



# 模型推理服务

## 用户使用手册

天翼云科技有限公司

2025 年 8 月

# 目录

1. 产品介绍 .....	4
1.1 产品定义 .....	4
1.2 产品优势 .....	4
1.3 功能特性 .....	5
1.4 应用场景 .....	5
1.5 术语解释 .....	6
1.6 使用限制 .....	7
<b>2. 计费说明 .....</b>	<b>8</b>
2.1 包周期计费方式 .....	8
2.2 按需计费模式-卡时 .....	9
2.3 按需计费模式-Tokens .....	10
2.3.1 Token 概念 .....	10
2.3.2 产品价格 .....	10
2.3.3 使用限制 .....	11
2.4 产品退订 .....	11
3. 快速入门 .....	13
3.1 准备工作 .....	13
3.2 快速入门 .....	13
4. 用户指南 .....	76
4.1 模型体验 .....	76
4.1.1. 模型广场 .....	76
4.1.2. 体验中心 .....	76
4.2 模型服务 .....	77

4.2.1 服务接入 .....	77
4.2.2 在线推理 .....	78
4.2.3 调用监控 .....	80
5. 最佳实践 .....	81
5.1 DeepSeek模型调用-快捷版 .....	81
5.2 DeepSeek模型调用-专业版 .....	102
5.3 使用专属节点部署DeepSeek服务 .....	150
6. 推理服务API .....	154
6.1. 如何调用API .....	154
6.1.1 终端节点 .....	154
6.2. 接口类型列表 .....	156
6.3. API列表 .....	157
6.4. 错误处理 .....	161
6.5. API .....	163
6.5.1. chat 对话API .....	163
6.5.2. image 文本生图API .....	179
6.5.3. Embeddings文本向量化API .....	187
6.5.4. 模型列表API .....	193
7. 平台功能OpenAPI .....	206
7.1 平台功能API使用说明 .....	206
8. 常见问题 .....	207
8.1. 计费相关 .....	207
8.2. 平台操作 .....	208
8.3. 用户权限 .....	208

8.4. 如何联系我们 .....	209
-------------------	-----

# 1. 产品介绍

---

## 1.1 产品定义

模型推理服务是面向企业开发者的一站式大模型服务平台，提供功能全面、安全可靠且具有价格优势的模型调用服务，同时配备完整的大模型开发工具链，帮助企业开发者实现大模型服务与应用对接。

### 【功能模块】

- **模型体验**：提供多场景模型展示与试用，支持开发者在模型广场快速体验主流AI模型，体验中心可零门槛测试基础推理能力。
- **模型服务**：主流大模型提供标准化API接口，支持高并发在线推理服务，内置调用量监控与延迟分析功能，保障企业级应用稳定高效运行。
- **插件服务**：支持接入私有知识库增强领域应答，集成联网搜索获取实时信息，通过插件组合扩展模型边界，应对复杂业务需求。

## 1.2 产品优势

### 1. 模型服务丰富

- a. 多元模型生态：集成主流先进第三方大模型，支持文本、图像、音视频等多模态处理能力。
- b. 多种商品适配：多种商品模式适配业务规模及性能要求，满足不同的业务需求。

### 2. 操作使用方便

- a. 标准化API：统一接口封装模型能力，开发者可直接调用API，实现分钟级集成
- b. 插件工具链：接入私有知识库增强领域应答，集成联网搜索获取实时信息，通过插件组合扩展模型边界，应对复杂业务需求。

### 3. 调用价格实惠

- a. 云上部署：采用云服务模式，替代传统本地部署，无需购置专用硬件设备。

- b. 免维护负担：节省设备运维、人力成本，轻松应对业务扩展与模型更新。
- c. 按需付费：通过按需付费机制实现资源消耗与实际支出的精准匹配

## 1.3 功能特性

- 极致推理加速

平台内置自研推理框架，集成模型量化、TP&EP 混合并行、Cache 感知调度、融合算子优化等先进技术，实现对 Qwen、Deepseek 等主流大模型的深度加速。通过对推理链路的精细化重构，显著降低推理延迟，优化资源占用，保障在复杂业务场景下的高性能响应，突破推理链路繁琐与性能瓶颈的行业限制。

- 灵活高效的推理网关

内置智能推理网关，基于用户行为、流量特征与集群负载状态，动态调度请求流向，提升服务稳定性与可用性。网关全面支持 OAI 标准接口协议，便于模型服务与各类应用系统快速集成，解决推理工具链拼接难题，提升部署效率。

- 智能推理调度

通过动态负载均衡自动分配流量，结合优先级调度确保关键任务资源优先，并支持按需弹性扩展资源池，从而全面提升服务处理效率与稳定性，有效应对高峰需求。

## 1.4 应用场景

- 模型推理

向下纳管智算硬件资源，提供技术运维服务及推理加速。向上提供高性能的模型调用服务，可以通过标准的 API 接口将平台提供的大模型服务集成到自己的业务系统中，提供模型推理一站式部署服务。

主要面向用户：各种软件开发商，特别是行业软件开发商，以及科研院所、大专院校和教育机构、政府、金融机构、工业企业、科技单位、医院等行业客户。

- 模型应用

大模型与插件工具的结合能够有效拓展其在复杂任务中的实用性，支持接入私有知识库增强领域应答，集成联网搜索获取实时信息。

主要面向用户：需要组合模型和工具提供智能办公、智能客服、智能运维能力以及业务主要为知识密集型

场景和特定内容生成的企业。

## 1.5术语解释

- 预置模型

是指平台提供的原始模型，您可以通过选择预置模型进行训练从而得到行业或细分场景模型，不同的基础模型的参数和能力不同，我们将持续推出不同能力方向的模型。

- 函数调用 Function Calling

Function Calling 是一种将大模型与外部工具和 API 相连的关键功能，作为自然语言与信息接口之间的“翻译官”，它能够将用户的自然语言请求智能地转化为对特定工具或 API 的调用，从而高效满足用户的特定需求。

- Token

在自然语言处理中，token 通常指的是将文本分割成的最小单位，比如词语、子词或字符。在调用模型推理服务时，会将输入内容进行分词（tokenize），转化为模型可以理解的 token，经过模型处理后，同样输出 token，并转化为您需要的文本或者其他内容载体。而模型处理（包括输入、输出）的 token 数量会被作为模型推理服务用量的一个重要计量单位。由于不同模型采用的分词策略不同，同一段文本可能会被转化为不同数量的 token。

- 迭代轮次

是指模型训练过程中模型学习数据集的次数，可理解为学习几遍数据，可依据需求进行调整。

- 批处理大小

是指在模型训练过程中，每次处理的数据样本的数量，可理解为模型每看多少数据即更新一次模型参数，在选择批处理大小时需要综合考虑各种因素。

- 学习率

是指更新模型参数的系数，它决定了在每次迭代中，模型参数应该沿着梯度下降的方向更新多少，需要根据具体情况来仔细选择和调整学习率。

- 训练数据集

是机器学习或深度学习模型训练过程中的重要组成部分。训练数据集是一组已知输入和对应输出的数据，用于训练模型以学习从输入到输出的映射关系。构建合适训练集，通过模型调优可增强模型能力，提升预测效果。

- 测试数据集

在机器学习和深度学习中扮演着至关重要的角色，它用于评估模型在未见过的数据上的性能。与训练数据集不同，测试数据集在模型训练过程中是不可见的，也就是说，模型在接触到测试数据之前已经完成了所有的训练和调整。

## 1.6使用限制

模型使用限制：通过体验中心或调用 API 使用模型时，每个模型在上下文长度、最大输出长度、并发数、最大 TPM/RPM 数等方面均有不同数值的限制，部分限制信息可参见[模型广场](#)模型卡片信息。

## 2. 计费说明

### 2.1 包周期计费方式

包周期付费指按订单的购买周期计费，是一种预付费模式，即先付费再使用。您可以通过包周期计费提前预留资源，同时享受比按量计费更大的价格优惠。包周期计费适用于多种场景，尤其是需要稳定资源并长期使用的情况。

产品名称	规格	CPU	内存 (GB)	显卡类型	显卡数	说明	标准价格 (含税)	单位
训推一体 服务 DO- LC-扩展 型包周期	184C 1520 G 8*910B- 64G (液 冷) 或以上	184	1520	910B	8	共享集群	198182. 95	元/服务* 月
训推一体 服务 DO- LC-独立 型包周期	184C 1520 G 8*910B- 64G (液 冷) 或以上	184	1520	910B	8	独立集群	196000. 00	元/服务* 月
训推一体 服务 DO- AC-扩展 型包周期	184C 1520 G 8*910B- 64G(风冷) 或以上	184	1520	910B	8	共享集群	174518. 09	元/服务* 月
训推一体 服务 DO- AC-独立 型包周期	184C 1520 G 8*910B- 64G(风冷) 或以上	184	1520	910B	8	独立集群	172000. 00	元/服务* 月

训推一体 服务 FO- L2	224C 2048 G 8*H800- 640G 或以 上	224	2048	H800	8	独立集群	241212. 00	元/服务* 月
----------------------	--	-----	------	------	---	------	---------------	------------

## 2.2 按需计费模式-卡时

卡时按需付费是一种灵活的计费模式，您可以通过按量计费灵活开通和释放资源，无需提前购买。卡时按需付费的优势在于灵活性和节约成本，适用于需要灵活调整资源、业务不稳定或资金有限的场景。在选择计费模式时，应结合业务需求和实际情况来做出合适的选择。

产品名称	规格	CPU	内存 (GB)	显卡类型	显卡数	说明	标准价格 (含税)	单位
训推一体 DO-LC- 标准型	16C 128G  1*910B- 64G (液 冷) 或以上	16	128	910B	1	共享集群	41.29	元/服务* 小时
训推一体 DO-LC- 扩展型	184C 152 0G 8*910 B-64G (液 冷) 或以上	184	1520	910B	8	共享集群	330.30	元/服务* 小时
训推一体 服务 DO- AC-标准 型	16C 128G  1*910B- 64G(风冷) 或以上	16	128	910B	1	共享集群	36.36	元/服务* 小时

训推一体	184C 152							
服务 DO-	0G 8*910	184	1520	910B	8	共享集群	290.86	元/服务*
AC-扩展	B-64G(风							小时
型	冷)或以上							

## 2.3 按需计费模式-Tokens

### 2.3.1 Token 概念

在模型服务中，一个核心概念是“token”。当您调用模型进行推理服务时，输入内容会首先经过分词处理，转换成模型能够识别的 token 形式。随后，这些 token 经过模型的处理，再以 token 的形式输出，并最终转化为您所需的文本。

**模型服务费用 = token 使用数量 \* token 单价**

不同的模型可能采用不同的分词策略，同一段文本在不同模型上的 tokens 计量会存在差异。

### 2.3.2 产品价格

按 token 使用量付费

- 出账时间：每小时出账。
- 付款类型：按量后付费。
- 免费额度：使用模型时，将优先消耗免费额度，免费额度耗尽后，调用将会失败；如需继续使用，需要手动开通对应的模型付费服务。
- 模型单价（刊例价）：如下表，实际购买价格以下单页面为准。

模型名称		服务类型	计费单位 (元/百万 tokens)	免费说明
Deepseek系列	DeepSeek-R1	输入	4	每个模型可免费使用2500万 tokens，从第一次使用开始
		输出	16	

	DeepSeek-V3	输入	2	限期两周。免费额度用完或到期后，可以付费开通服务。
		输出	8	
	DeepSeek-V3-0324	输入	2	
		输出	8	
	DeepSeek-R1-Distill-Llama-70B	输入	4.1	
		输出	4.1	
DeepSeek-R1-Distill-Qwen-32B	输入	1.3		
	输出	1.3		
千问系列	Qwen-vl-Chat	输入	0.8	每个模型可免费使用100万tokens，从第一次使用开始限期两周。免费额度用完或到期后，可以付费开通服务。
		输出	1.6	
	Qwen2.5-72B-instruct	输入	4.13	
		输出	4.13	
	Qwen2.5-VL-72B-Instruct	输入	4.13	
		输出	4.13	
其他系列		每个模型可免费使用100万tokens，用尽后不支持tokens计费模式，可以通过按卡时计费模式下单		

## 2.3.3使用限制

### 模型推理限制

当前对模型设置了 RPM（Requests Per Minute，每分钟请求数）及 TPM（Tokens Per Minute，每分钟 token 数量）的调用限制，具体限制可在模型广场模型详情页中查看。

## 2.4产品退订

服务开通后 7 天内如未使用则支持退订，退订后即可关闭。

退订地址：[我的一费用中心—订单管理—退订管理](#)

另外，您可通过天翼云官网工单或者客服电话【400-810-9889】沟通申请退款，款项会原路退回。



## 3. 快速入门

### 3.1 准备工作

- 注册天翼云账号

在开通和使用模型推理服务平台之前，您需要先[注册](#)天翼云账号。

1. 打开天翼云门户网站，点击【注册】。
2. 在注册页面，请填写【邮箱地址】、【登录密码】、【手机号码】，并点击【同意协议并提交】按钮，如 1 分钟内手机未收到验证码，请再次点击【免费获取短信验证码】按钮。
3. 注册成功后，可到邮箱激活您的账号，即可体验天翼云。
4. 如需实名认证，请参考账号中心-[实名认证](#)。

- 充值主账户

\* 使用息壤模型推理服务平台之前，请保证您的账户有充足的余额，账户余额需要大于 100 元。

\* 请前往[费用-充值](#)为账户充值。

\* 平台支持按卡时后计费 and 包周期预付费。

### 3.2 快速入门

#### 1. 前提条件

登录模型推理服务控制台。

#### 2. 第一步：获取 App Key 和模型 ID

登录模型推理服务控制台

方式一：

快速获取方式

在左侧菜单栏进入“概览”，点击“API 接入”

在 API 快捷接入页面：

选择服务组：从下拉菜单中选择第一步创建的服务组。如果尚未创建，可点击“确认创建并选择”快速创建默认服务组

选择模型：从模型下拉菜单中选择需要接入的模型

选择完成后，页面下方将自动展示该模型支持的所有编程语言的示例代码

方式二：

在左侧菜单栏进入“服务接入”，点击“+创建服务组”，填写服务组名称、服务组描述、生效时间、配置服务等信息。

创建服务组后，从服务组上面获取 APP KEY。

说明：

- 请妥善保管 App Key，避免公开共享，以防安全风险和资金损失。强烈建议您不要将其直接写入到调用模型的代码中。

通过模型广场选择模型并在详情页内获取模型 ID。

说明：每个模型配有免费token额度用于体验，免费额度用完或到期后请前往在线推理页面开通token付费服务或将模型部署为我的服务（按卡时或包周期付费）。

## 第二步：获取调用模型服务的示例代码并发起调用

支持挑选 Python、Java、GoLang、PHP、Nodejs、curl 中任一调用模型服务。根据选择的语言和框架，可能需要按照必要的依赖库，具体可见以下示例代码的说明：

### Python

推荐安装Python3.x版, openai库可用如下命令安装：

```
pip3 install openai -i https://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com
```

文本生成

```
import openai

from openai import OpenAI

# 需要补充一下属性

baseUrl = "https://wishhub-x6.ctyun.cn/v1/"

# 从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

appKey="your_app_key" # 替换为实际的App Key
model_id = "xirang_model_id" # 替换为实际的modelId
prompt="你好，3.01+103.1等于多少" #对话问题，可替换

def main():
    tihuan    client = OpenAI(base_url=baseUrl,
api_key=appKey)

    messages = [
        {"role": "user", "content": prompt}
    ]
    try:
        res = client.chat.completions.create(
            model=model_id,
            messages=messages,
            stream=False
        )
        print(res.choices[0].message.content or "",
end="", flush=True)

    except openai.APIStatusError as e:
        print(f"APIStatusError: {e.status_code},
{e.message}, {e.body}")
```



图像理解

```
import requests

url = "https://wishhub-x6.ctyun.cn/v1/chat/completions"

headers = {
    "Authorization": "Bearer your_app_key", # 替换为您的
实际app_key
    "Content-Type": "application/json"
}

payload = {
    "model": "xirang_model_id", # 替换为实际的modelId
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "描述下这张图"
                },
                {
                    "type": "image_url",
                    "image_url": {
                        "url":
"https://hangzhou7.zos.ctyun.cn/keywords-
txt/20250708181010/184-200x200.jpg" # 需要替换图片路径
                    }
                }
            ]
        }
    ]
}
```



Image 文本生图

```
import requests

import base64

import os

def generate_image_simple(prompt):

    """极简图像生成函数"""

    """

    your_app_key和xirang_model_id需要替换为实际的值

    """

    url = "https://wishhub-

x6.ctyun.cn/v1/images/generations"

    headers = {

        "Authorization": "Bearer your_app_key",

        "Content-Type": "application/json"

    }

    data = {

        "model": "xirang_model_id",

        "prompt": prompt,

        "n": 1,

        "size": "1024x1024"

    }

    try:

        # 跳过SSL验证并发送请求

        response = requests.post(url, headers=headers,

json=data, timeout=10)

        image_data =

response.json()["data"][0]["b64_json"]
```

## embedding 文本向量化

```
import json

import requests

URL = "https://wishhub-x6.ctyun.cn/v1/embeddings"

headers = {

    "Authorization": "Bearer your_app_key", # 替换为您的实际app_key

    "Content-Type": "application/json"

}

data = {

    "model": "xirang_model_id", # 替换为实际的模型ID

    "input" : "A cute baby sea otter" # 替换文本描述

}

try:

    response = requests.post(URL, headers=headers,

json=data)

    if response.status_code != 200:

        print(response.json())

    else:

        embedding =

response.json()["data"][0]["embedding"]

        print(f"response embedding is {embedding}")

except Exception as e:

    print(f"Exception: {e}")
```

## rerank 重排序

```
import requests

"""
your_app_key和xirang_model_id需要替换为实际的值
"""

API_URL = "https://wishhub-x6.ctyun.cn/v1/rerank"
API_KEY = "your_app_key"
MODEL_ID = "xirang_model_id"

def call_rerank(query, documents, top_n=3):
    """
    调用天翼云 Rerank API
    - query: 查询文本
    - documents: 待排序文档列表
    - top_n: 返回最相关的文档数量
    """
    headers = {
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    }

    data = {
        "query": query,
        "documents": documents,
        "model": MODEL_ID,
        "top_n": top_n
    }

    try:
```

## Java

适用于 Spring AI 框架调用 Open API

支持对文本生成、图像理解、文本生成、embedding 文本向量化以及 rerank 重排序 API 的调用  
环境变量配置

```
spring:
  ai:
    openai:
      api-key: your_app_key #替换成,自己的API KEY
      base-url: https://wishhub-x6.ctyun.cn
    chat:
      options:
        model: xirang_model_id #替换成,自己要用的模型
      embedding:
        options:
          model: xirang_model_id #替换成,自己要用的模型
      image:
        options:
          model: xirang_model_id #替换成,自己要用的模型
      rerank:
        option:
          model: xirang_model_id #替换成,自己要用的模型

server:
  port: 9090
  servlet:
    encoding:
      charset: UTF-8
      force: true
      enabled: true
```

```
<?xml version="1.0" encoding="UTF-8"?>
  <project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-
parent</artifactId>
      <version>3.2.2</version>
      <relativePath/> <!-- lookup parent from
repository -->
    </parent>
    <groupId>com.demo</groupId>
    <artifactId>aitest</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>aitest</name>
    <description>aitest</description>
    <properties>
      <java.version>17</java.version>
      <spring-ai.version>0.8.1</spring-ai.version>
      <spring-cloud.version>2023.0.0</spring-
cloud.version>
    </properties>
    <dependencies>
      <dependency>
```

配置类

```
package com.demo.aitest;

import jakarta.annotation.PostConstruct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import org.springframework.ai.openai.OpenAiChatClient;
import org.springframework.ai.openai.OpenAiChatOptions;
import org.springframework.ai.openai.api.OpenAiApi;

@Configuration
public class OpenAIConfig {

    @Value("${spring.ai.openai.api-key}")
    private String apiKey;

    @Value("${spring.ai.openai.base-url}")
    private String baseUrl;

    @Value("${spring.ai.openai.chat.options.model}")
    private String chatModel;

    @Value("${spring.ai.openai.chat.options.model}")
    private String embeddingModel;
```



OpenAPI Http 客户端

```
package com.demo.aitest.service;

import
com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.Collections;

@Service
@Slf4j
public class HttpClientService {

    @Value("${spring.ai.openai.api-key}")
    private String apiKey;

    @Value("${spring.ai.openai.base-url}")
    private String baseUrl;

    @Autowired
    private RestTemplate restTemplate;
```

## ChatController

已封装对文本生成 API 的支持

```
package com.demo.aitest.controller;

import
com.demo.aitest.controller.request.MultimodalRequest;
import com.demo.aitest.service.HttpClientService;
import
com.fasterxml.jackson.core.JsonProcessingException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.ai.chat.ChatResponse;
import org.springframework.ai.chat.messages.UserMessage;
import org.springframework.ai.chat.prompt.Prompt;
import org.springframework.ai.openai.OpenAiChatClient;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
import
org.springframework.web.servlet.mvc.method.annotation.SseEm
itter;

import java.io.IOException;

@RestController
public class ChatController {

    private static final Logger logger =
LoggerFactory.getLogger(ChatController.class);
```



图像理解请求

```
package com.demo.aitest.controller.request;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

import java.util.List;

@JsonInclude(JsonInclude.Include.NON_NULL)
@Data
public class MultimodalRequest {
    private String model;
    private List<Message> messages;
    @JsonProperty(value = "stream_options")
    private StreamOptions streamOptions;
    @JsonProperty(value = "parallel_tool_calls")
    private boolean parallelToolCalls;
    private boolean stream;

    @JsonInclude(JsonInclude.Include.NON_NULL)
    @Data
    public static class Message {
        private String role;
        @JsonProperty(value = "content")
        private List<Content> content;
    }

    @JsonInclude(JsonInclude.Include.NON_NULL)
```

文本生图请求

```
package com.demo.aitest.controller.request;

public class ImageGenerationRequest {

    private String model;

    private String prompt;

    private Integer n;

    private String size;

}
```

ImageController

```
package com.demo.aitest.controller;

import
com.demo.aitest.controller.request.ImageGenerationRequest;
import
com.demo.aitest.controller.request.MultimodalRequest;
import com.demo.aitest.service.HttpClientService;
import
com.fasterxml.jackson.core.JsonProcessingException;
import org.apache.commons.lang3.StringUtils;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class ImageController {

    @Value("${spring.ai.openai.chat.options.model}")
    private String chatModel;

    @Value("${spring.ai.openai.image.options.model}")
```



Embedding Controller

```
package com.demo.aitest.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.ai.embedding.EmbeddingResponse;
import
org.springframework.ai.openai.OpenAiEmbeddingClient;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RestController;

import java.util.Collections;

@RestController

public class EmbeddingController {

    private static final Logger logger =
LoggerFactory.getLogger(EmbeddingController.class);

    @Autowired

    private OpenAiEmbeddingClient openAiEmbeddingClient;
```

## Rerank 请求

```
package com.demo.aitest.controller.request;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

@Data
public class RerankRequest {

    private String query;

    private String[] documents;

    private String model;

    @JsonProperty(value = "top_n")
    private Integer topn;
}
```

## Rerank Controller

```
package com.demo.aitest.controller;

import com.demo.aitest.controller.request.RerankRequest;
import com.demo.aitest.service.HttpClientService;
import
com.fasterxml.jackson.core.JsonProcessingException;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.http.*;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RestController;

@RestController
@Slf4j
public class RerankController {

    @Value("${spring.ai.openai.rerank.options.model}")
    private String rerankModel;

    private final HttpClientService httpClientService;
```

访问地址:

- 文本生成: GET <http://localhost:9090/chat?message=Hello>
- 文本生图: <http://localhost:9090/image/generation>

```
{"model": "xirang_model_id", "prompt": "A cute baby sea otter", "n": 1, "size": "512x512"}
```

图像理解: POST <http://localhost:9090/image/understanding>

```
{"model": "88003ac1ca7a4e4e8efa7caee648323b", "messages": [{"role": "user", "content": [{"type": "text", "text": "描述下这张图"}, {"type": "image_url", "image_url": {"url": "https://hangzhou7.zos.ctyun.cn/keywords-txt/20250708181010/184-200x200.jpg"}]}]}, "stream_options": {"include_usage": true}, "parallel_tool_calls": true, "stream": false}
```

embedding 向量化: GET <http://localhost:9090/embedding?text=A cute baby sea otter>

rerank 重排序: POST <http://localhost:9090/rerank>

```
{"query": "Python 教程", "documents": ["Python 基础语法", "Python 高级特性", "Java 编程入门"], "model": "xirang_model_id", "top_n": 2}
```

## GoLang

注意：

- golang 版本要求 1.18 以上；
- go.mod 引用相关类库 go get [github.com/sashabaranov/go-openai](https://github.com/sashabaranov/go-openai)。

文本生成

```
package main

import (
    "context"
    "fmt"
    openai "github.com/sashabaranov/go-openai" // 统一使
用这个库
)

func main() {
    baseUrl := "https://wishhub-x6.ctyun.cn/v1" // api前
缀地址
    appkey := "your_app_key" // 替换成自己的App Key
    modelId := "xirang_model_id" // 替换成自己要用的模型
    prompt := "你是谁" //
对话问题

    // 创建自定义配置
    config := openai.DefaultConfig(appkey)
    config.BaseURL = baseUrl

    // 使用配置创建客户端
    client := openai.NewClientWithConfig(config)

    // 构建请求参数
    resp, err := client.CreateChatCompletion(
        context.Background(),
        openai.ChatCompletionRequest{
            Model: modelId,
            Messages: []openai.ChatCompletionMessage{
```

图像理解

```
package main

import (
    "bytes"
    "crypto/tls"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
)

func main() {
    url := "https://wishhub-
x6.ctyun.cn/v1/chat/completions"
    apiKey := "your_app_key" // 替换为自己的App key
    modelId := "xirang_model_id" // 替换为实际的model Id

    // 创建不验证证书的 HTTP 客户端
    tr := &http.Transport{
        TLSClientConfig: &tls.Config{InsecureSkipVerify:
true},
    }
    client := &http.Client{Transport: tr}

    // 构建请求负载
    payload := map[string]interface{}{
        "model": modelId,
        "messages": []map[string]interface{}{
            {
```



Image 文本生图

```
package main

import (
    "crypto/tls"
    "encoding/base64"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
    "path/filepath"
    "strings"
    "time"
)

type ImageResponse struct {
    Created int `json:"created"`
    Data    []struct {
        URL      string `json:"url"`
        B64JSON  string `json:"b64_json"`
    } `json:"data"`
}

func generateImage(prompt, apiKey, modelId, outputDir,
size string) (string, error) {
    // 创建输出目录
    if err := os.MkdirAll(outputDir, os.ModePerm); err
!= nil {
```



embedding 文本向量化

```
package main

import (
    "context"
    "fmt"
    openai "github.com/sashabaranov/go-openai"
)

func main() {
    baseUrl := "https://wishhub-x6.ctyun.cn/v1" // API前
    缀地址
    appkey := "your_app_key" // 替换成自己的App Key
    modelId := "xirang_model_id" // 替换成实际的模型ID
    text := "测试文本" //
    需要生成嵌入向量的文本

    // 创建自定义配置
    config := openai.DefaultConfig(appkey)
    config.BaseURL = baseUrl

    // 创建客户端
    client := openai.NewClientWithConfig(config)

    // 构建Embedding请求
    resp, err := client.CreateEmbeddings(
        context.Background(),
        openai.EmbeddingRequest{
            Model: openai.EmbeddingModel(modelId),
            Input: []string{text}, // 注意Input是字符串切片
        }
    )
}
```

rerank 重排序

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
)

func main() {
    apiURL := "https://wishhub-x6.ctyun.cn/v1/rerank"
    apiKey := "your_app_key" // 替换成自己的App Key
    modelID := "xirang_model_id" //替换成实际的modelId

    // 请求数据
    requestData := map[string]interface{}{
        "query":      "Python教程",
        "documents": []string{"Python基础语法", "Python高级
特性", "Java编程入门"},
        "model":      modelID,
        "top_n":      2,
    }

    // 编码为 JSON
    jsonData, _ := json.Marshal(requestData)

    // 创建请求
    req, _ := http.NewRequest("POST", apiURL,
```

## PHP

### 注意

- php 版本要求 8.3.13 以上;
- 安装 composer 组件;

运行以下命令:

```
composer require openai-php/client  
composer dump-autoload
```

文本生成

```
<?php

require __DIR__ . '/vendor/autoload.php';

use OpenAI\Client;

$url="https://wishhub-x6.ctyun.cn/v1"; //api前缀地址
//从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

$appkey = "your_app_key";
$modelId="xirang_model_id"; //替换成自己要用的模型
$prompt="你好，介绍一下自己"; //对话问题

$client = OpenAI::factory()
    ->withApiKey($appkey)
    ->withBaseUri($url)
    ->make();

$response = $client->chat()->create([
    'model' => $modelId,
    'messages' => [
        ['role' => 'user', 'content' => $prompt],
    ],
]);

foreach ($response->choices as $result) {
    echo $result->message->content;
}

?>
```

图像理解

```
<?php
$url = 'https://wishhub-
x6.ctyun.cn/v1/chat/completions';
$apiKey = 'your_app_key';
$modelId = 'xirang_model_id';
// 请求负载
$payload = [
    'model' => $modelId,
    'messages' => [
        [
            'role' => 'user',
            'content' => [
                [
                    'type' => 'text',
                    'text' => '描述下这张图'
                ],
                [
                    'type' => 'image_url',
                    'image_url' => [
                        'url' =>
'https://hangzhou7.zos.ctyun.cn/keywords-
txt/20250708181010/184-200x200.jpg'
                    ]
                ]
            ]
        ]
    ],
    'stream_options' => [
        'include_usage' => true
    ]
];
```



Image 文本生图

```
<?php
function generateImage($prompt, $apiKey, $outputDir =
'images', $size = '1024x1024') {
    // 创建输出目录
    if (!is_dir($outputDir) && !mkdir($outputDir, 0777,
true)) {
        throw new Exception("创建目录失败: $outputDir");
    }

    // API配置
    $url = 'https://wishhub-
x6.ctyun.cn/v1/images/generations';

    // 构建请求体
    $data = [
        'model' => 'xirang_model_id', // 替换为实际的
modelId
        'prompt' => $prompt,
        'n' => 1,
        'size' => $size
    ];

    // 使用 cURL 发送请求
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_HTTPHEADER, [
        'Authorization: Bearer ' . $apiKey,
```

embedding 文本向量化

```
<?php

require __DIR__ . '/vendor/autoload.php';

use OpenAI\Client;

$baseUrl = "https://wishhub-x6.ctyun.cn/v1"; // API前缀地
址
$appkey = "your_app_key"; // 替换成自己的App Key
$modelId = "xirang_model_id"; // 替换成Embedding模型ID
$text = "这是一段测试文本"; // 需要生成嵌入向量的文本

$client = OpenAI::factory()
    ->withApiKey($appkey)
    ->withBaseUri($baseUrl)
    ->make();

// 调用Embedding API
$response = $client->embeddings()->create([
    'model' => $modelId,
    'input' => $text, // 支持字符串或字符串数组
]);

// 处理结果
$embedding = $response->embeddings[0]->embedding; // 获
取第一个文本的嵌入向量
$tokenUsage = $response->usage->totalTokens; // 获取使用
的令牌数

// 打印结果
echo "嵌入向量维度: " . count($embedding) . "\n";
```

rerank 重排序

```
<?php

$apiUrl = "https://wishhub-x6.ctyun.cn/v1/rerank";
$apiKey = "your_app_key"; //换成自己的App Key
$modelId = "xirang_model_id"; // 换成实际的model Id

// 请求数据
$data = [
    'query' => 'Python教程',
    'documents' => ['Python基础语法', 'Python高级特性',
'Java编程入门'],
    'model' => $modelId,
    'top_n' => 2
];

// 准备请求
$options = [
    'http' => [
        'method' => 'POST',
        'header' => "Authorization: Bearer $apiKey\r\n"
        .
        "Content-Type:
application/json\r\n",
        'content' => json_encode($data),
        'timeout' => 10 // 超时时间
    ]
];

// 发送请求
$content = stream_context_create($options);
```

## Nodejs

注意

测试用的node版本 $\geq$ v17.9.1,npm版本 $\geq$ 8.11.0;

运行前需执行一下命令:

```
npm init -y
```

```
npm install openai在生成的package.json文件中添加"type":"module";
```

替换baseUrl,appkey,modelId,prompt等字段的值。

文本生成

```
import OpenAI from 'openai';

const baseUrl="https://wishhub-x6.ctyun.cn/v1" //api前缀
地址
//从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

const appkey= "your_app_key"

const modelId="xirang_model_id" //模型id

const prompt="你好啊,给一篇100字左右的作文" //对话问题

const client = new OpenAI({
  apiKey: appkey,
  baseUrl:baseUrl
});

async function chatCompletion() {
  const completion = await
client.chat.completions.create({
  model: modelId,
  messages: [{ role: 'user', content: prompt }],
});

console.log(completion.choices[0]?.message?.content);
}

chatCompletion();
```

图像理解

```
import https from 'https';
import axios from 'axios';

const url = 'https://wishhub-
x6.ctyun.cn/v1/chat/completions';
const apiKey = 'your_app_key';

// 创建不验证证书的 HTTPS Agent
const agent = new https.Agent({
  rejectUnauthorized: false
});

// 请求负载
const payload = {
  model: 'xirang_model_id',
  messages: [
    {
      role: 'user',
      content: [
        {
          type: 'text',
          text: '描述下这张图'
        },
        {
          type: 'image_url',
          image_url: {
            url:
'https://hangzhou7.zos.ctyun.cn/keywords-
txt/20250708181010/184-200x200.jpg'
```

Image 文本生图

```
import https from 'https';
import fs from 'fs';
import path from 'path';
import { fileURLToPath } from 'url';

// 获取当前文件的目录路径（替代 __dirname）
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

async function generateImage(prompt, apiKey, outputDir =
'images', size = '1024x1024') {
    // 创建输出目录
    const fullOutputDir = path.resolve(__dirname,
outputDir);
    if (!fs.existsSync(fullOutputDir)) {
        fs.mkdirSync(fullOutputDir, { recursive: true
});
    }

    // API配置
    const url = 'https://wishhub-
x6.ctyun.cn/v1/images/generations';

    // 构建请求体
    const postData = JSON.stringify({
        model: 'xirang_model_id', // 替换为实际的model Id
        prompt: prompt,
        n: 1,
        size: size
```

embedding 文本向量化

```
import OpenAI from 'openai';

const baseUrl = "https://wishhub-x6.ctyun.cn/v1"; // API
前缀地址
const appkey = "your_app_token"; // 替换为你的实际API密钥
const modelId = "xirang_model_id"; // 替换为支持Embedding
的模型ID

const text = "这是一段测试文本"; // 需要生成嵌入向量的文本

const client = new OpenAI({
  apiKey: appkey,
  baseUrl: baseUrl
});

async function createEmbedding() {
  try {
    // 调用Embedding API

    const embedding = await client.embeddings.create({
      model: modelId,
      input: text, // 支持字符串或字符串数组
    });

    // 处理结果

    const vector = embedding.data[0].embedding; // 获取嵌
入向量
    const tokenUsage = embedding.usage.total_tokens; //
获取令牌使用量

    // 打印结果

    console.log(`嵌入向量维度: ${vector.length}`);
```



rerank 文本向量化

```
import https from 'https';
import axios from 'axios';

const apiUrl = "https://wishhub-x6.ctyun.cn/v1/rerank";
const apiKey = "your_app_key";
const modelId = "xirang_model_id";

// 请求数据
const requestData = {
  query: "Python教程",
  documents: ["Python基础语法", "Python高级特性", "Java编程入门"],
  model: modelId,
  top_n: 2
};

// 发送请求
axios.post(apiUrl, requestData, {
  headers: {
    'Authorization': `Bearer ${apiKey}`,
    'Content-Type': 'application/json'
  },
  timeout: 10000 // 10秒超时
})
.then(response => {
  // 处理结果
  const results = response.data.results;

  console.log(`查询: '${requestData.query}'`);
```

## cURL

注意：

以下curl脚本在Linux (MAC)和windows环境均可运行

参照第一步替换公共字段appkey和modelId的值

其他字段的替换方式如下：

- 文本生成的示例代码可替换content字段的值
- 图像理解的示例代码可替换image\_url中的图片地址，产品可通过运营后台-banner位上传样例图片
- 文本生图的示例代码替换prompt,n,size为文本描述，图像数和图片大小
- embedding的示例代码可替换input为字符串或数组类型
- rerank的示例代码可替换query和documents为检索的关键词和文本数组

文本生成

```
curl --location "https://wishhub-
x6.ctyun.cn/v1/chat/completions" --header "Content-Type:
application/json; charset=utf-8" --header "Authorization:
Bearer your_app_key" --data "{\"messages\": [{\"role\":
\"user\", \"content\": \"你好\"}], \"model\":
\"xirang_model_id\"}"
```

#### 图像理解

```
curl --location "https://wishhub-
x6.ctyun.cn/v1/chat/completions" \
--header "Authorization: Bearer
adc8765ffc324f4885dcb444f06af93a" \
--header "Content-Type: application/json" \
--data "{\"model\":
\"88003ac1ca7a4e4e8efa7caee648323b\", \"messages\":
[{\\"role\": \"user\", \"content\": [{\\"type\": \"text\",
\"text\": \"描述下这张图\"}, {\\"type\": \"image_url\",
\"image_url\": {\\"url\":
\"https://hangzhou7.zos.ctyun.cn/keywords-
txt/20250708181010/184-200x200.jpg\"}]}]}],
\"stream_options\": {\\"include_usage\": true},
\"parallel_tool_calls\": true, \"stream\": false}"
```

#### Image 文本生图

```
curl "https://wishhub-x6.ctyun.cn/v1/images/generations"
--header "Content-Type: application/json; charset=utf-8" --
header "Authorization: Bearer your_app_key" --header
"Accept: */*" --header "Accept-Encoding: gzip, deflate, br"
--data "{\"model\": \"xirang_model_id\", \"prompt\": \"A
cute baby sea otter\", \"n\": 1, \"size\": \"512x512\"}"
```

embedding 文本向量化

```
curl --location "https://wishhub-
x6.ctyun.cn/v1/embeddings" --header "Authorization: Bearer
your_app_token" --header "User-Agent: PostmanRuntime-
ApiPostRuntime/1.1.0" --header "Accept-Encoding: gzip,
deflate" --header "Content-Type: application/json" --data
"{\"input\": [\"A cute baby sea otter\"], \"model\":
\"xirang_model_id\"}"
```

rerank 重排序

```
curl --location "https://wishhub-x6.ctyun.cn/v1/rerank"  
--header "Accept: */*" --header "Accept-Encoding: gzip,  
deflate, br" --header "Authorization: Bearer your_app_key"  
--header "Content-Type: application/json" --data  
"{\"query\": \"Python教程\", \"documents\": [\"Python基础语法  
\", \"Python高级特性\", \"Java编程入门\"], \"model\":  
\"xirang_model_id\", \"top_n\": 2}"
```

## 4. 用户指南

### 4.1 模型体验

#### 4.1.1. 模型广场

模型广场汇集了当下丰富的开源大模型，可查看及体验模型在文本、图像等方面的先进能力。模型广场包含文本大模型、视觉大模型、向量化模型，后续还会支持更多大模型，请敬请期待。

##### 操作步骤

1. 登录模型推理服务。
2. 在左侧导航栏选择“模型广场”。
3. 点击目标模型卡片进入模型详情页，可查看模型介绍、API文档和任务记录。
  - a. 点击“任务记录”，可查看该模型的任务历史，包括模型精调、模型评估、模型部署等任务。
4. 在模型卡片单击“部署”可进行一键部署操作。当按钮置灰时，表示模型暂不支持该任务。
  - a. 关于如何部署模型，请参见在线推理-我的服务。

#### 4.1.2. 体验中心

提供在线体验的功能，可即刻体验模型效果。

##### 前提条件

- 体验预置服务：该模型具有免费 Token 额度且在有效期内，或已开通商用。
- 体验我的服务：该模型服务处于“运行中”或“更新中”状态。

##### 操作步骤

1. 登录模型推理服务。
2. 任选以下方式进行体验。
  - a. 在左侧导航栏选择“体验中心”。
  - b. 在左侧导航栏选择“模型广场”，单击目标模型卡片的“体验”按钮。
  - c. 在左侧导航栏选择“在线推理”，单击预置服务操作列的“体验”按钮。
3. 在“体验中心”页面选择模型类型和模型服务。

- a. 文本生成类模型支持调整模温度、多样性等参数设置。

参数	说明
温度	Temperature控制生成文本的多样性。较高的温度值会使生成的文本更加随机和多样化，而较低的温度值会使生成的文本更加确定和一致。
多样性	TopP影响输出文本的多样性，取值越大，生成文本的多样性越强。
重复惩罚	Frequency_penalty影响模型生成重复词汇的倾向。通过增加重复词汇的惩罚权重，降低模型逐字重复的可能性。
TopK	TopK指在模型的输出中选择概率最高的前K个结果。有助于加快生成过程，并且能够提高生成文本的质量。
系统人设	设定模型的行为和背景，告知模型需要扮演的角色。例如：“假如你是一个AI助手”。

- b. 文本生图类模型支持设置绘画描述、图片风格、图片比列和迭迭代步数等参数。

4. 在对话输入问题即可查看返回结果。
5. 点击右上角“历史记录”，系统会展开历史的对话记录，可以查看统计大模型的回答质量，保存上限为 200 条。

## 4.2 模型服务

### 4.2.1 服务接入

服务接入模块可以将预置服务及我的服务打包为服务组，生成 AppKey 供用户调用。

#### 创建服务组

1. 登录模型推理服务。
2. 在左侧导航栏选择“服务接入”。
3. 点击“创建服务组”，进入创建页。填写服务名称、服务描述和生效时间等参数，并选择需要关联的服务（支持选择预置服务、我的服务），提交完成创建。
4. 完成创建后，系统会自动创建一个调用服务的密钥，即生成该服务的密钥，即“AppKey”

#### 查看服务组详情

1. 点击服务组卡片【查看详情】，可进入服务组详情页，查看该服务组关联的服务，点击【了解更多】可查看模型详情。

#### 服务组管理

点击服务组卡片对应按钮，可以支持对“AppKey”进行复制和重置。

1. 停用：支持手动“停用”和已过生效期后自动停用。
2. 启用：重新启用已停用的服务组。已过生效期的服务组可一键启用为永久生效服务组，或通过“编辑”延长生效时间后启用。

3. 编辑：编辑服务组名称、描述、生效时间和服务配置。
4. 删除：删除整个服务组。
5. 监控：可以支持停用/启用、编辑、删除服务组和查看服务组监控。

### 账号权限说明

主账号可查看和管理子账号所创建服务组。

## 4.2.2 在线推理

平台在线推理模块提供预置服务和我的服务两类，预置服务是平台已部署好的服务，无等待时间，可直接调用；我的服务支持用户所部署的模型。

### 预置服务

1. 登录模型推理服务。
2. 在左侧导航栏选择“在线推理”，选择“预置服务”。
3. 在预置服务列表可查看目前平台预置的所有服务，预置服务支持免费试用和付费使用。

**免费试用：**每个模型被赋予不等的免费额度和免费试用期限，具体免费额度可在页面中查看，免费期限从第一次使用该模型开始计算。免费额度用完或到期后，可以付费开通服务。

- 共享机制：主/子账号共用免费额度及有效期周期额度
- 不可分配：不支持主账号向子账号分配额度
- 有效期触发：主账号任一账号首次使用模型服务即启动2周有效期倒计时

**付费使用：**部分模型支持开通付费token服务。暂不支持开通付费token的模型可转为部署为“我的服务”。

- 独立开通：各账号可单独开通付费服务，操作互不影响。
- 状态独立：各账号页面显示自身付费状态，不关联其他账号。
- 统一扣费：所有子账号消费均从主账号余额扣除。

举例：子账号 A 已开通付费，成功后即可付费使用 tokens；子账号 B 未开通，则子账号 B 不可付费使用 tokens。

### 我的服务

1. 登录模型推理服务。
2. 在左侧导航栏选择“在线推理”，选择“我的服务”。
3. 点击“部署我的模型”可部署预置模型服务。

### 新建我的服务

参数类型	参数名称	说明
模型服务信息	服务名称	必填，填写服务的名称

	服务地址	服务接口地址
模型信息	选择模型	必填，选择待发布的模型
	训练框架	必选，根据所选模型显示支持的框架。目前支持 PyTorch 和 MindSpore
资源配置	资源规格	必填，选择资源的规格。具体资源规格参见 <a href="#">规格详情</a> 。 说明：为保证模型性能，资源规格列表已自动过滤不可用规格。
	实例数量	必填，部署该服务的实例数量，默认为 1。
	单实例部署机器数	一个实例需要部署的机器数量。参数量级较大的模型需要进行分布式推理部署。

### 管理我的服务

●在列表可查看模型是否部署成功，在操作列可进行模型查看、更新、停止、重启、修改、上下线、删除等操作。停止服务后计费也会停止，再次启动服务即可开通计费。

●操作列点击【查看】可进入该服务的详情页，查看部署的模型列表、服务监控、配置历史、运行记录、事件日志、服务日志。

●状态为运行中的模型服务可正常调用。需要使用 location+modelId+appKey 请求调用。具体调用方式如下：

1. 点击【查看】进入该服务的详情页，可查看该服务的 API 文档，通过详情页中的“modelId”和“接口地址”条目获取 modelId 和 location。
2. 创建或编辑服务组，选择对应服务并提交，通过服务组卡片上的“App Key”条目获取 AppKey。
3. 根据平台规范构造请求，调用对应服务，目前支持部署 Chat 类型的模型，请求样例如下：

```
curl --location '${location}' \  
--header 'Content-Type: application/json' \  
--header 'Authorization: Bearer ${appKey}' \  
--data '{  
    "model": "${modelId}",  
    "messages": [  
        {  
            "role": "user",  
            "content": [  
                {  
                    "type": "text",  
                    "text": "xxx"  
                }  
            ]  
        }  
    ]  
}'
```

### 4.2.3 调用监控

支持查看预置服务的调用总量、调用 token 数和调用失败等数据。

#### 操作步骤

1. 登录模型推理服务。
2. 在左侧导航栏选择“调用监控”。
3. 选择目标预置服务，点击“查看监控”，进入该模型的调用数据详情页，可以查看具体的模型在特定服务组、特定服务中的调用监控情况。
  - a. 点击“调用失败明细”，可以查看调用失败的次数、占比、错误信息等数据。
  - b. 点击“导出”，可以直接导出相关数据到本地。

#### 账号权限说明

主账号可查看子账号所创建服务组的监控。

## 5. 最佳实践

### 5.1 DeepSeek模型调用-快捷版

说明：

目前天翼云息壤的新老用户均可**免费体验 2500 万 Tokens**，限时两周。模型广场支持

DeepSeek-R1、DeepSeek-V3 等多个版本模型的调用。

支持在对话工具（比如 Chatbox、CherryStudio 等）以 OpenAI-API-compatible 的方式快速添加使用。

#### 1.天翼云官网获取模型调用APP key等信息

##### 模型API终端请求地址

请求地址

API终端请求地址：<https://wishhub-x6.ctyun.cn/v1/chat/completions>

通信协议

接口通过 HTTPS 进行通信，保护用户数据的机密性和完整性，确保网络通信的安全性。

请求方法

- 支持的 HTTP 请求方法：POST。
- POST 请求支持的 Content-Type 类型：application/json。

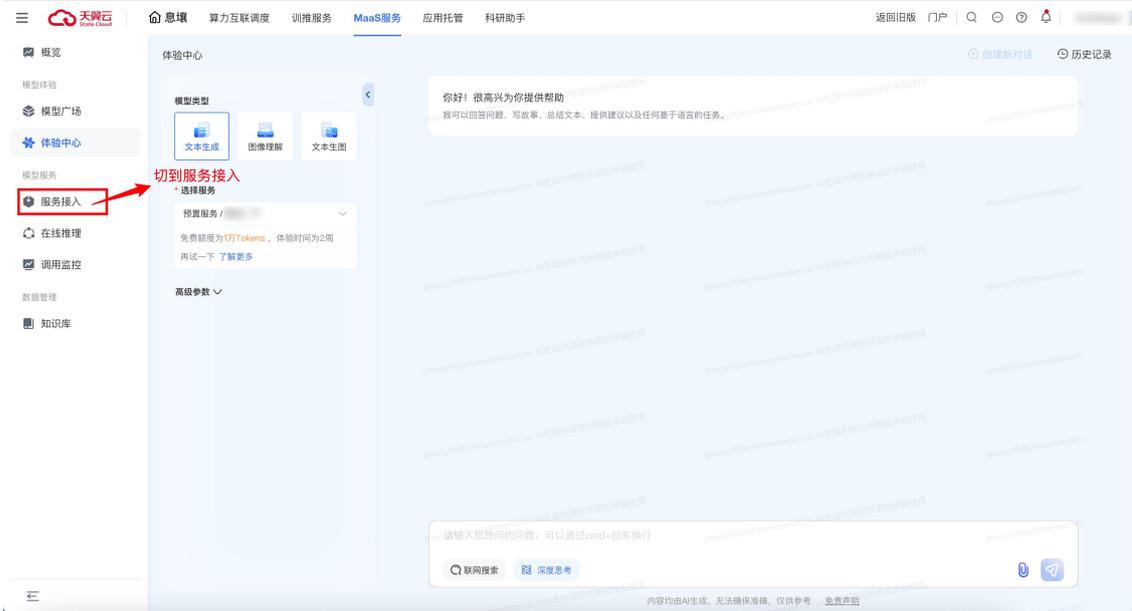
##### 获取模型调用APP key

- 访问天翼云息壤deepseek活动页地址：

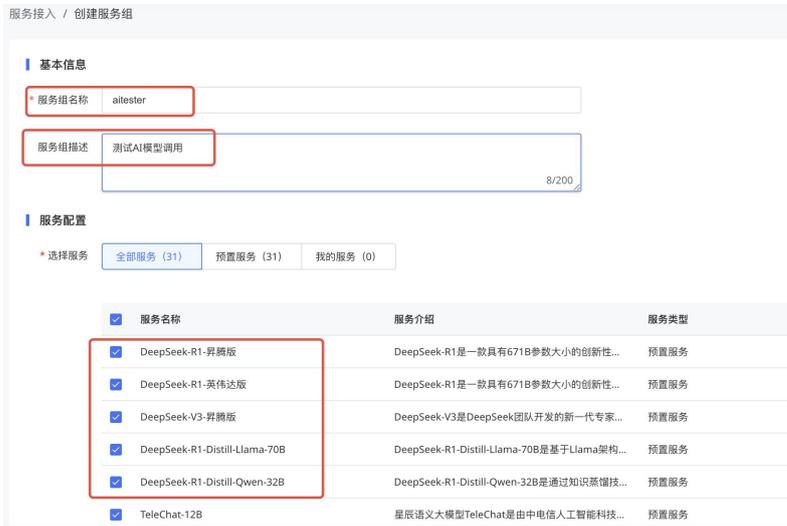
<https://www.ctyun.cn/act/xirang/deepseek>，选择“立即体验”。



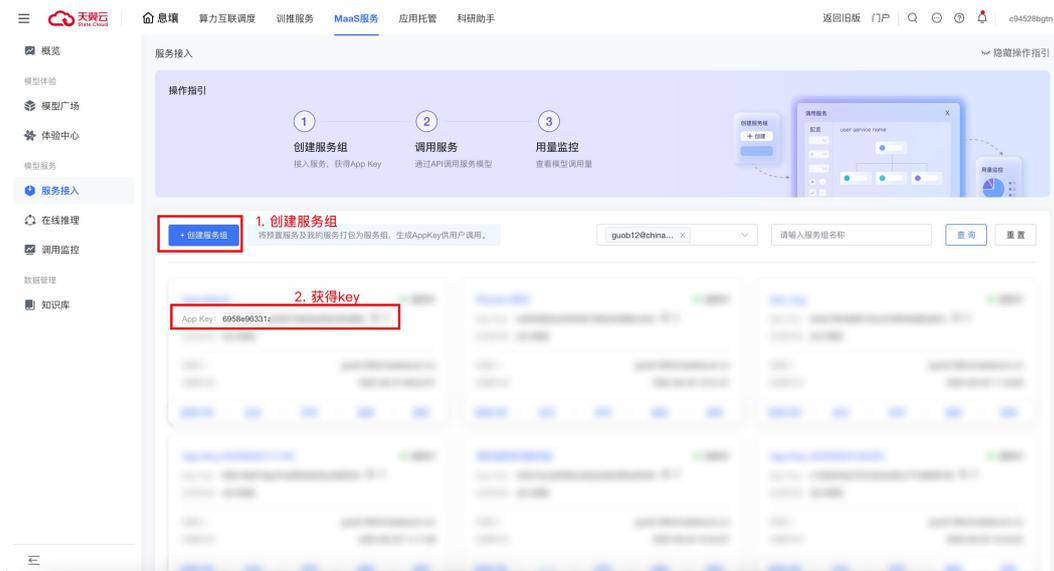
- 登录天翼云账户, 进入“体验中心”后切换到“服务接入”。



- 在“服务接入”下面, 创建建服务组。选择默认勾选的模型即可。

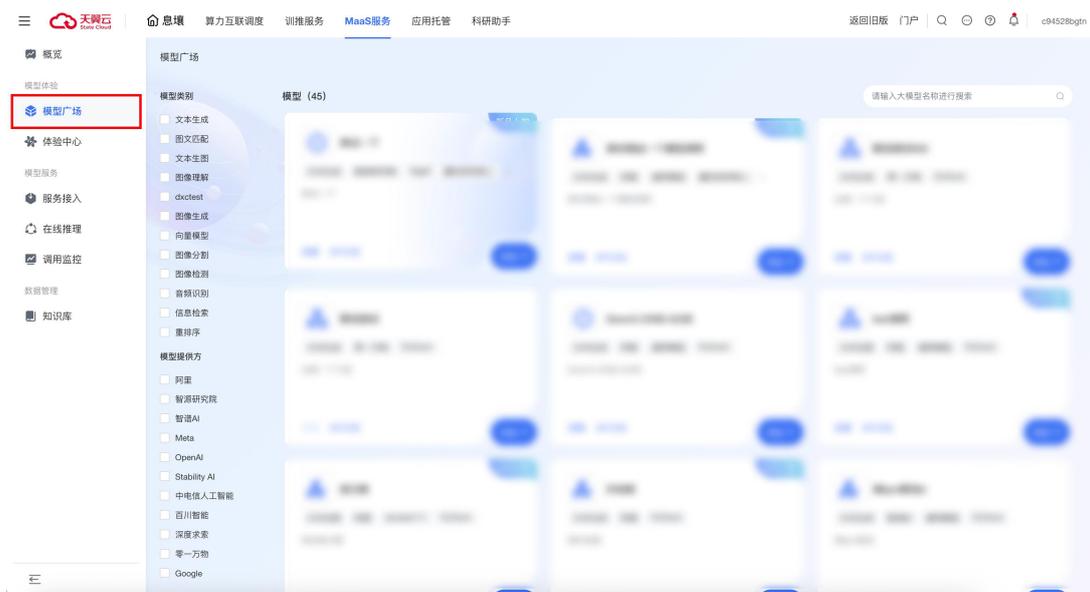


- 创建服务组后，从服务组上面获取APP KEY。



## 获取调用的模型名称

- 左侧导航栏切换到“模型广场”，选择对应想要调用的模型。



- 获取模型ID作为模型名称。

例如下图中模型ID为：4bd107bff85941239e27b1509eccfe98



Dee

pSeek模型版本选择参考：

- DeepSeek-R1：适合需要高级推理能力和高精度的复杂任务，如科研、金融建模等。响应时间相对长。
- DeepSeek-V3：适合需要快速响应和高性价比的通用自然语言处理任务，如智能客服、内容生成等。响应时间相对短。

## 2. 手动验证key的可用性

下列请求为DeepSeek-R1-昇腾版请求样例，如果正常返回deepseek回答，代表没有问题。

### 注意

以下curl请求中替换成自己的APP KEY后类似“--header 'Authorization: Bearer abcdefghxxsssss' \”

```
curl --location 'https://wishhub-
x6.ctyun.cn/v1/chat/completions' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer 替换成自己的APP KEY' \
--data '{
  "messages": [
    {
      "role": "system",
      "content": "You are a test assistant."
    },
    {
      "role": "user",
      "content": "Testing. Just say hi and
nothing else."
    }
  ],
  "model": "4bd107bff85941239e27b1509eccfe98"
}'
```

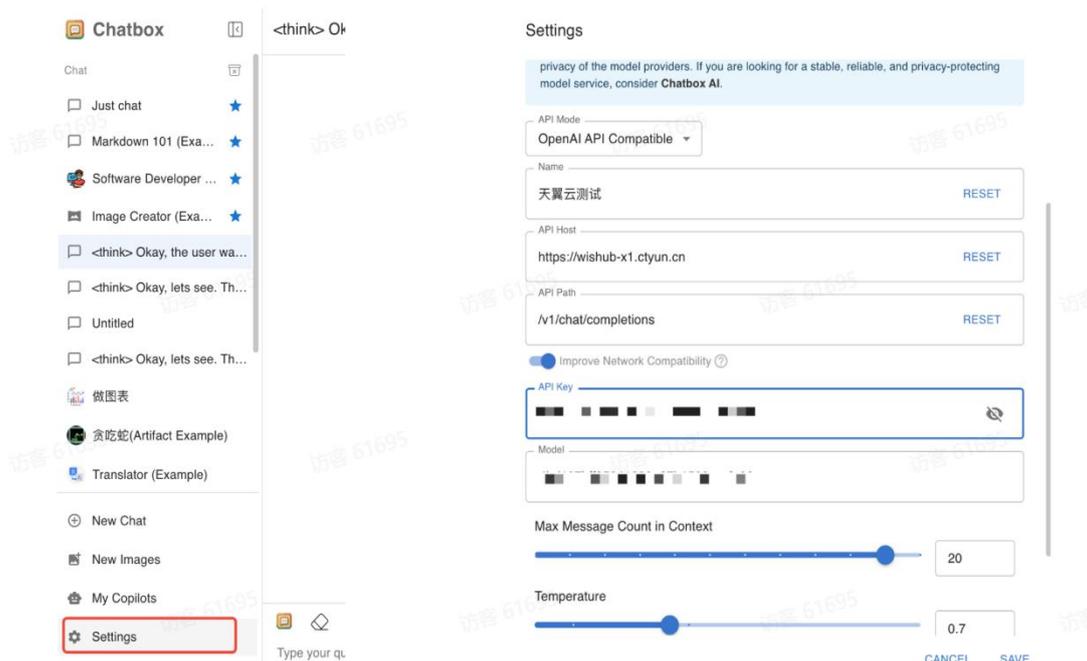
#### 说明

- 部分第三方工具调用可能存在接口不适配问题，建议优先使用以下几款工具。
- 配置下列工具时，API key等同于上述内容中创建的服务组APP key。

#### Chatbox

访问链接：<https://web.chatboxai.app/>

点击Setting，然后配置Api Host、API path、API key和Model，就可以使用了。



## 注意

chatbox web如果api setting配置错误，更改后，仍然会报错。报错如下：



如遇

上述报错，建议重新开启一个对话，再次尝试即可。

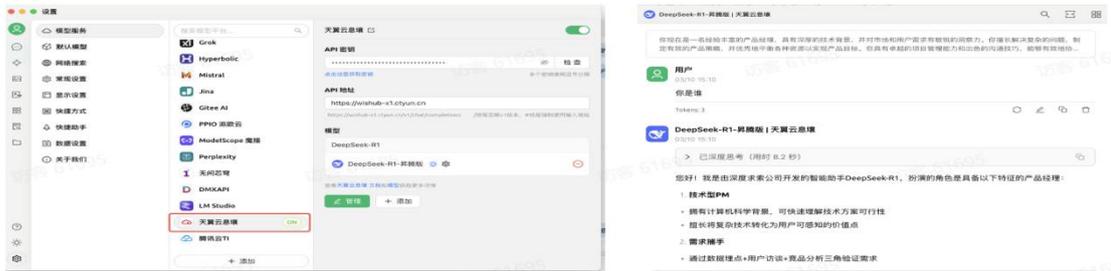


Ch

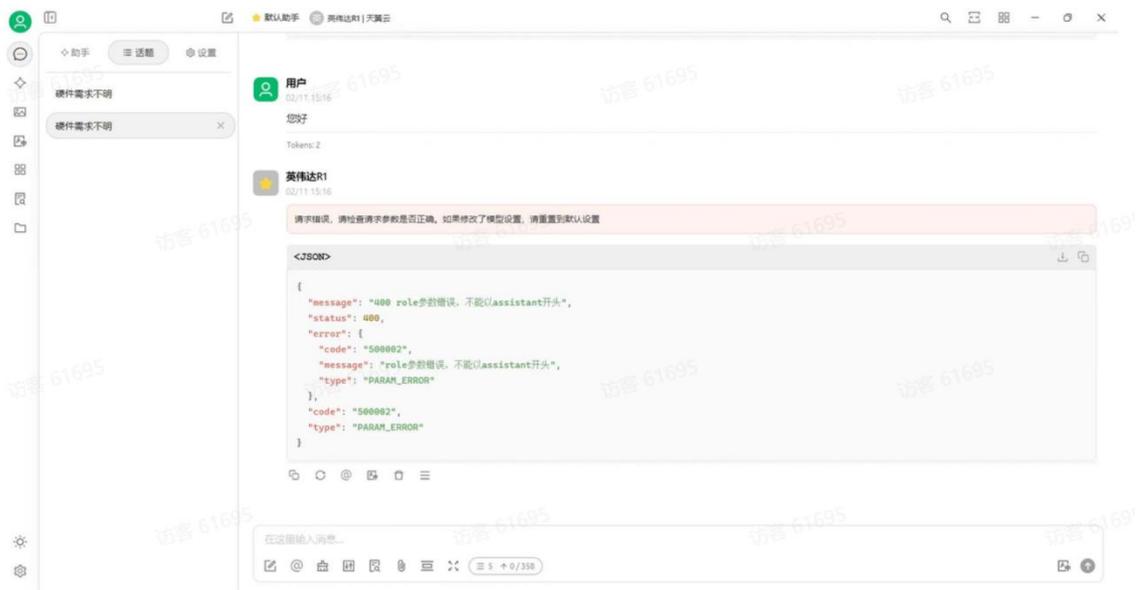
## erryStudio

访问链接: <https://cherry-ai.com/>

按照下图配置即可。



一旦出现以下报错，重新开启一个对话，再次尝试，则可以看到正确的对话。



## Langchain

参考langchain的自定义模型文档通过OpenAI协议调用即可。

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage

def main():
    # 初始化LangChain模型，配置自定义API端点
    llm = ChatOpenAI(
        model_name="9dc913a037774fc0b248376905c85da5", #
        # 自定义模型名称
        # 从环境变量获取API密钥，如果没有设置，也可以直接终端执行
        # export TYY_API_KEY="xxx"
        openai_api_key=os.getenv("XIRANG_API_KEY")
        openai_api_base="https://wishhub-x6.ctyun.cn/v1",
        # API基础URL
        temperature=0.7
    )

    # 创建一个聊天消息
    message = HumanMessage(content="你是谁")

    # 使用LangChain进行对话
    response = llm([message])
    print(response.content)

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

### 说明

- 前置条件：本地下载并安装Docker。
- 访问 Docker 官方网站，点击“Download Docker Desktop”按钮，选择合适的 Windows、macOS 或 Linux版本下载并安装。

### 拉取dify镜像到本地

```
# 如果本地已安装dify可以忽略  
# 假设当前最新版本为 0.15.3  
git clone https://github.com/langgenius/dify.git --branch
```

### 修改配置参数

```
cd docker  
mv .env.example .env
```

### 启动并验证

```
#执行：  
docker compose up -d  
# 等服务启动访问  
http://localhost/apps
```

### 配置天翼云息壤模型

配置方式一：直接配置方式（所有版本均可使用）

通过直接配置OpenAI-API-compatible支持，配置方式如下：



## 配置方式二：使用息壤镜像配置方式（仅支持0.15.3版本）

### 修改env配置

```
cd docker

#在.env文件添加

CUSTOM_MODEL_ENABLED=true

OLLAMA_API_BASE_URL=host.docker.internal:11434
```

修改下api和work的镜像地址，改为增加了xirang模型供应商的dify 0.15.3版本。

修改后需要重启dify。

services:

```
# API service with Xirang model provider
```

api:

```
image: telecom-eci-huadong1-crs-registry-huadong1.crs.ctyun.cn/xirang/dify-with-xirang:0.15.3 # 原来为 langgenius/dify-api:0.15.3
```

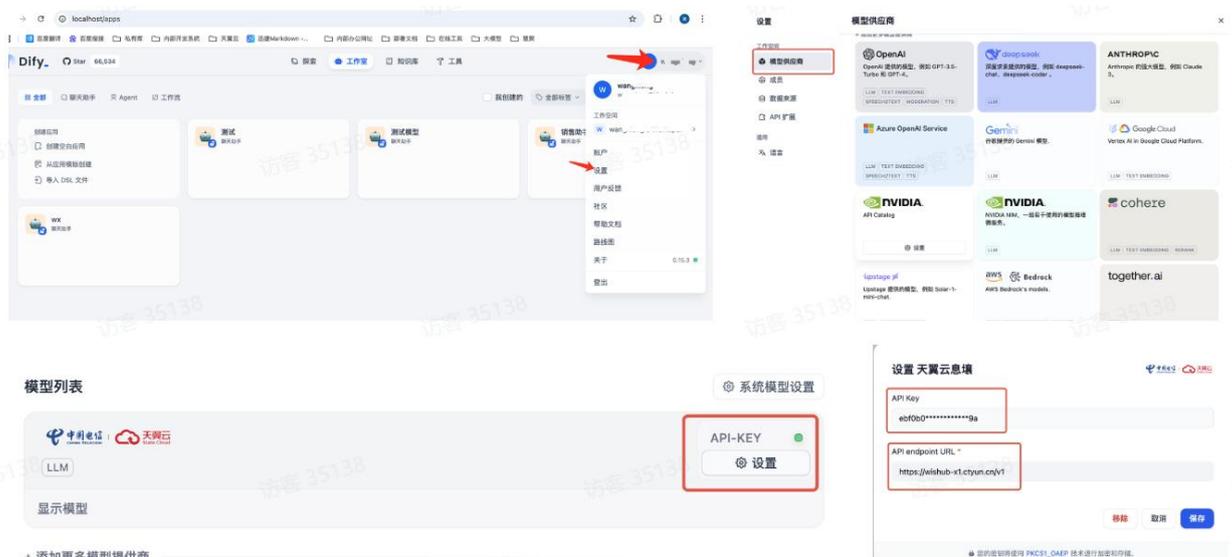
```
# worker service
```

```
# The Celery worker for processing the queue.
```

worker:

```
image: telecom-eci-huadong1-crs-registry-huadong1.crs.ctyun.cn/xirang/dify-with-xirang:0.15.3 # 原来为 langgenius/dify-api:0.15.3
```

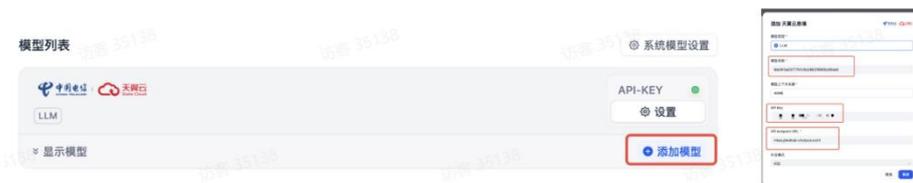
在个人设置里面点击模型供应商，选择天翼云模型。



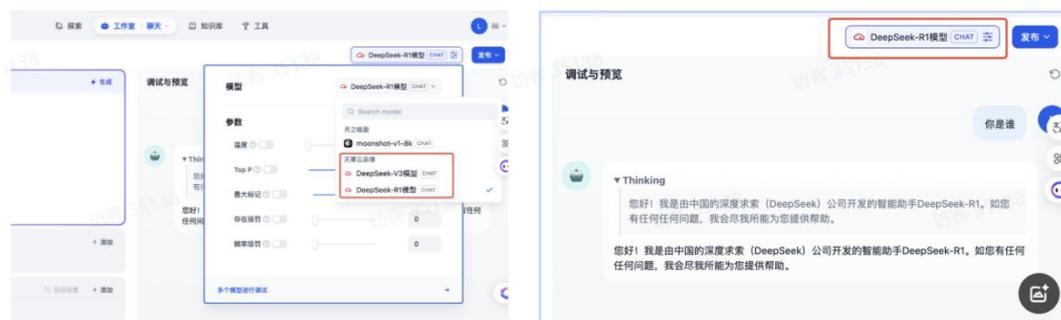
点击保存后，可以看到模型的配置。



其他非预置模型：



配置对话助手



### 3.常见问题（0.15.3版本）

如遇到无法展示问题，可能是因为您的dify版本为最新版，我们目前还不支持，可执行以下操作切换到0.15.3版本并重启服务。

diffy插件需要切换版本重新启动服务才可以达成兼容

```
git clone https://gitcode.com/gh_mirrors/di/dify.git
```

```
cd dify
```

```
git checkout 7796984444191c639bd3c541a44e832b17ad1cae
```

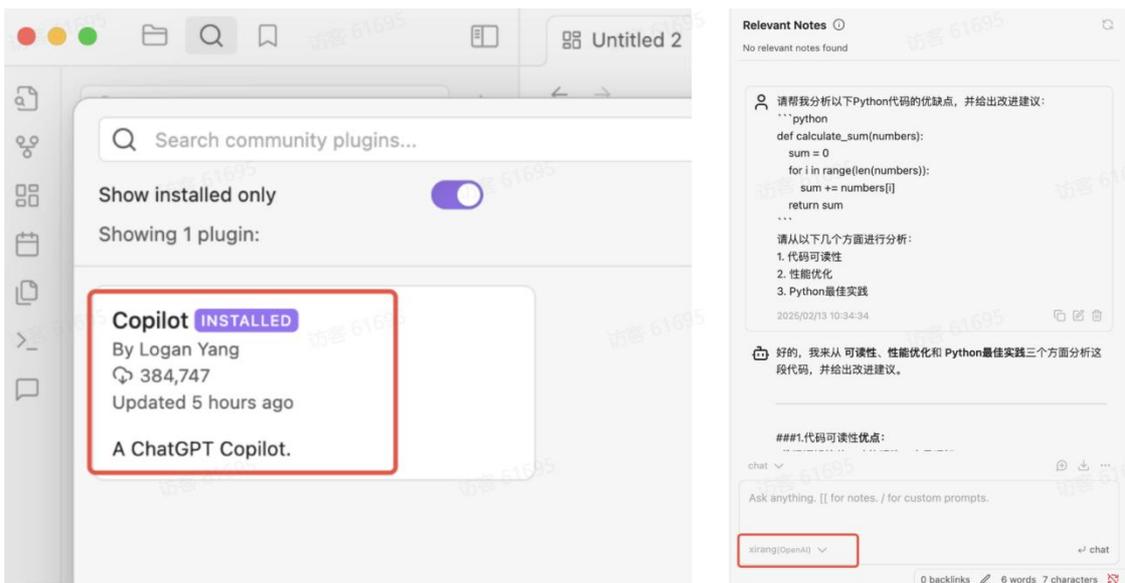
如遇到息壤镜像失败问题，可切换到dify原始镜像，通过配置直接配置OpenAI-API-compatible支持，配置方式如下：



Obs

idian copilot

- 打开 Obsidian，进入设置面板，选择“社区插件”。
- 关闭“安全模式”，点击“浏览”按钮搜索“Copilot for Obsidian”并安装。
- 安装完成后，启用插件。



R

agflow

### 注意

目前息壤model不支持embedding model能力，因此只能作为chat model提供能力。

- 下载镜像git clone <https://github.com/infiniflow/ragflow.git>
- 通过docker安装并启动。

```
cd ragflow/docker  
docker compose -f docker-compose-CN.yml up -d
```

- 检查 RAGFlow 服务是否正常启动。

```
docker logs -f ragflow-server
```

如果看到类似以下输出，则表示启动成功：

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:9380
* Running on http://x.x.x.x:9380
```

打开浏览器配置模型Xinference服务提供商。

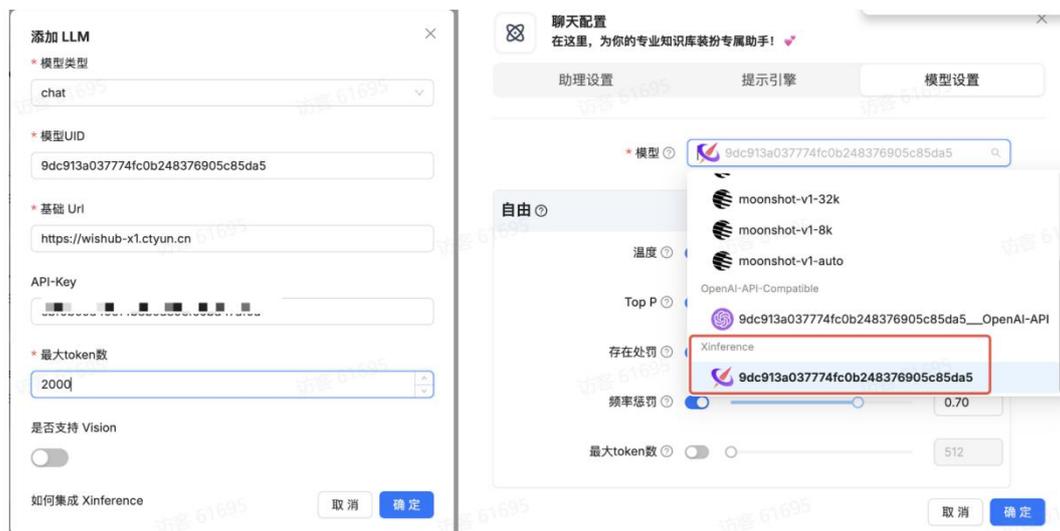
在浏览器中输入服务器的 IP 地址访问 RAGFlow。默认情况下，RAGFlow 使用 HTTP 端口 80，因此无需输入端口号。

选择“Xinference”模型服务商（Xinference是工厂配置，支持自定义模型名称）。



配

置chatmodel的界面如下：



回到对话界面，配置聊天助手：



开启对

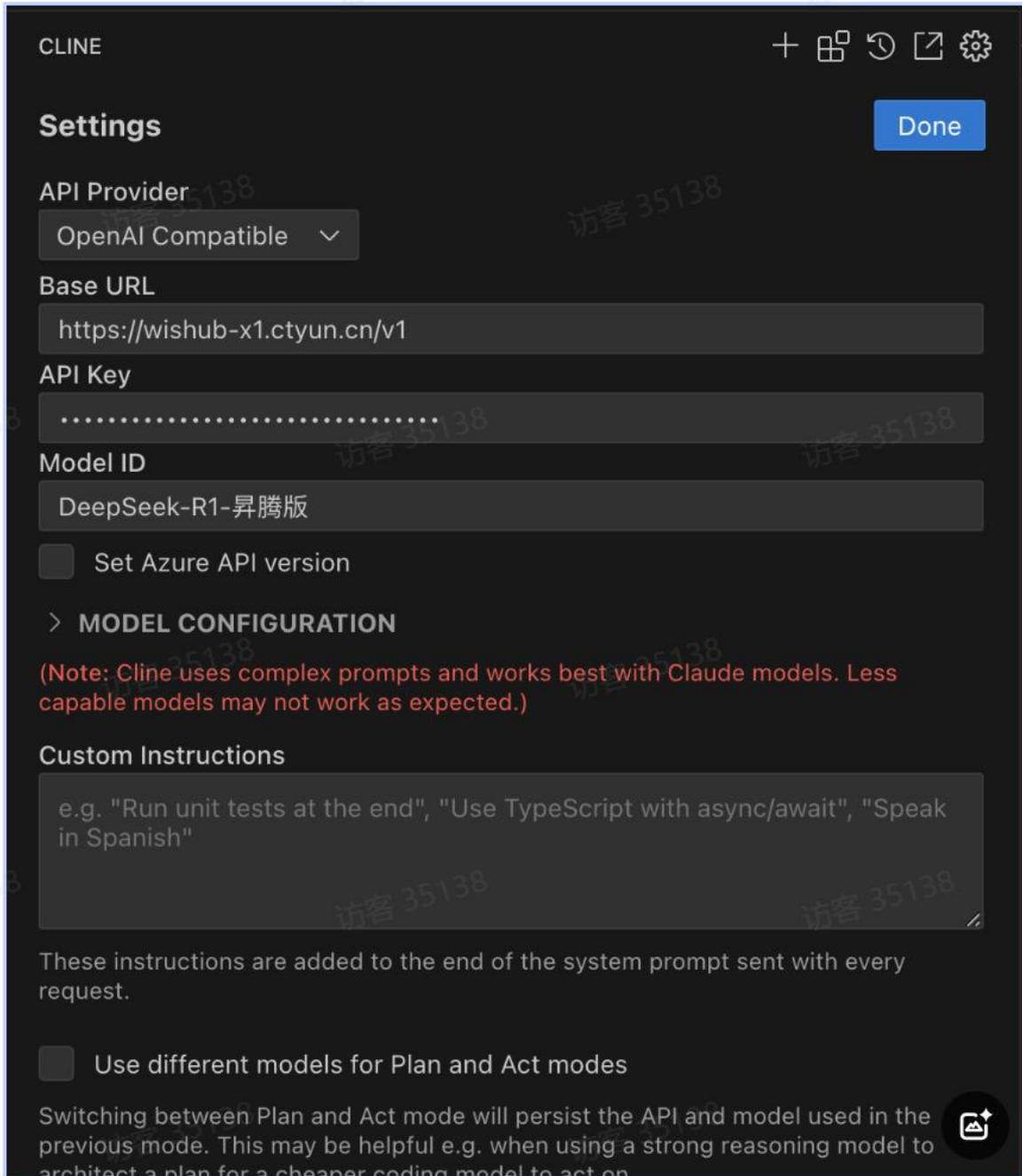
话：



Clin

e

按照下图配置即可。注意baseUrl配置：<https://wishhub-x6.ctyun.cn/v1>



配置完成后即可开启对话：

CLINE + 田 ↻ ↗ ⚙

Task ×

你好，你是谁？

Tokens: ↑ 23.1k ↓ 520

Context Window: 12.1k  128.0k

API Cost: \$0.0000 🗑 11.6 kB

⚠ No workspace detected. Please open Cline in a workspace to use checkpoints.

然后，我回顾我的角色设定：我是Cline，一个高度熟练的软件工程师，具备多种编程语言、框架、设计模式和最佳实践的知识。我的主要职责是帮助用户完成各种任务，提供技术解决方案。

最后，我组织语言，确保回答直接、专业，不使用任何Markdown格式，保持自然流畅。同时，避免使用任何工具，因为这只是一个自我介绍的任务。

你好！我是Cline，一个由深度求索公司开发的智能助手，我擅长帮助用户完成各种任务，包括但不限于编程、数据分析、文档编写和问题解答。如果你有任何需要帮助的地方，随时告诉我！

✓ **API Request** ▾

首先，我需要确定使用哪个工具。用户的问题是关于自我介绍，没有涉及文件操作、系统命令或其他工具。因此，最合适的工具是attempt\_completion，因为它用于在完成任务后向用户展示结果。

接下来，我需要构建attempt\_completion的结构。根据工具的使用指南，result部分是必须的，用来描述任务的结果。我应该简洁明了地介绍自己，避免使用任何Markdown格式，保持纯文本。

然后，我检查是否有需要的参数。attempt\_completion只需要result，可选的command参数在这里不需要，因为不需要执行任何命令来展示结果。

最后，我确保回应符合格式要求，使用XML-style标签，并且内容准确无误。这样，用户就能清楚地了解我是谁，同时遵守了工具使用的规范。

✓ **Task Completed**

I am Cline, an AI assistant created by DeepSeek. I specialize in helping users with programming, data analysis, document writing, and answering questions. Feel free to ask me for assistance with any task you need help with.

Auto-approve: None >

Start New Task

Type a message... ▶

@ openai-compat:DeepSeek-R1-昇腾版 Plan 

## Maxkb

通过docker命令拉取部署

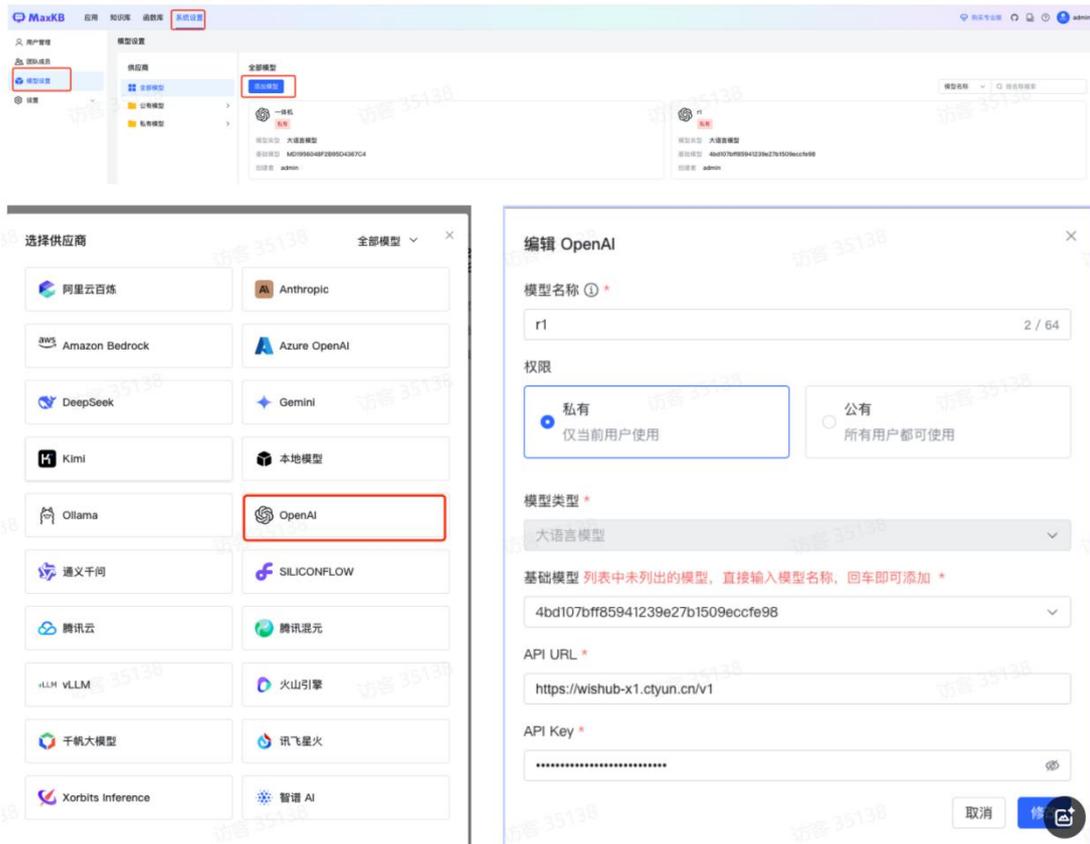
```
docker run -d --name=maxkb --restart=always -p
8080:8080 -v ~/.maxkb:/var/lib/postgresql/data -v
~/.python-packages:/opt/maxkb/app/sandbox/python-packages
https://registry.fit2cloud.com/maxkb/maxkb
```

启动后，浏览器访问<http://localhost:8080>

初始用户名：admin

初始密码：见[https://maxkb.cn/docs/installation/offline\\_installtion/#4](https://maxkb.cn/docs/installation/offline_installtion/#4)

按照下图配置即可。注意baseUri配置：<https://wishhub-x6.ctyun.cn/v1>



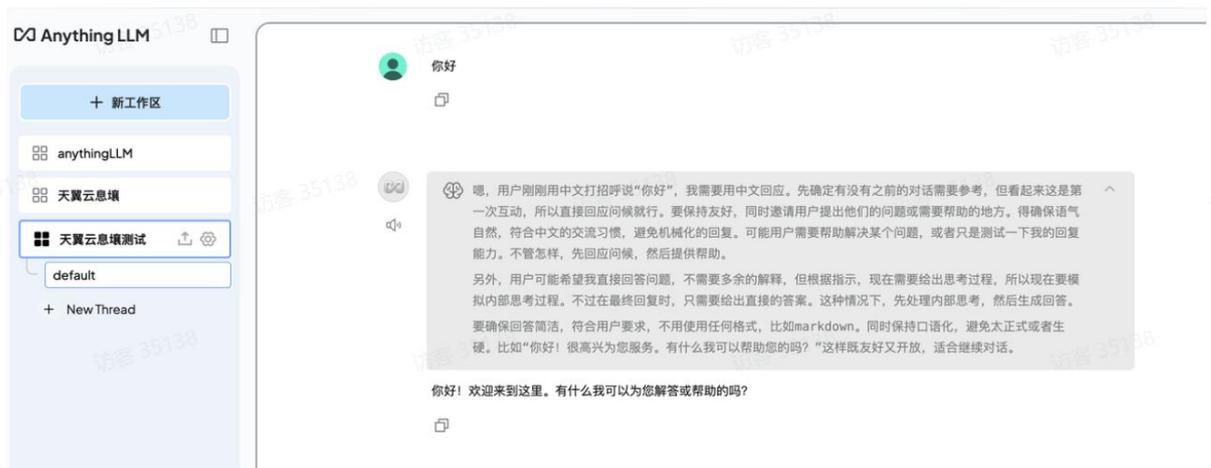
An

## ythingLLM

按照下图配置即可。注意baseUri配置：<https://wishhub-x6.ctyun.cn/v1>



配置完成后即可开启对话：



### 常见报错

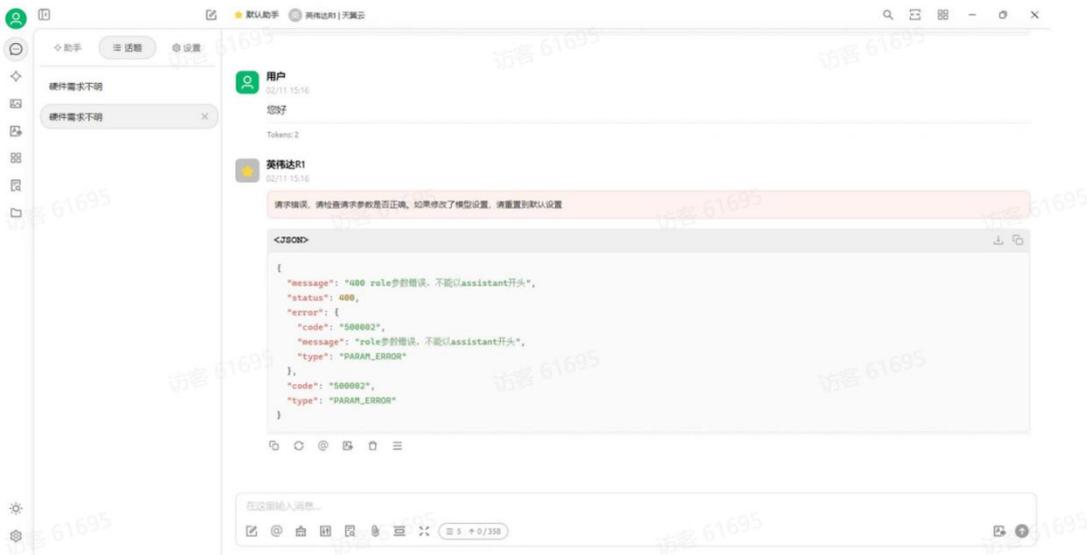
role参数错误，user后应为assistant



- 问题原因：模型setting首次配置错误，更改配置后，因本次对话包含历史信息，仍然会报错。

- 解决方案：开启一个新的对话即可。

400 role参数错误，不能以assistant开头



- 问题原因：模型setting首次配置错误，更改配置后，因为本次对话包含历史错误信息，导致仍然会报错。

- 解决方案：开启一个新的对话即可。

API statuscode429: DEEP\_SEEK\_MODEL\_CONCURRENCY\_LIMIT

报错信息为:

```
[openai_api_compatible]Error:APIrequestfailedwithstatusCode429:{"code":700007,"detail":"Deepseek模型并发已达上限","message":"DEEP_SEEK_MODEL_CONCURRENCY_LIMIT","error":{"code":"700007","message":"Deepseek模型并发已达上限","type":"DEEP_SEEK_MODEL_CONCURRENCY_LIMIT"}}
```

- 问题原因: 大模型并发是大模型的并发能力, 和用户、tokens是否用完都无关, 只和机器负载能力有关。目前单个模型有固定的并发度, 这是所有ds模型共享的并发度, 只能支持固定个请求同时访问。

- 解决方案: 等一段时间再调用, 或增加重试机制。

API 直连调用耗时长

- 问题原因: 大模型API同步耗时长是因为其推理过程复杂, 需要逐个生成token并拼接结果, 且在处理长文本或复杂问题时, 等待完整结果返回的时间过长。而流式API可以在推理过程中逐步返回结果, 用户无需等待全部结果生成, 从而提升交互效率。

- 解决方案: 改为流式调用。

## 5.2 DeepSeek模型调用-专业版

目前天翼云息壤的新老用户均可**免费体验 2500 万 Tokens**, 限时两周。模型广场支持 DeepSeek-R1、DeepSeek-V3 等多个版本模型的调用。支持开发者以 OpenAI-API-compatible 接口方式集成调用。

# 1 天翼云官网获取模型调用API key等信息

## 1. 模型API终端请求地址

请求地址

1. API终端请求地址是<https://wishhub-x6.ctyun.cn/v1/chat/completions>

通信协议

接口通过 HTTPS 进行通信，保护用户数据的机密性和完整性，确保网络通信的安全性。

请求方法

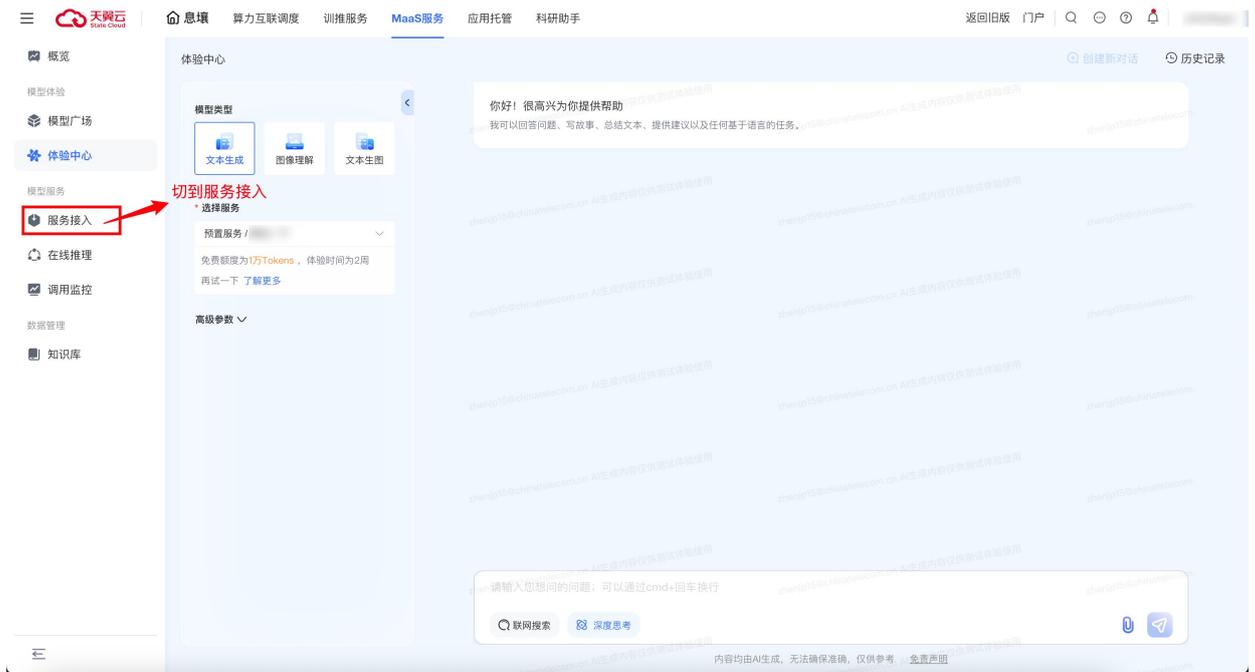
- 支持的 HTTP 请求方法：POST。
- POST 请求支持的 Content-Type 类型：application/json。

## 2. 获取模型调用API key

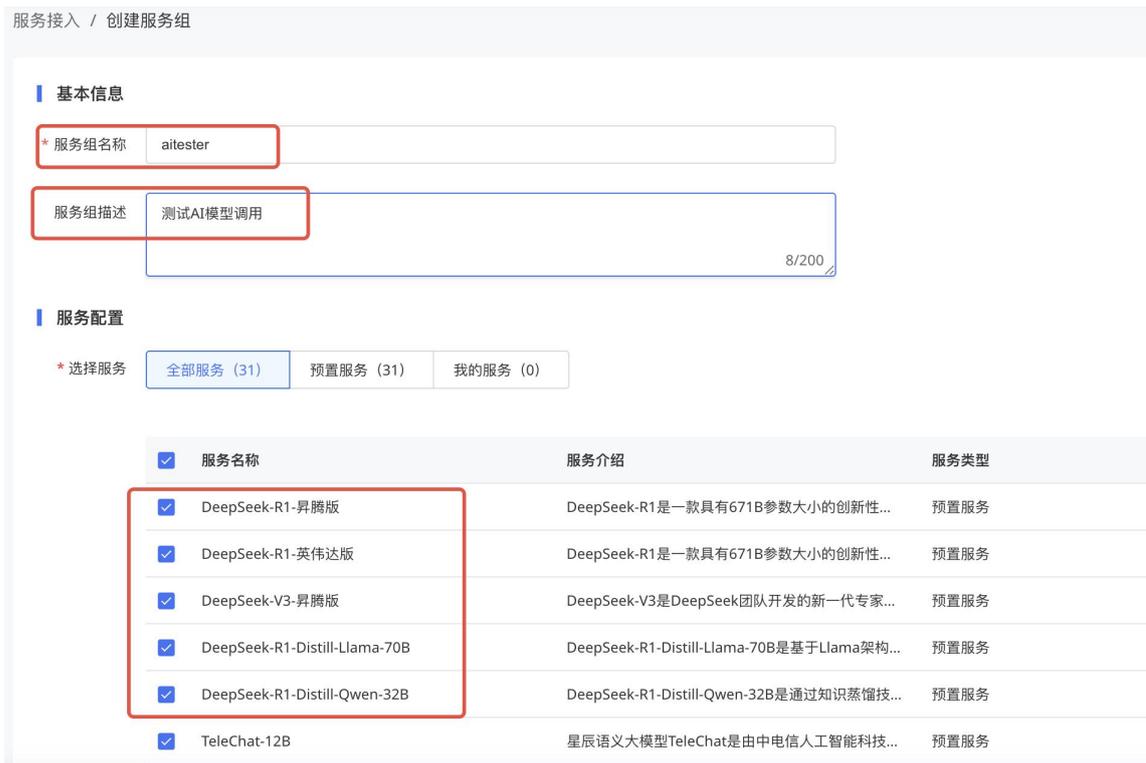
1. 访问天翼云息壤deepseek地址：<https://www.ctyun.cn/act/xirang/deepseek>，选择立即体验。



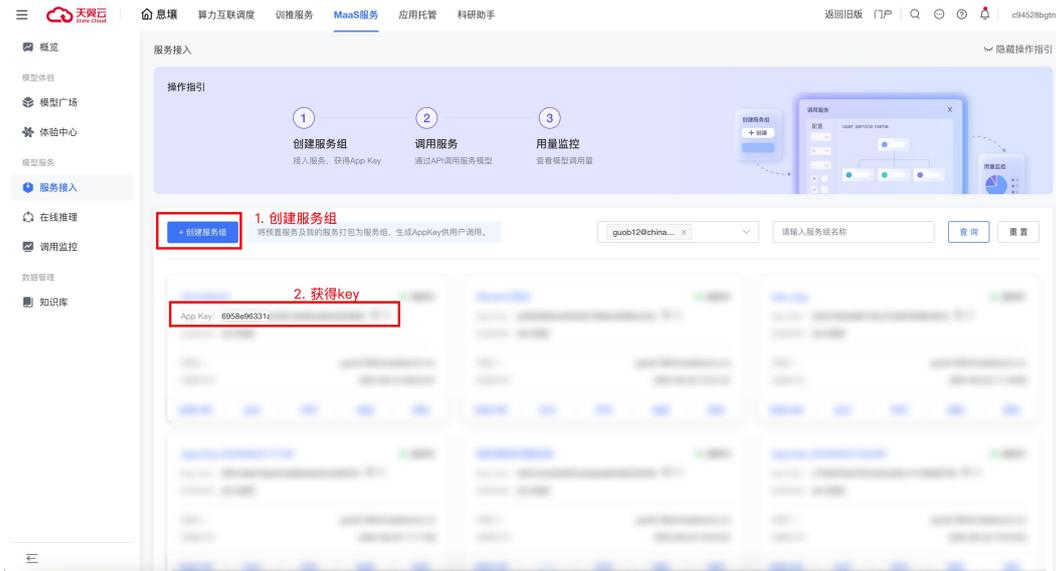
2. 进入“体验中心”后切换到服务接入。



3. 在“服务接入”下面，创建服务组。选择需要的模型勾选即可。

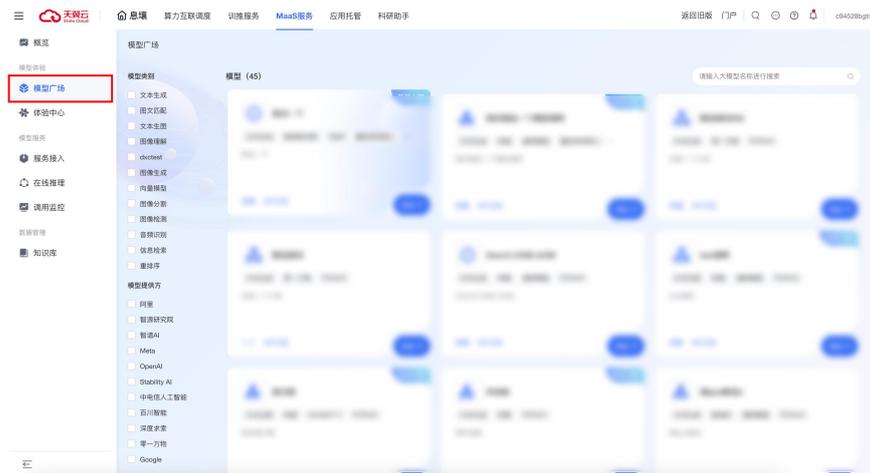


4. 创建服务组后，从服务组上面获取API KEY。



### 3. 获取调用的模型名称

1. 左侧导航栏切换到模型广场，选择对应想要调用的模型。



2. 获取模型ID或者模型名称作为模型名称。

如下图：模型ID：4bd107bff85941239e27b1509eccfe98，模型名称：DeepSeek-R1-昇腾版

< **DeepSeek-R1-昇腾版** 文本生成 DeepSeek 通用模型 PyTorch

模型ID: 4bd107bff85941239e27b1509eccfe98

模型名称: DeepSeek-R1-昇腾版

📘 调用本模型API时需要的model参数值，取模型id或模型名称之一即可

DeepSeek-R1是一款具有671B参数大小的创新性大语言模型，该模型基于transformer架构，通过对海量语料数据进行预训练等多种能力。R1模型在多个NLP基准测试中表现出色，具备较强的泛化能力和适应性。

模型推理计费: 输入计费 **4** 元/百万tokens 输出计费 **16** 元/百万tokens

[模型介绍](#) [API文档](#) [任务记录](#)

## 1.模型简介

DeepSeek-R1是一款具有671B参数大小的创新性大语言模型，该模型基于transformer架构进行对齐，具备语义分析、计算推理、问答对话、篇章生成、代码编写等多种能力。F

## 2.基础信息

- 上下文长度:

32K tokens (输出长度上限为16K)。

DeepSeek模型版本选择参考:

- DeepSeek-R1: 适合需要高级推理能力和高精度的复杂任务，如科研、金融建模等。响应时间相对长。
- DeepSeek-V3: 适合需要快速响应和高性价比的通用自然语言处理任务，如智能客服、内容生成等。响应时间相对短。

## 2.手动验证下key的可用性

下列请求为DeepSeek-R1-昇腾版请求样例，如果正常返回deepseek回答，代表没有问题。

注意：下面curl请求中替换成□自己的API KEY后类似“--header 'Authorization: Bearer abcdefghxxxxsss' \ ”

```
curl --location 'https://wishhub-
x6.ctyun.cn/v1/chat/completions' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer 替换成自己的APP KEY' \
--data '{
  "messages": [
    {
      "role": "system",
      "content": "You are a test assistant."
    },
    {
      "role": "user",
      "content": "Testing. Just say hi and nothing
else."
    }
  ],
  "model": "4bd107bff85941239e27b1509eccfe98"
}'
```

### 3 调用代码示例

Python

同步调用

```
import os

import openai

from openai import OpenAI

# 需要补充一下属性

baseUrl = "https://wishhub-x6.ctyun.cn/v1"

# 从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

appKey=os.getenv("XIRANG_app_key")

DeepSeek_R1_model_id =
"4bd107bfff85941239e27b1509eccfe98"

DeepSeek_V3_model_id =
"9dc913a037774fc0b248376905c85da5"

prompt="你好，3.01+103.1等于多少" #对话问题

def main():

    client = OpenAI(base_url=baseUrl, api_key=appKey)
    messages = [
        {"role": "user", "content": prompt}
    ]
    try:
        res = client.chat.completions.create(
            model=DeepSeek_R1_model_id,
            messages=messages,
            stream=False
        )
        print(res.choices[0].message.content or "",
end="", flush=True)

    except openai.APIStatusError as e:
```



流式调用

```
import os

import openai

from openai import OpenAI

# 需要补充一下属性

baseUrl = "https://wishhub-x6.ctyun.cn/v1"

# 从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

appKey=os.getenv("XIRANG_app_key")

DeepSeek_R1_model_id =
"4bd107bff85941239e27b1509eccfe98"

DeepSeek_V3_model_id =
"9dc913a037774fc0b248376905c85da5"

prompt= "你是谁"

def main():

    client = OpenAI(base_url=baseUrl, api_key=appKey)

    messages = [

        {"role": "user", "content": prompt}

    ]

    try:

        stream = client.chat.completions.create(

            model=DeepSeek_R1_model_id,

            messages=messages,

            stream=True

        )

        # 流式

        for chunk in stream:

            print(chunk.choices[0].delta.content or "",

end="", flush=True)
```

## Java

建议与Spring AI框架直接集成，而非使用openai-java版本库。

参考第四章「开源框架集成」Spring AI章节实现。

## Golang

注意：

1、golang版本要求1.18以上

2、go.mod引用相关类库 `go get github.com/openai/openai-go`

同步调用

```
package main

import (
    "context"
    "fmt"
    "github.com/openai/openai-go"
    "github.com/openai/openai-go/option"
    "os"
)

func main() {
    baseUrl := "https://wishhub-x6.ctyun.cn/v1"
//api前缀地址
    //从环境变量获取API密钥，如果没有设置，也可以直接终端执行export XIRANG_app_key="xxx"
    appkey := os.Getenv("XIRANG_app_key")
    modelId := "xxxxxx" //替换成自己要用的模型
    prompt := "你是谁" //对话问题

    client := openai.NewClient(
        option.WithAPIKey(appkey),
        option.WithBaseURL(baseUrl),
    )

    completion, err :=
client.Chat.Completions.New(context.Background(),
openai.ChatCompletionNewParams{
    Messages:
openai.F([]openai.ChatCompletionMessageParamUnion{
```



流式调用

```
package main

import (
    "context"
    "fmt"
    "github.com/openai/openai-go"
    "github.com/openai/openai-go/option"
    "os"
)

func main() {
    baseUrl := "https://wishub.ctyun.cn:1443/v1" //api
    前缀地址
    //从环境变量获取API密钥，如果没有设置，也可以直接终端执行
    export XIRANG_app_key="xxx"
    appkey := os.Getenv("XIRANG_app_key")
    modelId := "xxxxxx" //模型id
    prompt := "你好啊" //对话问题

    client := openai.NewClient(
        option.WithAPIKey(appkey),
        option.WithBaseURL(baseUrl),
    )

    stream :=
    client.Chat.Completions.NewStreaming(context.Background(),
    openai.ChatCompletionNewParams{
        Messages:
    openai.F([]openai.ChatCompletionMessageParamUnion{
        openai.ChatCompletionMessage{Content:
```

注意:

1、php版本要求8.3.13以上

2、安装composer组件

3、运行以下命令

```
composer require openai-php/client
```

```
composer dump-autoload
```

同步调用

```
<?php
require __DIR__ . '/vendor/autoload.php';

use OpenAI\Client;

$baseUrl="https://wishhub-x6.ctyun.cn/v1"; //api前缀地址
//从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

$appkey = getenv("XIRANG_app_key");
$modelId="xxxxx"; //替换成自己要用的模型
$prompt="你好，介绍一下自己"; //对话问题

$client = OpenAI::factory()
    ->withApiKey($appkey)
    ->withBaseUri($baseUrl)
    ->make();

$response = $client->chat()->create([
    'model' => $modelId,
    'messages' => [
        ['role' => 'user', 'content' => $prompt],
    ],
]);

foreach ($response->choices as $result) {
    echo $result->message->content;
}
```



流式调用

```
<?php
require __DIR__ . '/vendor/autoload.php';

use OpenAI\Client;

$baseUrl="https://wishhub-x6.ctyun.cn/v1"; //api前缀地址
//从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

$appkey = getenv("XIRANG_app_key");
$modelId="xxxxx"; //替换成自己要用的模型
$prompt="你好，介绍一下自己"; //对话问题

$client = OpenAI::factory()
    ->withApiKey($appKey)
    ->withBaseUri($baseUrl)
    ->make();

$stream = $client->chat()->createStreamed([
    'model' => $modelId,
    'messages' => [
        ['role' => 'user', 'content' => $prompt],
    ],
]);

foreach($stream as $response){
    $data=$response->choices[0];
    echo json_encode($data,
        JSON_UNESCAPED_UNICODE,JSON_PRETTY_PRINT).PHP_EOL;
}
```

## Nodejs

注意:

1、测试用的node版本 $\geq$ v17.9.1,npm版本 $\geq$ 8.11.0

2、运行前需执行一下命令

```
npm init -y
```

```
npm install openai
```

3、在生成的package.json文件中添加"type":"module"

4、替换baseUrl,appkey,modelId,prompt等字段的值

## 同步调用

```
import OpenAI from 'openai';

const baseUrl="https://wishhub-x6.ctyun.cn/v1" //api前
缀地址
//从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

const appkey= process.env.XIRANG_app_key

const modelId="xxxxx" //模型id

const prompt="你好啊,给一篇100字左右的作文" //对话问题

const client = new OpenAI({
  apiKey: appkey,
  baseUrl:baseUrl
});

async function chatCompletion() {
  const completion = await
client.chat.completions.create({
  model:modelId,
  messages: [{ role: 'user', content: prompt }],
});

console.log(completion.choices[0]?.message?.content);
}

chatCompletion();
```



流式调用

```
import OpenAI from 'openai';

const baseUrl="https://wishhub-x6.ctyun.cn/v1" //api前
缀地址
//从环境变量获取API密钥，如果没有设置，也可以直接终端执行export
XIRANG_app_key="xxx"

const appkey= process.env.XIRANG_app_key

const modelId="xxxxx" //模型id

const prompt="你好啊,给一篇100字左右的作文" //对话问题

const client = new OpenAI({
  apiKey: appkey,
  baseUrl:baseUrl
});

async function stream_chatCompletion() {
  const stream = await
client.chat.completions.create({
  model:modelId,
  messages: [{ role: 'user', content: prompt }],
  stream: true,
});
  for await (const chunk of stream) {

process.stdout.write(chunk.choices[0]?.delta?.content ||
'');
  }
}
```

## 4. 开源框架集成

### 说明

- 部分第三方工具调用可能存在接口不适配问题，如涉及工具调用时，建议优先使用下文提及的工具。
- 涉及第三方工具配置时，API key 等同于上文中创建的服务组 APP key。

### Langchain调用

参考langchain的自定义模型文档通过OpenAI协议调用即可。

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage

def main():
    # 初始化LangChain模型，配置自定义API端点
    llm = ChatOpenAI(
        model_name="4bd107bff85941239e27b1509eccfe98",
# 自定义模型名称
        # 从环境变量获取API密钥，如果没有设置，也可以直接终端执行
export XIRANG_app_key="xxx"
        openai_app_key=os.getenv("XIRANG_APP_KEY")
        openai_api_base="https://wishhub-
x6.ctyun.cn/v1", # API基础URL
        temperature=0.7
    )

    # 创建一个聊天消息
    message = HumanMessage(content="你是谁")

    # 使用LangChain进行对话
    response = llm([message])
    print(response.content)

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

## Dify

安装条件：本地下载并安装docker。（Docker 官网下载页面：访问 Docker 官方网站。在页面中点击“Download Docker Desktop”按钮，即可选择适合 Windows、macOS 或 Linux 的版本进行下载。）

拉取dify镜像到本地

```
# 如果本地已安装dify可以忽略  
# 假设当前最新版本为 0.15.3  
git clone https://github.com/langgenius/dify.git --  
branch 0.15.3
```

修改配置参数

```
cd docker  
mv .env.example .env
```

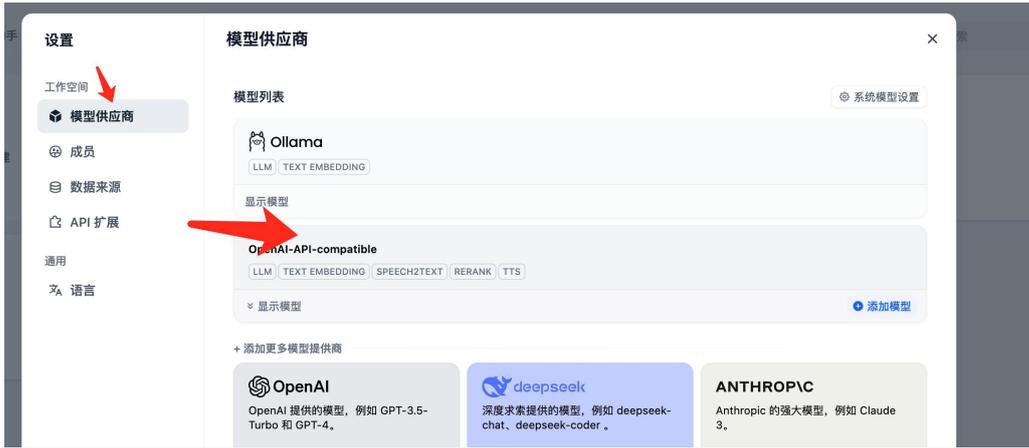
启动并验证

```
#执行：  
docker compose up -d  
# 等服务启动访问  
http://localhost/apps
```

配置天翼云息壤模型

配置方式一：直接配置方式（所有版本均可使用）

通过直接配置OpenAI-API-compatible支持，配置方式如下：



### 添加 OpenAI-API-compatible

OpenAI-API-compatible

模型类型 \*

LLM

Text Embedding

Speech2text

Rerank

TTS

模型名称 \*

9dc913a037774f...c85da5

对应 oai 模型 id

API Key

e31...bbd3ed55a0760139f

对应服务组 id

API endpoint URL \*

https://...cn/v1

oai调用地址 path 只到 /v1

Completion mode

对话

模型上下文长度 \*

4096

最大 token 上限 \*

4096

Function calling

取消

保存

配置方式二：使用镜像配置方式（仅支持0.15.3版本）

修改env配置

```
cd docker  
  
#在.env文件添加  
  
CUSTOM_MODEL_ENABLED=true  
  
OLLAMA_API_BASE_URL=host.docker.internal:11434
```

修改下api和work的镜像地址，改为增加了xirang模型供应商的dify 0.15.3版本。

修改后需要重启dify。

```
services:
  # API service with Xirang model provider
  api:
    image: telecom-eci-huadong1-crs-registry-
huadong1.crs.ctyun.cn/xirang/dify-with-xirang:0.15.3 # 原来
为langgenius/dify-api:0.15.3

  # worker service
  # The Celery worker for processing the queue.
  worker:
    image: telecom-eci-huadong1-crs-registry-
huadong1.crs.ctyun.cn/xirang/dify-with-xirang:0.15.3 # 原来
为langgenius/dify-api:0.15.3
```

在个人设置里面点击模型供应商，选择天翼云模型。



设置

工作空间

模型供应商

成员

数据来源

API 扩展

通用

语言

模型供应商

所有模型均支持流式输出

<p><b>OpenAI</b> OpenAI 提供的模型，例如 GPT-3.5-Turbo 和 GPT-4。</p> <p>LLM   TEXT EMBEDDING SPEECH2TEXT   MODERATION   TTS</p>	<p><b>deepseek</b> 深度求索提供的模型，例如 deepseek-chat、deepseek-coder。</p> <p>LLM</p>	<p><b>ANTHROPIC</b> Anthropic 的强大模型，例如 Claude 3。</p> <p>LLM</p>
<p><b>Azure OpenAI Service</b></p> <p>LLM   TEXT EMBEDDING SPEECH2TEXT   TTS</p>	<p><b>Gemini</b> 谷歌提供的 Gemini 模型。</p> <p>LLM</p>	<p><b>Google Cloud</b> Vertex AI in Google Cloud Platform.</p> <p>LLM   TEXT EMBEDDING</p>
<p><b>NVIDIA</b> API Catalog</p> <p>设置</p>	<p><b>NVIDIA</b> NVIDIA NIM，一组易于使用的模型推理微服务。</p> <p>LLM</p>	<p><b>cohere</b></p> <p>LLM   TEXT EMBEDDING   RERANK</p>
<p><b>upstage</b> Upstage 提供的模型，例如 Solar-1-mini-chat。</p>	<p><b>aws Bedrock</b> AWS Bedrock's models.</p>	<p><b>together.ai</b></p>

### 模型列表

系统模型设置

  
LLM  
显示模型

API-KEY ●

设置

+ 添加更多模型提供商

## 设置 天翼云息壤



API Key

ebf0b0\*\*\*\*\*9a

API endpoint URL \*

https://wishub-x1.ctyun.cn/v1

移除

取消

保存

您的密钥将使用 PKCS1\_OAEP 技术进行加密和存储。

点击保存后，可以看到模型的配置。

### 设置

工作空间

模型供应商

成员

数据来源

API 扩展

通用

语言

### 模型供应商

#### 模型列表

系统模型设置

Moonsht AI

LLM

API-KEY ●

设置

显示模型

天翼云 logo

LLM

API-KEY ●

设置

2 个模型

添加模型

DeepSeek-V3模型 LLM CHAT 4K

DeepSeek-R1模型 LLM CHAT 8K

+ 添加更多模型提供商

OpenAI

OpenAI 提供的模型，例如 GPT-3.5-Turbo 和 GPT-4。

LLM TEXT EMBEDDING

SPEECH2TEXT MODERATION TTS

deepseek

深度搜索提供的模型，例如 deepseek-chat、deepseek-coder。

LLM

ANTHROPIC

Anthropic 的强大模型，例如 Claude 3。

LLM

其他非预置模型：

模型列表

系统模型设置

中国电信 | 天翼云

LLM

API-KEY 设置

显示模型 添加模型

### 添加 天翼云息壤



模型类型 \*

LLM

模型名称 \*

9dc913a037774fc0b248376905c85da5

模型上下文长度 \*

4096

API Key

[Redacted API Key]

API endpoint URL \*

https://wishub-x1.ctyun.cn/v1

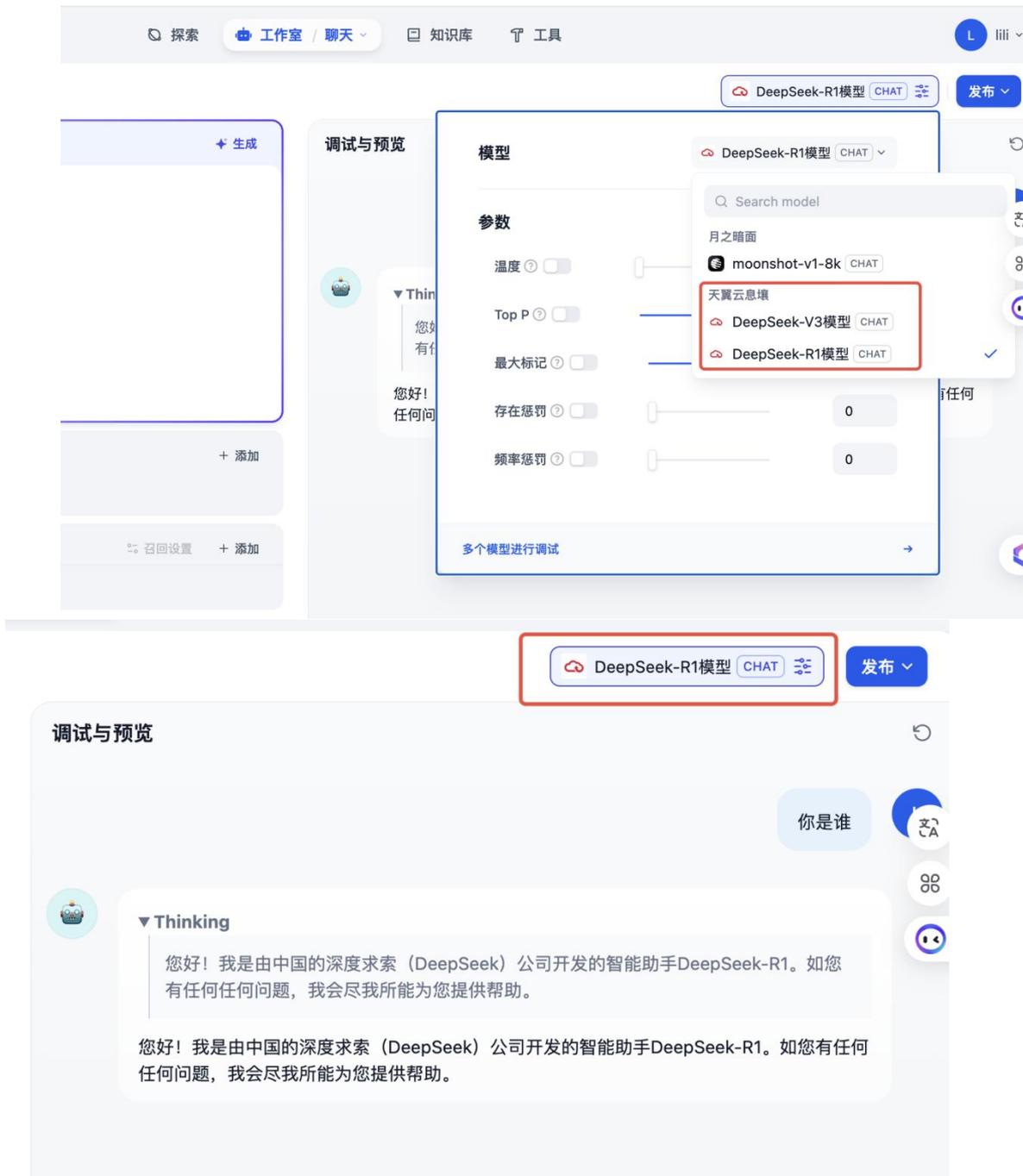
补全模式

对话

取消

保存

配置对话助手



### 常见问题 (0.15.3版本)

如遇到无法展示问题, 可能是因为您的dify版本为最新版, 我们目前还不支持, 可执行以下操作切换到0.15.3版本并重启服务。

dify插件需要切换版本重新启动服务才可以达成兼容

```
git clone https://gitcode.com/gh_mirrors/di/dify.git
```

```
cd dify
```

```
git checkout 7796984444191c639bd3c541a44e832b17ad1cae
```

如遇到息壤镜像失败问题，可切换到dify原始镜像，通过配置直接配置OpenAI-API-compatible支持，配置方式如下：



### 添加 OpenAI-API-compatible

OpenAI-API-compatible

模型类型 \*

LLM  
 Text Embedding  
 Speech2text  
 Rerank  
 TTS

模型名称 \*

9dc913a037774f...85da5

对应 oai 模型 id

API Key

e31...bbd3ed55a0760139f

对应服务组 id

API endpoint URL \*

https://...cn/v1

oai调用地址 path 只到 /v1

Completion mode

对话

模型上下文长度 \*

4096

最大 token 上限 \*

4096

Function calling

取消

保存



**Spring AI**

Spring AI要求使用JDK 17或更高版本；Spring AI需要在Spring Boot 3.2.x或更高版本的环境中运行。

核心代码如下：

OpenAIConfig配置类：

```
package com.demo.aitest;

import jakarta.annotation.PostConstruct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import org.springframework.ai.openai.OpenAiChatClient;
import org.springframework.ai.openai.OpenAiChatOptions;
import org.springframework.ai.openai.api.OpenAiApi;

@Configuration
public class OpenAIConfig {

    @Value("${spring.ai.openai.api-key}")
    private String apiKey;

    @Value("${spring.ai.openai.base-url}")
    private String baseUrl;

    @Value("${spring.ai.openai.chat.options.model}")
    private String model;

    @Bean
    public OpenAiChatClient openAiChatClient() {
        OpenAiApi openAiApi = new OpenAiApi(baseUrl,
apiKey);
```



ChatController 类:

```
package com.lili.aitest;

import org.springframework.ai.openai.OpenAiChatClient;
import org.springframework.ai.chat.messages.UserMessage;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.ai.chat.ChatResponse;
import org.springframework.ai.chat.prompt.Prompt;
import org.springframework.web.bind.annotation.*;
import
org.springframework.web.servlet.mvc.method.annotation.SseEmitt
er;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;

@RestController
public class ChatController {
    private static final Logger logger =
LoggerFactory.getLogger(ChatController.class);

    @Autowired
    private OpenAiChatClient chatClient;

    @GetMapping("/chat")
    public String chat(@RequestParam String message) {
        try {
            ChatResponse response = chatClient.call(new
```

application.yml配置如下：

```
spring:
  ai:
    openai:
      api-key: Your_APP_KEY    #替换成,自己的API KEY
      base-url: https://wishhub-x6.ctyun.cn
      chat:
        options:
          model: 4bd107bff85941239e27b1509eccfe98. #替换成
,自己要用的模型
      server:
        servlet:
          encoding:
            charset: UTF-8
            force: true
            enabled: true
```

pom.xml依赖如下

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.demo</groupId>
  <artifactId>aitest</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>aitest</name>
  <description>aitest</description>

  <properties>
    <java.version>17</java.version>
    <spring-ai.version>0.8.1</spring-ai.version>
    <spring-cloud.version>2023.0.0</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.ai</groupId>
```



访问地址:

- 同步: <http://localhost:8080/chat?message=Hello>
- 流式访问: <http://localhost:8080/chat/stream?message=Hello>

Ragflow

注意：目前息壤model不支持embedding model能力，因此只能作为chat model提供能力。

1. 下载镜像git clone <https://github.com/infiniflow/ragflow.git>
2. 通过docker安装并启动：

```
cd ragflow/docker  
docker compose -f docker-compose-CN.yml up -d
```

3. 检查 RAGFlow 服务是否正常启动：

```
docker logs -f ragflow-server
```

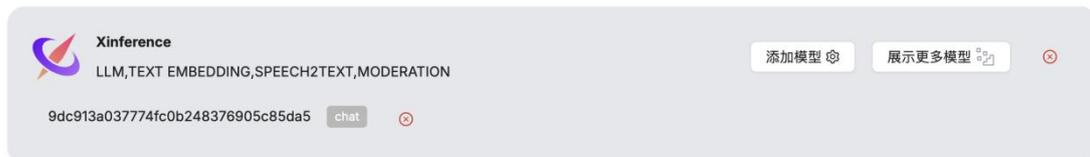
如果看到类似以下输出，则表示启动成功：

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:9380
* Running on http://x.x.x.x:9380
```

#### 4. 打开浏览器配置模型Xinference服务提供商

在浏览器中输入服务器的 IP 地址访问 RAGFlow。默认情况下，RAGFlow 使用 HTTP 端口 80，因此无需输入端口号。

选择“Xinference”模型服务商( Xinference是工厂配置，支持自定义模型名称)



配置chatmodel的界面如下：



聊天配置  
在这里，为你的专业知识库装扮专属助手! ❤️

助理设置      提示引擎      模型设置

\* 模型 ?

自由 ?

- moonshot-v1-32k
- moonshot-v1-8k
- moonshot-v1-auto
- OpenAI-API-Compatible
- 9dc913a037774fc0b248376905c85da5\_\_OpenAI-API
- Xinference
- 9dc913a037774fc0b248376905c85da5**

温度 ?

Top P ?

存在处罚 ?

频率惩罚 ?

最大token数 ?

取消      确定

回到对话界面，配置聊天助手：

RAGFlow      知识库      聊天      搜索      Agent      文件管理

新建助理      聊天 0

- jarvis3-openai  
A helpful dialog
- jarvi2  
A helpful dialog
- jarvis  
A helpful dialog

聊天配置  
在这里，为你的专业知识库装扮专属助手! ❤️

助理设置      提示引擎      模型设置

\* 助理姓名

助理描述

助理头像

空回复 ?

设置开场白 ?

显示引文 ?

开启对话：



a

帮我生成一篇描写春天的文章



知识库中未找到您要的答案！不过，我可以为您创作一篇描写春天的文章。以下是文章内容：

#### 春天的赞歌

春天，是一年中富有生机与希望的季节。当冬天的严寒逐渐退去，大地开始苏醒，万物复苏的景象让人心旷神怡。

清晨，阳光透过薄雾洒在大地上，温暖而柔和。树枝上冒出了嫩绿的新芽，仿佛在向人们宣告春天的到来。草地上的露珠晶莹剔透，像一颗颗珍珠般闪烁着光芒。

花园里，百花争艳。桃花、杏花、樱花竞相开放，红的、粉的、白的交织在一起，形成了一幅美丽的画卷。蜜蜂和蝴蝶在花丛中忙碌地穿梭着，为这片美景增添了无限活力。

河边的柳树也换上了新装，柔软的枝条随风摇曳，宛如少女的长发飘逸动人。河水清澈见底，鱼儿在水中自由自在地游弋，偶尔跃出水面，激起一圈圈涟漪。

田野间，农民们开始忙碌起来，播种希望，期待着丰收的到来。孩子们在田间奔跑嬉戏，欢声笑语回荡在空中，构成了一幅和谐美好的田园风光图。

春天不仅带来了自然界的美丽变化，更给人们的心灵带来了温暖与希望。它让我们感受到生命的力量和大自然的魔力，激励着我们以更加积极向上的态度面对生活，迎接未来的挑战。

在这个充满生机的季节里，让我们放慢脚步，去感受大自然的美好，去珍惜身边的每一个人、每一件事，让心灵得到净化与升华，共同谱写一曲属于春天的赞歌吧！

希望这篇文章能够满足您的需求！如果您有其他要求或需要进一步修改，请随时告诉我哦 😊🌸

## 5. 常见报错

### 触发限流

报错信息为:

```
[openai_api_compatible]Error:APIrequestfailedwithstatusCode429:{"code":700007,"detail":"Deepseek模型并发已达上限","message":"DEEP_SEEK_MODEL_CONCURRENCY_LIMIT","error":{"code":"700007","message":"Deepseek模型并发已达上限","type":"DEEP_SEEK_MODEL_CONCURRENCY_LIMIT"}}
```

- 问题原因: 大模型并发是大模型的并发能力, 和用户、tokens是否用完都无关, 只和机器负载能力有关。目前单个模型有固定的并发度, 这是所有ds模型共享的并发度, 只能支持固定个请求同时访问。

- 解决方案: 等一段时间再调用, 或增加重试机制。

同步调用响应耗时长

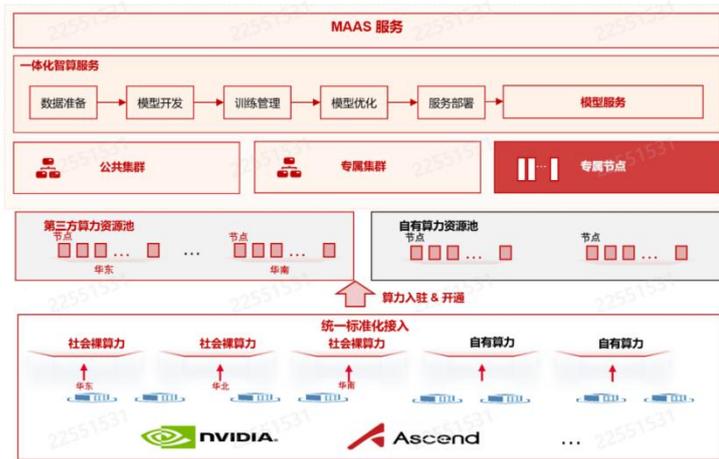
- 问题原因: 大模型API同步耗时长是因为其推理过程复杂, 需要逐个生成token并拼接结果, 且在处理长文本或复杂问题时, 等待完整结果返回的时间过长。而流式API可以在推理过程中逐步返回结果, 用户无需等待全部结果生成, 从而提升交互效率。

- 解决方案: 改为流式调用。

## 5.3 使用专属节点部署DeepSeek服务

### 服务说明

使用专属节点部署DeepSeek服务是基于息壤调度平台, 支持 H20 第三方算力接入, 通过智算服务平台使用多样化算力满足 DeepSeek 等推理服务部署及调用。



- 算力接入**
  - 统一算力调度平台接入多方算力，支持第三方算力统一接入，实现跨服务商、跨地域、跨架构统一算力接入。
- 算力调度**
  - 统一调度不同算力型号（英伟达 H20、H800 等；昇腾 910b 等），高效利用资源。
- 资源池化**
  - 支持第三方算力纳入边缘算力资源池。支持 DeepSeek 推理服务使用第三方算力（英伟达 H20、4090，昇腾 910 B 等）。
- 体验一致**
  - 统一第三方算力资源池、自有资源池的监控、监控能力统一展示和查看，管理，统一运维体验。
  - 统一运营，基于不同资源池使用智算服务，用户体验计费，购买入口和体验一致，统一购买体验。

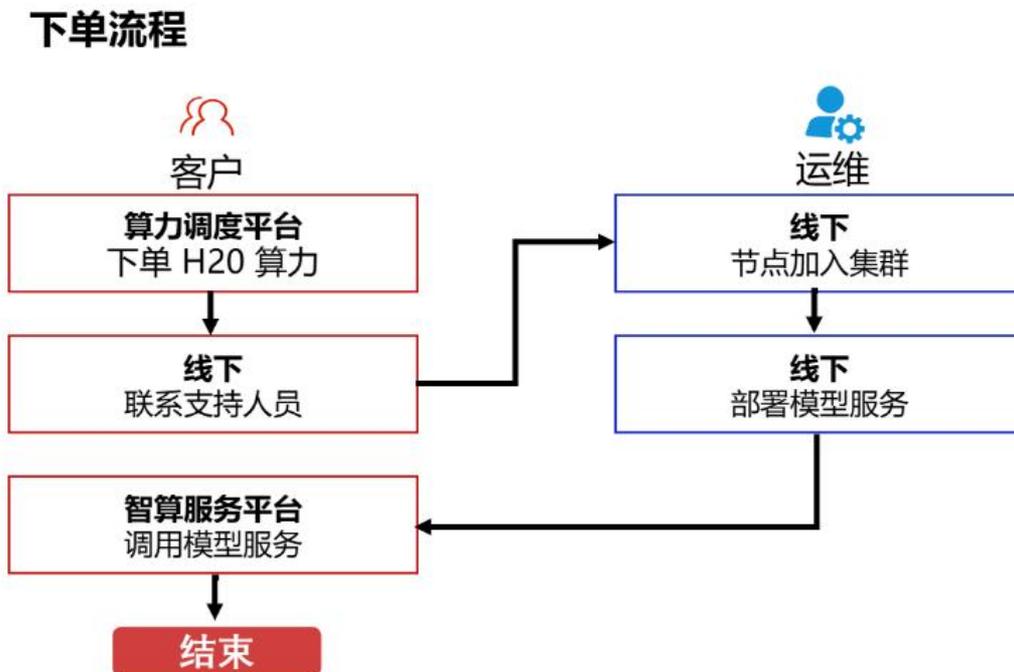
专

属节点是指用户购买的H20节点仅供用户自己使用，确保服务的独立性与安全性。

专属节点部署 DeepSeek 服务是指用户在完成专属H20节点购买后，主动联系天翼云工作人员告知其DeepSeek模型部署需求，天翼云工作人员将依据用户要求，帮助用户完成 DeepSeek 模型的部署工作，部署完成后，用户可直接使用DeepSeek模型服务。

。

服务整体流程如下图：



## 使用步骤

若需使用专属节点部署DeepSeek服务，请参照如下步骤进行：

登录天翼云官网，通过“首页”->“产品”->“人工智能”->“天翼云息壤一体化智算服务平台”路径，进入“模型推理服务”产品详情页。

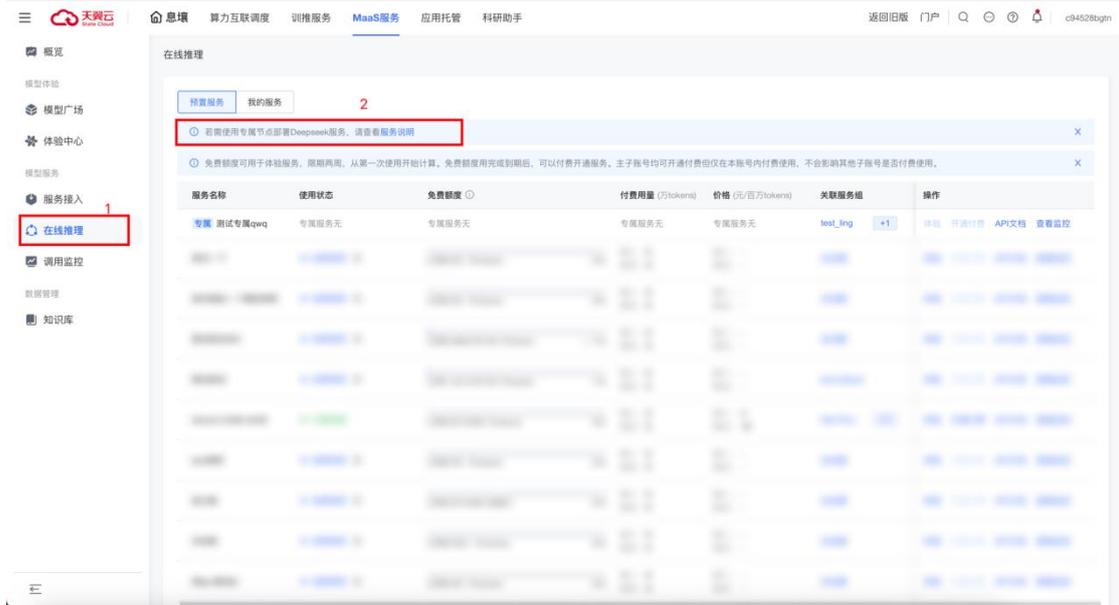


点击“立即体验”即可进入模型推理服务平台控制台。

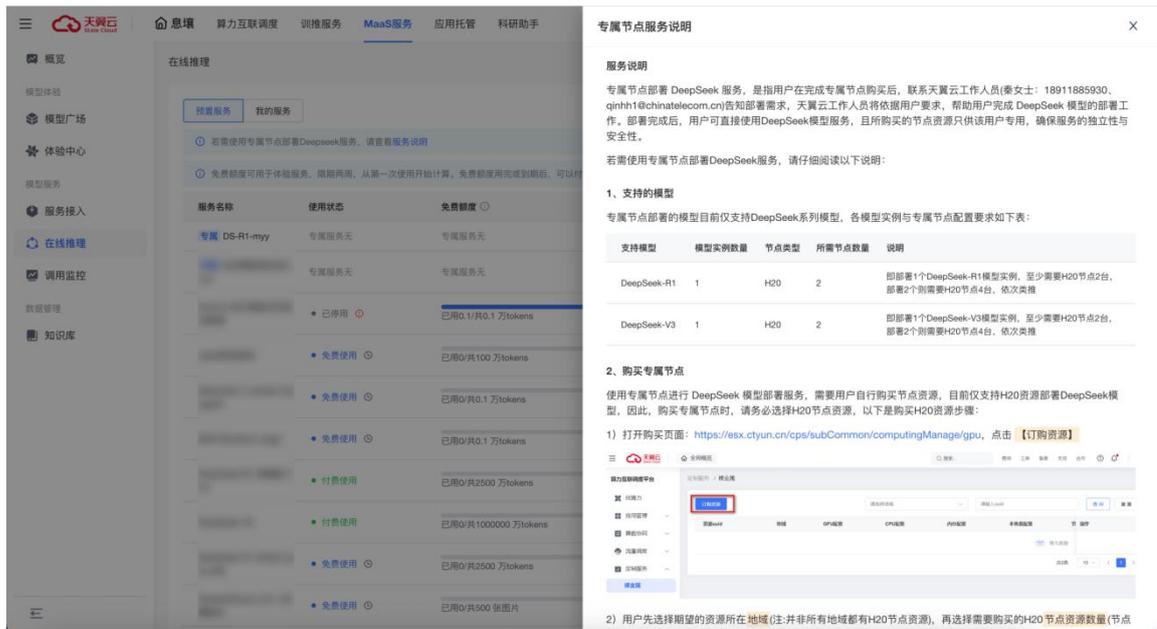


### 查看服务说明

点击“在线服务”菜单，进入“在线服务”页面，查看专属节点部署 DeepSeek 服务说明。



服务说明详细介绍了如何购买专属节点、支持的模型、如何部署模型以及后续操作，请仔细阅读内容。

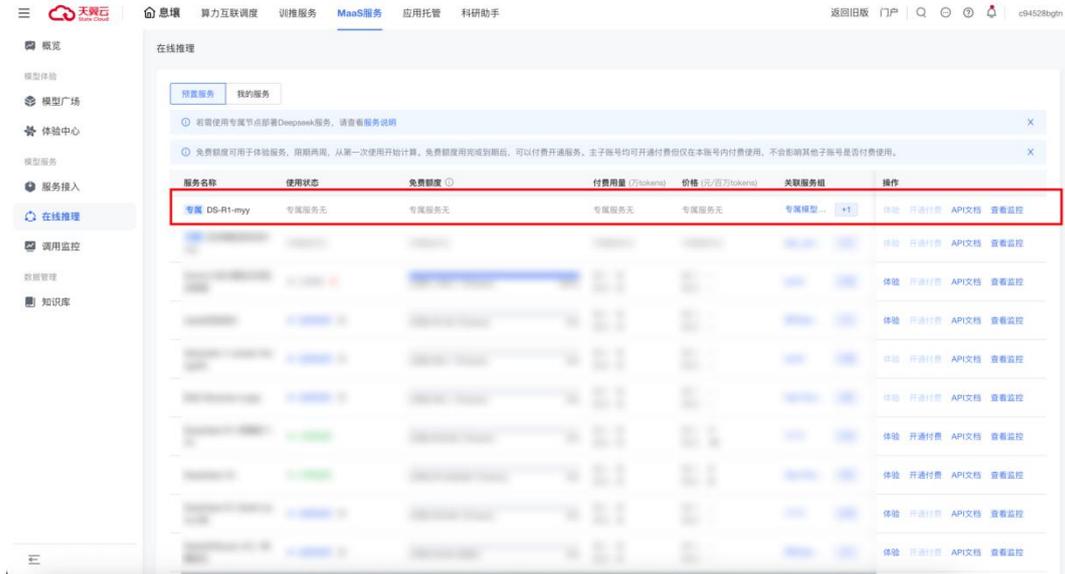


## 购买资源

按照服务说明列出的购买步骤执行购买流程，完成专属H20节点购买后，需主动联系天翼云工作人员，告知其DeepSeek模型部署需求，天翼云工作人员将根据您的需求，帮助您完成 DeepSeek 模型的部署工作，部署完成后，即可使用DeepSeek模型服务。

## 查看已部署服务

天翼云工作人员部署好模型服务后，您可以进入“在线服务”页面查看并使用已部署的模型服务。



## 6. 推理服务 API

### 6.1. 如何调用API

#### 6.1.1 终端节点

- 主要作用：用户信息的发送和接收，信令信息的控制处理、安全保护等作用。
- 终端节点：<https://wishhub-x6.ctyun.cn>

#### 6.1.2 接口构造

##### 请求域名

- 终端请求地址：<https://wishhub-x6.ctyun.cn>（具体模型终端请求地址详见单个模型的 API 文档）。

##### 通信协议

- 接口通过 HTTPS 进行通信，保护用户数据的机密性和完整性，确保网络通信的安全性。

## 版本管理

- 为区分接口版本，在 http 请求路径中加入版本信息，目前版本为 v1，因而请求的 url 前缀为：  
`https://wishhub-x6.ctyun.cn/v1`

## 请求鉴权

- 请求 header 中需要填入 Authorization 鉴权信息，Authorization 对应值应为 Bearer + 平台上获取的服务组 App Key
- AppKey 获取方法：登录平台门户，模型推理服务->模型服务->服务接入->创建服务组->绑定服务->获取 appKey
- 为避免被安全护栏拦截，建议在 http 请求 header 中填入 User-Agent 信息
- 浏览器或客户端标识：Chrome/58.0.3029.110、Mozilla/5.0、AppleWebKit/537.36、Safari/537.36、Windows NT 10.0、PostmanRuntime-ApipostRuntime/1.1.0 等

## 请求示例

请求路径: `https://wishhub-x6.ctyun.cn/v1/xxx/yyy`, 其中  
`/xxx/yyy` 为具体的功能路径, 如 `/chat/completions`

请求方式: POST

请求header必填项:

Authorization: Bearer AppKey

Content-Type: application/json

其他header:

User-Agent: PostmanRuntime-Apipostruntime/1.1.0

## 6.2. 接口类型列表

功能分类	支持模型类别	请求路径后缀	请求完整路径	功能描述
chat	文本生成、图像理解	<code>/chat/completions</code>	<code>https://wishhub-x6.ctyun.cn/v1/chat/completions</code>	针对描述会话的消息列表, 模型将返回响应。
image	文本生图	<code>/images/generations</code>	<code>https://wishhub-x6.ctyun.cn/v1/images/generations</code>	基于提示创建图像。
Embeddings	文本向量化	<code>/embeddings</code>	<code>https://wishhub-x6.ctyun.cn/v1/embeddings</code>	创建表示输入文本的嵌入向量
Reranker	重排序	<code>/rerank</code>	<code>https://wishhub-x6.ctyun.cn/v1/rerank</code>	通常用于计算查询和文档之间的相关性分数。

## 6.3. API列表

模型	模型简介	模型 ID
DeepSeek-R1-昇腾版	DeepSeek-R1 是一款具有创新性的大语言模型，由杭州深度求索人工智能基础技术研究有限公司开发。该模型基于 transformer 架构，通过对海量语料数据进行预训练，结合注意力机制，能够理解和生成自然语言。它经过监督微调、人类反馈的强化学习等技术进行对齐，具备语义分析、计算推理、问答对话、篇章生成、代码编写等多种能力。R1 模型在多个 NLP 基准测试中表现出色，具备较强的泛化能力和适应性。	4bd107bff85941239e27b1509eccfe98
DeepSeek-R1-昇腾版 2	DeepSeek-R1 是一款具有创新性的大语言模型，该模型基于 transformer 架构，通过对海量语料数据进行预训练，结合注意力机制，经过监督微调、人类反馈的强化学习等技术进行对齐，具备语义分析、计算推理、问答对话、篇章生成、代码编写等多种能力。R1 模型在多个 NLP 基准测试中表现出色，具备较强的泛化能力和适应性。	7ba7726dad4c4ea4ab7f39c7741aea68
DeepSeek-V3-昇腾版	DeepSeek-V3 是 DeepSeek 团队开发的新一代专家混合 (MoE) 语言模型，共有 671B 参数，在 14.8 万亿个 Tokens 上进行预训练。该模型采用多头潜在注意力 (MLA) 和 DeepSeekMoE 架构，继承了 DeepSeek-V2 模型的优势，并在性能、效率和功能上进行了显著提升。	9dc913a037774fc0b248376905c85da5
DeepSeek-R1-Distill-Llama-70B	DeepSeek-R1-Distill-Llama-70B 是基于 Llama 架构并经过强化学习和蒸馏优化开发的高性能语言模型。该模型融合了 DeepSeek-R1 的先进知识蒸馏技术与 Llama-70B 模型的架构优势。通过知识蒸馏，在保持较小参数规模的同时，具备强大的语言理解和生成能力。	515fdb33cc84aa799bbd44b6e00660d
DeepSeek-R1-Distill-Qwen-32B	DeepSeek-R1-Distill-Qwen-32B 是通过知识蒸馏技术从 DeepSeek-R1 模型中提炼出来的小型语言模型。它继承了 DeepSeek-R1 的推理能力，专注于数学和逻辑推理任务，但体积更小，适合资源受限的环境。	b383c1eecf2c4b30b4bcc7f019cf90d
Baichuan2-Turbo	Baichuan-Turbo 系列模型是百川智能推出的大语言模型，采用搜索增强技术实现大模型与领域知识、全网知识的全面链接。	43ac83747cb34730a00b7cfe590c89ac

Qwen2-72B-Instruct	Qwen2 是 Qwen 大型语言模型的新系列。Qwen2 发布了 5 个尺寸的预训练和指令微调模型，包括 Qwen2-0.5B、Qwen2-1.5B、Qwen2-7B、Qwen2-57B-A14B 以及 Qwen2-72B。这是指令调整的 72B Qwen2 模型，使用了大量数据对模型进行了预训练，并使用监督微调和直接偏好优化对模型进行了后训练。	2f05789705a64606a552fc2b30326bba
ChatGLM3-6B	ChatGLM3-6B 是 ChatGLM 系列最新一代的开源模型，在保留了前两代模型对话流畅、部署门槛低等众多优秀特性的基础上，ChatGLM3-6B 引入了更强大的基础模型、更完整的功能支持、更全面的开源序列几大特性。	7450fa195778420393542c7fa13c6640
TeleChat-12B	星辰语义大模型 TeleChat 是由中电信人工智能科技有限公司研发训练的大语言模型，TeleChat-12B 模型基座采用 3 万亿 Tokens 中英文高质量语料进行训练。TeleChat-12B-bot 在模型结构、训练数据、训练方法等方面进行了改进，在通用问答和知识类、代码类、数学类榜单上相比 TeleChat-7B-bot 均有大幅提升。	fdc31b36028043c48b15131885b148ce
Llama3-8B-Instruct	Meta 开发并发布了 Meta Llama 3 系列大型语言模型 (LLM)，包含 8B 和 70B 两种参数大小，Llama3-8B-Instruct 是经过指令微调的版本，针对对话用例进行了优化，在常见的行业基准测试中优于许多可用的开源聊天模型。	bda59c34e4424598bbd5930eba713fbf
Llama3-70B-Instruct	Meta 开发并发布了 Meta Llama 3 系列大型语言模型 (LLM)，包含 8B 和 70B 两种参数大小，Llama3-70B-Instruct 是经过指令微调的版本，针对对话用例进行了优化，在常见的行业基准测试中优于许多可用的开源聊天模型。	6192ed0cb6334302a2c32735dbbb6ce3
Qwen-VL-Chat	Qwen-VL-Chat 模型是在阿里云研发的大规模视觉语言模型 Qwen-VL 系列的基础上，使用对齐机制打造的视觉 AI 助手，该模型有更优秀的中文指令跟随，支持更灵活的交互方式，包括多图、多轮问答、创作等能力。	e8c39004ff804ca699d47b9254039db8
StableDiffusion-V2.1	StableDiffusion-V2.1 是由 Stability AI 公司推出的基于深度学习的文生图模型，它能够根据文本描述生成详细的图像，同时也可以应用于其他任务，例如图生图，生成筒短视频等。	40f9ae16e840417289ad2951f5b2c88f
Deepseek-V2-Lite-Chat	Deepseek-V2-Lite-Chat 是一款强大的开源专家混合 (MoE) 语言聊天模型，具有 16B 参数，2.4B 活动参数，使用 5.7T	0855b510473e4ec3a029569853f64974

	令牌从头开始训练，其特点是同时具备经济的训练和高效的推理。	
Qwen2.5-72B-Instruct	Qwen2.5 系列发布了许多基本语言模型和指令调整语言模型，参数范围从 0.5 到 720 亿个参数不等。Qwen2.5-72B-Instruct 模型是 Qwen2.5 系列大型语言模型指令调整版本。	d9df728b30a346afb74d2099b6c209aa
Gemma2-9B-IT	Gemma2-9B-IT 是 Google 最新发布的具有 90 亿参数的开源大型语言模型的指令调优版本。模型在大量文本数据上进行预训练，并且在性能上相较于前一代有了显著提升。该版本的性能在同类产品中也处于领先地位，超过了 Llama3-8B 和其他同规模的开源模型。	4dae2b9727db46b7b86e84e8ae6530a9
Llama3.2-3B-Instruct	Meta Llama3.2 多语言大型语言模型 (LLMs) 系列是一系列预训练及指令微调的生成模型，包含 1B 和 3B 参数规模。Llama3.2 指令微调的纯文本模型专门针对多语言对话应用场景进行了优化，包括代理检索和摘要任务。它们在通用行业基准测试中超越了许多可用的开源和闭源聊天模型。这是 Llama3.2-3B-Instruct 版本。	f7d0baa95fd2480280214bfe505b0e2e
ChatGLM3-6B-32K	ChatGLM3-6B-32K 模型在 ChatGLM3-6B 的基础上进一步强化了对于长文本的理解能力，能够更好的处理最多 32K 长度的上下文。具体对位置编码进行了更新，并设计了更有针对性的长文本训练方法，在对话阶段使用 32K 的上下文长度训练。	98b6d84f6b15421886d64350f2832782
CodeGemma-7B-IT	CodeGemma 是构建在 Gemma 之上的轻量级开放代码模型的集合。CodeGemma-7B-IT 模型是 CodeGemma 系列模型之一，是一种文本到文本和文本到代码的解码器模型的指令调整变体，具有 70 亿参数，可用于代码聊天和指令跟随。	fa8b78d2db034b6798c894e30fba1173
Qwen2.5-Math-7B-Instruct	Qwen2.5-Math 系列是数学专项大语言模型 Qwen2-Math 的升级版。系列包括 1.5B、7B、72B 三种参数的基础模型和指令微调模型以及数学奖励模型 Qwen2.5-Math-RM-72B，Qwen2.5-Math-7B-Instruct 的性能与 Qwen2-Math-72B-Instruct 相当。	ea056b1eedfc479198b49e2ef156e2aa
<u>DeepSeek-Coder-V2-Lite-Instruct</u>	DeepSeek-Coder-V2-Lite-Instruct 是一款强大的开源专家混合 (MoE) 语言聊天模型，具有 16B 参数，2.4B 活动参数。该模型基于 DeepSeek-V2 进一步预训	f23651e4a8904ea589a6372e0e860b10

	<p>练，增加了 6T Tokens，可在特定的代码任务中实现与 GPT4-Turbo 相当的性能。</p>	
BGE-m3	<p>BGE-m3 是智源发布的通用语义向量模型 BGE 家族新成员，支持超过 100 种语言，具备领先的多语言、跨语言检索能力，全面且高质量地支撑“句子”、“段落”、“篇章”、“文档”等不同粒度的输入文本，最大输入长度为 8192，并且一站式集成了稠密检索、稀疏检索、多向量检索三种检索功能，在多个评测基准中达到最优水平。</p>	46c1326f63044fbe80443af579466fe3
DeepSeek-V3-0324	<p>DeepSeek-V3-0324 是 DeepSeek 团队于 2025 年 3 月 24 日发布的 DeepSeek-V3 语言模型的新版本。是一个专家混合 (MoE) 语言模型，总参数为 6710 亿个，每个 Token 激活了 370 亿个参数。0324 版本开创了一种用于负载均衡的辅助无损策略，并设定了多令牌预测训练目标以提高性能。该模型版本在几个关键方面比其前身 DeepSeek-V3 有了显著改进。</p>	11bd888a35434486bf209066c7dad0ee
Qwen2-7B-Instruct	<p>Qwen2-7B-Instruct 是 Qwen2 大型语言模型系列中覆盖 70 亿参数的指令调优语言模型，支持高达 131,072 个令牌的上下文长度，能够处理大量输入。</p>	0e97efbf3aa042ebbaf0b2d358403b94
Qwen3-235B-A22B	<p>Qwen3-235B-A22B 是 Qwen3 系列大型语言模型的旗舰模型。拥有 2350 多亿总参数和 220 多亿激活参数。在代码、数学、通用能力等基准测试中，与 DeepSeek-R1、o1、o3-mini、Grok-3 和 Gemini-2.5-Pro 等顶级模型相比，表现出极具竞争力的结果。</p>	35af69e0d4af492ca366cf2df03c3172
Qwen3-32B	<p>Qwen3 是 Qwen 系列中最新一代的大型语言模型，提供一整套密集 (Dense) 模型和混合专家 (MoE) 模型。Qwen3 基于广泛的培训而构建，在推理、指令遵循、代理功能和多语言支持方面取得了突破性的进步。Qwen3-32B 是参数量为 32.8B 的密集 (Dense) 模型。</p>	3836b8d2ec5d46fc94cc7891064940aa
QwQ-32B	<p>QwQ-32B 是一款拥有 320 亿参数的推理模型，其性能可与具备 6710 亿参数 (其中 370 亿被激活) 的 DeepSeek-R1 媲美。该模型集成了与 Agent 相关的能力，使其能够在使用工具的同时进行批判性思考，并根据环境反馈调整推理过程。</p>	b9293363bfbf4db2bccb839ff4300d17
Qwen2.5-VL-72B-Instruct	<p>Qwen2.5-VL-72B-Instruct 模型是阿里云通义千问开源的全新视觉模型，具有</p>	88003ac1ca7a4e4e8efa7caee648323b

	720 亿参数规模，以满足高性能计算场景的需求。目前共推出 3B、7B、32B 和 72B 四个尺寸的版本。这是旗舰版 Qwen2.5-VL-72B 的指令微调模型，在 13 项权威评测中夺得视觉理解冠军，全面超越 GPT-4o 与 Claude3.5。	
DeepSeek-R1-0528	DeepSeek-R1-0528 是 DeepSeek 团队推出的最新版模型。模型基于 DeepSeek-V3-0324 训练，参数量达 660B。该模型通过利用增加的计算资源并在后训练期间引入算法优化机制，显著提高了其推理和推理能力的深度。该模型在各种基准测试评估中表现出出色的性能，包括数学、编程和一般逻辑。它的整体性能现在接近 O3 和 Gemini 2.5 Pro 等领先机型。	ff3f5c450f3b459cbe5d04a5ea9b2511

## 6.4. 错误处理

- 请求处理过程中出现异常时，服务会对外抛出非 200 的 http 状态码，表明当前请求无法正常完成。
- 对于异常的请求响应，请求体中会返回 error 结构，error 中返回具体的错误信息。
- 特殊地，在流式请求中：
  - 如果在流式请求接收处理之前发生了异常，如鉴权、参数校验等问题，与普通的非流式一样返回 http code，并带有 error 结构。
  - 如果在流式请求已经接收，会先对外返回流式请求连接建立的信息，此时 http code 为 200，而在后续模型流式返回过程中发生了异常，会在流失返回的 chunk 返回 error 结构，并终止当前的流式请求。

### 错误error结构

字段名称	二级字段	类型	必选	描述
error		object	是	错误信息
-	code	string	是	平台错误码
-	type	string	是	平台错误类型
-	message	string	是	平台错误详情

## 错误结果示例

```

{
  "error" : {
    "code" : "500001",
    "type" : "INVOKE_MODEL_ERROR",
    "message" : "服务接口异常，请联系管理员"
  }
}

```

## 错误码

HTTP 状态码	错误码	错误信息	错误描述
500	INVOKE_MODEL_ERROR	服务接口异常，请联系管理员	推理接口异常
400	PARAM_ERROR	参数不正确，请联系管理员	参数校验失败
500	SYSTEM_ERROR	系统错误，请联系管理员	系统内部错误
401	INCORRECT_API_KEY_PROVIDED	AppKey 不正确，请使用正确的 AppKey	鉴权信息错误
400	INVOKE_TEXT_AUDIT_ERROR	调用审查服务接口发生异常，请联系管理员	内容审核失败
400	TEXT_AUDIT_QUESTION_NOT_PASS	很抱歉，关于这个问题我无法提供相应的信息。如果您有其他问题，我将很愿意为您解答。	问题未通过内容审核
400	TEXT_AUDIT_ANSWER_NOT_PASS	很抱歉，关于这个问题我无法提供相应的信息。如果您有其他问题，我将很愿意为您解答。	答案未通过内容审核
429	RESOURCES_TIPS	免费额度已用完或已到期，或账户金额不足。请开通付费	免费试用限制

429	USER_RPM_RATELIMITING	用户请求 RPM 超限，请稍后重试	RPM (Requests Per Minute) 针对用户每分钟允许的请求次数超过限制
429	USER_TPM_RATELIMITING	用户请求 TPM 超限，请减少 tokens 后重试	TPM (Tokens Per Minute) 针对用户每分钟消耗的 token 超出限制
429	AK_BANNED	您的请求 AK 已被封禁，请更换 AK 或联系后台人员解封后重试	App Key 被封禁
429	MODEL_RPM_RATELIMITING	模型请求 RPM 超限，请稍后重试	RPM (Requests Per Minute) 针对模型侧每分钟允许的请求次数超过限制
429	MODEL_TPM_RATELIMITING	模型请求 TPM 超限，请减少 tokens 后重试	TPM (Tokens Per Minute) 针对模型侧的每分钟消耗的 token 超出限制

## 6.5. API

### 6.5.1. chat 对话API

#### 1 接口描述

-	<b>描述</b>
接口名称	对话
请求路径	<a href="https://wishhub-x6.ctyun.cn/v1/chat/completions">https://wishhub-x6.ctyun.cn/v1/chat/completions</a>

功能描述	针对描述会话的消息列表，模型将返回响应
------	---------------------

## 2 请求参数

### 2.1 请求头参数

参数	示例值	描述
Authorization	Bearer AppKey	鉴权信息填入 AppKey。
Content-Type	application/json	

### 2.2 请求参数

备注：此参数为全平台模型通用，每个模型支持的参数、参数范围可能因模型不同而有所差异，详细可见模型广场内每个模型的 API 文档。

参数名称	二级参数	三级参数	四级参数	类型	必选	描述
model				string	是	模型 ID。
messages				array	是	<p>用户当前输入的期望模型执行指令。一个列表内多个字典，支持多轮对话。</p> <p>对话列表，每个列表项为一个 message object，message object 中包含用户 role 和 content 两部分信息：</p> <p>role 可选值为 user、assistant、system；</p> <p>role 为 system 时，不校验 content 空值，且 message 中 system 只能位于开头，即 messages[0] 位置；</p> <p>role 为 user 时说明是用户提问，role 为 assistant 时说明是模型回答，而 content 为实际的对话内容；</p> <p>单轮/多轮对话中，最后一个 message 中 role 必须为 user，</p>

						content 为用户输入的最新问题，其余结果除 system 角色外都为历史信息拼接入 messages 中，assistant 和 user 的 role 只能交替出现，assistant 后只能跟 user，user 后只能跟 assistant。
-	role			string	否	对话角色，role 类型枚举值：user、assistant、system。
-	content			string/array	是	对话内容，内容目前有两种格式：string,array。 string 类型：表示文本对话内容。 array 类型：表示多个对话内容列表，每个列表项为一个 content object，每个 content object 包含 type、image_url、text 等信息。 type 可选值为 text、image_url。 type 为 text 时，取 text 字段作为对话内容。 type 为 image_url 时，取 image_url 字段作为对话内容。
-	-	type		string	否	对话内容类型，type 类型枚举值：text,image_url。
-	-	text		string	否	文本对话内容，type 为 text 时传入。
-	-	image_url		object	否	图片对话内容，type 为 image_url 时传入。
-	-	-	url	string	否	图片对话内容中的图片地址，目前可以为二进制数据的 base64 编码。
frequency_penalty				float	否	频率惩罚。它影响模型如何根据文本中词汇 token 的现有频率惩罚新词汇 token。值大于 0，会根据新标记在文本中的现有频率来惩罚新标记，从而降低模型逐字重复同一行的可能性。

						一般取值范围[-2, 2]， <b>具体取值范围、默认值需见对应模型。</b>
max_tokens				int	否	最大生成长度。控制最大生成长度，超过该值则截断。 一般取值范围(0, 2048]， <b>具体取值范围、默认值需见对应模型。</b>
n				int	否	1-n 个 choices。
presence_penalty				float	否	存在惩罚。用户控制模型生成时整个序列中的重复度。 一般取值范围[-2.0, 2.0]， <b>具体取值范围、默认值需见对应模型。</b>
response_format				object	否	返回格式。
-	type			string	否	返回格式枚举值： text, json_object。
seed				int	否	随机种子。用于指定推理过程的随机种子，相同的 seed 值可以确保推理结果的可重现性，不同的 seed 值会提升推理结果的随机性。 一般取值范围(0, 9223372036854775807]， <b>具体取值范围、默认值需见对应模型。</b>
stop				string/array	否	生成停止标识。当模型生成结果以 stop 中某个元素结尾时，停止文本生成。
stream				bool	否	是否以流式接口的形式返回数据。默认为 False，非流式。
stream_options				object	否	流式选项，stream 为 True 有效。
-	include_usage			bool	否	是否在返回中包含 usage，stream 为 True 有效。 <b>取值为 True 时，会在流式返回的最后一个 chunk 里返回 usage 信息，并该 chunk 中 choices 列</b>

						表为空。
temperature				float	否	温度采样。该值越高生成文本的多样性越高，该值越低生成文本的确定性越高。 一般取值范围(0, 2)， <b>具体取值范围、默认值需见对应模型。</b>
top_k				int	否	top_k 采样。取值越大，生成的随机性越高；取值越小，生成的确定性越高。 一般取值范围[1, 100]， <b>具体取值范围、默认值需见对应模型。</b>
top_p				float	否	top_p 采样。该值越高生成文本的多样性越高，该值越低生成文本的确定性越高。该值为 0 时没有随机性。 一般取值范围(0, 1]， <b>具体取值范围、默认值需见对应模型</b>
user				string	否	用户唯一身份 ID。

请求参数示例

```
{
  "model": "1234567890",    // 模型ID
  "messages": [
    {
      "role": "user",
      "content": "Hello!"
    }
  ]
}
```

## 3 请求返回

### 3.1 非流式返回

#### 3.1.1 非流式正常返回

字段名称	二级字段	三级字段	字段类型	描述
id			string	唯一标识符
choices			string	choices 列表
-	index		int	choice 索引
-	message		object	模型生成的消息
-	-	role	string	对话角色
-	-	content	string	对话消息内容

-	finish_reason		string	模型停止生成标记的原因。 stop: 模型生成遇到自然停止点或提供的停止序列; length: 达到请求中指定的最大标记数; content_filter: 如果由于内容过滤器中的标志而省略了内容 tool_calls/function_call: 模型调用了函数。
created			int	Unix 时间戳 (以秒为单位)。
model			string	调用的模型名称。
object			string	返回的对象类型。非流式返回始终为: chat.completion
usage			object	请求使用情况的统计信息。
-	completion_tokens		int	生成 token 数。
-	prompt_tokens		int	输入 token 数。
-	total_tokens		int	使用的 token 总数 (prompt + completion)。

#### 返回结果示例

```
{
  "id": "chatcml-123",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "xxx-chat",
  "choices": [{
    "index": 0,
    "finish_reason": "stop",
    "message": {
      "role": "assistant",
      "content": "\n\nHello there, how may I assist you
today?"
    }
  }],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21
  }
}
```

### 3.1.2 非流式异常返回

非流式异常返回时：

- http code 返回非 200。
- http body 中返回 error 结构，error 结构中包含 code、type、message、param 等信息，具体可见【推理服务 API-错误处理】章节内容。

**错误结果示例**

```

{
  "error" : {
    "code" : "500001",
    "type" : "INVOKE_MODEL_ERROR",
    "message" : "服务接口异常，请联系管理员"
  }
}

```

## 3.2 流式返回

### 3.2.1 流式正常返回

字段名称	二级字段	三级字段	字段类型	描述
id			string	唯一标识符
choices			string	choices 列表
-	index		int	choice 索引
-	delta		object	模型生成的消息
-	-	role	string	对话角色
-	-	content	string	对话消息内容
-	finish_reason		string	模型停止生成标记的原因。 stop: 模型生成遇到自然停止点或提供的停止序列; length: 达到请求中指定的最大标记数; content_filter: 如果由于内容过滤器中的标志而省略了内容 tool_calls/function_call: 模型调用了函数。

<b>created</b>			int	Unix 时间戳（以秒为单位）。
<b>model</b>			string	调用的模型名称。
<b>object</b>			string	返回的对象类型。流式返回始终为： chat.completion.chunk
<b>usage</b>			object	请求使用情况的统计信息。 仅在 stream_options: { "include_usage": true } 设置时显示。如 果存在，则它包含一个 null 值，但最后一个 块包含整个请求的 token 使用情况的统计信 息。
-	completi on_token s		int	生成 token 数。
-	prompt_t okens		int	输入 token 数。
-	total_to kens		int	使用的 token 总数 (prompt + completion) 。

#### 返回结果示例

```
    {"id":"chatcpl-123", "object":"chat.completion.chunk", "created":1694268190, "model":"xxx-chat", "choices":[{"index":0, "delta":{"role":"assistant"}, "finish_reason":null}]}

    {"id":"chatcpl-123", "object":"chat.completion.chunk", "created":1694268190, "model":"xxx-chat", "choices":[{"index":0, "delta":{"content":"Hello"}, "finish_reason":null}]}

    ....

    {"id":"chatcpl-123", "object":"chat.completion.chunk", "created":1694268190, "model":"xxx-chat", "choices":[{"index":0, "delta":{}, "finish_reason":"stop"}]}
```

`stream_options.include_usage` 为 `True` 时多返回一条包含 `usage` 流式消息。

```
{"id":"chatcpl-123", "object":"chat.completion.chunk", "created":1694268199, "model":"xxx-chat", "choices":[], "usage": {"prompt_tokens":9, "completion_tokens": 120, "total_tokens": 129}}
```

### 3.2.2 流式异常返回

**流式异常分为两种：**

- 如果在流式请求接收处理之前发生了异常，如鉴权、参数校验等问题，与普通的非流式一样返回 http code，并带有 error 结构。
- 如果在流式请求已经接收，会先对外返回流式请求连接建立的信息，此时 http code 为 200，而在后续模型流式返回过程中发生了异常，会在流失返回的 chunk 返回 error 结构，并终止当前的流式请求。

**流式请求建立后的异常返回示例**

```
    {"id":"chatcpl-123", "object":"chat.completion.chunk", "created":1694268190, "model":"xxx-chat", "choices":[{"index":0, "delta":{"role":"assistant"}, "finish_reason":null}]}

    {"id":"chatcpl-123", "object":"chat.completion.chunk", "created":1694268190, "model":"xxx-chat", "choices":[{"index":0, "delta":{"content":"Hello"}, "finish_reason":null}]}

    ....

    {"error":{"code":"500001", "type":"INVOKE_MODEL_ERROR", "message":"服务接口异常。请联系管理员"}}
```

## 4 请求示例代码

假设平台用户组AppKey=884c8fc4054548a7b1ca1123592f5b7，模型ID=96dcaaaaaaaaaaaaaa5ff55ea377831a，以此为例进行说明。

## 4.1 curl方式请求

```
curl --request POST \  
  --url https://wishhub-x6.ctyun.cn/v1/chat/completions \  
 \  
  --header 'Accept: */*' \  
  --header 'Accept-Encoding: gzip, deflate, br' \  
  --header 'Authorization: Bearer  
884c8fc4054548a7b1ca1123592f5b7' \  
  --header 'Content-Type: application/json' \  
  --data '{  
    "model": "96dcaaaaaaaaaaaaaa5ff55ea377831a",  
    "messages": [  
      {  
        "role": "user",  
        "content": "Hello"  
      }  
    ]  
  }'
```

## 4.2 python方式请求

```
import json

import requests

URL = "https://wishhub-x6.ctyun.cn/v1/chat/completions"

headers = {

    "Authorization": "Bearer

884c8fc4054548a7b1ca1123592f5b7",

    "Content-Type": "application/json"

}

data = {

    "model": "96dcaaaaaaaaaaaaa5ff55ea377831a",

    "messages": [

        {"role": "user", "content": "Hello"}

    ],

    "stream": True

}

try:

    response = requests.post(URL, headers=headers,

json=data, stream=True)

    if response.status_code != 200:

        print(response.text)

    else

        for line in response.iter_lines(chunk_size=8192,

decode_unicode=True):

            ## 处理请求

            if line :

                if "[DONE]" == line:

                    break

            # 去除data前缀:

            json_string = line.removeprefix("data:")

            # 转为json:
```

### 4.3 openai 客户端示例代码

```
import openai

from openai import OpenAI

client = OpenAI(base_url="https://wishhub-x6.ctyun.cn/v1",
api_key="884c8fc4054548a7b1ca1123592f5b7")

messages = [
    {"role": "user", "content": "Hello"}
]

try:
    stream = client.chat.completions.create(
        model="96dcaaaaaaaaaaaaa5ff55ea377831a",
        messages=messages,
        stream=True
    )
    # 流式
    for chunk in stream:
        print(chunk.choices[0].delta.content or "", end="",
flush=True)
    except openai.APIStatusError as e:
        print(f"APIStatusError: {e.status_code}, {e.message},
{e.body}")
    except openai.APIError as e:
        print(f"APIError: {e.body}")
    except Exception as e:
        print(f"Exception: {e}")
```

## 6.5.2. image 文本生图API

### 1 接口描述

-	描述
接口名称	文本生图
请求路径	https://wishhub-x6.ctyun.cn/v1/images/generations
功能描述	基于提示创建图像

### 2 请求参数

#### 2.1 请求头参数

参数	示例值	描述
Authorization	Bearer AppKey	鉴权信息填入 AppKey。
Content-Type	application/json	

#### 2.2 请求参数

备注：此参数为全平台模型通用，每个模型支持的参数、参数范围可能因模型不同而有所差异，详细可见模型广场内每个模型的 API 文档。

参数名称	参数类型	必选	描述
model	string	是	模型 ID。
prompt	string	是	所需图像的文本描述，最大长度 1000。

negative_prompt	string	否	避免生成负面图像的提示，最大长度 1000。
n	int	否	要生成的图像数。 一般取值为 1， <b>具体取值范围、默认值需见对应模型。</b>
num_inference_steps	int	否	去噪步骤的数量。去噪步骤越多，通常可以得到更高质量的图像，但推理速度会变慢。 一般取值范围 [1, 50]， <b>具体取值范围、默认值需见对应模型。</b>
quality	int	否	生成图像质量。 openai 参数，平台未启用。
size	string	否	输出的图像尺寸（高 x 宽）。 一般取值范围 360x640, 480x640, 512x512, 640x360, 640x480, 720x1280, 1024x1024, 1280x720 ，默认值为 512x512， <b>具体取值范围、默认值需见对应模型。</b>
response_format	string	否	返回图像的格式。 一般取值范围 url, b64_json, 默认值为 b64_json， <b>具体取值范围、默认值需见对应模型。</b>
style	string	否	生成图像风格。 一般取值范围 standard, portrait, landscape, cartoon, technology, photography, concept, 默认值为 standard， <b>具体取值范围、默认值需见对应模型。</b>

user	string	否	用户唯一身份 ID。
------	--------	---	------------

请求参数示例

```

{
  "model": "bfetrcdggdsdfsdf",
  "prompt": "A cute baby sea otter",
  "n": 1,
  "size": "512x512"
}

```

### 3 请求返回

#### 3.1 请求正常返回

字段名称	二级字段	字段类型	描述
created		string	Unix 时间戳（以秒为单位）。
data		array	图像列表
-	b64_json	string	b64_json 格式图片
-	url	string	url 格式图片
-	revised_prompt	string	实际使用的修改后的 prompt

返回结果示例

```
{
  "id": "1714459832",
  "data": [{
    "b64_json": "xxxxxxxxxxxxxxxxxxxxxxx"
  }]
}
```

### 3.2异常返回

异常返回时：

- http code 返回非 200。
- http body 中返回 error 结构，error 结构中包含 code、type、message、param 等信息，具体可见【推理服务 API-错误处理】章节内容。

错误结果示例

```
{
  "error" : {
    "code" : "500001",
    "type" : "INVOKE_MODEL_ERROR",
    "message" : "服务接口异常，请联系管理员"
  }
}
```

## 4 请求示例代码

假设平台用户组AppKey=884c8fc4054548a7b1ca1123592f5b7，模型  
ID=96dcaaaaaaaaaaaaa5ff55ea377831a，以此为例进行说明。

## 4.1 curl方式请求

```
curl --request POST \  
--url https://wishhub-x6.ctyun.cn/v1/images/generations \  
--header 'Accept: */*' \  
--header 'Accept-Encoding: gzip, deflate, br' \  
--header 'Authorization: Bearer  
884c8fc4054548a7b1ca1123592f5b7' \  
--header 'Content-Type: application/json' \  
--data '{  
  "model": "96dcaaaaaaaaaaaaaa5ff55ea377831a",  
  "prompt" : "A cute baby sea otter",  
  "n" : 1,  
  "size" : "512x512"  
}'
```

## 4.2 python方式请求

```
import json

import requests

URL = "https://wishhub-x6.ctyun.cn/v1/images/generations"

headers = {

    "Authorization": "Bearer

884c8fc4054548a7b1ca1123592f5b7",

    "Content-Type": "application/json"

}

data = {

    "model": "96dcaaaaaaaaaaaaa5ff55ea377831a",

    "prompt" : "A cute baby sea otter",

    "n" : 1,

    "size" : "512x512"

}

try:

    response = requests.post(URL, headers=headers, json=data)

    if response.status_code != 200:

        print(response.json())

    else:

        b64_file_json =

response.json()["data"][0]["b64_json"]

    except Exception as e:

        print(f"Exception: {e}")
```

### 4.3 openai 客户端示例代码

```
import openai

from openai import OpenAI

client = OpenAI(base_url="https://wishhub-x6.ctyun.cn/v1",
api_key="884c8fc4054548a7b1ca1123592f5b7")

try:
    response = client.images.generate(
        model="96dcaaaaaaaaaaaaa5ff55ea377831a",
        prompt="A cute baby sea otter",
    )
    print(f"{response.data[0].b64_json}")
except openai.APIStatusError as e:
    print(f"APIStatusError: {e.status_code}, {e.message},
{e.body}")
except openai.APIError as e:
    print(f"APIError: {e.body}")
except Exception as e:
    print(f"Exception: {e}")
```

### 6.5.3. Embeddings文本向量化API

#### 1 接口描述

-	描述
接口名称	嵌入
请求路径	<a href="https://wishhub-x6.ctyun.cn/v1/embeddings">https://wishhub-x6.ctyun.cn/v1/embeddings</a>

功能描述	创建表示输入文本的嵌入向量
------	---------------

## 2 请求参数

### 2.1 请求头参数

参数	示例值	描述
Authorization	Bearer 123456789	鉴权信息填入 AppKey
Content-Type	application/json	

### 2.2 请求参数

参数名称	参数类型	必选	描述
model	string	是	模型 ID。
input	object	是	字符串或数组类型，表示输入的文本，编码为字符串或令牌数组（整型数组、字符串数组、整型数组的数组）。要在单个请求中为多个输入获取嵌入，请传递字符串数组或令牌数组的数组。每个输入的长度不得超过 8192 个令牌。
encoding_format	string	否	编码格式，float 或 base64，默认为 float。
dimensions	int	否	输出的 embeddings 维度，仅支持 text-embedding-3 及其高版本。

user	string	否	用户唯一身份 ID。
------	--------	---	------------

### 3 请求返回

#### 3.1 请求正常返回

字段名称	二级字段	三级字段	字段类型	描述
model			string	调用的模型名称。
object			string	返回的对象类型，默认为 list
data			array	embedding 列表
-	index		int	index 索引
-	embeddings		array/string	embedding 嵌入向量的 float 数组 或 base64 字符串
-	object		string	默认为 embedding
usage			object	请求使用情况的统计信息。
-	prompt_tokens		int	输入 token 数。
-	total_tokens		int	使用的 token 总数。

#### 3.2 异常返回

异常返回时：

- http code 返回非 200。
- http body 中返回 error 结构，error 结构中包含 code、type、message、param 等信息，具体可见 OpenAPI 接口文档中的 error 结构描述及错误码部分介绍。

错误结果示例

```
{
  "error" : {
    "code" : "500001",
    "type" : "INVOKE_MODEL_ERROR",
    "message" : "服务接口异常，请联系管理员"
  }
}
```

## 4 请求示例代码

假设模型推理服务平台用户组 AppKey=884c8fc4054548a7b1ca1123592f5b7，模型 ID=96dcaaaaaaaaaaaaa5ff55ea377831a，以此为例进行说明。

### 4.1 curl方式请求

```
curl --request POST \  
  --url https://wishhub-x6.ctyun.cn/v1/embeddings \  
  --header 'Accept: */*' \  
  --header 'Accept-Encoding: gzip, deflate, br' \  
  --header 'Authorization: Bearer  
884c8fc4054548a7b1ca1123592f5b7' \  
  --header 'Content-Type: application/json' \  
  --data '{  
    "model": "4b38a36cabe247c8a996206984c9e2a61",  
    "input" : "A cute baby sea otter"  
  }'
```

## 4.2 python方式请求

```
import json

import requests

URL = "https://wishhub-x6.ctyun.cn/v1/embeddings"

headers = {

    "Authorization": "Bearer 884c8fc4054548a7b1ca1123592f5b7",

    "Content-Type": "application/json"

}

data = {

    "model": "4b38a36cabe247c8a996206984c9e2a61",

    "input" : "A cute baby sea otter"

}

try:

    response = requests.post(URL, headers=headers, json=data)

    if response.status_code != 200:

        print(response.json())

    else:

        embeddings = response.json()["data"][0]["embeddings"]

except Exception as e:

    print(f"Exception: {e}")
```

### 4.3 openai 客户端示例代码

```
import openai

from openai import OpenAI

client = OpenAI(base_url="https://wishhub-x6.ctyun.cn/v1",
api_key="884c8fc4054548a7b1ca1123592f5b7")

try:
    response = client.embeddings(
        model="4b38a36cabe247c8a996206984c9e2a61",
        input="A cute baby sea otter",
    )
    print(f"{response}")
except openai.APIStatusError as e:
    print(f"APIStatusError: {e.status_code}, {e.message},
{e.body}")
except openai.APIError as e:
    print(f"APIError: {e.body}")
except Exception as e:
    print(f"Exception: {e}")
```

## 6.5.4. Reranker 重排序API

### 1.接口描述

-	描述
接口名称	重排序
请求路径	<a href="https://wishhub-x6.ctyun.cn/v1/rerank">https://wishhub-x6.ctyun.cn/v1/rerank</a>

功能描述	通常用于计算查询和文档之间的相关性分数。
------	----------------------

## 2. 请求参数

### 2.1 请求头header参数

参数	类型	必选	示例值	描述
Content-Type	String	是	application/json	json格式
Authorization	String	是	Bearer APP_KEY	请求鉴权的 App Key, 由平台提供。Bearer 开头

### 2.2 请求体body参数

名称	类型	必选	示例值	描述
model	String	是	96dcaaaaaaaaaaaaa5ff55ea377831a	模型id, 请在模型详情页顶部区域查看, 样例 : 96dcaaaaaaaaaaaaa5ff55ea377831a
query	String	是	人工智能的发展趋势	待重排序的查询文本 (例如用户搜索词)
documents	Array	是	["人工智能正在快速发展", "机器学习是AI的核心技术", "大语言模型改变了AI领域"]	待排序的文档列表 (每个元素为一个文档文本, 例如搜索结果中的候选文本)
truncate_prompt_tokens	Int	否	8192	最大token值, 取值范围 (1, 8192)
top_n	Int	否	2	返回top_n个文档, 取值范围大于0, 且小于documents的数量
return_documents	Boolean	否	false	是否返回原文档

### 2.3 请求代码示例

使用前请替换具体的\${APP\_KEY}和\${MODEL\_ID}值

```
curl --location 'https://wishub-x6.ctyun.cn/v1/rerank' \
--header 'Authorization: Bearer ${APP_KEY}' \
--header 'Content-Type: application/json' \
--data '{
  "model": "${MODEL_ID}",
```

```

"query": "人工智能的发展趋势",
"documents": [
    "人工智能正在快速发展。",
    "机器学习是AI的核心技术。",
    "大语言模型改变了AI领域。",
    "自然语言是AI领域的里程碑"
],
"top_n": 2,
"return_documents": true
}'

```

### 3.返回值说明

#### 3.1 请求成功返回响应参数

名称	类型	描述	示例值	下级对象
code	Int	接口状态码，返回状态，返回0表示成功	0	
id	String	请求标识	endpoint_common_11684	
results	Array	模型返回的结果	-	results
created	Int	回答的创建时间，由模型侧返回	1719904068	
model	String	模型名称	Bge-Rerank-Large	

表results

名称	类型	描述	示例值	下级对象
----	----	----	-----	------

index	Int	在原文档中对应索引	0	
relevance_score	Float	分数	0.992	
document	Object	文档内容，当return_documents为true时，无此字段	-	document

### 表document

名称	类型	描述	示例值	下级对象
text	String	目标文档	人工智能正在快速发展	

## 3.2请求失败返回响应参数

字段名	二级字段	类型	描述	示例值
error	-	Object	错误信息	-
-	code	String	错误码，返回错误代码参考下面的错误代码列表	500002
-	type	String	返回对应的错误码	PARAM_ERROR
-	message	String	返回对应的错误信息	modelId不正确，请重试

## 3.3错误码

如遇模型调用失败且返回报错信息，可访问官网[错误处理](#)页面，寻求解决方法。

## 3.4返回值示例

### 请求成功返回值示例

```
{
  "code": 0,
  "id": "34870e0590084e40800f856bc8e08201",
  "model": "b2d640f305f64c52ab5a04cd8565a233",
  "results": [
```

```
{
  "index": 0,
  "document": {
    "text": "人工智能正在快速发展。"
  },
  "relevance_score": 0.992597
},
{
  "index": 2,
  "document": {
    "text": "大语言模型改变了AI领域。"
  },
  "relevance_score": 0.8079157
}
]
```

#### 请求失败返回值示例

```
{
  "error": {
    "code": "500001",
    "message": "算法服务接口异常，请联系管理员",
    "type": "INVOKE_MODEL_ERROR"
  }
}
```

## 6.5.5. 模型列表API

### 1.接口描述

-	描述
接口名称	模型列表
请求路径	https://wishhub-x6.ctyun.cn/v1/models
功能描述	用于获取用户模型列表

### 2.请求参数

#### 2.1 请求头参数

参数	示例值	描述
Authorization	Bearer AppKey	鉴权信息填入 AppKey。
Content-Type	application/json	

### 3.请求返回

#### 3.1请求正常返回

字段名称	二级字段	字段类型	描述
object		string	返回的对象类型，默认为 list

<code>data</code>		<code>array</code>	模型列表
-	<code>id</code>	<code>string</code>	模型 id 或者模型名称
-	<code>object</code>	<code>string</code>	默认为 model
	<code>created</code>	<code>string</code>	创建时间，默认为空
-	<code>owned_by</code>	<code>string</code>	组织 机构，默认为空

返回结果示例

```
{
  "object": "list",
  "data": [
    {
      "id": "测试模型名称1",
      "object": "model",
      "created": "",
      "owned_by": ""
    }
  ]
}
```

## 3.2异常返回

异常返回时：

- http code 返回非 200。
- http body 中返回 error 结构，error 结构中包含 code、type、message、param 等信息，具体可见【推理服务 API-错误处理】章节内容。

错误结果示例

```
{  
  "code": 500004,  
  "detail": "AppKey不正确, 请使用正确的AppKey",  
  "message": "INCORRECT_API_KEY_PROVIDED",  
  "error": {  
    "code": "500004",  
    "message": "AppKey不正确, 请使用正确的AppKey",  
    "type": "INCORRECT_API_KEY_PROVIDED"  
  }  
}
```

## 4.请求示例代码

假设平台用户组AppKey=884c8fc4054548a7b1ca1123592f5b7，以此为例进行说明

。

## 4.1 curl方式请求

```
curl --request get \  
  --url https://wishhub-x6.ctyun.cn/v1/models \  
  --header 'Accept: */*' \  
  --header 'Accept-Encoding: gzip, deflate, br' \  
  --header 'Authorization: Bearer  
884c8fc4054548a7b1ca1123592f5b7' \  
  --header 'Content-Type: application/json' \  

```

## 4.2 python方式请求

```
import json

import requests

URL = "https://wishhub-x6.ctyun.cn/v1/models"

headers = {

    "Authorization": "Bearer 884c8fc4054548a7b1ca1123592f5b7",

    "Content-Type": "application/json"

}

try:

    response = requests.get(URL, headers=headers)

    if response.status_code != 200:

        print(response.text)

    else:

        data = response.json()["data"]

        print(data)

except Exception as e:

    print(f"Exception: {e}")
```

### 4.3 openai 客户端示例代码

```
import openai

from openai import OpenAI

client = OpenAI(base_url="https://wishhub-
x6.ctyun.cn/v1/models", api_key="884c8fc4054548a7b1ca1123592f5b7")

try:
    response = openai.models.list()
    print(f"{response}")
except openai.APIStatusError as e:
    print(f"APIStatusError: {e.status_code}, {e.message},
{e.body}")
except openai.APIError as e:
    print(f"APIError: {e.body}")
except Exception as e:
    print(f"Exception: {e}")
```

## 7. 平台功能 OpenAPI

### 7.1 平台功能API使用说明

OpenAPI 门户提供了产品的 API 文档、API 调试、SDK 中心等。

关于用户如何使用产品功能 API 的详细介绍，请参见 [API 文档](#)。您可以在 OpenAPI 门户了解到具体的调用前必知、API 概览、如何调用 API 以及具体的 API 的接口详细说明。

## 8. 常见问题

### 8.1. 计费相关

#### 1. 模型推理服务支持哪些计费方式？

模型推理服务支持包周期计费模式、按需计费模式-卡时和按需计费模式-Tokens三种计费方式。

#### 2. 预付费和后付费的账单分别是以何种维度结算？

模型推理服务后付费账单是按月结算，预付费账单是按小时结算。

#### 3. 账户欠费后如何充值？

请前往天翼云官网进行充值，详细操作参见[在线充值](#)。

#### 4. 为什么账户欠费后仍在出账？

欠费自账单出具起算，而账单出账通常存在约1小时的延迟。在此期间，系统按照实际使用量正常出账。

#### 5. 大模型免费token额度使用规则是什么？

每个模型提供一定额度且限期2周的免费调用量，限期计算以首次使用时间为起始时间，主/子账号共用免费额度及有效期周期额度。

免费额度用完或到期后可开通付费服务继续付费使用，开通付费服务后仍优先消耗剩余免费额度，免费额度用完或到期后自动转入付费使用。具体模型对应免费额度可前往平台“在线推理--预置服务”列表查看。

#### 6. 大模型免费额度已用完，但该模型未开通计费服务怎么办？

方式1：将该模型部署为我的服务继续使用，详细操作参见“在线推理-我的服务”

方式2：请前往[天翼云官网工单管理模块](#)提交提额申请。

## 8.2. 平台操作

### 1. 平台已预置的模型有哪些？

进入模型服务模块，选择在线服务，点击【预置服务】，可以看到平台预置的模型，平台预置了多款等基础大模型，包括DeepSeek、通义千问、Llama、ChatGLM 等系列，可以直接使用。不同的基础模型的参数和能力不同，我们将持续推出不同能力方向的模型。

### 2. 使用第三方工具对接API访问报错404：请求路径错误怎么解决？

修改配置的请求路径配置到/v1，不同工具的具体配置请参考最佳实践文档。

### 3. 模型回答内容重复如何处理？

请求模型的重复惩罚度参数设置过高，会导致重复输出概率增大。建议将重复惩罚度参数设置为0。

### 4. API接口调用错误（错误码：500001）怎么解决？

大概率为客户端主动断开连接问题，请检查客户端设置的超时时间是否过短或者存在主动断开情况。

## 8.3. 用户权限

### 1. 主账号如何管理子账号的App Key？

主账号自身可在“服务接入”模块，选择对应子账号的用户名称查询其创建的服务组，支持主账号对子账号创建服务组进行创建、编辑、停用、删除等操作。

### 2. 主子账号在token额度使用中是什么规则？

免费额度：每个模型被赋予不等的免费额度和免费试用期限，具体免费额度可在页面中查看，免费期限从第一次使用该模型开始计算。免费额度用完或到期后，可以付费开通服务。暂不支持开通付费服务的模型可选择部署为我的服务后继续使用。

- 共享机制：主/子账号共用免费额度及有效期周期额度
- 不可分配：不支持主账号向子账号分配额度
- 有效期触发：主账号任一账号首次使用模型服务即启动2周有效期倒计时

付费使用：部分模型支持开通付费token服务。暂不支持开通付费token的模型可转为部署为“我的服务”。

- 独立开通：各账号可单独开通付费服务，操作互不影响。
- 状态独立：各账号页面显示自身付费状态，不关联其他账号。
- 统一扣费：所有子账号消费均从主账号余额扣除。

举例：子账号A已开通付费，成功后即可付费使用tokens；子账号B未开通，则子账号B不可付费使用tokens。

## 8.4. 如何联系我们

1、产品使用方面的问题如何反馈？

您可通过天翼云官网工单或者客服电话【400-810-9889】进行反馈。