高性能应用服务

最佳实践

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

最佳实践
快速使用 Stable Diffusion 文生图应用

最近更新时间：2024-05-10 16:40:52

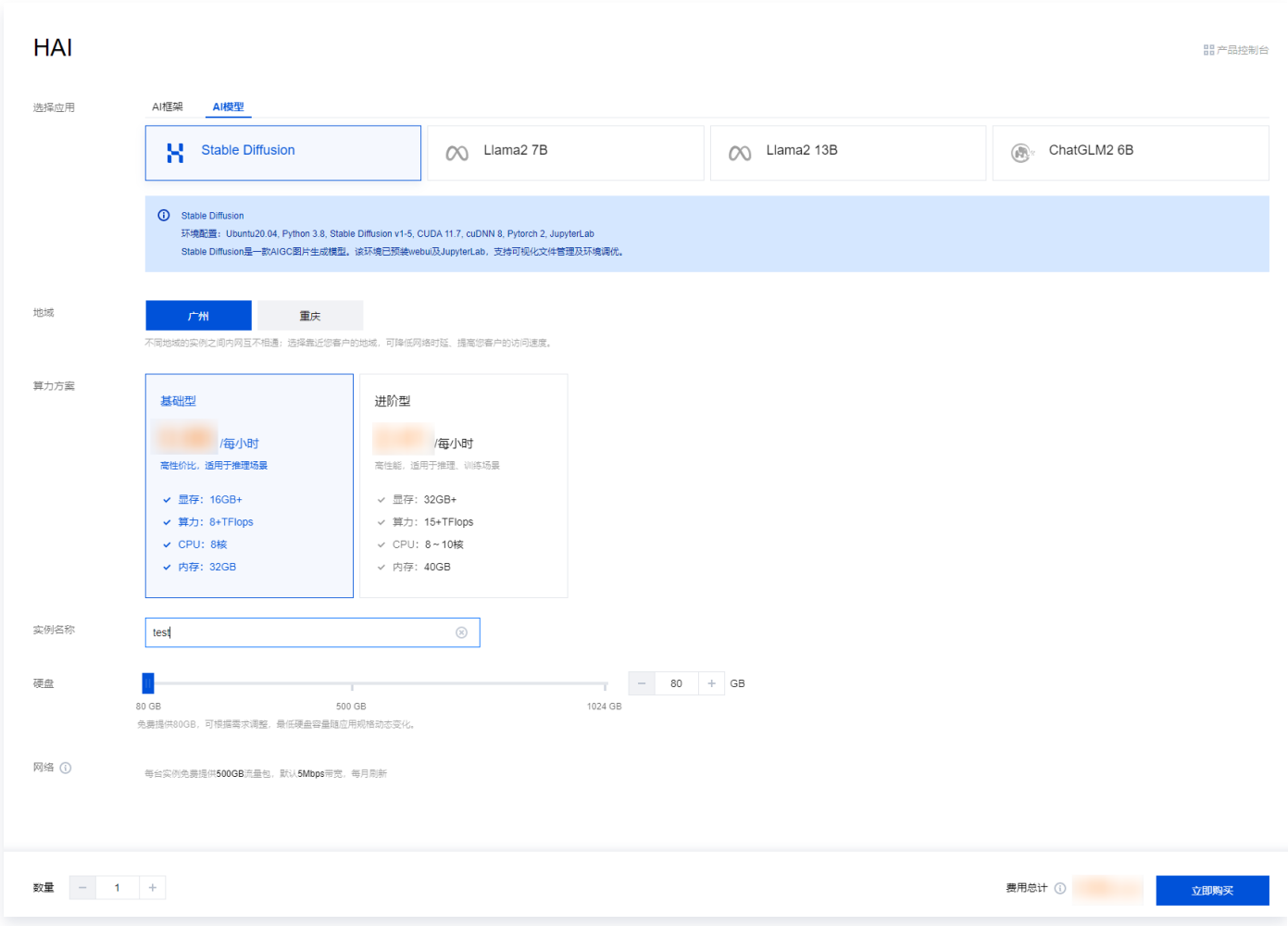
本次我们使用 腾讯云高性能应用服务 HAI 体验快速搭建并使用 AI 模型 StableDiffusion 进行文生图推理，实现思路如下：

- 体验 高性能应用服务HAI 一键部署 StableDiffusion AIGC。
- 启动 StableDiffusionWebUI 进行文生图模型推理。

操作步骤

步骤1：创建高性能应用服务

1. 登录 高性能应用服务控制台 。
2. 单击**新建**，进入 高性能应用服务购买页面 。

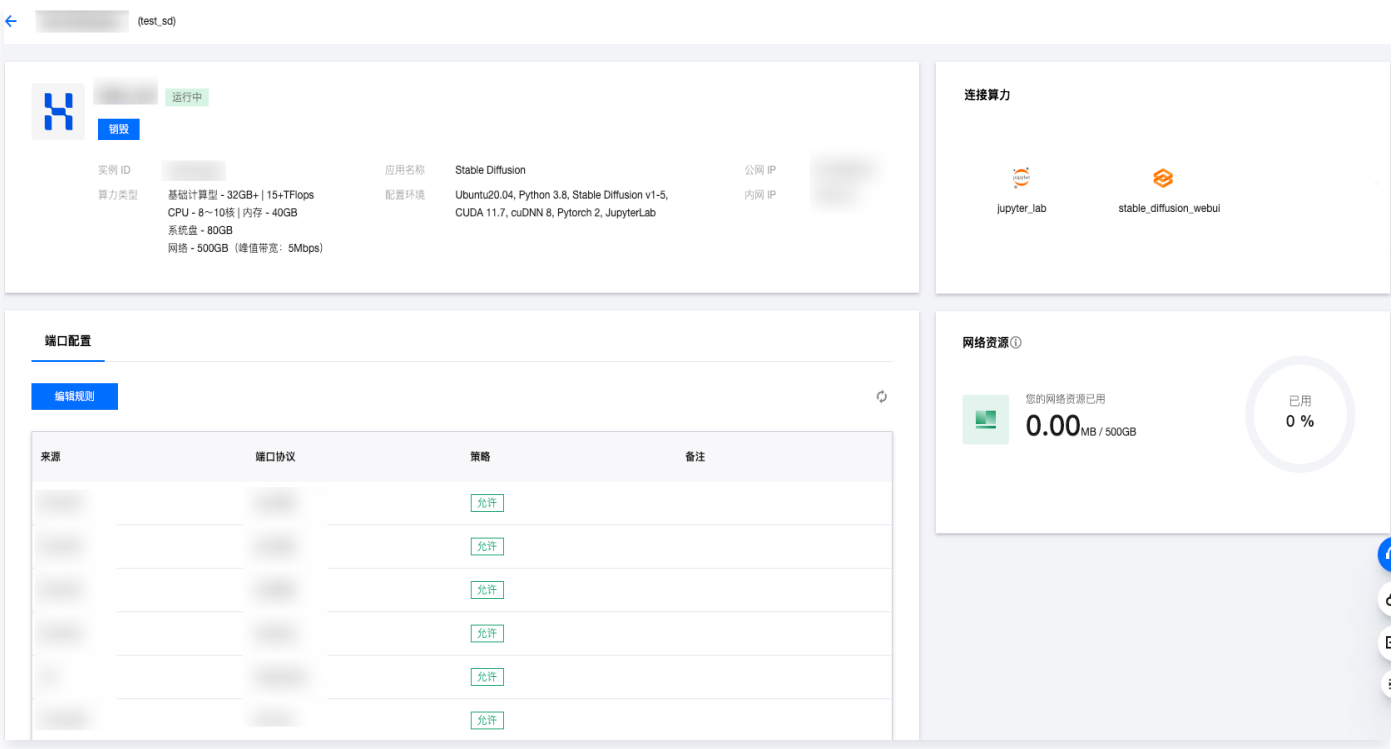


- **选择应用：**目前提供 AI 框架、AI 模型两类应用，请根据实际需求进行选择。
- **地域：**建议选择靠近目标客户的地域，降低网络延迟、提高您的客户的访问速度。

- **算力方案：**支持基础型及进阶型两类算力方案，本次算力方案选择进阶型，生成图片效率更高。
- **实例名称：**自定义实例名称，若不填则默认使用实例 ID 替代。
- **硬盘：**默认提供 80GB 免费空间，可根据实际使用需求进行调整。
- **网络：**每台实例每月免费提供 500GB 流量包，默认10Mbps 带宽，每月刷新。
- **购买数量：**默认1台。

3. 单击**立即购买**。
4. 核对配置信息后，单击**提交订单**，并根据页面提示完成支付。
5. 等待创建完成。单击实例**任意位置**并进入该实例的详情页。

6. 您可以在此页面查看 Stable Diffusion 详细的配置信息。



步骤2：启动 HAI 实例进行文生图模型推理

1. 在实例列表中选择**算力连接** > **Gradio WebUI** 并进入该实例的详情页。

2. 使用高性能应用服务 HAI 部署的 Stable Diffusion WebUI 快速进行 AI 绘画。

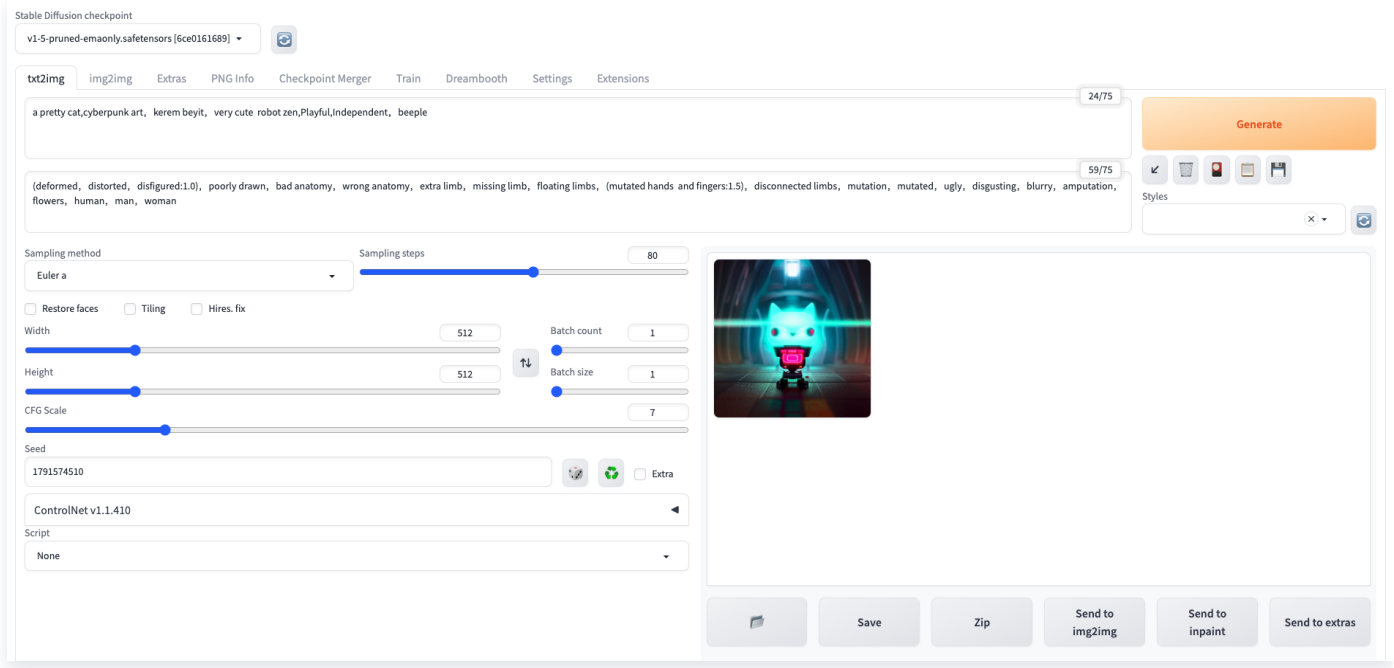
> **注意：**
> 提示词（Prompt）越多，AI 绘图结果会更加精准。另外，目前中文提示词的效果不好，还得使用英文提示词。

接下来我们使用 Stable Diffusion WebUI 生成一张猫咪图片，配置以下参数后，单击 **Generate** 即可。

参数名	描述	值
Prompt	主要描述图像，包括内容风格等信息，原始的 WebUI会对此处有字数	a pretty cat,cyberpunk art，kerem beyit，very cute robot

	的限制，您可以通过安装一些插件来突破字数的限制。	zen,Playful,Independent，beeple
Negative prompt	为了提供给模型，您不需要的风格。	(deformed，distorted，disfigured:1.0)，poorly drawn，bad anatomy，wrong anatomy，extra limb，missing limb，floating limbs，(mutated hands and fingers:1.5)，disconnected limbs，mutation，mutated，ugly，disgusting，blurry，amputation，flowers，human，man，woman
CFG scale	分类器自由引导尺度，图像与提示符的一致程度。越低的值产生的结果越有创意，数值越大成图越贴近描述文本。一般设置为7。	7
Sampling method	扩散算法的去噪声采样模式会影响最终效果，不同的采样模式的结果会有很大差异，一般是默认选择 euler。	Euler a
Sampling steps	在使用扩散模型生成图片时所进行的迭代步骤。需要注意的是，更高的迭代步数会消耗更多的计算时间和成本，但并不意味着一定会得到更好的结果。	80
Seed	随机数种子，生成每张图片时的随机种子。	1791574510

3. 配置并生成如下图：


Stable Diffusion checkpoint
v1-5-pruned-emaonly.safetensors [6ce0161689]
txt2img
img2img
Extras
PNG Info
Checkpoint Merger
Train
Dreambooth
Settings
Extensions
a pretty cat,cyberpunk art, kerem beyit, very cute robot zen,Playful,Independent, beeple
24/75
(deformed, distorted, disfigured:1.0), poorly drawn, bad anatomy, wrong anatomy, extra limb, missing limb, floating limbs, (mutated hands and fingers:1.5), disconnected limbs, mutation, mutated, ugly, disgusting, blurry, amputation, flowers, human, man, woman
59/75
Generate
Styles
Sampling method
Euler a
Sampling steps
80
Restore faces
Tiling
Hires. fix
Width
512
Height
512
Batch count
1
Batch size
1
CFG Scale
7
Seed
1791574510
Extra
ControlNet v1.1.410
Script
None
Save
Zip
Send to img2img
Send to inpaint
Send to extras

快速构建 Stable Diffusion 文生图 API 服务

最近更新时间：2024-04-15 15:56:41

本次我们使用 腾讯云高性能应用服务 HAI 体验快速搭建并使用 AI 模型 StableDiffusion ，实现思路如下：

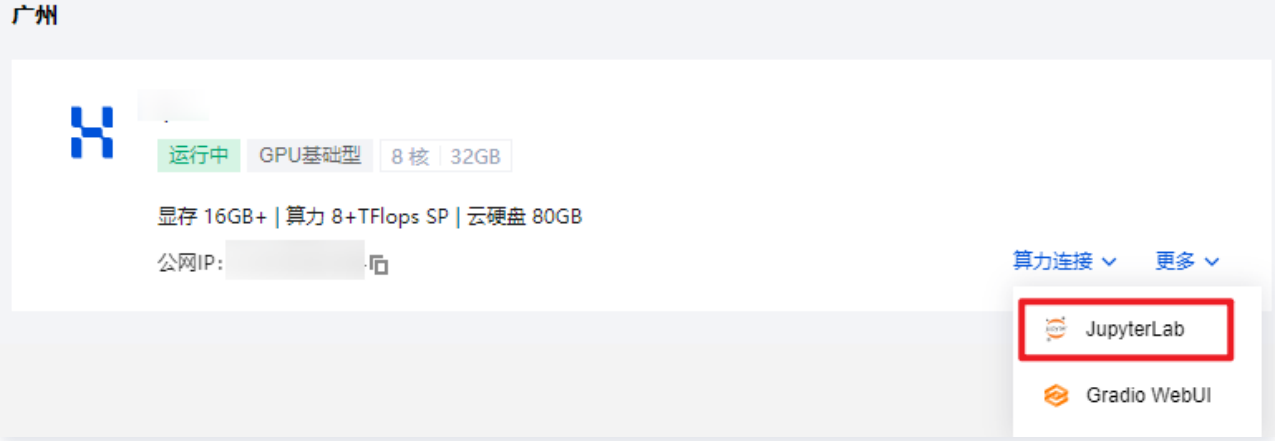
- 提前通过高性能应用服务 HAI 部署成功 StableDiffusion 应用。
- 基于部署好的应用，利用体验 JupyterLab 进行 StableDiffusion API 的部署。

前提

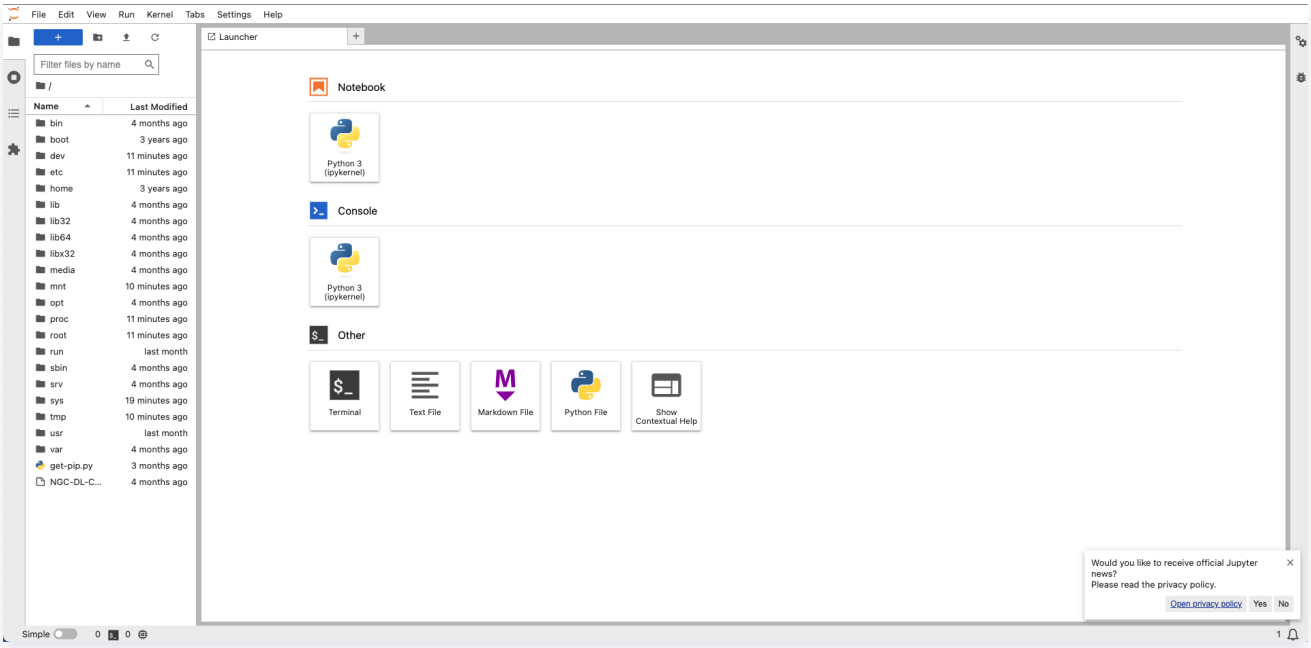
在部署 API 服务之前，请确保您已成功部署 StableDiffusion 应用。详细步骤可参见 快速使用 Stable Diffusion 文生图应用 。

部署 API 服务

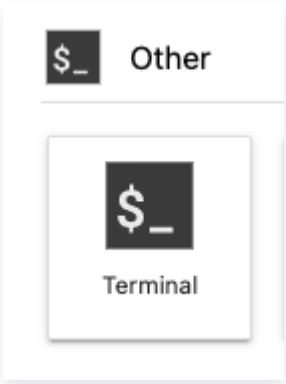
1. 进入 jupyter_lab 控制台 操作界面。
   1.1 在实例列表中选择**更多** > **JupyterLab** 并进入该实例的详情页。



   1.2 初步认识并操作 JupyterLab。

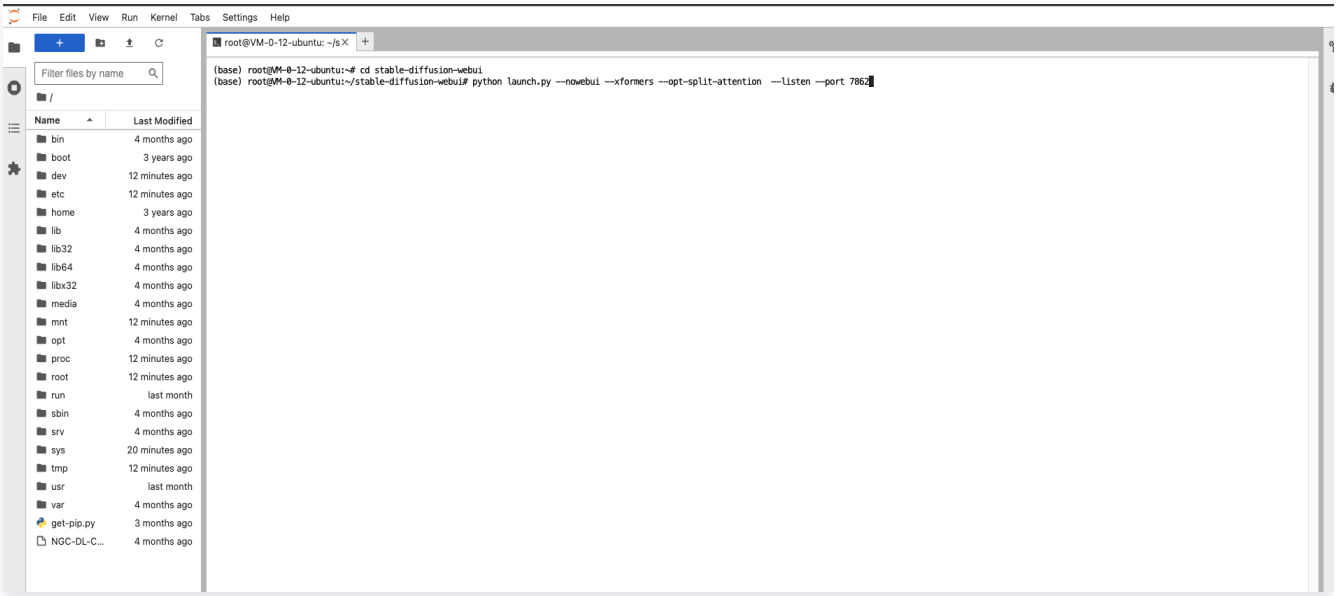


1.3 选择使用终端命令行操作。



输入代码：

```
cd stable-diffusion-webui
python launch.py --nowebui --xformers --opt-split-attention  --listen --port 7862
```



命令参数描述如下图：

命令	描述
--nowebui	以 API 模式启动。
--xformers	改善内存消耗和速度。
--opt-split-attention	Cross attention layer optimization 优化显着减少了内存使用。
--listen	默认启动绑定的 IP 是 127.0.0.1。
--port	默认端口是7860，可以配置并修改该参数，例如：--port 7862。
--gradio-auth username:password	如果希望给 WebUI 设置登录密码，可以配置该参数，例如：--gradio-auth GitLqr:123456。

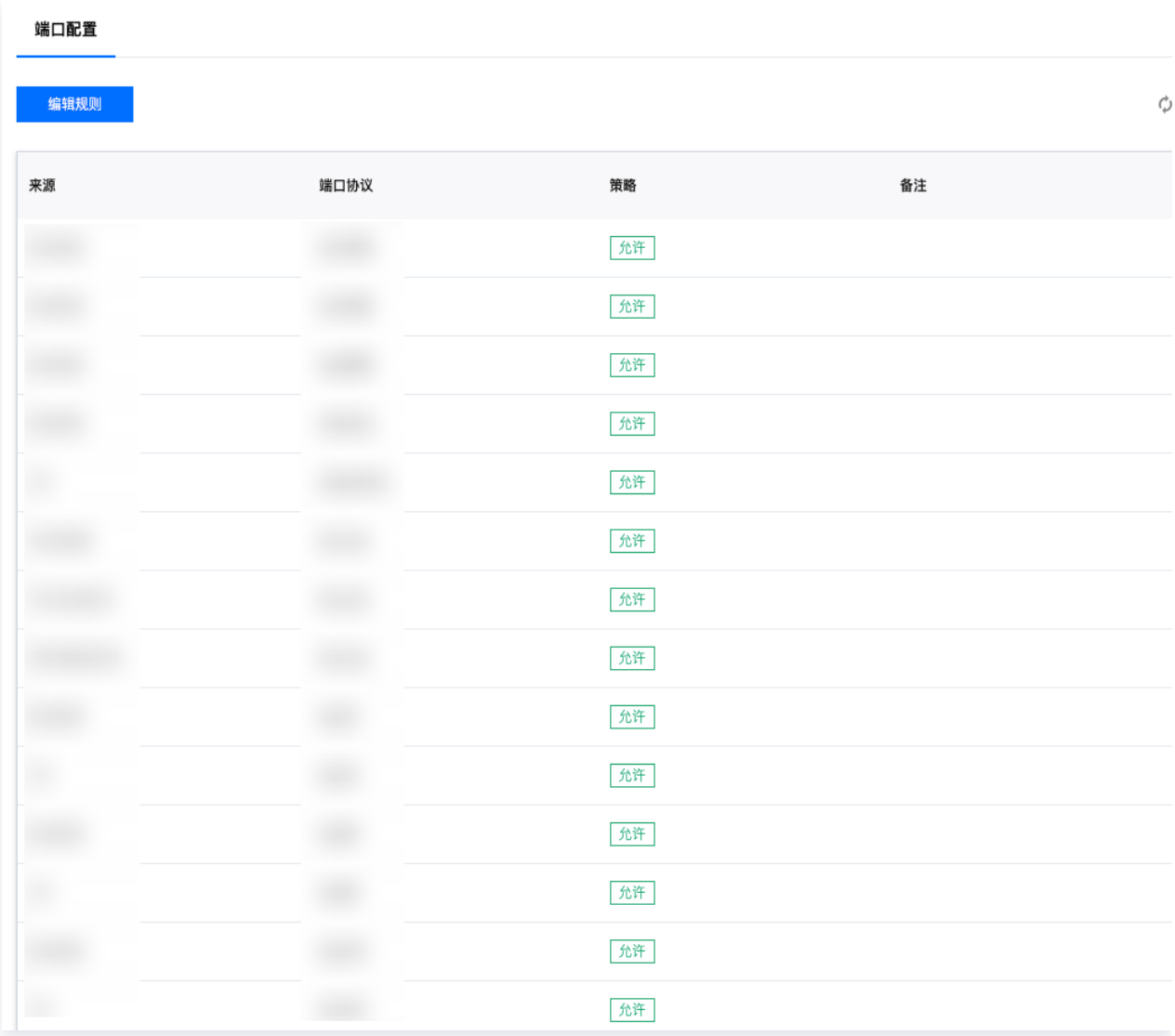
操作截图如下图所示：

```
(base) root@VM-0-12-ubuntu:~/stable-diffusion-webui# python launch.py --nowebui --xformers --opt-split-attention  --listen --port 7862
Python 3.8.10 (default, Jun  4 2021, 15:09:15)
[GCC 7.5.0]
Version: v1.5.2
Commit hash: c9c8485bc1e8720aba70f029d25cba1c4abf2b5c
Installing requirements
If submitting an issue on github, please provide the full startup log for debugging purposes.

Initializing Dreambooth
Dreambooth revision: cf086c536b141fc522ff11f6cffc8b7b12da04b9
Successfully installed accelerate-0.21.0 fastapi-0.94.1 gitpython-3.1.40 transformers-4.30.2

Does your project take forever to startup?
Repetitive dependency installation may be the reason.
Automatic1111's base project sets strict requirements on outdated dependencies.
If an extension is using a newer version, the dependency is uninstalled and reinstalled twice every startup.

[+] xformers version 0.0.17 installed.
[+] torch version 2.0.1 installed.
[+] torchvision version 0.15.2 installed.
[+] accelerate version 0.21.0 installed.
[+] diffusers version 0.19.3 installed.
[+] transformers version 4.30.2 installed.
[+] bitsandbytes version 0.35.4 installed.


Launching API server with arguments: --nowebui --xformers --opt-split-attention --listen --port 7862
[2023-11-01 06:34:58,839][DEBUG][git.cmd] - Popen(['git', 'version'], cwd=/root/stable-diffusion-webui, stdin=None, shell=False, universal_newlines=False)
[2023-11-01 06:34:58,867][DEBUG][git.cmd] - Popen(['git', 'version'], cwd=/root/stable-diffusion-webui, stdin=None, shell=False, universal_newlines=False)
================================================================================
You are running xformers 0.0.17.
The program is tested to work with xformers 0.0.20.
To reinstall the desired version, run with commandline flag --reinstall-xformers.

Use --skip-version-check commandline argument to disable this check.
================================================================================
2023-11-01 06:34:59,976 - ControlNet - INFO - ControlNet v1.1.410
ControlNet preprocessor location: /root/stable-diffusion-webui/extensions/sd-webui-controlnet/annotator/downloads
2023-11-01 06:35:00,089 - ControlNet - INFO - ControlNet v1.1.410
Loading weights [6ce0161689] from /root/stable-diffusion-webui/models/Stable-diffusion/v1-5-pruned-emaonly.safetensors
Creating model from config: /root/stable-diffusion-webui/configs/v1-inference.yaml
LatentDiffusion: Running in eps-prediction mode
DiffusionWrapper has 859.52 M params.
Model loaded in 2.6s (load weights from disk: 0.2s, create model: 0.5s, apply weights to model: 0.5s, apply half(): 0.3s, move model to device: 0.6s, calculate empty prompt: 0.4s).
[2023-11-01 06:35:03,462][DEBUG][api.py] - SD-Webui API layer loaded
Applying attention optimization: xformers... done.
[2023-11-01 06:35:03,818][DEBUG][api.py] - Loading Dreambooth API Endpoints.
Startup time: 48.4s (launcher: 37.5s, import torch: 2.5s, import gradio: 0.8s, setup paths: 1.2s, other imports: 1.9s, load scripts: 4.4s).
INFO:     Started server process [220]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:7862 (Press CTRL+C to quit)
```

1.4 添加高性能应用服务 HAI 的端口配置，使外部网络能够顺利地访问该服务器提供的 API 服务。

1.4.1 在算力管理页面。单击实例空白进入详情设置页。

1.4.2 在端口配置弹窗中，单击**编辑规则**。



1.4.3 在**安全组规则**页面中，在**入站规则**页签单击**添加规则**。

配置参考如下：

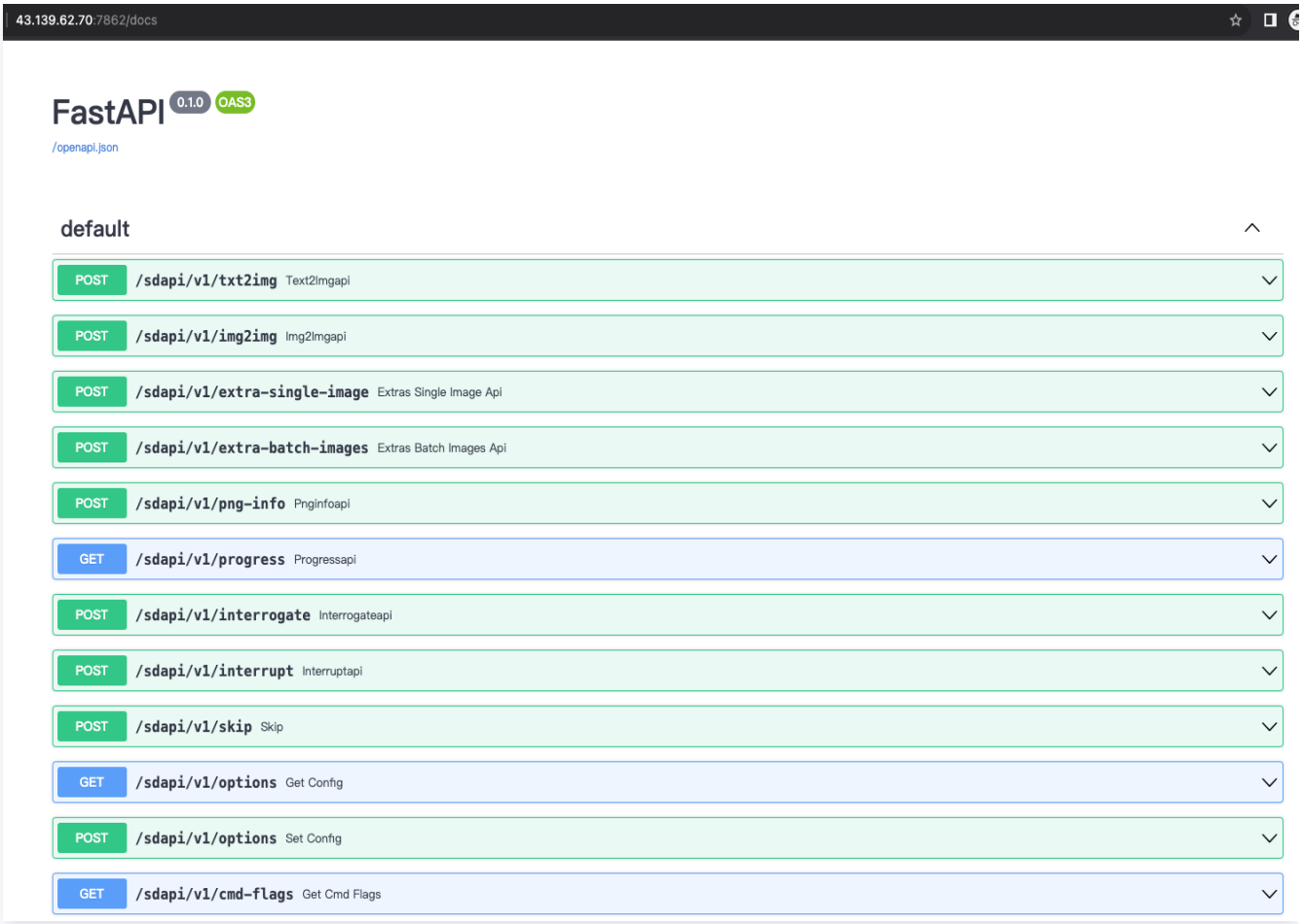
来源：0.0.0.0/0
协议端口：TCP:7862（**根据您配置的端口填写**）

2. 启动 StableDiffusion API 接口使用指南

2.1 配置完成后，在浏览器地址栏输入服务器 IP 地址:端口号/docs 可查看相关的 API 接口使用指南。

官方提供的常用 API 如下：

```
/sdapi/v1/txt2img文字生图 POST
/sdapi/v1/img2img图片生图 POST
/sdapi/v1/options获取设置 GET | 更新设置 POST（可用来更新远端的模型）
/sdapi/v1/sd-models获取所有的模型 GET
```



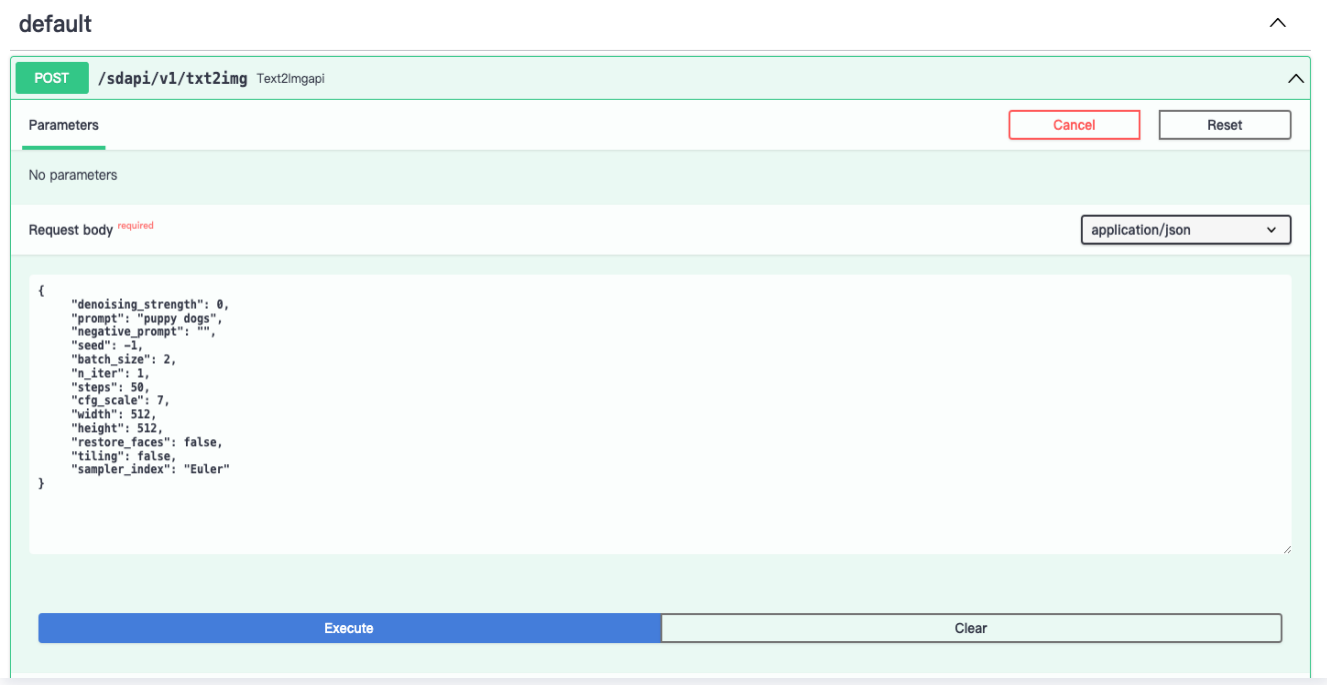
2.2 查看相关接口示例（`/sdapi/v1/txt2img`）。

常用输入如下：

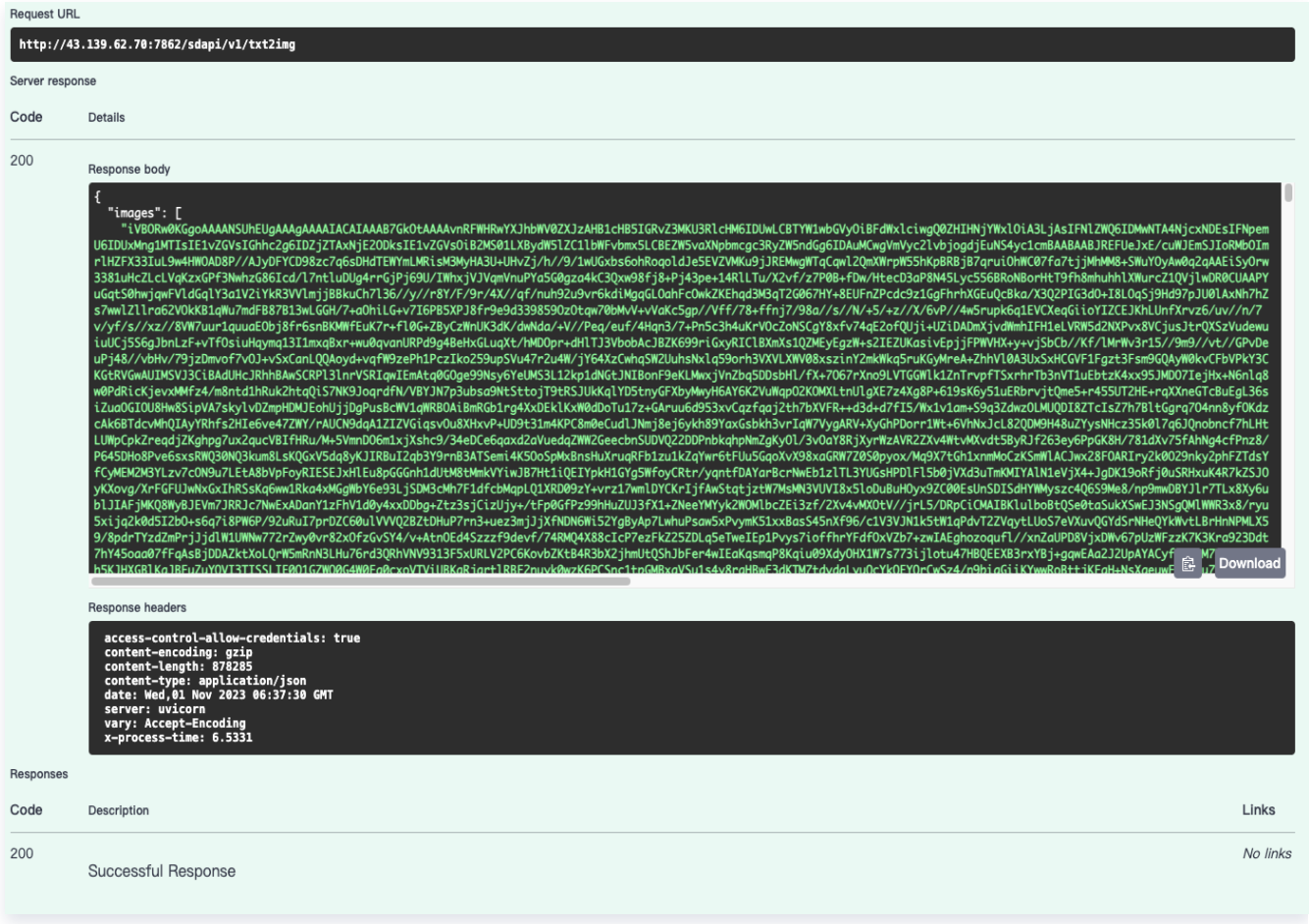
```
{
    "denoising_strength": 0,
    "prompt": "puppy dogs",
    "negative_prompt": "",
    "seed": -1,
    "batch_size": 2,
    "n_iter": 1,
    "steps": 50,
    "cfg_scale": 7,
    "width": 512,
    "height": 512,
    "restore_faces": false,
    "tiling": false,
    "sampler_index": "Euler"
}
```

可复制以上参数到 Request body 中。

名称	说明
prompt	提示词
negative_prompt	反向提示词
seed	种子，随机数
batch_size	每次张数
n_iter	生成批次
steps	生成步数
cfg_scale	关键词相关性
width	宽度
height	高度
restore_faces	脸部修复
tiling	可平铺
sampler_index	采样方法



请求 API 接口成功截图如下：



返回的格式如下：

```
{
  "images": [...], // 这里是一个base64格式的字符串数组，根据请求的图片数量而定
  "parameters": {  ... },//此处为输入的body
  "info": "{...}"// 返回的图片的信息
}
```

当看到类似上图的消息时，说明已经成功与远端的服务器进行连接！如果希望验证结果的图片的实际展示效果，可以复制 images 中的其中一张图片的 base64 格式的字符串，到相关的网站下转换为 jpg 格式。

3. 使用 Python 向高性能应用服务 HAI 提供的 StableDiffusionAPI 发送请求。

以下演示如何使用 Python 向 StableDiffusion API 发出请求。向应用程序的 txt2img（即**文本到图像**）API 发送 POST 请求以简单地生成图像。

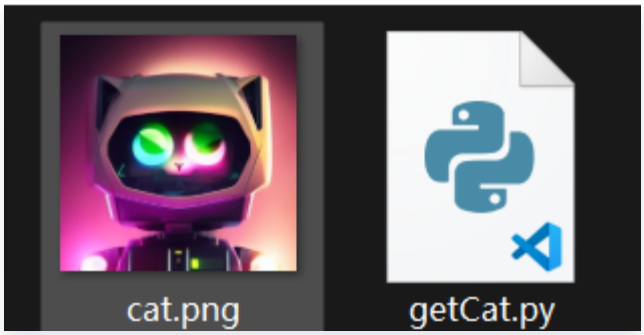
我们将使用 requests 包，如果您还没有安装，请使用安装脚本：

```
pip install requests
```

我们可以发送一个包含提示的请求作为一个简单的字符串。服务器将返回一个图像作为 base64 编码的 PNG 文件，我们需要对其进行解码。 要解码 base64 图像，我们只需使用 base64.b64decode(b64_image)。以下使用 Python 作为脚本代码测试：

```
import json
import base64
import requests

your_ip = '0.0.0.0' # HAI服务器IP地址
your_port =7862  # SD api 监听的端口

def submit_post(url: str,data: dict):
  """
  Submit a POST request to the given URL withthe given data.
  """
  return requests.post(url,data=json.dumps(data))

def save_encoded_image(b64_image: str,output_path: str):
  """
  Save the given image to the given outputpath.
  """
  with open(output_path,"wb") asimage_file:
      image_file.write(base64.b64decode(b64_image))

if __name__ == '__main__':
  #/sdapi/v1/txt2img
  txt2img_url = f'http://{your_ip}:{your_port}/sdapi/v1/txt2img'
  data = {
     'prompt': 'a pretty cat,cyberpunk art,kerem beyit,verycute robot
zen,Playful,Independent,beeple |',
     'negative_prompt':'(deformed,distorted,disfigured:1.0),poorlydrawn,bad
anatomy,wrong anatomy,extra limb,missing limb,floating limbs,(mutatedhands and
fingers:1.5),disconnectedlimbs,mutation,mutated,ugly,disgusting,blurry,amputatio
n,flowers,human,man,woman',
     'Steps':50,
     'Seed':1791574510
  }
  response = submit_post(txt2img_url,data)
  save_encoded_image(response.json()['images'][0],'cat.png')
```

请记住，您的结果会与上述示例有所不同。 如果遇到问题，请仔细检查运行 StableDiffusionAPI 应用程序的终端的输出。 如果您遇到**404 Not Found** 的问题，请仔细检查 URL 是否输入正确并指向正确的地址（例如 127.0.0.1）。



服务端可查看每一次接口调用详情：

```
100%| | 50/50 [00:07<00:00,  7.00it/s]
[2023-09-24 04:17:42,047][INFO][modules.shared] - Ending job scripts_txt2img (7.38 seconds)
INFO:      [redacted]:4207 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:18:32,754][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  7.01it/s]
[2023-09-24 04:18:40,121][INFO][modules.shared] - Ending job scripts_txt2img (7.37 seconds)
INFO:      [redacted]:4240 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:18:51,476][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  6.99it/s]
[2023-09-24 04:18:58,864][INFO][modules.shared] - Ending job scripts_txt2img (7.39 seconds)
INFO:      [redacted]:4252 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:18:59,581][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  7.04it/s]
[2023-09-24 04:19:06,918][INFO][modules.shared] - Ending job scripts_txt2img (7.34 seconds)
INFO:      [redacted]:4258 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:19:19,226][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  6.93it/s]
[2023-09-24 04:19:26,679][INFO][modules.shared] - Ending job scripts_txt2img (7.45 seconds)
INFO:      [redacted]:4267 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:21:08,028][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  6.99it/s]
[2023-09-24 04:21:15,416][INFO][modules.shared] - Ending job scripts_txt2img (7.39 seconds)
INFO:      [redacted]:4345 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:23:03,304][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  7.02it/s]
[2023-09-24 04:23:10,662][INFO][modules.shared] - Ending job scripts_txt2img (7.35 seconds)
INFO:      [redacted]:4450 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:25:37,844][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  7.05it/s]
[2023-09-24 04:25:45,181][INFO][modules.shared] - Ending job scripts_txt2img (7.34 seconds)
INFO:      [redacted]:4604 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:26:32,920][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  7.00it/s]
[2023-09-24 04:26:40,295][INFO][modules.shared] - Ending job scripts_txt2img (7.38 seconds)
INFO:      [redacted]:4626 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:27:40,003][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  7.03it/s]
[2023-09-24 04:27:47,359][INFO][modules.shared] - Ending job scripts_txt2img (7.36 seconds)
INFO:      [redacted]:4722 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:28:31,724][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  6.98it/s]
[2023-09-24 04:28:39,120][INFO][modules.shared] - Ending job scripts_txt2img (7.40 seconds)
INFO:      [redacted]:4737 - "POST /sdapi/v1/txt2img HTTP/1.1" 200 OK
[2023-09-24 04:29:21,187][INFO][modules.shared] - Starting job scripts_txt2img
100%| | 50/50 [00:07<00:00,  6.99it/s]
[2023-09-24 04:29:28,576][INFO][modules.shared] - Ending job scripts_txt2img (7.39 seconds)
```

快速使用 ChatGLM 对话模型应用

最近更新时间：2024-05-10 16:40:52

背景介绍

腾讯云高性能应用服务 HAI 是为开发者量身打造的澎湃算力平台。无需复杂配置，便可享受即开即用的 GPU 云服务体验。在 HAI 中，根据应用智能匹配并推选出最适合的 GPU 算力资源，以确保您在数据科学、LLM、AI 作画等高性能应用中获得最佳性价比。

HAI 服务优势

- 智能选型：根据应用匹配推选 GPU 算力资源，实现最高性价比。同时，打通必备云服务组件，大幅简化云服务配置流程。
- 一键部署：分钟级自动构建 LLM、AI 作画等应用环境。提供多种预装模型环境，包含如 StableDiffusion、ChatGLM2 等热门模型。
- 可视化界面：友好的图形界面，AI 调试更为简单。

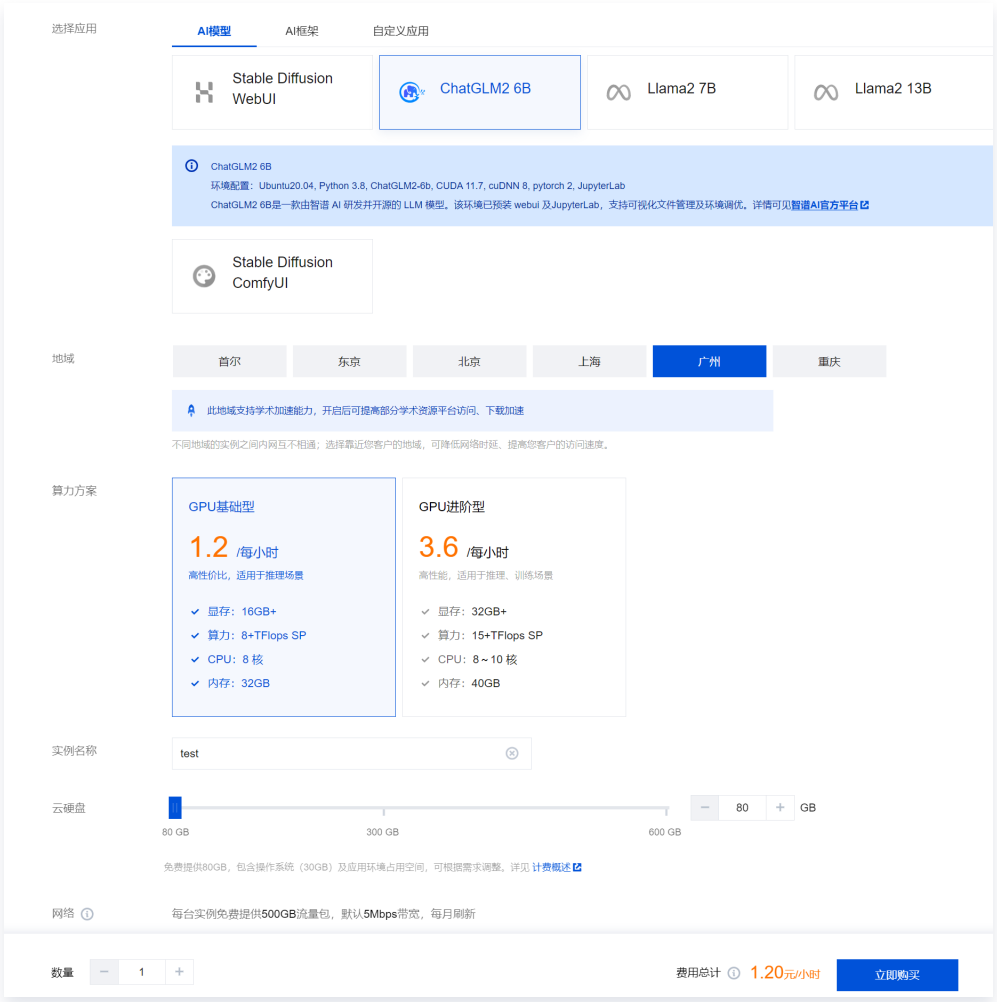
场景介绍

本次我们使用 腾讯云高性能应用服务 HAI 体验快速搭建并使用 AI 模型 **ChatGLM2-6B**，实现思路如下：

- 体验高性能应用服务 HAI 一键部署 **ChatGLM2-6B**。
- 启动 **Gradio WebUI** 进行对话生成。

步骤一：快速部署

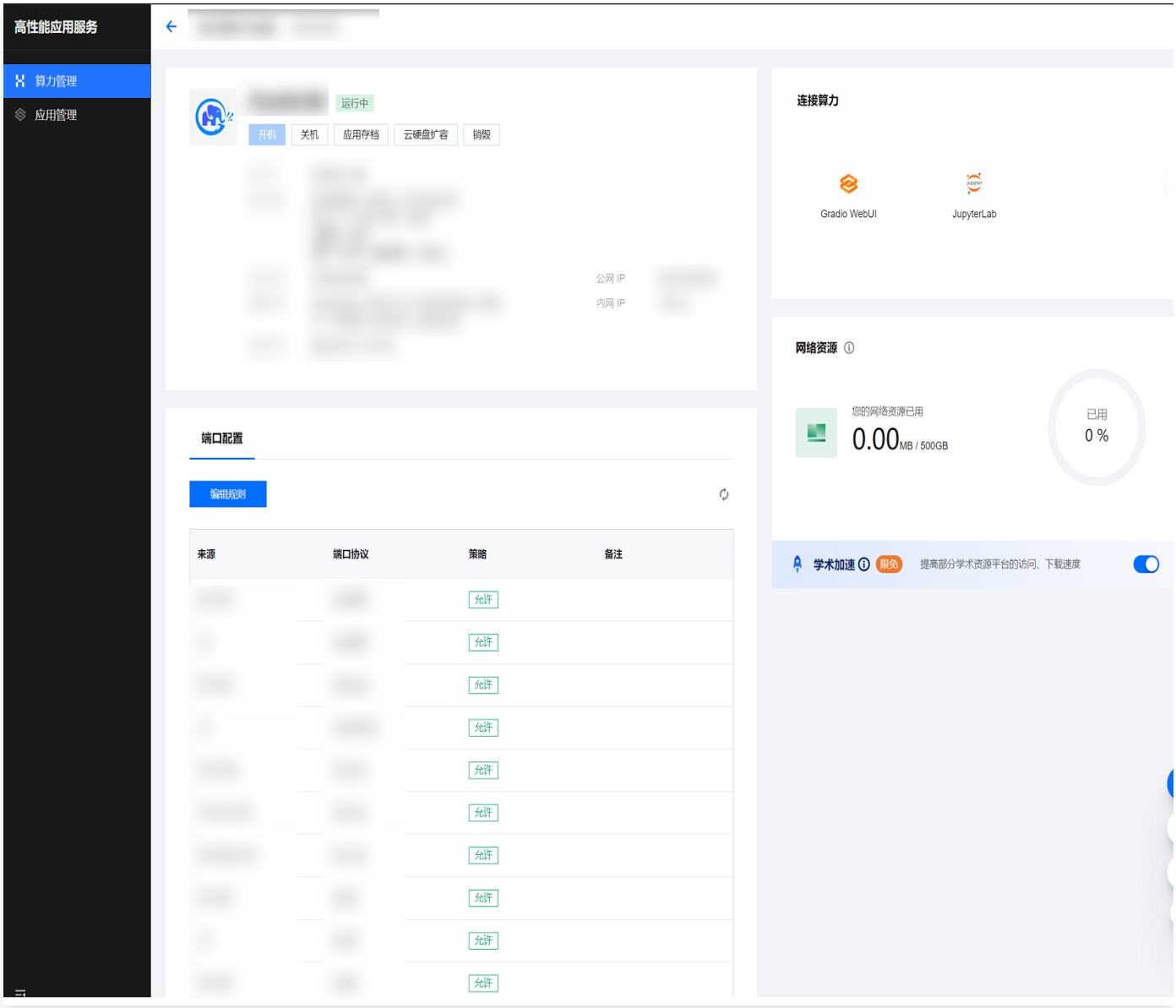
1. 登录 高性能应用服务控制台 。
2. 单击**新建**，进入 高性能应用服务购买页面 。



- **选择应用：**目前提供 AI 框架、AI 模型两类应用，请根据实际需求进行选择。
- **地域：**建议选择靠近目标客户的地域，降低网络延迟、提高您的客户的访问速度。
- **算力方案：**支持基础型及进阶型两类算力方案，本次算力方案选择进阶型，对话生成效率更高。
- **实例名称：**自定义实例名称，若不填则默认使用实例 ID 替代。
- **硬盘：**默认提供 80GB 免费空间，可根据实际使用需求进行调整。
- **网络：**每台实例每月免费提供 500GB 流量包，默认 10Mbps 带宽，每月刷新。
- **购买数量：**默认1台。

3. 单击**立即购买**。
4. 核对配置信息后，单击**提交订单**，并根据页面提示完成支付。
5. 等待创建完成。单击实例**任意位置**并进入该实例的详情页。

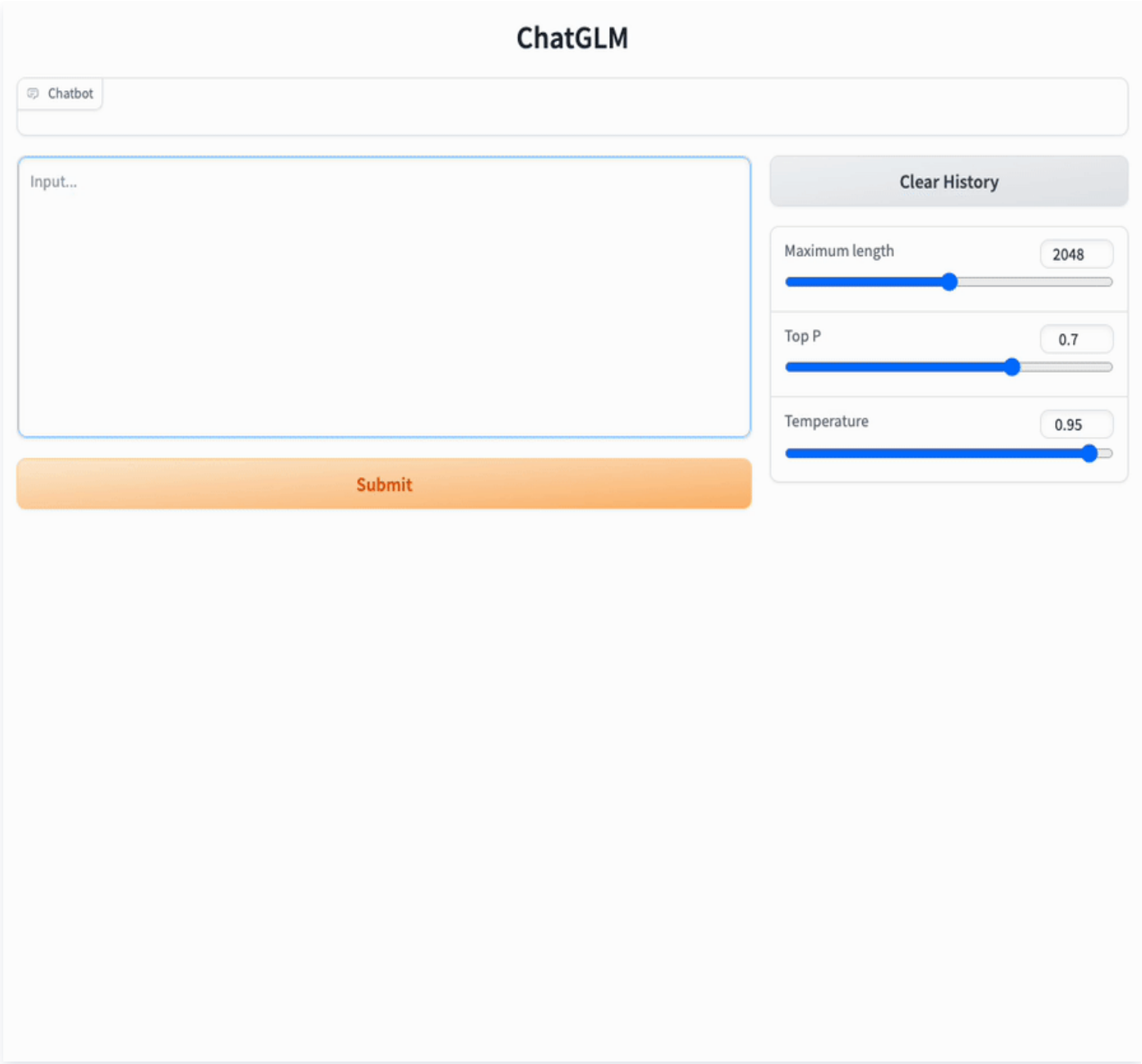
6. 您可以在此页面查看 ChatGLM2-6B 详细的配置信息。



步骤二：启动图形界面

1. 在实例列表中选择**算力连接** > **Gradio WebUI** 并进入该实例的详情页。

2. 使用高性能应用服务 HAI 部署的 **ChatGLM2-6B** 体验简单的对话。



参数说明

- **Maximum length 参数**

  通常用于限制输入序列的最大长度。

  - 因为 ChatGLM-6B 是2048长度推理的，一般这个保持默认就行，太大可能会导致性能下降。

- **Temperature 参数**

  用于控制模型输出的结果的随机性。

  - 设置为0，对每个 prompt 都生成固定的输出。
  - 较低的值，输出更集中，更有确定性。
  - 较高的值，输出更随机，更有创意。

- **Top-p 参数**

用于控制模型生成文本的概率分布。

- 较小的 Top-p 值会导致模型更加倾向于选择高频词汇。
- 而较大的 Top-p 值则会使模型更加注重选择低频词汇。
- 合适的 Top-p 值能够平衡生成文本的准确性和多样性。

快速使用 ChatGLM 对话模型 API 服务

最近更新时间：2024-04-15 15:56:41

背景介绍

腾讯云高性能应用服务 HAI 是为开发者量身打造的澎湃算力平台。无需复杂配置，便可享受即开即用的 GPU 云服务体验。在 HAI 中，根据应用智能匹配并推选出最适合的 GPU 算力资源，以确保您在数据科学、LLM、AI 作画等高性能应用中获得最佳性价比。

HAI 服务优势

- **智能选型：** 根据应用匹配推选 GPU 算力资源，实现最高性价比。同时，打通必备云服务组件，大幅简化云服务配置流程。
- **一键部署：** 分钟级自动构建 LLM、AI作画等应用环境。提供多种预装模型环境，包含如 StableDiffusion、ChatGLM2 等热门模型。
- **可视化界面：** 友好的图形界面，AI 调试更为简单。

场景介绍

本次我们使用 腾讯云高性能应用服务 HAI 体验快速搭建并使用 AI 模型 **ChatGLM2-6B**，实现思路如下:

- 体验 高性能应用服务 HAI 一键部署 **ChatGLM2-6B。具体步骤见** 快速使用 ChatGLM 对话模型应用 。
- 开发者体验 JupyterLab 、 Cloud Studio 进行 ChatGLM2-6B API 的配置调用。
- 开发者使用 Cloud Studio 应用推荐 ChatGPT Next Web 快速开发调用 ChatGLM2-6B OpenAI API 服务，搭建自己的 GPT。
- 开发者使用 高性能应用服务 HAI 快速部署 ChatGLM2-6B-int4 本地模型及基于 P-Tuning v2 的微调。

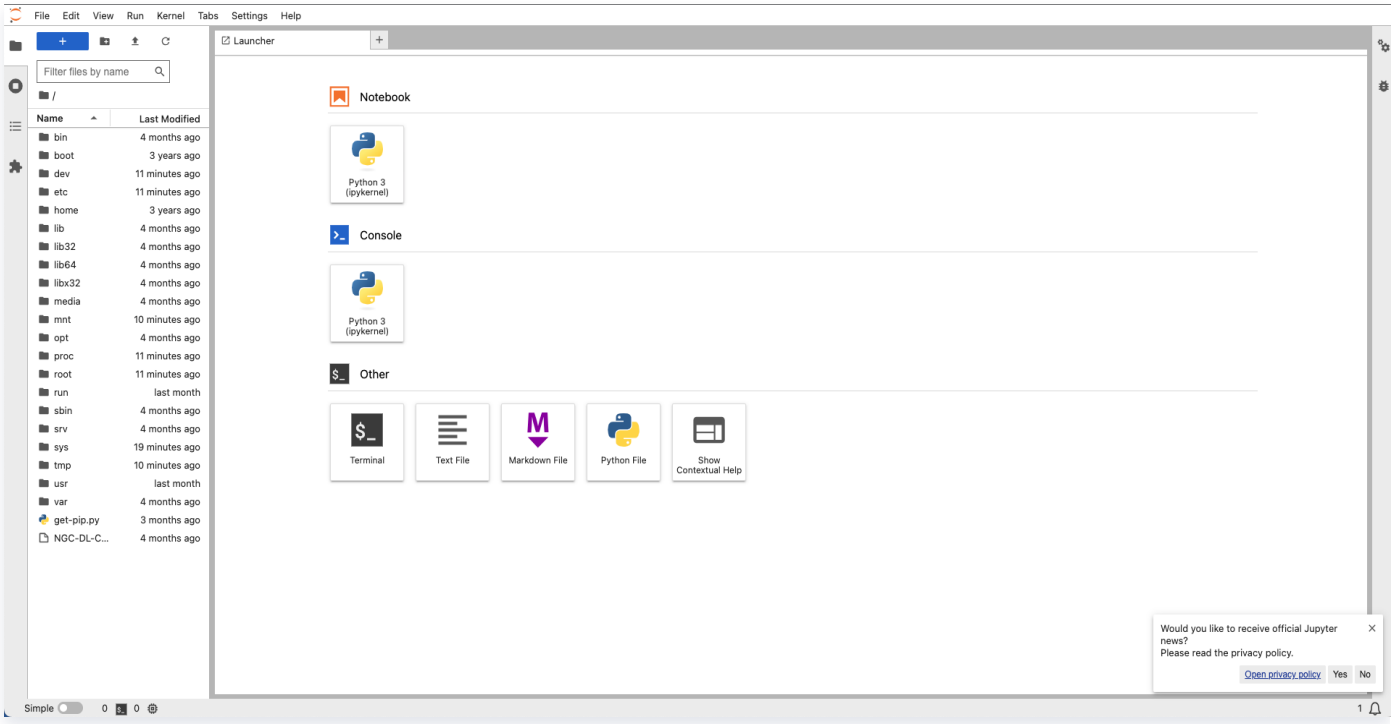
API 服务启用

使用 JupyterLab 启动

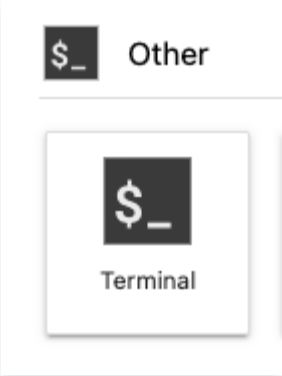
步骤一：进入 jupyter_lab 页面

1. 进入 jupyter_lab 控制台 操作界面。
2. 在实例列表中选择**更多** > **JupyterLab** 并进入该实例的详情页。

3. 初步认识并操作 JupyterLab。



4. 选择使用终端命令行操作。



> **注意：**
> 如果您购买使用的是 基础型算力服务器（0.88元/小时） 请您在开始实验前输入以下关闭 webui 功能的命令，提高服务器的性能，以便后续实验能快速正常进行：

输入代码：

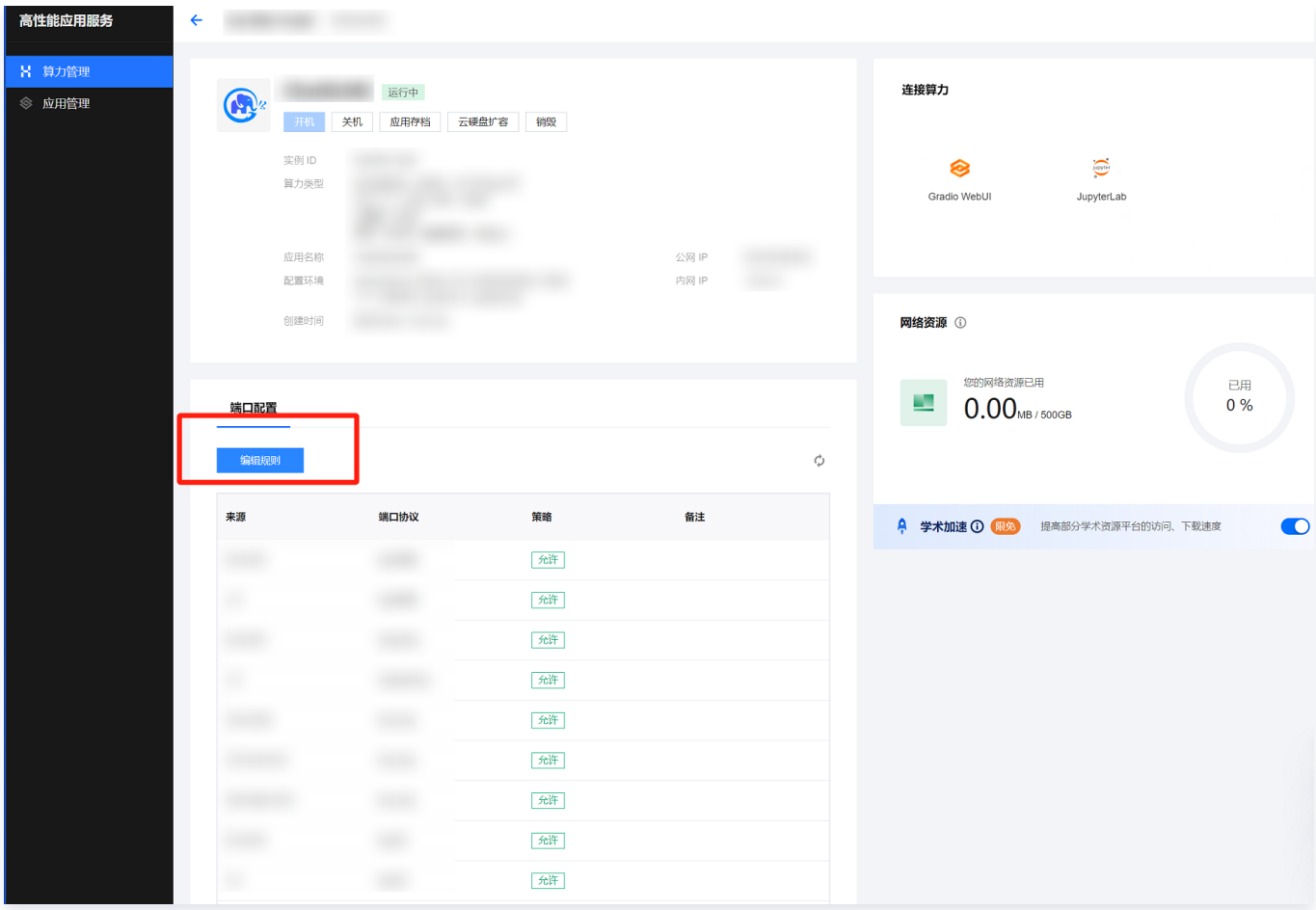
```
apt-get update && apt-get install sudo
sudo apt-get update
sudo apt-get install psmisc
sudo fuser -k 6889/tcp #执行这条命令将关闭 HAI提供的 chatglm2_gradio webui功能
```

输入命令 用于开启 API 服务：

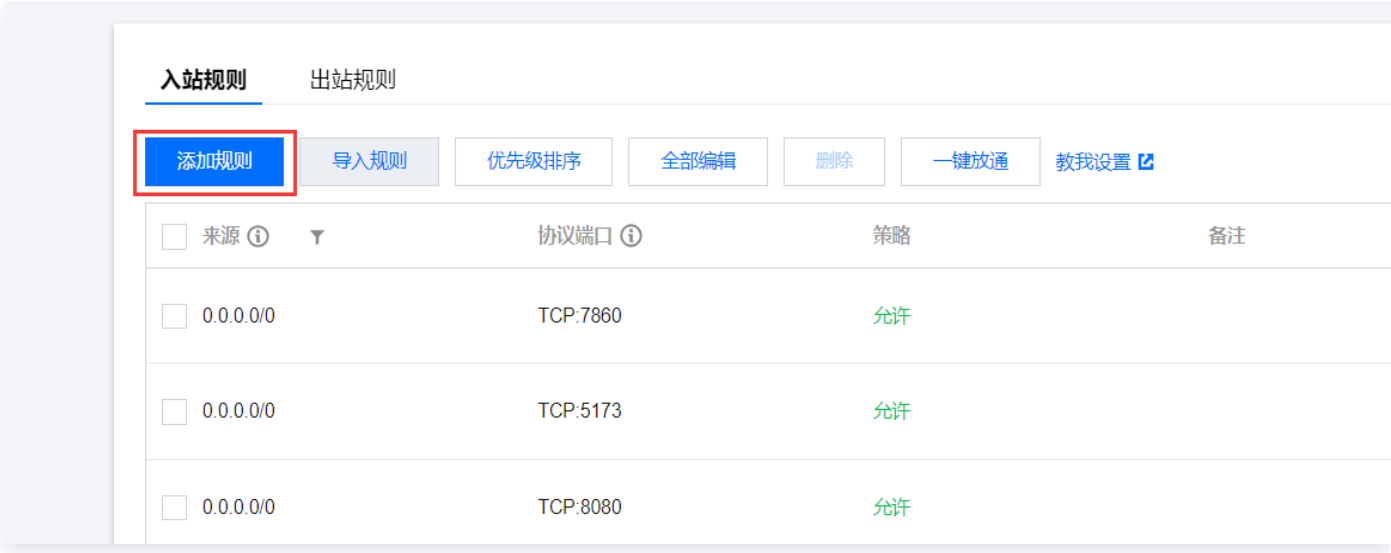
```
cd ./ChatGLM2-6B
python api.py
```

步骤二：新增服务器端口规则

1. 找到需要新增的实例，单击 ID/实例名，进入实例详情页面。
2. 在端口配置页面。单击编辑规则。



3. 在选择入站规则窗口中单击添加规则。

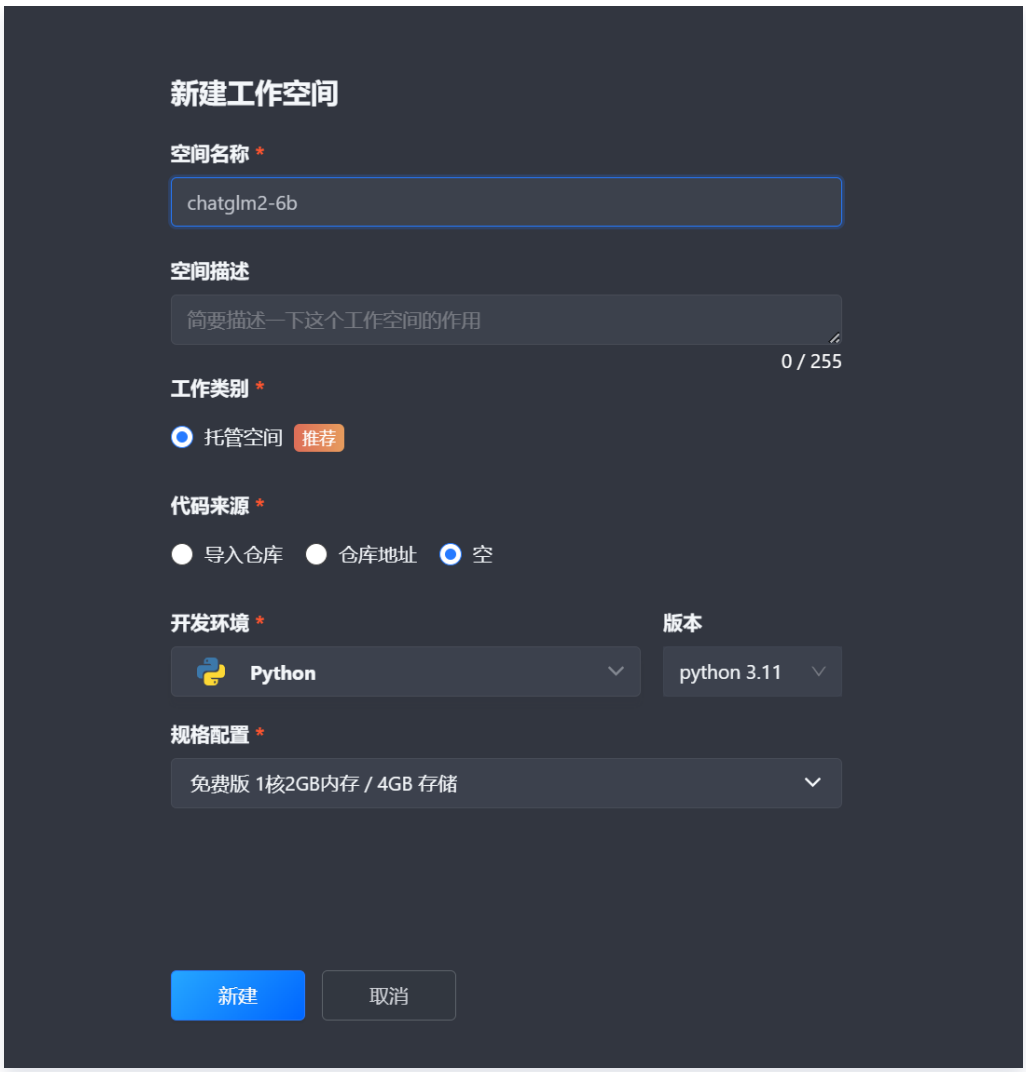


4. 添加入站规则 (来源： 0.0.0.0/0 协议端口： TCP:8000)

使用 Cloud Studio 启动

步骤一：打开 Cloud Studio 并创建开发空间

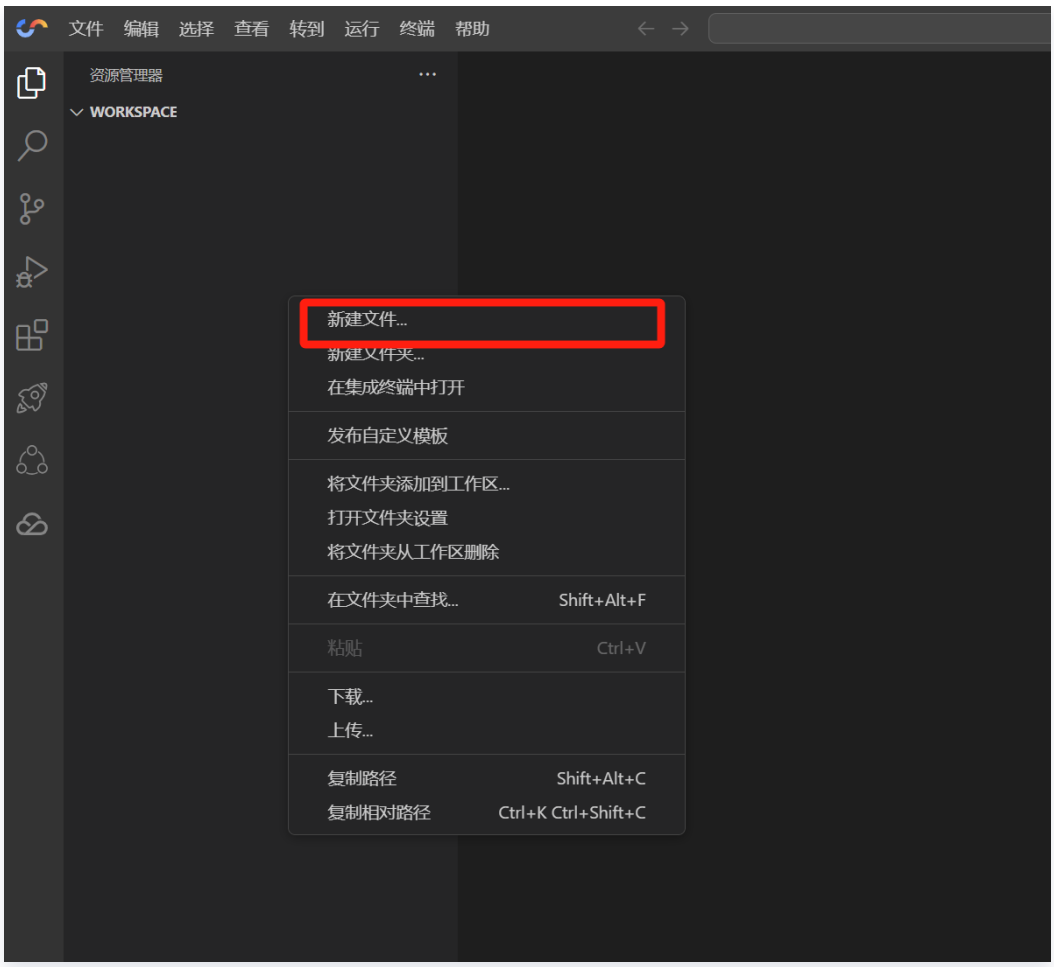
1. 进入 Cloud Studio 页面，单击立即使用。
2. 在 Cloud Studio 页面，单击右侧的新建模板。
3. 在新建工作空间弹窗中，输入**空间名称**，选择**代码来源**为空，开发环境为 Python 即可。



4. 单击新建。并完成工作空间创建。

步骤二：编写调用代码并运行测试

1. 在 workspace 下鼠标右键选择并选择新建文件。



2. 创建 `get_api.py` 代码文件，并复制以下代码用于测试请求。

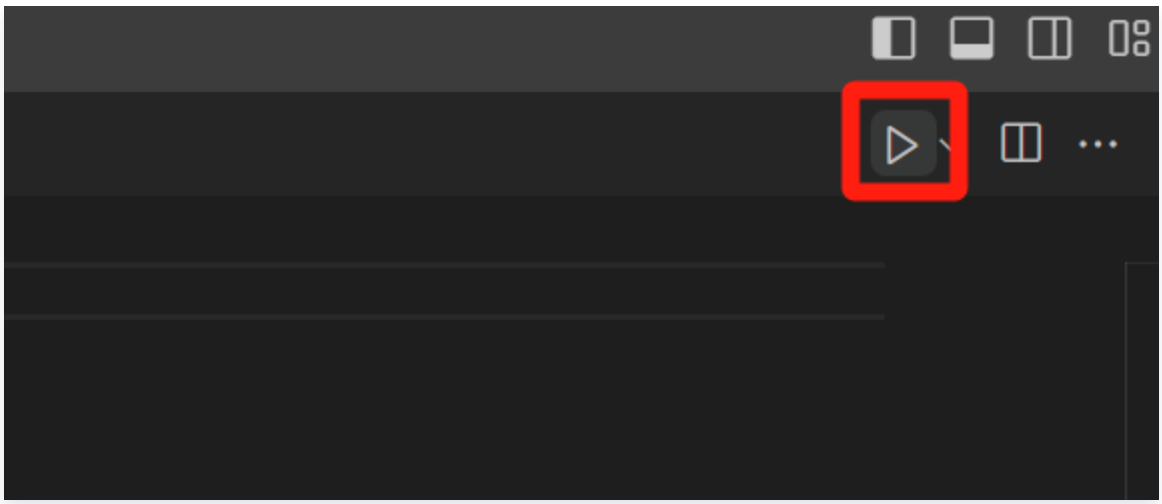
> **注意：**
> 请确保将代码中的地址和端口更改为您的 API 服务器的实际地址和端口。

```
import requests

# 定义测试数据，以及FastAPI服务器的地址和端口
server_url = "http://0.0.0.0:8000"  # 请确保将地址和端口更改为您的API服务器的实际地址和端口
test_data = {
    "prompt": "'你好，发热了怎么办？'",
    "history": [],
    "max_length": 50,
    "top_p": 0.7,
    "temperature": 0.95
}

# 发送HTTP POST请求
response = requests.post(server_url, json=test_data)
```

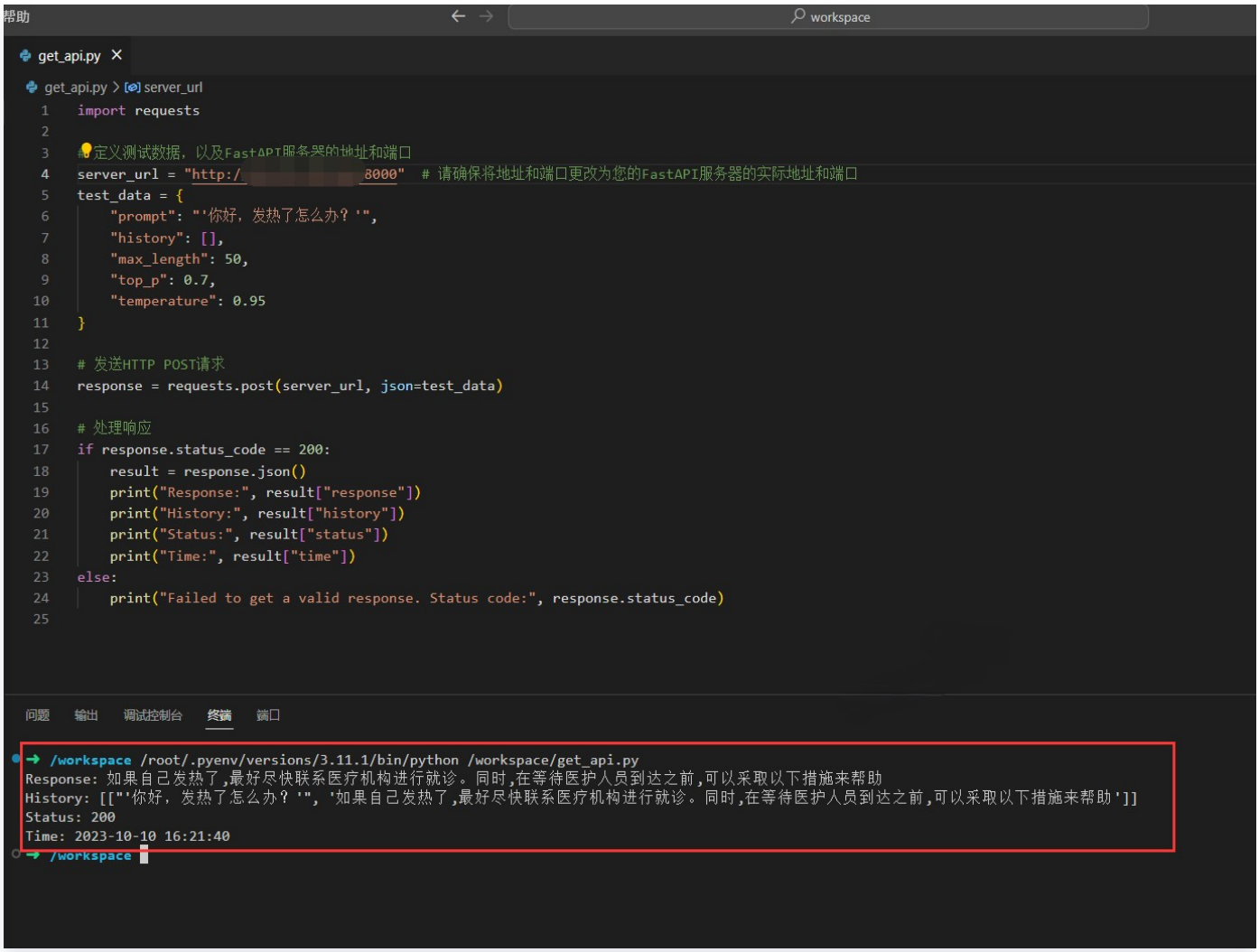
```
# 处理响应
if response.status_code == 200:
    result = response.json()
    print("Response:", result["response"])
    print("History:", result["history"])
    print("Status:", result["status"])
    print("Time:", result["time"])
else:
    print("Failed to get a valid response. Status code:", response.status_code)
```

3. 查看 ChatGLM2-6B 实例公网地址，并修改代码中服务器地址部分。

4. 修改完代码文件，可单击右上角的运行进行测试。



5. 返回请求结果如下图所示：

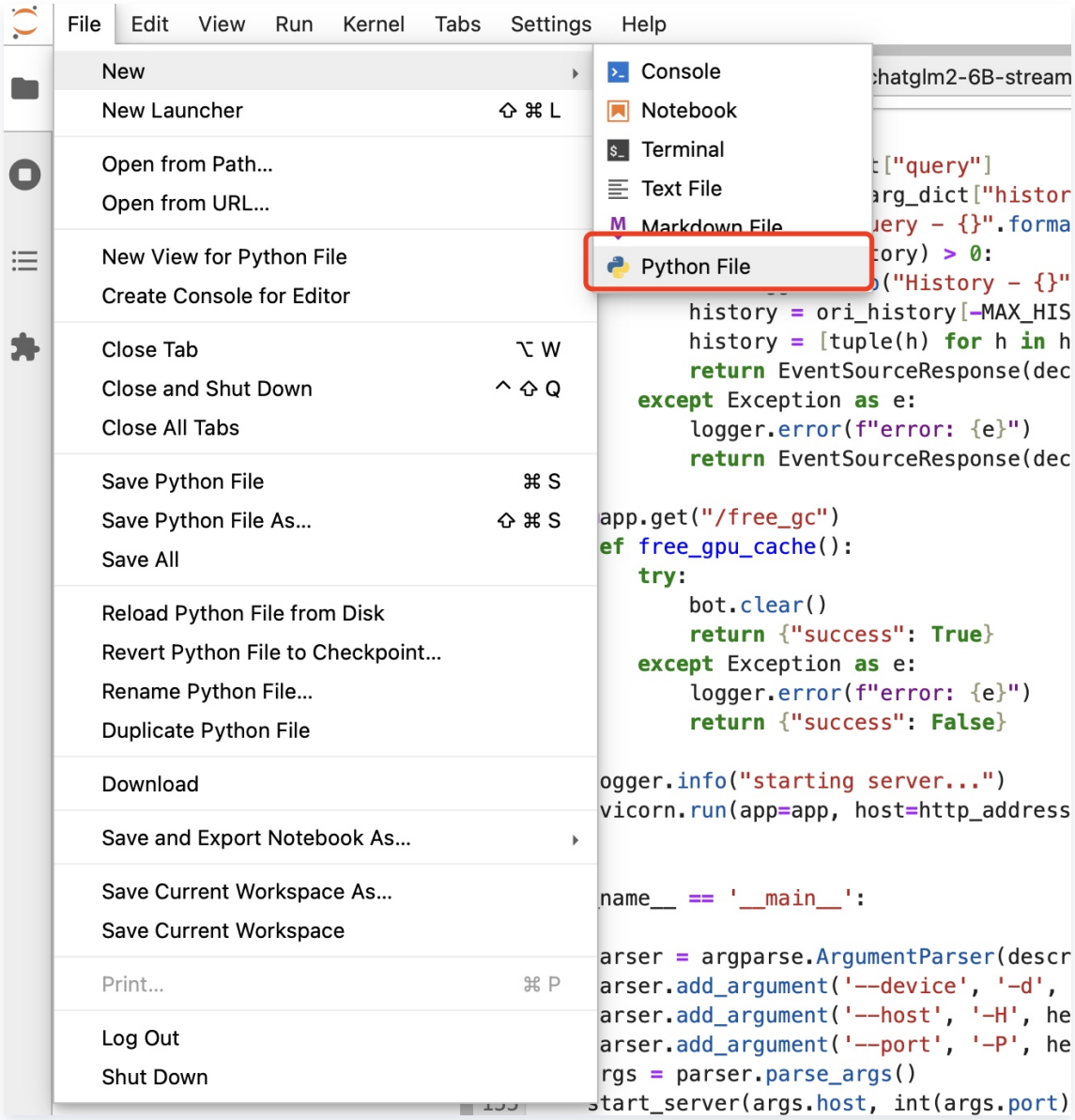


使用 JupyterLab 开发并使用 Cloud Studio 调用测试

步骤一：使用 JupyterLab 编写基于 FastAPI 编写的服务器端代码并开启服务

1. 在 JupyterLab 中选择文件夹操作界面，依次打开 root 文件夹下的 ChatGLM2-6B 文件夹，并创建一个 Python File ，拷贝一下代码并保存，同时将文件名修改为 chatglm2-6b-stream-api.py ，最后开启

API 服务。



2. 粘贴 chatglm2-6b-stream-api.py 代码。

```
# -*-coding:utf-8-*-
'''
File Name:chatglm2-6b-stream-api.py
Author:Luofan
Time:2023/6/26 13:33
'''

import os
import sys
import json
import torch
import uvicorn
```

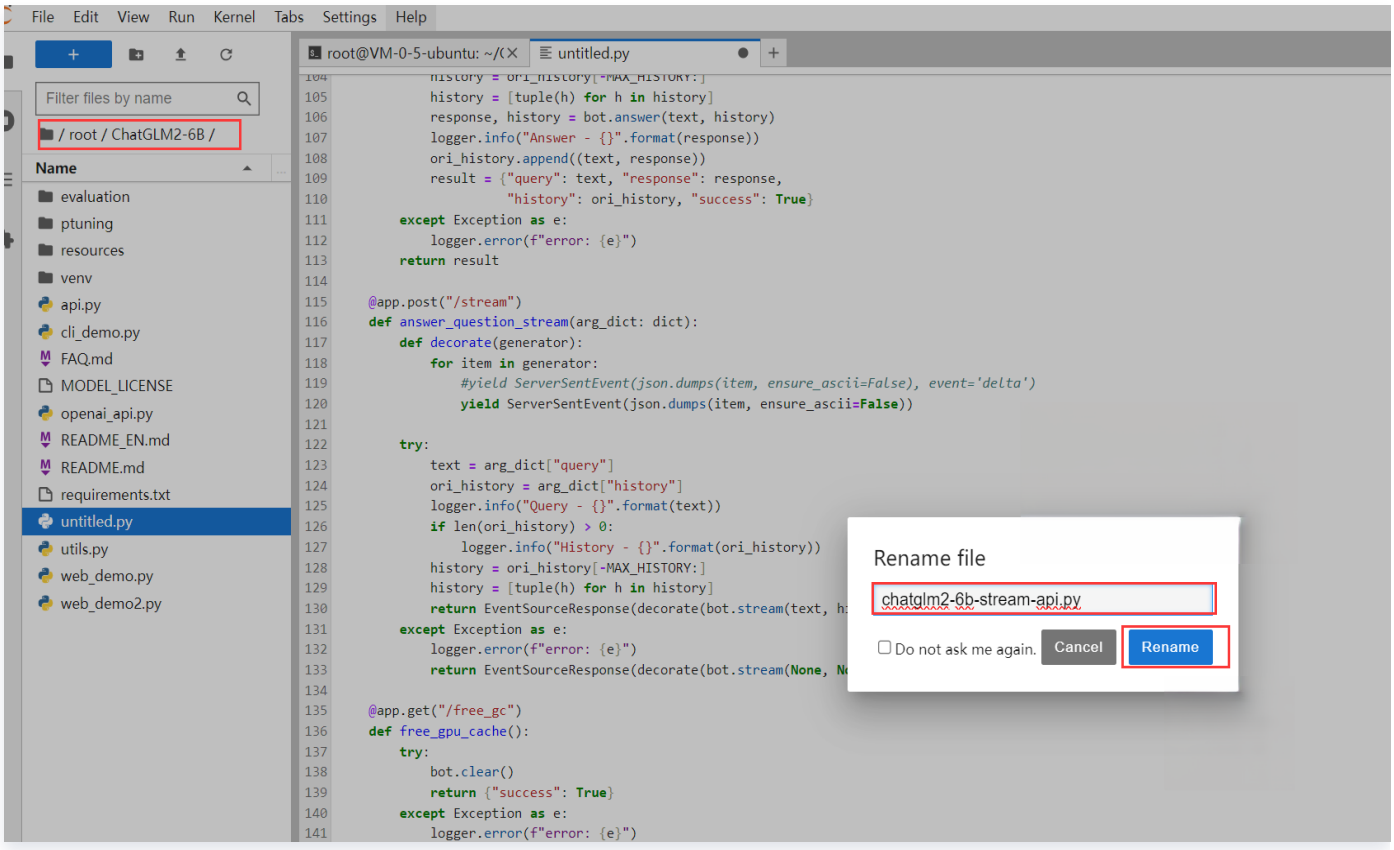
```
import logging
import argparse
from fastapi import FastAPI
from transformers import AutoTokenizer, AutoModel
from fastapi.middleware.cors import CORSMiddleware
from sse_starlette.sse import ServerSentEvent, EventSourceResponse


def getLogger(name, file_name, use_formatter=True):
    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)
    console_handler = logging.StreamHandler(sys.stdout)
    formatter = logging.Formatter('%(asctime)s    %(message)s')
    console_handler.setFormatter(formatter)
    console_handler.setLevel(logging.INFO)
    logger.addHandler(console_handler)
    if file_name:
        handler = logging.FileHandler(file_name, encoding='utf8')
        handler.setLevel(logging.INFO)
        if use_formatter:
            formatter = logging.Formatter('%(asctime)s - %(name)s - %(message)s')
            handler.setFormatter(formatter)
        logger.addHandler(handler)
    return logger


logger = getLogger('ChatGLM', 'chatlog.log')

MAX_HISTORY = 3


class ChatGLM():
    def __init__(self) -> None:
        logger.info("Start initialize model...")
        self.tokenizer =  AutoTokenizer.from_pretrained("THUDM/chatglm2-6b",
revision="v1.0", trust_remote_code=True)
        self.model = AutoModel.from_pretrained("THUDM/chatglm2-6b",
revision="v1.0", trust_remote_code=True).cuda()
        self.model.eval()
        logger.info("Model initialization finished.")

    def clear(self) -> None:
        if torch.cuda.is_available():
            with torch.cuda.device(f"cuda:{args.device}"):
                torch.cuda.empty_cache()
```

```
            torch.cuda.ipc_collect()

    def answer(self, query: str, history):
        response, history = self.model.chat(self.tokenizer, query, history=history)
        history = [list(h) for h in history]
        return response, history

    def stream(self, query, history):
        if query is None or history is None:
            yield {"query": "", "response": "", "history": [], "finished": True}
        size = 0
        response = ""
        for response, history in self.model.stream_chat(self.tokenizer, query, history):
            this_response = response[size:]
            history = [list(h) for h in history]
            size = len(response)
            yield {"delta": this_response, "response": response, "finished": False}
        logger.info("Answer - {}".format(response))
        yield {"query": query, "delta": "[EOS]", "response": response, "history":
history, "finished": True}


def start_server(http_address: str, port: int, gpu_id: str):
    os.environ['CUDA_DEVICE_ORDER'] = 'PCI_BUS_ID'
    os.environ['CUDA_VISIBLE_DEVICES'] = gpu_id

    bot = ChatGLM()

    app = FastAPI()
    app.add_middleware(CORSMiddleware,
                       allow_origins=["*"],
                       allow_credentials=True,
                       allow_methods=["*"],
                       allow_headers=["*"]
                       )

    @app.get("/")
    def index():
        return {'message': 'started', 'success': True}

    @app.post("/chat")
    async def answer_question(arg_dict: dict):
        result = {"query": "", "response": "", "success": False}
        try:
            text = arg_dict["query"]
```

```
            ori_history = arg_dict["history"]
            logger.info("Query - {}".format(text))
            if len(ori_history) > 0:
                logger.info("History - {}".format(ori_history))
            history = ori_history[-MAX_HISTORY:]
            history = [tuple(h) for h in history]
            response, history = bot.answer(text, history)
            logger.info("Answer - {}".format(response))
            ori_history.append((text, response))
            result = {"query": text, "response": response,
                      "history": ori_history, "success": True}
        except Exception as e:
            logger.error(f"error: {e}")
        return result

    @app.post("/stream")
    def answer_question_stream(arg_dict: dict):
        def decorate(generator):
            for item in generator:
                #yield ServerSentEvent(json.dumps(item, ensure_ascii=False),
event='delta')
                yield ServerSentEvent(json.dumps(item, ensure_ascii=False))

        try:
            text = arg_dict["query"]
            ori_history = arg_dict["history"]
            logger.info("Query - {}".format(text))
            if len(ori_history) > 0:
                logger.info("History - {}".format(ori_history))
            history = ori_history[-MAX_HISTORY:]
            history = [tuple(h) for h in history]
            return EventSourceResponse(decorate(bot.stream(text, history)))
        except Exception as e:
            logger.error(f"error: {e}")
            return EventSourceResponse(decorate(bot.stream(None, None)))

    @app.get("/free_gc")
    def free_gpu_cache():
        try:
            bot.clear()
            return {"success": True}
        except Exception as e:
            logger.error(f"error: {e}")
            return {"success": False}
```

```
    logger.info("starting server...")
    uvicorn.run(app=app, host=http_address, port=port, workers=1)


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Stream API Service for
ChatGLM2-6B')
    parser.add_argument('--device', '-d', help='device，-1 means cpu, other means
gpu ids', default='0')
    parser.add_argument('--host', '-H', help='host to listen', default='0.0.0.0')
    parser.add_argument('--port', '-P', help='port of this service', default=8000)
    args = parser.parse_args()
    start_server(args.host, int(args.port), args.device)
```

3. 在 **JupyterLab** 中完成文件的创建并重命名 **chatglm2-6b-stream-api.py** 成功：



4. 在 **JupyterLab 终端界面** 中输入命令开启 **chatglm2-6b-stream-api.py** 服务。

```
python chatglm2-6b-stream-api.py
```

> ⚠ **注意：**
>
> 请将上一个 API服务关闭，否则服务无法启动成功。

步骤二：使用 Cloud Studio 编写客户端代码

> **说明：**
> **使用普通 Http 请求调用 /chat 接口。**

1. 在 Cloud Studio 工作空间下继续创建 **Python** 代码文件 **use_chatglm2-6b-stream-api.py**。

> **注意：**
> **请将代码中的地址和端口更改为实际的服务器地址和端口。**

use_chatglm2-6b-stream-api.py 代码文件：

```
import requests
import json

# 设置服务器地址和端口
server_address = "http://0.0.0.0"  # 请将地址和端口更改为实际的服务器地址和端口
server_port = 8000

# 构造请求数据
request_data = {
    "query": "你好，发热了怎么办？",
    "history": []
}

# 发送HTTP POST请求到聊天端点
response = requests.post(f"{server_address}:{server_port}/chat",
json=request_data)
```

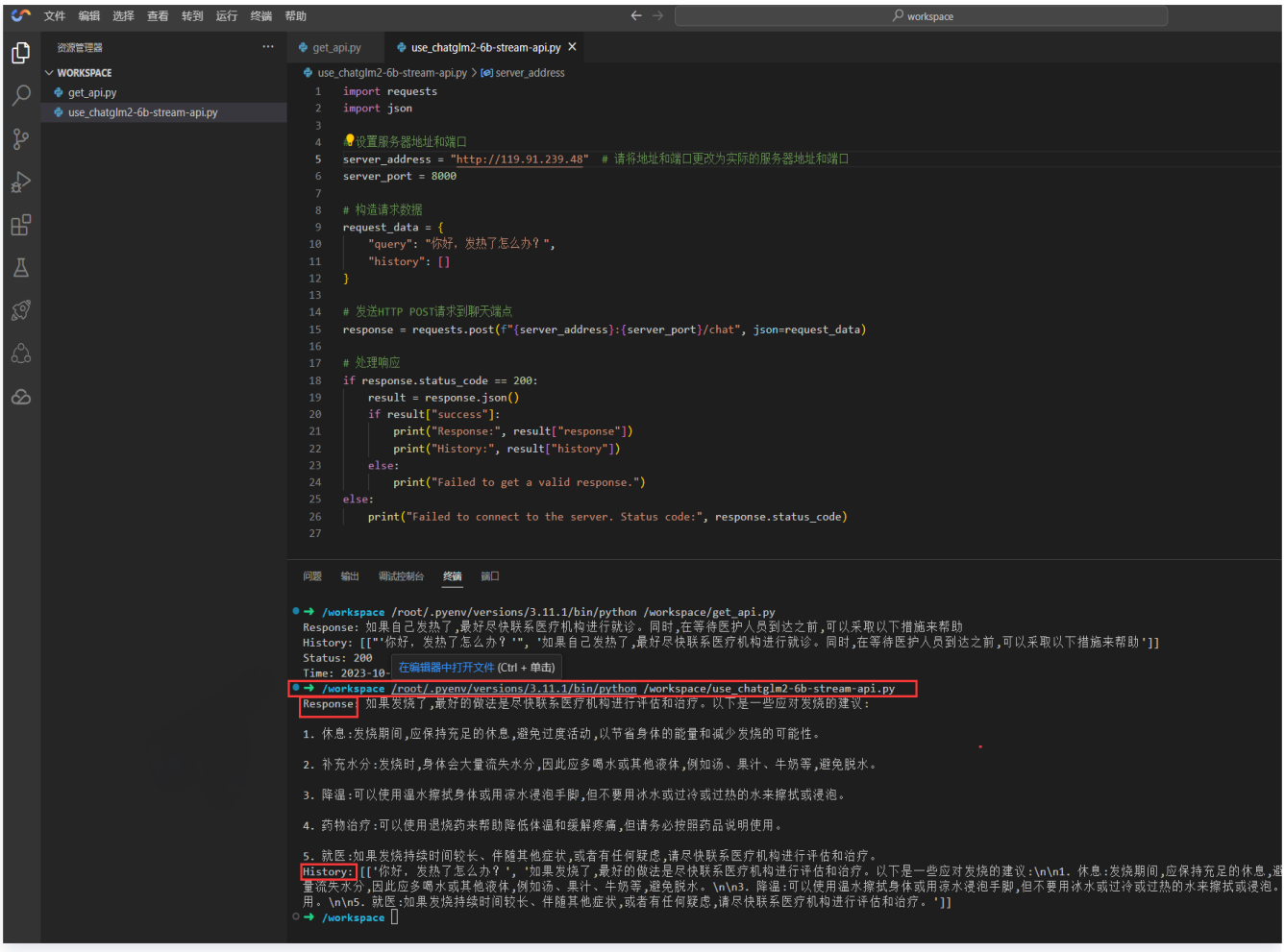
```
# 处理响应
if response.status_code == 200:
    result = response.json()
    if result["success"]:
        print("Response:", result["response"])
        print("History:", result["history"])
    else:
        print("Failed to get a valid response.")
else:
    print("Failed to connect to the server. Status code:", response.status_code)
```

创建完成后运行并查看返回结果：

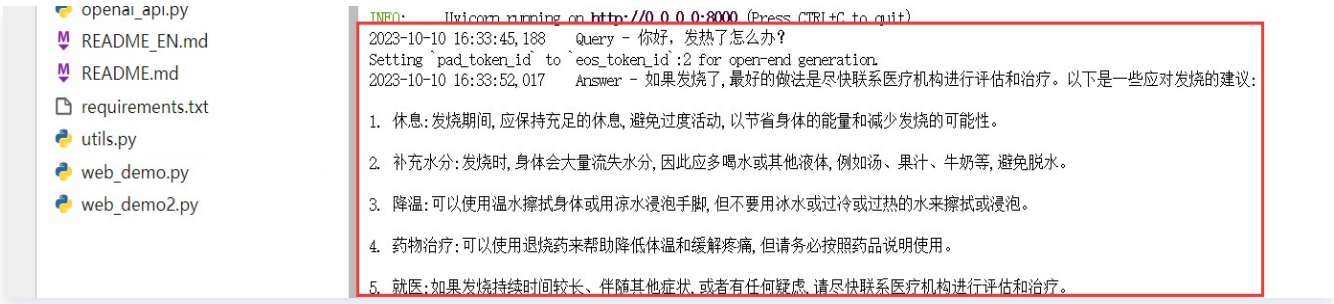
- **在菜单栏中点击 终端 选择 新建终端，输入命令执行代码。**

```
python use_chatglm2-6b-stream-api.py
```

- **调用接口成功。**



- **服务端查看记录：**



> **说明：**
> **使用 AioHttp 调用 /stream 流式接口。**

2. **在 Cloud Studio 工作空间下继续创建 Python 代码文件 use_stream_chatglm2-6b-stream-api.py。**

> **注意：**
> **请将代码中的地址和端口更改为实际的服务器地址和端口**

use_stream_chatglm2-6b-stream-api.py 代码文件：

```
import aiohttp
import json
import asyncio

async def main():
    async with aiohttp.ClientSession() as session:
        server_address = "http://0.0.0.0"  # 请将地址更改为实际的服务器地址
        server_port = 8000
        endpoint = f"{server_address}:{server_port}/stream"  # 流式处理端点

        request_data = {
            "query": "你好，发热怎么办？",
            "history": []
        }

        try:
            async with session.post(endpoint, json=request_data, timeout=None) as
response:
                if response.status == 200:
                    async for line in response.content.iter_any():
                        data = line.decode('utf-8')
                        data = data.replace('event: delta\r\ndata: ','')
                        data = data.replace('\r\n','')
                        data = data.replace('data: ','')
```

```
                try:
                    result = json.loads(data)
                    if result.get("finished"):
                        print("Response:", result["response"])
                        print('\r\n')
                        print("History:", result["history"])
                    #else:
                    #    print("Delta:", result["delta"])
                except json.JSONDecodeError:
                    print("Failed to parse JSON:")
            else:
                print(f"Failed to connect to the server. Status code:
{response.status}")
        except aiohttp.ClientError as e:
            print(f"Client error occurred: {e}")
            await asyncio.sleep(1)  # Avoid continuous failures, wait a bit before
retrying
            print(f"Unexpected error occurred: {e}")
            await asyncio.sleep(1)  # Avoid continuous failures, wait a bit before
retrying

if __name__ == '__main__':
    asyncio.run(main())
```

在终端中输入命令 安装 aiohttp 模块。

```
pip install aiohttp
```

运行并查看返回结果：
在终端中输入命令执行代码。

```
python use_stream_chatglm2-6b-stream-api.py
```

使用 ChatGPT Next Web 模板启动

> **说明：**
> **开发者使用 Cloud Studio 创建应用推荐 ChatGPT Next Web 开源项目 并快速开发调用 ChatGLM2-6B OpenAI API 服务。**

1. **使用 JupyterLab 修改 openai_api.py 代码并开启服务。**

> **注意：**

- 打开 **JupyterLab** 后关闭上一次开启的 API 服务，使用 **Ctrl+C** 关闭服务。
- 如果直接调用 **openai_api.py** 代码会报错，因此复制以下代码覆盖文件并保存文件。

```
# coding=utf-8
# Implements API for ChatGLM2-6B in OpenAI's format.
(https://platform.openai.com/docs/api-reference/chat)
# Usage: python openai_api.py
# Visit http://localhost:8000/docs for documents.

import time
import torch
import uvicorn
from pydantic import BaseModel, Field
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
from typing import Any, Dict, List, Literal, Optional, Union
from transformers import AutoTokenizer, AutoModel
from sse_starlette.sse import ServerSentEvent, EventSourceResponse


@asynccontextmanager
async def lifespan(app: FastAPI): # collects GPU memory
    yield
    if torch.cuda.is_available():
        torch.cuda.empty_cache()
        torch.cuda.ipc_collect()


app = FastAPI(lifespan=lifespan)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class ModelCard(BaseModel):
    id: str
    object: str = "model"
    created: int = Field(default_factory=lambda: int(time.time()))
    owned_by: str = "owner"
```

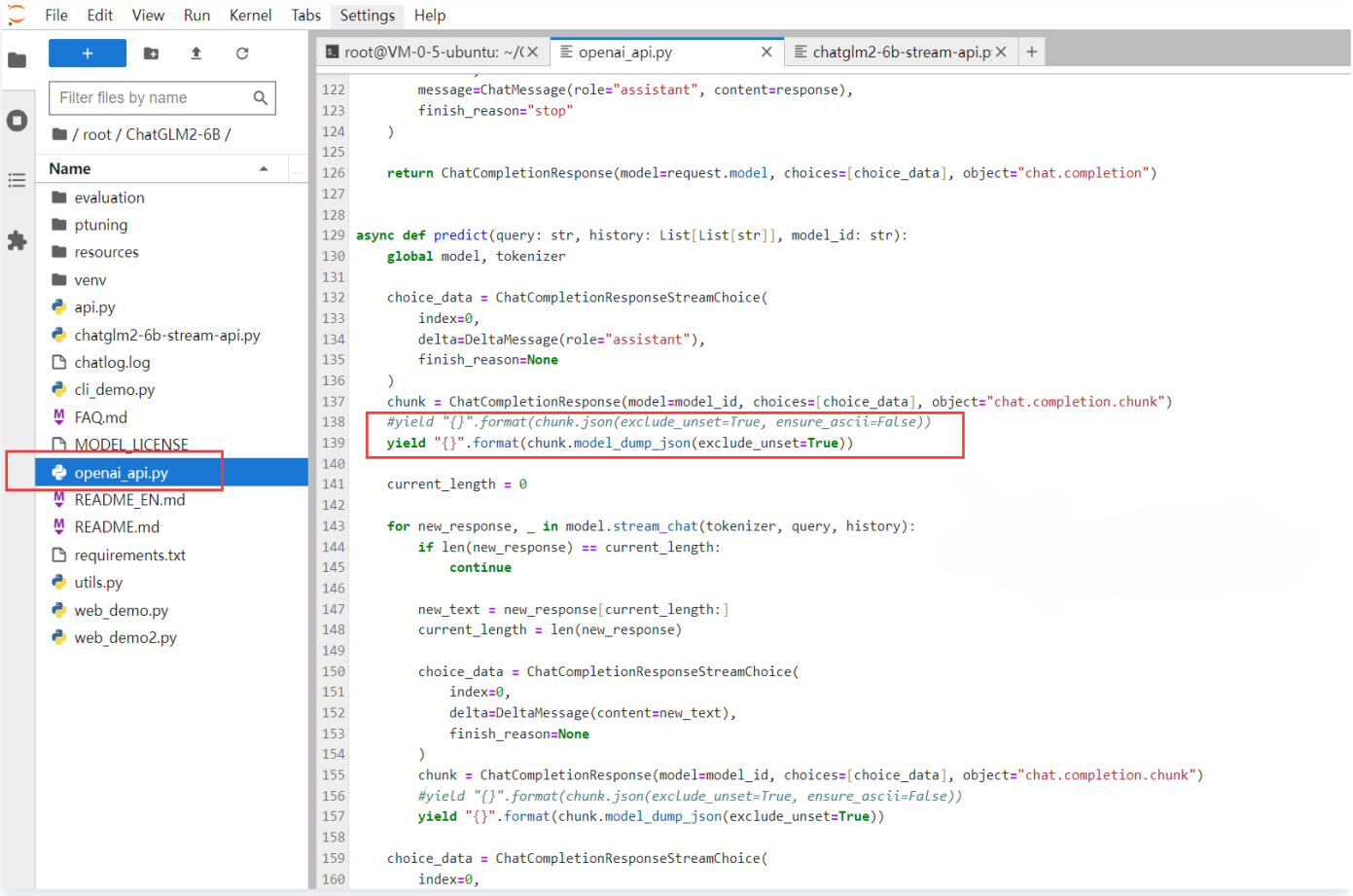
```
    root: Optional[str] = None
    parent: Optional[str] = None
    permission: Optional[list] = None


class ModelList(BaseModel):
    object: str = "list"
    data: List[ModelCard] = []


class ChatMessage(BaseModel):
    role: Literal["user", "assistant", "system"]
    content: str


class DeltaMessage(BaseModel):
    role: Optional[Literal["user", "assistant", "system"]] = None
    content: Optional[str] = None


class ChatCompletionRequest(BaseModel):
    model: str
    messages: List[ChatMessage]
    temperature: Optional[float] = None
    top_p: Optional[float] = None
    max_length: Optional[int] = None
    stream: Optional[bool] = False


class ChatCompletionResponseChoice(BaseModel):
    index: int
    message: ChatMessage
    finish_reason: Literal["stop", "length"]


class ChatCompletionResponseStreamChoice(BaseModel):
    index: int
    delta: DeltaMessage
    finish_reason: Optional[Literal["stop", "length"]]


class ChatCompletionResponse(BaseModel):
    model: str
    object: Literal["chat.completion", "chat.completion.chunk"]
```

```
    choices: List[Union[ChatCompletionResponseChoice,
ChatCompletionResponseStreamChoice]]
    created: Optional[int] = Field(default_factory=lambda: int(time.time()))


@app.get("/v1/models", response_model=ModelList)
async def list_models():
    global model_args
    model_card = ModelCard(id="gpt-3.5-turbo")
    return ModelList(data=[model_card])


@app.post("/v1/chat/completions", response_model=ChatCompletionResponse)
async def create_chat_completion(request: ChatCompletionRequest):
    global model, tokenizer

    if request.messages[-1].role != "user":
        raise HTTPException(status_code=400, detail="Invalid request")
    query = request.messages[-1].content

    prev_messages = request.messages[:-1]
    if len(prev_messages) > 0 and prev_messages[0].role == "system":
        query = prev_messages.pop(0).content + query

    history = []
    if len(prev_messages) % 2 == 0:
        for i in range(0, len(prev_messages), 2):
            if prev_messages[i].role == "user" and prev_messages[i+1].role ==
"assistant":
                history.append([prev_messages[i].content,
prev_messages[i+1].content])

    if request.stream:
        generate = predict(query, history, request.model)
        return EventSourceResponse(generate, media_type="text/event-stream")

    response, _ = model.chat(tokenizer, query, history=history)
    choice_data = ChatCompletionResponseChoice(
        index=0,
        message=ChatMessage(role="assistant", content=response),
        finish_reason="stop"
    )

    return ChatCompletionResponse(model=request.model, choices=[choice_data],
object="chat.completion")
```

```
async def predict(query: str, history: List[List[str]], model_id: str):
    global model, tokenizer

    choice_data = ChatCompletionResponseStreamChoice(
        index=0,
        delta=DeltaMessage(role="assistant"),
        finish_reason=None
    )
    chunk = ChatCompletionResponse(model=model_id, choices=[choice_data],
object="chat.completion.chunk")
    #yield "{}".format(chunk.json(exclude_unset=True, ensure_ascii=False))
    yield "{}".format(chunk.model_dump_json(exclude_unset=True))

    current_length = 0

    for new_response, _ in model.stream_chat(tokenizer, query, history):
        if len(new_response) == current_length:
            continue

        new_text = new_response[current_length:]
        current_length = len(new_response)

        choice_data = ChatCompletionResponseStreamChoice(
            index=0,
            delta=DeltaMessage(content=new_text),
            finish_reason=None
        )
        chunk = ChatCompletionResponse(model=model_id, choices=[choice_data],
object="chat.completion.chunk")
        #yield "{}".format(chunk.json(exclude_unset=True, ensure_ascii=False))
        yield "{}".format(chunk.model_dump_json(exclude_unset=True))

    choice_data = ChatCompletionResponseStreamChoice(
        index=0,
        delta=DeltaMessage(),
        finish_reason="stop"
    )
    chunk = ChatCompletionResponse(model=model_id, choices=[choice_data],
object="chat.completion.chunk")
    #yield "{}".format(chunk.json(exclude_unset=True, ensure_ascii=False))
    yield "{}".format(chunk.model_dump_json(exclude_unset=True))
    yield '[DONE]'
```

```
if __name__ == "__main__":
    tokenizer = AutoTokenizer.from_pretrained("THUDM/chatglm2-6b",
revision="v1.0", trust_remote_code=True)
    model = AutoModel.from_pretrained("THUDM/chatglm2-6b", revision="v1.0",
trust_remote_code=True).cuda()
    # 多显卡支持，使用下面两行代替上面一行，将num_gpus改为你实际的显卡数量
    # from utils import load_model_on_gpus
    # model = load_model_on_gpus("THUDM/chatglm2-6b", num_gpus=2)
    model.eval()

    uvicorn.run(app, host='0.0.0.0', port=8000, workers=1)
```

修改后的 openai_api.py 示意图：



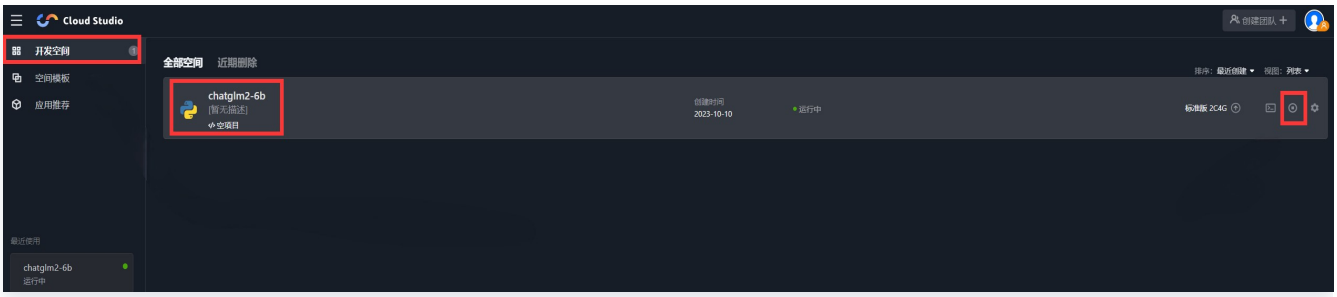
服务端开启服务：

```
python openai_api.py
```

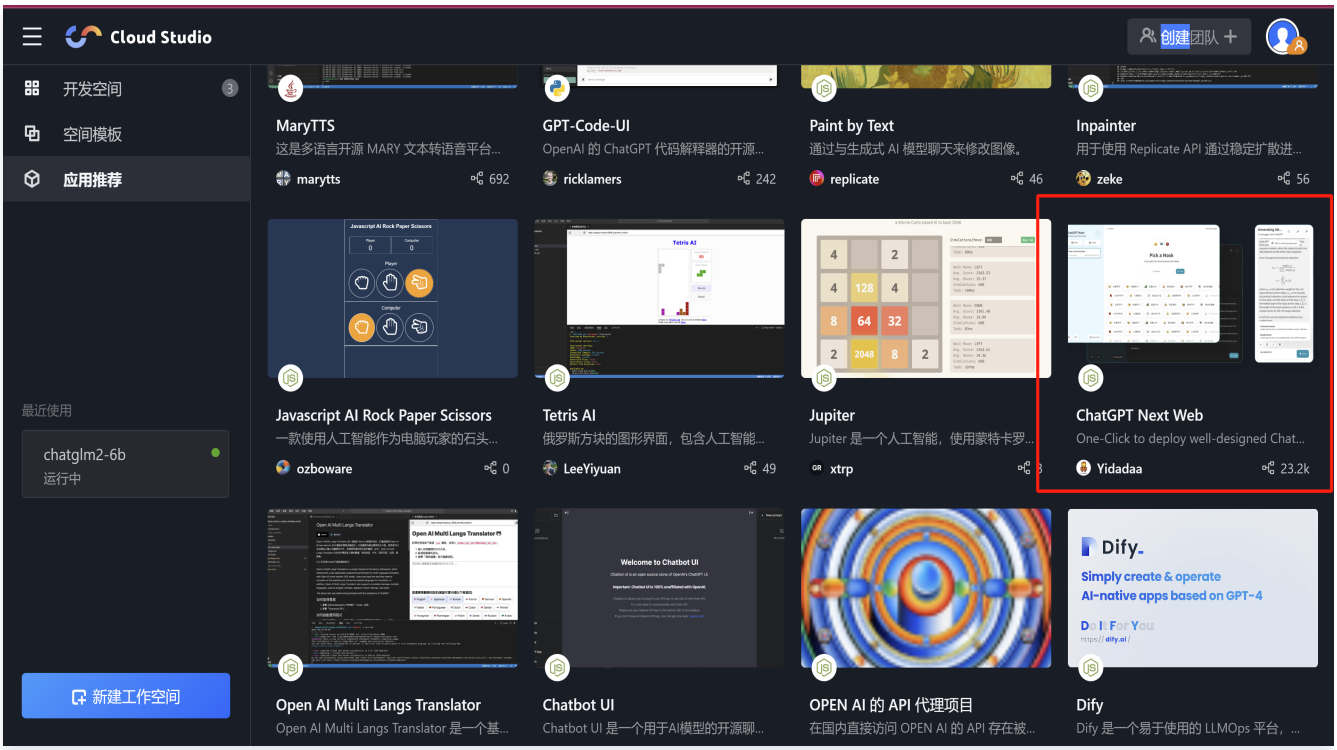
```
(base) root@VM-0-5-ubuntu:~/ChatGLM2-6B# python openai_api.py
Loading checkpoint shards: 100%|
INFO:     Started server process [684]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

2. 使用 Cloud Studio 快速创建 应用推荐 下的 ChatGPT Next Web 开源项目。

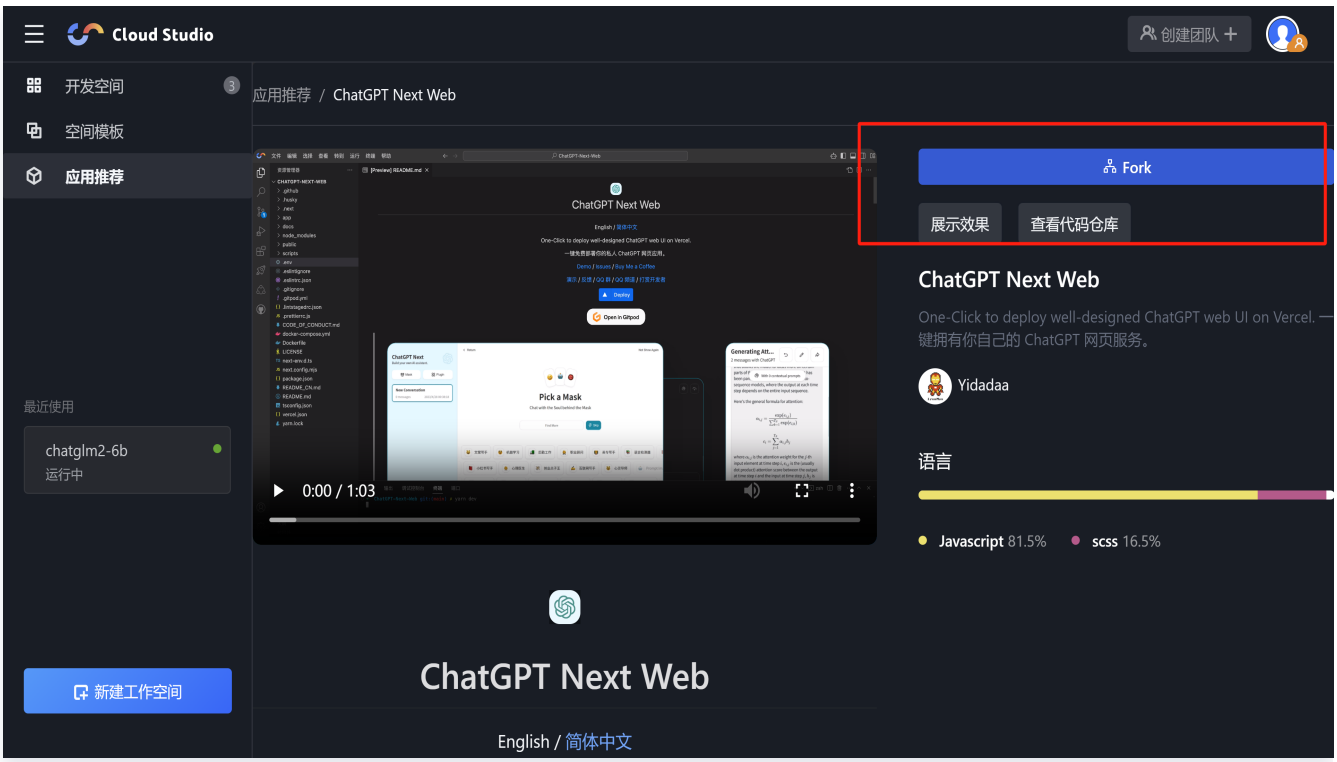
2.1 打开 Cloud Studio 开发空间下，我们创建的项目，并停止服务。



2.2 成功后，打开 应用推荐 选择 ChatGPT Next Web 项目。



选择 Fork。



3. 使用 Cloud Studio 快速配置并启动项目。

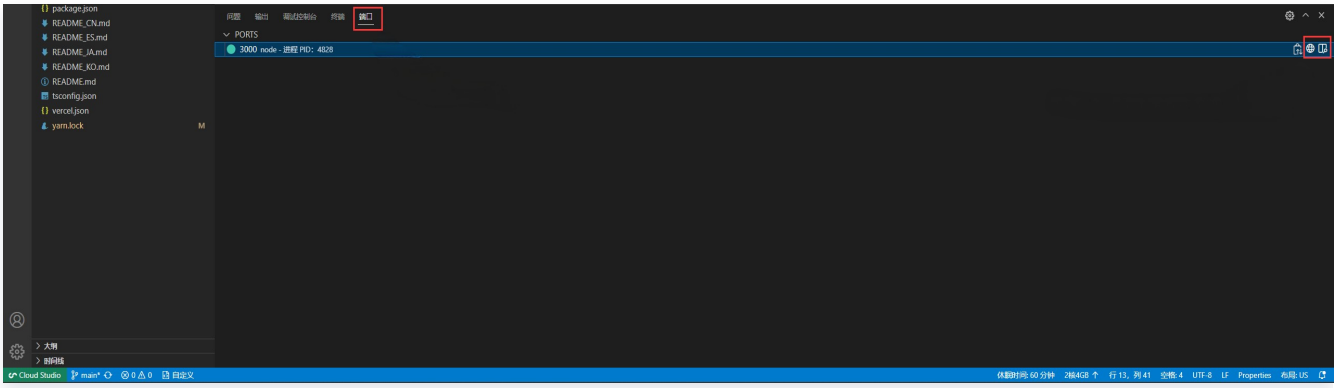
3.1 Fork 完成后，选择 **.env.template** 文件，修改 **OPENAI_API_KEY** 为非空字符串，并配置 API 地址和端口，然后在终端输入命令：

```
npm install
```

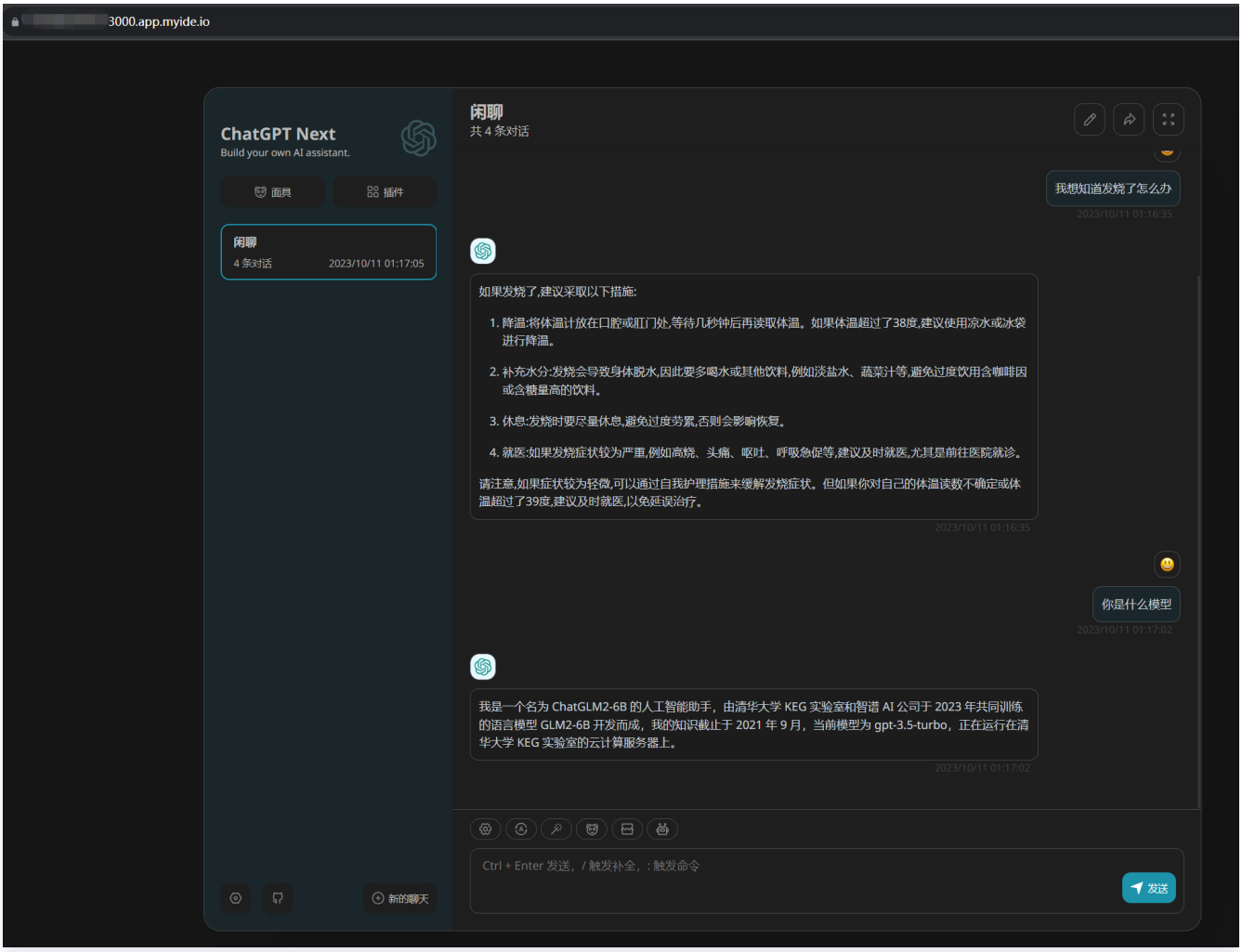
3.2 依赖安装完成后，输入命令开启服务。

```
yarn dev
```

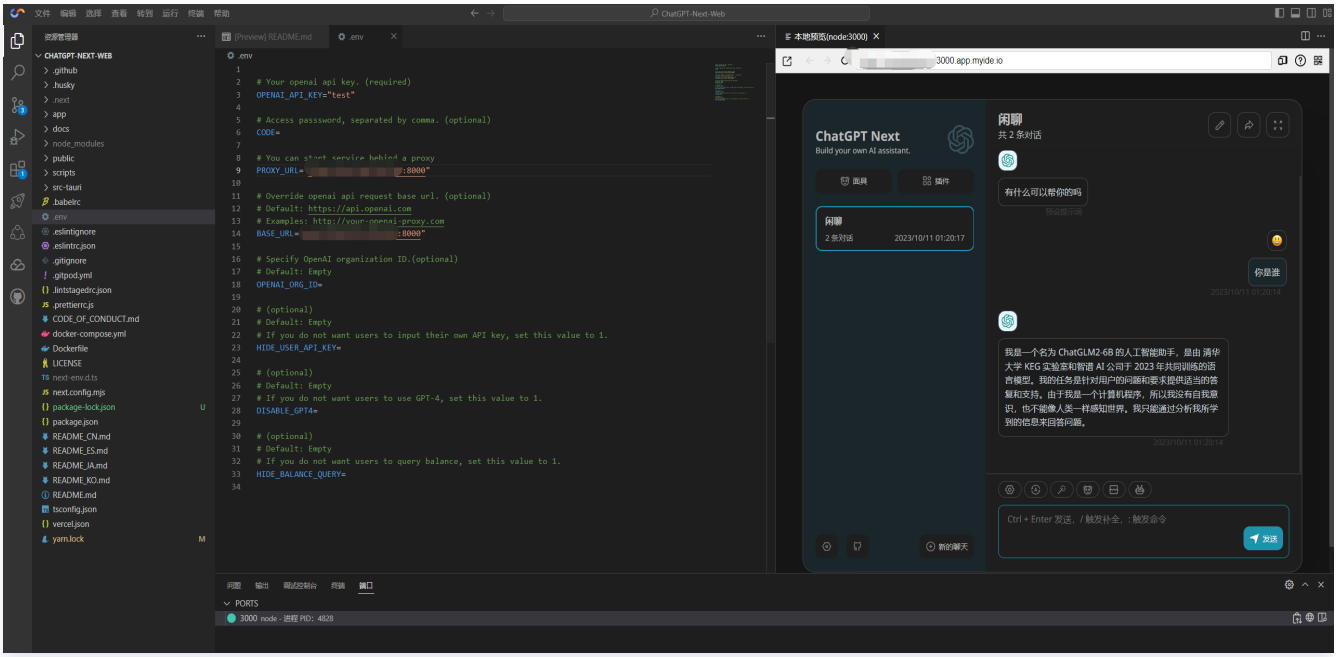
3.3 单击**端口**，可使用 **浏览器或标签页** 两种方式运行项目。



web浏览器测试：



Cloud Studio 标签页查看：



服务端可查看相关的请求记录：

```
(base) root@VM-0-5-ubuntu:~/ChatGLM2-6B# python openai_api.py
Loading checkpoint shards: 100%|
INFO:     Started server process [684]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     141 - "POST /v1/chat/completions?path=v1&path=chat&path=completions HTTP/1.1" 200 OK
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
INFO:     56141 - "POST /v1/chat/completions?path=v1&path=chat&path=completions HTTP/1.1" 200 OK
INFO:     :61669 - "POST /v1/chat/completions?path=v1&path=chat&path=completions HTTP/1.1" 200 OK
INFO:     :2291 - "POST /v1/chat/completions?path=v1&path=chat&path=completions HTTP/1.1" 200 OK
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
INFO:     2291 - "POST /v1/chat/completions?path=v1&path=chat&path=completions HTTP/1.1" 200 OK
```

批量导出算力连接方式

最近更新时间：2024-05-10 16:40:52

面向场景

针对**高校课程、教培、企业内部使用**等需要批量创建多台算力，批量导出算力连接方式的场景。

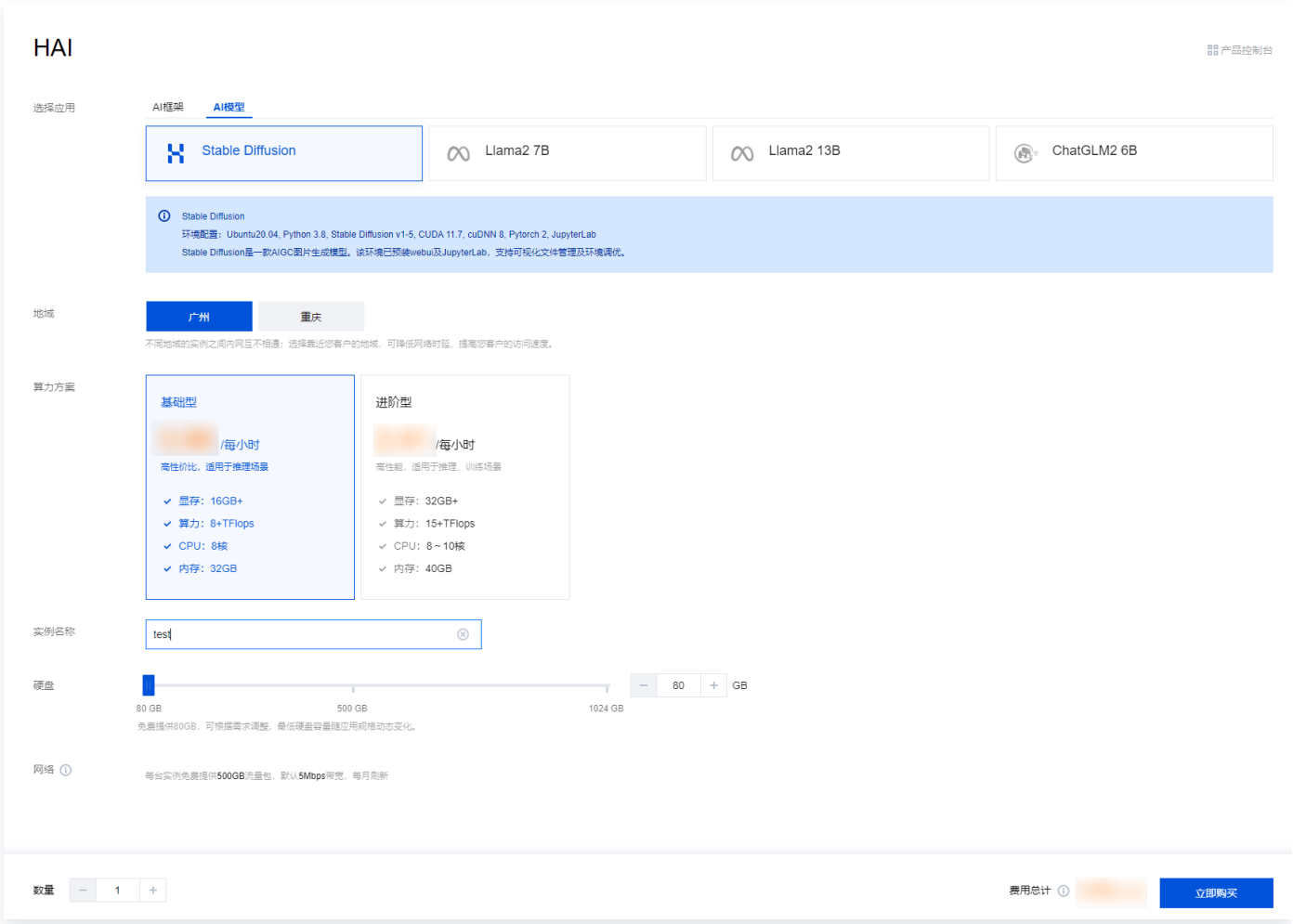
实现效果

支持批量创建多台算力，并将算力连接方式批量导出。导出后您可将算力分发给内部用户使用。

操作步骤

步骤1：创建高性能应用服务

1. 登录 高性能应用服务控制台 。
2. 单击**新建**，进入 高性能应用服务购买页面 。



- **选择应用**：目前提供 AI 框架、AI 模型两类应用，请根据实际需求进行选择。
- **地域**：建议选择靠近目标客户的地域，降低网络延迟、提高您的客户的访问速度。

- **算力方案：**支持基础型及进阶型两类算力方案，本次算力方案选择进阶型，生成图片效率更高。
- **实例名称：**自定义实例名称，若不填则默认使用实例 ID 替代。
- **硬盘：**默认提供 80GB 免费空间，可根据实际使用需求进行调整。
- **网络：**每台实例每月免费提供 500GB 流量包，默认 10Mbps 带宽，每月刷新。
- **购买数量：**默认1台。您可在此处选择多台，目前支持一次性创建10台。

3. 单击**立即购买**。
4. 核对配置信息后，单击**提交订单**，并根据页面提示完成支付。
5. 等待创建完成。单击实例**任意位置**并进入该实例的详情页。

步骤2：批量导出算力连接方式

目前暂不支持通过控制台批量导出算力连接方式。您可 提交工单 联系工作人员，我们会给您提供导出脚本及对应的使用指引。

步骤 3：分发算力连接方式

在完成 步骤 2 后，您将获得一个包含您账号下所有实例连接方式的 Excel 文件。