搭建langchain+LLM框架，部署知识库体系总结

1、启动服务器环境

本次启动服务器环境为AutoDL平台，V100-32GB * 1卡，镜像 PyTorch 2.0.0、Python 3.8(ubuntu20.04)、Cuda 11.8

2、安装项目代码和项目库

在启动jupyter-lab环境中启动Terminal后输入以下内容，首先更新apt-get，防止apt-get命令找不到git-lfs

```
apt-get update
```

apt-get下载git-lfs

```
apt-get install git-lfs
```

然后git clone拉取项目代码，本次测试使用的LLM模型为Qwen-1_8B-Chat，因此首先需要拉取Qwen的git官方代码

```
git clone https://github.com/QwenLM/Qwen.git
```

拉取到Qwen的模型代码后，再拉取Langchain知识库的项目，本次使用的是支持可视化导入的Langchain-ChatChat

```
git clone https://github.com/chatchat-space/Langchain-Chatchat.git
```

3、安装LLM模型和向量化模型

本次测试使用的LLM是Qwen-1.8B-chat，知识向量化模型使用m3e-base，可根据自身需要进行选择，由于国内使用git 可能无法直接拉取hugging-face的模型文件，因此选择安装另外一个hug镜像库去拉取模型文件，代码如下

```
pip install -U huggingface_hub
```

安装后，直接新建ipynb文件，输入以下内容，开始拉取Qwen-1.8B-chat和m3e-base的模型文件

```
import os
your_path = '/mnt/workspace/Qwen-1.8b'

# 设置环境变量
os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'

# 下载模型
os.system('huggingface-cli download --resume-download Qwen/Qwen-1_8B-Chat --local-dir Qwen-1_8B_Chat')
```

```
import os
your_path = '/mnt/workspace/m3e-base'

# 设置环境变量
os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'

# 下载模型
os.system('huggingface-cli download --resume-download moka-ai/m3e-base --local-dir m3e-base')
```

4、安装python虚拟环境

经过多次踩坑和测试，安装如Langchain等项目文件的需求算子库时最好构建python的虚拟环境，在Terminal中输入代码如下

```
conda create -n lyx python=3.8
```

安装完虚拟环境后，初始化并启动

```
conda init
```

```
conda activate lyx
```

5、安装项目需求算子库

在虚拟环境下，先cd到LangChain-Chatchat的项目文件下，然后依次运行以下代码

```
pip install -r requirements.txt
pip install -r  requirements_api.txt
pip install -r  requirements_webui.txt
```

6、更改项目配置文件

安装完项目所需的算子库，需要更改项目中的config文件，位置在Langchain-Chatchat/config下，配置方法和示例文档基本一致，主要将加载路径进行改变，需要依次对config目录下的5个文件进行修改

复制模型相关参数配置模板文件 configs/model_config.py.example 存储至项目路径下 `./configs` 路径下，并重命名为 `model_config.py`，本次测试使用的模型是Qwen-1.8b-chat，知识向量化使用m3e模型，则配置文件举例如下：

```
import os

# 可以指定一个绝对路径，统一存放所有的Embedding和LLM模型。
# 每个模型可以是一个单独的目录，也可以是某个目录下的二级子目录。
# 如果模型目录名称和 MODEL_PATH 中的 key 或 value 相同，程序会自动检测加载，无需修改 MODEL_PATH 中的路径。
MODEL_ROOT_PATH = ""

# 选用的 Embedding 名称
```

```
EMBEDDING_MODEL = "m3e-base"

# Embedding 模型运行设备。设为"auto"会自动检测，也可手动设定为"cuda","mps","cpu"其中之一。
EMBEDDING_DEVICE = "auto"

# 如果需要在 EMBEDDING_MODEL 中增加自定义的关键字时配置
EMBEDDING_KEYWORD_FILE = "keywords.txt"
EMBEDDING_MODEL_OUTPUT_PATH = "output"

# 要运行的 LLM 名称，可以包括本地模型和在线模型。列表中本地模型将在启动项目时全部加载。
# 列表中第一个模型将作为 API 和 WEBUI 的默认模型。
# 在这里，我们使用目前主流的两个离线模型，其中，chatglm3-6b 为默认加载模型。
# 如果你的显存不足，可使用 Qwen-1_8B-Chat, 该模型 FP16 仅需 3.8G显存。
LLM_MODELS = ["Qwen-1_8B-Chat"] # "Qwen-1_8B-Chat",

# AgentLM模型的名称 (可以不指定，指定之后就锁定进入Agent之后的Chain的模型，不指定就是LLM_MODELS[0])
Agent_MODEL = None

# LLM 运行设备。设为"auto"会自动检测，也可手动设定为"cuda","mps","cpu"其中之一。
LLM_DEVICE = "auto"

# 历史对话轮数
HISTORY_LEN = 3

# 大模型最长支持的长度，如果不填写，则使用模型默认的最大长度，如果填写，则为用户设定的最大长度
MAX_TOKENS = None

# LLM通用对话参数
TEMPERATURE = 0.7
# TOP_P = 0.95 # ChatOpenAI暂不支持该参数

ONLINE_LLM_MODEL = {
    # 线上模型。请在server_config中为每个在线API设置不同的端口

    "openai-api": {
        "model_name": "gpt-3.5-turbo",
        "api_base_url": "https://api.openai.com/v1",
        "api_key": "",
        "openai_proxy": "",
    },

    # 具体注册及api key获取请前往 http://open.bigmodel.cn
    "zhipu-api": {
        "api_key": "",
        "version": "chatglm_turbo",  # 可选包括 "chatglm_turbo"
        "provider": "ChatGLMWorker",
    },


    # 具体注册及api key获取请前往 https://api.minimax.chat/
    "minimax-api": {
        "group_id": "",
        "api_key": "",
```

```
        "is_pro": False,
        "provider": "MiniMaxWorker",
    },


    # 具体注册及api key获取请前往 https://xinghuo.xfyun.cn/
    "xinghuo-api": {
        "APPID": "",
        "APISecret": "",
        "api_key": "",
        "version": "v1.5",  # 你使用的讯飞星火大模型版本，可选包括 "v3.0", "v1.5", "v2.0"
        "provider": "XingHuoWorker",
    },

    # 百度千帆 API，申请方式请参考 https://cloud.baidu.com/doc/WENXINWORKSHOP/s/4lilb2lpf
    "qianfan-api": {
        "version": "ERNIE-Bot",  # 注意大小写。当前支持 "ERNIE-Bot" 或 "ERNIE-Bot-turbo"， 更多的见官方文档。
        "version_url": "",  # 也可以不填写version，直接填写在千帆申请模型发布的API地址
        "api_key": "",
        "secret_key": "",
        "provider": "QianFanWorker",
    },

    # 火山方舟 API，文档参考 https://www.volcengine.com/docs/82379
    "fangzhou-api": {
        "version": "chatglm-6b-model",  # 当前支持 "chatglm-6b-model"， 更多的见文档模型支持列表中方舟部分。
        "version_url": "",  # 可以不填写version，直接填写在方舟申请模型发布的API地址
        "api_key": "",
        "secret_key": "",
        "provider": "FangZhouWorker",
    },

    # 阿里云通义千问 API，文档参考 https://help.aliyun.com/zh/dashscope/developer-reference/api-details
    "qwen-api": {
        "version": "qwen-turbo",  # 可选包括 "qwen-turbo", "qwen-plus"
        "api_key": "",  # 请在阿里云控制台模型服务灵积API-KEY管理页面创建
        "provider": "QwenWorker",
    },

    # 百川 API，申请方式请参考 https://www.baichuan-ai.com/home#api-enter
    "baichuan-api": {
        "version": "Baichuan2-53B",  # 当前支持 "Baichuan2-53B"， 见官方文档。
        "api_key": "",
        "secret_key": "",
        "provider": "BaiChuanWorker",
    },

    # Azure API
    "azure-api": {
        "deployment_name": "",  # 部署容器的名字
```

```
        "resource_name": "",  # https://{resource_name}.openai.azure.com/openai/ 填写
resource_name的部分，其他部分不要填写
        "api_version": "",  # API的版本，不是模型版本
        "api_key": "",
        "provider": "AzureWorker",
    },

    # 昆仑万维天工 API https://model-platform.tiangong.cn/
    "tiangong-api": {
        "version": "SkyChat-MegaVerse",
        "api_key": "",
        "secret_key": "",
        "provider": "TianGongWorker",
    },

}

# 在以下字典中修改属性值，以指定本地embedding模型存储位置。支持3种设置方法：
# 1、将对应的值修改为模型绝对路径
# 2、不修改此处的值（以 text2vec 为例）：
#       2.1 如果{MODEL_ROOT_PATH}下存在如下任一子目录：
#           - text2vec
#           - GanymedeNil/text2vec-large-chinese
#           - text2vec-large-chinese
#       2.2 如果以上本地路径不存在，则使用huggingface模型
MODEL_PATH = {
    "embed_model": {
        # "ernie-tiny": "nghuyong/ernie-3.0-nano-zh",
        # "ernie-base": "nghuyong/ernie-3.0-base-zh",
        # "text2vec-base": "shibing624/text2vec-base-chinese",
        # "text2vec": "GanymedeNil/text2vec-large-chinese",
        # "text2vec-paraphrase": "shibing624/text2vec-base-chinese-paraphrase",
        # "text2vec-sentence": "shibing624/text2vec-base-chinese-sentence",
        # "text2vec-multilingual": "shibing624/text2vec-base-multilingual",
        # "text2vec-bge-large-chinese": "shibing624/text2vec-bge-large-chinese",
        # "m3e-small": "moka-ai/m3e-small",
        "m3e-base": "/root/moka-ai"
        # "m3e-large": "moka-ai/m3e-large",
        # "bge-small-zh": "BAAI/bge-small-zh",
        # "bge-base-zh": "BAAI/bge-base-zh",
        # "bge-large-zh": "BAAI/bge-large-zh",
        # "bge-large-zh-noinstruct": "BAAI/bge-large-zh-noinstruct",
        # "bge-base-zh-v1.5": "BAAI/bge-base-zh-v1.5",
        # "bge-large-zh-v1.5": "BAAI/bge-large-zh-v1.5",
        # "piccolo-base-zh": "sensenova/piccolo-base-zh",
        # "piccolo-large-zh": "sensenova/piccolo-large-zh",
        # "nlp_gte_sentence-embedding_chinese-large": "damo/nlp_gte_sentence-
embedding_chinese-large",
        # "text-embedding-ada-002": "your OPENAI_API_KEY",
    },

    "llm_model": {
        # 以下部分模型并未完全测试，仅根据fastchat和vllm模型的模型列表推定支持
```

```
        # "chatglm2-6b": "THUDM/chatglm2-6b",
        # "chatglm2-6b-32k": "THUDM/chatglm2-6b-32k",
        #
        # "chatglm3-6b": "THUDM/chatglm3-6b",
        # "chatglm3-6b-32k": "THUDM/chatglm3-6b-32k",
        # "chatglm3-6b-base": "THUDM/chatglm3-6b-base",

        # "Qwen-1_8B": "Qwen/Qwen-1_8B",
        "Qwen-1_8B-Chat": "/root/Qwen-1_8B_Chat"
        # "Qwen-1_8B-Chat-Int8": "Qwen/Qwen-1_8B-Chat-Int8",
        # "Qwen-1_8B-Chat-Int4": "Qwen/Qwen-1_8B-Chat-Int4",
        #
        # "Qwen-7B": "Qwen/Qwen-7B",
        # "Qwen-7B-Chat": "Qwen/Qwen-7B-Chat",
        #
        # "Qwen-14B": "Qwen/Qwen-14B",
        # "Qwen-14B-Chat": "Qwen/Qwen-14B-Chat",
        # "Qwen-14B-Chat-Int8": "Qwen/Qwen-14B-Chat-Int8",
        # "Qwen-14B-Chat-Int4": "Qwen/Qwen-14B-Chat-Int4",
        #
        # "Qwen-72B": "Qwen/Qwen-72B",
        # "Qwen-72B-Chat": "Qwen/Qwen-72B-Chat",
        # "Qwen-72B-Chat-Int8": "Qwen/Qwen-72B-Chat-Int8",
        # "Qwen-72B-Chat-Int4": "Qwen/Qwen-72B-Chat-Int4",
        #
        # "baichuan2-13b": "baichuan-inc/Baichuan2-13B-Chat",
        # "baichuan2-7b": "baichuan-inc/Baichuan2-7B-Chat",
        #
        # "baichuan-7b": "baichuan-inc/Baichuan-7B",
        # "baichuan-13b": "baichuan-inc/Baichuan-13B",
        # "baichuan-13b-chat": "baichuan-inc/Baichuan-13B-Chat",
        #
        # "aquila-7b": "BAAI/Aquila-7B",
        # "aquilachat-7b": "BAAI/AquilaChat-7B",
        #
        # "internlm-7b": "internlm/internlm-7b",
        # "internlm-chat-7b": "internlm/internlm-chat-7b",
        #
        # "falcon-7b": "tiiuae/falcon-7b",
        # "falcon-40b": "tiiuae/falcon-40b",
        # "falcon-rw-7b": "tiiuae/falcon-rw-7b",
        #
        # "gpt2": "gpt2",
        # "gpt2-xl": "gpt2-xl",
        #
        # "gpt-j-6b": "EleutherAI/gpt-j-6b",
        # "gpt4all-j": "nomic-ai/gpt4all-j",
        # "gpt-neox-20b": "EleutherAI/gpt-neox-20b",
        # "pythia-12b": "EleutherAI/pythia-12b",
        # "oasst-sft-4-pythia-12b-epoch-3.5": "OpenAssistant/oasst-sft-4-pythia-12b-epoch-
3.5",
        # "dolly-v2-12b": "databricks/dolly-v2-12b",
        # "stablelm-tuned-alpha-7b": "stabilityai/stablelm-tuned-alpha-7b",
```

```
        #
        # "Llama-2-13b-hf": "meta-llama/Llama-2-13b-hf",
        # "Llama-2-70b-hf": "meta-llama/Llama-2-70b-hf",
        # "open_llama_13b": "openlm-research/open_llama_13b",
        # "vicuna-13b-v1.3": "lmsys/vicuna-13b-v1.3",
        # "koala": "young-geng/koala",
        #
        # "mpt-7b": "mosaicml/mpt-7b",
        # "mpt-7b-storywriter": "mosaicml/mpt-7b-storywriter",
        # "mpt-30b": "mosaicml/mpt-30b",
        # "opt-66b": "facebook/opt-66b",
        # "opt-iml-max-30b": "facebook/opt-iml-max-30b",
        #
        # "agentlm-7b": "THUDM/agentlm-7b",
        # "agentlm-13b": "THUDM/agentlm-13b",
        # "agentlm-70b": "THUDM/agentlm-70b",
        #
        # "Yi-34B-Chat": "https://huggingface.co/01-ai/Yi-34B-Chat",
    },
}


# 通常情况下不需要更改以下内容

# nltk 模型存储路径
NLTK_DATA_PATH = os.path.join(os.path.dirname(os.path.dirname(__file__)), "nltk_data")

VLLM_MODEL_DICT = {
    "aquila-7b": "BAAI/Aquila-7B",
    "aquilachat-7b": "BAAI/AquilaChat-7B",

    "baichuan-7b": "baichuan-inc/Baichuan-7B",
    "baichuan-13b": "baichuan-inc/Baichuan-13B",
    "baichuan-13b-chat": "baichuan-inc/Baichuan-13B-Chat",

    "chatglm2-6b": "THUDM/chatglm2-6b",
    "chatglm2-6b-32k": "THUDM/chatglm2-6b-32k",
    "chatglm3-6b": "THUDM/chatglm3-6b",
    "chatglm3-6b-32k": "THUDM/chatglm3-6b-32k",

    "BlueLM-7B-Chat": "vivo-ai/BlueLM-7B-Chat",
    "BlueLM-7B-Chat-32k": "vivo-ai/BlueLM-7B-Chat-32k",

    # 注意：bloom系列的tokenizer与model是分离的，因此虽然vllm支持，但与fschat框架不兼容
    # "bloom": "bigscience/bloom",
    # "bloomz": "bigscience/bloomz",
    # "bloomz-560m": "bigscience/bloomz-560m",
    # "bloomz-7b1": "bigscience/bloomz-7b1",
    # "bloomz-1b7": "bigscience/bloomz-1b7",

    "internlm-7b": "internlm/internlm-7b",
    "internlm-chat-7b": "internlm/internlm-chat-7b",
    "falcon-7b": "tiiuae/falcon-7b",
```

```
    "falcon-40b": "tiiuae/falcon-40b",
    "falcon-rw-7b": "tiiuae/falcon-rw-7b",
    "gpt2": "gpt2",
    "gpt2-xl": "gpt2-xl",
    "gpt-j-6b": "EleutherAI/gpt-j-6b",
    "gpt4all-j": "nomic-ai/gpt4all-j",
    "gpt-neox-20b": "EleutherAI/gpt-neox-20b",
    "pythia-12b": "EleutherAI/pythia-12b",
    "oasst-sft-4-pythia-12b-epoch-3.5": "OpenAssistant/oasst-sft-4-pythia-12b-epoch-3.5",
    "dolly-v2-12b": "databricks/dolly-v2-12b",
    "stablelm-tuned-alpha-7b": "stabilityai/stablelm-tuned-alpha-7b",
    "Llama-2-13b-hf": "meta-llama/Llama-2-13b-hf",
    "Llama-2-70b-hf": "meta-llama/Llama-2-70b-hf",
    "open_llama_13b": "openlm-research/open_llama_13b",
    "vicuna-13b-v1.3": "lmsys/vicuna-13b-v1.3",
    "koala": "young-geng/koala",
    "mpt-7b": "mosaicml/mpt-7b",
    "mpt-7b-storywriter": "mosaicml/mpt-7b-storywriter",
    "mpt-30b": "mosaicml/mpt-30b",
    "opt-66b": "facebook/opt-66b",
    "opt-iml-max-30b": "facebook/opt-iml-max-30b",

    "Qwen-1_8B": "Qwen/Qwen-1_8B",
    "Qwen-1_8B-Chat": "Qwen/Qwen-1_8B-Chat",
    "Qwen-1_8B-Chat-Int8": "Qwen/Qwen-1_8B-Chat-Int8",
    "Qwen-1_8B-Chat-Int4": "Qwen/Qwen-1_8B-Chat-Int4",

    "Qwen-7B": "Qwen/Qwen-7B",
    "Qwen-7B-Chat": "Qwen/Qwen-7B-Chat",

    "Qwen-14B": "Qwen/Qwen-14B",
    "Qwen-14B-Chat": "Qwen/Qwen-14B-Chat",
    "Qwen-14B-Chat-Int8": "Qwen/Qwen-14B-Chat-Int8",
    "Qwen-14B-Chat-Int4": "Qwen/Qwen-14B-Chat-Int4",

    "Qwen-72B": "Qwen/Qwen-72B",
    "Qwen-72B-Chat": "Qwen/Qwen-72B-Chat",
    "Qwen-72B-Chat-Int8": "Qwen/Qwen-72B-Chat-Int8",
    "Qwen-72B-Chat-Int4": "Qwen/Qwen-72B-Chat-Int4",

    "agentlm-7b": "THUDM/agentlm-7b",
    "agentlm-13b": "THUDM/agentlm-13b",
    "agentlm-70b": "THUDM/agentlm-70b",

}

# 你认为支持Agent能力的模型，可以在这里添加，添加后不会出现可视化界面的警告
# 经过我们测试，原生支持Agent的模型仅有以下几个
SUPPORT_AGENT_MODEL = [
    "azure-api",
    "openai-api",
    "qwen-api",
    "Qwen",
```

```
    "chatglm3",
    "xinghuo-api",
]
```

复制服务相关参数配置模板文件 configs/server_config.py.example 存储至项目路径下 `./configs` 路径下，并重命名为 `server_config.py`

```
import sys
from configs.model_config import LLM_DEVICE

# httpx 请求默认超时时间（秒）。如果加载模型或对话较慢，出现超时错误，可以适当加大该值。
HTTPX_DEFAULT_TIMEOUT = 300.0

# API 是否开启跨域，默认为False，如果需要开启，请设置为True
# is open cross domain
OPEN_CROSS_DOMAIN = False

# 各服务器默认绑定host。如改为"0.0.0.0"需要修改下方所有XX_SERVER的host
DEFAULT_BIND_HOST = "0.0.0.0" if sys.platform != "win32" else "127.0.0.1"

# webui.py server
WEBUI_SERVER = {
    "host": DEFAULT_BIND_HOST,
    "port": 8501,
}

# api.py server
API_SERVER = {
    "host": DEFAULT_BIND_HOST,
    "port": 7861,
}

# fastchat openai_api server
FSCHAT_OPENAI_API = {
    "host": DEFAULT_BIND_HOST,
    "port": 20000,
}

# fastchat model_worker server
# 这些模型必须是在model_config.MODEL_PATH或ONLINE_MODEL中正确配置的。
# 在启动startup.py时，可用通过`--model-name xxxx yyyy`指定模型，不指定则为LLM_MODELS
FSCHAT_MODEL_WORKERS = {
    # 所有模型共用的默认配置，可在模型专项配置中进行覆盖。
    "default": {
        "host": DEFAULT_BIND_HOST,
        "port": 20002,
        "device": LLM_DEVICE,
        # False,'vllm',使用的推理加速框架,使用vllm如果出现HuggingFace通信问题，参见doc/FAQ
        # vllm对一些模型支持还不成熟，暂时默认关闭
        # fschat=0.2.33的代码有bug，如需使用，源码修改fastchat.server.vllm_worker，
        # 将103行中sampling_params = SamplingParams的参数stop=list(stop)修改为stop= [i for i
in stop if i!=""]
```

```
        "infer_turbo": False,

        # model_worker多卡加载需要配置的参数
        # "gpus": None, # 使用的GPU，以str的格式指定，如"0,1"，如失效请使用
CUDA_VISIBLE_DEVICES="0,1"等形式指定
        # "num_gpus": 1, # 使用GPU的数量
        # "max_gpu_memory": "20GiB", # 每个GPU占用的最大显存

        # 以下为model_worker非常用参数，可根据需要配置
        # "load_8bit": False, # 开启8bit量化
        # "cpu_offloading": None,
        # "gptq_ckpt": None,
        # "gptq_wbits": 16,
        # "gptq_groupsize": -1,
        # "gptq_act_order": False,
        # "awq_ckpt": None,
        # "awq_wbits": 16,
        # "awq_groupsize": -1,
        # "model_names": LLM_MODELS,
        # "conv_template": None,
        # "limit_worker_concurrency": 5,
        # "stream_interval": 2,
        # "no_register": False,
        # "embed_in_truncate": False,

        # 以下为vllm_worker配置参数,注意使用vllm必须有gpu，仅在Linux测试通过

        # tokenizer = model_path # 如果tokenizer与model_path不一致在此处添加
        # 'tokenizer_mode':'auto',
        # 'trust_remote_code':True,
        # 'download_dir':None,
        # 'load_format':'auto',
        # 'dtype':'auto',
        # 'seed':0,
        # 'worker_use_ray':False,
        # 'pipeline_parallel_size':1,
        # 'tensor_parallel_size':1,
        # 'block_size':16,
        # 'swap_space':4 , # GiB
        # 'gpu_memory_utilization':0.90,
        # 'max_num_batched_tokens':2560,
        # 'max_num_seqs':256,
        # 'disable_log_stats':False,
        # 'conv_template':None,
        # 'limit_worker_concurrency':5,
        # 'no_register':False,
        # 'num_gpus': 1
        # 'engine_use_ray': False,
        # 'disable_log_requests': False

    },
    # 可以如下示例方式更改默认配置
    "Qwen-1_8B-Chat": { # 使用default中的IP和端口
```

```
        "device": "cuda",
    },
    # "chatglm3-6b": {  # 使用default中的IP和端口
    #     "device": "cuda",
    # },

    # 以下配置可以不用修改，在model_config中设置启动的模型
    "zhipu-api": {
        "port": 21001,
    },
    "minimax-api": {
        "port": 21002,
    },
    "xinghuo-api": {
        "port": 21003,
    },
    "qianfan-api": {
        "port": 21004,
    },
    "fangzhou-api": {
        "port": 21005,
    },
    "qwen-api": {
        "port": 21006,
    },
    "baichuan-api": {
        "port": 21007,
    },
    "azure-api": {
        "port": 21008,
    },
    "tiangong-api": {
        "port": 21009,
    },
}

# fastchat multi model worker server
FSCHAT_MULTI_MODEL_WORKERS = {
    # TODO:
}

# fastchat controller server
FSCHAT_CONTROLLER = {
    "host": DEFAULT_BIND_HOST,
    "port": 20001,
    "dispatch_method": "shortest_queue",
}
```

复制服务相关参数配置模板文件 configs/basic_config.py.example 存储至项目路径下 `./configs` 路径下，并重命名为 `basic_config.py`

```
import logging
```

```
import os
import langchain
import tempfile
import shutil


# 是否显示详细日志
log_verbose = False
langchain.verbose = False

# 通常情况下不需要更改以下内容

# 日志格式
LOG_FORMAT = "%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s"
logger = logging.getLogger()
logger.setLevel(logging.INFO)
logging.basicConfig(format=LOG_FORMAT)


# 日志存储路径
LOG_PATH = os.path.join(os.path.dirname(os.path.dirname(__file__)), "logs")
if not os.path.exists(LOG_PATH):
    os.mkdir(LOG_PATH)

# 临时文件目录，主要用于文件对话
BASE_TEMP_DIR = os.path.join(tempfile.gettempdir(), "chatchat")
if os.path.isdir(BASE_TEMP_DIR):
    shutil.rmtree(BASE_TEMP_DIR)
os.makedirs(BASE_TEMP_DIR, exist_ok=True)
```

复制服务相关参数配置模板文件 configs/kb_config.example 存储至项目路径下 `./configs` 路径下，并重命名为 `kb_config.py`

```
import os

# 默认使用的知识库
DEFAULT_KNOWLEDGE_BASE = "samples"

# 默认向量库/全文检索引擎类型。可选：faiss, milvus(离线) & zilliz(在线), pgvector,全文检索引擎es
DEFAULT_VS_TYPE = "faiss"

# 缓存向量库数量（针对FAISS）
CACHED_VS_NUM = 1

# 缓存临时向量库数量（针对FAISS），用于文件对话
CACHED_MEMO_VS_NUM = 10

# 知识库中单段文本长度(不适用MarkdownHeaderTextSplitter)
CHUNK_SIZE = 250

# 知识库中相邻文本重合长度(不适用MarkdownHeaderTextSplitter)
OVERLAP_SIZE = 50
```

```
# 知识库匹配向量数量
VECTOR_SEARCH_TOP_K = 3

# 知识库匹配相关度阈值，取值范围在0-1之间，SCORE越小，相关度越高，取到1相当于不筛选，建议设置在0.5左右
SCORE_THRESHOLD = 1

# 默认搜索引擎。可选：bing，duckduckgo，metaphor
DEFAULT_SEARCH_ENGINE = "duckduckgo"

# 搜索引擎匹配结题数量
SEARCH_ENGINE_TOP_K = 3


# Bing 搜索必备变量
# 使用 Bing 搜索需要使用 Bing Subscription Key,需要在azure port中申请试用bing search
# 具体申请方式请见
# https://learn.microsoft.com/en-us/bing/search-apis/bing-web-search/create-bing-search-service-resource
# 使用python创建bing api 搜索实例详见:
# https://learn.microsoft.com/en-us/bing/search-apis/bing-web-search/quickstarts/rest/python
BING_SEARCH_URL = "https://api.bing.microsoft.com/v7.0/search"
# 注意不是bing Webmaster Tools的api key,

# 此外，如果是在服务器上，报Failed to establish a new connection: [Errno 110] Connection timed out
# 是因为服务器加了防火墙，需要联系管理员加白名单，如果公司的服务器的话，就别想了GG
BING_SUBSCRIPTION_KEY = ""

# metaphor搜索需要KEY
METAPHOR_API_KEY = ""


# 是否开启中文标题加强，以及标题增强的相关配置
# 通过增加标题判断，判断哪些文本为标题，并在metadata中进行标记；
# 然后将文本与往上一级的标题进行拼合，实现文本信息的增强。
ZH_TITLE_ENHANCE = False


# 每个知识库的初始化介绍，用于在初始化知识库时显示和Agent调用，没写则没有介绍，不会被Agent调用。
KB_INFO = {
    "知识库名称": "知识库介绍",
    "samples": "关于本项目issue的解答",
}


# 通常情况下不需要更改以下内容

# 知识库默认存储路径
KB_ROOT_PATH = os.path.join(os.path.dirname(os.path.dirname(__file__)), "knowledge_base")
if not os.path.exists(KB_ROOT_PATH):
    os.mkdir(KB_ROOT_PATH)
```

```
# 数据库默认存储路径。
# 如果使用sqlite，可以直接修改DB_ROOT_PATH；如果使用其它数据库，请直接修改SQLALCHEMY_DATABASE_URI。
DB_ROOT_PATH = os.path.join(KB_ROOT_PATH, "info.db")
SQLALCHEMY_DATABASE_URI = f"sqlite:///{DB_ROOT_PATH}"

# 可选向量库类型及对应配置
kbs_config = {
    "faiss": {
    },
    "milvus": {
        "host": "127.0.0.1",
        "port": "19530",
        "user": "",
        "password": "",
        "secure": False,
    },
    "zilliz": {
        "host": "in01-a7ce524e41e3935.ali-cn-hangzhou.vectordb.zilliz.com.cn",
        "port": "19530",
        "user": "",
        "password": "",
        "secure": True,
        },
    "pg": {
        "connection_uri":
"postgresql://postgres:postgres@127.0.0.1:5432/langchain_chatchat",
    },

    "es": {
        "host": "127.0.0.1",
        "port": "9200",
        "index_name": "test_index",
        "user": "",
        "password": ""
    }
}

# TextSplitter配置项，如果你不明白其中的含义，就不要修改。
text_splitter_dict = {
    "ChineseRecursiveTextSplitter": {
        "source": "huggingface",   # 选择tiktoken则使用openai的方法
        "tokenizer_name_or_path": "",
    },
    "SpacyTextSplitter": {
        "source": "huggingface",
        "tokenizer_name_or_path": "gpt2",
    },
    "RecursiveCharacterTextSplitter": {
        "source": "tiktoken",
        "tokenizer_name_or_path": "cl100k_base",
    },
    "MarkdownHeaderTextSplitter": {
        "headers_to_split_on":
```

```
            [
                ("#", "head1"),
                ("##", "head2"),
                ("###", "head3"),
                ("####", "head4"),
            ]
    },
}

# TEXT_SPLITTER 名称
TEXT_SPLITTER_NAME = "ChineseRecursiveTextSplitter"

# Embedding模型定制词语的词表文件
EMBEDDING_KEYWORD_FILE = "embedding_keywords.txt"
```

复制服务相关参数配置模板文件 configs/prompt_config.example 存储至项目路径下 `./configs` 路径下，并重命名为 `prompt_config.py`

```
# prompt模板使用Jinja2语法，简单点就是用双大括号代替f-string的单大括号
# 本配置文件支持热加载，修改prompt模板后无需重启服务。

# LLM对话支持的变量：
#   - input: 用户输入内容

# 知识库和搜索引擎对话支持的变量：
#   - context: 从检索结果拼接的知识文本
#   - question: 用户提出的问题

# Agent对话支持的变量：

#   - tools: 可用的工具列表
#   - tool_names: 可用的工具名称列表
#   - history: 用户和Agent的对话历史
#   - input: 用户输入内容
#   - agent_scratchpad: Agent的思维记录

PROMPT_TEMPLATES = {
    "llm_chat": {
        "default":
            '{{ input }}',

        "with_history":
            'The following is a friendly conversation between a human and an AI. '
            'The AI is talkative and provides lots of specific details from its context. '
            'If the AI does not know the answer to a question, it truthfully says it does
not know.\n\n'
            'Current conversation:\n'
            '{history}\n'
            'Human: {input}\n'
            'AI:',

        "py":
```

```
            '你是一个聪明的代码助手，请你给我写出简单的py代码。 \n'
            '{{ input }}',
    },


    "knowledge_base_chat": {
        "default":
            '<指令>根据已知信息，简洁和专业的来回答问题。如果无法从中得到答案，请说“根据已知信息无法回答该问题”，'
            '不允许在答案中添加编造成分，答案请使用中文。 </指令>\n'
            '<已知信息>{{ context }}</已知信息>\n'
            '<问题>{{ question }}</问题>\n',

        "text":
            '<指令>根据已知信息，简洁和专业的来回答问题。如果无法从中得到答案，请说“根据已知信息无法回答该问题”，答案请使用中文。 </指令>\n'
            '<已知信息>{{ context }}</已知信息>\n'
            '<问题>{{ question }}</问题>\n',

        "empty":  # 搜不到知识库的时候使用
            '请你回答我的问题:\n'
            '{{ question }}\n\n',
    },


    "search_engine_chat": {
        "default":
            '<指令>这是我搜索到的互联网信息，请你根据这些信息进行提取并有调理，简洁的回答问题。'
            '如果无法从中得到答案，请说“无法搜索到能回答问题的内容”。 </指令>\n'
            '<已知信息>{{ context }}</已知信息>\n'
            '<问题>{{ question }}</问题>\n',

        "search":
            '<指令>根据已知信息，简洁和专业的来回答问题。如果无法从中得到答案，请说“根据已知信息无法回答该问题”，答案请使用中文。 </指令>\n'
            '<已知信息>{{ context }}</已知信息>\n'
            '<问题>{{ question }}</问题>\n',
    },


    "agent_chat": {
        "default":
            'Answer the following questions as best you can. If it is in order, you can use some tools appropriately. '
            'You have access to the following tools:\n\n'
            '{tools}\n\n'
            'Use the following format:\n'
            'Question: the input question you must answer1\n'
            'Thought: you should always think about what to do and what tools to use.\n'
            'Action: the action to take, should be one of [{tool_names}]\n'
            'Action Input: the input to the action\n'
            'Observation: the result of the action\n'
```

```
            '... (this Thought/Action/Action Input/Observation can be repeated zero or more
times)\n'
            'Thought: I now know the final answer\n'
            'Final Answer: the final answer to the original input question\n'
            'Begin!\n\n'
            'history: {history}\n\n'
            'Question: {input}\n\n'
            'Thought: {agent_scratchpad}\n',

        "ChatGLM3":
            'You can answer using the tools, or answer directly using your knowledge
without using the tools. '
            'Respond to the human as helpfully and accurately as possible.\n'
            'You have access to the following tools:\n'
            '{tools}\n'
            'Use a json blob to specify a tool by providing an action key (tool name) '
            'and an action_input key (tool input).\n'
            'Valid "action" values: "Final Answer" or  [{tool_names}]'
            'Provide only ONE action per $JSON_BLOB, as shown:\n\n'
            '```\n'
            '{{{{\n'
            '  "action": $TOOL_NAME,\n'
            '  "action_input": $INPUT\n'
            '}}}}\n'
            '```\n\n'
            'Follow this format:\n\n'
            'Question: input question to answer\n'
            'Thought: consider previous and subsequent steps\n'
            'Action:\n'
            '```\n'
            '$JSON_BLOB\n'
            '```\n'
            'Observation: action result\n'
            '... (repeat Thought/Action/Observation N times)\n'
            'Thought: I know what to respond\n'
            'Action:\n'
            '```\n'
            '{{{{\n'
            '  "action": "Final Answer",\n'
            '  "action_input": "Final response to human"\n'
            '}}}}\n'
            'Begin! Reminder to ALWAYS respond with a valid json blob of a single action.
Use tools if necessary. '
            'Respond directly if appropriate. Format is Action:```$JSON_BLOB```then
Observation:.\n'
            'history: {history}\n\n'
            'Question: {input}\n\n'
            'Thought: {agent_scratchpad}',
    }
}
```

7、启动项目文件

首先运行知识库初始化文件，在Terminal中依次运行以下代码：

第一次运行项目的时候，

```
python init_database.py --recreate-vs
```

不是第一次运行项目，

```
python init_database.py
```

（默认samples为知识库，可以理解为存放知识库文件的目录）

初始化知识库后，向量化模型会启动（将指定知识库内文件转换为向量并入库）。

接下来启动模型文件，启动方法和官方指定开启方法一致（在项目文件目录下执行）

```
python startup.py -a
```

8、查看启动测试页面

由于LangChain-ChatChat是支持可视化页面操作，更方便了知识库的构建，因此正常情况，在启动项目文件后，Terminal中会提示启动端口支持访问，由于本次测试环境使用的是AutoDL，这里附上将LLM启动端口映射到本地访问的方法：

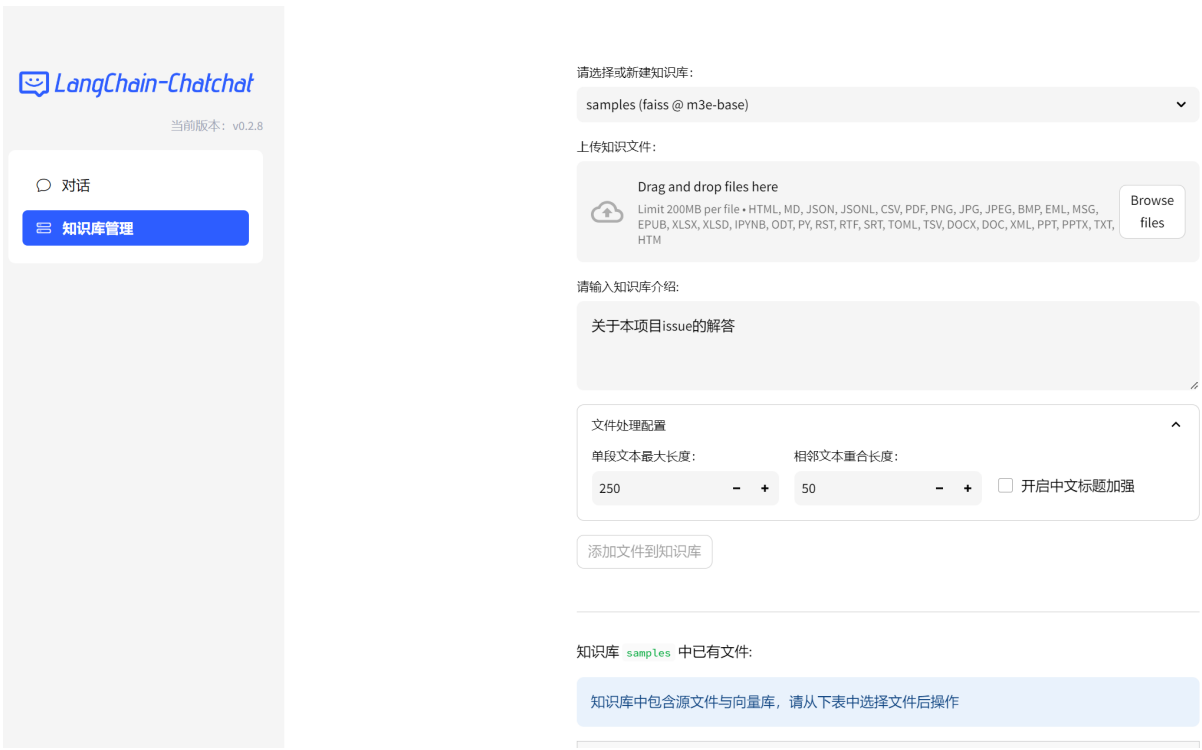
首先，点击AutoDL的容器实例，找到自定义服务

按照提示，下载autodl-SSH隧道工具，并打开，将上图中的SSH登录下的信息填写到工具的对应位置，例如

点击开始代理后，可以看到如上图一样的画面，这样你就得到了映射端口（可以任意指定），这个端口与你Terminal中启动文件中提示的端口是一致的，可以任意添加和修改

映射成功后，你将看到如下页面，以8501端口为例，这里选择"知识库问答"模式，对默认知识库中的问题进行大模型对话

当然，也可以自己上传文档，格式支持任意格式上传，上传后，模型将自动开始将上传文件进行向量化存储，在完成后，即可与LLM进行上传指定的知识库进行问答



至此，恭喜你已经完成了全部的项目复现

如果你感觉此文档对你有所帮助，希望你留下star，共同讨论，一起进步！