1、系统概述

1.1、系统简介

(1) 项目背景及基本情况介绍

1. 时间，这位无声无息的存在，自宇宙诞生之初便存在，见证了无数变迁。人类生命虽短暂，却拥有无比珍贵的东西。时间无情地带走了我们的体力与记忆，却加深了对故人的思念。因此，我们应珍惜与亲人朋友相处的每一刻，并学会释放过去的遗憾和痛苦。
2. 修复老照片系统是保存珍贵记忆的重要工具。随着时间的推移，老照片逐渐模糊和受损。该系统通过先进的数字图像处理技术，为这些记忆提供了修复、保护和传承的可能。它能够去除污渍、划痕和褪色，修复破损的边缘和细节，让照片重现光彩。这不仅是对美好瞬间的延续，也是对家族文化传统的传承。
3. 修复老照片系统的重要性不仅在于技术价值，更在于人文意义。通过修复这些照片，我们能更好地缅怀过去，回忆与亲人共度的时光。这些照片见证了我们的成长、家庭的变迁和时代的演进，是我们情感连接的纽带和文化认同的基石。
4. 因此，修复老照片系统不仅是对技术的运用，更是对人文精神的传承和发扬。它让我们在时间的长河中留住珍贵的过往与情感，让记忆得以永恒。通过这个系统，我们可以更好地纪念过去，陶冶心灵，为未来的生活留下更多宝贵的回忆。
5. 系统开发的过程中，我们贯彻的是模块化的编程思想，首先是前端代码和后端代码的分离，对于后端主要分为两个模块一个静态修复模块，一个是动态修复模块，将训练模型，调用，接口等文件解耦合，这种设计模式有助于提高代码的易维护性和易拓展性。具体实现的过程中，我们调用了腾讯和百度两个公司的大模型，采用超分算法用于处理图片，前后端连接采用 Flask 架构，前端使用 Vue 框架搭建网站。

1.2、术语表

序号	术语或缩略语	说明性定义
1	Flask 框架	1. Flask 框架的全称是"A Python Microframework based on Werkzeug and Jinja2"，通常简称为 Flask。它是一个用 Python 编写的轻量级 Web 应用框架。Flask 的设计哲学是提供一个简单、易扩展的核心，并允许开发者根据自己的需求添加其他功能。 2. Flask 基于两个主要库: (1) Werkzeug：一个 WSGI（Web Server Gateway Interface）工具库，用于处理网络请求和响应 (2) Jinja2：一个现代且设计友好的模板引擎，用于生成 HTML 等文本输出。 3. Flask 因其简洁、灵活和易于上手而受到许多 Python 开发者的喜爱，特别适合用于快速开发小型到中型的 Web 应用。
2	HTTP 响应	1. HTTP 响应（HTTP Response）是指当客户端（如 Web 浏览器）向服务器发送 HTTP 请求后，服务器返回给

		客户端的应答信息。HTTP 响应包含了服务器对请求的处理结果，以及可能伴随的数据或状态指示。 2. HTTP 响应主要包含以下几个部分，状态行（Status Line），响应头（Response Headers），空行（Blank Line），响应体（Response Body）。
3	@app.route	Flask 的装饰器，用于定义路由（URL 路径）和处理该路径的函数。
4	RESTful API	(Representational State Transfer Application Programming Interface)，是一种设计风格，用于构建可扩展的 Web 服务。REST 是一种架构风格，它定义了一组用于创建 Web 应用程序的约束。当一个服务遵循这些约束时，它可以被称为是 RESTful 的。
5	cors	跨域资源共享（Cross-Origin Resource Sharing），是一种机制，允许在某些受限的情况下，不同域之间的网页脚本可以相互访问资源。
6	Vue 框架	1.Vue 框架是一款轻量且强大的 JavaScript 框架，专注于构建用户界面。它采用声明式编程和组件化开发，使开发者能高效构建复杂的前端应用。Vue 的响应式系统能自动跟踪数据变化并更新视图，简化开发流程。 2. Vue.js（通常简称 Vue）是一个用于构建用户界面的渐进式 JavaScript 框架。它由前谷歌工程师尤雨溪（Evan You）创建，并于 2014 年发布。Vue 的设计目标是采用自底向上增量开发的设计，使得它既可以用于简单的网页交互，也可以驱动复杂的单页应用程序（SPA）。 3.Vue.js 的核心库专注于视图层，并且非常容易学习，同时也便于与第三方库或现有项目集成。Vue 还提供了配套工具，如 Vuex 用于状态管理，Vue Router 用于路由管理，以及 Vue CLI 用于快速搭建项目脚手架。
7	Mysql	基于 SQL 查询的开源跨平台数据库管理系统)，在我们的系统中，用于存储用户账号中的各种信息，并计划将其部署到云端服务器上便于远程访问。
8	GFPGAN	1. 腾讯在人像复原、超分等方面的开源模型，我们的系统准备利用它的照片超分功能实现老照片的修复功能。 2. GFP-GAN （ Generative Facial Prior-Generative Adversarial Network）是一种先进的图像处理技术，特别是在人脸图像增强和修复领域。这个技术结合了生成对抗网络（GAN）和人脸先验知识，以实现高质量的人脸图像恢复。 3. GAN 是一种深度学习模型，由两个主要部分组成：生成器（Generator）和判别器（Discriminator）。生成器的任务是生成与真实数据相似的样本，而判别器的任务是区分生成的样本和真实样本。在训练过程中，生成器和判别器相互竞争，生成器试图生成

		更真实的样本以欺骗判别器，而判别器则试图提高其区分真伪的能力。这种竞争过程最终导致生成器能够生成高质量的图像。 4. GFP-GAN 的关键创新在于它利用了人脸的先验知识，这些知识通常来自于大量的人脸数据集，包含了人脸的结构、特征点、表情等信息。这些先验知识帮助生成器更好地理解和重建人脸图像，即使在输入图像质量较差或部分缺失的情况下也能产生逼真的结果。
9	PaddleGAN	1. 百度生成对抗网络开发的模型套件，包括多种功能，包括但不限于超分、插帧、上色、换妆、图像动画生成等功能，我们打算利用其中的动作迁移模型（first-order-demo）实现照片动起来的功能. 2. PaddleGAN 是百度开源的一个基于飞桨（PaddlePaddle）深度学习平台的生成对抗网络（GAN）工具库。它提供了一系列的 GAN 模型和相关的工具，用于图像和视频的生成、编辑、修复等任务。PaddleGAN 旨在简化 GAN 模型的开发和部署，使得研究人员和开发者能够更容易地利用 GAN 技术进行各种创意和应用开发。
10	Json	JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成。它基于 JavaScript 语言的一个子集，但已经成为一种独立于语言的数据格式，广泛用于网络通信和数据存储。
11	Element Plus	Element Plus 是一个基于 Vue.js 的组件库，它提供了一套丰富的 UI 组件，用于帮助开发者快速构建美观、易用的 Web 应用程序。它是 Element UI 的升级版本，专为 Vue 3 设计，同时也支持 Vue 2。Element Plus 继承了 Element UI 的设计理念和组件风格，同时进行了性能优化和功能增强。

1.3、系统运行环境

操作系统	Windows 64 位操作系统
数据库系统	MySQL 数据库
编程平台	本地使用 Pycharm WebStorm Vscode 进行编程
网络协议	TCP 传输控制协议 ,HTTP 协议， post get 请求
硬件平台	Dell G15 5510 笔记本电脑, WD Elements SE 固态存储硬盘

1.4、开发环境

工具软件	PyCharm 2019.3.3 x64 WebStorm 2023.1 Vscode1.88.0
开发语言	Python3.7 .Vue3.2.37 .Web 开发(HTML5 CSS3) MySQL8.0
模块(库)版本	后端: tqdm~=4.66.2 PyYAML==5.3 scikit-image~=0.20.0 scipy~=1.9.1 opencv-python==4.9.0.80 imageio==2.6.1 imageio-ffmpeg==0.4.9 librosa==0.8.1 numba==0.48.0 easydict==1.12 munch==4.0.0 natsort==8.4.0 matplotlib==3.1.3 basicsr==1.4.2 facexlib==0.3.0 lmdb==1.4.1 numpy~=1.24.4 tb-nightly==2.12.0a20230113 torch==1.13.1 torchvision==0.14.1 yapf==0.28.0 前端: @element-plus/icons-vue 2.0.9 axios 0.27.2 body-parser 1.20.2 cors 2.8.5 echarts 5.5.0 express 4.19.2 jsonwebtoken 9.0.2 md-editor-v3 2.2.1 pinia 2.0.20 vue: ^3.2.37, vue-cropperjs: ^5.0.0, vue-router: ^4.3.0, vue-schart: ^2.0.0, vuex: ^4.1.0, wangeditor: ^4.7.15, xlsx: ^0.18.5

2、数据结构说明

2.1、常量

网页前端 server 文件夹中：

1. db：位于 index.js 文件中，MySQL 操作对象，const db = mysql.createPool，连接数据库，并使用数据库的 query 函数执行 sql 语句
2. sql，sql1，sql2：位于 login.js,，变量类型字符串文件中，定义为要执行的 sql 语句

网页前端 src 文件夹中：

1. router：位于 router 文件夹下 index 文件中，用于导出，规定界面路由，规定网页跳转
2. baseURL：位于 main.js 文件中，定义连接端口，用于前后端的连接

后端常量：

位于 app.py 文件中：

1. UPLOAD_FOLDER：字符串类型，定义上传照片路径，用户上传的图片保存到相应的文件夹
2. project_root_path：字符串类型，定义当前文件所在的目录，用于确定结果的保存路径
3. current_file_path：字符串类型，定义当前文件的绝对路径
4. current_selectimage：字符串类型，定义修复好的照片路径，根据路径上传结果到前端显示
5. model_name：位于 inference-gfpgan.py 文件中，字符串类型，定义选用的修复模型。

位于 fisrt-oder-demo.py 文件中：

1. UPLOAD_FOLDER：字符串类型，定义用户上传图片到后端的路径
2. result_file_path：字符串类型，定义生成结果动态太图片的地址，用以传到前端

2.2、变量

前端变量：

1. radioValue.value：位于 uploadFile_Dynamic.vue 文件夹中，变量类型 int，接收单选框的值，并把它传入后端，根据不同的 value 值选择不同的驱动视频,，做出不同的表情。
2. fullscreenLoading.value：位于 uploadFile_static_state.vue 文件夹中，bool 型变量，从后端上传结果，上传成功为 true，上传失败为 false
3. username：位于 login.vue 文件中，字符串类型，接受用户的输入用户名，在下方函数中传入数据库或匹配
4. password：位于 login.vue 文件中，字符串类型，接受用户的输入的密码，在下方函数中传入数据库或匹配
5. confirmpassword：位于 login.vue 文件中，字符串类型，接受用户的输入，注册时重复输入密码，并验证密码是否一致

后端变量：

1. value：位于 fisrt-oder-demo.py 文件中，为 int 类型，作全局变量接收前端传入的 value 值。
2. DRIVING_VIDEO：位于 fisrt-oder-demo.py 文件中，字符串类型，更具前端传入的 value 值，选着不同驱动视频的路径，实现不同的表情。
3. result_base64：位于 fisrt-oder-demo.py 文件中，获取生成动态照片的视频，并将其

转换为 base64 的编码形式，并将该编码形式的视频通过 json 传入前端。

4. relative：位于 fisrt-oder-demo.py 文件中，bool 类型，表示是否采用相对坐标识别人脸位置，使生成视频动作迁移更加合理。
5. face_enhancement：位于 fisrt-oder-demo.py 文件中，bool 类型，表示是否脸部效果加强，使脸部动作细节跟丰富。
6. filename：位于 app.py 文件中，字符串类型，用于接收前端上传图片的文件名

2.3、数据结构

包括数据结构名称，功能说明，具体数据结构说明（定义、注释、取值）等

- app.py 文件中

在这段代码中，主要涉及的数据结构包括字典（Dictionary）和字符串（String）。下面是对这些数据结构的分析：

一.字典（Dictionary）：

（1）功能说明: 字典用于存储键值对，其中键通常是唯一的，值则可以重复。在这段代码中，字典主要用于存储配置信息和请求参数。

（2）具体数据结构说明:

1.在 app.config 中，字典用于存储应用程序的配置信息，如上传文件夹的路径、当前选择的图像路径等。例如：

```
app.config['UPLOAD_FOLDER'] = '/static/images/uploads'
app.config['current_selectimage'] = os.path.join(project_root_path,output_path, "restored_imgs")
```

2.在请求处理函数中，字典用于获取请求的参数，如 request.args 用于获取 GET 请求的参数。例如：

```
input_path = request.args.get('input')
```

二.字符串（String）：

(1) 功能说明: 字符串用于存储和操作文本数据。在这段代码中，字符串主要用于构建文件路径、URL 和文件名。

(2) 具体数据结构说明:

1.文件路径的构建，如：

```
file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
```

2.URL 的构建，如：

```
result_image = ''.join([
'http://127.0.0.1:5000/',
handle_res[0],])
```

3.文件名的生成，使用了时间戳和原始文件名的一部分，如：

```
filenam=secure_filename(file.filename).split('.')[0]+"_"+str(int(time.time()))+'.'+secure_filename(file.filename).split('.)[0]
```

三.列表（List）：

(1)功能说明:列表用于存储多个元素，这些元素可以是不同的数据类型。在这段代码中，列表主要用于存储多个字符串元素，以便进行字符串拼接。

(2)具体数据结构说明:

1.在字符串拼接时，使用了列表来存储多个字符串片段，如：

```
result_image = ''.join([
    'http://127.0.0.1:5000/',
```

handle_res[0],])

四.元组（Tuple）

(1)功能说明:元组用于存储多个元素，这些元素可以是不同的数据类型，但元组一旦创建后其元素不可更改。在这段代码中，元组用于返回多个值。

(2)具体数据结构说明:

1.在函数`image_handle`的调用结果中，可能返回一个元组，如：

```
handle_res = image_handle({
  'input': input_path_full,
  'output': 'static/images/results', })
```

这些数据结构在代码中被灵活运用，以实现文件上传、图像处理和结果返回等功能。

- first-order-demo.py 文件中

  本段代码中涉及到的数据结构有全局变量 (value)，Flask 应用 (app)，路由装饰器 (@app.route)，请求对象 (request)，响应对象 (jsonify)，argparse.ArgumentParser，字典 (data)，文件对象 (file)，字符串 (default_filepath, result_base64, result_json)，列表，FirstOrderPredictor，配置参数 (args)，JSON 对象，Base64 编码。

一. Flask 应用实例 (app)

功能说明: 作为 Web 服务器的核心，处理 HTTP 请求和响应。

定义: app = Flask(__name__)

注释: 创建 Flask 应用实例。

取值: __name__ 是当前 Python 脚本的名称。

二.CORS (CORS(app))

功能说明: 启用跨源资源共享，允许不同源的 Web 页面调用 API。

定义: CORS(app)

注释: 应用跨源资源共享中间件。

取值: 无特定取值，作用于整个 Flask 应用。

三.全局变量 (value)

功能说明: 存储从客户端接收的值。

定义: value=0

注释: 初始化全局变量 value。

取值: 通过 POST 请求的 JSON 体更新其值。

四.路由装饰器 (@app.route)

功能说明: 定义路由，指定 URL 和处理函数。

定义: @app.route('/value', methods=['POST'])

注释: 为'/value'路径设置一个 POST 请求处理器。

取值: 路径字符串和请求方法列表。

五.请求对象 (request)

功能说明: 包含请求的信息，如 headers、form data、files 等。

定义: data = request.json 和 file = request.files['file']

注释: 从请求中获取 JSON 数据和文件。

取值: 请求中的数据和文件。

六.JSON 响应 (jsonify)

功能说明: 创建 JSON 格式的响应。

定义: return jsonify({'message': 'Received selected value: {}'.format(value)})

注释: 将 Python 字典转换为 JSON 格式并返回。

取值: 字典对象，包含要发送的消息。

七.argparse.ArgumentParser (parser)
功能说明: 解析命令行参数。
定义: parser = argparse.ArgumentParser()
注释: 创建一个新的 ArgumentParser 对象用于添加和解析命令行参数。
取值: 通过 add_argument 方法添加的参数。

八.FirstOrderPredictor (predictor)
功能说明: PPGAN 库中的类，用于执行视频生成任务。
定义: predictor = FirstOrderPredictor(...)
注释: 初始化 FirstOrderPredictor 对象，设置视频生成的各种参数。
取值: 根据构造函数参数定义的配置。

九.os 模块
功能说明: 与操作系统交互，如文件路径操作。
定义: os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
注释: 用于拼接文件路径。
取值: 文件的保存路径。

十.base64 模块
功能说明: 对二进制数据进行编码和解码。
定义: result_base64 = 'data:video/mp4;base64,' base64.encode(result_data).decode('utf-8')
注释: 将读取的文件内容编码为 base64 字符串。
取值: base64 编码后的字符串。

十一.json 模块
功能说明: 处理 JSON 数据格式。
定义: result_json = json.dumps({"result_base64": result_base64})
注释: 将包含 base64 编码视频的字典转换为 JSON 格式的字符串。
取值: JSON 格式的字符串。

十二.文件操作 (open, file.read(), file.save())
功能说明: 对文件进行读取、写入和保存操作。
定义: with open(result_file_path, 'rb') as file: result_data = file.read()
注释: 打开文件，读取内容，然后关闭文件。
取值: 文件路径和读取的文件数据。
这些数据结构和模块共同协作，使得 Flask 应用能够接收上传的文件，处理视频生成任务，并将结果以 JSON 格式返回给客户端。

- inference_gfpgan.py 文件

这个文件中主要包含了一个 Python 脚本，主要用于图像恢复任务，特别是针对人脸恢复。
数据结构名称
ArgumentParser: 用于解析命令行参数的类。
GFPGANer: 一个用于图像恢复的类，具体实现未在文档中给出。
RRDBNet: 一个用于背景恢复的网络结构，具体实现未在文档中给出。
RealESRGANer: 一个用于背景恢复的类，使用 RealESRGAN 模型。
imwrite: 一个用于保存图像的函数。

一.ArgumentParser:
add_argument: 用于添加命令行参数的函数，如-i（输入图像或文件夹）、-o（输出文件夹）、-v（GFPGAN 模型版本）、-s（图像放大倍数）、--bg_upsampler（背景恢复器）、--bg_tile

（背景采样器的瓦片大小）、--suffix（恢复人脸的后缀）、--ext（图像扩展名）等。
parse_args: 解析命令行参数并返回一个包含参数对象的元组。

二.GFPGANer:
model_path: 模型文件的路径。
upscale: 图像放大倍数。
arch: 架构类型。
channel_multiplier: 通道乘数。
bg_upsampler: 背景恢复器。

三.RRDBNet:
num_in_ch: 输入通道数。
num_out_ch: 输出通道数。
num_feat: 特征图数量。
num_block: 块的数量。
num_grow_ch: 增长通道数。
scale: 模型缩放比例。

四.RealESRGANer:
scale: 模型缩放比例。
model_path: 模型文件的 URL。
model: RRDBNet 模型。
tile: 是否在测试时使用瓦片。
tile_pad: 瓦片填充。
pre_pad: 预填充。
half: 是否使用半精度。

五.imwrite:
save_crop_path: 裁剪后人脸的保存路径。
save_face_name: 恢复后人脸的保存路径。
save_restore_path: 恢复图像的保存路径。
cmp_img: 裁剪和恢复图像的拼接。

- uploadFile_static_state.vue 文件

query ：一个包含查询参数的对象，包括 curPage （当前页码）、 pageSize （每页大小）、 tableName （表名）和 keyword （关键字）。用于存储查询参数，通过 reactive 函数创建响应式对象，使得查询参数能够在组件的不同部分进行响应式更新。

pageTotal ：一个表示总页数的计数器。用于记录总页数，初始化为 0。

fullscreenLoading ：一个表示是否全屏加载的标志。用于控制全屏加载的显示，通过 ref 函数创建引用，使得可以在组件中直接操作该标志。

url ：一个表示图片路径的变量。用于存储图片的路径，初始化为默认的图片路径。

srcList ：一个包含图片预览源的列表。用于存储图片的预览源列表，初始化为包含默认预览源的列表。

- uploadFile_Dynamic.vue 文件

query ：一个包含查询参数的对象。它具有属性 curPage （当前页码）、 pageSize （每页大小）、 tableName （表名）和 keyword （关键字）。

videoValue ：一个引用类型，初始化为 null 。用于存储视频的数据源。

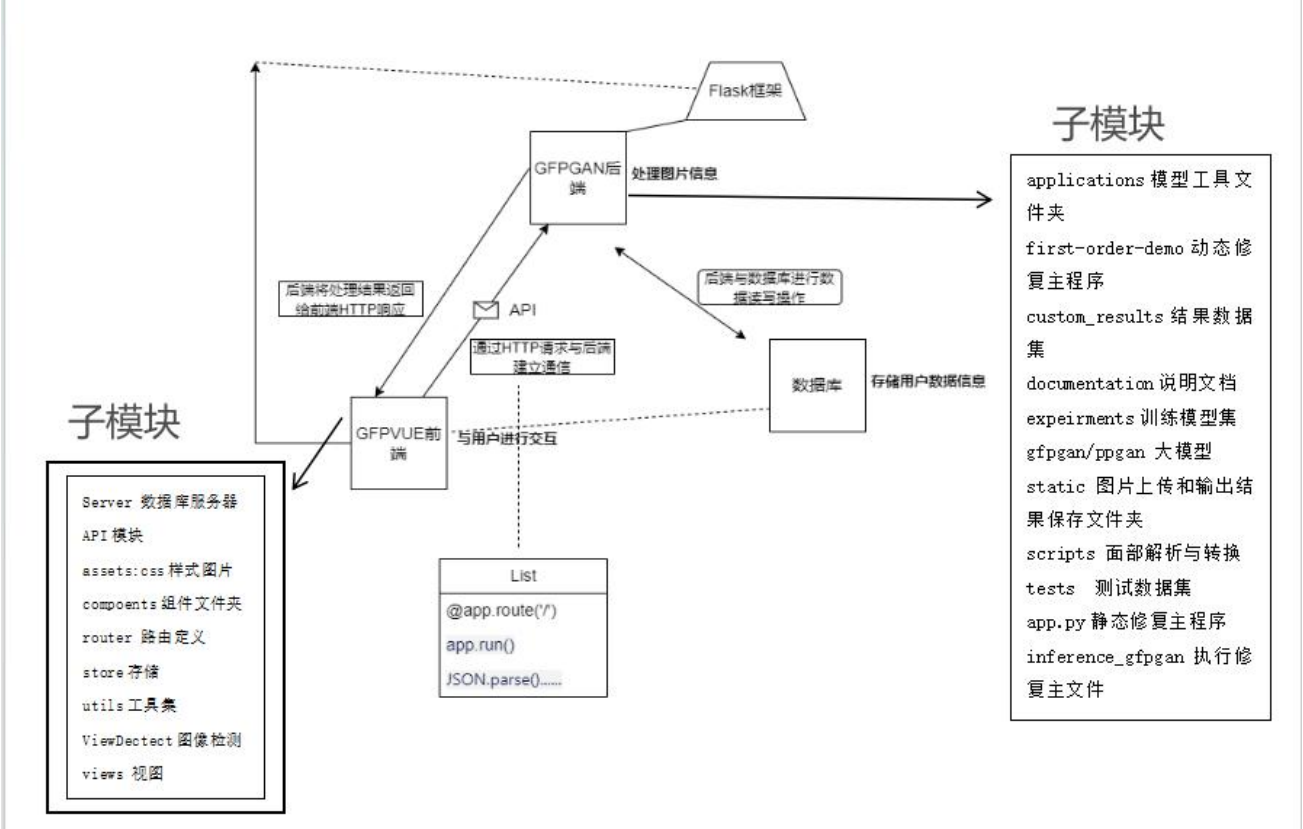
pageTotal ：一个数字类型的引用，初始化为 0 。用于记录总页码数。

fullscreenLoading ：一个布尔类型的引用，初始化为 false 。用于控制全屏加载状态。

radio ： 一个引用类型，初始化为 3 。用于单选框的选中值。
radioValue ： 一个数字类型的引用，初始化为 null 。用于存储选中的单选框值。
response ： 一个对象类型的引用。用于存储后端返回的响应数据。
err ： 一个对象类型的引用。用于存储上传文件时发生的错误信息。
fileList ： 一个数组类型的引用。用于存储上传的文件列表。
videoPlayer ： 一个引用类型，用于视频播放器组件。
url ： 一个字符串类型的引用，初始化为 '../src\assets\img\null.png'。用于存储图片的默认链接。
srcList ： 一个数组类型的引用，初始化为 ['http://localhost:5000/img/10045_1710698433.png', '']。用于存储图片的源列表。

3、 模块设计

3.1、软件结构



3.2、功能设计说明

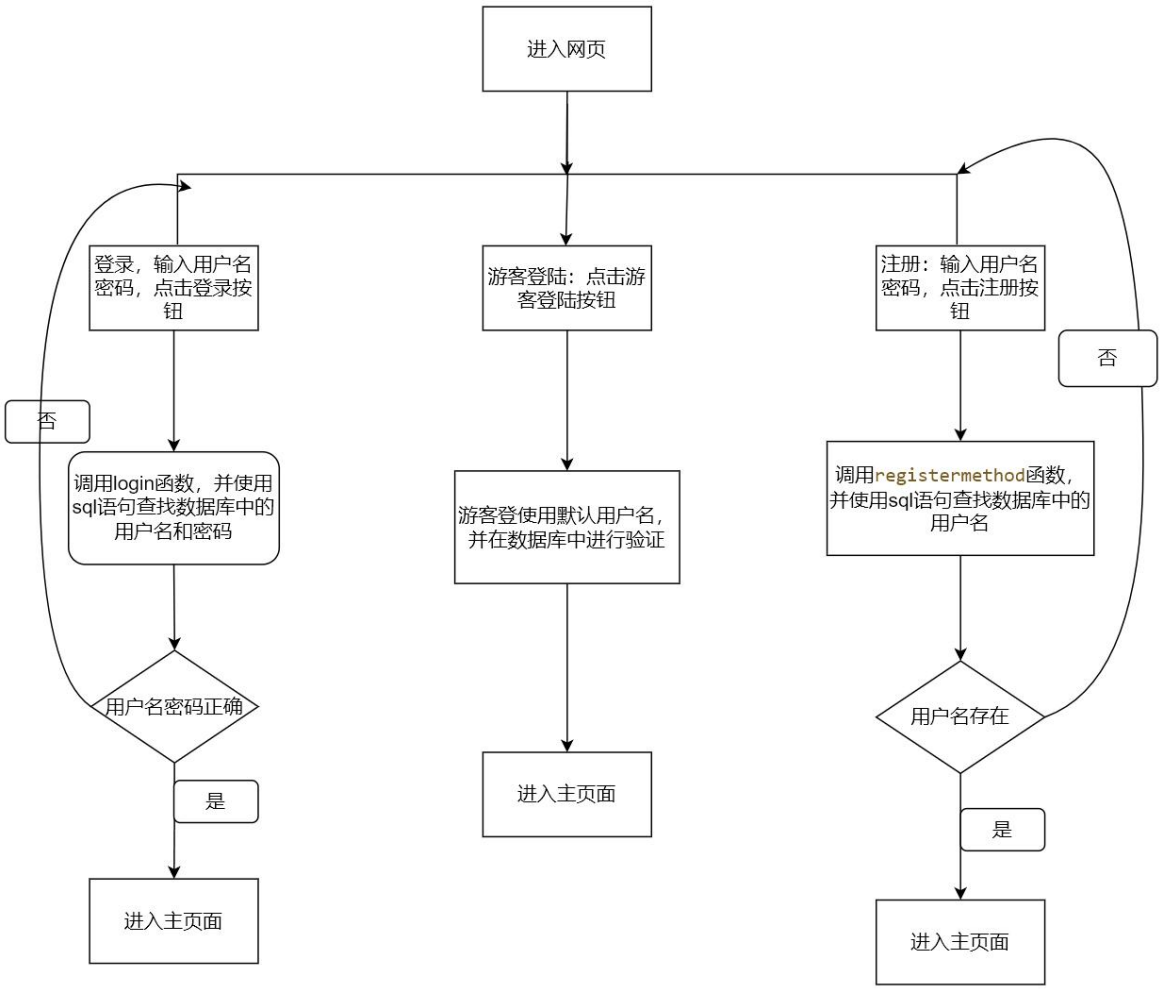
1. 首先我们将 GFPGAN 后端 数据库部分和 GFPVUE 前端互相解耦合，这样使代码具有更高的可维护性和可拓展性，便于开发。
2. 对于后端，该软件基于 Flask 框架构建，通过采用这个轻量级且易于扩展的 web 框架，可以实现高效的 web 应用开发。Flask 框架的灵活性使得开发者能够根据需要定制和扩

展软件功能，满足特定需求。

3. 其次，该软件在设计中强调数据的处理和存储。后端服务器与数据库进行数据读写操作，确保数据的准确性和一致性。在处理 GFPGAN 生成的图像信息时，后端服务器进行必要的后处理，并将处理结果返回给前端。这种设计思想使得软件能够高效地处理大量图像数据，并为用户提供高质量的图像输出。
4. 此外，该软件还注重前后端的交互和通信。前端通过 HTTP 请求与后端建立通信，获取处理后的结果。同时，前端还使用 GFPVUE 与用户进行交互，接收用户的输入或输出数据。这种设计使得软件能够实时响应用户的操作，提供流畅的用户体验。
5. 最后，软件在整体架构上采用模块化的设计思路。通过使用@app.route(...)装饰器定义路由和 URL 模式，使得不同的功能模块可以独立开发和维护。同时，app.run()命令用于启动应用程序，简化了软件的部署和启动过程。

3.3、模块 1 登录注册模块

3.3.1、设计图



3.3.2、功能描述

本模块主要实现用户登录，注册，以及游客登陆功能的实现，以及与数据库的连接。

3.3.3、输入数据

用户名：username，校验规则：不能为空，不能超过 50 个字符

密码：password，校验规则：不能为空

密码验证：confirmpassword，校验规则：与密码一致

3.3.4、输出数据

无返回值，直接调用请求业务相应的处理函数

3.3.5、数据设计

string 可以存储 IP 地址、服务器名、消息，以及 sql 语句等字符串，用户名和密码都一字符形式储存到数据库里。

json：一个开源的 JSON 库，用于进行字符串和 JSON 对象之间的转换。

3.3.6、算法和流程

三个不同的按键（button）绑定不同的三个函数，登录：loginmethod；注册：registermethod；游客登陆：youke。用户再输入完用户名密码后，通过 form 组件绑定的事件，获取用户名和密码，点击登录按钮，调用 loginmethod 函数，使用 axios.post('/login'）方法，访问后端'/login'接口，并把用户输入的用户名和密码传入后端，通过这个接口调用后端的 login 函数，然后查找数据库里的值，对比用户名和密码。游客登陆也是通过这样的访问后端接口'/login'，只不过传入的用户名和密码为默认值，以此实现快速登录。点击注册按钮，调用 registermethod 函数，使用 axios.post('/register'）方法，访问后端接口'/register'，并把用户名和密码传入后端，通过接口调用 register 函数，调用 sql 语句将用户名和密码存入 MySQL 数据库。

3.3.7、函数说明

1. login 函数：位于 server 文件夹 login.js 文件中，功能：调用函数 db.query(sql)执行 sql 语句，查找用户名和密码，并返回结果；参数 req：前端请求传入的参数（用户名密码）, res（调用成功返回的结果）；返回值：调用函数 db.query(sql)失败，返回值 res 返回：status: 400, message: "登录失败"；函数 db.query(sql)查找成功，返回值 res 返回：status: 200, message: "登录成功"；函数 db.query(sql)查找失败，返回值 res 返回：status: 202, message: "用户名密码错误"
2. register 函数：位于 server 文件夹 login.js 文件中，功能：先调用函数 db.query(sql)判断用户名是否重复（即查找用户名），用户名没有重复，再调用函数 db.query(sql)存储用户名和密码，并返回结果；参数 req：前端请求传入的参数（用户名密码）, res（调用成功返回的结果）；返回值：调用函数 db.query(sql)失败，返回值 res 返回：status: 500, message: "操作失败"；函数 db.query(sql)查找成功，用户名已存在，返回值 res 返回：status: 202, message: "用户名已存在"；函数 db.query(sql)查找失败，调用函数 db.query(sql)存储，调用失败，返回值 res 返回：status: 400, message: "注册失败"，调用成功，返回值 res 返回：status: 200, message: "注册成功"。
3. 函数 loginmethod：位于 src 文件夹下 views 文件夹下的 uploadFile_Dynamic.vue 文件夹中，功能：通过'/login'调用后端 login 函数验证用户名密码是否正确，并根据 login 返回值 res 中的 status 的值抛出提示信息。
4. 函数 registermethod：位于 src 文件夹下 views 文件夹下的 uploadFile_Dynamic.vue 文件夹中，功能：通过'/register'调用后端 login 函数验证用户名密码是否正确，并根据 register 返回值 res 中的 status 的值抛出提示信息。
5. 函数 youke：位于 src 文件夹下 views 文件夹下的 uploadFile_Dynamic.vue 文件夹中，通过'/login'调用后端 login 函数验证用户名密码是否正确，并根据 login 返回值 res

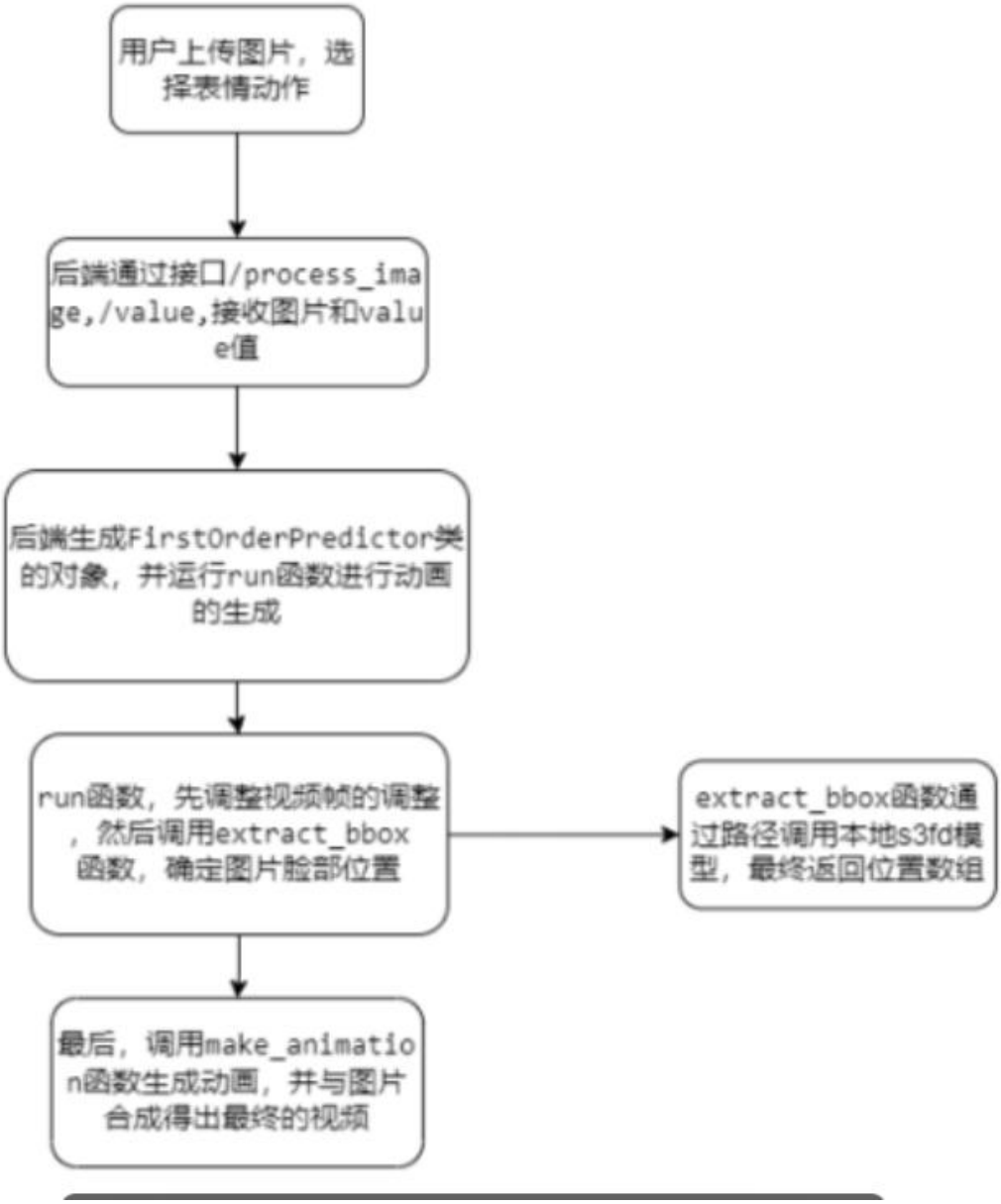
中的 status 的值抛出提示信息。

3.3.8 全局数据结构与该模块的关系

访问了全局 username，password 等字符串数据

3.4、模块 2 动态处理模块

3.4.1、设计图



3.4.2、功能描述

结合前端上传操作，后端根据用户的选择，对用户上传的图片进行不同的动态处理，生成不同的表情。

3.4.3、输入数据

1. value：与单选组件 radio 相连接，根据不同的选项，获得不同的 value 值

对于上传的 value 值，代码会进行以处理：
接口方式： '/value', methods=['POST']，后端接口以及请求方式
引入全局变量：global value
定义变量 data 获取 json 变量： request.json。
获取传输值：value = data.get('selectedValue')

2. 图片：用户上传与需要动态处理的的图片

对于上传的图片文件，代码会进行以下有效性检验：
接口方式：'/process_image', methods=["GET",'POST']
引入全局变量：global value
文件部分： 'file' not in request.files 确保请求中包含文件部分。
文件名： file.filename == '' 检查文件名是否为空。
文件名不为空，对于 '/process_image'，接口中的图片路径，将直接存入 UPLOAD_FOLDER 指向的文件夹中

3.4.4、输出数据

输出根据上传图片和 value 值生成的动态视频。

定义结果保存的路径 result_file_path，
通过 base64.b64encode，将视频重新编码，
生成 json 字典，并发往前端，json.dumps({"result_base64": result_base64})

3.4.5、数据设计

json：一个开源的 JSON 库，用于进行字符串和 JSON 对象之间的转换。

app：Flask 应用程序实例，用于处理 HTTP 请求和响应。

request：Flask 的请求对象，包含客户端发送的请求信息，如请求方法、URL、表单数据等。

jsonify：用于将 Python 对象转换为 JSON 格式的响应。

make_response：用于创建 Flask 的响应对象。

DRIVING_VIDEO：驱动视频文件保存地址

UPLOAD_FOLDER：上传文件路径保存地址

bboxes：作为 extract_bbox 方法的返回值，该方法检测源图像中的所有人脸，并返回一个边界框（bounding boxes）的数组 bboxes，每个边界框是一个包含五个元素的数组 [x1, y1, x2, y2, area]，其中 (x1, y1) 是边界框左上角的坐标，(x2, y2) 是右下角的坐标，area 是边界框的面积。

3.4.6、算法和流程

用户通过前端动态处理界面上传图片，并选择想要生成的表情，前端通过接口/process_image 将图片传输到后端，通过/value 接口将用户选择对应的选择的 value 值，后端通过接口接收到图片和 value 值，后端会通过 value 值选择驱动视频的地址的 DRIVING_VIDEO 以此进行图片动态地处理。通过 /process_image 接口生成 FirstOrderPredictor 类的对象，调用其中的 run 函数进行图片动画的生成。run 函数首先通过

cv2 模块读取视频，并改变视频帧的大小，使得其与图片大小归一化处理，然后调用 extract_bbox 函数检测人脸，并返回人脸所在区域边框的数组，然后再调用 make_animation 函数生成表情动画，然后对于驱动视频中的每一帧，根据源图像和生成的动画帧合成最终的视频帧。最后将合成好的视频保存到 result_file_path 结果保存路径里。然后后端通过 with open 函数访问结果视频，并使用 base64.b64encode 修改视频编码形式，一 json 字典的形式将结果视频传回前端网页并显示。

3.4.7、函数说明

具体说明模块中的各个函数，包括函数名称及其所在文件，功能，格式，参数，全局变量，局部变量，返回值，算法说明，使用约束等。

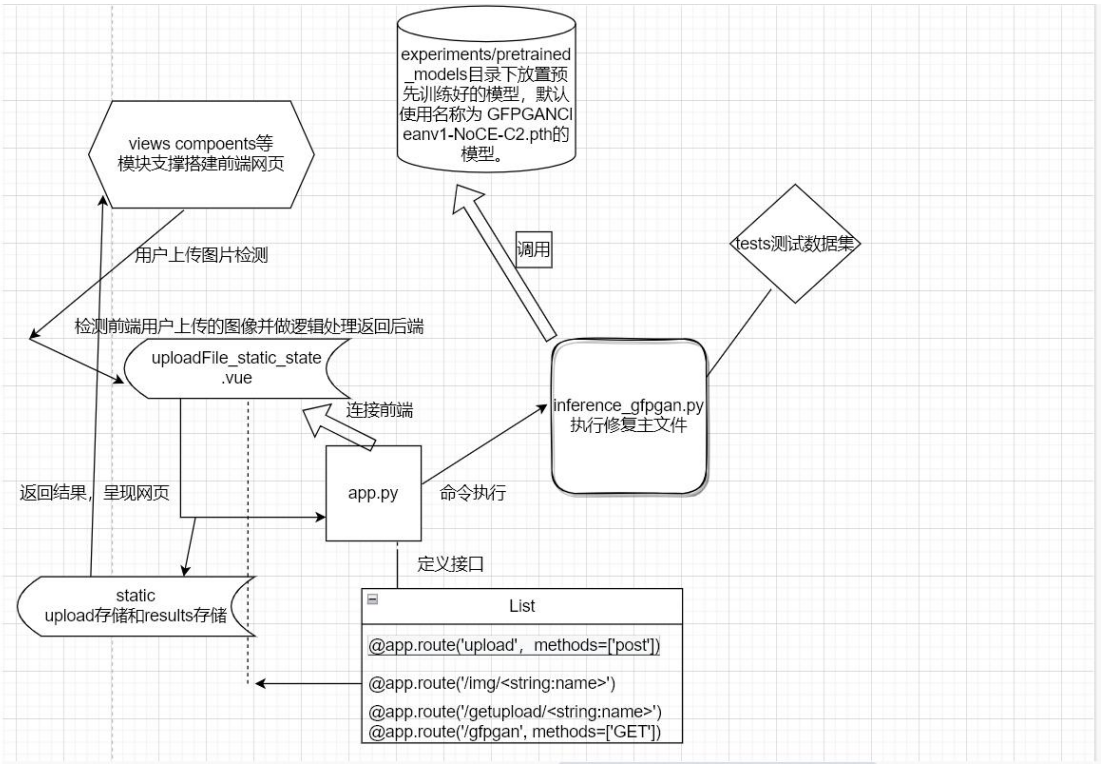
1. run 函数：位于 ppgan 文件夹中的 first_order_predicter 文件中，功能：调用其他函数模块，根据驱动视频生成图片的动态处理动画。参数：source_image（用户上传的图片）, driving_video（用户选的表情对应的驱动视频），无返回值。
2. get_prediction 函数：位于 ppgan 文件夹中的 first_order_predicter 文件中，功能：调用函数 make_animation 生成图片动态处理的动画。参数：face_image：通过函数 extract_bbox 检测人脸后返回的人脸边界框数组，从源图像中裁剪出来的包含单个面孔的图像区域，返回值为：所有生成的动画帧合并成一个 NumPy 数组
3. extract_bbox 函数：位于 ppgan 文件夹中的 first_order_predicter 文件中，功能：识别人脸，检测人脸个数，并返回人脸所在区域的边框。参数：image（用户传输图片）；返回值：边界框（bounding boxes）的数组 bboxes，每个边界框是一个包含五个元素的数组 [x1, y1, x2, y2, area]，其中 (x1, y1) 是边界框左上角的坐标，(x2, y2) 是右下角的坐标，area 是边界框的面积。算法原理：生成类 FaceAlignment 的对象，通过调用类 FaceAlignment 的 get_detections_for_batch 方法调用类 SFDDetector 的方法，使用 s3fd 模型，进行人脸位置的识别
4. load_checkpoints 函数：位于 ppgan 文件夹中的 first_order_predicter 文件中，功能：load_checkpoints 方法负责根据提供的配置和权重文件初始化 OcclusionAwareGenerator 和 KPDetector 类的实例（生成器（generator）和关键点检测器（kp_detector）），加载预训练的权重，并设置模型为评估模式。这样，这两个模型实例就可以用于生成动画帧，而无需从头开始训练。参数：config：配置好的权重参数；返回值：配置好模型参数的生成器（generator）和关键点检测器（kp_detector）
5. make_animation 函数：位于 ppgan 文件夹中的 first_order_predicter 文件中，功能：生成动画，并以数组形式返回。参数：source_image：源图像；driving_video：驱动视频，其动作将被转移到源图像上；relative：布尔值，指示是否使用相对坐标进行动作迁移；generator：用于生成动画帧的模型；adapt_movement_scale：布尔值，指示是否适应运动比例。返回值：所有生成的动画帧合并成一个 NumPy 数组；算法原理：函数 extract_bbox 读取视频后和源图后，则将驱动视频分割成多个批次以进行批量处理，使用 kp_detector 检测源图像和驱动视频中的关键点，遍历驱动视频的每一帧，将源图像的关键点与当前帧的关键点结合起来，通过 generator 生成动画帧。最后返回由动画桢组成的 NumPy 数组。

3.4.8 全局数据结构与该模块的关系

传递了全局变量 value，根据不同的 value 值选择不同的驱动视频路径，生成不同的动画表情。

3.5、模块 3 静态处理模块

3.5.1、设计图



3.5.2、功能描述

结合前端上传操作，后端根据用户的上传的照片，对用户上传的图片进行静态人像的修复效果处理最后将得到的结果，即成功修复之后的图像返回给前端网页，供用户进行保存。

3.5.3、输入数据

1. 图片：用户上传与需要处理的的图片
2. 文件路径: gfpgan 接口中输入的图片路径

对于上传的图片文件，代码会进行以下有效性检验：
文件部分： 'file' not in request.files 确保请求中包含文件部分。
文件名： file.filename == '' 检查文件名是否为空。
对于 gfpgan 接口中的图片路径，代码会进行以下有效性检验：
路径是否存在： not os.path.exists(input_path_full) 检查提供的路径是否实际存在。
图片格式的有效性检验：直接将上传的图片文件与 JPG 和 PNG 格式进行比较。如果上传的文件格式不符合要求，将提示用户选择正确的图片格式。

从物理模型中的数据库表获取数据的相关信息如下：
在 upload_pic 路由中，图片文件被保存到 UPLOAD_FOLDER 目录下的一个临时文件中。
在 show_img 路由中，从 current_selectimage 路径下获取显示的图片数据。
在 get_upload 路由中，从 current_uploadimage 路径下获取上传的图片数据。
在 gfpgan 路由中，根据输入的图片路径从文件系统中读取图片，并使用 image_handle 处理图片，处理后的结果存储在 static/images/results 目录下，并通过 result_image 返回。

3.5.4、输出数据

1. 图片数据:输出根据上传图片生成的修复图片。
2. 响应数据：在处理图片或其他操作完成后，会返回相应的 JSON 数据，包括处理结果、输入图片的链接和输出图片的链接等。

3.5.5、数据设计

1.本程序中涉及到的局部数据结构如下：

app：Flask 应用程序实例，用于处理 HTTP 请求和响应。

request：Flask 的请求对象，包含客户端发送的请求信息，如请求方法、URL、表单数据等。

jsonify：用于将 Python 对象转换为 JSON 格式的响应。

make_response：用于创建 Flask 的响应对象。

db：数据库连接对象，用于与数据库进行交互。

cors：CORS 扩展，用于处理跨域请求。

UPLOAD_FOLDER：上传文件的存储目录。

current_selectimage：已选择图像的存储目录。

current_uploadimage：上传图像的存储目录。

2.相关数据库表和数据存储设计如下：

本程序中未使用数据库，所有数据都存储在文件系统中。

UPLOAD_FOLDER 目录用于存储上传的图像文件。

current_selectimage 目录用于存储处理后的图像文件。

current_uploadimage 目录用于存储原始图像文件。

3.前端局部模块:

数据结构名称	功能说明	具体数据结构说明（定义、注释设计、取值）
query	用于存储查询条件的对象	{curPage: 1, pageSize: 10, tableName: "file", keyword: ""}
pageTotal	存储总页数的变量	ref(0)
fullscreenLoading	表示全屏加载状态的变量	ref(false)
url	存储图片的 URL 的变量	ref('../src\assets\img\null.png')
srcList	存储图片的预览源列表的变量	ref(['http://localhost:5000/img/10045_1710698433.png', ''])

3.5.6、算法和流程

对于前端静态修复局部模块，该算法的输入数据是用户选择的图片文件，输出数据是处理后的图片的路径和预览图片。用户上传图片文件后，触发 dealimage 函数。在 dealimage 函数中，使用 axios 发送 GET 请求到后端接口，获取处理后的图片路径。将处理后的图片路径赋值给 url 变量，并将原始图片路径和处理后图片路径添加到 srcList 数组中。
在页面中展示处理后的图片预览。

对于后端修复局部模块，该算法的主要目的是实现图像的上传、展示以及基于 GFPGAN 的图像修复功能。用户首先上传图像，用户通过浏览器选择要上传的图像，并提交到服务器。服务器将图像保存到指定的文件夹中，并返回成功上传的消息和图像的路径。然后进行图像的处理和修复，服务器接收输入路径，并调用 image_handle 函数进行图像修复。image_handle 函数根据输入路径读取图像，并将处理后的结果保存到指定的文件夹

中。服务器返回修复后的图像路径和原始图像的路径。接下来程序会展示图像，用户请求特定名称的图像。服务器从指定文件夹中读取图像数据，并以 image/jpg 格式返回给用户。

3.5.7、函数说明

1. os.makedirs 函数:位于 inference_gfpgan.py 文件中,其功能是创建目录，格式是 os.makedirs(path, exist_ok=False)参数为 args.output（输出目录路径），exist_ok=True（如果目录已存在，则不抛出异常），没有涉及到全局变量和局部变量，如果指定的目录不存在，则创建它。使用约束是需要有效的目录路径.
2. glob.glob 函数：位于 inference_gfpgan.py 文件中，其功能是从目录中搜索符合特定规则的文件路径，格式是 glob.glob(pattern)，参数是此脚本中用于获取输入目录下的所有文件路径，没有涉及到全局变量，局部变量 img_list，返回值是返回文件路径列表，根据提供的模式（通常是路径）搜索文件，并返回匹配的文件列表。使用约束是，需要有效的搜索模式。
3. cv2.imread 函数：位于 inference_gfpgan.py 文件中，其功能是读取图像文件，格式是 cv2.imread(filename, flags),参数 img_path(图像文件路径)，cv2.IMREAD_COLOR（读取彩色图像），没有涉及到全局变量，其中涉及到的局部变量是 input_img，返回值是返回读取的图像数组，根据提供的文件路径和读取模式读取图像。使用约束是，需要有效的图像文件路径。
4. GFPGANer.enhance 方法:位于 inference_gfpgan.py 文件中，其功能是增强图像中的人脸,格式是 restorer.enhance(input_img, ...),参数是 input_img（输入图像），以及其他参数如是否对齐、是否只恢复中心人脸等，没有涉及全局变量，局部变量有 cropped_faces, restored_faces, restored_img，返回值是，裁剪的人脸、恢复后的人脸和（可选）恢复后的完整图像，算法说明就是使用 GFPGAN 模型对输入图像中的人脸进行增强。使用约束是需要初始化的 GFPGANer 对象和有效的输入图像。
5. imwrite 函数：位于 inference_gfpgan.py 文件中，其功能是将图像数组写入文件，格式是 imwrite(img, path)，参数是 img（要保存的图像数组），path（保存路径），没有涉及到全局变量和局部变量，算法说明是将图像数组保存为文件。使用约束:就是需要有效的图像数组和文件路径。
6. dealimage 函数：位于 uploadFile_static_state.vue，功能是处理图片上传成功后的逻辑，格式是 dealimage(response: any, file: any, filelist: any) => void，具体的参数有，response: 从后端接收的响应对象。file: 上传的文件对象。filelist: 当前已上传的文件列表。没有涉及到局部变量，涉及到全局变量为 fullscreenLoading, url, srcList，没有涉及到局部变量，无返回值，算法说明是打印响应中的图片路径。显示全屏加载动画。发送 GET 请求到/gfpgan，传递图片路径作为参数。根据响应结果更新图片 URL 和源列表。使用约束是需要后端服务支持/gfpgan 路由。
7. openFullScreen1 函数:位于 uploadFile_static_state.vue，功能是控制全屏加载动画的显示状态。格式是 openFullScreen1() => void，无参数，全局变量是 fullscreenLoading（一个响应式引用，用于控制加载动画的显示状态）。，没有涉及到局部变量，没有返回值，算法说明，设置 fullscreenLoading.value 为 false，关闭加载动画。无特定约束.
8. uploadFileError 函数:位于 uploadFile_static_state.vue，功能是处理图片上传失败的逻辑。格式是 uploadFileError(err: any, file: any, fileList: any) => void，参数 err: 错误对象。file: 上传失败的文件对象。filelist: 当前已上传的文件列表。没有涉及到全局变量和全局变量。没有涉及到返回值，算法说明，打印“上传失败”到控制台。，无使用约束性。

9. onMounte 函数；位于 uploadFile_static_state.vue，功能是 Vue 生命周期钩子，组件挂载后执行。格式:是 onMounted(() => {...}) => void 无参数，无局部变量和全局变量，没有返回值，算法说明是，组件挂载后执行的代码块，当前为空，可能用于调用获取数据的方法。使用约束:是需要在 Vue 组件中使用。
10. upload_pic 函数：位于 app.py，功能是处理文件上传，格式为 @app.route('/upload', methods=['POST']) ，没有参数，全局变量是 UPLOAD_FOLDER ，指定上传文件的存储路径，局部变量是 file ，接收上传的文件，返回值是如果上传成功，返回 jsonify({'message': '文件上传成功', 'path': file_path}) ，其中 path 为上传文件的路径；否则返回错误信息。算法说明是获取上传的文件，生成唯一的文件名，保存文件到指定路径，并返回上传成功的消息和文件路径。使用约束：需要在请求中包含 file 字段来上传文件。
11. show_img 函数：位于 app.py，功能是处理文件上传，功能是展示指定图片，格式为@app.route('/img/<string:name>', methods=['GET']) ，参数是 name ，图片的名称，全局变量是 current_selectimage ，指定选择图片的存储路径，局部变量是 name ，图片的名称，返回值是如果图片存在，返回包含图片数据的响应；否则返回错误信息。算法说明是根据提供的图片名称，读取图片文件的内容，并设置响应的内容类型为 image/jpg ，然后返回图片数据。使用约束是需要提供图片的名称作为参数。
12. get_upload 函数：位于 app.py，功能是处理文件上传，功能是获取指定上传文件的内容，格式是 @app.route('/getupload/<string:name>', methods=['GET']) ，参数是 name ，上传文件的名称，全局变量是 current_uploadimage ，指定上传文件的存储路径，局部变量是 name ，上传文件的名称，返回值是如果文件存在，返回包含文件内容的响应；否则返回错误信息。算法说明是根据提供的上传文件名称，读取文件的内容，并设置响应的内容类型为 image/jpg ，然后返回文件内容。使用约束是需要提供上传文件的名称作为参数。
13. gfpgan 函数:位于 app.py，功能是处理文件上传，功能是进行 GFPGAN 处理,格式是 @app.route('/gfpgan', methods=['GET']) 参数是 input ，输入图片的路径全局变量是 current_selectimage ，指定选择图片的存储路径局部变量是 i nput_path_full ，转换后的图片路径返回值是 如果处理成功，返回包含处理结果的 JSON 数据，包括输入图片的链接、处理后的图片链接和处理成功的消息；否则返回错误信息。算法说明是 首先根据输入的路径获取输入图片，然后使用 image_handle 函数进行处理，并获取处理后的图片路径。最后，返回处理结果的 JSON 数据。使用约束是 需要在请求中提供输入图片的路径。
14. image_handle 函数：是位于 app.py，功能是应用 GFPGAN 进行图像处理，格式是 image_handle({...}) ，参数是一个包含输入和输出路径的字典，全局变量无，局部变量是 handle_res ，处理后的图片路径，返回值是处理后的图片路径，算法说明是使用 subprocess 模块调用 GFPGAN 进行处理，并返回处理后的图片路径。使用约束是需要在项目根目录下有 GFPGAN 文件夹，并且 GFPGAN 可执行文件在系统路径中。
15. make_respons 函数：位于 app.py，功能是创建响应对象，格式是 make_response(response_data) ，参数是 response_data ，要返回的数据，全局变量无，局部变量是 response ，响应对象，返回值为响应对象，算法说明是创建一个响应对象，并设置响应的数据和内容类型。，使用约束是通常用于将数据转换为响应并返回给客户端。
16. secure_filename 函数：位于 app.py，功能是确保文件名的安全性，格式是

secure_filename(filename) ，参数是文件名，全局变量无，局部变量是 filename ，要处理的文件名，返回值是处理后的文件名，算法说明是对文件名进行一些安全处理，去除可能不安全的字符。，使用约束是用于确保文件名在文件系统中是合法的。

17. os.path.join 函数：位于 app.py，功能是拼接文件路径，格式是 os.path.join(directory, filename) 参数是 directory ，目录路径；filename，文件名，全局变量无，局部变量是 directory ，目录路径 filename ,文件名，返回值是拼接后的文件路径，算法说明是将目录路径和文件名组合成一个完整的文件路径。使用约束是用于构建文件或目录的完整路径。
18. uuid.uuid4 函数：位于 app.py，功能是生成 UUID（Universally Unique Identifier），格式是 uuid.uuid4() ，参数无，全局变量无，局部变量是 uuid ，生成的 UUID，返回值是生成的 UUID，算法说明是使用 UUID 生成器生成一个唯一的 UUID。使用约束是用于生成唯一的标识符。
19. open 函数：位于 app.py，功能是打开文件，格式：是 open(file_path, mode) ，参数是 file_path ，文件路径； mode ，文件打开模式，全局变量无，局部变量是 file_path 文件路径，mode ，文件打开模式，返回值是文件对象，算法说明是根据指定的路径和模式打开文件，并返回文件对象。使用约束是用于读取或写入文件。

3.5.8 全局数据结构与该模块的关系

query :用于存储查询条件，包括 curPage 、pageSize 、tableName 和 keyword 。

pageTotal :用于存储总页数。

fullscreenLoading :用于控制全屏加载的显示和隐藏。

url :用于存储图片的 URL。

srcList :用于存储图片的预览地址列表。

app:Flask 应用程序实例。

db 数据库连接。

cors:cors 扩展。

UPLOAD_FOLDER :上传文件的保存路径。

output_path :处理后图片的保存路径。

app.config :应用程序的配置字典，包含上传文件的保存路径等信息。

4、接口设计

4.1、 用户接口

1.Element Plus Upload: 这是一个图片上传组件，允许用户通过拖拽或点击上传图片。它提供了一些属性，如 action（上传地址），limit（限制上传数量），show-file-list（是否显示文件列表），on-success（上传成功的回调），on-error（上传失败的回调）等。
2.fullscreenLoading: 这是一个 ref，用于控制全屏加载动画的显示和隐藏。
3.uploadFileError: 这是一个函数，当图片上传失败时被调用。它接收三个参数：err（错误对象），file（上传的文件对象），fileList（文件列表）。此函数目前只是打印了“上传失败”。
4.Delete, Edit, Search, Plus: 这些是从 @element-plus/icons-vue 导入的图标组件，可以在界面上显示相应的图标。向用户展示。

4.2、 外部接口

本软件是一个相对独立的集前后端项目共存的一个系统，没有设置外部接口供外部访问与调用。

再来介绍一下硬件和模型相关的，GPU 加速接口:脚本检查是否有可用的 CUDA 设备(torch.cuda.is_available())来使用 GPU 加速处理。如果使用 RealESRGAN 作为背景上采样器，并且 CUDA 不可用，脚本会发出警告并可能不使用 RealESRGAN。GFPGAN 模型接口是通过 GFPGANer 类实例化 GFPGAN 模型，用于面部增强和恢复。根据用户选择的版本下载或加载预训练模型。RealESRGAN 模型接口 (可选)是如果用户选择使用 RealESRGAN 作为背景上采样器，脚本会加载或下载相应的模型。RealESRGAN 用于提高背景图像的质量。硬件接口是脚本可以运行在支持 CUDA 的 GPU 上以加速处理，也可以在 CPU 上运行。软件依赖接口是脚本依赖于多个 Python 库，如 OpenCV (cv2)、NumPy (numpy)、PyTorch (torch)、BasicSR (basicsr)和 GFPGAN (gfpgan)。

4.3、 内部接口

4.3.1、 接口说明

一.

1. 静态处理模块的后端程序通过 **axios.get('/gfpgan', {params: { input: response.path }})** 从静态处理模块的前端中获得用户上传的图片。
2. 这是一个通过 axios 发起的 GET 请求，用于将上传的图片发送到后端服务器进行处理。请求的 URL 是 /gfpgan，参数 input 包含了上传图片的路径。

二.

1. 动态处理模块的前端程序通过 **axios.post('http://127.0.0.1:5000/value', {...})**从动态处理模块的前端中获得单选按钮选中的值(Value)。
2. 这是一个用于将单选按钮选中的值发送到后端服务器的 POST 请求。它通过 axios 发送数据，请求的 URL 是 "http://127.0.0.1:5000/value"。

三.

1. 静态处理模块的后端程序通过**/upload (POST 方法)**从静态处理模块的前端中处理图片上传的逻辑实现.
2. 这个接口用于处理图片上传。它接收一个文件作为 POST 请求的一部分，并将其保存在服务器的指定目录中。如果上传成功，它将返回一个包含成功消息和文件路径的 JSON 对象。给出接口的定义

```
@app.route('/upload', methods=['POST'])
def upload_pic():
    print("进行上传")
    if 'file' not in request.files:
        return jsonify({'error': '没有文件部分'}), 400
    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': '没有选择文件'}), 400
    if file:
        filename = secure_filename(file.filename).split('.')[0] + "_" + str(int(time.time())) + '.'
+ secure_filename(file.filename).split('.')[1]
        print("filename:", filename)
file_path=os.path.join(r'D:\python\project-finally\GFPGAN\GFPGAN\static\images\uploads',
filename)
        file.save(os.path.join(project_root_path, file_path))
```

```
        return jsonify({'message': '文件上传成功', 'path': file_path}), 200
```

四.

1. 静态处理模块的后端通过**/img/string:name (GET 方法)**从静态处理模块前端获取指定名称的图片，
2. 这个接口用于获取指定名称的图片。它从请求的 URL 中获取图片的名称，并尝试从服务器的 current_selectimage 目录中读取对应的图片文件。如果文件存在，它将图片作为响应返回。给出接口的定义

```
@app.route('/img/<string:name>', methods=['GET'])
def show_img(name):
    img_url = os.path.join(app.config['current_selectimage'], name)
    if name:
        image_data = open(img_url, "rb").read()
        response = make_response(image_data)
        response.headers['Content-Type'] = 'image/jpg'
        return response
```

五.

1. 静态处理模块的后端通过**/getupload/string:name (GET 方法)**从后端上传图片文件夹中获取上传的图片。
2. 这个接口用于获取上传的图片。它同样从请求的 URL 中获取图片的名称，并尝试从服务器的 current_uploadimage 目录中读取对应的图片文件。如果文件存在，它将图片作为响应返回。给出接口的定义

```
@app.route('/getupload/<string:name>', methods=['GET'])
def get_upload(name):
    img_url = os.path.join(app.config['current_uploadimage'], name)
    if name:
        image_data = open(img_url, "rb").read()
        response = make_response(image_data)
        response.headers['Content-Type'] = 'image/jpg'
        return response
```

六.

1. 静态处理模块的后端通过**/gfpgan (GET 方法)**从本模块中获取 input 参数。
2. 这个接口用于执行图像处理。它接收一个 input 参数，该参数指定了要处理的图片的路径。然后，它调用 image_handle 函数来处理图片，并返回处理结果的 URL。如果处理成功，它将返回一个包含成功消息、输入图片的 URL 和输出图片的 URL 的 JSON 对象。给出接口的定义

```
@app.route('/gfpgan', methods=['GET'])
def gfpgan():
    input_path = request.args.get('input')
    input_path_full = os.path.join(project_root_path, input_path)
    if not input_path_full:
        return jsonify({'error': 'Input path is required'}), 400
    getupload = f"http://localhost:5000/getupload/{os.path.basename(input_path)}"
    handle_res = image_handle({
        'input': input_path_full,
```

```
            'output': 'static/images/results',
        })
        result_image = f"http://127.0.0.1:5000/{handle_res[0]}"
        return jsonify({'message': 'Face restoration completed', 'input': getupload, 'output': result_image}), 200
```

七.

1. 动态处理模块的后端通过**/value (POST 方法)**从本模块中获取前端用户选择的 value 数值，进而用来确定驱动视频的选择。
2. 这个接口用于接收一个 JSON 对象，该对象包含一个 selectedValue 属性。这个属性的值从请求的 JSON 数据中获取，并存储在全局变量 value 中。接口返回一个 JSON 对象，包含一条消息，表明已接收到选定的值。给出接口的定义

```
@app.route('/value', methods=['POST'])
def value():
    data = request.json
    global value
    value = data.get('selectedValue')
    print("value:", value)
    return jsonify({'message': 'Received selected value: {}'.format(value)})
```

八.

1. 动态处理模块的后端通过**/process_image (GET 和 POST 方法)**获取动态处理模块前端用户上传的图片并进行处理图片和视频的编辑任务。
2. 这个接口用于处理图片和视频的编辑任务。它首先根据全局变量 value 的值设置环境变量 DRIVING_VIDEO，该变量指向一个特定的视频文件，这个视频文件将用于后续的图像处理。给出接口的定义

```
@app.route('/process_image', methods=["GET", 'POST'])
def process_image():
    # ... 省略了部分代码 ...
```

九.

(1)登录注册模块通过**注册接口 (/register) 登录接口 (/login) 游客登录接口 (/login)**三大接口来获取前端网页中用户的登录输入信息.游客登录响应以及注册信息.

(2)

1. 注册接口 (/register):

请求类型: POST

请求发送者: 用户注册时前端页面。

请求体: 包含新用户的 username 和 password。

请求处理: 后端服务器接收请求后，应将新用户的数据存储到数据库中，并返回相应的状态码和消息。

2.登录接口 (/login):

请求类型: POST

请求发送者: 用户登录时前端页面。

请求体: 包含用户的 username 和 password。

请求处理: 后端服务器验证提供的凭据，如果与数据库中的记录匹配，则允许用户登录，并返回相应的状态码和消息。

3.游客登录接口 (/login):

请求类型: POST
请求发送者: 用户选择以游客身份登录时前端页面。
请求体: 包含预设的 username 和 password（在这个例子中是 '1'）。
请求处理: 后端服务器可能有一个特殊的逻辑来处理游客登录，允许用户以游客身份访问某些功能或页面。
4.这些接口的实现细节（如验证逻辑、密码加密存储、会话管理等）都在后端服务器上完成。前端代码只是发起请求并处理响应。响应中包含状态码，如 200 表示成功，202 表示用户名已存在或密码错误，400 表示请求错误，500 表示服务器错误等。
以下是 axios 请求的示例代码：

```
// 注册请求示例
await axios.post('/register', {
    username: this.Registerform.username,
    password: this.Registerform.password
});
// 登录请求示例
await axios.post('/login', {
    username: this.Loginform.username,
    password: this.Loginform.password
});
```

在这些示例中，axios.post 方法用于发送 POST 请求到指定的 URL（/register 或 /login），请求体中包含相应的用户数据。请求成功后，会根据后端返回的状态码和消息进行相应的处理。

4.3.2、 调用方式

一.静态处理模块:
使用 axios.get 和 axios.post 方法与后端进行通信。
/upload 接口用于上传图片，返回文件保存路径。
/img/<string:name> 和 /getupload/<string:name> 接口用于获取图片，根据图片名称返回图片数据。
/gfpgan 接口用于图像处理，接收图片路径作为参数，并返回处理结果。

二.动态处理模块:

使用 axios.post 方法发送用户选择的值到后端。
/process_image 接口用于处理图片和视频编辑任务，根据全局变量 value 设置环境。

三.登录注册模块:
提供 /register, /login, 和 /tourist_login 三个接口。
这些接口处理用户的注册、登录和以游客身份登录的请求。

5、数据库设计

数据层 db，这里我们使用 mysql 数据库，通过 login. js 文件中函数的编写，实现用户的注册输入，登陆判定，头像上传，在线用户的登录信息更改，用户操作记录等众多数据信息的存储与增删改查操作。在数据库后端文件中写出 login，register 函数并在 3000 端口上运行后端数据库（app. listen(3000)），在前端 api 文件中定义函数连接端口，以便全局通过 axios. post 访问后端函数接口。

我们存储的数据主要为 varchar 一种类型，varchar 所设定的最大长度大多为 50，用于满足用户昵称密码设置需求和记录用户的上传记录。

数据库命名为 gan，其中表格命名为 user 表。

user 表下有 username，password，record 三个字段，分别对应用户名，密码与用户上传记录。字段类型都是 varchar。username 考虑到不可重名设定为 UNIQUE，所有字段都默认为 NULL

username	password	record
1	1	10.png,20.jpg

在前端，我们利用 mysql 数据库的 mysql.createPool 进行与数据可的连接，连接成功后，利用 query（）函数进行 sql 语句的更新或者查询操作，这是基本的更新与查询数据库操作，具体实现则是在 server 文件中。

6、系统出错处理

6.1、 出错信息

用一览表的方式说明每种可能的错误和故障，以及系统输出信息的形式、含义和处理方式。

表 6-1 可能的报错信息一览表

报错	报错信息	问题解释	解决方案
密码用户名错误	error(response.data.message || '登录失败，请检查输入信息')	用户名密码不匹配	重新输入正确的用户名密码
重复登录	response["errmsg"] = "this account is using, input another!";	所登录账户已经登录	换账户登录
登录时登录不上去	error('未知错误，请稍后再试');	数据库连接失败	重新启动数据库后端，检查数据库接口

没有信息提示，但登录注失败	(error.response) { console.error('服务器返回错误', error.response.data); this.$message.error('注册失败，请稍后再试');	数据库后端端口函数访问失败，发送请求没有被接收	重新重新启动数据库后端，检查数据库接口，检查 axios.post 请求路径
登陆失败，没有提示信息	请求设置错误 router is not defined loginmethod	请求端口错误	更改端口设置
Mysql 报错	LOG_INFO << "mysql connection fail";	数据库连接失败	检查服务器上的数据库是否正常运行
后端服务器连接失败	UploadAjaxError: fail to post http://127.0.0.1:5000/process_image 0	后端服务器未启动，前后端端口未连接	启动后端服务器，检查前后端端口
未找到后端接口	POST http://localhost:5000/upload 404 (Not Found)	接口方式错误，前后端口不一致，后端未启动，后端端口被占用	检查前后端，数据库端口，保证前后端口一至，数据库端口不同

6.2、 补救措施

说明故障出现后可能采取的补救措施，如恢复、再启动技术等。

重新启动软件；

重新启动服务器；

重新加载 nginx 服务；

重新启动后端。

重新启动数据库。

修改数据库，前端，后端的运行端口。

7、其他设计

如系统安全设计、性能设计等。

1.系统安全设计

主要针对于前端登录注册模块。后端图片指定存储于加密。

数据传输安全

使用 HTTPS 协议：确保所有数据传输都通过 HTTPS 进行，以避免中间人攻击。

数据加密：敏感数据在传输过程中应进行加密。

用户认证与授权 强密码策略：前端可以强制用户创建复杂性较高的密码。 二次验证：注册和登录过程中可以增加二次验证步骤，如短信验证码或邮箱验证。 权限检查：后端应实现权限检查，确保用户只能访问授权的资源。
输入验证 避免 SQL 注入：后端应验证和清理所有输入，避免 SQL 注入攻击。 避免 XSS 攻击：前端应使用库（如 Vue.js 的内置功能）来自动转义 HTML 内容，防止 XSS 攻击
错误处理与会话管理 避免泄露信息：错误消息不应提供敏感的系统信息，应使用通用错误消息。 日志记录：后端应记录详细的错误日志，以便于问题追踪和分析。 Session 超时：会话在一段时间无活动后应自动过期。 安全 Cookie：使用 HttpOnly 和 Secure 标志设置 Cookie，防止客户端脚本访问。
密码存储和跨站脚本（XSS）防护 加密存储：密码应在数据库中以加密形式存储，不应以明文形式存储。 输入清理：对所有输入数据进行清理，避免 XSS 攻击。 内容安全策略（CSP）：使用 CSP 可以减少 XSS 攻击的风险。

2.性能设计

性能设计主要针对于主功能，就是静态处理模块和动态处理模块。

多线程和并行处理 单进程处理：代码中没有使用多线程或多进程来处理图像。对于大量图像处理任务，这可能会导致处理速度较慢。 GPU 加速：代码中提到了检查 torch.cuda.is_available()，这意味着如果 GPU 可用，代码将利用 GPU 加速处理过程。
.内存管理 图像读取：图像是逐个读取和处理的，这有助于控制内存使用量。 分步处理：代码将处理过程分解为读取、增强和保存，这有助于逐步释放不再使用的内存。
I/O 操作 批量处理：代码支持批量处理图像，通过 glob.glob 获取所有输入图像的路径，这有助于减少 I/O 操作的次数。 文件系统：输出目录使用 os.makedirs 创建，确保了即使目录不存在也能正常运行。
可配置性 命令行参数：使用 argparse 库来处理命令行参数，这增加了程序的灵活性和可配置性。 预设模型：用户可以通过命令行参数选择不同的 GFPGAN 模型版本。
错误处理 文件检查：代码检查预训练模型文件是否存在，如果不存在则尝试从 URL 下载。 警告：在 CPU 模式下使用 RealESRGAN 时给出了警告，提示用户可能的性能问题。
代码结构 模块化：代码结构清晰，函数和类的定义有助于代码的可读性和维护性。 函数封装：主要逻辑封装在 main 函数中，而图像处理的具体步骤则由 GFPGANer 类的实例管理。
性能优化和可扩展性

模型加载：模型是在运行时加载的，这可能会增加启动时间，但允许灵活地选择不同的模型。 参数调整：用户可以通过命令行参数调整模型的权重，这有助于优化特定图像的处理效果。 模型版本：代码支持不同的 GFPGAN 模型版本，这有助于适应未来模型的更新。
输出格式 应用程序将处理结果转换为 base64 编码的字符串，这有助于在不支持二进制数据的环境下传输数据。