交互、部署与接口调用

其实这一块我主要学习的就是flask框架，主要内容在爬虫里学了（^_^），而apifox和后端中使用的postman用法相似

HTTP协议：

http称为超文本传输协议，其核心是客户端发送请求 → 服务器返回响应的过程。而https就是相对于http进行了SSL/TLS加密。现在的大部分网站都使用的是https。一个常见的http只要包含request（请求），respond（响应）。在request中，主要包含请求行，请求头，请求体。其中请求行只要包含url，协议，和请求方式。而请起头在反爬中十分重要。其中包含几个方面：user-agent，referer，cookie，这三者在反爬中都有十分重要的作用。而响应中主要包含状态行，响应头，响应体。

请求方式：

get和post分别称为显示提交和隐式提交，简单来说，一般而言get用于搜索，post用于增加。但我们又不是做前端的，我们只需要明确网页是用get还是post来访问的即可

不同的状态码：

200，301等2,3开头的一般都代表成功，而4开头一般都代表着用户端失败，其中最常见的404一般代表着未找到，很有可能是你的程序被反爬了。而403则主要代表你没有权限。5开头一般代表着服务器端失效，一般而言是请求超时。

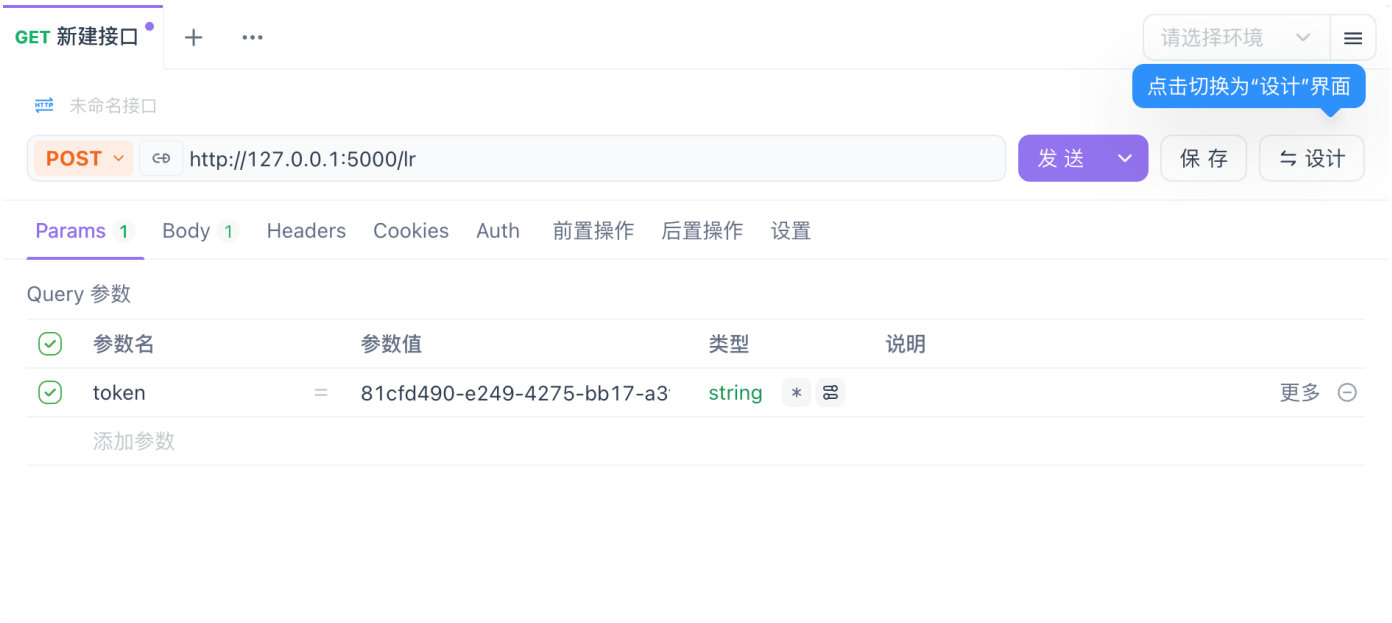
掌握如何使用其中一个框架搭建基本的 API 服务，定义预测接口，并实现请求处理与模型调用的逻辑。

我在这里是基于flask框架来实现的。这是flask框架的一个基本信息

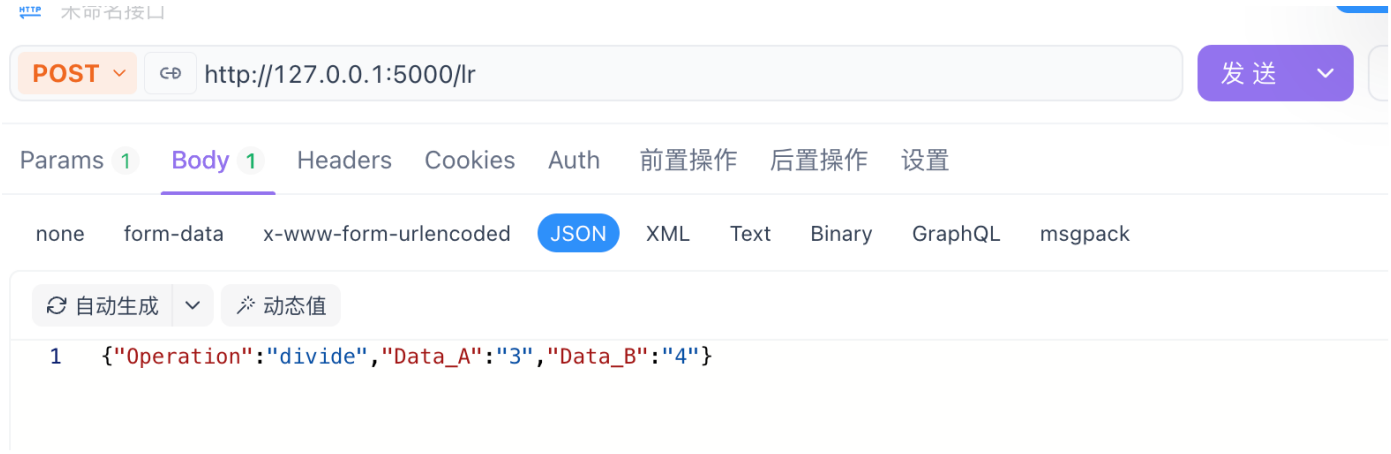
```
@app.route('/lr', methods=['POST'])
def lr():
     return("xxx")
if __name__ == '__main__':
    app.run()
```

这就是一个基本的api服务，同时我们可以使用request.args.get或者request.from.get来获取数据（接口），提取到数据后就可以使用我们自己的模型进行训练并将结果以json的形式返回（但原则上其只能返回字符串，所以我们最好使用json.dumps或者flask框架内置的jsonify）

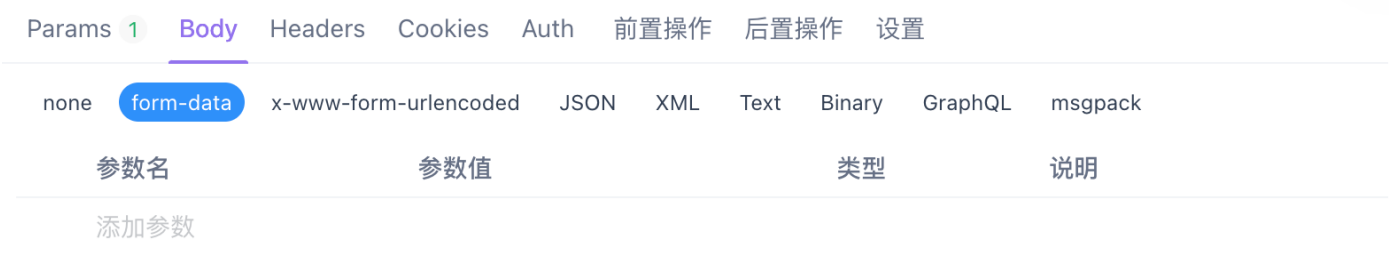
学会使用apifox发送模拟请求的操作，并学会如何使用apifox发送不同的表单数据信息



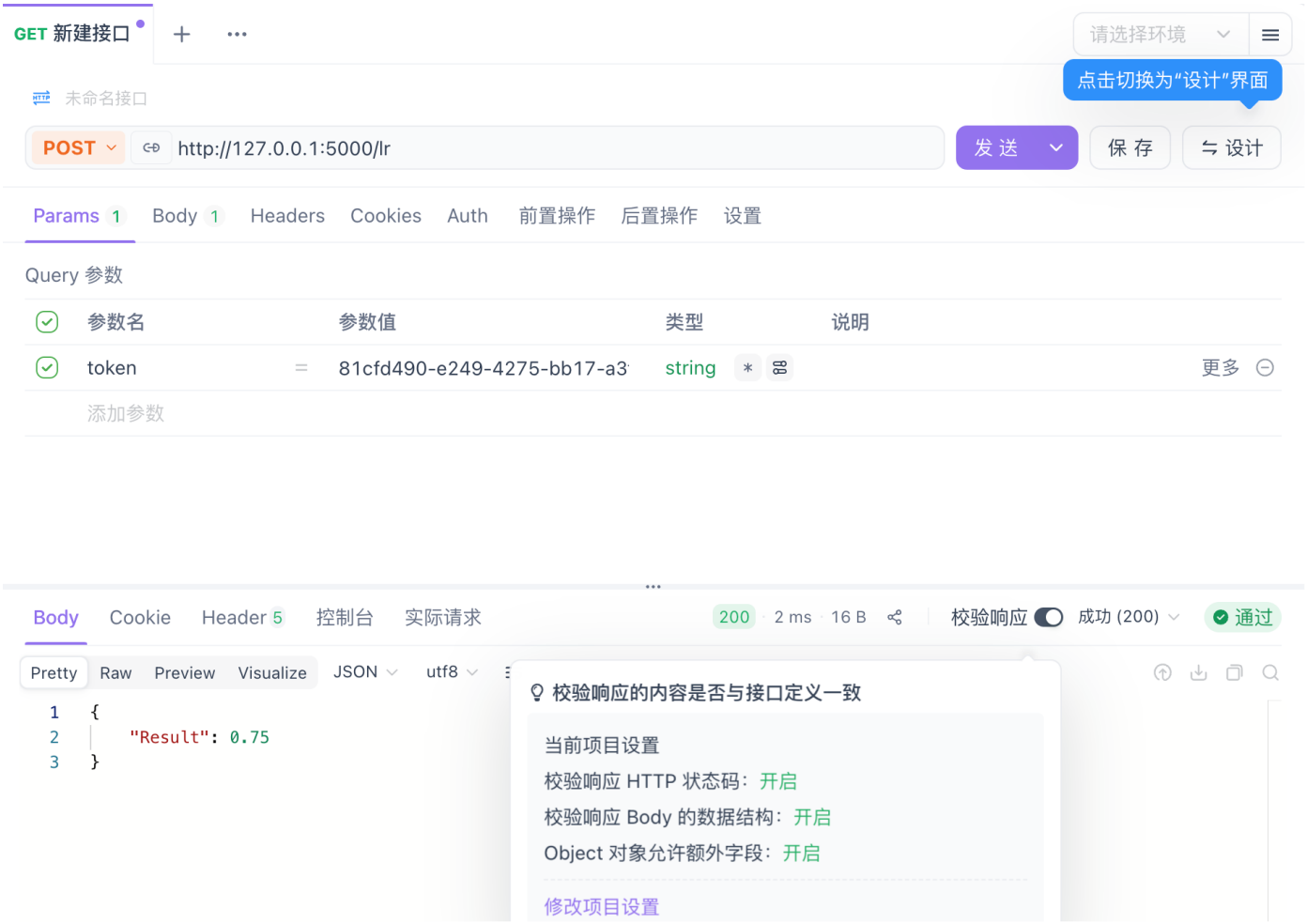
这里左上角决定了我们发送的是get请求还是post请求，如果是post请求我们则可以在body处编写请求携带的内容



这是已json的形式发送请求，同时我们也可以data的形式发送



实现一个请求处理函数




```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [20/Oct/2025 20:31:00] "POST /lr HTTP/1.1" 200 -
127.0.0.1 - - [20/Oct/2025 20:32:25] "POST /lr?token=80180088-9009-4b75-9479-0798afad806e HTTP/1.1" 200 -
user_id X
127.0.0.1 - - [20/Oct/2025 20:32:51] "POST /lr?token=81cfd490-e249-4275-bb17-a3f3c9514cb6 HTTP/1.1" 200 -
user_id 凌睿

进程已结束，退出代码为 0
```

调用模型

这里我选择的模型为在花卉小能手中训练的花卉预测模型

原模型：

```python
import torch
import torch.nn as nn
from PIL import Image
from torchvision import transforms

class Residual(nn.Module):
    def __init__(self, input_channels, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
```

```
        self.conv1 = nn.Conv2d(input_channels, num_channels, kernel_size=3, padding=1,
stride=strides)
        self.conv2 = nn.Conv2d(num_channels, num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2d(input_channels, num_channels, kernel_size=1,
stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm2d(num_channels)
        self.bn2 = nn.BatchNorm2d(num_channels)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, X):
        Y = self.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return self.relu(Y)


b1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
)


def resnet_block(input_channels, num_channels, num_residuals, first_block=False):
    blk = []
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.append(Residual(input_channels, num_channels, use_1x1conv=True,
strides=2))
        else:
            blk.append(Residual(num_channels, num_channels))
    return blk


b2 = nn.Sequential(*resnet_block(64, 64, 2, first_block=True))
b3 = nn.Sequential(*resnet_block(64, 128, 2))
b4 = nn.Sequential(*resnet_block(128, 256, 2))
b5 = nn.Sequential(*resnet_block(256, 512, 2))

net = nn.Sequential(
    b1, b2, b3, b4, b5,
    nn.AdaptiveAvgPool2d((1, 1)),
    nn.Flatten(),
    nn.Linear(512, 5)
)
```

```
net.load_state_dict(torch.load('CNN.params'))


image_path = 'c773156011c4b1085db8cb113b2a0cd2.jpg'
image = Image.open(image_path)

def get_transforms():
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

trans = get_transforms()
features=trans(image)
features=features.reshape(1, 3, 224, 224)
labels=net(features)
labels = labels.argmax(axis=1)
def get_labels(labels):
    text_labels=['daisy','dandelion','rose','sunflower','tulip']
    return [text_labels[int(i)] for i in labels]
print(get_labels(labels))
```

基于此，通过flask框架开发的程序为：

```
from flask import Flask, request, jsonify
from werkzeug.utils import secure_filename
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image
import io

class Residual(nn.Module):
    def __init__(self, input_channels, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
        self.conv1 = nn.Conv2d(input_channels, num_channels, kernel_size=3, padding=1,
stride=strides)
        self.conv2 = nn.Conv2d(num_channels, num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2d(input_channels, num_channels, kernel_size=1,
stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm2d(num_channels)
        self.bn2 = nn.BatchNorm2d(num_channels)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, X):
        Y = self.relu(self.bn1(self.conv1(X)))
```

```
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return self.relu(Y)


class FlowerResNet:
    def __init__(self, model_path='CNN.params'):
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.model = self._build_model()
        self.model.load_state_dict(torch.load(model_path, map_location=self.device))
        self.model.to(self.device)
        self.model.eval()
        self.transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
        ])

        self.text_labels = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']

    def _build_model(self):
        b1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )

        def resnet_block(input_channels, num_channels, num_residuals, first_block=False):
            blk = []
            for i in range(num_residuals):
                if i == 0 and not first_block:
                    blk.append(Residual(input_channels, num_channels, use_1x1conv=True,
strides=2))
                else:
                    blk.append(Residual(num_channels, num_channels))
            return blk
        b2 = nn.Sequential(*resnet_block(64, 64, 2, first_block=True))
        b3 = nn.Sequential(*resnet_block(64, 128, 2))
        b4 = nn.Sequential(*resnet_block(128, 256, 2))
        b5 = nn.Sequential(*resnet_block(256, 512, 2))
        net = nn.Sequential(
            b1, b2, b3, b4, b5,
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten(),
            nn.Linear(512, 5)
        )

        return net
```

```
    def predict(self, image_file, **kwargs):
        try:

            if isinstance(image_file, str):
                image = Image.open(image_file).convert('RGB')
            else:
                image = Image.open(io.BytesIO(image_file.read())).convert('RGB')
            features = self.transform(image)
            features = features.unsqueeze(0)
            features = features.to(self.device)
            with torch.no_grad():
                outputs = self.model(features)
                probabilities = torch.nn.functional.softmax(outputs, dim=1)
                confidence, predicted = torch.max(probabilities, 1)
            predicted_class = int(predicted.item())
            confidence_score = float(confidence.item())
            predicted_label = self.text_labels[predicted_class]
            all_probabilities = {
                label: float(prob)
                for label, prob in zip(self.text_labels, probabilities[0].cpu().numpy())
            }

            result = {
                "predicted_class": predicted_label,
                "confidence": confidence_score,
                "probabilities": all_probabilities,
            }

            return result

        except Exception as e:
            return {"error": f"预测失败: {str(e)}"}


app = Flask(__name__)
flower_model = FlowerResNet()
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'bmp'}


def allowed_file(filename):
    return '.' in filename and \
        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': '文件提取失败'}), 400
    file = request.files['file']
    if file.filename == '':
```

```
        return jsonify({'error': '文件提取失败'}), 400
    if file and allowed_file(file.filename):
        try:
            form_data = {}
            for key in request.form:
                form_data[key] = request.form[key]
            # （重要********）调用模型进行预测（直接使用文件对象，避免保存到磁盘）
            prediction_result = flower_model.predict(file, **form_data)
            if 'error' in prediction_result:
                return jsonify({
                    'status': 'error',
                    'message': prediction_result['error']
                }), 500

            return jsonify({
                'status': 'success',
                'prediction': prediction_result
            })

        except Exception as e:
            return jsonify({
                'status': 'error',
                'message': f'处理文件时发生错误: {str(e)}'
            }), 500

    else:
        return jsonify({
            'error': '图片解析失败'
        }), 400


@app.route('/predict', methods=['POST'])
def api_predict():
    json_data = {}
    if request.content_type == 'application/json':
        json_data = request.get_json() or {}

    if 'file' not in request.files:
        return jsonify({'error': '文件提取失败'}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({'error': '文件提取失败'}), 400

    if not allowed_file(file.filename):
        return jsonify({'error': '文件提取失败'}), 400

    try:
        all_params = {}
        for key in request.form:
            all_params[key] = request.form[key]
```

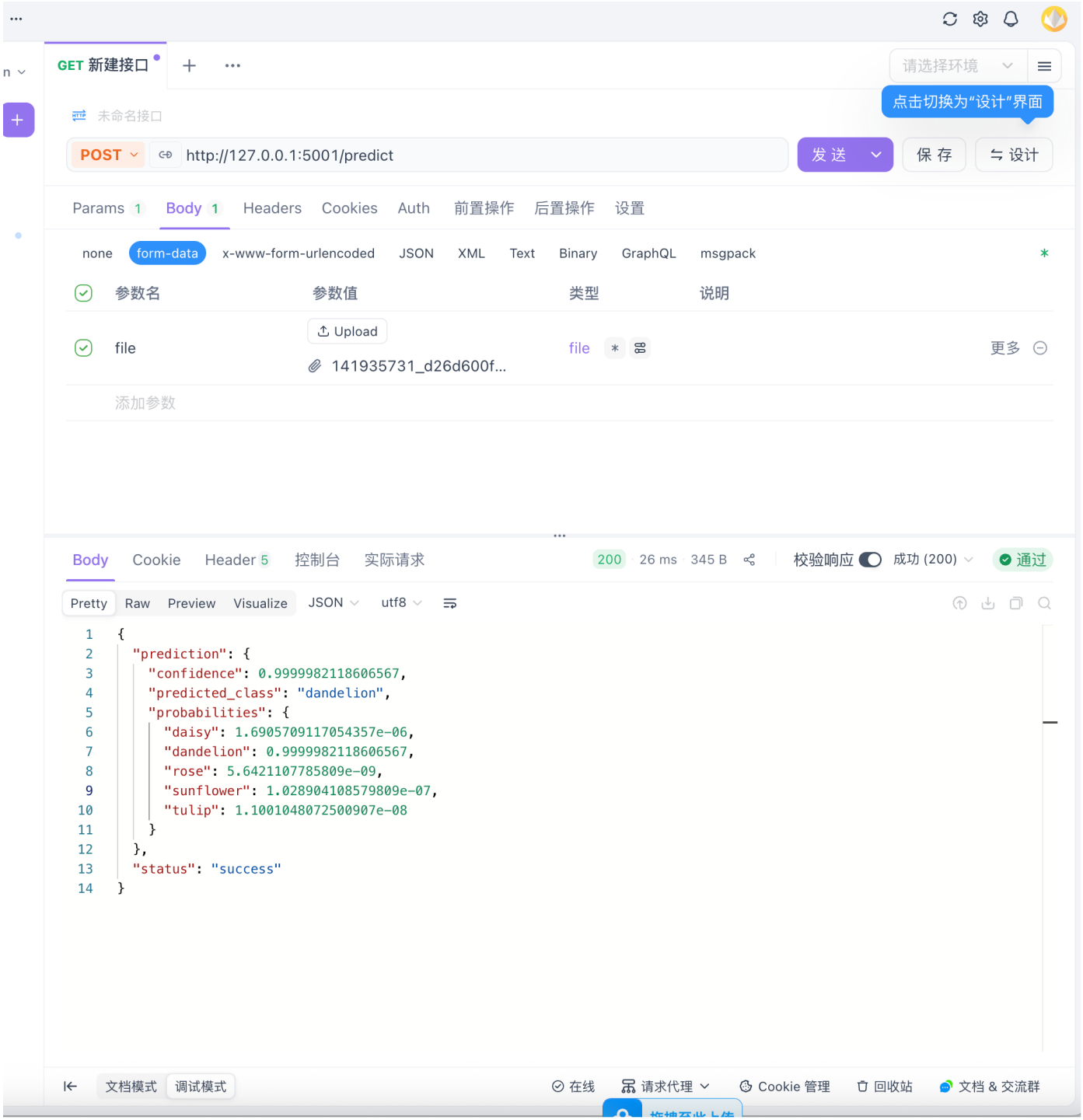
```
        all_params.update(json_data)
        prediction_result = flower_model.predict(file, **all_params)
        if 'error' in prediction_result:
            return jsonify({
                'status': 'error',
                'message': prediction_result['error']
            }), 500
        response = {
            'status': 'success',
            'data': {
                'filename': secure_filename(file.filename),
                'prediction_result': prediction_result,
            }
        }

        return jsonify(response)

    except Exception as e:
        return jsonify({
            'status': 'error',
            'message': f'预测过程中发生错误: {str(e)}'
        }), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001, debug=True)
    #上一个程序用的port为5000
```

在apifox上进行调试：


GET 新建接口
请选择环境
点击切换为"设计"界面
未命名接口
POST
http://127.0.0.1:5001/predict
发 送
保 存
设计
Params 1
Body 1
Headers
Cookies
Auth
前置操作
后置操作
设置
none
form-data
x-www-form-urlencoded
JSON
XML
Text
Binary
GraphQL
msgpack
参数名
参数值
类型
说明
file
Upload
141935731_d26d600f...
file
更多
添加参数
Body
Cookie
Header 5
控制台
实际请求
200
26 ms
345 B
校验响应
成功 (200)
通过
Pretty
Raw
Preview
Visualize
JSON
utf8
{
"prediction": {
"confidence": 0.9999982118606567,
"predicted_class": "dandelion",
"probabilities": {
"daisy": 1.6905709117054357e-06,
"dandelion": 0.9999982118606567,
"rose": 5.6421107785809e-09,
"sunflower": 1.028904108579809e-07,
"tulip": 1.1001048072500907e-08
}
},
"status": "success"
}
文档模式
调试模式
在线
请求代理
Cookie 管理
回收站
文档 & 交流群

这里使用的图像如下：