一、模型代码准备

1. 可用模型调用代码

2. 服务配置文件：config.json

示例如下（见inference_example文件夹）。

```
{
    "name":"regnet-classification",
    "input":[
        {
            "path":"input/1.jpeg",
            "type":"image"
        },
        {
            "path":"2.txt",
            "type":"text"
        }
    ],
    "output":[
        {
            "path":"output/output.json",
            "type":"json"
```

```
        }
    ],
    "base_image": "pytorch/pytorch:2.0.0-cuda11.7-cudnn8-runtime",
    "entrypoint": "python inference.py"
}
```

其中：（以下字段均为必填，与模型调用代码有关的要求均用斜体标出）

- name：一个字符串，表示自定义项目名称；
- input：一个列表，表示模型调用需要的输入文件列表，每个元素是一个字典，包括path和type两个字段，分别表示模型调用代码中使用的输入文件路径和类型；
  - 每个模型可以有多个输入
  - 目前支持的输入类型只有：image、text
  - *要求模型调用代码中必须以文件的形式作为输入*
- output：一个列表，表示模型调用后存放结果的文件列表，每个元素是一个字典，包括path和type两个字段，分别表示输出文件路径和类型；
  - *要求所有输出都在一个json文件中*
  - 目前支持的输出类型只有：json
  - 目前只支持一个输出文件
- base_image: 默认pytorch/pytorch:2.0.0-cuda11.7-cudnn8-runtime作为运行模型代码调用的基础镜像
  - 如果使用其他基础镜像可能存在网络不佳无法下载或版本兼容性问题
  - 镜像中配置及其他基础镜像选择参见 https://hub.docker.com/r/pytorch/pytorch/tags
- entrypoint: 模型调用代码的入口命令，例如python inference.py

3. requirements.txt

用于指定模型调用代码的依赖包，将在构建镜像时通过`pip install -r requirements.txt`安装模型调用的依赖。

4. 其他事宜

- 将以上所有内容打包为zip文件，作为模型调用代码的压缩包；压缩包中内容需要精简，大小最好不要超过100M，否则可能上传出错或者Docker镜像构建失败。
- 建议把模型封装在本地，不使用网络下载，因为不保证部署地点的网络环境。

二、调用接口生成工具：(example/call_tool.py)

1. 创建新的模型服务

注意：此处返回的模型服务名称，不是config文件中的自定义项目名称；后续操作需要使用此处返回的服务名称

```
'''
输入：zip文件路径
输出：一个json字典，包括{"call": 一个简单的模型服务调用示例, "url": 模型服务的url,
"name": 模型服务的名称}
'''
def create(filepath):
    pass
```

在示例代码中，create接口生成的简单的模型服务调用示例会被保存在call_gen.py中，可以直接运行call_gen.py调用生成的服务（注意需要指定作为模型调用输入的文件路径，并解析为和示例中一致的格式，如图片需要base64encode，文本使用字符串即可）。

2. 删除已有的模型服务

```
'''
输入：创建时返回的模型服务名称
输出：无
'''
def delete(name):
    pass
```

3. 获取模型服务信息

```
'''
输入：创建时返回的模型服务名称
输出：一个json字典，包括{"call": 一个简单的模型服务调用示例, "url": 模型服务的url,
"name": 模型服务的名称}
'''
```

```
def get(name):
    pass
```

4. 获取模型服务列表

```
'''
输入：无
输出：一个json字典，包括{"message": 所有目前已有服务的列表}
'''
def list():
    pass
```

5. 更新模型服务

```
'''
输入：创建时返回的模型服务名称，zip文件路径
输出：保留原有服务的url不变，返回一个json字典，包括{"call": 一个简单的模型服务调用示例,
"url": 模型服务的url, "name": 模型服务的名称}
'''
def update(name, filepath):
    pass
```

其他说明

- 代码仓库：https://github.com/morgen52/ML_autoservice.git
- Demo：https://disk.pku.edu.cn:443/link/CFEDE1AD31B839BCCEE927CE3BC302FE

工具部署：

1. 修改IP地址（工具的和ML服务的）

- 指定工具URL：修改example/call_tool.py中的base_url，指定工具的IP（本机IP地址）和端口号（默认8000）

- 指定ML服务URL：修改tool.py中的CONST_IP为本机IP，工具将基于该IP地址提供ML服务URL。

2. 创建执行环境并部署工具：

```
conda create --name <env> --file ./requirements.txt
conda activate <env>
python tool.py
```