**Flask 应用代码说明文档**

**一、整体概述**

这是一个基于 Flask 框架开发的 Web 应用，主要用于处理与图像生成、存储、查询和管理相关的操作。该应用允许前端通过 API 接口发送请求，实现图像的生成、历史图像的查询、单张图像的获取、图像的保存和删除等功能。

**二、代码结构和各部分功能**

**（一）导入必要的库和模块**

收起

python

from flask import Flask, jsonify, request, send\_file

from flask\_cors import CORS

import io

import base64

import emotion

import image\_generate

import image\_keywords

import link\_mysql

1. **Flask 相关导入**：Flask、jsonify、request 和 send\_file 是 Flask 框架提供的核心工具。Flask 用于创建 Web 应用；jsonify 用于处理请求和响应，将数据转换为 JSON 格式返回；request 用于获取前端发送的请求数据；send\_file 用于返回文件给前端。
2. **CORS 导入**：CORS 用于解决跨域资源共享问题，允许不同域名的前端页面访问该 API。通过 CORS(app) 配置，允许所有来源的请求访问该应用的 API。
3. **其他导入**：
   * io 模块：提供了对二进制数据的处理功能，用于操作图像的二进制流，在处理图像数据时起到关键作用。
   * base64 模块：用于对图像数据进行 Base64 编码和解码，因为在网络传输中，图像数据常以 Base64 编码形式传递。
   * emotion、image\_generate、image\_keywords 和 link\_mysql 是自定义的模块，分别用于情感关键词转换、图像生成、关键词选择和与 MySQL 数据库交互。

**（二）创建 Flask 应用并启用 CORS**

收起

python

app = Flask(\_\_name\_\_)

# 允许所有来源访问

CORS(app)

1. **创建 Flask 应用实例**：Flask(\_\_name\_\_) 创建了一个 Flask 应用实例，\_\_name\_\_ 参数用于确定应用的根路径，以便 Flask 能够正确加载资源。
2. **启用 CORS**：CORS(app) 这行代码启用了跨域资源共享。在现代 Web 开发中，前端和后端可能部署在不同的域名下，启用 CORS 可以避免因跨域问题导致前端无法访问后端 API 的情况。

**（三）定义 /main 路由**

收起

python

@app.route('/main', methods=['POST'])

def main():

try:

# 从前端获取小朋友的输入

data = request.get\_json()

message = data.get('emo', "I'm happy")

print(message)

# 调用函数得到情感关键词

emo = emotion.emo\_transform(message)

print(emo)

# 将关键词放入词库中搜索

keywords\_list = image\_keywords.choose\_keywords(emo)

keywords = ','.join(keywords\_list)

print(keywords)

# 将关键词作为生图参数

img\_io = image\_generate.generate\_image(keywords)

print(img\_io)

if img\_io:

# 获取图像的二进制数据 存入 mysql

image\_data = img\_io.getvalue()

link\_mysql.save\_mysql(image\_data, message)

return send\_file(img\_io, mimetype='image/png')

else:

return jsonify({"error": "No images generated"}), 400

except Exception as e:

print(f"Error in main route: {e}")

return jsonify({"error": str(e)}), 500

1. **功能**：该路由接收前端发送的 POST 请求，根据请求中的情感文本生成对应的图像，并将图像保存到 MySQL 数据库中，最后返回生成的图像给前端。
2. **步骤**：
   * **获取情感文本**：使用 request.get\_json() 从请求中获取 JSON 格式的数据，然后通过 data.get('emo', "I'm happy") 获取情感文本。如果请求中没有提供 emo 字段，则使用默认值 "I'm happy"。
   * **生成情感关键词**：调用 emotion 模块中的 emo\_transform 函数，将获取到的情感文本转换为情感关键词，并打印输出。
   * **选择图像关键词**：将生成的情感关键词作为参数，调用 image\_keywords 模块中的 choose\_keywords 函数，从词库中选择相关的图像关键词。将选择的关键词列表转换为以逗号分隔的字符串形式，并打印输出。
   * **生成图像**：以生成的图像关键词字符串为参数，调用 image\_generate 模块中的 generate\_image 函数，生成图像。该函数返回图像的二进制流 img\_io，并打印输出。
   * **保存图像到数据库并返回图像**：如果成功生成图像（即 img\_io 不为空），通过 img\_io.getvalue() 获取图像的二进制数据，然后调用 link\_mysql 模块中的 save\_mysql 函数，将图像的二进制数据和原始情感文本保存到 MySQL 数据库中。最后，使用 send\_file 函数将生成的图像以 image/png 的 MIME 类型返回给前端。
   * **处理生成图像失败的情况**：如果生成图像失败（即 img\_io 为空），返回一个包含错误信息 "No images generated" 的 JSON 响应，并设置状态码为 400。
   * **异常处理**：如果在整个过程中发生任何异常，打印错误信息到控制台，并返回一个包含错误信息的 JSON 响应，设置状态码为 500。

**（四）定义 /history-images 路由**

收起

python

@app.route('/history-images', methods=['GET'])

def history\_images():

try:

images = link\_mysql.get\_history\_images()

return jsonify(images)

except Exception as e:

print(f"Error in history-images route: {e}")

return jsonify({"error": str(e)}), 500

1. **功能**：该路由接收前端发送的 GET 请求，从 MySQL 数据库中查询所有历史图像的信息，并以 JSON 格式返回给前端。
2. **步骤**：
   * **查询历史图像信息**：调用 link\_mysql 模块中的 get\_history\_images 函数，从数据库中获取所有历史图像的信息。
   * **返回查询结果**：将查询到的图像信息以 JSON 格式通过 jsonify 函数返回给前端。
   * **异常处理**：如果在查询过程中发生任何异常，打印错误信息到控制台，并返回一个包含错误信息的 JSON 响应，设置状态码为 500。

**（五）定义 /image/<int:image\_id> 路由**

收起

python

@app.route('/image/<int:image\_id>', methods=['GET'])

def get\_image(image\_id):

try:

image\_data = link\_mysql.get\_image\_by\_id(image\_id)

if image\_data:

img\_io = io.BytesIO(image\_data)

img\_io.seek(0)

return send\_file(img\_io, mimetype='image/png')

else:

return jsonify({"error": "Image not found"}), 404

except Exception as e:

print(f"Error in get\_image route: {e}")

return jsonify({"error": str(e)}), 500

1. **功能**：该路由接收前端发送的 GET 请求，根据图像的 ID 从 MySQL 数据库中查询对应的图像数据，并返回给前端。
2. **步骤**：
   * **获取图像 ID**：从 URL 中提取 <int:image\_id> 部分，作为要查询的图像的 ID。
   * **查询图像数据**：调用 link\_mysql 模块中的 get\_image\_by\_id 函数，根据图像 ID 从数据库中查询对应的图像数据。
   * **返回图像数据**：如果查询到图像数据，将其转换为 io.BytesIO 类型的二进制流 img\_io，并通过 img\_io.seek(0) 将流指针移动到起始位置。然后使用 send\_file 函数将图像以 image/png 的 MIME 类型返回给前端。
   * **处理图像未找到的情况**：如果未查询到图像数据，返回一个包含错误信息 "Image not found" 的 JSON 响应，并设置状态码为 404。
   * **异常处理**：如果在查询过程中发生任何异常，打印错误信息到控制台，并返回一个包含错误信息的 JSON 响应，设置状态码为 500。

**（六）定义 /save 路由**

收起

python

@app.route('/save', methods=['POST'])

def save\_image():

try:

data = request.get\_json()

image\_data = data.get('imageData')

image\_name = data.get('imageName') # 获取图片名称

if not image\_name:

return jsonify({"error": "Missing image name"}), 400

# 去除 data URI 前缀

image\_data = image\_data.split(',')[1]

# 解码 base64 数据

decoded\_image = base64.b64decode(image\_data)

# 保存到数据库，传递图片名称

link\_mysql.save\_edited\_image(decoded\_image, image\_name)

return jsonify({"message": "Image saved successfully"}), 200

except Exception as e:

print(f"Error in save\_image route: {e}")

return jsonify({"error": str(e)}), 500

1. **功能**：该路由接收前端发送的 POST 请求，将前端上传的图像数据保存到 MySQL 数据库中。
2. **步骤**：
   * **获取图像数据和名称**：使用 request.get\_json() 从请求中获取 JSON 格式的数据，然后通过 data.get('imageData') 获取图像的 Base64 编码数据，通过 data.get('imageName') 获取图像名称。
   * **检查图像名称**：如果没有获取到图像名称，返回一个包含错误信息 "Missing image name" 的 JSON 响应，并设置状态码为 400。
   * **处理图像数据**：去除 Base64 编码数据的前缀，通过 image\_data.split(',')[1] 实现。然后使用 base64.b64decode 函数对处理后的 Base64 数据进行解码，得到原始的图像二进制数据 decoded\_image。
   * **保存图像到数据库**：调用 link\_mysql 模块中的 save\_edited\_image 函数，将解码后的图像数据 decoded\_image 和图像名称 image\_name 保存到数据库中。
   * **返回保存成功信息**：如果保存成功，返回一个包含成功信息 "Image saved successfully" 的 JSON 响应，并设置状态码为 200。
   * **异常处理**：如果在保存过程中发生任何异常，打印错误信息到控制台，并返回一个包含错误信息的 JSON 响应，设置状态码为 500。

**（七）定义 /delete-image/<int:image\_id> 路由**

收起

python

@app.route('/delete-image/<int:image\_id>', methods=['DELETE'])

def delete\_image(image\_id):

try:

link\_mysql.delete\_image\_by\_id(image\_id)

return jsonify({"message": "Image deleted successfully"}), 200

except Exception as e:

print(f"Error in delete\_image route: {e}")

return jsonify({"error": str(e)}), 500

1. **功能**：该路由接收前端发送的 DELETE 请求，根据图像的 ID 从 MySQL 数据库中删除对应的图像数据。
2. **步骤**：
   * **获取图像 ID**：从 URL 中提取 <int:image\_id> 部分，作为要删除的图像的 ID。
   * **删除图像数据**：调用 link\_mysql 模块中的 delete\_image\_by\_id 函数，根据图像 ID 从数据库中删除对应的图像数据。
   * **返回删除成功信息**：如果删除成功，返回一个包含成功信息 "Image deleted successfully" 的 JSON 响应，并设置状态码为 200。
   * **异常处理**：如果在删除过程中发生任何异常，打印错误信息到控制台，并返回一个包含错误信息的 JSON 响应，设置状态码为 500。

**（八）启动 Flask 应用**

收起

python

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

1. **功能**：在脚本作为主程序运行时，启动 Flask 应用，并开启调试模式。
2. **说明**：if \_\_name\_\_ == '\_\_main\_\_': 这个条件判断确保只有当该脚本直接运行时，才会执行 app.run(debug=True) 启动应用。debug=True 开启了调试模式，在调试模式下，代码修改后应用会自动重启，方便开发和调试过程中快速看到代码修改的效果。同时，调试模式还会提供详细的错误信息，有助于排查问题。

**三、自定义模块功能**

1. **emotion 模块**：包含 emo\_transform 函数，用于将情感文本转换为情感关键词。
2. **image\_generate 模块**：包含 generate\_image 函数，用于根据图像关键词生成图像，并返回图像的二进制流。
3. **image\_keywords 模块**：包含 choose\_keywords 函数，用于根据情感关键词从词库中选择相关的图像关键词。
4. **link\_mysql 模块**：包含 save\_mysql、get\_history\_images、get\_image\_by\_id、save\_edited\_image 和 delete\_image\_by\_id 等函数，用于与 MySQL 数据库进行交互，实现图像数据的保存、查询和删除等操作。