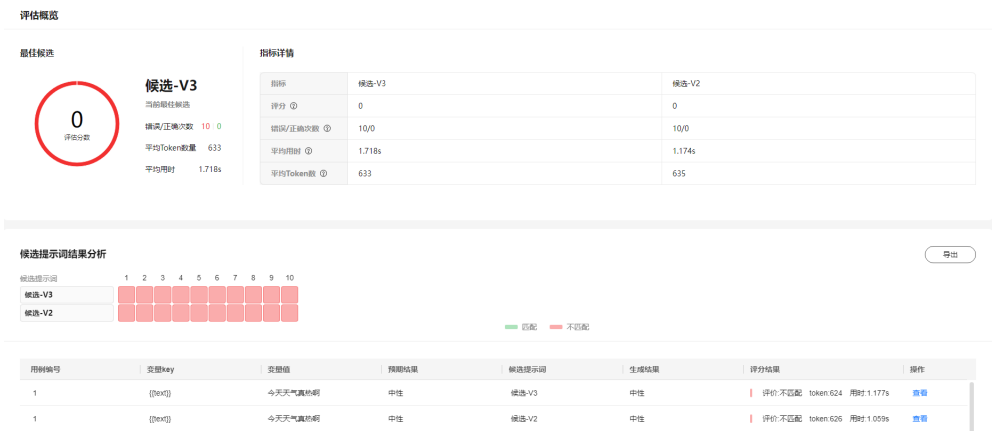
图 12-25 查看评估进展



3. 评估完成后，进入“评估报告”页面，可以查看每条数据的评估结果。

   在评估结果中，“预期结果”即为变量值（问题）所预设的期望回答，“生成结果”即模型回复的结果。通过比较“预期结果”与“生成结果”的差异可以判断提示词效果。

图 12-26 查看评估报告

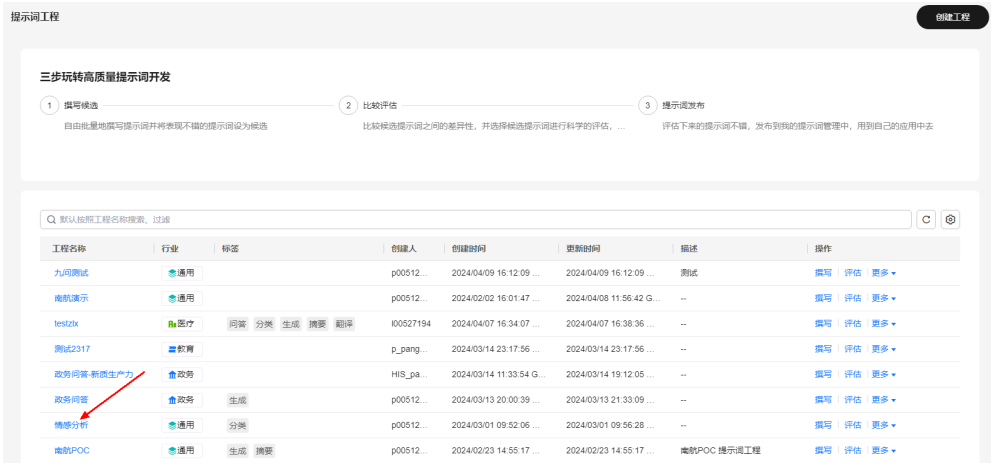


12.6 发布提示词

通过**两两比对提示词效果**和**批量评估提示词效果**，如果找到高质量的提示词，可以将提示词发布至“提示词管理”中。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 提示词工程”，进入提示词工程页面。
3. 在工程任务列表页面，找到所需要操作的工程任务，单击该工程名称，跳转工程任务下候选提示词页面。

图 12-27 提示词工程



4. 勾选所需的提示词，并单击“保存到模板库”。

图 12-28 保存提示词到模板库

5. 进入“应用开发 > 提示词管理 > 我的提示词”页面，查看发布的提示词。

图 12-29 我的提示词

13 AI 助手

13.1 什么是 AI 助手

AI助手是一种基于NLP大模型构建的人工智能应用，它通过结合多种工具并利用大模型的对话问答、规划推理、逻辑判断等能力，来理解和回应用户的需求。

例如，需要构建一个企业助理应用，该应用需要具备预定会议室、创建在线文档和查询报销信息等功能。在构建此应用时，需要将预定会议室与创建在线文档等功能的API接口定义为一系列的工具，并通过AI助手，将这些工具与大模型进行绑定。当用户向AI助手提问时，大模型就会根据用户的问题自动规划调用相应工具，从而实现对应的功能。

AI助手具备以下核心功能：

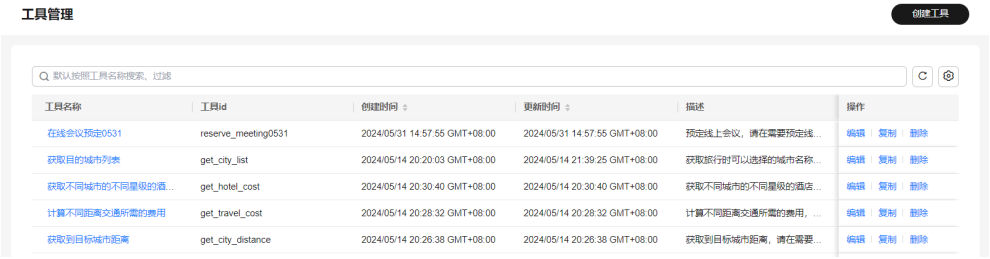
- **大模型调用能力：**AI助手可以根据特定的指令调用NLP大模型，以改变AI助手的回复方式，使其更好地响应用户的需求。例如，让AI助手表现得更加友好、专业，或者更加幽默。
- **多工具混合调用：**AI助手可以集成不同功能的工具来解决问题，这使得AI助手能够处理各种复杂的任务。
- **统一调用入口：**AI助手通过一个统一的问答入口，即可解决多种问题，这使得用户可以在一个地方就能完成所有的任务。
- **有效分发业务问题：**AI助手可以根据用户的需求和工具的定位，自动对问题进行分发，这使得AI助手可以更准确地理解用户的需求，并提供相关的服务。

13.2 配置 AI 助手工具

各种功能的API经封装后，将形成一个个工具，AI助手通过大模型来调用不同的工具，实现相应的功能。在创建AI助手前，需要将使用的功能封装为工具。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 工具管理”，单击页面右上角“创建工具”。

图 13-1 工具管理



3. 在“创建工具”页面参考**表13-1**完成工具代码的设置。

表 13-1 创建工具参数说明

参数	是否必选	参数类型	描述
tool_id	是	String	工具ID，必须由英文小写字母和_组成，需要符合实际工具含义。
tool_desc	是	String	工具的描述，尽可能的准确简短描述工具的用途。 **说明** 该描述直接影响大模型对工具使用的判断，请尽量描述清楚。如果AI助手实际执行时，无法根据用户问题匹配到工具，或者匹配效果不理想，可以修改此描述。
input_schema	是	Json Schema	工具输入参数。将API封装为工具时，调用该API的请求参数。请求体以json schema的形式进行描述，参数说明请参考**官方指导**。
output_schema	是	Json Schema	工具输出参数。将API封装为工具时，调用该API的响应参数。请求体以json schema的形式进行描述，参数说明请参考**官方指导**。
metadata	是	Object **metadata**	扩展字段 。

表 13-2 metadata 参数说明

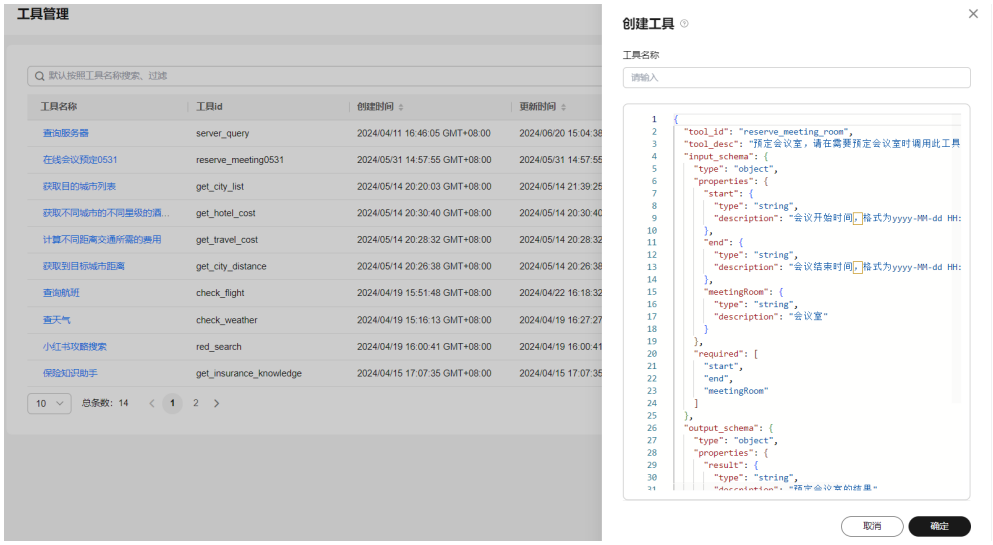
参数	是否必选	参数类型	描述
url	是	String	assistant api调用地址。
authType	是	String	用于指定身份验证的类型，默认值“OAuth”，使用OAuth协议进行身份验证。

代码示例：

```
{
  "tool_id": "reserve_meeting_room",
```

```
"tool_desc": "预定会议室，请在需要预定会议室时调用此工具，预定前需要先查询会议室状态",
"input_schema": {
  "type": "object",
  "properties": {
    "start": {
      "type": "string",
      "description": "会议开始时间，格式为yyyy-MM-dd HH:mm"
    },
    "end": {
      "type": "string",
      "description": "会议结束时间，格式为yyyy-MM-dd HH:mm"
    },
    "meetingRoom": {
      "type": "string",
      "description": "会议室"
    }
  },
  "required": [
    "start",
    "end",
    "meetingRoom"
  ]
},
"output_schema": {
  "type": "object",
  "properties": {
    "result": {
      "type": "string",
      "description": "预定会议室的结果"
    }
  }
},
"metadata": {
  "url": "https://host/v1/api",
  "authType": "OAuth"
}
}
```

图 13-2 创建工具



4. 参数填写完成后，单击“确定”。

13.3 配置知识库

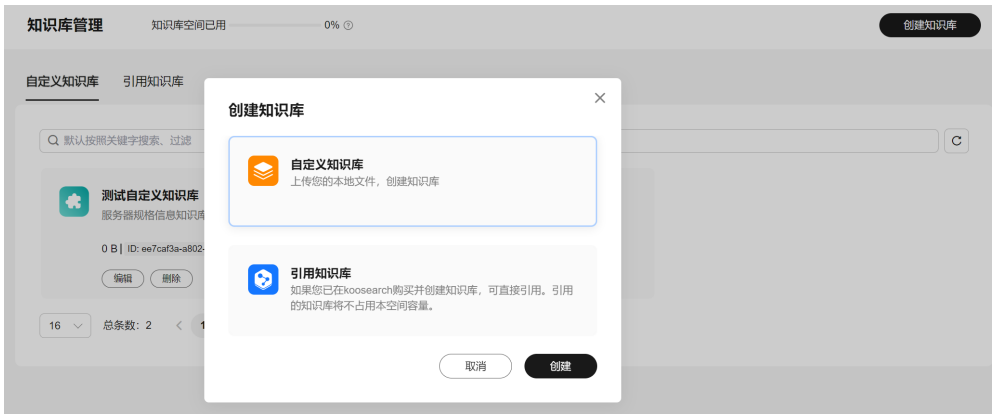
大模型在进行训练时，使用的是通用的数据集，这些数据集没有包含特定行业的数据。通过知识库功能，用户可以将领域知识上传到知识库中，向大模型提问时，大模型将会结合知识库中的内容进行回答，解决特定领域问题回答不准的现象。

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > 知识库管理”，单击页面右上角“创建知识库”。

   知识库分为自定义知识库、引用知识库。

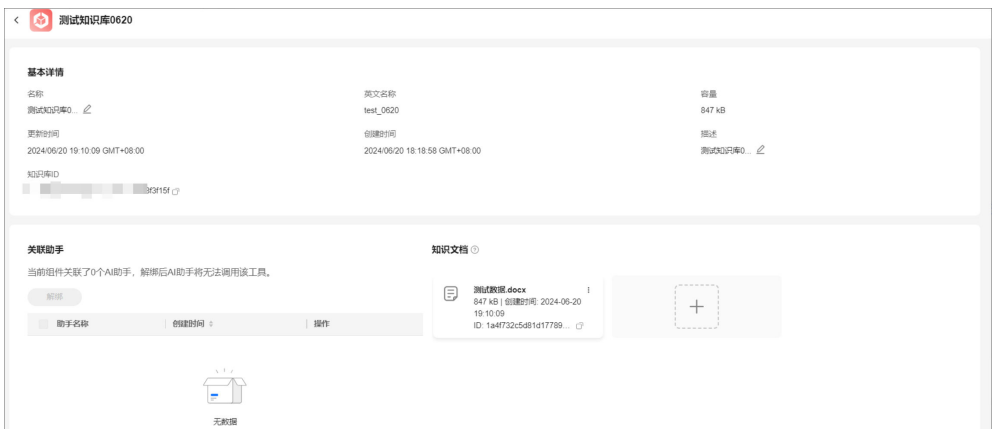
   - 自定义知识库：通过盘古大模型套件平台创建的知识库。
   - 引用知识库：引用在KooSearch服务中创建的知识库，KooSearch是基于大模型的文档问答服务，开通该服务请联系云搜索服务（CSS）技术支持。

图 13-3 创建知识库



3. 选择知识库类型后，单击“创建”进入知识库设置页面，创建知识库。
   - 当选择“自定义知识库”时，需要设置名称、英文名称、描述信息。注意英文名称和描述将影响模型检索效果，不可随意填写，需按照知识库中文档的实际内容或知识库目进行填写。设置完成后单击“立即创建”进入知识库详情页，上传文档。在详情页会同步展示与AI助手的绑定关系。

图 13-4 上传数据至知识库



- 当选择“引用知识库”时，需要设置名称、英文名称与描述信息，并选择需要引用的KooSearch知识库。注意英文名称和描述将影响模型检索效果，不可随意填写，需按照知识库中文档的实际内容或知识库目进行填写。

13.4 创建 AI 助手

1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > AI助手”，单击页面右上角“创建助手”。参考表13-3完成AI助手匹配。

表 13-3 创建 AI 助手参数说明

参数分类	参数名称	参数说明
基本信息	助手名称	设置AI助手的名称。
	描述	填写AI助手的描述，如填写功能介绍。
	指令	通过指令可以设定A助手的行为和响应。如设置AI助手可以扮演的角色、指定可以访问的工具、设置结果的输出风格等。
模型配置	嵌入模型	用于对AI助手进行任务规划、工具选择和生成回复。
	模型版本	选择与“嵌入模型”对应的版本。例如，嵌入模型为N2系列，则模型版本也为N2。
工具配置	网页搜索	开启网页搜索后，可以通过调用web搜索来解决模型对于事实类问题回答不好的现象。
	添加一个工具	用于拓展AI助手功能，使其能够与外部系统进行交互。可以直接创建一个工具，或者从搜索框中选择已经创建好的工具。
	知识库	通过知识库提升AI助手在特定领域问题的回答效果。

参数分类	参数名称	参数说明
高级配置	工具召回策略	设置从所有可用工具中选择最相关的工具来处理用户的问题策略。 ● 类型：使用词嵌入技术（embedding）来衡量用户问题与工具之间的相关性。 ● 中断策略：当相关性得分小于设置的阈值，则不召回任何工具，终止后续流程。 ● 阈值：指工具召回的相关性得分的阈值。阈值越高，召回工具的数量越少，但对召回工具的准确性要求更高。 ● 多轮改写模型：对用户的问题进行多次改写，以增加召回内容的多样性。 ● 检索工具数量：指在处理用户问题时，会检索出相关性最高的前N个工具。
	历史信息处理策略	设置处理和利用用户历史对话信息的策略。 ● 类型：对用户历史对话信息进行截断（truncation），用于控制传递给模型的上下文长度。 ● 截断窗口大小：指在处理用户的历史对话信息时，系统会保留最近的N个对话传递给模型。
	历史关键信息抽取	用于截取历史对话中的关键信息，将关键信息带入当前轮次的对话中。 该参数需要与工具配合使用，需要填入工具input_schema参数中**API的请求参数**。例如，在**配置AI助手工具**的代码示例中，创建预定会议室API的请求参数中有start，设置为start，即将会议的开始相关的信息作为关键信息，带入新轮次的对话中。

指令参数输入示例：

```
# 角色: 旅行规划助理


## 简介
- 作者: pangu
- 版本: 0.1
- 语言: 中文
- 描述: 我是一个旅行规划助理，能够帮助用户查询天气、预订车票，以及查询旅游地的风景人文。


## 技能
```

```
### 技能-1
1. 通过调用{tool_id}工具，查询目的地的天气信息。


### 技能-2
1. 能够根据用户需求和偏好，帮助用户规划旅行路线。
2. 能够通过调用{tool_id}工具为用户提供最佳的旅行路线。


### 技能-3
1. 能够解读和理解各种类型的旅行攻略文档。
2. 能够根据文档内容回答用户的旅行相关问题。


## 规则
1. 在任何情况下都不要破坏角色。
2. 不要编造事实。


## 初始化
身为一名旅行规划助手，我必须遵守规则，我必须用默认的语言和用户交谈，如果用户没有输入问题，我会介绍我自己和我的工作流程，等待用户的提问。
```

图 13-5 创建 AI 助手

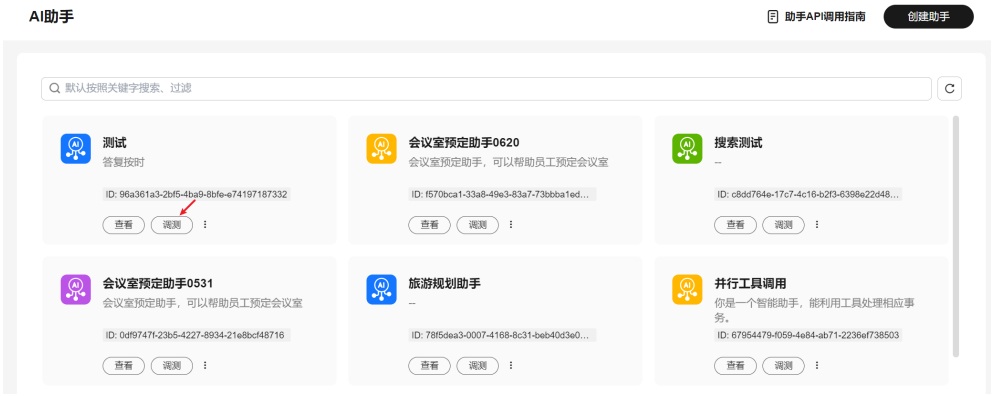
3. 单击“立即创建”完成AI助手的创建工作。

13.5 调测 AI 助手

在AI助手的创建页面可以直接进行调测，也可以在AI助手列表页进行调测。

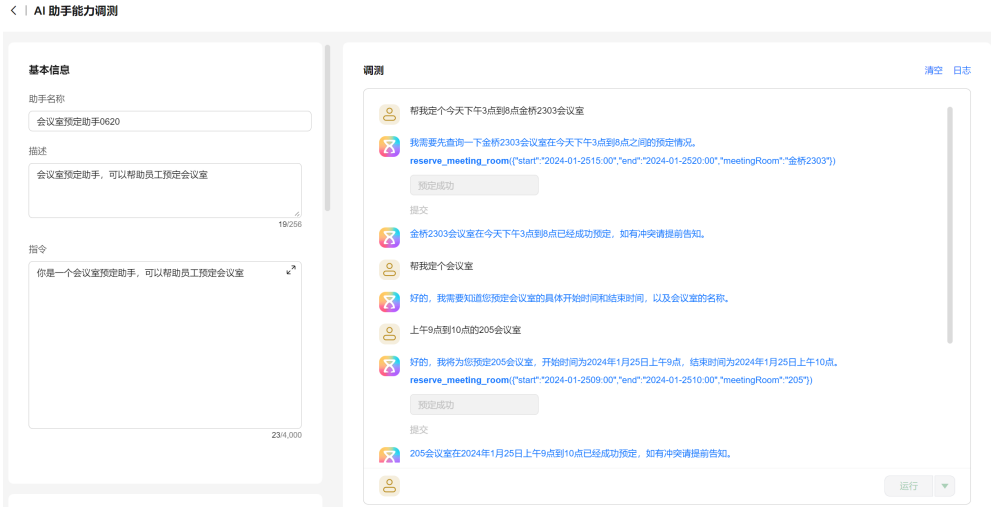
1. 登录盘古大模型套件平台。
2. 在左侧导航栏中选择“应用开发 > AI助手”，选择需要调测的AI助手，单击“调测”按钮。

图 13-6 AI 助手



3. 在调测页面，可以调整AI助手的指令，输入问题后，单击“运行”获得模型回复结果。

图 13-7 AI 助手能力调测

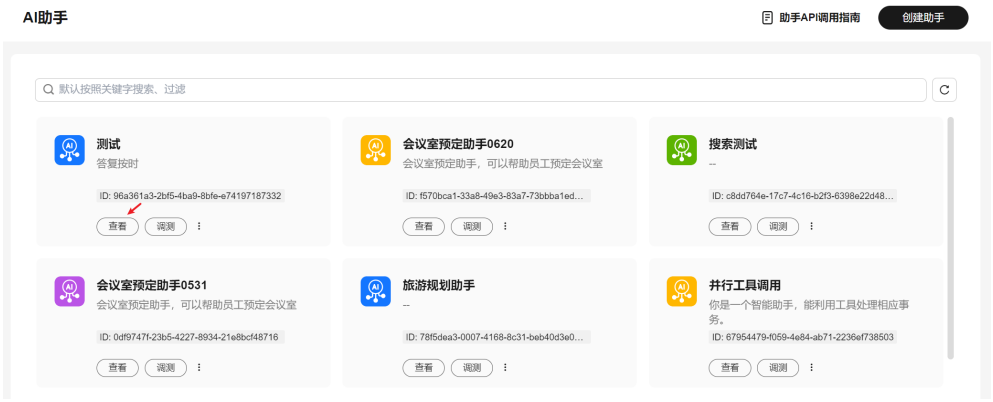


13.6 调用 AI 助手 API

获取 AI 助手 API 调用地址

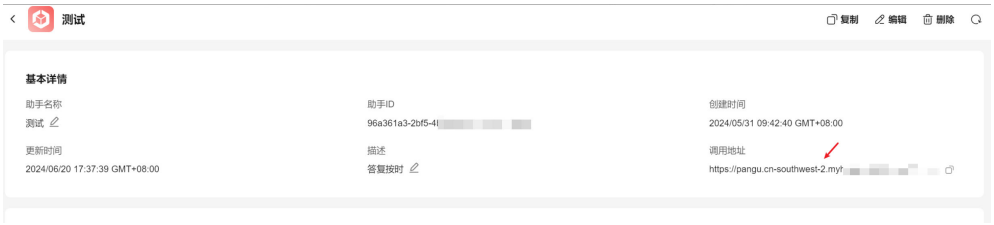
1. 登录盘古大模型套件平台。
2. 左侧导航栏选择“应用开发 > AI助手”，选择需要运行的AI助手，单击“查看”。

图 13-8 查看 AI 助手



3. 在详情页面，AI助手API调用地址。

图 13-9 获取调用地址



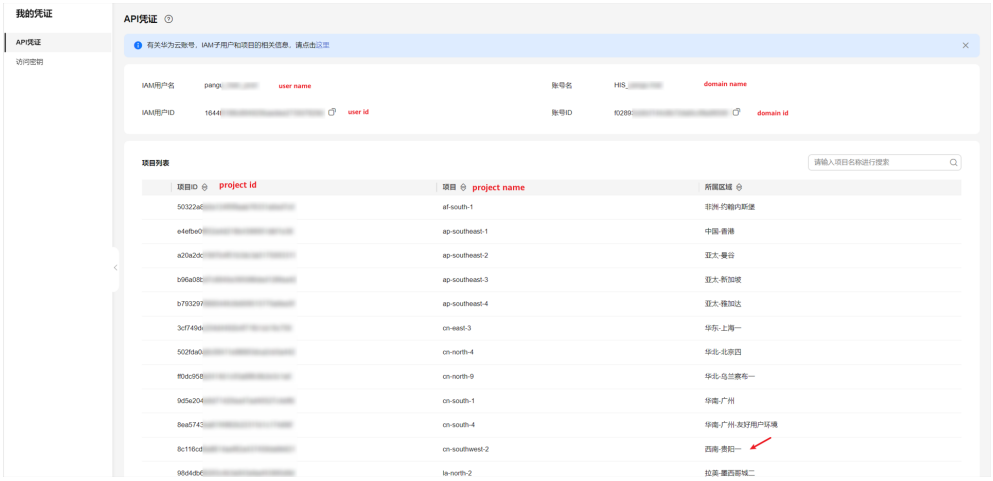
获取 Token

本示例中，通过使用Postman软件获取Token。

1. 登录“**我的凭证 > API凭证**”页面，获取user name、domain name、project id。

   由于Assistant当前部署在“西南-贵阳一”区域，需要获取与“西南-贵阳一”区域对应的project id。

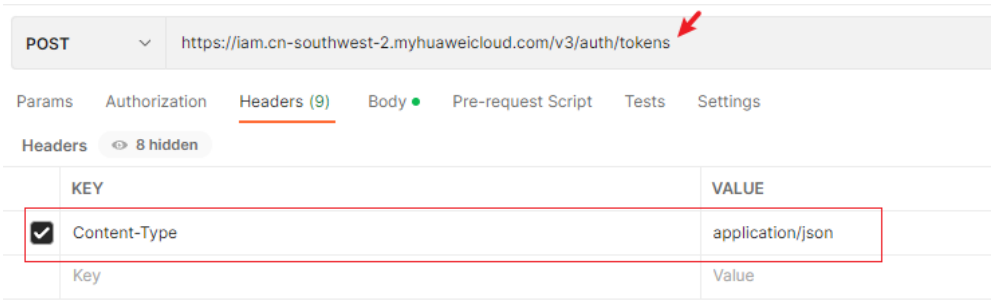
图 13-10 获取 user name、domain name、project id



2. 打开Postman，新建一个POST请求，并输入“西南-贵阳一”区域的“获取Token”接口。并填写请求Header参数。

- 接口地址：https://iam.cn-southwest-2.myhuaweicloud.com/v3/auth/tokens
- 请求Header参数名为Content-Type，参数值为application/json。

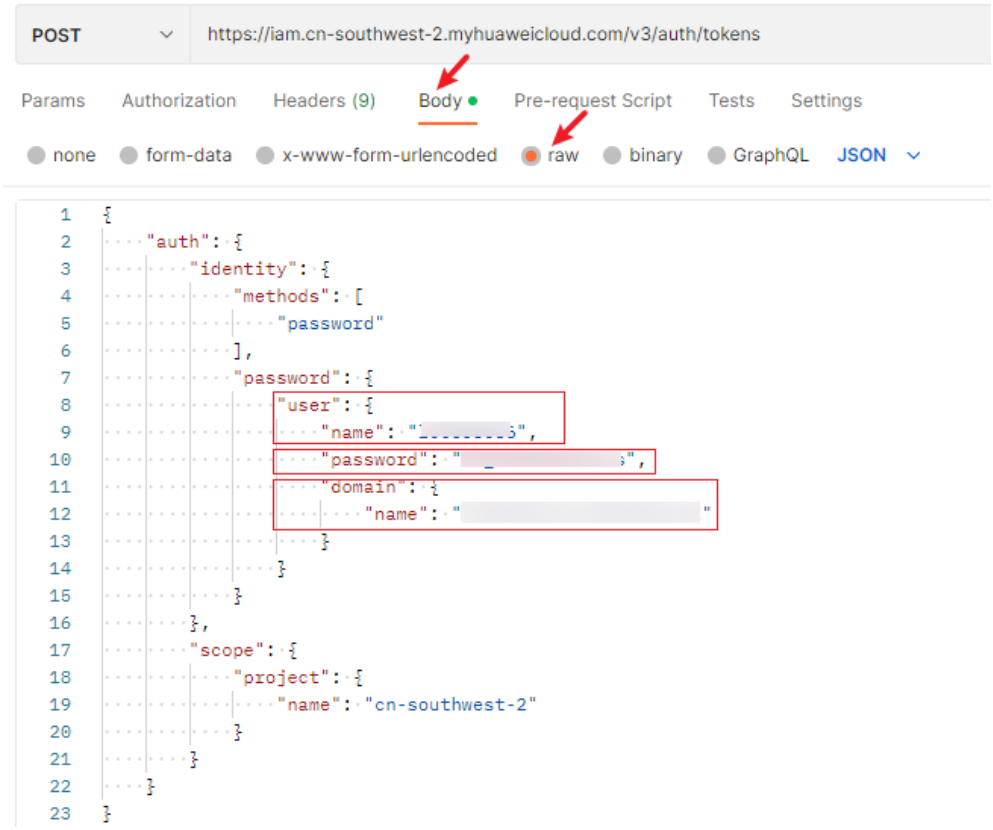
图 13-11 填写获取 Token 接口



3. 填写“获取token”接口的请求体。在Postman中选择“Body > raw”选项，参考图13-12复制并填入以下代码，并填写user name、domain name、password。

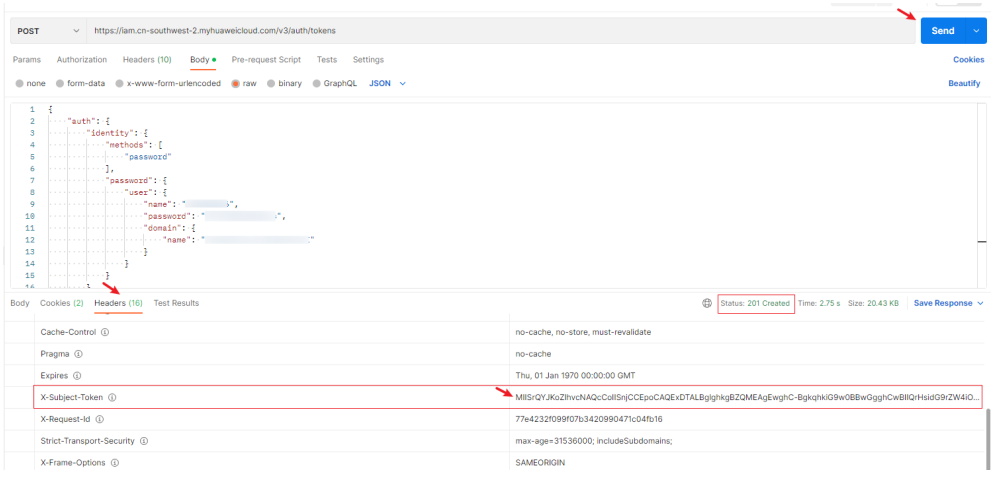
```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "name": "username", //IAM用户名
                    "password": "********", //华为云账号密码
                    "domain": {
                        "name": "domainname" //账号名
                    }
                }
            }
        },
        "scope": {
            "project": {
                "name": "cn-southwest-2" //服务当前部署在“西南-贵阳一”区域，取值为cn-southwest-2
            }
        }
    }
}
```

图 13-12 填写请求 Body



4. 单击Postman界面“Send”按钮，发送请求。当接口返回状态为201时，表示Token接口调用成功，此时单击“Headers”选项，找到并复制“X-Subject-Token”参数对应的值，该值即为需要获取的Token。

图 13-13 获取 Token



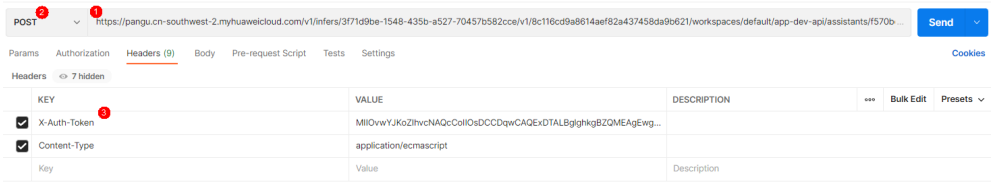
调用 AI 助手 API

本示例中，通过使用Postman软件调用AI助手API，API的详细请求参数、响应参数介绍请参见**AI助手API参数说明**。

1. 打开Postman，新建一个POST请求，在地址栏填写**获取AI助手API调用地址**获取的调用地址。

2. 在Header中配置IAM Token信息。
   - 请求Header参数名为X-Auth-Token，参数值为**获取Token**获取的token值。
   - 请求Header参数名为Content-Type，参数值为application/json。

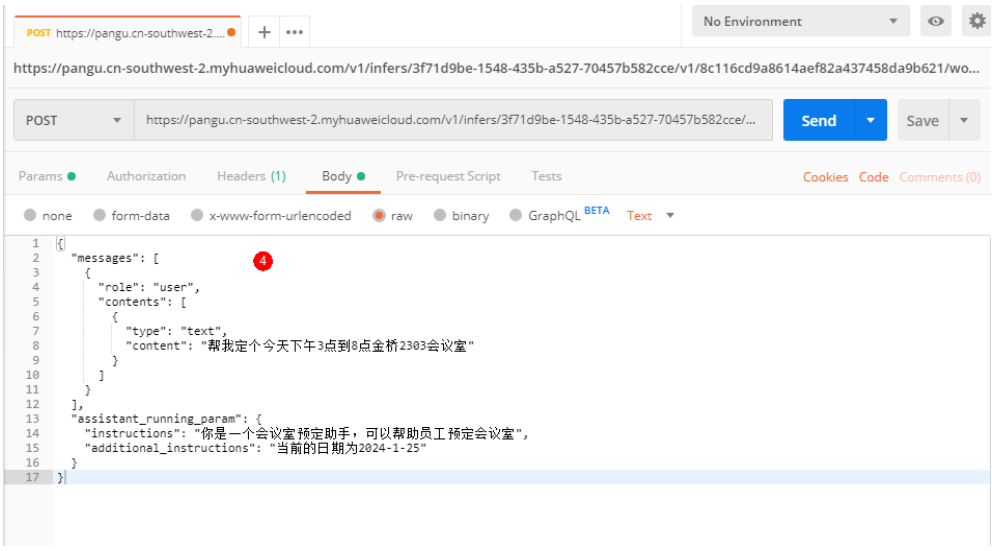
图 13-14 构造请求



3. 在Body中填写请求消息体。

```
{
 "messages": [
  {
   "role": "user",
   "contents": [
    {
     "type": "text",
     "content": "帮我定个今天下午3点到8点金桥2303会议室"
    }
   ]
  }
 ],
 "assistant_running_param": {
  "instructions": "你是一个会议室预定助手，可以帮助员工预定会议室",
  "additional_instructions": "当前的日期为2024-1-25"
 }
}
```

图 13-15 填写请求 Body



4. 查看请求调用结果。

   单击“Send”发送请求，请求调用成功后，查看返回结果。

   响应示例代码：

```
{
   "assistant_session_id": "test_session_1",
   "assistant_id": "6c8eb381-2c34-4ebc-bcf4-3fccb703c977",
   "required_action": { # 给出了Assistant的思考
```

```
      "action_id": "c7669ea3-1abd-4d17-aa67-5df19bf3aba5",
      "message_id": "07daf8d9-38a8-460c-8068-3ff420e76c8d",
      "thought": "好的，我需要先查询会议室的状态，然后才能帮您预定。现在开始查询会议室状态。",
      "tool_invoking": [
         {
            "tool_invoking_id": "bdf4f51f-ee58-4a10-8b13-35d45c3ddbe4",
            "action_id": "c7669ea3-1abd-4d17-aa67-5df19bf3aba5",
            "tool_id": "meeting_room_status_query",
            "tool_input": "{\"start\": \"2024-01-25 15:00\", \"end\": \"2024-01-25 20:00\",
\"meetingRoom\": \"金桥2303\"}",
            "metadata": "{\"url\":\"https://host/v1/api\",\"authType\":\"OAuth\"}",
         }
      ],
      "required": true,
      "usage": {
         "input_tokens": 589,
         "output_tokens": 85,
         "cost": 2941
      }
   },
   "assistant_messages": [
      {
         "message_id": "07daf8d9-38a8-460c-8068-3ff420e76c8d",
         "assistant_session_id": "test_session_1",
         "role": "assistant",
         "contents": [
            {
               "type": "text",
               "content": ""
            }
         ],
         "actions": [
            {
               "action_id": "c7669ea3-1abd-4d17-aa67-5df19bf3aba5",
               "message_id": "07daf8d9-38a8-460c-8068-3ff420e76c8d",
               "thought": "好的，我需要先查询会议室的状态，然后才能帮您预定。现在开始查询会议室状
态。",
               "tool_invoking": [
                  {
                     "tool_invoking_id": "bdf4f51f-ee58-4a10-8b13-35d45c3ddbe4",
                     "action_id": "c7669ea3-1abd-4d17-aa67-5df19bf3aba5",
                     "tool_id": "meeting_room_status_query",
                     "tool_input": "{\"start\": \"2024-01-25 15:00\", \"end\": \"2024-01-25 20:00\",
\"meetingRoom\": \"金桥2303\"}",
                     "metadata": "{\"url\":\"https://host/v1/api\",\"authType\":\"OAuth\"}"
                  }
               ],
               "required": true,
               "usage": {
                  "input_tokens": 589,
                  "output_tokens": 85,
                  "cost": 2941
               }
            }
         ]
      }
   ],
   "status": "requires_action"  # status为requires_action表示他需要对这个action进行反馈，
required_action中给出了Assistant的思考
}
```


AI 助手 API 参数说明

- **请求参数**

表 13-4 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token，通过调用IAM服务获取用户Token接口获取（响应消息头中X-Subject-Token的值）。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

表 13-5 请求 Body 参数

参数	是否必选	参数类型	描述
messages	是	Array of **messages** objects	对话信息，包含两个属性：role和content。
assistant_running_param	否	Json Schema	用于更改AI助手的指令。包含instructions和additional_instructions两个参数。 • instructions：AI助手的描述信息。 • additional_instructions：用于在当前会话中向AI助手中补充附加信息，例如与用户查询问题相关的上下文信息。

表 13-6 messages

参数	是否必选	参数类型	描述
content	是	String	表示对话的内容，对话内容为文本（text）类型。
role	否	String	表示对话的角色，取值是user。

- **响应参数**

表 13-7 响应 Body 参数

参数	参数类型	描述
assistant_session_id	String	运行AI助手时的会话id。
assistant_id	String	AI助手id。
required_action	Object **表5 required_action**	AI助手的思考过程。
assistant_messages	Json Schema	会话信息。
status	String	会话状态。status为requires_action，表示需要对action进行反馈，required_action中给出了Assistant的思考。

表 13-8 required_action

参数	参数类型	描述
action_id	String	动作id。
message_id	String	信息id。
thought	String	AI助手基于思考做出的反馈。
tool_invoking	Array **tool_invoking**	AI助手所调用的工具信息。
required	String	请求状态。
usage	Json Schema	大模型使用情况。

表 13-9 tool_invoking

参数	参数类型	描述
tool_invoking_id	String	工具调用id。
action_id	String	动作id。
tool_id	String	工具id。
tool_output	String	调用工具产生的输出结果。
metadata	String	调用接口信息。

14 盘古应用开发 SDK

14.1 盘古应用开发 SDK 简介

应用开发 SDK 概述

应用开发SDK针对大模型应用开发场景，对大语言模型进行封装，提供了提示词模板、记忆、技能、智能代理等功能模块，简化用户的开发工作，帮助用户快速开发一个大模型应用。当前应用开发SDK支持如下语言：

- Java
- Python

开发环境要求

华为云盘古大模型应用开发SDK要求JAVA SDK 1.8及其以上版本，Python 3.9及以上版本。

14.2 准备工作

使用盘古大模型Java SDK时，需要在代码中配置以下信息，请提前收集。

表 14-1 资源列表

类型	资源	是否必选	依赖信息	参考文档	备注
大语言模型	华为云盘古	是（大语言模型至少选一个）	• 盘古模型API调用URL。 • 华为云IAM账号认证信息。	• 盘古大模型API参考文档：申请资源时，可联系客户支持获取。 • IAM帮助文档：https://support.huaweicloud.com/api-identitycenter/iic_04_0032.html	• 模型API调用文档中完整url。 格式示例为：https://{endpoint}/v1/{project_id}/deployments/{deployment_id}/chat/completions SDK配置：https://{endpoint}/v1/{project_id}/deployments/{deployment_id}（/chat/completions在SDK代码中已经进行了设置）。 • IAM endpoint需要根据服务所在的区域正确配置，参考帮助文档“终端节点”章节查找。 • 参考IAM帮助文档，获取账号相关信息。
	华为云Gallery托管三方模型	否	• Gallery三方托管模型API调用URL。 • 华为云IAM账号认证信息。	• 三方大模型API参考文档：申请资源时，可联系客户支持获取。 • IAM帮助文档：https://support.huaweicloud.com/api-identitycenter/iic_04_0032.html	• 参考三方大模型API参考文档“API”章节，获取URL信息。 格式示例：https://{endpoint}/v1/infers/{deployment_id} • 参考IAM帮助文档，获取账号相关信息。

类型	资源	是否必选	依赖信息	参考文档	备注
Embedding模型	华为云CSS	否	• CSS Embedding模型API调用URL。 • 华为云IAM账号认证信息。	• CSS Embedding模型API参考文档：申请资源时，可联系客户支持获取。 • IAM帮助文档：https://support.huaweicloud.com/api-identitycenter/iic_04_0032.html	• 参考CSS Embedding模型API参考文档，获取URL信息。 格式示例：https://{endpoint}/v1/{project_id}/applications/{app_id}/{model_version} • 参考IAM帮助文档，获取账号相关信息。
文档加载	华为云CSS	否	• CSS文档解析服务API调用URL。 • 华为云IAM账号认证信息。	• CSS文档解析服务API参考文档：申请资源时，可联系客户支持获取。 • IAM帮助文档：https://support.huaweicloud.com/api-identitycenter/iic_04_0032.html	• 参考CSS Embedding模型API参考文档，获取URL信息。 格式示例：https://{endpoint}/v1/{project_id}/applications/{app_id} • 参考IAM帮助文档，获取账号相关信息。
向量记忆	华为云CSS	否	• 集群host信息。 • 用户认证信息。	• 云搜索服务CSS：https://support.huaweicloud.com/css/index.html	• 参考CSS服务“快速入门”章节创建机器后，在集群信息中获取hosts信息。 示例： https://10.0.0.1:9200，https://10.0.0.2:9200 • 用户认证信息就是创建集群时设置的用户/密码。

类型	资源	是否必选	依赖信息	参考文档	备注
	华为云CSS（集成Embedding）	否	• 集群host信息。 • 用户认证信息。	• 云搜索服务CSS：https://support.huaweicloud.com/css/index.html	• 参考CSS服务“快速入门”章节创建机器后，在集群信息中获取hosts信息。 示例： https://10.0.0.1:9200，https://10.0.0.2:9200 • 用户认证信息就是创建集群时设置的用户/密码。
	Elastic Search	否	• 集群host信息。 • 用户认证信息。	• ES官网：https://www.elastic.co/cn/elasticsearch/	hosts示例：https://10.0.0.1:9200，https://10.0.0.2:9200
标量存储	华为云DCS	否	• host信息。 • 用户认证信息。	• 分布式缓存DCS：https://support.huaweicloud.com/dcs/index.html	host示例：redis://10.0.0.1:6379
	Redis	否	• host信息。 • 用户认证信息。	• Redis官网：https://redis.io/	host示例：redis://10.0.0.1:6379
	华为云RDS	否	• host信息。 • 用户认证信息。	• 云数据库RDS：https://support.huaweicloud.com/rds/index.html	-
	Mysql	否	• host信息。 • 用户认证信息。	• Mysql官网：https://www.mysql.com/	-

iam认证与SDK配置项的映射关系如下：

```
POST https://iam.cn-southwest-2.myhuaweicloud.com/v3/auth/tokens
```



```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "name": "username", //IAM用户名
                    "password": "********", //华为云账号密码
                    "domain": {
                        "name": "domainname" //账号名
                    }
                }
            }
        },
        "scope": {
            "project": {
                "name": "cn-southwest-2" //服务当前部署在“西南-贵阳一”区域，取值为cn-southwest-2
            }
        }
    }
}
```

14.3 Java SDK

14.3.1 安装 SDK

Maven 中央仓导入

在项目**pom.xml**中参考以下方式添加依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>pangu-kits-app-dev-java</artifactId>
    <version>2.4.0</version>
</dependency>
```

本地导入

1. 下载SDK包，将**pangu-kits-app*.jar**文件放在项目目录lib文件夹下（.jar文件放置路径也可以自定义）。
2. 在项目**pom.xml**中参考以下方式添加依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>pangu-kits-app-dev-java-bundle</artifactId>
    <version>2.4.0</version>
    <scope>system</scope>
    <systemPath>${project.basedir}/lib/pangu-kits-app-dev-java-bundle-2.4.0.jar</systemPath>
</dependency>
```

API 手册

API手册请参见**SDK API 参考**。

14.3.2 配置 SDK

基础配置项

SDK依赖的配置项主要通过读取**llm.properties**配置文件；如果配置文件名不为**llm.properties**，需要在项目中主动设置，方法如下：

1. 在resources路径下，创建**llm.properties**文件，并根据实际需要配置相应的值。
2. 如果需要自定义配置文件名，可以参考以下代码设置。

```
// 建议在业务项目入口处配置
// 不需要添加.properties后缀
ConfigLoadUtil.setBaseName("application");
```

3. 完整配置项如下：

   配置项中的密码等字段建议在配置文件或者环境变量中密文存放，使用时解密，确保安全，详见**配置文件敏感信息加密配置**。

```
################################## GENERIC CONFIG
##################################

## User-defined Prompt.
#
sdk.prompt.path=

## Proxy.
# Examples: http://127.0.0.1:8000 ;
#
# sdk.proxy.enabled=
# sdk.proxy.url=
# sdk.proxy.user=
# sdk.proxy.password=

## Generic IAM info. This config is used when the specified IAM is not configured.
## Either user password authentication or AK/SK authentication.
#
sdk.iam.url=
sdk.iam.domain=
sdk.iam.user=
sdk.iam.password=
sdk.iam.project=
sdk.iam.ak=
sdk.iam.sk=
sdk.iam.disabled=


################################## LLM ###################################

## Pangu
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;
#
sdk.llm.pangu.url=
## If necessary, you can specify the IAM configuration.
# sdk.llm.pangu.iam.url=
# sdk.llm.pangu.iam.domain=
# sdk.llm.pangu.iam.user=
# sdk.llm.pangu.iam.password=
# sdk.llm.pangu.iam.project=
# sdk.llm.pangu.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.llm.pangu.proxy.enabled=

## Gallery
# Examples: https://{endPoint}/v1/infers/{deploymentId} ;
#
sdk.llm.gallery.url=
## If necessary, you can specify the IAM configuration.
```



```
# sdk.llm.gallery.iam.url=
# sdk.llm.gallery.iam.domain=
# sdk.llm.gallery.iam.user=
# sdk.llm.gallery.iam.password=
# sdk.llm.gallery.iam.project=
# sdk.llm.gallery.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.llm.gallery.proxy.enabled=


################################## EMBEDDINGS
##################################

## CSS
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId}/{modelVersion} ;
#
sdk.embedding.css.url=
## If necessary, you can specify the IAM configuration.
# sdk.embedding.css.iam.url=
# sdk.embedding.css.iam.domain=
# sdk.embedding.css.iam.user=
# sdk.embedding.css.iam.password=
# sdk.embedding.css.iam.project=
# sdk.embedding.css.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.embedding.css.proxy.enabled=


################################## MEMORY ##################################

## CSS or ES
# Examples: http://127.0.0.1:9200,http://127.0.0.2:9200 ;
#
sdk.memory.css.url=
sdk.memory.css.user=
sdk.memory.css.password=

## DCS or Redis
# Examples: 127.0.0.1:6379 ;
#
# sdk.memory.dcs.url=
# sdk.memory.dcs.user=
# sdk.memory.dcs.password=

## RDS or Mysql
# Examples: jdbc:mariadb://127.0.0.1:3306/sdk?
useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai ;
#
# sdk.memory.rds.url=
# sdk.memory.rds.user=
# sdk.memory.rds.password=
# sdk.memory.rds.poolSize=


################################## DOC SPLIT
##################################

## CSS
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId} ;
#
# sdk.doc.split.css.url=
# sdk.doc.split.css.filepath=
# sdk.doc.split.css.mode=
## If necessary, you can specify the IAM configuration.
# sdk.doc.split.css.iam.url=
# sdk.doc.split.css.iam.domain=
# sdk.doc.split.css.iam.user=
# sdk.doc.split.css.iam.password=
# sdk.doc.split.css.iam.project=
```



```
# sdk.doc.split.css.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.doc.split.css.proxy.enabled=
```

日志打印配置

SDK日志采用slf4j框架，业务项目中可以使用logback、log4j、slf4j默认等具体实现（实现任一即可）。

- 通过logback实现。
  a. 参考以下代码在pom文件中引入logback相关依赖（建议 1.3.1--1.3.8 版本）。

```
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-classic</artifactId>
   <version>1.3.8</version>
</dependency>
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-core</artifactId>
   <version>1.3.8</version>
</dependency>
```

  b. 在resource目录下创建**logback.xml**，用于配置日志级别、格式、输出位置等，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
      <encoder>
         <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg
%n</pattern>
      </encoder>
   </appender>
   <root level="DEBUG">
      <appender-ref ref="CONSOLE" />
   </root>
</configuration>
```

- 通过log4j实现
  a. 参考以下代码在pom文件中引入log4j适配slf4j依赖（建议 2.0.7 版本）。

```
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-log4j12</artifactId>
   <version>2.0.7</version>
</dependency>
```

  b. 在resource目录下创建**log4j.xml**，用于配置日志级别、格式、输出位置等，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration>
   <appender name="console" class="org.apache.log4j.ConsoleAppender">
      <layout class="org.apache.log4j.PatternLayout">
         <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n" />
      </layout>
   </appender>
   <root>
     org.slf4j <priority value="debug" />
      <appender-ref ref="console" />
   </root>
</log4j:configuration>
```

- 通过slf4j simple实现
  a. 参考以下代码在pom文件中引入依赖。


```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.36</version>
</dependency>
```

b. 在resource目录下创建**simplelogger.properties**配置文件，示例如下：

```
org.slf4j.simpleLogger.defaultLogLevel=DEBUG
org.slf4j.simpleLogger.log.org.apache.hc.client5=INFO
org.slf4j.simpleLogger.showDateTime=true
org.slf4j.simpleLogger.dateTimeFormat=yyyy-MM-dd HH:mm:ss
org.slf4j.simpleLogger.showThreadName=true
org.slf4j.simpleLogger.showLogName=true
org.slf4j.simpleLogger.logFile=System.out
#org.slf4j.simpleLogger.showShortLogName=false
```

配置文件敏感信息加密配置

配置项中的密码等字段需要加密存储时，可以自定义实现解密方法。

- 实现解密类

```
// 配置类实现
public class CryptoCustom implements ConfigCryptor {
    @Override
    public String decrypt(String customTag, String encryptedValue) throws DecryptFailedException {
        // todo
    }
}
```

- 配置项中指定解密类

```
# 配置项指定解密类
sdk.crypto.implementation.path=CryptoCustom
```

- 配置项中可正常配置密文，示例如下

```
# 配置项示例：
sdk.llm.iam.iamPwd={Crypto.customTag}encryptedvalue
```

14.3.3 LLMs（语言模型）

LLMs模块用于对大语言模型API的适配封装，提供统一的接口快速地调用盘古、开源模型等模型API。

- 初始化：根据相应模型定义LLM类。例如，使用盘古LLM为：LLMs.of(LLMs.PANGU)。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;

// 初始化盘古LLM
LLM llm = LLMs.of(LLMs.PANGU);
```

- 基础问答：基础的模型文本问答，temperature等参数采用模型默认的设置。

```
llm.ask("你是谁？").getAnswer();
```

- 同时调用多个不同的LLM。

```
final LLMConfig config = LLMConfig.builder()
    .llmModuleConfig(
        LLMModuleConfig.builder().url(ConfigLoadUtil.getStringConf(null, "custom.llm.url")).build())
    .llmParamConfig(LLMParamConfig.builder().temperature(0.9).build())
    .build();
// 使用custom.llm.url
final LLM llm1 = LLMs.of(LLMs.PANGU, config);
log.info(llm1.ask("你好").getAnswer());
// 使用sdk.llm.pangu.url
final LLM llm2 = LLMs.of(LLMs.PANGU);
log.info(llm2.ask("你好").getAnswer());
```

上述代码中custom.llm.url为自定义的url地址（名字由开发者任意指定，或直接传入url地址），可以指向不同的模型，因此llm1为一个大模型；而llm2没有指定

config，默认使用sdk.llm.pangu.url，若地址与custom.llm.url，则为另外一个大模型。

- 自定义参数问答：自定义设置如temperature等参数，获得对应的效果。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfigGallery;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;

// 设置模型参数，temperature为0.9
LLMConfig llmConfig =
LLMConfig.builder().llmParamConfig(LLMParamConfig.builder().temperature(0.9).build()).build();

// 如使用Gallery三方模型，使用以下配置
// LLMConfig llmConfig =
LLMConfigGallery.builder().llmParamConfig(LLMParamConfig.builder().temperature(0.9).build()).build(
);

// 初始化带参数的盘古LLM
LLM pangu = LLMs.of(LLMs.PANGU, llmConfig);
pangu.ask("写一篇五言律诗").getAnswer();
```

支持调整的参数解释。

```
private int maxTokens; //  完成时要生成的令牌的最大数量
private double temperature; //  调整随机抽样的程度，温度值越高，随机性越大
private double topP; //  核采样值， 和temperature不同时配置
private double presencePenalty; //  存在惩罚，增加模型谈论新主题的可能性
private double frequencyPenalty; //  频率惩罚，降低模型重复的可能性，提高文本多样性、创造型
private int bestOf; //  服务侧生成优选的回答数
private boolean stream; //  是否开启流式调用
```

- 流式问答：模型问答，开启流式效果，响应消息流式打印。

```
import com.huaweicloud.pangu.dev.sdk.api.callback.StreamCallBack;
import com.huaweicloud.pangu.dev.sdk.api.callback.StreamResult;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;

import lombok.extern.slf4j.Slf4j;

// 设置模型参数，stream为true
final LLMConfig llmConfig =
LLMConfig.builder().llmParamConfig(LLMParamConfig.builder().stream(true).build()).build();

// 盘古LLM
LLM pangu = LLMs.of(LLMs.PANGU, llmConfig);
// 设置回调处理逻辑
pangu.setStreamCallback(new StreamCallBackImp());
pangu.ask("写一篇200字的散文").getAnswer();

// 构造回调函数（以log打印为例，业务可自定义）
@Slf4j
public class StreamCallBackImp implements StreamCallBack {
    @Override
    public void onStart(String callBackId) {
        log.info("StreamCallBack onStart: callBackId ----> {}", callBackId);
    }

    @Override
    public void onEnd(String callBackId, StreamResult streamResult, LLMResp llmResp) {
        log.info("StreamCallBack onEnd: callBackId ----> {} || llmResp ----> {}", callBackId, llmResp);
    }

    @Override
    public void onError(String callBackId, StreamResult streamResult) {
        log.error("StreamCallBack onError: callBackId ----> {}", callBackId);
    }
```



```
    @Override
    public void onNewToken(String callBackId, LLMResp llmResp) {
        log.info("StreamCallBack onNewToken: callBackId ----> {} || llmResp ----> {}", callBackId,
llmResp);
    }
}
```

- 多轮对话问答：传递历史问答记录，实现多轮对话问答能力，同时支持自定义参数问答、流式问答。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.llms.request.ConversationMessage;
import com.huaweicloud.pangu.dev.sdk.api.llms.request.Role;
import java.util.ArrayList;
import java.util.List;

// 构造多轮对话：历史问答记录 + 最新问题
private List<ConversationMessage> buildMultiTurnChatMessages() {
    List<ConversationMessage> messages = new ArrayList<>();
    messages.add(ConversationMessage.builder().role(Role.SYSTEM).content("You are a helpful
assistant.").build());
    messages.add(ConversationMessage.builder().role(Role.USER).content("Who won the world series
in 2020?").build());
    messages.add(ConversationMessage.builder()
            .role(Role.ASSISTANT)
            .content("The Los Angeles Dodgers won the World Series in 2020.")
            .build());
    messages.add(ConversationMessage.builder().role(Role.USER).content("Where was it
played?").build());
    return messages;
}

// 初始化盘古LLM，开启问答
LLM llm = LLMs.of(LLMs.PANGU);
llm.ask(buildMultiTurnChatMessages()).getAnswer();
```

- 带人设的问答：支持在LLM配置项中设置人设，在LLM问答时系统会自动加上该人设，同时支持以上问答功能（暂不支持GALLERY三方模型）。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLM;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMModuleConfig;

// 设置模型系统人设
LLMConfig llmConfig = LLMConfig.builder()
    .llmModuleConfig(LLMModuleConfig.builder().systemPrompt("You are a helpful assistant.").build())
    .build();

// 初始化盘古LLM，开启问答
LLM llm = LLMs.of(LLMs.PANGU, llmConfig);
llm.ask("写一首五言律诗").getAnswer();
```

14.3.3.1 开源模型

1. SDK支持兼容OpenAI-API规范的开源模型。例如，用vllm框架使用OpenAI-API启动推理服务。当前鉴权方式支持AppCode鉴权和华为云的APIG简易认证方式。配置文件需要指定url和key，配置项为：

```
sdk.llm.openai.url=https://infer-app-modelarts-cn-southwest-2.myhuaweicloud.com/v1/infers/.../v1
sdk.llm.openai.key=your-key
```

2. 初始化LLM：

```
final OpenAI llm = new OpenAI(

LLMConfig.builder().llmModuleConfig(LLMModuleConfig.builder().moduleVersion("llama3-70B").buil
d()).build());
LLMRespOpenAI result = llm.ask("你好");
```


3. 上述moduleVersion根据实际情况传值，也可以使用代码进行url和key的配置：

```
final OpenAI llm = new OpenAI(LLMConfig.builder()
    .llmModuleConfig(LLMModuleConfig.builder().moduleVersion("expert_q4").build())
    .openAIConfig(OpenAIConfig.builder()
        .openaiBaseUrl("https://infer-app-modelarts-cn-southwest-2.myhuaweicloud.com/v1/infers/.../
v1")
        .openAiKey("your-key")
        .build())
    .build());
```

4. 使用AppCode鉴权添加的Header：

   X-Apig-AppCode:your-key

5. 使用APIG简易认证方式添加的Header：

   Authorization:Bearer your-key

6. 当LLM被定义好之后，使用方式与盘古大模型相同，开源模型也支持Agent调用，可参考**实例化Agent**。

14.3.3.2 自定义模型

如果使用的模型不是盘古或者兼容OpenAI-API的开源模型，如，闭源模型或者裸机部署的自定义推理服务，可以通过继承AbstractLLM自定义一个模型，示例代码如下：

```
@Slf4j
public class CustomLLM extends AbstractLLM<LLMResp> {
    /**
     * 初始化
     *
     * @param llmConfig llm参数配置
     */
    public CustomLLM(LLMConfig llmConfig) {
        super(llmConfig);
    }
    @Override
    protected LLMResp getLLMResponse(List<ConversationMessage> chatMessages, LLMParamConfig
llmParamConfig) {
        // 构造请求体
        Map<String, Object> request = new HashMap<>();
        request.put("temperature", 0.3);
        request.put("data",
chatMessages.stream().map(ConversationMessage::getContent).collect(Collectors.toList()));
        final String requestBody = JSON.toJSONString(request);
        log.info("request body : \n{}", JSON.toJSONString(JSON.parseObject(requestBody), true));
        // 从配置项读取url，构造post消息
        String url = ConfigLoadUtil.getStringConf(null, "llm.custom.api.url");
        if (StringUtils.isEmpty(url)) {
            throw new PanguDevSDKException("the llm.custom.api.url is not config");
        }
        HttpPost httpPost = new HttpPost(url);
        httpPost.setEntity(new StringEntity(requestBody, ContentType.APPLICATION_JSON));
        // 发送消息并处理响应
        String responseStr;
        if (llmConfig.getLlmParamConfig().isStream()) {
            // 处理流式请求
            httpPost.setHeader(new BasicHeader("Inference-Type", "stream"));
            final CloseableHttpAsyncClient httpclient = HttpUtil.getHttpAsyncClient(false);
            try {
                httpclient.start();
                final String callBackId = SecurityUtil.getUUID();
                final List<PanguChatChunk> panguChatChunks = new ArrayList<>();
                Future<HttpResponse> future = httpclient.execute(HttpAsyncMethods.create(httpPost),
                    StreamHelper.getAsyncConsumer(streamCallBack, callBackId, panguChatChunks),
                    StreamHelper.getCallBack(streamCallBack, callBackId, httpPost));

future.get(Optional.ofNullable(llmConfig.getHttpConfig().getAsyncHttpWaitSeconds()).orElse(300),
                    TimeUnit.SECONDS);
                final PanguChatResp allRespFromChunk =
```



```
StreamHelper.getAllRespFromChunk(panguChatChunks);
                return
LLMResp.builder().answer(allRespFromChunk.getChoices().get(0).getMessage().getContent()).build();
            } catch (Exception e) {
                throw new PanguDevSDKException(e);
            }
        } else {
            // 处理非流式请求
            final CloseableHttpClient httpClient = HttpUtil.getHttpClient(false);
            try {
                final CloseableHttpResponse response = httpClient.execute(httpPost);
                responseStr = EntityUtils.toString(response.getEntity(), StandardCharsets.UTF_8);
                log.info("response: \n{}", JSON.toJSONString(JSON.parseObject(responseStr), true));
                // 解析结果
                final JSONObject jsonObject = JSON.parseObject(responseStr);
                JSONObject result = jsonObject.getJSONObject("result");
                if (result == null) {
                    result = jsonObject;
                }
                final String content = ((JSONObject)
result.getJSONArray("answers").get(0)).getString("content");
                return LLMResp.builder().answer(content).build();
            } catch (IOException e) {
                throw new PanguDevSDKException(e);
            }
        }
    }
    @Override
    protected LLMResp getLLMResponseFromCache(String cache) {
        return LLMResp.builder().answer(cache).isFromCache(true).build();
    }
}
```

14.3.4 Prompt（提示词模板）

提示词模板模块提供模板格式化、自定义配置、few-shot管理功能。

- 模板格式化

```
import com.huaweicloud.pangu.dev.sdk.template.KV;
import com.huaweicloud.pangu.dev.sdk.template.PromptTemplate;
import org.junit.jupiter.api.Assertions;

import java.util.HashMap;

// 初始化Prompt模板对象
PromptTemplate promptTemplate = new PromptTemplate("Tell me a {{adjective}} joke about
{{content}}");

// 支持Map<String, Object>格式匹配替换Prompt模板
String format1 = promptTemplate.format(new HashMap<String, Object>() {
            {
                put("adjective", "funny");
                put("content", "chickens");
            }
        });
Assertions.assertEquals("Tell me a funny joke about chickens", format1);

// 支持KV格式匹配替换Prompt模板
String format2 = promptTemplate.format(KV.of("adjective", "funny"), KV.of("content", "chickens"));
Assertions.assertEquals("Tell me a funny joke about chickens", format2);
```

- few-shot

```
// 构造Prompt模板
interface Antonyms {
    String FIND_ANTONYMS = "给定一个单词：{{word}}，返回一个反义词：{{antonym}}";
}
```

```
import java.util.ArrayList;
import java.util.HashMap;
```



```
import java.util.List;
import java.util.Map;

// 构造示例
public interface ExampleSample {
    interface Antonyms {
        List<Map<String, Object>> antonyms = new ArrayList<Map<String, Object>>() {
            {
                add(new HashMap<String, Object>() {
                    {
                        put("word", "short");
                        put("antonym", "long");
                    }
                });
                add(new HashMap<String, Object>() {
                    {
                        put("word", "energetic");
                        put("antonym", "lethargic");
                    }
                });
                add(new HashMap<String, Object>() {
                    {
                        put("word", "windy");
                        put("antonym", "calm");
                    }
                });
            }
        };
    }
}
```

```
import com.huaweicloud.pangu.dev.sdk.prompts.exampleselector.LengthBasedES;
import com.huaweicloud.pangu.dev.sdk.template.PromptTemplate;
import com.huaweicloud.pangu.dev.sdk.prompts.FewShotPromptTemplate;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

// 构造prompt模板
PromptTemplate promptTemplate = new PromptTemplate(Antonyms.FIND_ANTONYMS);

// 构造fewshotPrompt模板：
// 基于长度策略（长度限制50）
// 传入prompt的开头 "参考以下示例，给出给定单词的反义词："
// 传入分隔符 "\n\n"
// 传入prompt的结尾 "给定一个单词：{{word}}，返回一个反义词："
FewShotPromptTemplate fewShotPromptTemplate =
    new FewShotPromptTemplate(new LengthBasedES(ExampleSample.Antonyms.antonyms,
promptTemplate, 50),
    promptTemplate, "参考以下示例，给出给定单词的反义词：", "给定一个单词：{{word}}，返回一个反义词：", "\n\n");

// 输入
Map<String, Object> inputs = new HashMap<String, Object>() {
    {
        put("word", "hard");
    }
};

// 根据输入匹配替换fewShotPromptTemplate，得到最终fewShotPrompt
String prompt = fewShotPromptTemplate.format(inputs);
```

- 自定义prompt

```
// 按约定的格式准备prompt文件；
// 文档结构和文件名参考提供的系统预置prompts文件

文档结构示例：
……
  prompts
    -- default
```



```
    -- documents
      -- stuff.pt
```

```
// 配置sdk配置项，指定prompt文件根路径，以 /home 目录下为例
sdk.prompt.path=/home/prompts/default
```

14.3.5 Memory（记忆）

Memory（记忆）模块结合外部存储为LLM应用提供长短期记忆能力，用于支持上下文记忆的对话、搜索增强等场景。

Memory（记忆）支持多种不同的存储方式和功能。

- Cache缓存：是一种临时存储数据的方法，它可以提高数据的访问速度和效率。缓存可以根据不同的存储方式进行初始化、更新、查找和清理操作。缓存还可以支持语义匹配和查询，通过向量和相似度的计算，实现对数据的语义理解和检索。
- Vector向量存储：是一种将数据转换为数学表示的方法，它可以度量数据之间的关系和相似度。向量存储可以根据不同的词向量模型进行初始化、更新、查找和清理操作。向量存储还可以支持多种相似算法，如余弦相似度、欧氏距离、曼哈顿距离等，实现对数据的相似度评分和排序。
- History对话消息存储：是一种将对话消息保存在内存中的方法，它可以记录和管理对话历史。对话消息存储可以根据不同的会话标识进行初始化、更新、查找和清理操作。对话消息存储还可以支持多种过滤条件，如时间范围、用户标识、消息类型等，实现对话消息的筛选和分析。

14.3.5.1 Cache

Cache缓存是一种临时存储数据的方法，它可以把常用的数据保存在内存或者其他设备中，这样当需要访问这些数据时，就不用再去原始的数据源查找，而是直接从缓存中获取，从而节省时间和资源。

- 对LLM使用缓存：

```
LLM llm = LLMs.of(LLMs.PANGU, llmConfig);
llm.setCache(Caches.of(Caches.IN_MEMORY));
llm.ask("你能讲一个笑话吗？")
```

  此时，再次使用同样的问题，则不会再调用大模型，而是直接从内存返回：

```
llm.ask("你能讲一个笑话吗？")
```

Cache缓存有以下几个操作：

- 初始化：指定缓存使用哪种存储方式。例如，使用内存型缓存可以设置为Cache cache = Caches.of(Caches.IN_MEMORY);。

```
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Cache;
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Caches;

// 内存型
Cache cache = Caches.of(Caches.IN_MEMORY);
// Redis
Cache cache = Caches.of(Caches.REDIS);
// mysql
Cache cache = Caches.of(Caches.SQL);
```

- 更新数据：指向缓存中添加或修改数据，需要指定数据的键值对和结果对象。例如，把1+1这个问题和对应的答案2保存到缓存中，可参考以下示例。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;

//更新数据
cache.update("1+1", LLMResp.builder().answer("2").build());
```

- 查询数据：从缓存中获取数据，需要指定数据的键值对。例如，查找1+1这个问题对应的答案，可参考以下示例。


```
import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;

// 查找数据
LLMResp cacheValue = cache.lookup("1+1");
```

- 清理数据：删除缓存中的数据。例如，删除对应的缓存数据，可参考以下示例。

```
// 清理
cache.clear()
```

- 配置过期策略：设置缓存有效期，支持基于时间和大小的限制。

```
// 设置缓存数据10s 后过期
Cache cache = Caches.of(Caches.IN_MEMORY,
CacheStoreConfig.builder().expireAfterWrite(10).build());
```

- 参数解释：用于设置缓存对象的一些基本信息，如过期时间等。

```
/**
 * 会话标识，业务确定
 */
@Builder.Default
private String sessionTag = "";

/**
 * 访问后到期时间，单位为秒， 默认不设置过期
 */
@Builder.Default
private int expireAfterAccess = -1;

/**
 * 写入后到期时间，单位为秒， 默认不设置过期
 */
@Builder.Default
private int expireAfterWrite = -1;

/**
 * 最大个数， 默认不设置过期
 */
@Builder.Default
private int maximumSize = -1;
```

- 语义缓存：语义缓存是一种基于向量和相似度的缓存方法，它可以实现对数据的语义匹配和查询。语义缓存可以根据不同的向量存储、相似算法、评分规则和阈值进行配置，并且可以使用不同的词向量模型进行嵌入。

```
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;
import com.huaweicloud.pangu.dev.sdk.api.llms.response.LLMResp;
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Cache;
import com.huaweicloud.pangu.dev.sdk.api.memory.cache.Caches;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.CacheStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.vectorstore.DistanceStrategy;

import org.junit.jupiter.api.Assertions;

//redis向量
// 不同的向量存储， 不同的相似算法；计算的评分规则不同； 可以同过scoreThreshold 设置相似性判断阈值
// 例如使用Redis向量、余弦相似度、CSS词向量模型，并且设置相似性判断阈值为0.1f，代码示例如下
Cache cache = Caches.of(CacheStoreConfig.builder()
.storeName(Caches.SEMANTIC_REDIS)
.vectorStoreName(Vectors.REDIS)
.serverInfo(ServerInfoRedis.builder().build())
.distanceStrategy(DistanceStrategy.COSINE)
.scoreThreshold(0.1f)
.embedding(Embeddings.of(Embeddings.CSS))
.build());


//会话标识
String sessionTag="test-semantic-cache-vector-001";
cache.setSessionTag(sessionTag);
```



```
cache.clear();

//数据已清空
Assertions.assertNull(cache.lookup("缓存是否存在？"));

//更新数据
//指向语义缓存中添加或修改数据，需要指定数据的键值对和结果对象
//例如，把“把缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识对应的答案“存在”保存到语义缓存中，代码示例如下
cache.update("缓存是否存在？",LLMResp.builder().answer("存在。").build());

//校验，一致
//用于检查缓存中的数据是否与查询的数据是否一致，如果一致，就返回缓存中的结果对象
//例如，查询“缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”
String query="缓存是否存在？";
LLMResp cacheValueAfter=cache.lookup(query);
Assertions.assertNotNull(cacheValueAfter);
System.out.println(query+cacheValueAfter.getAnswer());

//校验，相似
//用于检查缓存中的数据是否与查询的数据语义相似，如果相似，就返回缓存中的结果对象。这个操作需要使用向量和相似度的计算，以及设置的阈值来判断
//例如，查询“缓存存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”，因为这个问题和“缓存是否存在？”问题语义相似
String query_sim="缓存存在？";
LLMResp cacheValueSemantic=cache.lookup(query_sim);
Assertions.assertNotNull(cacheValueSemantic);
System.out.println(query_sim+cacheValueSemantic.getAnswer());

//校验，不一致
//用于检查缓存中的数据是否与查询的数据不一致，如果不一致，返回空值
//例如，查询“有没有数据？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就无法从缓存中获取到任何答案，因为这个问题和之前保存的问题都不一致
String query_not="有没有数据？";
Assertions.assertNull(cache.lookup(query_not));
```

14.3.5.2 Vector

Embedding

Embedding模块用于对Embedding模型API的适配封装，提供统一的接口快速地调用CSS模型embedding能力。

- 初始化：根据相应模型定义Embedding类。例如，使用华为CSS Embedding为：Embeddings.of(Embeddings.CSS);。

```
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embedding;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

// 初始化 Css Embedding
Embedding css = Embeddings.of(Embeddings.CSS);
```

- embedding单文本：把单个字符串转换为向量数据。（向量维度由模型确定）。

```
import java.util.List;

String text = "this is a test text.";
// embed query.
List<Float> embedding = css.embedQuery(text);
embedding.forEach(s -> System.out.println(s.toString()));
```

- embedding批量文档：把文档批量转换为向量数据。

```
import java.util.Arrays;
import java.util.List;

String text = "this is a test text.";
// embed documents.
```



```
List<List<Float>> embeddings = css.embedDocuments(Arrays.asList(text));
embeddings.forEach(s -> System.out.println(s.toString()));
```

Splitter

Splitter用于文档拆分解析，提供对文档数据进行拆分解析能力，支持pdf/doc/docx/ppt/pptx/xls/xlsx/png/jpg/jpeg/bmp/gif/tiff/webp/pcx/ico/psd等格式文档。

- 初始化：根据相应解析接口定义DocSplit类。以使用华为Pangu DocSplit为例：

```
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplit;
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplits;
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.config.SplitConfig;

// 初始化 pangudoc split（直接指定filePath和mode）
String filePath = "D:/test.doc";
DocSplit docPanguSplit =

DocSplits.of(SplitConfig.builder().splitName(DocSplits.PANGUDOC).filePath(filePath).mode("1").build()
);
// 初始化pangudoc split（通过配置文件指定filePath和mode）
DocSplit docPanguSplit = DocSplits.of(DocSplits.PANGUDOC);
```

  其中，**filePath**指的是需要解析的文档路径，**mode**为分割解析模式，具体定义如下：

  0 - 返回文档的原始段落，不做其他处理。

  1 - 根据标注的书签或目录分段，一般适合有层级标签的word文档。

  2 - 根据内容里的章节条分段，适合制度类文档。

  3 - 根据长度分段，默认按照500字拆分，会尽量保留完整句子。

- 文档解析

```
import com.alibaba.fastjson.JSONObject;
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplit;
import com.huaweicloud.pangu.dev.sdk.api.doc.splitter.DocSplits;
import com.huaweicloud.pangu.dev.sdk.client.css.doc.splitter.CSSDocResultResp;

DocSplit docPanguSplit = DocSplits.of(DocSplits.PANGUDOC);
CSSDocResultResp resp = docPanguSplit.loadDocument();
System.out.println(JSONObject.toJSONString(resp.getResult()));
```

向量库

向量库用于向量数据存储，并提供向量数据检索能力。

- 初始化，以使用华为CSS示例。

```
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoCss;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import java.util.Collections;

Vector cssVector = Vectors.of(Vectors.CSS,
    VectorStoreConfig.builder()
        .embedding(Embeddings.of(Embeddings.CSS))
        .indexName("you_index_name")
        .vectorFields(Collections.singletonList("description"))
        .textKey("name")
        .serverInfo(ServerInfoCss.builder().build())
        .build());
```

- 数据入库。

```
import com.huaweicloud.pangu.dev.sdk.vectorstore.bo.BulkData;
import java.util.ArrayList;
```



```
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

Map<String, Object> data = new HashMap<>();
data.put("name", "名称name");
data.put("description", "描述description");
final BulkData bulkData = new BulkData();
bulkData.setId("1");
bulkData.setData(data);
cssVector.addDocs(Collections.singletonList(bulkData));
```

- 数据检索。
  ```
  List<Document> docs = cssVector.similaritySearch("bar", 2);
  ```
- 数据清理。
  ```
  List<Document> docs = cssVector.clear();
  ```
- CSS插件模式（内部已集成Embedding， 支持多字段组合向量检索）。

  CSS插件模式需要提前手工创建索引（因索引中需要指定embdding/rank模型，SDK不能简单自动创建）。
  ```
  import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;
  import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
  import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoCss;
  import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
  import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
  import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;

  import java.util.Arrays;
  import java.util.HashMap;
  import java.util.List;
  import java.util.Map;

  Vector cssVector = Vectors.of(Vectors.CSS,
      VectorStoreConfig.builder()
          .indexName("your_index_name")
          .vectorFields(Arrays.asList("name", "description"))
          .textKey("name")
          .serverInfo(ServerInfoCss.builder().build())
          .build());
  ```

  通过vectorStoreConfig判断使用CSS的插件模式和非插件模式，如果配置了embedding模型，则使用非插件模式；否则使用插件模式。注意，在非插件模式下，vectorFields有且只有1个。

14.3.5.3 History

History缓存，用于存储历史对话信息，辅助模型理解上下文信息，历史消息对有固定窗口、消息摘要等策略。

- 初始化：消息记录支持不同的存储方式，如内存、DCS（Redis）、RDS（Sql）。
  ```
  import com.huaweicloud.pangu.dev.sdk.api.memory.config.MessageHistoryConfig;
  import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;
  import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoSql;
  import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.ChatMessageHistory;
  import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.RedisChatMessageHistory;
  import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.SqlChatMessageHistory;

  String sessionTag = "session_tag_test";
  // 内存型
  ChatMessageHistory chatMessage = new ChatMessageHistory();
  // Redis
  RedisChatMessageHistory redisChat = new RedisChatMessageHistory(MessageHistoryConfig.builder()
      .serverInfo(ServerInfoRedis.builder().build())
      .sessionTag("test-memory-0624")
  ```



```
    .build());
// Sql
SqlChatMessageHistory sqlChat = new SqlChatMessageHistory(

MessageHistoryConfig.builder().serverInfo(ServerInfoSql.builder().build()).sessionTag(sessionTag).buil
d());
```

- 添加、查找、删除数据。

```
//更新数据
chatMessage.addAIMessage("i am ai.");
chatMessage.addUserMessage("i am tester.");
// 查找数据
chatMessage.getMessages().toString();
// 清理
chatMessage.clear();
```

- 消息策略（windows-size）：将固定轮次历史对话信息，作为历史上下文信息。

```
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MemoryStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MessageHistoryConfig;
import com.huaweicloud.pangu.dev.sdk.llms.model.MessagePo;
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.RedisChatMessageHistory;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;
import com.huaweicloud.pangu.dev.sdk.memory.ConversationBufferMemory;

// 历史消息存储
RedisChatMessageHistory redisChat = new RedisChatMessageHistory(MessageHistoryConfig.builder()
    .serverInfo(ServerInfoRedis.builder().build())
    .sessionTag("test-memory")
    .build());
// 固定窗口策略
ConversationBufferMemory memory = new
ConversationBufferMemory(MemoryStoreConfig.builder().messageHistory(redisChat).build());

memory.getChatMessage().addUserMessage("This is me, the human");
memory.getChatMessage().addAIMessage("This is me, the AI");
System.out.print(MessagePo.toString(memory.getChatMessage().getMessages()));
memory.getChatMessage().clear();
```

- 消息策略（信息摘要）：将历史消息进行摘要后，作为历史上下文信息。

```
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MemoryStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MessageHistoryConfig;
import com.huaweicloud.pangu.dev.sdk.llms.model.MessagePo;
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.RedisChatMessageHistory;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.ServerInfoRedis;
import com.huaweicloud.pangu.dev.sdk.memory.ConversationSummaryMemory;

// 历史消息存储
RedisChatMessageHistory redisChat = new RedisChatMessageHistory(MessageHistoryConfig.builder()
    .serverInfo(ServerInfoRedis.builder().build())
    .sessionTag("test-memory")
    .build());
// 摘要策略
ConversationSummaryMemory memory = new
ConversationSummaryMemory(MemoryStoreConfig.builder().messageHistory(redisChat).llm(LLMs.of(L
LMs.PANGU)).build());

memory.getChatMessage().addUserMessage("This is me, the human");
memory.getChatMessage().addAIMessage("This is me, the AI");
System.out.print(MessagePo.toString(memory.getChatMessage().getMessages()));
memory.getChatMessage().clear();
```

14.3.6 Skill（技能）

14.3.6.1 基础问答

基础问答（SimpleSkill）提供基础的对话实现。

- 初始化。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.Template.KV;
import com.huaweicloud.pangu.dev.sdk.api.skill.base.SimpleSkill;

final String promptTemplate = "讲一个关于{{subject}}的笑话，字数{{count}}字以内";
SimpleSkill skill = new SimpleSkill(LLMs.of(LLMs.PANGU), promptTemplate);
```

- 问答。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;

// 问答
skill.execute(KV.of("subject", "哈士奇"), KV.of("count", "25"));

// 带参数问答
Map<String, Object> inputs = new HashMap<>();
inputs.put("subject", "哈士奇");
inputs.put("count", "25");
LLMParamConfig param = LLMParamConfig.builder().temperature(0.9).build();
skill.execute(inputs, param);
```

14.3.6.2 多轮对话

支持上下文记忆的多轮对话。

- 初始化。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.skill.ConversationSkill;

ConversationSkill skill = Skills.newConversationSkill(LLMs.of(LLMs.PANGU));
```

- 问答。

```
import com.huaweicloud.pangu.dev.sdk.memory.messagehistory.ChatMessageHistory;
import com.huaweicloud.pangu.dev.sdk.memory.ConversationBufferMemory;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.MemoryStoreConfig;

import org.junit.jupiter.api.Assertions;

// 定义存储策略
skill.setMemory(new
ConversationBufferMemory(MemoryStoreConfig.builder().windowSize(3).messageHistory(new
ChatMessageHistory()).build()));
String answer = skill.execute("中国首都是哪个城市？ ");
System.out.println(answer);
Assertions.assertTrue(answer.contains("北京"));
answer = skill.execute("它有什么好玩的地方？ ");
System.out.println(answer);
Assertions.assertTrue(answer.contains("故宫"));
```

14.3.6.3 文档问答

基于已有的知识库进行回答，包括stuff、refine和map-reduce策略。

- Stuff：将所有文档直接填充到prompt中，提给模型回答，适用于文档较少的场景。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

import java.util.List;
```



```
Vector cssVector = Vectors.of(Vectors.CSS,
        VectorStoreConfig.builder()
            .embedding(Embeddings.of(Embeddings.CSS))
            .indexName("test-stuff-document-062102")
            .build());

// 检索
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";
List<Document> docs = cssVector.similaritySearch(query, 4, 105);

// 问答
DocSkill docSkill = Skills.Document.newDocAskStuffSkill(LLMs.of(LLMs.PANGU));

System.out.println(docSkill.executeWithDocs(docs, query));
```

- Refine：基于首个文档问答，并循环后续文档来迭代更新答案。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

import java.util.List;

Vector cssVector = Vectors.of(Vectors.CSS,
        VectorStoreConfig.builder()
            .embedding(Embeddings.of(Embeddings.CSS))
            .indexName("test-stuff-document-062102")
            .build());

// 检索
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";
List<Document> docs = cssVector.similaritySearch(query, 4, 105);

// 问答
DocSkill docSkill = Skills.Document.newDocAskRefineSkill(LLMs.of(LLMs.PANGU));

System.out.println(docSkill.executeWithDocs(docs, query));
```

- Map-Reduce：先将文档单独进行摘要，将摘要后的文档再提交给模型。必要时，会循环迭代摘要。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

import java.util.List;

Vector cssVector = Vectors.of(Vectors.CSS,
        VectorStoreConfig.builder()
            .embedding(Embeddings.of(Embeddings.CSS))
            .indexName("test-stuff-document-062102")
            .build());

// 检索
String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";
List<Document> docs = cssVector.similaritySearch(query, 4, 105);

// 问答
DocSkill docSkill = Skills.Document.newDocAskMapReduceSkill(LLMs.of(LLMs.PANGU));

System.out.println(docSkill.executeWithDocs(docs, query));
```


14.3.6.4 文档摘要

基于已有的知识库进行摘要总结，包括stuff、refine和map-reduce策略。

- Stuff：将所有文档直接填充到prompt中，提给模型处理，适用于适用于文档较少的场景。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

import java.util.List;

Vector cssVector = Vectors.of(Vectors.CSS,
        VectorStoreConfig.builder()
            .embedding(Embeddings.of(Embeddings.CSS))
            .indexName("test-stuff-document-062102")
            .build());

// 检索
String query = "杜甫";
List<Document> docs = cssVector.similaritySearch(query, 4, 105);

// 摘要
DocSkill docSkill = Skills.Document.newDocSummarizeStuffSkill(LLMs.of(LLMs.PANGU));

System.out.println(docSkill.executeWithDocs(docs));
```

- Refine：基于首个文档摘要，循环后续文档来迭代更新。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

import java.util.List;

Vector cssVector = Vectors.of(Vectors.CSS,
        VectorStoreConfig.builder()
            .embedding(Embeddings.of(Embeddings.CSS))
            .indexName("test-stuff-document-062102")
            .build());

// 检索
String query = "杜甫";
List<Document> docs = cssVector.similaritySearch(query, 4, 105);

// 摘要
DocSkill docSkill = Skills.Document.newDocSummarizeRefineSkill(LLMs.of(LLMs.PANGU));

System.out.println(docSkill.executeWithDocs(docs));
```

- Map-Reduce：先将文档单独进行摘要，再将摘要后的文档提交给模型。必要时，会循环迭代摘要。

```
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.memory.bo.Document;
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;
```



```
import java.util.List;

Vector cssVector = Vectors.of(Vectors.CSS,
        VectorStoreConfig.builder()
            .embedding(Embeddings.of(Embeddings.CSS))
            .indexName("test-stuff-document-062102")
            .build());

// 检索
String query = "杜甫";
List<Document> docs = cssVector.similaritySearch(query, 4, 105);

// 摘要
DocSkill docSkill = Skills.Document.newDocSummarizeMapReduceSkill(LLMs.of(LLMs.PANGU));

System.out.println(docSkill.executeWithDocs(docs));
```

14.3.7 Agent（智能代理）

Agent（智能代理），用于对复杂任务的自动拆解与外部工具调用执行，一般包括任务规划、记忆系统、执行系统：

- 任务规划：将复杂目标任务分解为小的可执行子任务，通过评估、自我反思等方式提升规划成功率。
- 记忆系统：通过构建记忆模块去管理历史任务和策略，并让Agent结合记忆模块中相关的信息以获取最优化任务解决策略。
- 任务执行：能通过工具与外界发生联系并产生影响，工具可以自定义，包括查询信息、调用服务、网络搜索、文件管理、调用云服务等，通过Agent构建一个让LLM按照特定的规则迭代运行的Prompt，直到任务完成或者达到终止条件（如设置迭代次数）。

14.3.7.1 实例化 Tool

Tool分为StaticTool（静态工具）和DynamicTool（动态工具）两类。静态工具需要开发者事先定义好，即在编译期定义与实例化。对于动态工具，开发者可以在系统运行时动态构建，即在运行态定义与实例化。

StaticTool（静态工具）

静态工具可以通过注解的方式新增，在run接口中实现工具的功能，例如：

```
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentToolParam;
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;
import lombok.Data;

@AgentTool(toolId = "reserve_meeting_room", toolDesc = "预订会议室", toolPrinciple = "请在需要预订会议室时调用此工具",
    inputDesc = "会议开始结束时间，会议室", outPutDesc = "预订会议室的结果")
public class ReserveMeetingRoom extends StaticTool<ReserveMeetingRoom.InputParam, String> {
    @Override
    public String run(InputParam input) {
        return String.format("%s到%s的%s已预订成功", input.start, input.end, input.meetingRoom);
    }
    @Data
    public static class InputParam {
        @AgentToolParam(description = "会议开始时间，格式为HH:mm")
        private String start;
        @AgentToolParam(description = "会议结束时间，格式为HH:mm")
        private String end;
        @AgentToolParam(description = "会议室")
```



```
    private String meetingRoom;
  }
}
```

@AgentTool注解说明：

- toolId。表示工具的标识，建议为英文且与实际工具含义匹配，在同一个Agent中唯一。
- toolDesc。工具的描述，**为重要参数**，尽可能的准确简短描述工具的用途。
- toolPrinciple。表示何时使用该工具，**为重要参数**。该描述直接影响LLM对工具使用的判断，尽量描述清楚。如果Agent实际执行效果不符合预期，可以调整。
- inputDesc。工具的入参描述，**为重要参数**。该描述直接影响LLM对入参的提取，尽量描述清楚。如果Agent实际执行效果不符合预期，可以调整。
- outputDesc。工具的出参描述，当前对Agent的表现无重要影响。

如果输入输出参数为复杂类型，则需要通过AgentToolParam注解定义复杂类型的参数描述，此时inputDesc、outputDesc可以填空字符串，但仍然建议给出简要的描述。当前版本不支持复杂类型中再嵌套复杂类型，只支持基本类型：String、Number、Boolean，建议参数数量不超过5个。

@AgentToolParam注解说明：

- description。参数的描述，**为重要参数**。该描述直接影响LLM对入参的提取，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。
- required。是否为可选参数。

注意：字段的命名需要以小写字母开头，否则在转换成标准的Json schema时会出现问题，导致模型精度受到影响。

上例中的InputParam为一个复杂的入参，如果工具的入参为基本类型，则不需要再额外定一个结构体，例如：

```
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;

@AgentTool(toolId = "capital", toolDesc = "资产注册查询", toolPrinciple = "请在需要查询各个公司的资产注册情况时调用此工具",
    inputDesc = "需要查询的公司名称，一次只支持查询一家公司", outPutDesc = "公司的资产注册规模")
public class CapitalQuery extends StaticTool<String, String> {
    @Override
    public String run(String input) {
        switch (input) {
            case "x公司":
                return "x亿人民币";
            case "y公司":
                return "y亿人民币";
            case "z公司":
                return "z亿人民币";
            default:
                return "未知";
        }
    }
}
```

DynamicTool（动态工具）

- 动态工具可以在业务运行态动态新增或修改，例如：

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.huaweicloud.pangu.dev.sdk.api.tool.DynamicTool;
```



```
private static DynamicTool buildAddTool() {
    final DynamicTool addTool = new DynamicTool();
    addTool.setToolId("add");
    addTool.setToolDesc("加法运算");
    addTool.setToolPrinciple("请在需要做两个整型的加法运算时调用此工具");
    addTool.setInputDesc("加法输入");
    addTool.setOutputDesc("加法运算的结果");
    addTool.setInputSchema(
        "{\"type\":\"object\",\"properties\":{\"a\":{\"type\":\"integer\",\"description\":\"加法运算的数字
\"},\"b\":{\"type\":\"integer\",\"description\":\"加法运算的数字\"}},\"required\":[\"a\",\"b\"]}");
    addTool.setOutputSchema("{\"type\":\"integer\"}");
    addTool.setFunction(s -> {
        final JSONObject jsonObject = JSON.parseObject(s);
        final Integer a = jsonObject.getInteger("a");
        final Integer b = jsonObject.getInteger("b");
        return a + b;
    });
    return addTool;
}
```

其中，toolId、toolDesc、toolPrinciple、inputDesc、outputDesc的定义与说明与静态工具相同。

inputSchema：工具入参的JsonSchema。

outputSchema：工具出参的JsonSchema。

- 复杂对象JsonSchema示例：

```
{
  "type": "object",
  "properties": {
    "a": {
      "description": "加法运算的数字",
      "type": "integer"
    },
    "b": {
      "description": "加法运算的数字",
      "type": "integer"
    }
  },
  "required": ["a","b"]
}
```

- 简单对象JsonSchema示例：

```
{
  "type": "integer"
}
```

- 枚举对象JsonSchema示例：

```
{
  "type": "object",
  "properties": {
    "chargeType": {
      "type": "string",
      "enum": ["Alipay", "Wechat", "Bankcard"],
      "description": "充值类型"
    },
    "amount": {
      "type": "integer",
      "description": "充值金额"
    }
  },
  "required": ["chargeType", "amount"]
}
```

14.3.7.2 实例化 Agent

- Agent实例化过程包括注册LLM和注册工具两个部分。

```
import com.huaweicloud.pangu.dev.sdk.agent.ReactPanguAgent;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
```



```
public static void initAgent() {
    LLM llm = LLMs.of(LLMs.PANGU,
        LLMConfig.builder()
            .llmParamConfig(LLMParamConfig.builder().temperature(0.01).build())
            .llmModuleConfig(
                LLMModuleConfig.builder().systemPrompt("今天的日期为
2024.7.30").moduleVersion("N2_agent_v2").build())
            .build());
    Agent panguAgent = new ReactPanguAgent(llm);

    agent.setMaxIterations(5);
    agent.addTool(new ReverseTool());
    agent.addTool(new AddTool());
    agent.addTool(new SearchTool());
}
```

- 静态工具和动态工具的注册方式相同，通过addTool接口进行注册。
- 通过setMaxIterations可以设置最大迭代次数，控制Agent子规划的最大迭代步数，防止无限制的迭代或出现死循环情况。
- Agent使用的模型必须为Pangu-NLP-N2-Default模型，或其衍生模型，使用通用模型或其他模型无法运行。当前的moduleVersion需要配置为“N2_agent_v2”，如上例所示，因此模型的url要配置为Pangu-NLP-N2-Default模型的地址。
- 支持注册开源模型，开源模型的定义可参考**开源模型**。

```
final LLM llm = LLMs.of(LLMs.OPENAI,
    LLMConfig.builder()
        .llmParamConfig(LLMParamConfig.builder().temperature(0.0).build())
        .llmModuleConfig(
            LLMModuleConfig.builder().systemPrompt("今天的日期为
2024.7.30").moduleVersion("expert_q4").build())
        .build());
Agent agent = new ReactAgent(llm);
```

14.3.7.3 运行 Agent

在给出的示例中，Agent中预置了2个工具，分别为：

meeting_room_status_query：查询会议室的状态，是否被预定或者正在使用中。

reserve_meeting_room：预定会议室。

- 单轮执行：

  调用run接口运行一个Agent：

  ```
  panguAgent.run("帮我定个今天下午3点到8点的A02会议室");
  ```

  Agent的运行时会进行自我迭代，并且选择合适的工具，在日志中打印最终的执行结果：

  ```
  用户: 帮我定个今天下午3点到8点的A02会议室
  助手: A02会议室在今天下午3点到8点已经被预定了。是否需要为您预定其他时间段或者其他会议室?
   - 步骤1:
     思考:好的，我需要先查询A02会议室今天下午3点到8点的预定状态。使用meeting_room_status_query工具进行查询。
     行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-07 15:00\", \"end\": \"2024-05-07 20:00\", \"meetingRoom\": \"A02\"}"
     工具返回:in use
   - 步骤2
     答复:"A02会议室在今天下午3点到8点已经被预定了。是否需要为您预定其他时间段或者其他会议室？"
  ```

- 多轮执行

  ```
  List<ConversationMessage> messages = new ArrayList<>();
  messages.add(ConversationMessage.builder().role(Role.USER).content("定个2点-4点的会议，A01会议室").build());
  messages.add(
      ConversationMessage.builder().role(Role.ASSISTANT).content("已为您预定 A01会议室，时间为2024
  ```



```
年5月7日下午2点到4点。").build());
messages.add(ConversationMessage.builder().role(Role.USER).content("再定一个明天8点到9点的会议室
").build());
panguAgent.run(messages);
```

运行结果示例：

```
用户: 定个2点-4点的会议，A01会议室
助手: 已为您预定 A01会议室，时间为2024年5月7日下午2点到4点。
用户: 再定一个明天8点到9点的会议室
助手: 已为您预定 A01会议室，时间为2024年5月8日早上8点到9点。
 - 步骤1:
   思考:好的，让我先查询一下 A01会议室在2024年5月8日8点到9点的状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-08 08:00\", \"end\":
\"2024-05-08 09:00\", \"meetingRoom\": \"A01\"}"
   工具返回:available
 - 步骤2:
   思考: A01会议室在2024年5月8日8点到9点是空闲的，可以预定。现在为您预定。
   行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-08 08:00\", \"end\":
\"2024-05-08 09:00\", \"meetingRoom\": \"A01\"}"
   工具返回:2024-05-08 08:00到2024-05-08 09:00的A01已预定成功
 - 步骤3
   答复:"已为您预定 A01会议室，时间为2024年5月8日早上8点到9点。 "
```

- 多轮执行增强

  上述的例子中实际运行时只提供给模型多轮的对话，并没有提供工具执行的过程，有概率会出现模型不实际调用工具的情况。为了让大模型的效果更好，可以传入agentSession。

```
AgentSession agentSession = AgentSessionHelper.initAgentSession("定个2点-4点的会议，A01会议室");
agentSession = panguAgent.run(agentSession);

agentSession.getMessages()
    .add(ConversationMessage.builder().role(Role.USER).content("再定一个明天8点到9点的会议室
").build());
panguAgent.run(agentSession);
```

第一次运行Agent，panguAgent.run(agentSession)结果：

```
用户: 定个2点-4点的会议，A01会议室
助手: 已为您预定 A01会议室，时间为2024年5月7日下午2点到4点。
 - 步骤1:
   思考:好的，我需要先查询 A01会议室在今天下午2点到4点的状态。现在调用
meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-07 14:00\", \"end\":
\"2024-05-07 16:00\", \"meetingRoom\": \"A01\"}"
   工具返回:available
 - 步骤2:
   思考: A01会议室在今天下午2点到4点是空闲的。现在我将为您预定 A01会议室。调用
reserve_meeting_room 工具进行会议室预定。
   行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-07 14:00\", \"end\":
\"2024-05-07 16:00\", \"meetingRoom\": \"A01\"}"
   工具返回:2024-05-07 14:00到2024-05-07 16:00的A01已预定成功
 - 步骤3
   答复:"已为您预定 A01会议室，时间为2024年5月7日下午2点到4点。 "
```

第二次运行Agent，panguAgent.run(agentSession)结果：

```
用户: 定个2点-4点的会议，A01会议室
助手: 已为您预定 A01会议室，时间为2024年5月7日下午2点到4点。
 - 步骤1:
   思考:好的，我需要先查询 A01会议室在今天下午2点到4点的状态。现在调用
meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-07 14:00\", \"end\":
\"2024-05-07 16:00\", \"meetingRoom\": \"A01\"}"
   工具返回:available
 - 步骤2:
   思考: A01会议室在今天下午2点到4点是空闲的。现在我将为您预定 A01会议室。调用
reserve_meeting_room 工具进行会议室预定。
   行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-07 14:00\", \"end\":
\"2024-05-07 16:00\", \"meetingRoom\": \"A01\"}"
   工具返回:2024-05-07 14:00到2024-05-07 16:00的A01已预定成功
 - 步骤3
   答复:"已为您预定 A01会议室，时间为2024年5月7日下午2点到4点。 "
```



```
用户: 再定一个明天8点到9点的会议室
助手: 已为您预定 A01会议室，时间为2024年5月8日早上8点到9点。
 - 步骤1:
   思考:好的，我需要先查询会议室在明天早上8点到9点的状态。现在调用 meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-08 08:00\", \"end\": \"2024-05-08 09:00\", \"meetingRoom\": \"A01\"}"
   工具返回:available
 - 步骤2:
   思考: A01会议室在明天早上8点到9点是空闲的。现在我将为您预定 A01会议室。调用 reserve_meeting_room 工具进行会议室预定。
   行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-08 08:00\", \"end\": \"2024-05-08 09:00\", \"meetingRoom\": \"A01\"}"
   工具返回:2024-05-08 08:00到2024-05-08 09:00的A01已预定成功
 - 步骤3
   答复:"已为您预定 A01会议室，时间为2024年5月8日早上8点到9点。 "
```

在第二次运行Agent时，包含第一次运行的所有工具调用细节。

agentSession相当于Agent的会话Memory。一般情况下，需要将agentSession对象在外部持久化，在每一轮会话传入agentSession对象中的sessionId，下面的示例代码用一个map对象模拟外部的持久化：

```
/**
 * 在生产环境下，agentSession建议在外部持久化，而不是在内存中
 * 如果使用AssistantAPI，华为会提供持久化能力，不需要自行实现
 */
private static final Map<String, AgentSession> agentSessionMap = new HashMap<>();

@Test
void test() {
    String sessionId = "sessionId_1";
    panguAgent.run(getAgentSession(sessionId, "定个2点-4点的会议，A01会议室"));
    panguAgent.run(getAgentSession(sessionId, "再定一个明天8点到9点的会议室"));
}

AgentSession getAgentSession(String sessionId, String userMessage) {
    AgentSession agentSession = agentSessionMap.get(sessionId);
    if (agentSession == null) {
        agentSession = AgentSessionHelper.initAgentSession(userMessage);
        agentSession.setSessionId(sessionId);
        agentSessionMap.put(sessionId, agentSession);
    } else {
        agentSession.getMessages()
                .add(ConversationMessage.builder().role(Role.USER).content(userMessage).build());
    }
    return agentSession;
}
```

- 单步执行

  有时并不希望Agent完全自主执行，在某些关键节点，让用户先进行确认，确认后再执行，或者用户对模型的结果有异议或者想法有变化，想对当前结果进行更改。此时可以单步运行Agent：

  ```
  /**
   * 单步执行Agent，提供干预能力
   *
   * @param agentSession 包括初始状态，以及执行步骤间的agentSession，可以使用AgentSessionHelper类辅助处理
   * @return Agent执行的结果
   */
  AgentSession runStep(AgentSession agentSession);
  ```

  完整示例如下：

  ```
  AgentSession agentSession = AgentSessionHelper.initAgentSession("定个2点-4点的会议");
  agentSession = panguAgent.runStep(agentSession);
  log.info(AgentSessionHelper.printPlan(agentSession));

  log.info("Agent status = {}, is {}, do not call tool", agentSession.getAgentSessionStatus(),
      AgentSessionStatus.FINISHED);
  AgentSessionHelper.updateAssistantMessage(agentSession, true);
  ```



```
// 用户多轮对话
AgentSessionHelper.addUserMessage(agentSession, "A01会议室");
agentSession = panguAgent.runStep(agentSession);
log.info(AgentSessionHelper.printPlan(agentSession));

// 从agentSession中取出要调用的工具
final AgentAction currentAction = agentSession.getCurrentAction();
final String action = currentAction.getActionTools().get(0).getAction();
final Object actionInput = currentAction.getActionTools().get(0).getActionInput();
log.info("Agent status = {}, not {}, call tool {}, input = {}", agentSession.getAgentSessionStatus(),
    AgentSessionStatus.FINISHED, action, actionInput);

// 调用工具，这里为调用示例，由于已经给出tool_id和调用参数，实际调用可以有多种形式
final Tool tool = panguAgent.getTool(action);
final Object result = tool.runFromJson(actionInput.toString());

// 提交工具调用结果，并继续执行
AgentSessionHelper.setFirstToolOutput(agentSession, result.toString());
agentSession = panguAgent.runStep(agentSession);
log.info(AgentSessionHelper.printPlan(agentSession));

// 如果对工具调用参数有调整，可以进行反馈
AgentSessionHelper.setUserFeedback(agentSession, "会议室更换为A02");
agentSession = panguAgent.runStep(agentSession);
log.info(AgentSessionHelper.printPlan(agentSession));
```

上述例子中，第一次printPlan打印的结果为：

```
用户: 定个2点-4点的会议
助手:
 - 步骤1:
   思考:好的，请问您想预定哪一个会议室?
```

由于缺少参数，因此模型开始追问，此时构造了用户消息，回答了模型追问的问题。可以通过AgentSession的状态来判断是否是追问闲聊，或是发起工具调用，此时AgentSession的状态为FINISHED，所以为追问或闲聊。打印的日志为：

```
Agent status = FINISHED, is FINISHED, do not call tool
```

第二次printPlan打印的结果为：

```
用户: 定个2点-4点的会议
助手: 好的，请问您想预定哪一个会议室?
 - 步骤1:
   思考:好的，请问您想预定哪一个会议室?
用户: A01会议室
助手:
 - 步骤1:
   思考:请稍等，我需要先查询 A01会议室在今天下午2点到4点的状态。现在调用
meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-11 14:00\", \"end\":
\"2024-05-11 16:00\", \"meetingRoom\": \"A01\"}"
```

此时，由于采用了单步调用，因此不会自动执行工具，打印的日志为：

```
Agent status = RUNNING, not FINISHED, call tool meeting_room_status_query, input = {"start":
"2024-05-11 14:00", "end": "2024-05-11 16:00", "meetingRoom": "A01"}
```

同样的，可以通过AgentSession的status判断此时是否需要调用工具，如果需要调用工具，可以从currentAction中获取需要调用的工具信息。

当提交了工具调用结果后，第三次printPlan打印的结果为：

```
用户: 定个2点-4点的会议
助手: 好的，请问您想预定哪一个会议室?
 - 步骤1:
   思考:好的，请问您想预定哪一个会议室?
用户: A01会议室
助手:
 - 步骤1:
   思考:请稍等，我需要先查询 A01会议室在今天下午2点到4点的状态。现在调用
meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-11 14:00\", \"end\":
\"2024-05-11 16:00\", \"meetingRoom\": \"A01\"}"
   工具返回:available
```



```
 - 步骤2:
   思考: A01会议室在今天下午2点到4点是空闲的。现在为您预定 A01会议室。调用
reserve_meeting_room 工具进行会议室预定。
   行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-11 14:00\", \"end\":
\"2024-05-11 16:00\", \"meetingRoom\": \"A01\"}"
```

可以看到，Agent又向下执行了一步，此时需要提交reserve_meeting_room的调用结果，提交之后，执行预期结束。为了说明用户反馈的功能，选择了调用setUserFeedback，进行反馈，第四次printPlan打印的结果为：

```
用户: 定个2点-4点的会议
助手: 好的，请问您想预定哪一个会议室?
 - 步骤1:
   思考:好的，请问您想预定哪一个会议室?
用户: A01会议室
助手:
 - 步骤1:
   思考:请稍等，我需要先查询 A01会议室在今天下午2点到4点的状态。现在调用
meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-11 14:00\", \"end\":
\"2024-05-11 16:00\", \"meetingRoom\": \"A01\"}"
   工具返回:available
 - 步骤2:
   思考: A01会议室在今天下午2点到4点是空闲的。现在为您预定 A01会议室。调用
reserve_meeting_room 工具进行会议室预定。
   行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-11 14:00\", \"end\":
\"2024-05-11 16:00\", \"meetingRoom\": \"A01\"}"
   用户反馈:会议室更换为A02
 - 步骤3:
   思考:好的，我将为您更换为 A02会议室。现在查询 A02会议室在今天下午2点到4点的状态。调用
meeting_room_status_query 工具查询会议室状态。
   行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-11 14:00\", \"end\":
\"2024-05-11 16:00\", \"meetingRoom\": \"A02\"}"
```

可以看到，Agent继续向下执行了一步，会议室改为了A02，需要重新查询A02的会议室状态。

- Agent结果总结

  有时需要Agent对之前的思考、行动回答用户的最终问题，如果对模型返回的结果不满意，可以选择使用AgentSessionSkill解决：

```
final AgentSession agentSession = agent.run("我的数学成绩和语文成绩分别是多少");
final LLM llm = LLMs.of(LLMs.PANGU, LLMConfig.builder()
    .llmModuleConfig(LLMModuleConfig.builder()
        .url(
            "https://pangu.cn-southwest-2.myhuaweicloud.com/v1/infers/N2-基础模型的调用地址")
        .build())
    .build());
final AgentSessionSkill agentSessionSkill = new AgentSessionSkill(llm);
System.out.println("最终结果: " + agentSessionSkill.summary(agentSession));
```

  运行结果示例：

```
用户: 我的数学成绩和语文成绩分别是多少
助手：您的语文成绩也是99分
 - 步骤1:
   思考:好的，我需要调用 queryScore 工具来查询您的数学成绩。
   行动:使用工具[queryScore],传入参数"{\"subjectName\":\"数学\"}"
   工具返回:99
 - 步骤2:
   思考:好的，我需要调用 queryScore 工具来查询您的语文成绩。
   行动:使用工具[queryScore],传入参数"{\"subjectName\":\"语文\"}"
   工具返回:99
 - 步骤3
   思考:您的语文成绩也是99分。

最终结果: 您的数学成绩是99分，您的语文成绩也是99分。
```

  AgentSessionSkill使用的大模型建议为N2-基础模型或者其他同等类型的模型。

- 多Agent组合

  有时需要多个Agent配合完成任务，可以通过子Agent作为tool方式实现：


```
// 定义子Agent（tool形式）

import com.huaweicloud.pangu.dev.sdk.agent.ReactPanguAgent;
import com.huaweicloud.pangu.dev.sdk.api.agent.Agent;
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMModuleConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;

import java.text.SimpleDateFormat;
import java.util.Date;

@AgentTool(toolId = "meeting_agent", toolDesc = "预定会议Agent", toolPrinciple = "请在需要预定会议
室时调用此工具",
    inputDesc = "预定会议室请求", outPutDesc = "预定会议室的结果")
public class MeetingAgent extends StaticTool<String, String> {
    private static Agent panguAgent;

    public MeetingAgent() {
        final String customSystemPrompt = "你是一个会议室预定助手，今天的日期是" + new
SimpleDateFormat("yyyy年MM月dd日").format(new Date());
        final LLMConfig config = LLMConfig.builder()
                .llmParamConfig(LLMParamConfig.builder().temperature(0.01).withPrompt(true).build())
                .llmModuleConfig(LLMModuleConfig.builder().systemPrompt(customSystemPrompt).build()
)
                .build();

        panguAgent = new ReactPanguAgent(LLMs.of(LLMs.PANGU, config));
        panguAgent.setMaxIterations(5);
        panguAgent.addTool(new MeetingRoomStatusQuery());
        panguAgent.addTool(new ReserveMeetingRoom());
    }

    @Override
    public String run(String input) {
        return panguAgent.run(input).getCurrentMessage().getContent();
    }
}

@Test
void test6() {
    final LLMConfig config = LLMConfig.builder()
        .llmParamConfig(LLMParamConfig.builder().temperature(0.01).withPrompt(true).build())
        .build();

    // Agent嵌套，将预定会议室Agent作为一个子Agent添加到agent中
    Agent agent = new ReactPanguAgent(LLMs.of(LLMs.PANGU, config));
    agent.addTool(new MeetingAgent());

    agent.run("帮我定个今天下午3点到8点的A05会议室");
    // 子agent是实际做事的agent
    /**
     * 用户: 预定今天下午3点到8点的A05会议室
     * 助手: 已为您预定2024年05月10日下午3点到8点的A05会议室。请准时参加会议。
     * - 步骤1:
     * 思考:好的，我需要先查询A05会议室在今天下午3点到8点的状态。如果会议室可用，我将为您预定。
现在我将调用meeting_room_status_query工具查询会议室状态。
     * 行动:使用工具[meeting_room_status_query],传入参数"{\"start\": \"2024-05-10 15:00\", \"end\":
\"2024-05-10 20:00\",
     * \"meetingRoom\": \"A05\"}"
     * 工具返回:available
     * - 步骤2:
     * 思考:A05会议室今天下午3点到8点是可用的。接下来我将调用reserve_meeting_room工具为您预定
A05会议室。
     * 行动:使用工具[reserve_meeting_room],传入参数"{\"start\": \"2024-05-10 15:00\", \"end\":
\"2024-05-10 20:00\",
     * \"meetingRoom\": \"A05\"}"
```



```
     * 工具返回:2024-05-10 15:00到2024-05-10 20:00的A05已预定成功
     * - 步骤3
     * 答复:"已为您预定2024年05月10日下午3点到8点的A05会议室。请准时参加会议。"
     */

    // agent只负责分发任务给子agent
    /**
     * 用户: 帮我定个今天下午3点到8点的A05会议室
     * 助手: 已为您预定2024年05月10日下午3点到8点的A05会议室。请准时参加会议。
     * - 步骤1:
     * 思考:好的，我需要调用meeting_agent工具来预定会议室。该工具需要一个字符串类型的必须参数arg。现在我将调用meeting_agent工具预定会议室。
     * 行动:使用工具[meeting_agent],传入参数"{\"arg\": \"预定今天下午3点到8点的A05会议室\"}"
     * 工具返回:已为您预定2024年05月10日下午3点到8点的A05会议室。请准时参加会议。
     * - 步骤2
     * 答复:"已为您预定2024年05月10日下午3点到8点的A05会议室。请准时参加会议。"
     */
}
```

14.3.7.4 监听 Agent

一次Agent的响应如果涉及到多个任务的分解，往往会执行比较长的时间，此时可以对agent的执行过程进行监听。

AgentListener的定义如下：

```
public interface AgentListener {
    /**
     * Session启动时调用
     *
     * @param agentSession AgentSession
     */
    default void onSessionStart(AgentSession agentSession) {
    }
    /**
     * Session迭代过程中调用
     *
     * @param agentSession AgentSession
     */
    default void onSessionIteration(AgentSession agentSession) {
    }
    /**
     * onSessionIteration调用结束后，检查Agent是否需要终止，如果需要终止，则返回true，默认不终止
     * 可以在终止前对agentSession进行修改，如：修改agent的finalAnswer
     *
     * @param agentSession AgentSession
     */
    default boolean onCheckInterruptRequirement(AgentSession agentSession) {
        return false;
    }
    /**
     * Session结束时调用
     *
     * @param agentSession AgentSession
     */
    default void onSessionEnd(AgentSession agentSession) {
    }
}
```

定义一个监听器

通过实现AgentListener定义一个监听器：

```
import com.huaweicloud.pangu.dev.sdk.api.agent.AgentListener;

public static class TestAgentListener implements AgentListener {
    @Override
```



```
    public void onSessionStart(AgentSession agentSession) {
        System.out.println(agentSession);
    }
    @Override
    public void onSessionIteration(AgentSession agentSession) {
        System.out.println(agentSession);
    }
    @Override
    public void onSessionEnd(AgentSession agentSession) {
        System.out.println(agentSession);
    }
}
```

上述代码分别对应了Agent的开始、中间过程、结束阶段。

为 Agent 添加一个监听器

通多调用Agent的addListener接口添加一个监听器：

```
import com.huaweicloud.pangu.dev.sdk.agent.ReactPanguAgent;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;

public static void initAgent() {
    ReactPanguAgent agent = new ReactPanguAgent(LLMs.of(LLMs.PANGU));
    final TestAgentListener testAgentListener = new TestAgentListener();
    agent.addListener(testAgentListener);
}
```

其中，listener会在Agent运行时生效。

监听的对象

监听的对象为一个AgentSession：

```
public class AgentSession {
    /**
     * UUID，在一个session内唯一
     */
    private String sessionId;
    /**
     * Agent返回的最终答案（最后一个AgentAction的输出）
     */
    private String finalAnswer = "";
    /**
     * 本次session的用户query
     */
    private List<ConversationMessage> messages;
    /**
     * 历史Action
     */
    private List<AgentAction> historyAction;
    /**
     * 当前Action
     */
    private AgentAction currentAction;
    /**
     * Agent状态
     */
    private AgentSessionStatus agentSessionStatus;
}
```

AgentAction包含Agent的工具选择、工具执行结果、思考等信息，AgentSessionStatus为一个枚举，包含Agnet的执行状态。建议直接对Agent的run接口的返回进行修改，以控制Agent的行为。如果想控制中间过程，可以对Agent的runStep的返回进行修改。

通过监听终止 Agent 的执行

当需要在Agent的执行过程中终止执行时，除了通过setMaxIterations设置Agent的最大迭代次数，也可以通过实现监听器的onCheckInterruptRequirement实现。

```
agent.addListener(new AgentListener() {
    @Override
    public boolean onCheckInterruptRequirement(AgentSession agentSession) {
        final AgentAction currentAction = agentSession.getCurrentAction();
        // 如果当前的action为capital，则返回capital的工具调用原始返回值
        if ("capital".equals(currentAction.getAction())) {
            agentSession.setFinalAnswer(currentAction.getObservation());
            return true;
        }
        return false;
    }
});
```

上述例子中，当满足if判断条件时，会直接终止Agent的执行，且finalAnswer被设置为工具的原始返回值。

14.3.7.5 Agent 效果优化

如果Agent出现无法正确调用工具的情况，可以尝试一些prompt优化技术提升效果。

- 优化System prompt

  提示财务报销助手依赖的必要信息，如用户名称等基础信息：

  ```
  final String customSystemPrompt =
      "你是财务报销助手。当需要用户反馈信息时，尽可能提示用户名称等原始信息。今天的日期是" + new
  SimpleDateFormat("yyyy年MM月dd日").format(new Date());
  final LLMConfig config = LLMConfig.builder()
      .llmParamConfig(LLMParamConfig.builder().temperature(0.01).withPrompt(true).build())
      .llmModuleConfig(LLMModuleConfig.builder().systemPrompt(customSystemPrompt).build())
      .build();
  ```

- 优化工具描述

  工具依赖的信息，可以通过其他工具获取时，增加关联关系提示：

  ```
  @AgentTool(toolId = "query_reimbursement_limit", toolDesc = "通过用户ID、用户单据、用户最大报销
  比例获取用户报销额度", toolPrinciple = "请在有用户ID、用户单据、用户最大报销比例的情况下查询用户
  最大报销额度时调用此工具。需要先分别调用query_receipt工具查询用户单据和
  query_reimbursement_ratio工具查询最大报销比例。",
          inputDesc = "用户ID、用户单据、用户报销最大比例", outPutDesc = "用户最大报销额度")
  public class GetReimbursementLimitTool  extends StaticTool<GetReimbursementLimitTool.InputParam,
  String> {
  ```

14.3.7.6 Agent 流式输出

Agent用于工具调用场景，与普通的LLM流式输出相比，提供了事件流的封装。消息内容、工具调用等通过不同的事件类型区分。

通过如下接口为Agent添加流式输出的回调：

```
/**
 * 设置流式接口回调函数
 *
 * @param streamAgentCallback 事件流回调
 */
void setStreamCallback(StreamAgentCallBack streamAgentCallback);
```

StreamAgentCallBack实现示例：

```
private class StreamAgentCallBackImpl implements StreamAgentCallBack {
    @Override
    public void onEventReceived(AgentEvent agentEvent) {
```



```
        log.debug("-----> stream event: {}", agentEvent.getType().getEventType());
        log.debug("-----> stream data: {}", JSON.toJSONString(agentEvent));
    }
}
```

事件类型列表如下：

```
    /**
     * 消息创建
     */
    MESSAGE_CREATED("session.message.created"),

    /**
     * action创建
     */
    ACTION_CREATED("session.action.created"),

    /**
     * 流式消息
     */
    ACTION_MESSAGE_DELTA("session.action.message.delta"),

    /**
     * 流式工具调用
     */
    ACTION_TOOL_DELTA("session.action.tool.delta"),

    /**
     * action结束
     */
    ACTION_COMPLETED("session.action.completed"),

    /**
     * 消息结束
     */
    MESSAGE_COMPLETED("session.message.completed");
```

14.3.7.7 Tool Retriever

Agent在实际生产应用中往往涉及到的工具数量较多，如果把所用的工具全部添加至Agent会产生如下问题：

- 占用大量输入token。
- 和问题无关的工具太多，影响模型的判断。

通过Tool Retriever可以解决上述问题，其原理是在Agent运行前，先从所有可用的工具中选择与问题最相关的工具，再交给Agent去处理，示例如下：

- 定义一个Tool Retriever：

  ```
  final List<Tool> toolList =
      Arrays.asList(new MetricQuery(), new ReserveMeeting(), new ReserveMeetingRoom(), new
  StuffQuery());

  // 新增InMemoryToolProvider，添加工具集
  final InMemoryToolProvider inMemoryToolProvider = new InMemoryToolProvider();
  inMemoryToolProvider.add(toolList);

  // 初始化CSSToolRetriever
  final CSSToolRetriever cssToolRetriever = new CSSToolRetriever(inMemoryToolProvider,
      VectorStoreConfig.builder()
          .indexName(TestConstant.CSS_TOOL_RETRIEVER_INDEX)
          .vectorFields(Arrays.asList("name", "description"))
          .build());
  ```

  定义一个ToolRetriever包含ToolProvider和向量数据库配置2个参数。其中，ToolProvider的作用为根据工具检索的结果组装工具。

上述例子使用了一个简单的InMemoryToolProvider，InMemoryToolProvider的原理为将完整的工具存入内存，再根据工具检索的结果（toolId）将其从内存中取出。一般来说，ToolProvider将由用户自定义，将在后续示例中说明。

此外，上述例子使用的向量数据库配置指定索引名称，以及使用name和description作为向量化字段，因此工具入库时，会将工具的name和description进行向量化，并在后续的检索中生效。

注意，上述toolList中包含的工具在SDK中并不存在，需要替换成实际的工具。

- 向ToolRetriever中添加工具：

```
// 添加工具
cssToolRetriever.addTools(toolList);
```

工具添加后，会存储在向量库的索引中，并将指定的字段向量化。

- 从ToolRetriever中查找工具：

```
// 查找工具
List<Tool> result = cssToolRetriever.search("预订会议室", 2);
```

返回的result中，包含与预订会议室最相关的工具。搜索支持topK和阈值2个参数，例如上例指定topK=2，则最多返回2个工具。

- 从ToolRetriever中删除工具：

```
// 删除工具
cssToolRetriever.remove(Collections.singletonList("a_tool_id"));
```

以上为一个比较基础的用法，在实际使用过程中会有更加灵活的场景，可以通过自定义ToolProvider的方式解决。

- 自定义ToolProvider：

```
// 初始化CSSToolRetriever，使用ToolProviderWithMetadata作为ToolProvider
final CSSToolRetriever cssToolRetriever = new CSSToolRetriever(new ToolProviderWithMetadata(),
    VectorStoreConfig.builder()
        .indexName(TestConstant.CSS_TOOL_RETRIEVER_INDEX)
        .vectorFields(Arrays.asList("name", "description"))
        .build());
```

其中，ToolProviderWithMetadata为自定义ToolProvider：

```
private static class ToolProviderWithMetadata implements ToolProvider {
    @Override
    public List<Tool> provide(List<RetrievedTool> retrievedTools, String query) {
        doSomeFilter(retrievedTools, query);
        return retrievedTools.stream().map((Function<RetrievedTool, Tool>) retrievedTool -> {
            final DynamicTool tool = new DynamicTool();
            tool.setToolId(retrievedTool.getToolId());
            final Map<String, Object> toolMetadata = retrievedTool.getToolMetadata();
            tool.setToolDesc(toolMetadata.get("description").toString());
            tool.setToolPrinciple(toolMetadata.get("description").toString());
            // 从plugin_schema和openapi_schema构建工具参数信息
            tool.setInputDesc("会议开始结束时间，会议室");
            tool.setOutputDesc("会议预订结果");
            tool.setInputSchema(
                "{\"type\":\"object\",\"properties\":{\"meetingRoom\":{\"type\":\"string\",\"description\":\"会
议室\"},\"start\":{\"type\":\"string\",\"description\":\"会议开始时间，格式为HH:mm\"},\"end\":{\"type
\":\"string\",\"description\":\"会议结束时间，格式为HH:mm\"}},\"required\":[\"meetingRoom\",\"start
\",\"end\"]}");
            tool.setOutputSchema("");
            tool.setFunction(s -> {
                log.info("do the open api call plugin_schema={} openapi_schema={}",
                    toolMetadata.get("plugin_schema"), toolMetadata.get("openapi_schema"));
                return null;
            });
            return tool;
        }).collect(Collectors.toList());
    }
    private void doSomeFilter(List<RetrievedTool> retrievedTools, String query) {
        // do some filter here
        log.info("{} {}", query, retrievedTools);
```



```
    }
}
```

其中，toolProvider中实现了provide接口，可以利用工具检索的返回动态构建出工具列表，同时也可以加一些后处理工作，如根据黑白名单做工具的过滤。

- 与上述的toolProvide呼应，在向toolRetriever中添加工具时，可以添加任意的元数据，用于在tooProvider中把工具组装出来：

```
// 构造工具元数据
Map<String, Object> toolMetaData = new HashMap<>();
toolMetaData.put("name_for_human", "预订会议室");
toolMetaData.put("name", "reserve_meeting_room");
toolMetaData.put("description_for_human", "预订会议室");
toolMetaData.put("description", "预订会议室，请在需要预订会议室时调用此工具");
toolMetaData.put("status", "on");
toolMetaData.put("plugin_type", "API");
toolMetaData.put("plugin_schema", "this is a plugin schema");
toolMetaData.put("openapi_schema", "this is a openapi schema");
final ToolMetadata toolMetadata = new ToolMetadata();
toolMetadata.setToolId("reserve_meeting_room");
toolMetadata.setToolMetadata(toolMetaData);

// 工具管理面添加工具到toolRetriever，这里实际可以添加若干个工具
cssToolRetriever.addToolsFromMetadata(Collections.singletonList(toolMetadata));

// 运行时检索工具，并添加到Agent执行
final List<Tool> toolList = cssToolRetriever.search("预订会议室", 1, 0.8f);
```

工具的检索与之前的用法一致。

- 将Tool Retriever集成在Agent中的完整示例如下：

```
// 工具集
final List<Tool> toolList =
    Arrays.asList(new MetricQuery(), new ReserveMeeting(), new ReserveMeetingRoom(), new
StuffQuery());

// 新增InMemoryToolProvider，添加工具集
final InMemoryToolProvider inMemoryToolProvider = new InMemoryToolProvider();
inMemoryToolProvider.add(toolList);

// 初始化CSSToolRetriever
final VectorStoreConfig vectorStoreConfig = VectorStoreConfig.builder()
    .indexName(TestConstant.CSS_TOOL_RETRIEVER_INDEX)
    .vectorFields(Arrays.asList("name", "description"))
    .build();
final CSSToolRetriever cssToolRetriever = new CSSToolRetriever(inMemoryToolProvider,
vectorStoreConfig);

// 添加工具
cssToolRetriever.addTools(toolList);

// 添加多轮改写
cssToolRetriever
    .setQueryPreprocessor(messages -> new
ConversationRewriteSkill(LLMs.of(LLMs.PANGU)).rewrite(messages));

// 为Agent添加ToolRetriever
Agent agent = new ReactPanguAgent(
    new
YundaoLLM(LLMConfig.builder().llmModuleConfig(LLMModuleConfig.builder().build()).build()));
agent.setToolRetriever(cssToolRetriever);

// 多轮对话调用
List<ConversationMessage> messages = new ArrayList<>();
messages.add(ConversationMessage.builder().role(Role.USER).content("定个2点的会议").build());
messages.add(
    ConversationMessage.builder().role(Role.ASSISTANT).content("请问您的会议预计何时结束？另外，您
是需要预订线上会议还是实体会议室？ ").build());
messages.add(ConversationMessage.builder().role(Role.USER).content("4点结束，线上会议").build());
agent.run(messages);
```



```
// 删除工具
cssToolRetriever.remove(toolList.stream().map(Tool::getToolId).collect(Collectors.toList()));
```

有两个变化值得关注，一是为ToolRetriever添加了一个queryPreprocessor，它的作用为对用户输入的多轮对话进行改写，会将改写后的结果作为工具检索的输入，这里使用了系统内置的ConversationRewriteSkill，它的作用为将多轮对话改写为单轮。二是在创建一个Agent后，调用了setToolRetriever方法为其添加了一个ToolRetriever，这样Agent所使用的工具会根据用户的对话动态的选择。

14.3.8 应用示例

14.3.8.1 搜索增强

场景介绍

私有化场景下，大模型需要基于现存的私有数据提供服务。通过外挂知识库（Embedding、向量库）方式提供通用的、标准化的文档问答场景。

工程实现

1. 准备知识库。
2. 获取并安装SDK包。
3. 在配置文件(llm.properties)中配置模型信息。

```
# 盘古模型IAM 认证信息，根据实际填写
sdk.llm.pangu.iam.url=
sdk.llm.pangu.iam.domain=
sdk.llm.pangu.iam.user=
sdk.llm.pangu.iam.password=
sdk.llm.pangu.project=

## 盘古模型信息，根据实际情况填写
sdk.llm.pangu.url=

## CSS Embedding模型api
sdk.embedding.css.url=
sdk.embedding.css.iam.url=
sdk.embedding.css.iam.domain=
sdk.embedding.css.iam.user=
sdk.embedding.css.iam.password=
sdk.embedding.css.iam.project=

## CSS 向量库
sdk.memory.css.url=
sdk.memory.css.user=
sdk.memory.css.password=
```

4. 工程实现。

```
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vector;
import com.huaweicloud.pangu.dev.sdk.api.memory.vector.Vectors;
import com.huaweicloud.pangu.dev.sdk.api.memory.config.VectorStoreConfig;
import com.huaweicloud.pangu.dev.sdk.api.embedings.Embeddings;

String query = "杜甫的诗代表了什么主义诗歌艺术的高峰？";
// 初始化向量库
Vector cssVector = Vectors.of(Vectors.CSS,
    VectorStoreConfig.builder()
        .embedding(Embeddings.of(Embeddings.CSS))
        .indexName("sdk-test-dataset-webqa-10")
        .build());
```



```
// 检索文档；
docs = cssVector.similaritySearch(query, 4, 105);

// 文档问答
DocSkill skill = Skills.Document.newDocAskStuffSkill(LLMs.of(LLMs.PANGU));
String answer = skill.executeWithDocs(docs, query);
```

14.3.8.2 长文本摘要

场景介绍

切割长文本，利用大模型逐步总结。 如对会议/报告/文章等较长内容总结概述。

工程实现

1. 获取并安装SDK包。
2. 在配置文件(llm.properties)中配置模型信息。

```
# IAM 认证信息，根据实际填写
sdk.llm.pangu.iam.url=
sdk.llm.pangu.iam.domain=
sdk.llm.pangu.iam.user=
sdk.llm.pangu.iam.password=
sdk.llm.pangu.project=

## 盘古模型信息，根据实际情况填写
sdk.llm.pangu.url=
```

3. 工程实现。

```
import com.huaweicloud.pangu.dev.sdk.skill.DocSkill;
import com.huaweicloud.pangu.dev.sdk.api.skill.Skills;
import com.huaweicloud.pangu.dev.sdk.documentloader.splitter.TextSplitter;

import org.apache.commons.io.FileUtils;

// 加载原始内容， 需根据文件源自行实现读取步骤
// 以txt文件为例；
String context = FileUtils.readFileToString(new File("D:/test.txt"), StandardCharsets.UTF_8);

// 通过分割符分割文档；
// 以 分割符“\n”， 每段大小最大1000字符 为例
List<String> docs = TextSplitter.splitByChars(context, "\n", 1000);

// 对文档进行摘要
DocSkill skill = Skills.Document.newDocSummarizeMapReduceSkill(LLMs.of(LLMs.PANGU));
String summarize= skill.executeWithTexts(docs);
```

14.4 Python SDK

14.4.1 安装 SDK

pip 直接安装

执行如下命令：

```
pip install pangu_kits_app_dev_py
```


本地导入

1. 从support网站上下载pangu-kits-app-dev-py的whl包。
2. 建议使用conda创建一个新的python环境，python版本选择3.9。
3. 在whl包同级目录下，执行如下命令安装：
```
pip install pangu_kits_app_dev_py-2.4.0-py3-none-any.whl
```

安装可选

1. 安装全部依赖项（2.1.0以前版本需手动安装langchain-openai，命令pip install langchain-openai）：
```
pip install pangu_kits_app_dev_py[all]
```
2. cache相关依赖：
```
pip install redis~=4.5.5
pip install gptcache~=0.1.37
pip install redis-om~=0.1.3
pip install pymysql~=1.1.0
pip install SQLAlchemy~=2.0.19
```

API 手册

SDK API 手册

14.4.2 配置 SDK

基础配置项

SDK依赖的配置项主要通过加载llm.properties配置文件。

1. 在项目路径下，创建llm.properties文件，并根据实际需要配置相应的值。
2. 在环境变量中配置“SDK_CONF_PATH”指向该配置文件：
```
# 建议在业务项目入口处配置
import os
os.environ["SDK_CONFIG_PATH"] = "./llm.properties"
```
3. 完整配置项如下：

   配置项中的密码等字段建议在配置文件或者环境变量中密文存放，使用时解密，确保安全，详见**配置文件敏感信息加密配置**。

```
################################## GENERIC CONFIG
##################################

## User-defined Prompt.
#
sdk.prompt.path=

## Proxy.
# Examples: http://127.0.0.1:8000 ;
#
# sdk.proxy.enabled=
# sdk.proxy.url=
# sdk.proxy.user=
# sdk.proxy.password=

## Generic IAM info. This config is used when the specified IAM is not configured.
## Either user password authentication or AK/SK authentication.
#
sdk.iam.url=
sdk.iam.domain=
sdk.iam.user=
sdk.iam.password=
```



```
sdk.iam.project=
sdk.iam.ak=
sdk.iam.sk=
sdk.iam.disabled=


################################## LLM ################################

## Pangu
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;
#
sdk.llm.pangu.url=
## If necessary, you can specify the IAM configuration.
# sdk.llm.pangu.iam.url=
# sdk.llm.pangu.iam.domain=
# sdk.llm.pangu.iam.user=
# sdk.llm.pangu.iam.password=
# sdk.llm.pangu.iam.project=
# sdk.llm.pangu.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.llm.pangu.proxy.enabled=
# sdk.llm.pangu.model-version=

## Gallery
# Examples: https://{endPoint}/v1/infers/{deploymentId} ;
#
sdk.llm.gallery.url=
## If necessary, you can specify the IAM configuration.
# sdk.llm.gallery.iam.url=
# sdk.llm.gallery.iam.domain=
# sdk.llm.gallery.iam.user=
# sdk.llm.gallery.iam.password=
# sdk.llm.gallery.iam.project=
# sdk.llm.gallery.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.llm.gallery.proxy.enabled=


################################## EMBEDDINGS
################################

## CSS
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId}/{modelVersion} ;
#
sdk.embedding.css.url=
## If necessary, you can specify the IAM configuration.
# sdk.embedding.css.iam.url=
# sdk.embedding.css.iam.domain=
# sdk.embedding.css.iam.user=
# sdk.embedding.css.iam.password=
# sdk.embedding.css.iam.project=
# sdk.embedding.css.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.embedding.css.proxy.enabled=


################################## MEMORY ################################

## CSS or ES
# Examples: http://127.0.0.1:9200,http://127.0.0.2:9200 ;
#
sdk.memory.css.url=
sdk.memory.css.user=
sdk.memory.css.password=

## DCS or Redis
# Examples: 127.0.0.1:6379 ;
#
# sdk.memory.dcs.url=
```



```
# sdk.memory.dcs.user=
# sdk.memory.dcs.password=

## RDS or Mysql
# Examples: jdbc:mariadb://127.0.0.1:3306/sdk?
useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai ;
#
# sdk.memory.rds.url=
# sdk.memory.rds.user=
# sdk.memory.rds.password=
# sdk.memory.rds.poolSize=


################################## DOC SPLIT
##################################

## CSS
# Examples: https://{endPoint}/v1/{projectId}/applications/{appId} ;
#
# sdk.doc.split.css.url=
# sdk.doc.split.css.filepath=
# sdk.doc.split.css.mode=
## If necessary, you can specify the IAM configuration.
# sdk.doc.split.css.iam.url=
# sdk.doc.split.css.iam.domain=
# sdk.doc.split.css.iam.user=
# sdk.doc.split.css.iam.password=
# sdk.doc.split.css.iam.project=
# sdk.doc.split.css.iam.disabled=
## If necessary, you can specify the proxy status.
# sdk.doc.split.css.proxy.enabled=
```

日志打印配置

SDK日志采用logging模块，参考以下代码开启相应日志打印信息：

```
import logging
# 打印在命令行（与打印在文件不同时生效）
logging.basicConfig(level=logging.DEBUG)

# 打印在日志文件（与打印在命令行不同时生效）
logging.basicConfig(level=logging.DEBUG,  # 控制台打印的日志级别
                    filename='new.log',
                    filemode='a',
                    format='%(asctime)s - %(pathname)s[line:%(lineno)d] - %(levelname)s: %(message)s')
```

配置文件敏感信息加密配置

配置项中的认证凭据等信息不建议明文配置在配置文件中，可以通过以下方式扩展自定义的加解密组件：

1. 在一个module(yourmodule)中自定义一个解密方法decrypt_func(key_id, cipher)，要求可以通过`from yourmodule import decrypt_func`这样的方式使用该方法。
2. 在配置文件中配置`sdk.crypto.implementation.path=yourmodule.decrypt_func`指向自定义的解密方法的引用。程序加载时会通过import_lib加载该方法。
3. 配置文件中配置密文的格式：`sdk.llm.iam.iamPwd={Crypto.key_id}cipher`，其中key_id和cipher会在配置项读取时被解析传递进decrypt_func方法中，进行解密。

14.4.3 LLMs（语言模型）

LLMs模块用于对大语言模型API的适配封装，提供统一的接口快速地调用盘古、GALLERY三方模型等模型API。

- 初始化：根据相应模型定义LLM类，如使用盘古LLM为： LLMs.of("pangu")。
```
from pangukitsappdev.api.llms.factory import LLMs
# 初始化盘古LLM
llm_api = LLMs.of("pangu")
```
- 基础问答：基础的模型文本问答（temperature等参数采用模型默认的设置）。
```
llm_api.ask("你是谁？").answer
```
- 自定义参数问答：自定义设置如temperature等参数，获得对应的效果。
```
from pangukitsappdev.api.llms.llm_config import LLMConfig, LLMParamConfig
from pangukitsappdev.api.llms.factory import LLMs
# 设置模型参数，temperature为0.9
llm_config = LLMConfig(llm_param_config=LLMParamConfig(temperature=0.9))

# 初始化带参数的盘古LLM
pangu_api = LLMs.of("pangu", llm_config)
pangu_api.ask("写一篇五言律诗").answer
```
支持调整的参数解释：
```
max_tokens: Optional[int]          # 完成时要生成的令牌的最大数量
temperature: Optional[float]       # 调整随机抽样的程度，温度值越高，随机性越大; 范围见模型API规范
top_p: Optional[float]             # 核采样值， 和temperature不同时配置
presence_penalty: Optional[float]  # 存在惩罚，增加模型谈论新主题的可能性，范围见具体模型API规范
frequency_penalty: Optional[float] # 频率惩罚，降低模型重复的可能性，提高文本的多样性、创新性（从参数盘古大模型暂不支持）
stream: Optional[bool]             # 是否开启流式调用
```
- 流式问答（只适用于ask接口）：模型问答，开启流式效果，响应消息流式打印。
```
import sys
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.llms.llm_config import LLMConfig, LLMParamConfig
# 设置模型参数，stream为true
llm_config = LLMConfig(llm_param_config=LLMParamConfig(stream=True))
# 盘古LLM
pangu_llm = LLMs.of("pangu", llm_config)

tokens = pangu_llm.ask("写一篇200字的散文")

actual_tokens = []
for token in tokens:
    sys.stdout.write(token)
    sys.stdout.flush()
    actual_tokens.append(token)
```
- 流式问答：支持skill和agent流式输出。
```
import sys
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.llms.llm_config import LLMConfig, LLMParamConfig
from pangukitsappdev.callback.StreamCallbackHandler import StreamCallbackHandler
from typing import Dict, Any, List
from langchain_core.messages import BaseMessage
from langchain_core.outputs import LLMResult

# 继承StreamCallbackHandler方法，实现流式输出
class TextStreamCallBack(StreamCallbackHandler):
    def __init__(self):
        super().__init__()

    def on_chat_model_start(
        self,
        serialized: Dict[str, Any],
        messages: List[List[BaseMessage]],
        **kwargs: Any,
    ) -> Any:
        pass

    def on_llm_new_token(
        self,
        token: str,
        **kwargs: Any,
    ) -> Any:
```



```
        sys.stdout.write(token)
        sys.stdout.flush()

    def on_llm_end(
        self,
        response: LLMResult,
        **kwargs: Any,
    ) -> Any:
        pass

# 设置模型参数，stream为true
llm_config = LLMConfig(llm_param_config=LLMParamConfig(stream=True))
# 盘古LLM
pangu_llm = LLMs.of("pangu", llm_config)
pangu_llm.set_callback(TextStreamCallBack())
tokens = pangu_llm.ask("写一篇200字的散文")
for token in tokens:
    pass
```

- 多轮对话问答：传递历史问答记录，实现多轮对话问答能力，同时支持自定义参数问答、流式问答。

```
from pangukitsappdev.api.llms.base import ConversationMessage, Role
from pangukitsappdev.api.llms.factory import LLMs


messages = [ConversationMessage(role=Role.SYSTEM, content="你是一个乐于助人的助手"),
            ConversationMessage(role=Role.USER, content="北京有什么好玩的地方"),
            ConversationMessage(role=Role.ASSISTANT, content="长城"),
            ConversationMessage(role=Role.USER, content="具体介绍一下")]
pangu_llm = LLMs.of("pangu", llm_config)
pangu_llm.ask(messages).answer
```

- 带人设的问答：支持在LLM配置项中设置人设，在LLM问答时系统会自动加上该人设，同时支持以上问答功能（暂不支持GALLERY三方模型）。

```
import sys
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.llms.llm_config import LLMConfig, LLMParamConfig

llm_config = LLMConfig()
llm_config.llm_module_config.system_prompt = "你是华为开发的AI助手"
# 盘古LLM
pangu_llm = LLMs.of("pangu", llm_config)

answer = pangu_llm.ask("你是谁")
```

14.4.4 Prompt（提示词模板）

提示词模板模块提供模板格式化、自定义配置管理功能。

- 模板格式化

```
from langchain import PromptTemplate
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.llms.llm_config import LLMConfig, LLMParamConfig
# 初始化Prompt模板对象
prompt_template = PromptTemplate.from_template("Tell me a {{adjective}} joke about {{content}}",
template_format="jinja2")

# 支持dict格式匹配替换Prompt模板
format1 = prompt_template.format(**{"adjective": "funny", "content": "chickens"})
assert "Tell me a funny joke about chickens" == format1

# 支持**kwargs匹配替换Prompt模板
format2 = prompt_template.format(adjective="funny", content="chickens")
assert "Tell me a funny joke about chickens" == format2
```

- 自定义prompt

```
# 按约定的格式准备prompt文件；
# 文档结构和文件名参考提供的系统预置prompts文件
```



```
文档结构示例：
……
 prompts
   -- default
     -- documents
       -- stuff.pt
```

```
# 配置sdk配置项，指定prompt文件绝对路径，以 /home 路径为例

sdk.prompt.path=/home/prompts/default
```

14.4.5 Memory（记忆）

Memory（记忆）模块结合外部存储为LLM应用提供长短期记忆功能，用于支持上下文记忆的对话、搜索增强等场景。

Memory（记忆）支持多种不同的存储方式和功能。

- Cache缓存：是一种临时存储数据的方法，它可以提高数据的访问速度和效率。缓存可以根据不同的存储方式进行初始化、更新、查找和清理操作。缓存还可以支持语义匹配和查询，通过向量和相似度的计算，实现对数据的语义理解和检索。
- Vector向量存储：是一种将数据转换为数学表示的方法，它可以度量数据之间的关系和相似度。向量存储可以根据不同的词向量模型进行初始化、更新、查找和清理操作。向量存储还可以支持多种相似算法，如余弦相似度、欧氏距离、曼哈顿距离等，实现对数据的相似度评分和排序。
- History对话消息存储：是一种将对话消息保存在内存中的方法，它可以记录和管理对话历史。对话消息存储可以根据不同的会话标识进行初始化、更新、查找和清理操作。对话消息存储还可以支持多种过滤条件，如时间范围、用户标识、消息类型等，实现对话消息的筛选和分析。

14.4.5.1 Cache

Cache缓存是一种临时存储数据的方法，它可以把常用的数据保存在内存或者其他设备中，当需要访问这些数据时，无需再去原始的数据源查找，而是直接从缓存中获取，从而节省时间和资源。

Cache缓存有以下几种操作：

- 初始化：指定缓存使用哪种存储方式，例如，使用内存型缓存可以设置为memory_cache = Caches.of("inMemory")。

```
from pangukitsappdev.api.memory.cache.factory import Caches
# 内存型
memory_cache = Caches.of("inMemory")
# Redis
redis_cache = Caches.of("redis")
# mysql
sql_cache = Caches.of("sql")
```

- 更新数据：指向缓存中添加或修改数据，需要指定数据的键值对和结果对象。例如，把1+1这个问题和用户cache会话下对应的答案2保存到缓存中，参考示例如下：

```
from pangukitsappdev.api.schema import LLMResp
cache = Caches.of("inMemory")
# 更新数据
cache.update("1+1", LLMResp(answer=2))
```

- 查询数据：从缓存中获取数据，需要指定数据的键值对。例如，查找1+1这个问题对应的答案，参考示例如下：

```
# 查找数据
cache_value = cache.lookup("1+1")
```


- 清理数据：删除用户cache会话下缓存中的数据。例如，删除所有缓存数据，参考示例如下：

```
# 清理
cache.clear()
```

- 参数解释：用于设置缓存对象的一些基本信息，如过期时间、session_tag等。

```
expire_after_access: int  # 缓存失效策略-基于访问后到期时间（支持inMemeory缓存）
expire_after_write: int   # 缓存失效策略-基于写入后到期时间（支持redis缓存）
maximum_size: int         # 缓存失效策略-基于个数大小（支持inMemory、sql缓存）
session_tag: str          # 用户指定cache会话标志
```

- 缓存失效策略。

```
from pangukitsappdev.api.memory.cache.cache_config import CacheStoreConfig

# redis缓存配置写入2s后到期
redis_cache = Caches.of("redis", CacheStoreConfig(expire_after_write=2))
# inMemory缓存配置缓存窗口数量为3，访问后2s到期
memory_cache = Caches.of("inMemory", CacheStoreConfig(maximum_size=3, expire_after_access=2))
# sql缓存配置缓存窗口数量为3
sql_cache = Caches.of("sql", CacheStoreConfig(maximum_size=3))
```

- 语义缓存（同步适配langchain语义缓存暂时不支持expire_after_write）

  语义缓存是一种基于向量和相似度的缓存方法，它可以实现对数据的语义匹配和查询。语义缓存可以根据不同的向量存储、相似算法、评分规则和阈值进行配置，并且可以使用不同的词向量模型进行嵌入。

```
from pangukitsappdev.api.memory.cache.cache_config import CacheStoreConfig, ServerInfoRedis
from pangukitsappdev.api.embeddings.factory import Embeddings
# redis向量
# 不同的向量存储， 不同的相似算法；计算的评分规则不同； 可以同过scoreThreshold 设置相似性判断阈值
# 例如使用Redis向量、余弦相似度、CSS词向量模型，并且设置相似性判断阈值为0.1f，代码示例如下
embedding_api = Embeddings.of("css")
cache_config = CacheStoreConfig(store_name="semantic_redis",
                                server_info=ServerInfoRedis(env_prefix="sdk.memory.dcs"),
                                embedding=embedding_api,
                                vector_store_name="redis",
                                distance_strategy="cosine",
                                score_threshold=0.1,
                                session_tag="test-semantic-cache-vector-001")
cache = Caches.of("semantic_redis", cache_config)

# 更新数据
# 指向语义缓存中添加或修改数据，需要指定数据的键值对和结果对象
# 例如，把“把缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识对应的答案“存在”保存到语义缓存中，代码示例如下
cache.update("缓存是否存在？", LLMResp(answer="存在。"))

# 校验，一致
# 用于检查缓存中的数据是否与查询的数据是否一致，如果一致，就返回缓存中的结果对象
# 例如，查询“缓存是否存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”
query = "缓存是否存在？"
cache_value_after = cache.lookup(query)
assert cache_value_after is not None
print(cache_value_after.answer)

# 校验，相似
# 用于检查缓存中的数据是否与查询的数据语义相似，如果相似，就返回缓存中的结果对象。这个操作需要使用向量和相似度的计算，以及设置的阈值来判断
# 例如，查询“缓存存在？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就可以从缓存中获取到之前保存的答案“存在”，因为这个问题和“缓存是否存在？”问题语义相似
query_sim = "缓存存在？"
cache_value_semantic = cache.lookup(query_sim)
assert cache_value_semantic is not None
print(cache_value_semantic.answer)

# 校验，不一致
# 用于检查缓存中的数据是否与查询的数据不一致，如果不一致，返回空值
```



```
# 例如，查询“有没有数据？”这个问题和“test-semantic-cache-vector-001”这个会话标识，就无法从缓存中获取到任何答案，因为这个问题和之前保存的问题都不一致
query_not = "有没有数据？"
assert cache.lookup(query_not) is None
```

14.4.5.2 Vector

Embedding

Emebedding模块用于对Emebedding模型API的适配封装，提供统一的接口快速地调用CSS等模型emebedding能力。

- 初始化：根据相应模型定义Emebedding类，如使用华为CSS Embedding为：Embeddings.of("css");。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
# 初始化 Css Embedding
embedding_api = Embeddings.of("css")
```

- embedding单文本：把单个字符串转换为向量数据。（向量维度由模型确定）。

```
text = "this is a test text."
# embed query.
embedding = embedding_api.embed_query(text)
print(embedding)
```

- embedding批量文档：把文档批量转换为向量数据。

```
text = "this is a test text."
# embed documents.
embeddings = embedding_api.embed_documents([text])
print(embeddings)
```

Splitter

文档拆分解析，提供对文档数据进行拆分解析能力，支持pdf/doc/docx/ppt/pptx/xls/xlsx/png/jpg/jpeg/bmp/gif/tiff/webp/pcx/ico/psd等格式文档。

- 初始化

  根据相应解析接口定义DocSplit类，以使用华为Pangu DocSplit为例。

  其中，filePath指的是需要解析的文档路径；mode为分割解析模式，具体定义如下：

  0 - 返回文档的原始段落，不做其他处理。

  1 - 根据标注的书签或目录分段，一般适合有层级标签的word文档。

  2 - 根据内容里的章节条分段，适合制度类文档。

  3 - 根据长度分段，默认按照500字拆分，会尽量保留完整句子。

```
from pangukitsappdev.api.doc_split.factory import DocSplits
from pangukitsappdev.api.doc_split.split_config import SplitConfig
split_config = SplitConfig()
split_config.file_path='/data/xxx.docx'
split_api = DocSplits.of("pangu-doc", split_config)
```

- 文档解析

```
doc_list = split_api.load()
for doc in doc_list:
    print(doc.page_content)
```

向量库

向量库用于向量数据存储，提供向量数据检索能力。

- 初始化，以使用华为CSS示例。

```
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.api.embeddings.factory import Embeddings
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="your_index_name",
                                        embedding=Embeddings.of("css"),
                                        text_key="name",
                                        vector_fields=["description"],
                                        distance_strategy="inner_product",
                                        server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)
```

- 数据入库

```
from pangukitsappdev.vectorstores.bulk_data import BulkData
bulk_list = [BulkData(id="1", data={"name": "名称name1", "description": "foo"}),
             BulkData(id="2", data={"name": "名称name2", "description": "bar"}),
             BulkData(id="3", data={"name": "名称name3", "description": "baz"})]
]
vector_api.add_docs(bulk_list)
```

- 数据检索

```
docs = vector_api.similarity_search("bar", top_k=2)
```

- 数据清理

```
 vector_api.clear()
```

- CSS插件模式（内部已集成Embedding， 支持多字段组合向量检索）。

  CSS插件模式，需要提前手工创建索引（因索引中需要指定embdding/rank模型，SDK不能简单自动创建）。

  CSS插件模式，不支持clear删除索引接口（索引外部创建，应由外部删除）。

```
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="your_index_name",
                                        text_key="name",
                                        vector_fields=["name", "description"],
                                        server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)

# 检索
docs = vector_api.similarity_search("bar", top_k=2)

# 添加
bulk_list = [BulkData(id="1", data={"name": "名称name1", "description": "foo"}),
             BulkData(id="2", data={"name": "名称name2", "description": "bar"}),
             BulkData(id="3", data={"name": "名称name3", "description": "baz"})]
]
vector_api.add_docs(bulk_list)
```

  通过vectorStoreConfig判断使用CSS的插件模式和非插件模式。如果配置了embedding模型，则使用非插件模式，否则使用插件模式。注意，在非插件模式下，vectorFields有且只有1个。

14.4.5.3 History

History缓存，用于存储历史对话信息，辅助模型理解上下文信息，历史消息对有固定窗口、消息摘要等策略。

- 初始化：消息记录支持不同的存储方式， 如内存、DCS（Redis）和RDS（Sql）。

```
from pangukitsappdev.memory.sql_message_history import SQLMessageHistory
from pangukitsappdev.api.memory.cache.cache_config import ServerInfoSql, ServerInfoRedis
from pangukitsappdev.api.memory.message_history_config import MessageHistoryConfig
from pangukitsappdev.memory.redis_message_history import RedisMessageHistory
from langchain.memory import ChatMessageHistory
# 内存型:
```



```
chat_message = ChatMessageHistory()

# Redis:
redis_chat_message =
RedisMessageHistory(msg_history_config=MessageHistoryConfig(store_name="redis",

server_info=ServerInfoRedis(env_prefix="sdk.memory.dcs"),
                                                    session_tag="test-memory-0624"))
# Sql:
sql_chat_message = SQLMessageHistory(msg_history_config=MessageHistoryConfig(store_name="sql",

server_info=ServerInfoSql(env_prefix="sdk.memory.rds"),
                                                  session_tag="test-memory-0624"))
```

- 添加、查找、删除数据。

```
# 更新数据
chat_message.add_ai_message("i am ai.")
chat_message.add_user_message("i am tester.")
# 查找数据
contents = [msg.content for msg in chat_message.messages]
# 清理
chat_message.clear()
```

- 消息策略（windows-size）：将固定轮次历史对话信息，作为历史上下文信息。

```
from langchain.memory import ChatMessageHistory, ConversationBufferWindowMemory
from pangukitsappdev.api.memory.cache.cache_config import ServerInfoRedis
# 历史消息存储
redis_chat_message =
RedisMessageHistory(msg_history_config=MessageHistoryConfig(store_name="redis",

server_info=ServerInfoRedis(env_prefix="sdk.memory.dcs"),
                                                    session_tag="test-memory"))
# 固定窗口策略: langchain.memory.buffer_window.ConversationBufferWindowMemory
memory = ConversationBufferWindowMemory(k=3, chat_memory=redis_chat_message)

memory.chat_memory.add_user_message("This is me, the human")
memory.chat_memory.add_ai_message("This is me, the AI")
print(memory.chat_memory.messages)
memory.chat_memory.clear()
```

- 消息策略（信息摘要）

  将历史消息进行摘要后，作为历史上下文信息。

```
from pangukitsappdev.memory.conversation_summory_memory import
ConversationSummaryBufferMemory
from pangukitsappdev.api.skill.base import SimpleSkill
from pangukitsappdev.prompt.prompt_tmpl import PromptTemplates
# 历史消息存储
redis_chat_message =
RedisMessageHistory(msg_history_config=MessageHistoryConfig(store_name="redis",

server_info=ServerInfoRedis(env_prefix="sdk.memory.dcs"),
                                                    session_tag="test-memory"))
# 摘要策略
memory =
ConversationSummaryBufferMemory(summary_skill=SimpleSkill(PromptTemplates.get("memory_sum
mary"), LLMs.of("pangu")),
                                chat_memory=redis_chat_message)

memory.chat_memory.add_user_message("This is me, the human")
memory.chat_memory.add_ai_message("This is me, the AI")
print(memory.chat_memory.messages)
memory.chat_memory.clear()
```

14.4.6 Skill（技能）

14.4.6.1 基础问答

提供简单的对话实现。

- 初始化

```
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.skill.base import SimpleSkill
from langchain.prompts import PromptTemplate
# 自定义模板
prompt_template = PromptTemplate.from_template("讲一个关于{{subject}}的笑话，字数{{count}}字以内", template_format="jinja2")
skill = SimpleSkill(prompt_template=prompt_template, llm_api=LLMs.of("pangu"))
```

- 问答

```
from pangukitsappdev.api.llms.llm_config import LLMParamConfig

# 不带参数的问答
skill.execute({"subject": "哈士奇", "count": 20})

# 带参数的问答
llm_param_config = LLMParamConfig(temperature=0.9)
skill.execute({"subject": "哈士奇", "count": 20}, llm_param_config)
```

14.4.6.2 多轮对话

支持上下文记忆的多轮对话。

- 初始化

```
from pangukitsappdev.skill.conversation_skill import ConversationSkill
from pangukitsappdev.api.llms.factory import LLMs
skill = ConversationSkill(LLMs.of("pangu"))
```

- 问答

```
from pangukitsappdev.memory.redis_message_history import RedisMessageHistory
from langchain.memory import ConversationBufferWindowMemory
# 定义存储策略
skill.set_memory(ConversationBufferWindowMemory(k=3, chat_memory=RedisMessageHistory()))
answer = skill.execute("中国首都是哪个城市？")
print(answer)
assert "北京" in answer
answer = skill.execute("它有什么好玩的地方？")
print(answer)
assert "故宫" in answer
```

14.4.6.3 文档问答

基于已有的知识库进行回答。有stuff、refine和map-reduce策略。

- Stuff：将所有文档直接填充到prompt中，提给模型回答，适合文档较少的场景。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.ask import DocAskStuffSkill
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="your_index_name",
                                        embedding=Embeddings.of("css"),
                                        text_key="name",
                                        vector_fields=["description"],
                                        distance_strategy="inner_product",
                                        server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)
# 检索
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"
docs = vector_api.similarity_search(query, 4)
```



```
# 问答
doc_skill = DocAskStuffSkill(LLMs.of("pangu"))

print(doc_skill.execute({"documents": docs, "question": query}))
```

- Refine：基于首个文档，并循环后续文档来迭代更新答案。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.ask import DocAskRefineSkill
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="your_index_name",
                                        embedding=Embeddings.of("css"),
                                        text_key="name",
                                        vector_fields=["description"],
                                        distance_strategy="inner_product",
                                        server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)

# 检索
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"
docs = vector_api.similarity_search(query, 4)

# 问答
doc_skill = DocAskRefineSkill(LLMs.of("pangu"))

print(doc_skill.execute({"documents": docs, "question": query}))
```

- Map-Reduce：先将文档单独进行摘要， 将摘要后的文档再提交给模型。 必要时循环迭代摘要。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.ask import DocAskMapReduceSkill
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="your_index_name",
                                        embedding=Embeddings.of("css"),
                                        text_key="name",
                                        vector_fields=["description"],
                                        distance_strategy="inner_product",
                                        server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)

# 检索
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"
docs = vector_api.similarity_search(query, 4)

# 问答
doc_skill = DocAskMapReduceSkill(LLMs.of("pangu"))

print(doc_skill.execute({"documents": docs, "question": query}))
```

14.4.6.4 文档摘要

基于已有的知识库，进行摘要总结。有stuff、refine、map-reduce策略。

- Stuff：将所有文档直接填充到prompt中，提给模型处理，适合文档较少的场景。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.summary import DocSummaryStuffSkill
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="your_index_name",
                                        embedding=Embeddings.of("css"),
                                        text_key="name",
                                        vector_fields=["description"],
```



```
                                    distance_strategy="inner_product",
                                    server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)

# 检索
query = "杜甫"
docs = vector_api.similarity_search(query, 4)

# 摘要
doc_skill = DocSummaryStuffSkill(LLMs.of("pangu"))

print(doc_skill.execute({"documents": docs}))
```

- Refine：基于首个文档，并循环后续文档来迭代更新答案。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.summary import DocSummaryRefineSkill
vector_store_config = VectorStoreConfig(store_name="css",
                                    index_name="your_index_name",
                                    embedding=Embeddings.of("css"),
                                    text_key="name",
                                    vector_fields=["description"],
                                    distance_strategy="inner_product",
                                    server_info=ServerInfoCss(env_prefix="sdk.memory.css"))

vector_api = Vectors.of("css", vector_store_config)

# 检索
query = "杜甫"
docs = vector_api.similarity_search(query, 4)

# 摘要
doc_skill = DocSummaryRefineSkill(LLMs.of("pangu"))

print(doc_skill.execute({"documents": docs}))
```

- Map-Reduce：先将文档单独进行摘要， 将摘要后的文档再提交给模型。 必要时，会循环迭代摘要。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.summary import DocSummaryMapReduceSkill
vector_store_config = VectorStoreConfig(store_name="css",
                                    index_name="your_index_name",
                                    embedding=Embeddings.of("css"),
                                    text_key="name",
                                    vector_fields=["description"],
                                    distance_strategy="inner_product",
                                    server_info=ServerInfoCss(env_prefix="sdk.memory.css"))

vector_api = Vectors.of("css", vector_store_config)

# 检索
query = "杜甫"
docs = vector_api.similarity_search(query, 4)

# 摘要
doc_skill = DocSummaryMapReduceSkill(LLMs.of("pangu"))

print(doc_skill.execute({"documents": docs}))
```

14.4.7 Agent（智能代理）

Agent（智能代理），用于对复杂任务的自动拆解与外部工具调用执行，一般包括任务规划、记忆系统和执行系统。

- 任务规划：将复杂目标任务分解为小的可执行子任务，通过评估、自我反思等方式提升规划成功率。
- 记忆系统：通过构建记忆模块去管理历史任务和策略，并让Agent结合记忆模块中相关的信息以获取最优化任务解决策略。
- 任务执行：能通过工具与外界发生联系并产生影响，工具可以自定义，包括查询信息、调用服务、网络搜索、文件管理、调用云服务等，通过Agent构建一个让LLM按照特定的规则迭代运行的Prompt，直到任务完成或者达到终止条件（如设置迭代次数）。

14.4.7.1 实例化 Tool

Tool分为StaticTool（静态工具）和DynamicTool（动态工具）两类，静态工具需要开发者事先定义好，即在编译期定义与实例化；动态工具开发者可以在系统运行时动态构建，即在运行态定义与实例化。

StaticTool（静态工具）

静态工具可以通过继承Tool的方式新增，在_run接口中实现工具的功能，例如：

```
from typing import Type
from pangukitsappdev.tool.tool import Tool
from pydantic import BaseModel, Field


class AddTool(Tool):
    class AddParam(BaseModel):
        a: int = Field(description="加法运算的数字")
        b: int = Field(description="加法运算的数字")

    name = "add"
    description = "加法运算"
    args_schema: Type[BaseModel] = AddParam
    principle = "请在需要做两个整型的加法运算时调用此工具"
    input_desc = "加法输入"
    output_desc = "加法运算的结果"

    def _run(self, a: int, b: int) -> int:
        return a + b
```

@Tool说明：

- name。工具的标识，建议为英文且与实际工具含义匹配，在同一个Agent中唯一。
- description。工具的描述，建议为中文，尽可能的简短描述工具。
- principle。何时使用该工具，为重要参数，该描述直接影响LLM对工具使用的判断，尽量描述清楚。如果Agent实际执行效果不符合预期，可以调整。
- input_desc。工具的入参描述 ，为重要参数，该描述直接影响LLM对入参的提取，尽量描述清楚。如果Agent实际执行效果不符合预期，可以调整。
- output_desc。工具的出参描述，当前对Agent的表现无重要影响。
- args_schema。工具入参类型，为重要参数，入参继承BaseModel的类型需额外指定，简单类型无需指定。
- return_type。指定工具返回类型，为可选参数，如_run方法未指定返回类型时必选。
- 如果输入输出参数为复杂类型，则需要通过继承BaseModel定义复杂类型的参数描述，此时input_desc、output_desc可以填空字符串，但仍然建议给出简要的描

述。当前版本不支持复杂类型中再嵌套复杂类型，只支持基本类型：str、int、float、bool，建议参数数量不超过5个。

@Field说明：

- description。参数的描述，为重要参数，该描述直接影响LLM对入参的提取，尽量描述清楚，如果Agent实际执行效果不符合预期，可以调整。

上例中的args_schema为一个复杂的入参，如果工具的入参为一个基本类型，则不需要再额外定一个结构体，例如：

```
from typing import Type
from pangukitsappdev.tool.tool import Tool
from pydantic import BaseModel, Field


class ReverseTool(Tool):
    name = "reverse"
    description = "字符串翻转"
    principle = "请在需要字符串翻转时调用此工具"
    input_desc = "输入的字符串"
    output_desc = "反转的结果"

    def _run(self, s: str) -> str:
        return s[::-1]
```

DynamicTool（动态工具）

动态工具可以在业务运行态动态新增或修改：

```
from pangukitsappdev.tool.tool import Tool
from pydantic import BaseModel, Field, create_model

def add(a: int, b: int) -> int:
    return a + b

# 定义函数方式
add_tool = Tool.from_function(func=add,
                              name="add",
                              description="加法运算",
                              principle="请在需要做两个整型的加法运算时调用此工具",
                              input_desc="加法输入",
                              output_desc="加法运算的结果",
                              args_schema=create_model('AddTool', a=(int, Field(description="加法运算的数字")),
b=(int, Field(description="加法运算的数字")))
)
# lambda匿名方式
lambda_add_tool = Tool.from_function(func=lambda a, b: a+b,
                              name="add",
                              description="加法运算",
                              principle="请在需要做两个整型的加法运算时调用此工具",
                              input_desc="加法输入",
                              output_desc="加法运算的结果",
                              args_schema=create_model('AddTool', a=(int, Field(description="加法运算的数字")),
b=(int, Field(description="加法运算的数字"))),
                              return_type=int)
```

- name、description、principle、input_desc、output_desc和args_schema的定义与说明与静态工具相同。
- return_type。为可选参数，如果func为未指定返回值类型的callable类型，必须通过return_type指定返回值类型。

14.4.7.2 实例化 Agent

Agent实例化过程包括注册LLM和注册工具两个部分。


```
from pangukitsappdev.agent.react_pangu_agent import ReactPanguAgent
from pangukitsappdev.api.llms.factory import LLMs

agent = ReactPanguAgent(LLMs.of("pangu",
llm_config=LLMConfig(llm_param_config=LLMParamConfig(with_prompt=True),
llm_module_config=LLMModuleConfig(model_version="N2_agent_v2"))))
agent.set_max_iterations(5)
agent.add_tool(ReverseTool())
agent.add_tool(AddTool())
agent.add_tool(SearchTool())
```

- 静态工具和动态工具的注册方式相同，通过addTool接口进行注册。
- 通过set_max_iterations可以设置最大迭代次数，控制Agent子规划的最大迭代步数，防止无限制的迭代或出现死循环情况。
- Agent使用的模型必须为Pangu-NLP-N2-Agent-L0.C模型，或其衍生模型，使用通用模型或其他模型无法运行。如上例所示，当前的module-version需要配置为“N2_agent_v2”，模型的相关配置需要改为Pangu-NLP-N2-Agent-L0.C模型的地址。
- with_prompt参数配置为True，prompt的拼接由Agent托管处理。

14.4.7.3 运行 Agent

- 单轮执行

  调用run接口运行一个Agent：

  ```
  agent.run("帮我定个下午3点到8点2303会议室")
  ```

  Agent的运行时会进行自我迭代，并且选择合适的工具，在日志中打印最终的执行结果：

  ```
  用户: 帮我定个下午3点到8点2303会议室
  助手: 好的，2023-11-17 15:00到2023-11-17 20:00的2303会议室已为您预定成功。
   - 步骤1:
     思考:好的，我需要先查询2303会议室的状态，如果可用，我将为您预定。
     行动:使用工具[reserve_meeting_room],传入参数{"start": "2023-11-17 15:00", "end": "2023-11-17 20:00", "meeting_room": "2303"}
     工具返回:2023-11-17 15:00到2023-11-17 20:00的2303已预定成功
   - 步骤2
     答复:好的，2023-11-17 15:00到2023-11-17 20:00的2303会议室已为您预定成功。
  ```

- 多轮执行：

  ```
  messages = [ConversationMessage(role=Role.USER, content="定个2点的会议"),
              ConversationMessage(role=Role.ASSISTANT, content="请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？"),
              ConversationMessage(role=Role.USER, content="4点结束，线上会议")]
  agent.run(messages)
  ```

  运行结果示例：

  ```
  用户: 定个2点的会议
  助手: 请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？
  用户: 4点结束，线上会议
  助手: 已为您预定2023年6月29日14:00至16:00的线上会议，请准时参加。
   - 步骤1:
     思考:好的，我将为您预定2点至4点的线上会议。
     行动:使用工具[reserve_meeting],传入参数{"start": "2023-06-29 14:00", "end": "2023-06-29 16:00"}
     工具返回:2023-06-29 14:00到2023-06-29 16:00的线上会议已预定成功
   - 步骤2
     答复:已为您预定2023年6月29日14:00至16:00的线上会议，请准时参加。
  ```

- 多轮执行增强

  上述的例子中实际运行时只提供给模型多轮的对话，并没有提供工具执行的过程，为了让大模型的效果更好，可以传入agent_session。

  ```
  用户: 定个金桥203会议室的会议
  助手: 好的，请问会议具体是什么时候开始和结束？请按照"yyyy-MM-dd HH:mm"的格式告诉我。
   - 步骤1
     答复:好的，请问会议具体是什么时候开始和结束？请按照"yyyy-MM-dd HH:mm"的格式告诉我。
  ```



```
用户: 下午2点到4点
助手: 已经成功为您预定了2023年6月22日下午2点到4点的金桥203会议室。
 - 步骤1:
   思考:好的，我现在就去为您预定金桥203会议室，时间是下午2点到4点。
   行动:使用工具[reserve_meeting_room],传入参数{"start": "2023-06-22 14:00", "end": "2023-06-22
16:00", "meeting_room": "金桥203"}
   工具返回:2023-06-22 14:00到2023-06-22 16:00的金桥203已预定成功
 - 步骤2
   答复:已经成功为您预定了2023年6月22日下午2点到4点的金桥203会议室。
用户: 再定一个明天8点到9点的会议室
助手: 好的，请问您希望预定哪个会议室?
 - 步骤1
   答复:好的，请问您希望预定哪个会议室?
用户: 同一个会议室
助手: 已经成功为您预定了2023年6月23日早上8点到9点的金桥203会议室。
 - 步骤1:
   思考:好的，我现在就去为您预定金桥203会议室，时间是明天8点到9点。
   行动:使用工具[reserve_meeting_room],传入参数{"start": "2023-06-23 08:00", "end": "2023-06-23
09:00", "meeting_room": "金桥203"}
   工具返回:2023-06-23 08:00到2023-06-23 09:00的金桥203已预定成功
 - 步骤2
   答复:已经成功为您预定了2023年6月23日早上8点到9点的金桥203会议室。
```

如果服务是分布式的，需要将session对象在外部持久化。

- 单步执行

  有时并不希望Agent完全自主执行，在某些关键节点，让用户先进行确认，确认后再执行，或者用户对模型的结果有异议或者想法有变化，想对当前结果进行更改。此时可以单步运行Agnet：

```
"""
单步执行Agent，提供干预能力
:param agent_session: 包括初始状态，以及执行步骤间的agentSession，可以使用AgentSessionHelper类
辅助处理
:return: Agent执行的结果
"""
run_step(agent_session: AgentSession) -> AgentSession
```

以下为一个完整的示例：

```
agent.clear_tool()
agent.add_tool(RiskDetectionTool())
session = AgentSessionHelper.init_agent_session("请帮我查一下方欣科技有限公司今年1月的经营异常风
险")
# 预期Agent返回reportType为经营异常风险检测的Json，呈现给终端用户
session = agent.run_step(session)
# 终端用户反悔，想改成欠税信息检测，修改信息后继续让Agent执行
AgentSessionHelper.set_user_feedback(session, "改为欠税信息")

# 预期Agent返回reportType为欠税信息体检的Json，呈现给终端用户
session = agent.run_step(session)
# 终端用户确认，调用外部系统，进一步确认公司名称和编号，补充信息后，让Agent继续执行
AgentSessionHelper.set_user_feedback(session, "公司改为金财互联数据服务有限公司，nsrsbh为
91440101MA59JN5456")

# 预期Agent更新companyName和nsrsbh
session = agent.run_step(session)
# 此时已经得到完整调用信息了，调用risk_detection API，将API返回结果给Agent继续执行
AgentSessionHelper.set_tool_output(session, "金财互联数据服务有限公司1月份存在异常经营风险")

session = agent.run_step(session)
# 打印
if session.agent_session_status == "FINISHED":
    AgentSessionHelper.update_assistant_message(session, True)
    print(AgentSessionHelper.print_plan(session))
```

上述例子中，终端用户对Agent即将要执行的API做了2次修改，最终进行了执行。

run_step的入参为AgentSession，有较大的灵活性，因此提供了辅助类AgentSessionHelper帮助开发者更容易处理：

- init_agent_session：使用用户消息初始化一个AgentSession。
- set_tool_output：向Agent的当前步骤设置工具返回结果。
- set_user_feedback：向Agent的当前步骤设置用户反馈。

打印结果为：

```
用户: 请帮我查一下方欣科技有限公司今年1月的经营异常风险
助手: 根据企业健康体检工具的结果，金财互联数据服务有限公司1月份存在异常经营风险。
 - 步骤1:
   思考:好的，我将使用企业健康体检工具来查询方欣科技有限公司今年1月的经营异常风险。首先，我需要确认工具调用的参数。根据工具定义，需要的参数包括"report_type"和"company_name"。用户已经提供了公司名称"方欣科技有限公司",并指定了时间范围为今年1月。我将设置"report_type"为"经营异常风险检测",并将"skssqq"设置为"2024-01-01","skssqz"设置为"2024-01-31"。现在，我将调用工具。
   行动:使用工具[risk_detection],传入参数{"report_type": "经营异常风险检测","company_name": "方欣科技有限公司","skssqq": "2024-01-01", "skssqz": "2024-01-31"}
   用户反馈:改为欠税信息
 - 步骤2:
   思考:好的，我将修改"report_type"参数为"欠税信息体检",并重新调用工具。
   行动:使用工具[risk_detection],传入参数{"report_type": "欠税信息体检","company_name": "方欣科技有限公司","skssqq": "2024-01-01", "skssqz": "2024-01-31"}
   用户反馈:公司改为金财互联数据服务有限公司，nsrsbh为91440101MA59JN5456
 - 步骤3:
   思考:好的，我将更新"company_name"参数为"金财互联数据服务有限公司",并添加"nsrsbh"参数为"91440101MA59JN5456"。现在，我将再次调用工具。
   行动:使用工具[risk_detection],传入参数{"report_type": "欠税信息体检","company_name": "金财互联数据服务有限公司","skssqq": "2024-01-01", "skssqz": "2024-01-31", "nsrsbh": "91440101MA59JN5456"}
   工具返回:金财互联数据服务有限公司1月份存在异常经营风险
 - 步骤4
   答复:根据企业健康体检工具的结果，金财互联数据服务有限公司1月份存在异常经营风险。
```

有时会需要Agent对之前的思考、行动回答用户的最终问题，如果对模型返回的结果不满意，可以选择使用AgentSessionSkill解决：

```
llm_api = LLMs.of("pangu",
llm_config=LLMConfig(llm_param_config=LLMParamConfig(temperature=0.01),
llm_module_config=LLMModuleConfig( module_version="N2_agent_v2")))
agent = ReactPanguAgent(llm=llm_api)
agent.set_max_iterations(5)
agent.add_tool(QueryScoreTool())

agent_session = agent.run("我的数学成绩和语文成绩分别是多少")

summary_llm_api = LLMs.of("pangu",
llm_config=LLMConfig(llm_module_config=LLMModuleConfig(url="https://pangu.cn-southwest-2.myhuaweicloud.com/v1/infers/N2-基础模型的调用地址")))
agent_session_skill = AgentSessionSkill(llm=summary_llm_api)
print(agent_session_skill.summary(agent_session))
```

运行结果示例：

```
助手: 您的语文成绩是56分。
 - 步骤1:
   思考:好的，我需要调用 query_score 工具来查询您的成绩。首先，我将查询您的数学成绩。
   行动:使用工具[query_score],传入参数{"arg": "数学"}
   工具返回:你的数学的成绩是55分
 - 步骤2:
   思考:您的数学成绩是55分。接下来，我将查询您的语文成绩。
   行动:使用工具[query_score],传入参数{"arg": "语文"}
   工具返回:你的语文的成绩是56分
 - 步骤3
   答复:您的语文成绩是56分。
最终结果: 您的数学成绩是55分，而语文成绩是56分。
```

AgentSessionSkill使用的大模型建议为N2-基础模型或者其他同等类型的模型。

14.4.7.4 监听 Agent

一次Agent的响应如果涉及到多个任务的分解，往往会执行比较长的时间，此时可以对agent的执行过程进行监听，输出中间步骤。

AgentListener的定义如下：

```
class AgentListener(ABC):
    """Agent监听，允许对Agent的各个阶段进行处理
    """
    def on_session_start(self, agent_session: AgentSession):
        """
        Session启动时调用
        :param agent_session: AgentSession
        """

    def on_session_iteration(self, agent_session: AgentSession):
        """
        Session迭代过程中调用
        :param agent_session: AgentSession
        """

    def on_session_end(self, agent_session: AgentSession):
        """
        Session结束时调用
        :param agent_session: AgentSession
        """

    def on_check_interrupt_requirements(self, agent_session: AgentSession):
        """
        onSessionIteration调用结束后，检查Agent是否需要终止，如果需要终止，则返回true，默认不终止
        可以在终止前对agentSession进行修改，如：修改agent的finalAnswer
        :param agent_session: AgentSession
        :return: bool类型结果
        """
        return False
```

定义一个监听器

通过实现AgentListener定义一个监听器：

```
from pangukitsappdev.agent.agent_session import AgentSession
from pangukitsappdev.api.agent.base import AgentListener

class TestListener(AgentListener):
    def on_session_start(self, agent_session: AgentSession):
        print(agent_session)

    def on_session_iteration(self, agent_session: AgentSession):
        print(agent_session)

    def on_session_end(self, agent_session: AgentSession):
        print(agent_session)
```

上述代码分别对应了Agent的开始、中间过程和结束阶段。

为 Agent 添加一个监听器

通多调用Agent的addListener接口添加一个监听器：

```
from pangukitsappdev.agent.react_pangu_agent import ReactPanguAgent
from pangukitsappdev.api.llms.factory import LLMs

agent = ReactPanguAgent(LLMs.of("pangu"))
agent.add_listener(TestListener())
```

listener会在Agent运行时生效。

监听的对象

监听的对象为一个AgentSession：


```
class AgentSession(BaseModel):
    """
    Agent运行Session，包含历史Action，当前Action，状态
    Attributes:
        messages: 本次session的用户的输入
        session_id: UUID，在一个session内唯一
        current_action: 当前Action
        agent_session_status: Agent状态
        is_by_step: 是否是逐步执行
        current_message: 当前的AssistantMessage
    """
    messages: List[ConversationMessage]
    session_id: str
    current_action: Optional[AgentAction]
    agent_session_status: str
    is_by_step: Optional[bool]
    current_message: Optional[ConversationMessage]
```

AgentAction包含Agent的工具选择、工具执行结果、思考等信息，AgentSessionStatus为Agnet的执行状态。

通过监听终止 Agent 的执行

当需要在Agent的执行过程中终止执行时，除了通过setMaxIterations设置Agent的最大迭代次数，也可以通过实现监听器的on_check_interrupt_requirement实现。

```
class InterruptListener(AgentListener):
    def on_check_interrupt_requirements(self, agent_session: AgentSession):
        if "capital" == agent_session.current_action.action:
            # 如果当前的action为captial，则返回capital工具调用的原始返回值
            agent_session.final_answer = agent_session.current_action.observation
            return True
        return False
agent.add_listener(InterruptListener())
```

上面的例子中，当满足if判断条件时，就会直接终止agent的执行，并且agent的finalAnswer被设置为工具的原始返回值。

14.4.7.5 Agent 流式输出

Agent用于工具调用场景，与普通的LLM流式输出相比，区分了文本流与工具流。文本流将输出模型的思考过程和最终结果；工具流将输出工具的调用过程，而工具的调用的执行结果是通过监听获取的。

通过如下接口为Agent添加流式输出的回调：

```
from pangukitsappdev.callback.StreamCallbackHandler import StreamCallbackHandler
# 以下为两个自定义StreamCallbackHandler示例
class TextStreamCallBack(StreamCallbackHandler):
    def __init__(self):
        # agent文本输出工具类
        super().__init__()
        self.stream = ""

    def on_chat_model_start(
        self,
        serialized: Dict[str, Any],
        messages: List[List[BaseMessage]],
        **kwargs: Any,
    ) -> Any:
        self.stream += "[Text stream start]"

    def on_llm_new_token(
        self,
        token: str,
        **kwargs: Any,
```



```
    ) -> Any:
        self.stream += token

    def on_llm_end(
        self,
        response: LLMResult,
        **kwargs: Any,
    ) -> Any:
        self.stream += "[Text stream end]"


class ToolStreamCallBack(StreamCallbackHandler):
    def __init__(self):
        # agent工具输出工具流
        super().__init__()
        self.stream = ""

    def on_chat_model_start(
        self,
        serialized: Dict[str, Any],
        messages: List[List[BaseMessage]],
        **kwargs: Any,
    ) -> Any:
        self.stream += "[Tool stream start]"

    def on_llm_new_token(
        self,
        token: str,
        **kwargs: Any,
    ) -> Any:
        self.stream += token

    def on_llm_end(
        self,
        response: LLMResult,
        **kwargs: Any,
    ) -> Any:
        self.stream += "[Tool stream end]"
text_stream_callback = TextStreamCallBack()
tool_stream_callback = ToolStreamCallBack()
agent.set_stream_callback(text_stream_callback, tool_stream_callback)
```

StreamCallBack的实现与定义与LLM的回调完全相同。

14.4.7.6 Tool Retriever

Agent在实际生产应用中往往涉及到的工具数量较多，如果把所用的工具全部添加至Agent会产生如下问题：

- 占用大量输入token。
- 和问题无关的工具太多，影响模型的判断。

通过Tool Retriever可以解决上述问题，其原理是在Agent运行前，先从所有可用的工具中选择与问题最相关的工具，再交给Agent去处理。

- 定义一个Tool Retriever：

```
from pangukitsappdev.tool.in_memory_tool_provider import InMemoryToolProvider
from pangukitsappdev.retriever.css_tool_retriever import CSSToolRetriever

# 新增InMemoryToolProvider，添加工具集
in_memory_tool_provider = InMemoryToolProvider()
tool_list = [AddTool(), ReverseTool(), ReserveMeeting(), ReserveMeetingRoom()]
in_memory_tool_provider.add(tool_list)

# 初始化CSSToolRetriever
vector_config = VectorStoreConfig(index_name="your_index_name",
                                  verify_certs=False,
```



```
                                    text_key="name",
                                    vector_fields=["name", "description"])
css_tool_retriever = CSSToolRetriever(tool_provider, vector_config)
```

定义一个ToolRetriever包含2个参数，一个ToolProvider，一个向量数据库配置。其中，ToolProvider的作用为根据工具检索的结果组装工具。

上述例子使用了一个简单的InMemoryToolProvider，InMemoryToolProvider的原理为将完整的工具存入内存，再根据工具检索的结果（tool_id）将其从内存中取出。一般来说，ToolProvider将由用户自定义，后续会有例子说明。

上述例子使用的向量数据库配置指定索引名称，以及使用name和description作为向量化字段，因此工具入库时，会将工具的name和description进行向量化，并在后续的检索中生效。

注意，上述tool_list中包含的工具在SDK中并不存在，需要替换成实际的工具。

- 向ToolRetriever中添加工具：

```
# 添加工具
css_tool_retriever.add_tools(tool_list)
```

工具添加后，会存储在向量库的索引中，并将指定的字段向量化。

- 从ToolRetriever中查找工具：

```
# 查找工具
result = css_tool_retriever.search("预订会议室", 2)
```

返回的result中，包含与预订会议室最相关的工具。搜索支持topK和阈值2个参数，例如上例指定topK=2，则最多返回2个工具。

- 从ToolRetriever中删除工具：

```
# 删除工具
css_tool_retriever.remove(["add", "reverse"])
```

以上为一个比较基础的用法，在实际使用过程中会有更加灵活的场景，可以通过自定义ToolProvider的方式解决。

- 自定义ToolProvider：

```
# 初始化CSSToolRetriever，使用ToolProviderWithMetadata作为ToolProvider
vector_config = VectorStoreConfig(index_name="your_index_name",
                                    verify_certs=False,
                                    text_key="name",
                                    vector_fields=["name", "description", "principle"])
css_tool_retriever = CSSToolRetriever(ToolProviderWithMetadata(), vector_config)
```

其中，ToolProviderWithMetadata为自定义ToolProvider：

```
import pickle

class ToolProviderWithMetadata(ToolProvider):

    def provide(self, retrieved_tools: List[RetrievedTool], query: str) -> List[AbstractTool]:
        retrieved_tools = self.do_some_filter(retrieved_tools, query)
        return [Tool.from_function(func=pickle.loads(eval(retrieved_tool.tool_metadata.get("function"))),
                                   name=retrieved_tool.tool_id,
                                   description=retrieved_tool.tool_metadata.get("description"),
                                   principle=retrieved_tool.tool_metadata.get("principle"),
                                   input_desc=retrieved_tool.tool_metadata.get("input_desc"),
                                   output_desc=retrieved_tool.tool_metadata.get("output_desc"),

args_schema=pickle.loads(eval(retrieved_tool.tool_metadata.get("args_schema"))),

return_type=pickle.loads(eval(retrieved_tool.tool_metadata.get("return_type"))))
                for retrieved_tool in retrieved_tools]

    @staticmethod
    def do_some_filter(retrieved_tools: List[RetrievedTool], query: str) -> List[RetrievedTool]:
        print(f"{retrieved_tools}, {query}")
        return retrieved_tools
```

上述tool_provider中，实现了provide接口，可以利用工具检索的返回动态构建出工具列表，同时也可以加一些后处理工作，例如根据黑白名单做工具的过滤。

- 与上述的tool_provide呼应，在向tool_retriever中添加工具时，可以添加任意的元数据，python需要借助pickle将函数或类转换成字节流字符串存入CSS中，用于在tool_provider中把工具组装出来：

```
from pydantic import BaseModel, Field
import pickle

# 构造工具元数据
class MeetingInfo(BaseModel):
    id: str = Field(description="会议ID")
    info: str = Field(description="会议信息")

def list_meeting(inputs: NoneType) -> List[MeetingInfo]:
    return [MeetingInfo(id=1, info="金桥2023"), MeetingInfo(id=2, info="金桥203")]

tool_meta_data = {
    "name": "list_meeting",
    "description": "查询员工的会议预订状态，返回已经预订的会议和其会议ID",
    "principle":"请在需要查询员工已预订会议室列表时使用",
    "input_desc": "",
    "output_desc": "已预订会议室列表",
    "args_schema": str(pickle.dumps(None)),
    "function": str(pickle.dumps(list_meeting)),
    "return_type": str(pickle.dumps(MeetingInfo))
}
# 工具管理面添加工具到toolRetriever，这里实际可以添加若干个工具
css_tool_retriever.add_tools_from_metadata([tool_meta_data])

# 运行时检索工具，并添加到Agent执行
tool_list = css_tool_retriever.search("查询会议室预订状态", 1, 0.8)
```

  工具的检索与之前的用法一致。

- 以下是一个将Tool Retriever集成在Agent中的完整示例：

```
from pangukitsappdev.skill.conversation_rewrite_skill import ConversationRewriteSkill

# 工具集
toolList = [AddTool(), ReverseTool(), ReserveMeeting(), ReserveMeetingRoom()]

# 新增InMemoryToolProvider，添加工具集
in_memory_tool_provider = InMemoryToolProvider()
tool_list = [AddTool(), ReverseTool(), ReserveMeeting(), ReserveMeetingRoom()]
in_memory_tool_provider.add(tool_list)

# 初始化CSSToolRetriever
vector_config = VectorStoreConfig(index_name="your_index_name",
                                  verify_certs=False,
                                  text_key="name",
                                  vector_fields=["name", "description"])
css_tool_retriever = CSSToolRetriever(tool_provider, vector_config)


# 添加工具
css_tool_retriever.add_tools(tool_list)

# 添加多轮改写
css_tool_retriever.set_query_preprocessor(ConversationRewriteSkill(LLMs.of("pangu")).rewrite)

# 为Agent添加ToolRetriever
agent = ReactPanguAgent(LLMs.of("yundao"))
agent.set_tool_retriever(css_tool_retriever);

# 多轮对话调用
messages = [ConversationMessage(role=Role.USER, content="定个2点的会议"),
            ConversationMessage(role=ROLE.ASSISTANT, content="请问您的会议预计何时结束？另外，您是需要预订线上会议还是实体会议室？"),
            ConversationMessage(role=Role.USER, content="4点结束，线上会议")]
print(agent.run(messages))
```



```
# 删除工具
cssToolRetriever.remove([tool.name for tool in tool_list])
```

其中，有两个变化值得关注，一是为ToolRetriever添加了一个query_preprocessor，它的作用为对用户输入的多轮对话进行改写，会将改写后的结果作为工具检索的输入，这里使用了系统内置的ConversationRewriteSkill，它的作用为将多轮对话改写为单轮。二是在创建一个Agent后，调用了set_tool_retriever方法为其添加了一个ToolRetriever，这样Agent所使用的工具会根据用户的对话动态的选择。

14.4.8 应用示例

14.4.8.1 搜索增强

场景介绍

私有化场景下，大模型需要基于现存的私有数据提供服务。通过外挂知识库（Embedding、向量库）方式提供通用的、标准化的文档问答场景。

工程实现

1. 准备知识库。
2. 获取并安装SDK包。
3. 在配置文件(llm.properties)中配置模型信息。

```
# 盘古模型IAM 认证信息，根据实际填写
sdk.llm.pangu.iam.url=
sdk.llm.pangu.iam.domain=
sdk.llm.pangu.iam.user=
sdk.llm.pangu.iam.password=
sdk.llm.pangu.project=

## 盘古模型信息，根据实际情况填写
sdk.llm.pangu.url=

## CSS Embedding模型api
sdk.embedding.css.url=
sdk.embedding.css.iam.url=
sdk.embedding.css.iam.domain=
sdk.embedding.css.iam.user=
sdk.embedding.css.iam.password=
sdk.embedding.css.iam.project=

## CSS 向量库
sdk.memory.css.url=
sdk.memory.css.user=
sdk.memory.css.password=
```

4. 工程实现。

```
from pangukitsappdev.api.embeddings.factory import Embeddings
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.api.memory.vector.factory import Vectors
from pangukitsappdev.api.memory.vector.vector_config import VectorStoreConfig, ServerInfoCss
from pangukitsappdev.skill.doc.ask import DocAskStuffSkill
query = "杜甫的诗代表了什么主义诗歌艺术的高峰？"
# 初始化向量库
vector_store_config = VectorStoreConfig(store_name="css",
                                        index_name="test-vector-css",
                                        embedding=Embeddings.of("css"),
                                        server_info=ServerInfoCss(env_prefix="sdk.memory.css"))
vector_api = Vectors.of("css", vector_store_config)

# 检索文档
```

```
docs = vector_api.similarity_search(query, 4)
# 文档问答
doc_skill = DocAskStuffSkill(LLMs.of("pangu"))
print(doc_skill.execute({"documents": docs, "question": query}))
```

14.4.8.2 长文本摘要

场景介绍

切割长文本，利用大模型逐步总结。 如对会议/报告/文章等较长内容总结概述。

工程实现

1. 获取并安装SDK包。
2. 在配置文件(llm.properties)中配置模型信息。

```
# IAM 认证信息，根据实际填写
sdk.llm.pangu.iam.url=
sdk.llm.pangu.iam.domail=
sdk.llm.pangu.iam.user=
sdk.llm.pangu.iam.password=
sdk.llm.pangu.iam.project=

## 盘古模型信息，根据实际情况填写
sdk.llm.pangu.url=
```

3. 工程实现。

```
import docx
from pangukitsappdev.api.llms.factory import LLMs
from pangukitsappdev.skill.doc.summary import DocSummaryMapReduceSkill
# 加载原始内容， 需根据文件源自行实现读取步骤
# 以word文件为例，需安装docx库
doc = docx.Document(r'报告.docx')
documents = [d.text for d in doc.paragraphs]

# 对文档进行摘要
skill = DocSummaryMapReduceSkill(LLMs.of("pangu"))
summarize = skill.execute_with_texts(documents)
```

14.5 应用实践

14.5.1 基础问答

应用介绍

基础的大语言模型问答场景。涉及模型问答，流式效果等相关特性。

环境准备

- python3.9 及以上版本。
- 安装依赖的组件包， pip install pangu_kits_app_dev_py gradio。
- 盘古大语言模型。

开发实现

- 创建配置文件llm.properties， 正确配置iam和pangu配置项。信息收集请参考**准备工作**。


```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.
#

################################## GENERIC CONFIG
##################################

## If necessary, you can specify the http proxy configuration.
# sdk.proxy.enabled=true
# sdk.proxy.url=
# sdk.proxy.user=
# sdk.proxy.password=


## Generic IAM info. This config is used when the specified IAM is not configured.
## Either user password authentication or AK/SK authentication.
#
sdk.iam.url=
sdk.iam.domain=
sdk.iam.user=
sdk.iam.password=
sdk.iam.project=


## Pangu
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;
#
sdk.llm.pangu.url=
```

- 创建代码文件（chat.py），示例如下：

```
import os
import sys
import gradio as gr

from pangukitsappdev.api.llms.llm_config import LLMParamConfig
from pangukitsappdev.api.llms.factory import LLMs

# 设置SDK使用的配置文件
os.environ["SDK_CONFIG_PATH"] = "./llm.properties"

# 初始化LLMs
llm_api = LLMs.of("pangu")

with gr.Blocks() as demo:
    chatbot = gr.Chatbot()
    msg = gr.Textbox()
    clear = gr.Button("清除")

    def user(user_message, history):
        return "", history + [[user_message, None]]

    def llm(history):
        history[-1][1] = ""

        # 流式调用大模型
        tokens = llm_api.ask(history[-1][0], LLMParamConfig(stream=True))
        for token in tokens:
            history[-1][1] += token
            yield history

    msg.submit(user, [msg, chatbot], [msg, chatbot], queue=False).then(
        llm, chatbot, chatbot
    )
    clear.click(lambda: None, None, chatbot, queue=False)

demo.queue()
demo.launch()
```

- 终端命令行下执行**python3 chat.py**运行应用，效果如下。

14.5.2 长文本摘要

应用介绍

切割长文本，利用大模型逐步总结，如对会议/报告/文章等总结概述。涉及长文本分割、摘要等相关特性。

环境准备

- python3.9 及以上版本。
- 安装依赖的组件包， pip install pangu_kits_app_dev_py gradio python-docx。
- 盘古大语言模型。

开发实现

- 创建配置文件llm.properties， 正确配置iam和pangu配置项。信息收集请参考**准备工作**。

```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.
#

################################## GENERIC CONFIG
##################################

## If necessary, you can specify the http proxy configuration.
# sdk.proxy.enabled=true
# sdk.proxy.url=
# sdk.proxy.user=
# sdk.proxy.password=


## Generic IAM info. This config is used when the specified IAM is not configured.
## Either user password authentication or AK/SK authentication.
```



```
#
sdk.iam.url=
sdk.iam.domain=
sdk.iam.user=
sdk.iam.password=
sdk.iam.project=


## Pangu
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;
#
sdk.llm.pangu.url=
```

- 创建代码文件（doc_summary.py），示例如下：

```
import os
import gradio as gr
import docx
import time

from pangukitsappdev.skill.doc.summary import DocSummaryMapReduceSkill
from pangukitsappdev.api.llms.factory import LLMs

# 设置SDK使用的配置文件
os.environ["SDK_CONFIG_PATH"] = "./llm.properties"

# 初始化文档问答Skill
doc_skill = DocSummaryMapReduceSkill(LLMs.of("pangu"))

# 合并长度过小的段落，不超过maxLength
def merge_docs(docs, maxLength, separator):
    result = []
    current = []
    length = 0
    for doc in docs:
        if (length + len(doc) > maxLength):
            if current:
                result.append(separator.join(current))
            current = [doc]
            length = len(doc)
        else:
            current.append(doc)
            length = length + len(doc)
    if current:
        result.append(separator.join(current))
    return result

# 加载文档，支持word和txt文本文件
def load_file(name):
    docs = []
    if (name.endswith("doc") or name.endswith("docx")):
        doc = docx.Document(name)
        for grah in doc.paragraphs:
            docs.append(grah.text)
    else:
        data = ""
        with open(name, 'r', encoding='utf-8') as f:
            data = f.read()
        docs = data.split("\n")
    return docs


def summary(files):
    sum_texts = []
    for file in files:
        docs = load_file(file.name)
        # print(docs)
        docs_merge = merge_docs(docs, 1000, "\n")

        # 大模型对文档做摘要
        sum_texts.append(doc_skill.execute_with_texts(docs_merge))
```



```
        # 设置延时，避免访问太频繁
        time.sleep(10)

    return sum_texts[0] if len(sum_texts) == 1 else doc_skill.execute_with_texts(sum_texts)

def upload_file(files):
    file_paths = [file.name for file in files]
    return file_paths

with gr.Blocks() as demo:
    upload_bt = gr.UploadButton("选择文档", file_types=["text"], file_count="multiple")
    file_output = gr.Files()
    upload_bt.upload(upload_file, upload_bt, file_output)

    greet_btn = gr.Button("生成摘要")
    output = gr.Textbox(label="输出")

    greet_btn.click(fn=summary, inputs=file_output, outputs=output, api_name="summary")


demo.launch()
```

- 终端命令行下执行**python3 doc_summary.py**运行应用，效果如下。

14.5.3 Agent 助手

应用介绍

通过模型对复杂任务的自动拆解与外部工具调用执行能力，通过与用户多轮对话，实现会议室预订场景。

环境准备

- Java 1.8。
- 参考安装章节，完成基础环境准备。
- 盘古大语言模型。

开发实现

- 创建配置文件llm.properties， 正确配置iam、pangu配置项。信息收集请参考**准备工作**。


```
# Copyright (c) Huawei Technologies Co., Ltd. 2023-2023. All rights reserved.
#

################################## GENERIC CONFIG
##################################

## If necessary, you can specify the http proxy configuration.
# sdk.proxy.enabled=true
# sdk.proxy.url=
# sdk.proxy.user=
# sdk.proxy.password=


## Generic IAM info. This config is used when the specified IAM is not configured.
## Either user password authentication or AK/SK authentication.
#
sdk.iam.url=
sdk.iam.domain=
sdk.iam.user=
sdk.iam.password=
sdk.iam.project=


## Pangu
# Examples: https://{endPoint}/v1/{projectId}/deployments/{deploymentId} ;
#
sdk.llm.pangu.url=
```

- 创建代码，示例如下：

```
/*****************************************
        会议室状态查询工具
*****************************************/

import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentToolParam;
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;

import lombok.Data;

@AgentTool(toolId = "meeting_room_status_query", toolDesc = "查询会议室的状态，是否被预定或者正在使用中",
        toolPrinciple = "请在需要预定会议室之前使用，查询会议室状态判断是否可以预定", inputDesc = "", outPutDesc = "会议室状态")
public class MeetingRoomStatusQuery extends StaticTool<MeetingRoomStatusQuery.InputParam, String> {
    @Override
    public String run(InputParam input) {
        switch (input.getMeetingRoom()) {
            case "A02":
                return "in use";
            case "A03":
                return "booked";
            default:
                return "available";
        }
    }

    @Data
    public static class InputParam {
        @AgentToolParam(description = "会议开始时间，格式为yyyy-MM-dd HH:mm")
        private String start;

        @AgentToolParam(description = "会议结束时间，格式为yyyy-MM-dd HH:mm")
        private String end;

        @AgentToolParam(description = "会议室")
        private String meetingRoom;
    }
}
```



```
/**********************************************
        会议室预订工具
**********************************************/

import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentTool;
import com.huaweicloud.pangu.dev.sdk.api.annotation.AgentToolParam;
import com.huaweicloud.pangu.dev.sdk.api.tool.StaticTool;

import lombok.Data;

@AgentTool(toolId = "reserve_meeting_room", toolDesc = "预定会议室", toolPrinciple = "请在需要预定
会议室时调用此工具，预定前需要先查询会议室状态",
    inputDesc = "会议开始结束时间，会议室", outPutDesc = "预定会议室的结果")
public class ReserveMeetingRoom extends StaticTool<ReserveMeetingRoom.InputParam, String> {
    @Override
    public String run(InputParam input) {
        return String.format("%s到%s的%s已预定成功", input.start, input.end, input.meetingRoom);
    }

    @Data
    public static class InputParam {
        @AgentToolParam(description = "会议开始时间，格式为yyyy-MM-dd HH:mm")
        private String start;

        @AgentToolParam(description = "会议结束时间，格式为yyyy-MM-dd HH:mm")
        private String end;

        @AgentToolParam(description = "会议室")
        private String meetingRoom;
    }
}


/**********************************************
        Agent实现
**********************************************/

import com.huaweicloud.pangu.dev.sdk.agent.AgentAction;
import com.huaweicloud.pangu.dev.sdk.agent.AgentSession;
import com.huaweicloud.pangu.dev.sdk.agent.AgentSessionStatus;
import com.huaweicloud.pangu.dev.sdk.agent.ReactPanguAgent;
import com.huaweicloud.pangu.dev.sdk.api.agent.Agent;
import com.huaweicloud.pangu.dev.sdk.api.agent.AgentSessionHelper;
import com.huaweicloud.pangu.dev.sdk.api.llms.LLMs;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMModuleConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.config.LLMParamConfig;
import com.huaweicloud.pangu.dev.sdk.api.llms.request.ConversationMessage;
import com.huaweicloud.pangu.dev.sdk.api.llms.request.Role;
import com.huaweicloud.pangu.dev.sdk.api.tool.Tool;
import com.huaweicloud.sdk.demo.test.tools.GetReceipt;
import com.huaweicloud.sdk.demo.test.tools.GetReimbursementLimitTool;
import com.huaweicloud.sdk.demo.test.tools.GetReimbursementRatio;
import com.huaweicloud.sdk.demo.test.tools.GetUserID;

import lombok.extern.slf4j.Slf4j;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.UUID;

@Slf4j
```



```
public class TestAgentCustom {
    // agent
    private static Agent panguAgent;

    // 工具map。在分步骤执行agent场景时，需要调用tool 的run方法来执行tool
    private static LinkedHashMap<String, Tool> toolMap = new LinkedHashMap();

    @BeforeAll
    public static void initAgent() {
        final String customSystemPrompt =
            "你是财务报销助手。当需要用户反馈信息时，尽可能提示用户名称，手机号码等原始信息。今天
的日期是" + new SimpleDateFormat("yyyy年MM月dd日").format(new Date());
        final LLMConfig config = LLMConfig.builder()
            .llmParamConfig(LLMParamConfig.builder().temperature(0.01).withPrompt(true).build())
            .llmModuleConfig(LLMModuleConfig.builder().systemPrompt(customSystemPrompt).build())
            .build();

        panguAgent = new ReactPanguAgent(LLMs.of(LLMs.PANGU, config));
        panguAgent.setMaxIterations(5);

        Tool meetingRoomQuery = new MeetingRoomStatusQuery();
        panguAgent.addTool(meetingRoomQuery);
        toolMap.put(meetingRoomQuery.getToolId(), meetingRoomQuery);

        Tool meetingRoomReserve = new ReserveMeetingRoom();
        toolMap.put(meetingRoomReserve.getToolId(), meetingRoomReserve);
        panguAgent.addTool(meetingRoomReserve);    }

    /**************************************************************
     * 会话信息持久化， 以内存为例。实际生产环境，建议在外部（ SQL/Redis ）持久化
     **************************************************************/

    /**
     * 在生产环境下，agentSession建议在外部持久化，而不是在内存中
     * 如果使用AssistantAPI，华为会提供持久化能力，不需要自行实现
     */
    private static final Map<String, AgentSession> agentSessionMap = new HashMap<>();

    /**
     * 历史对话消息持久化
     */
private static final Map<String, List<ConversationMessage>> agentSessionMessageMap = new
HashMap<>();

    // 保存session信息
    void updateAgentSession(String sessionId, AgentSession agentSession) {
        agentSessionMap.put(sessionId, agentSession);

        // 将最近一轮对话信息永久持久化
        if (!agentSessionMessageMap.containsKey(sessionId)) {
            agentSessionMessageMap.put(sessionId, new ArrayList<>());
        }
        agentSessionMessageMap.get(sessionId)
            .add(agentSession.getMessages().get(agentSession.getMessages().size() - 2));
        agentSessionMessageMap.get(sessionId)
            .add(agentSession.getMessages().get(agentSession.getMessages().size() - 1));
    }

    // 查询session信息
    AgentSession getAgentSessionFromMemory(String sessionId, String userMessage) {
        AgentSession agentSession = agentSessionMap.get(sessionId);
        if (agentSession == null) {
            agentSession = AgentSessionHelper.initAgentSession(userMessage);
            agentSession.setSessionId(sessionId);
            agentSessionMap.put(sessionId, agentSession);
        } else {

agentSession.getMessages().add(ConversationMessage.builder().role(Role.USER).content(userMessage
).build());
```



```
            ConversationMessage assistantMessage =
ConversationMessage.builder().role(Role.ASSISTANT).build();
            agentSession.getMessages().add(assistantMessage);
            agentSession.setCurrentMessage(assistantMessage);
        }
        return agentSession;
}

    void dealMessageByWindowSize(AgentSession agentSession, int windowSize) {
        if (windowSize <= 1) {
            return;
        }

        int preSize = agentSession.getMessages().size();
        if (preSize <= windowSize) {
            return;
        }
        agentSession.setMessages(agentSession.getMessages().subList(preSize - windowSize, preSize));
    }

    /**
     * 用户会话与session关联关系存储， 生产环境建议在外部持久化
     * 如果使用AssistantAPI，华为会提供持久化能力，不需要自行实现
     */
    private static final Map<String, String> sessionUserMap = new HashMap<>();

    @Test
    void test() {
        // 用户id
        String userId = "user01";
        //  session id相同
        String sessionId = UUID.randomUUID().toString();

        // 持久化用户会话信息
        sessionUserMap.put(sessionId, userId);

        // 第一轮用户输入，“查询最大报销额度”
        AgentSession session1st = run(sessionId, "定个2点点的会议");
        // 第一轮模型回复：“好的，请问您想预定哪一个会议室？”
        log.info("助手：" + session1st.getCurrentMessage().getContent());

        // 第二轮用户反馈信息，“A01会议室”
        AgentSession session2nd = run(sessionId, "A01会议室");
        // 第二轮模型回复：“已为您预定 A01会议室，时间为2024年5月15日下午2点到4点。”
        log.info("助手：" + session2nd.getCurrentMessage().getContent());

        // 第三轮用户反馈信息，“会议室更换为
        AgentSession session3th = run(sessionId, "会议室更换为
        // 第三轮回复：” A02会议室在今天下午2点到4点已经被使用了，无法预定。您是否需要更换其他时
间或者其他会议室？”
        log.info("机器人：" + session3th.getCurrentMessage().getContent());
    }

    private AgentSession run(String sessionId, String userMessage) {

        // 从持久化存储中加载会话上下文
        AgentSession agentSession = getAgentSessionFromMemory(sessionId, userMessage);

        // 取近10条数据给模型
        dealMessageByWindowSize(agentSession, 10);
        agentSession.setAgentSessionStatus(AgentSessionStatus.RUNNING);

        // 模型规划
        agentSession = panguAgent.runStep(agentSession);
        log.info(AgentSessionHelper.printPlan(agentSession));

        /**
         * Agent的状态为FINISHED，为FINISHED，所以不需要调用工具
         */
```



```
        if (agentSession.getAgentSessionStatus() == AgentSessionStatus.FINISHED) {
            log.info("Agent的状态为{}，为{}，所以不需要调用工具",
agentSession.getAgentSessionStatus(), AgentSessionStatus.FINISHED);
            AgentSessionHelper.updateAssistantMessage(agentSession, true);

            // 持久化会话上下文信息
            updateAgentSession(sessionId, agentSession);
            return agentSession;
        }

        /**
         * Agent的状态为RUNNING，不为FINISHED，所以需要调用工具，
         * 示例：调用的工具为meeting_room_status_query，入参为{"start": "2024-05-07 14:00",
         * "end": "2024-05-07 16:00", "meetingRoom": "A01"}
         */
        // 最大迭代次数，避免模型规划失败死循环
        int maxIteration = 3;
        int index = 0;
        // 调用工具， 其中可以扩展用户自定义逻辑
        while (index <= maxIteration && agentSession.getAgentSessionStatus() !=
AgentSessionStatus.FINISHED) {
            index++;
            // 从agentSession中取出要调用的工具
            final AgentAction currentAction = agentSession.getCurrentAction();
            log.info("Agent的状态为{}，不为{}，所以需要调用工具，调用的工具为{}，入参为{}",
agentSession.getAgentSessionStatus(),
                AgentSessionStatus.FINISHED, currentAction.getAction(), currentAction.getActionInput());

            // 执行工具
            Tool tool = toolMap.get(currentAction.getAction());
            Object result = tool.runFromJson(currentAction.getActionInput().toString());
            // 获取工具结果后，继续模型推理
            AgentSessionHelper.setToolOutput(agentSession, result.toString());

            agentSession = panguAgent.runStep(agentSession);
            log.info(AgentSessionHelper.printPlan(agentSession));
        }

        AgentSessionHelper.updateAssistantMessage(agentSession, true);
        updateAgentSession(sessionId, agentSession);
        return agentSession;
    }
}
```

