Web 智寻 IntelliFind-

基于用户行为数据的网站体验评分算法系统

封面 TODO

目录

TODO（等所有弄好了最后制作目录）

摘要

TODO

1 作品概述

本章主要从前言、项目目标、项目命名、项目背景、创意描述、特色综述七个方面对 Web 智寻 IntelliFind 系统（后简称智寻系统）进行介绍。

1.1 前言

在科技快速发展的今天，网络成为了人们工作生活绝不可少的一环，而其中每个人的日常生活都离不开网站的使用。网站是人们获取并交互信息的主要途径，一个网站性能的好坏，很大程度上影响了人们的体验。

根据 2015 年 Aberdeen Group 的研究报告，对于 Web 网站，1 秒的页面加载延迟相当于少了 11% 的页面观浏览，相当于降低了 16% 的顾客满意度。如果从金钱的角度计算，就意味着：如果一个网站每天挣 10 万元，那么一年下来，由于页面加载速度比竞争对手慢 1 秒，可能导致总共损失 25 万元的销售额。同时该报告中指出分析了超过 150 个网站和 150 万个浏览页面，发现页面响应时间从 2 秒增长到 10 秒，会导致 38% 的页面浏览放弃率。

由此可见，网站性能与业务目标有着直接的关系，对网站进行性能测试非常重要。再结合用户的实际体验，启动一个软件如果很卡，就不太想用了，如果在中间使用时再很卡时，下次再想使用的欲望就会强烈减少，甚至会产生排斥心理。

但是对于大多数站点管理者而言，并没有太多时间和精力自己去检测网站的性能和用户体验问题，他们往往希望有一个专门的网站，能够检测用户在网站体验中存在的问题，并作出相应评分，甚至提出相关的优化建议，这样就能客观的反映出网站性能的好坏，站点的管理员也更能针对性地优化和修复网站性能上的问题。因此一个合适能够检测用户行为数据的网站体验评分算法是必不可少的。

因此我们分析项目可行性并进行实践开发，提出能够检测用户行为数据并反馈给用户的智寻系统。

1.2 项目目标

智寻系统目标达成以下几点：

1. 接收输入的用户行为数据，通过算法分析得出结果，并反馈给用户。

2. 以直观清晰的方式布局反馈的结果报告，让用户直观的感受网站的体验问题

和优化建议。

3. 模拟真实的网站，考虑工程上的更多的细节问题，让本系统在工程角度上更加完善和强大。

1.3 项目背景

党的十八大以来，习近平总书记高度重视网络安全和信息化工作，从信息化发展大势和国际国内大局出发，就网信工作提出了一系列新思想新观点新论断，深刻回答了一系列方向性、根本性、全局性、战略性重大问题，形成了内涵丰富、科学系统的习近平总书记关于网络强国的重要思想，为做好新时代网络安全和信息化工作指明了前进方向、提供了根本遵循。

习近平总书记关于网络强国的重要思想，坚持马克思主义立场观点方法，立足人类进入信息社会这一崭新时代背景，站在我们党“过不了互联网这一关，就过不了长期执政这一关”的政治高度，准确把握信息化变革带来的机遇和挑战，从国际和国内、历史和现实、理论和实践的结合上，深入回答了为什么要建设网络强国、怎样建设网络强国的一系列重大理论和实践问题，深化了我们党对信息时代共产党执政规律、社会主义建设规律、人类社会发展规律的认识，丰富了习近平新时代中国特色社会主义思想的科学内涵。

在这样的大背景下，网络中占据绝对地位的网站就更加需要改进自身。不仅需要改进内容，更加需要考虑到网站性能和体验上的问题，考虑到对网站浏览者，考虑到对用户的体验的维度。这样才能真正把一个网站的管理做好，才能真正坚持习近平总书记提出的网络强国的重要理论。

对于智寻系统，就是在这样的背景下，孕育而生，能够有效的帮助网站管理者优化改进自己的网站，这是顺应党的号召的良好表现。

1.4 项目命名

Web 智寻 IntelliFind 系统是一个基于用户行为数据的网站评分算法系统，为用户提供了方便、直观的服务，并且响应了党提出的网络强国的重要战略。

Web，代表本系统采用 BS 架构，是一个 Web 网站系统，用户使用浏览器就可以方便地访问和使用。

“智寻”二字，代表本系统可以检测用户提供的日志，通过分析得出直观清晰的结果。用户通过将自己网站记录的日志文件传递给本系统，进而通过算法调用，最后计算得出这些用户行为的得分、优化建议等，然后以各种数据图形图、数据等直观的展现在用户面前，让用户能够针对自己网站的体验做出优化

和修改。

IntelliFind，融合了单词 Intelligence 智能和 Find 寻找的含义，也是智寻二字的体现。

Web 智寻 IntelliFind 系统，致力于为用户提供可靠、方便、直观、清晰的检测、分析的服务。

1.5 创意描述

经过调研市面上其他网站分析系统，我们发现如下问题：

1. 对于非开发人员不友好。市面上有的系统在反馈解析结果时引用了大量的专业术语，不便于非专业人员的使用。

2. 网站功能老旧、界面粗糙或是操作复杂。

3. 不支持用户自定义，只能够按照自己的分析套路进行，容易误导用户。

4. 功能不完善，可视化界面较少，分析结果不准确。

1.6 特色描述

1. 支持自定义模式。用户可以根据自身需要。对不同问题设定不同权重，反映出自己更加注重哪一方面的问题，能够更好地帮助用户抓住网站问题所在，对症下药，寻求高效地答案。

2. 系统对于用户上传的 json 文件进行了预处理，经过了数据解析与拆分工作后再交由算法模型生成结果。保障了算法模型的运行效率，尽可能地减少用户等待的时间，提高用户体验。

3. 系统采用了新兴的算法大模型对用户上传文件进行解析，并且将解析结果尽可能地可视化呈现用户，用户能够更加直观地看出网站问题所在，更好地感受数字反映出的问题

4. 文件系统。系统设计了完善丰富地文件系统，能够很好地支持用户上传文件的请求，对于上传的文件，不仅数据库有上传记录，文件本身也统一保存在文件系统中。

5. 两层缓存保证数据查询效率。系统设计了基于 LRU 和 Redis 的两层缓存,极大地减少了数据查询的时间，提高了用户使用体验。

6. 分布式部署。web 服务和算法模型分布式部署在两个服务器上，减少了不同板块的相互影响。

1.7 本章小结

本章从前言、项目目标、项目命名、项目背景、创意描述、特色综述七个方面对智寻系统进行了介绍，主要对整体项目进行了概述，方便了解项目开发的目的以及解决的内容，以及项目的亮点情况。

2 需求分析与建模

本章主要从应用对象分析、功能性需求、非功能性需求、应用环境分析四个方面对智寻系统的需求分析进行介绍。

2.1 应用对象分析

智寻系统的使用者主要有：

用户：以网站的管理人员为主，在日常管理网站的过程中经常会排查网站的各种问题，并且需要及时根据结果优化自身的网站，以便及时修改优化网站，为自己的客户带来更好的体验。同时这些网站管理者也是本网站的用户，他们可以在本网站当中查看以往的相关的检测信息、结果等。具体如图 2-1：

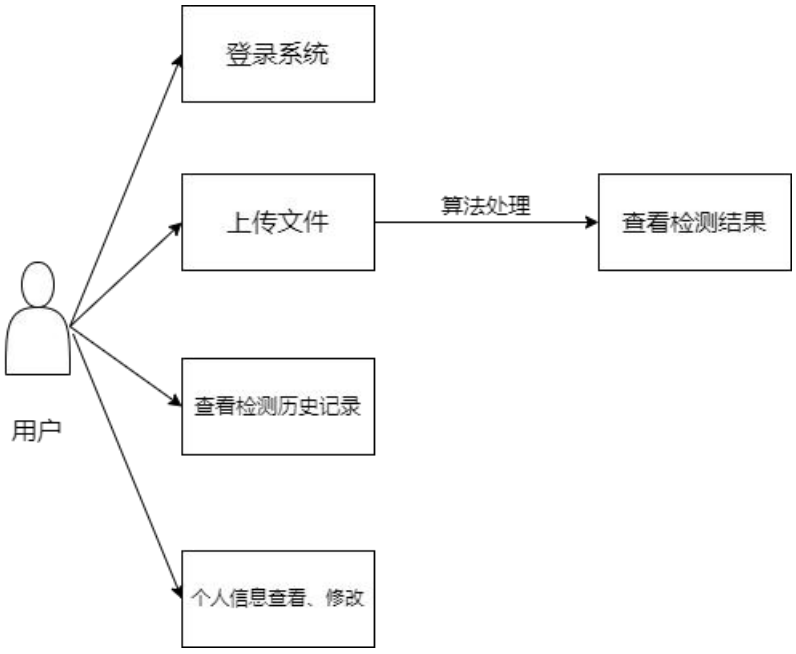

图 2-1

2.2 功能性需求

功能简介

作为一款力求模拟真实的网站评分系统，需要解析用户传入的数据，这些数据绝大部分就是用户自身网站的日志信息，里面就包含了网站体验过程中的各种问题，然后通过调用算法，最终得出清晰直观的结果，这就是一条完整的功能线。具体如下：

1. 登录功能：为了实现记录不同用户的检测历史和用户信息等功能，故设定登录功能，包括注册、邮箱验证、忘记密码找回、图形验证码等真实网站具有的流程。同时为了让用户使用十分简易方便，登录的所有功能均直接在同一页面，用户只需要直观的自行选择即可。

2. 用户个人信息查看、修改：对于有登录的网站而言，用户自然可以设置用户的用户名、邮箱、地址、个性签名、头像等，使得用户之间鲜明的区别开来。

3. 检索历史检测记录：对用户的不同次检测，在数据库中存储了当次检测的相关数据，例如相应问题的数据字段、检测得分等等。用户点击即可跳转到对应结果页面。

4. 上传文件：核心功能。实现了用户从上传文件到后台读取、分析、检测文件，并通过算法调用，得出最终检测结果并渲染成为检测结果页面的过程。

2.3 非功能性需求

1. 可用性需求：系统应易于使用，功能明确，操作简单，用户能够轻松理解如何使用网站，并进行文件的上传和结果的展示。

2. 性能需求：系统应快速处理传入文件中的内容，并通过算法分析，能够快速响应并返回结果，渲染在网站上。结果应当在 10 秒以内呈现给用户。

3. 可维护性需求：系统应易于维护和更新，有良好的文档和代码结构，支持快速诊断和解决问题，同时能支持性能优化。

4. 可扩展性需求：系统应易于扩展和定制，能够容易地添加新的功能或更新已有功能，未来能够支持新的文献类型和识别技术。

2.4 应用环境分析

1. PC 端环境：

a) 系统需支持浏览器的使用，推荐为 Chrome 浏览器。

b) 推荐使用 windows 系统，至少需要配备 Intel Core i5 以上的处理器、8GB 以上内存。

c) 系统需支持 PDF 文件的浏览，推荐使用 Windows 7 及以上操作系统、Microsoft Office 2013 及以上。

2. 网络连接：

a) 需要稳定的网络连接，推荐使用宽带或高速移动网络。

b) 可以通过浏览器或专用客户端访问系统，需要支持 HTTP 或 HTTPS 协议。

2.5 本章小结

本章从应用对象分析、功能性需求、非功能性需求、应用环境分析四个方面对智寻系统的需求分析进行了介绍。分别介绍了使用智寻系统的应用对象：客户；智寻系统的功能需求，并针对核心功能的流程进行了剖析；并且分析了智寻系统的非功能性需求，要求对可用性、可扩展性方面进行实现；最后对应用的环境进行了分析，提出项目运行的条件。

3 实施方案与可行性研究

本章从可行性分析和实施方案两个方面进行阐述，主要介绍项目实现前的分析方案和实现的具体细节。

3.1 可行性分析

3.1.1 市场可行性

在开发本系统以前，我们对市面上已有的产品进行了调研，目前市面上有很多网站性能检测网站和服务可用性检测网站，例如百川云、GTmetrix、Pingdom、Uptime Robot 等[1]，但是这些产品如果照搬到本系统中的需求当中，还是有很多难以达到要求的地方，故做如下分析：

1. 业务倾向不同：例如，百川云是一家专注于网络安全的厂商，GTmetrix 是一家提供托管服务的公司。业务倾向的不同导致了这些产品在实际运作的时候，对于本项目而言，这些网站的业务就不是很适合了。

2. 不支持个性化需求：对于本项目，各个网站的管理员是我们的用户，本系统希望为不同的用户提供个性化的服务，例如展示解析历史变化图，存储历史报告等等。上述的网站基本都不支持与本项目业务相关的个性化服务。

3. 数据隐私和安全性的要求：由于本系统会处理用户的网站日志数据，因此对数据隐私和安全性的要求可能会与市面上的产品有所不同。系统需要得到用户的信赖，确保用户的数据得到保护，而现有的一些产品可能并未专门考虑这些方面的需求，因此需要定制化开发。

4. 需要适应快速迭代和定制化开发：由于本系统需要根据不同用户的需求提供个性化的服务，因此需要具备快速迭代和定制化开发的能力。市面上的通用产品较难满足这种灵活性和定制化的开发需求，因此需要建立自己的开发团队或寻找专业的定制化开发服务提供商。

3.1.2 技术可行性

对于技术可行性，技术可行性需要考虑所选择的技术是否成熟，是否能够满足构成本系统的要求。我们将从前端、后端、算法三个方面阐述技术可行性。

1. 前端

a) 通过 Vite + Vue3 构建项目前端模块，通过数据的响应式交互让用户能够在操作时实时看到数据的更新。

b) 通过 Element-Plus 和 ECharts 组件丰富页面内容，美化数据的展示形式，提升用户体验。

c) 通过 Axios 发送异步请求，通过 Pinia 将 Token 暂存于本地，进而实现前后端分离式项目请求响应交互时的鉴权和认证。

2. 后端

a) 使用基于 golang 的 gin 框架。golang 是 web 开发的常用编程语言之一，其执行速度效率接近 c++，其并发模型基于协程（goroutines）和通道（channels），这使得编写高并发服务器变得简单而高效。

b) 使用 MySQL 数据库存储。MySQL 具备作为数据存储系统的多项优势，包括可靠性、性能、灵活性、成本效益、社区支持、扩展性和兼容性。

c) 基于 Redis 和 LRU 的两级缓存。使用 LRU 算法作为项目的第一级缓存，使用 Redis 作为第二级缓存。这两种缓存在技术上已经较为成熟，使用较为广泛，认可度较高。

3. 算法

a) 对于用户上传的文件，算法需要对其进行解析，最后返回响应的结果，我们将这个过程分为两个部分，人为解析和模型处理。

b) 人为解析：通过 python 脚本解析用户提供的 json 文件，从中提取有用的数据字段，获得有效数据，交予模型处理。

c) 模型处理：我们选择使用开源大模型 Llama2，并且经过官方的预训练和我们的训练，同时使用开源库 Transformers 运行模型，以做到保证效率和性能。

综上所述，本系统的技术可行性较高，市场上已经有比较成熟的技术栈可以支持实现该项目的各项功能，同时需要根据具体的业务需求和技术方案进行选择和整合。

3.1.3 资源投入可行性

对于资源投入的可行性分析，我们考虑以下几个方面：

1. 人力资源：开发智寻系统需要有相应的技术人员，包括前端、后端和算法部分人员，以及日常维护人员。在招募人员时考虑人员的专业技术和经验，以确

保能够顺利完成项目。

2. 硬件资源：开发和运行智寻系统需要相应的硬件资源，包括服务器、计算机等。需要根据系统的规模和预期的用户量来确定相应的硬件资源投入。

3. 软件资源：开发和运行这个系统需要使用相应的软件资源，包括操作系统、数据库、开发工具等。

4. 时间资源：开发一个完整的网站体验评分系统需要相当长的时间，包括需求分析、设计、开发、测试、部署等各个阶段。需要合理安排时间资源，确保项目能够按时完成。

综合考虑以上各个方面的资源投入，我们进行相应的成本预算和时间计划。人力资源而言，我们具有各类技术人员，前端两名、后端一名、算法一名、机动一名，分工明确，并且组长规定时间进度，队员按照要求进行开发和测试；硬件资源方面选用阿里云轻量级服务器进行初步开发，之后考虑用户数量扩大更换服务器；软件资源方面，选用大量开源技术，如 MySQL、Vue 、Flask 等；时间资源方面我们严格指定了项目的开发时间，能按时保质保量的完成。

故智寻系统的开发资源投入是可行的。

3.2 实施方案

3.2.1 项目系统结构和模块设计

1. 系统架构

在项目设计初期，本团队就对系统架构进行了设计，根据业务逻辑的不同层次，分别划分为用户层、业务层、技术层和存储层，如图 3-1：

a) 用户层：用户层考虑了使用的用户群体，对于本项目而言，用户绝大多数都是各自网站的站点管理员。

b) 业务层：智寻系统主要实现四大业务，分别对应登录功能、个人信息相关、查看检测历史和解析文件的相关业务，为业务层实现提供方法。

c) 技术层：智寻系统采用四大关键技术。使用 Vue3 和 Gin 技术实现 Web 页面前后端的搭建，使用 Python 语言解析用户提供的 json 日志文件，并使用 Llama2 模型及相关技术进行模型处理返回响应的结果。

d) 存储层：使用 MySQL 数据库分别构建用户信息存储数据库和用户检测存储数据库。

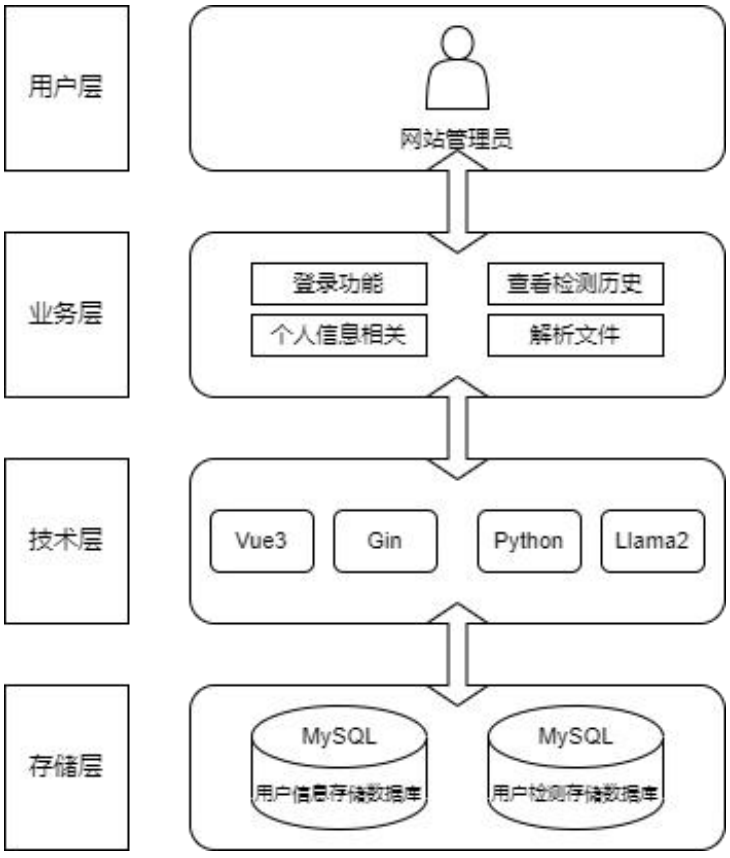

图 3-1

2. 模块设计

模块设计如图 3-2 所示，我们按照功能将其分为登录、用户、解析、历史四大模块。

登录模块：提供了完善的登录工具链，包括账号登录，账号注册和修改密码的功能。账号登录要求用户输入账号、密码，之后通过图形验证码进行人类检测，最后登录；帐号注册要求用户输入用户名、密码和注册使用的邮箱，其中邮箱注册之后无法更改，并且通过我们绑定的自定义域名邮箱 xxx@lzx0626.me 进行邮箱验证，最后注册账号；修改密码同样要求用户通过邮箱验证，最后输入新密码进行重置。

用户模块：存储了用户的基本信息包括用户邮箱、注册时间，还有个性化信息包括姓名、城市、生日、性别、电话、头像等，为用户提供了自定义个性化的服务。

解析模块：接收用户传入的 json 文件，同时可以接收用户自定义输入的权重配置信息，然后通过算法调用，展示解析之后的结果，包括用户环境配置信息、权重配置信息、还有解析之后的各项得分和综合建议。

历史模块：罗列了用户的解析记录，点击详情可以跳转到某次的解析结果；另将最近七次的解析反馈数据绘制为折线图，展示用户解析结果的变化趋势。

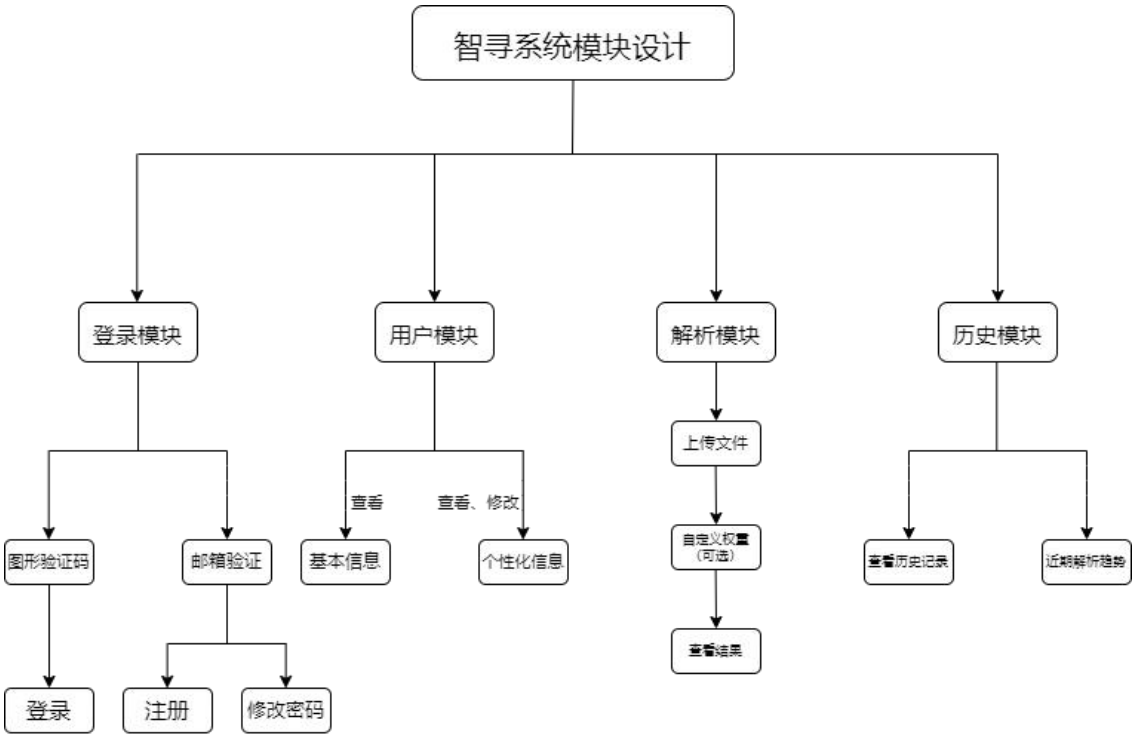

图 3-2

其中解析模块的具体流程如图 3-3 所示：

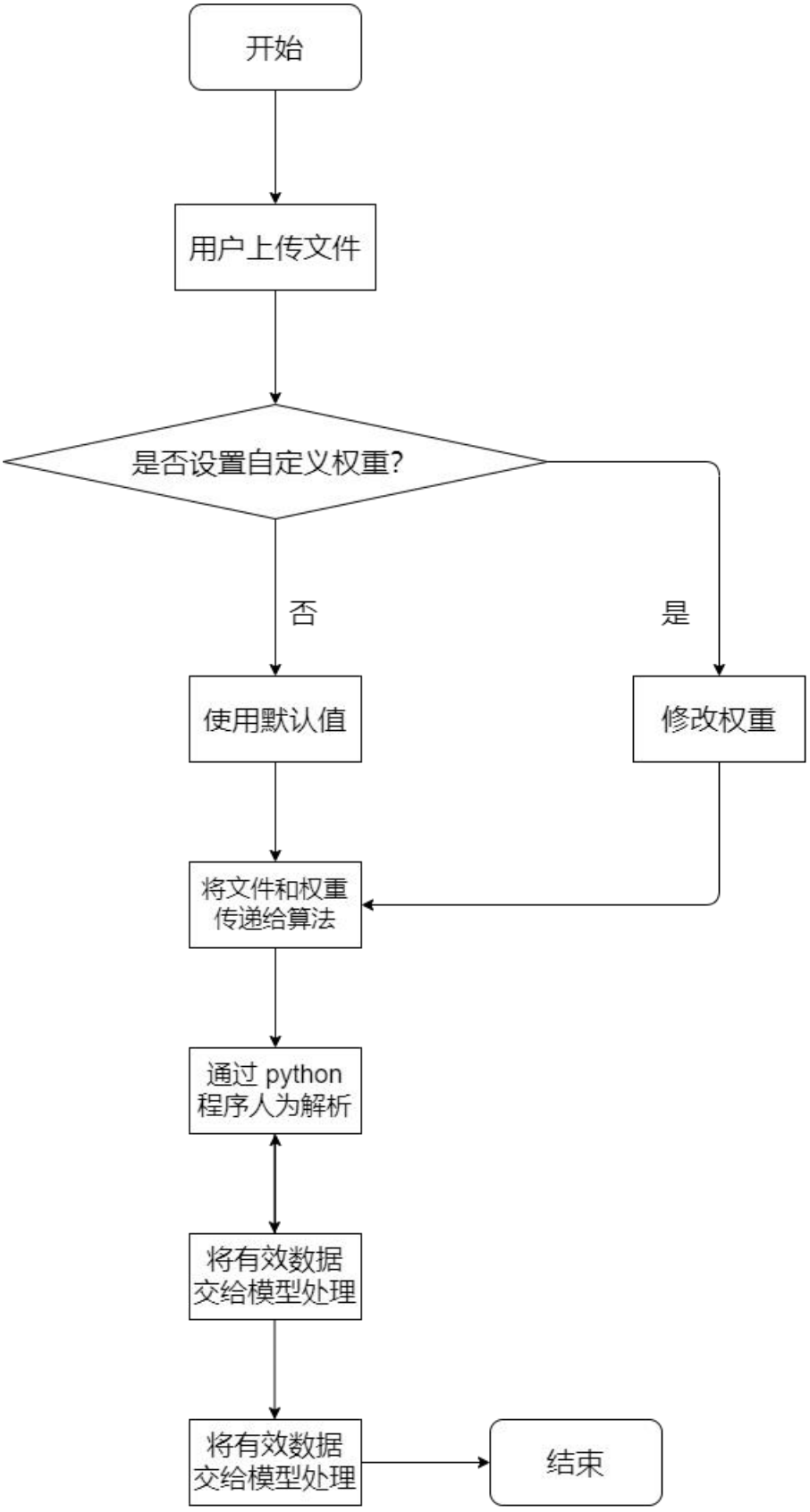

图 3-3

3.2.2 开发工具与技术

1. 开发工具：阿里云云服务器、VSCode、chrome 浏览器、GoLand、DataGrip；

2. 开发技术：Vue3、Element-Plus、flask、openAI 接口、Gin、MySql、Redis、Nginx、Transformers、Pytorch、AutoGPTQ、ApiFox；

详细的开发工具和技术见表 3-1：

环境名称	配置参数或版本号
云服务器	阿里云轻量应用服务器/2 核 2G
操作系统	Ubuntu 22.04
VsCode	1.87.2
chrome 浏览器	123.0.6312.106
GoLand	2023.0.1
DataGrip	2023.3.4
Vue	3.0
Element-Plus	2.5.0
flask	3.0.2
openAI	chatGPt3.5-Turbo
Gin	1.91
MySql	5.5.62
Redis	7.2.4
Nginx	1.23.4

Transformers	4.39.0
Pytorch	2.0
AutoGPTQ	0.7.0
ApiFox	2.4.11

3.2.3 主要技术选择

前端技术

1. 前端使用 VSCode 与 chrome 进行开发，同时利用 icon-font、Get started with Boostrap 等实用型工具进行构建。

Vue3：由于 Vue3 新的编译器能生成更小、更快的运行时代码，这意味着更快的加载和渲染速度。同时 Vue3 对 TypeScript 的支持更加友好，包括更好的类型推断和更清晰的类型定义，接着 vue3 通过 ES Module 的静态分析，实现了更好的 Tree-Shaking，使得只有实际使用到的代码才会被打包，减小了应用的体积。而且 Vue3 应式系统更加灵活，支持更多种类的数据响应式，比如响应式 Refs 和响应式对象。

2. 使用 Element-Plus 组件。

Element Plus 是基于 Vue 3 开发的，充分利用了 Vue 3 的新特性，例如 Composition API，提供了更好的性能和开发体验，同时相比于 Element UI，更加轻量级，精简了代码并优化了性能，减少了包的体积，提高了加载速度。而且 Element-Plus 支持 Tree-Shaking，能够按需加载组件和样式，减少了整体的体积，优化了应用的性能。并且 Element-Plus 提供了丰富的 UI 组件，涵盖了常用的表单、布局、导航、数据展示等功能，能够项目的所有需求。

后端技术

1. 后端使用 go 作为主要开发语言，使用 go 的主流 web 框架 gin 框架，利用 MySQL、Redis 进行存储和缓存，设计了完善的鉴权、限流、路由等中间件，很好地实现了项目核心功能。

2. Golang：Go 语言以其简洁、高效、并发支持和优秀的性能成为了构建现代软件的理想选择之一，其语法设计简洁清晰，易于理解和学习。它摒弃了一些复杂的特性和语法糖，使得代码更易读、易维护。 Go 语言内置了轻量级的并发支持，通过 goroutine 和 channel 实现并发编程变得简单高效。这使得 Go 语言在处理高并发、大规模并行任务时非常有效，特别适用于构建网络服务和分布

式系统。由于对于协程的支持，Go 语言的编译器和运行时性能优秀，生成的执行文件体积小、启动速度快。Go 语言还支持跨平台编译，可以轻松地在不同操作系统上进行部署。同时，Go 语言独特的 Go Get 机制，使得开发者能够很方便地调用第三方库，具有健壮的社区有支持性。

3. 存储系统：在数据存储方面，本项目主要使用关系数据库 MySQL 和非关系数据库 Redis 作为存储载体。

a) 本项目中使用 MySQL 数据库存储主要数据，如用户信息，文件解析结果等。MySQL 是一种成熟、稳定的关系型数据库管理系统（RDBMS），拥有长期的发展历史和广泛的应用基础，可以在各种操作系统上运行，包括 Windows、Linux、macOS 等，具有良好的跨平台性。MySQL 提供了丰富的功能和特性，包括事务支持、触发器、存储过程、视图等，可以满足各种复杂的业务需求，在处理大量数据时具有良好的性能表现，能够有效地处理高并发访问和复杂查询。

b) 在存储方面，Redis 主要被用用来辅助校验验证码是否正确。Redis 数据存储在内存中，因此读写速度非常快，适用于对响应速度要求高的应用场景。Redis 支持多种数据结构，如字符串、哈希、列表、集合、有序集合等，可以满足各种复杂的数据存储和操作需求。Redis 支持多个客户端并发访问，并提供了原子性操作和事务支持，能够处理高并发的请求。

4. 缓存机制：由于项目中部分数据的访问量较大，为了减轻了对数据库等资源的频繁访问，降低系统的负载，提高了系统的稳定性和可靠性。项目使用了基于 LRU 和 Redis 的两级缓存系统，将一些可能会被频繁访问的数据（如用户设置，用户信息等）缓存下来，这样的实现相信能够在一定程度上加快数据的加载速度，使得用户能够更快地获取到所需的信息，从而提高用户的满意度和体验。

a) 基于 LRU 的内存缓存：LRU（Least Recently Used，最近最少使用）是一种常见的缓存机制。LRU 缓存算法的实现相对简单，易于理解和部署。其核心思想是基于数据的使用频率，将最近最少使用的数据替换掉。LRU 缓存算法能够在缓存空间不足时快速且准确地识别出最近最少使用的数据项进行淘汰，保证了缓存空间被高频使用的数据所占用，从而最大程度地提高了缓存命中率。

b) 基于 Redis 的内存缓存：Redis 将数据存储在内存中，并且采用高效的数据结构和算法，因此具有非常快速的读写性能，适用于对响应时间有要求的场景。

算法技术

1. 人为解析

a) 通过构建 python 项目来实现对 json 文件的解析，python 内置的 json 模块能够方便将文件读取的字符串转化为 python 语言的字典类型，通过简单的语法就能够取出对应的数据。

b) 在 python 项目中，第三方模块的引入是非常频繁的事情，但是 python 通过 pip 包管理器能够非常方便的管理第三方包，也就能使用更多的模块，来完成我们的功能。

2. 模型处理

a) 算法的评估响应方面使用开源大模型 Llama2，该模型经过大规模的预训练，在各种 NLP 任务上具有优秀的性能。相比于其他大模型，Llama2 具有轻量和高效的优点，在相同的计算条件下能达到更好的性能并提供更快速的服务。[2]算法采用模型微调和量化技术，增强了 Llama2 在中文下的性能和表现能力，使其能更高效、准确地提供文本服务。

b) 模型运行采用 Transformers 库，这是 Huggingface 开发的高性能模型推理和运行框架。运行时采用 Flash Attention 2[2]技术，该技术应用了 tiling 技术来减少内存访问，可以将内存开销降低到线性级别，实现了数倍的加速，同时避免了对中间结果的频繁读写，从而提高了计算效率。Flash Attention 2 在 Flash Attention 算法基础上进行了调整，减少了非矩阵乘法运算（non-matmul）的 FLOPs，实现了 200%的效率提升。

c) 大型语言模型在常规机器上的运行效率比较低，并且因为模型参数规模庞大，常规的模型量化方法会影响模型的响应效果。使用 AutoGPTQ 工具对它进行量化转换，采用去年推出的模型量化技术 GPTQ。GPTQ[3]作为一种高效的一次性量化技术，能够在相对较短的时间内将大规模 LLM 模型的参数量化为较低的位宽，无需对模型进行重新训练。GPTQ 能够在几乎不影响模型准确性的情况下将参数位宽减小，压缩后的模型仍然能够保持与未压缩相近的表现水平。该技术的应用能使较复杂的模型在部署环境的单显卡上运行。

其他方面

1. 测试工具

测试方面主要主要选择接口管理工具 apifox 来测试。apifox 是 API 文档、API 调试、API Mock、API 自动化测试一体化协作平台，更先进的 API 设计/开发/测试工具，定位 Postman + Swagger + Mock + JMeter。作为前后端分离的项

目，使用 apifox 作为接口管理工具。项目开发中接口管理如下图：

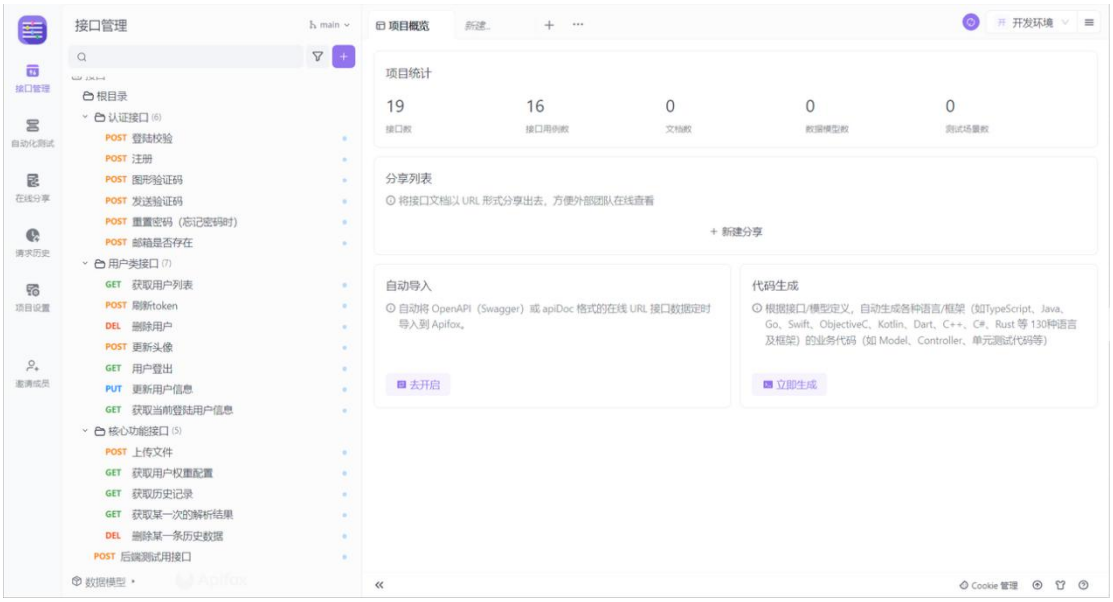

图 3-1 项目接口的开发管理

apifox 在本项目开发中主要有如下作用：

a) 接口文档。开发人员可以在 apifox 中定义接口的基本信息：如 url,请求参数，返回参数等。明确这些信息有助于提高开发效率。如下图是本项目开发中的接口管理。

b) 测试工具。apifox 可以模拟前端向后端发送请求，这使得后端开发者多了一种自测的工具，可以更加便捷地模拟网络请求，特别是 post 类请求。

2. 部署

为了更好地使项目落地，方便用户访问，同时提高开发测试效率，我们选择租用了阿里云服务器。在项目初步开发后，需要将项目部署到服务器上，以此将资源暴露在公网中，供用户访问使用。

a) 服务器操作系统内核。服务器内核为 CentOS7，CentOS 7 因其稳定性、性能优化、安全性、技术支持、兼容性、成本效益、易管理性、长期支持、丰富的文档和资源以及容器支持等优点，成为了许多用户选择云服务器内核时的优先选项。

b) 服务器可视化面板。为了方便服务器的管理，安装了宝塔可视化面板。

c) nginx 路由转发。为了更好地控制接口的请求，配置了 nginx 进行路由转发，Nginx 在处理静态文件和反向代理方面具有很高的性能，它使用事件驱动的异步非阻塞模型，可以同时处理大量并发连接，而不会像传统的服务器那样为每个请求生成一个新的进程或线程。对于 Nginx 的配

置如下：

```
server
{
    listen 80;
    server_name _;  #_是指所有的域名匹配
    location /{
        proxy_pass http://127.0.0.1:8080/;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
 }
location /avatars {
    root /www/wwwroot/backend/public/uploads/;#指定图片存放路径
  }
}
```

代码 3-1 nginx 配置

3.2.4 项目整体规划和开发进程

1. 团队组织方式

a) 在赛题确定初期，小组成员开会讨论项目想法、思路等，并按照功能完成分工。

b) 小组每周五晚上进行例会，会上每个人汇报本周进展以及下周工作安排，并针对焦点功能或问题进行讨论，提出解决思路或方案。

c) 使用 git 托管代码，项目在 github 上开源，实现版本控制和团队协作

2. 项目时间节点

a) 2.19 - 2.29：完成项目的大体逻辑设计，前后端、算法部分对各自部分进行各自的设计并相互交流对接，形成需求文档与实现方案。

b) 3.1 - 3.17：前端完成页面框架的搭建，后端完成大部分接口的编写，算法部分完成初版实现。

c) 3.18 - 3.31：前端完成功能的完善和页面的优化，后端完成接口的编写，并于前端和算法进行对接测试，算法进行优化修改。

d) 4.1 - 4.15：完成后续收尾工作，完善项目文档、PPT 和演示视频，并讨论和检查项目细节漏洞，确认无误后提交项目。

3. 任务分工

成员姓名	身份	负责内容
刘治学	队长	1. 确认项目进度，设立时间节点 2. 协助前端和算法部分同学完成开发 3. 主要负责编写项目文档和演示视频的录制剪辑，一起协作完成 PPT 的制作
李航宇	副队长	1. 负责 go-lang 后端，完成登录页面、文件上传等功能的实现

		2. 编写项目文档复杂工程问题的后端部分 3. 协助完成演示视频的录制，协作完成 PPT 的制作
李远行	队员	1. 负责 vue 前端，完成登录界面的相关工作，同时协助完成核心页面的工作 2. 编写项目文档复杂工程问题的前端部分 3. 协作完成 PPT 的制作
郑笑宇	队员	1. 负责 vue 前端，完成核心页面和功能的设计、实现 2. 编写项目文档复杂工程问题的前端部分 3. 协作完成 PPT 的制作
吉劲帆	队员	1. 负责算法部分，完成文件的解析、数据的分析，最终返回检测结果 2. 编写项目文档复杂工程问题的算法部分 3. 协助完成 PPT 的制作

4. 周会记录

3.3 周会记录	1. 前后端进行设计交流，接口确认 2. 算法部分解读 json 文件
3.10 周会记录	1. 前端部分编写登录功能和核心功能

	2. 后端部分确认核心功能总体设计 3. 算法部分提出 json 文件新思路，确认实施方向
3.15 周会记录	1. 前端部分完成部分接口的工作 2. 后端部分完成核心功能的编写，如文件上传、登录验证等 3. 算法部分完成算法初版，等待后续调试和对接
3.24 周会记录	1. 前端部分对接核心功能部分的图标设计，流程沟通 2. 后端部分基本完成功能，准备与前端和算法部分进行对接 3. 算法部分优化算法部分功能和流程
3.30 周会记录	1. 前端部分完成核心功能的开发，与后端进行对接，等待后续页面优化 2. 后端部分与算法部分完成对接，并正在和前端进行对接 3. 算法部分作优化修改

3.3 本章小结

本章从可行性分析和实施方案两个方面进行阐述，主要介绍了项目实现前的分析方案和实现的具体细节。首先确定了项目的整体架构，接着介绍了项目中的各个模块，对应了智寻系统的各个功能；接着是对系统的开发工具和技术进行分析，针对技术选型做了分析。

4 复杂工程问题归纳

本章主要讲述复杂问题归纳和存在问题与解决方案，提出项目难点以及问题，并对存在问题进行分析和解决。

4.1 复杂工程问题归纳

在本项目的设计与开发中，由于团队水平有限，遇到了一些富有挑战并且认为值得记录的复杂工程问题，这些问题的提出与解决极大地推动了项目的进展，保证了项目的质量。

4.1.1 前端方面

1. 利用 Pinia 实现的本地持久化数据存储

a) 在登录页面登录成功后，后端会相应数据 Token，但是这个数据如果随路由跳转并不安全，于是本系统引入 Pinia 实现 Token 的本地持久化存储管理，在后续页面中直接从本地仓库获取 Token 并置于请求头，这样数据会更加安全。

b) 页面通过内部内容与外部 Container 框架构成，即内部页面的具体操作与请求与外部框架相割离，这样会导致内外部数据渲染的不同步，通过 Pinia，可以将页面内部接收的数据暂存同时外部框架通过从本地持久化仓库获取到该数据，二者获取到数据后同时刷新，即可实现内外部同步数据渲染。

2. 页面得分的权重设计

a) 不同用户对页面关心的侧重点不同，大概可以分为报错、加载慢、无响应等八个类别以及多个问题同时出现。

b) 用户可以为这九个选项配置不同的权重值，但是页面中如果全部展示会显得臃肿，应该设计为按需展示，即用户可以自由选择配置权重项的数量，页面也展示对应数量的操作组件，这样也可以使页面更简洁。

c) 可以展示的操作组件最多不超过九个，并且构成为一个下拉组件，一个数值输入框和一个删除按钮。

d) 对每个下拉组件的选项进行操作，使其在点击时不再展示已经在其他组件中被选中的权重名。

3. 部署到服务器上的样式异常

a) 通过深度选择器和 !important 属性来保证样式的正常渲染。

4.1.2 后端方面

1. 文件上传和管理系统

a) 文件上传接口的设计：限制上传的频率，设置合适的上传限制，防止别有用心的人恶意访问导致服务器负载过大。

b) 文件系统的管理：合理存储用户上传文件，存储用户上传文件的记录。

2. 用户上传待分析的 json 文件中包含网站访问者的信息和其遇到的问题，需要对这些信息进行筛选清洗。

a) 复杂 json 的解析：用户上传的 json 文件可能结构较为复杂，字段较多，嵌套结构较深。此时如果直接将信息传递给算法模型，会导致算法效率下降，降低判断的准确率，最终影响用户体验。

b) 合理存储文件中的有效信息：用户上传的 json 文件中主要包含两个部分：网站访问者的个人信息（如 ip，city 等）和访问者在访问时遇到的问题，后端需要合理的存储这些信息。

3. 算法模型的对接：为了保证项目运行的流畅性，算法模型并不会直接和前端进行交互，其函数的调用以及返回都是直接和后端进行交互。因此后端需要考虑如何正确的调用算法，传递参数并获取到解析结果。

4. 缓存系统的设计

a) LRU 算法的实现与改进：LRU 作为系统的第一层缓存，存储了系统中最容易被访问的数据。如何合理实现第一层缓存与第二层缓存的交互。

b) Redis 订阅机制的实现。

4.1.3 算法方面

1. Python 项目的管理

a) 版本问题：在不同的环境上，python 的版本可能不同，python 的包管理器 pip 版本可能不同，使用的第三方模块版本也可能不同，如何做到统一就是比较棘手的事情了。

b) 文件结构问题：无论是什么项目，合理的文件结构是必不可少的。如何合理的设计、分配代码的架构图，一定是整个开发过程中的重中之重。

c) 路径问题：对于 Python 而言，由于是解释型语言，因此一个 Python 文件相当于就是一个可执行文件，在不同的工作目录下执行同一个 Python

文件可能会导致路径出现各种问题，这也是需要注意的点。

2. 用户体验问题的设计和检测

a) 用户提供的日志文件里面条条项项这么多，如何对用户的问题进行归类总结就是一个问题了，这也关系到后续解析和展示的相关问题。

b) 对问题进行归类以后，如何计算得分就是一个问题了，因此还需要涉及到问题之间的权重问题。

3. 提供网络接口

算法需要提供网络接口供后端进行调用，合理的接口设计关系到参数的传递以及结果相应的返回。

a) web 框架的选择。python 部分有许多成熟的 web 框架，例如 flask，django 等，选择合适的 web 框架来构建网络请求是接口设计的基础。

b) 接口设计。接口设计需要和后端对接请求参数和响应参数，需要确定参数的格式以及参数名字。

4. 模型响应服务的部署

本地模型在部署至 GPU 设备后需要设计响应服务的交互格式和模型调用方案，模型推理所需要的参数和 prompt 应分类和收集之后传入并进行一次模型推理，要保证模型推理结果贴近应用的需要，服务接口的设计合理易用。

4.2 存在问题与解决方案

4.2.1 前端方面

1. Token 的传递与存储：

问题：在登录页面登陆成功后，后端会返回 Token 数据，但是如果将这个数据随路由跳转并不安全。

解决：

a) 本系统引入 Pinia 对数据进行本地持久化存储管理，将登录成功后返回的 Token 暂存于本地用户仓库。

```
import { useUserStore } from "@/stores/modules/user"
import { useRouter } form "vue-router"

const loginForm = ref()
const userStore = useUserStore()
const router = uesRouter()

// 登录操作
const doLogin = async () => {

  // 校验登录表单信息
  loginForm.value.validate()
   ...

  // 接收响应数据
  const res = await getLoginToken(loginForm.value)
   ...

  // 将 Token 存入 Pinia 本地持久化仓库
  userStore.setToken(res.data.Token)

  // 登陆成功，路由跳转到用户信息页
  router.push("/user/profile")
}
```

b) 在后续页面操作时便可以直接从本地仓库中获取 Token 置于请求头来发送请求，这样数据也会更加安全。

```
import { useUserStore } from '@/stores/modules/user'
import axios from 'axios'

const baseURL = 'http://...'
const userStore = ref()
```

```
6.  const instance = axios.create({
7.   // 基础地址，超时时间
8.   baseURL,
9.   timeout: 10000
10. })
11.
12. // 请求拦截器
13. instance.interceptors.request.use(
14.   async (config) => {
15.
16.     // 判断本地持久化仓库是否有数据
17.     if (!userStore.value) {
18.       userStore.value = useUserStore()
19.     }
20.
21.     // 判断仓库内是否有暂存的 Token
22.     if (userStore.value.token) {
23.       // 如果有 Token，则将其配置于请求头的 Authorization
24.       config.headers.Authorization = userStore.value.token
25.     }
26.     return config
27.   },
28.
29.   // 如果请求有异常，则拦截此次请求
30.   (err) => Promise.reject(err)
31. )
```

2. 页面内部图片渲染与外部 Container 框架用户头像渲染

问题：由于使用 Container 搭建了页面整体框架，所以其与页面内部的具体操作和请求相割离，那么在页面内通过 axios 发送请求获取到相应数据并实时渲染时，外部框架并不能实时获取到响应值，导致内外渲染不同步。

解决：

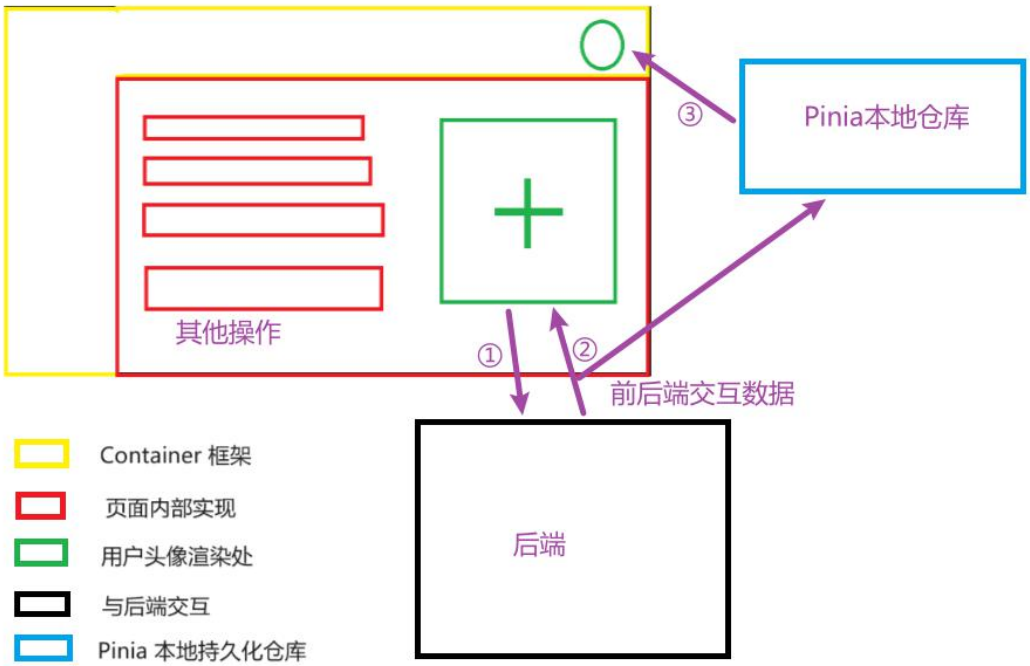

图 4.2.1-1 页面内部与外部框架渲染示意图

a) 配置本地持久化仓库 User 中有关用户头像的方法

```
import { defineStore } from 'pinia'
import { userGetInfoService } from '@/api/user'

// 定义本地持久化仓库来暂存数据
export const useUserStore = defineStore(
  'user', () => {

    const user = ref({})
    // 配置 userStore 的各种方法
    const setUser = (obj) => {
      user.value = obj
    }
    const getUser = () => {
      const res = await userGetInfoService()
      user.value = res.data.data
    }
     ...

    return { user, getUser, setUser, ... }
  }
)
```

b) 页面内仍然通过 axios 交互的方式响应式获取后端数据并渲染，在交互的同时通过 Pinia 将数据进行本地暂存。随后，在页面中接收到请求后，分别调用页面内的刷新方法和持久化仓库的刷新方法，使页面内和框架都能被响应数据渲染。

```vue
import { useUserStore } from '@/stores'
import PageContainer from '@/components/PageContainer.vue'
import { userUpdateAvatarService } from '@/api/user'

const userStore = useUserStore()
const uploadRef = ref()
const imgUrl = ref(userStore.user.user_pic)

// 更新用户头像方法
const onUpdateAvatar = async () => {

  // 将用户头像添加到请求数据中
  const formData = new FormData()
  formData.append('avatar', uploadRef.value)

  try {
    // 发送请求更新头像
    await userUpdateAvatarService(formData)

    // 图片信息暂存于 userStore
    userStore.setUser({ user_pic: imgUrl.value })
    // 通过 userStore 重新渲染外部 Container 框架中的用户头像
    userStore.getUser()

    // 提示用户
    ElMessage.success('头像更新成功')
  }
  catch (error) {
    ElMessage.error('更新头像失败，请稍后重试')
  }
}
```

3. 在解析记录展示表中，实现点击某条记录，会跳转到这条记录的解析详情页

解决：

a) 在解析记录页配置点击事件，在路由跳转时传递参数

```
import { useRouter } from 'vue-router'

const router = useRouter()

// 添加解析
const addJsonSolve = () => {
  router.push('/json/solve')
}

// 路由跳转并携带 file_name 参数
const onShow = (row) => {
  router.push({
    path: '/json/show',
    query: {file_name: row.file_name}
  })
}
```

b) 在跳转目标页面，即解析详情页接收路由跳转参数并作为获取信息详情请求参数发送请求

```
import { useRoute } from 'vue-router'
import { getJsonSolveData } from '@/api/json.js'

// 接收路由跳转参数
const props = defineProps({
 file_name: {
  type: String,
  required: true
 }
})

const route = useRoute()
const file_name = ref('')

// 获取解析数据
const getJsonSolveChartData = async () => {

 // 通过 file_name 作为参数发送请求
 const { data: { data } } = await getJsonSolveData(file_name.value)
 // 接收参数
 xxxData = data.xxx
  ...
```

```
23.
24. }
25.
26. // 在 ECharts 组件实例创建前获取其渲染参数
27. onMounted( async () => {
28.  // 获取路由传参
29.  file_name.value = route.query.file_name
30.
31.  // 获取解析数据
32.  await getJsonSolveChartData()
33.
34.  // 通过数据渲染 ECharts 图表
35.  ...
36. }
```

4. 在前端项目的本地构建时运行正常但是在部署到服务器上后组件却出现了样式异常。

解决：通过深度选择器和 !important 属性确保此处渲染被执行。

```
1.  /* 使用深度选择器来覆盖 Element UI 组件的内部样式 */
2.  ::v-deep(.el-select .el-input__inner) {
3.   width: 200px !important;
4.  }
5.  ::v-deep(.el-input__inner) {
6.   width: 200px !important;
7.  }
```

5. 用户共有 9 个选项可以为其分别配置不同比重，但是页面同时展示 9 个 input 输入框又会显得页面很臃肿。

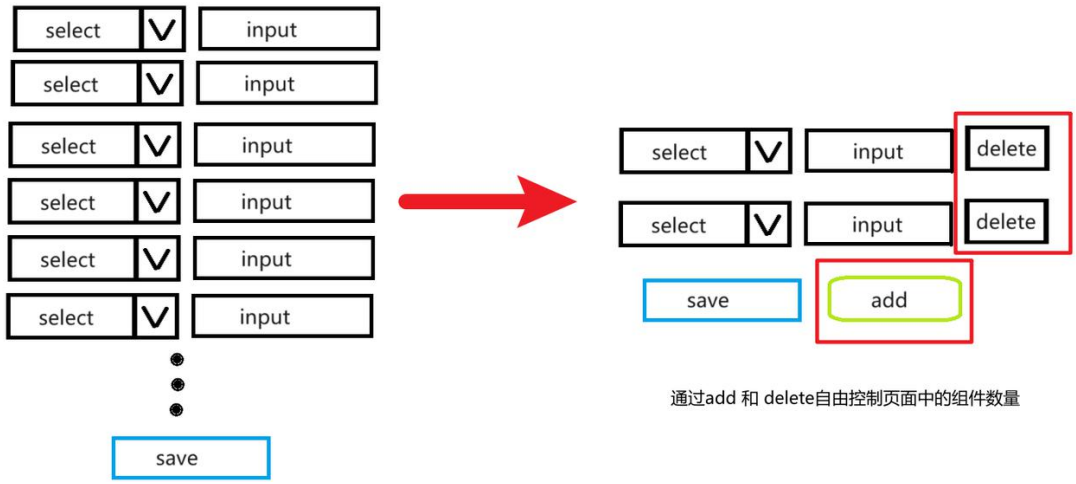

图 4.2.1-2 自定义权重组件优化示意图

解决：

a) 通过新增配置选项和删除配置选项按钮，使用户可以自行调节配置权重项的数量

```
<el-form-item>
  <div>
    <el-select></el-select>
    <el-input></el-input>
    <el-button type="danger" @click="removeItem(index)">删除</el-button>
  </div>
</el-form-item>

<el-form-item>
  <el-button type="primary" @click="submitForm">保存配置</el-button>
  <el-button @click="addItem">新增权重配置</el-button>
</el-form-item>


// 用户配置表单
const formRef = ref()

// 默认展示两个权重值配置组件
const count = ref(2)
const formModel = ref({
 items: [{
  name: '',
  word: '',
  value: '50'
 }, {
  name: '',
  word: '',
  value: '50'
 }]
})

const addItem = () => {
 if (count.value >= 9) {
  ElMessage.error('最多只能添加 9 个权重配置项')
  return
 }
 formModel.value.items.push({ word: '', name: '', value: '' })
 selectedItems.value.push({ word: '' })
 count.value ++
}

const removeItem = (index) => {
 formModel.value.items.splice(index, 1)
 count.value --
}
```

b) 权重配置部分的框架为一个 el-select 对应一个 el-input 的循环遍历展示，在下拉表的选项管理中，已经被选中的权重选项将不再出现在其他下拉表中

```
<el-form-item
  v-for="(item, index) in formModel.items"
  :key="index"
  :prop="'items.' + index + '.word'"

 >
  <div>
   <el-select v-model="selectedItems[index].word"
   @change="handleSelectChange(index)"
   placeholder="请选择权重属性">
    <el-option
     v-
for="(option, optionIndex) in getAvailableOptions(index)"
     :key="optionIndex"
     :label="option.word"
     :value="option.word">
    </el-option>
   </el-select>
   <el-input
    v-model="item.value"
    type="number"
    placeholder="请输入 0~100 之间的数值"></el-input>
   <el-button >删除</el-button>
  </div>
 </el-form-item>


// 遍历选中的选项
const selectedItems = ref(formModel.value.items
  ? formModel.value.items.map(item => ({ word: item.word ??
'' }))
  : [{ word: '', value: '', name: ''}])

// 权重配置表单数据发生变化时
const handleSelectChange = (index) => {
 if (formModel.value && formModel.value.items) {

 // 获取当前选中的选项
 const selectedItem = selectedItems.value[index];
 const duplicatewords = selectedItems.value.filter((item,
idx) =>
    idx !== index && item.word === selectedItem.word)
}

```

```
// 获取可供选择的权重属性(去除已经被选中的)
const getAvailableOptions = computed(() => {
 return (index) => {

  // 获取当前所有被选中的 word, 并过滤掉空字符串
  const selectedWords = selectedItems.value.map(item => item.word)
                .filter(word => word);
  // 如果一个都没有被选中, 返回 ratioList
  if (selectedWords.length === 0) {
   return ratioList.value.map(option => ({ word: option.word}))
  }

  // 否则, 过滤 ratioList 排除已经被选中的 word
  const excludedWords = [...selectedWords.slice(0, index), ...selectedWords.slice(index + 1)];
  return ratioList.value.map(option => ({ word: option.word})).filter(item => !excludedWords.includes(item.word));
 };
});
```

4.2.2 后端方面

1. 后端和算法交互问题。

由于算法模型主要是使用 python 开发，后端主要使用 golang 开发，因此如何让后端和算法直接正常的交互是项目中较为重要，也是十分影响效率的一个问题。在开发前，我们经过了大量的调研以及测试，发现有两种较为可行的方案：

a) 通过 golang 的第三方包 go-python 直接调用 python 算法。Python 提供了丰富的 C-API。而 C 和 Go 又可以通过 cgo 无缝集成。所以，直接通过 Golang 调用 libpython，就可以实现 Go 调 Python 的功能了。但是过程比较复杂，而 go-python 提供了针对 CPython-2 的 C-API 提供了 native-binding 能力，方便实现了 Go 到 Python 的调用。这样的实现方式能够保证算法模型的原生性与项目服务的连贯统一，使得 python 算法和 golang 后端能够运行在同一个服务中。然而，这样的调用，可能会存在性能的损耗，造成对 cpu 和内存不必要的开销。Python 和 Go 在运行时有不同的性能特性，调用 Python 算法可能导致额外的开销。使用第三方库 go-python 可能会增加项目的依赖性和复杂性，可能需要额外的配置和管理。

b) 使用 Flask 框架在 Python 中创建 API，然后通过 Go 请求 Flask 的 API。因此，我们选择了第二种方案，在算法模型之上构建一个轻量的 web 服务，提供算法的 api 接口，通过网络请求的方式在后端和算法模型中传递参数和结果。经过调研和讨论，我们发现这种方案有如下优势：

   i. Flask 框架简单灵活。Flask 是一个轻量级的 Python Web 框架，核心代码量很小，因此它运行起来非常快速，并且对系统资源的消耗也比较低。这使得它成为构建轻量级 Web 应用或微服务的理想选择。

   ii. 由于整个应用程序都是用 Go 编写的，在不考虑网络路由时，请求 Flask API 可能比在 Go 中直接调用 Python 算法更加高效。Go 在处理并发和性能方面通常比 Python 更有效率。

   iii. 通过将算法封装在 Flask API 中，可以更轻松地将其部署为分布式系统，因为可以通过网络在不同的机器上访问 Flask 服务。为了节省成本并提高算法响应效率，我们选择在本地构建和运行算法服务，通过内网穿透的方式将 api 暴露给指定 ip，使得前后端服务所

在服务器网关能够访问到私有 ip，后端能够成功请求到对应接口。

```
1.  req := HttpRequest.NewRequest()
2.  req.SetHeaders(map[string]string{"Content-
    Type": "application/json"})
3.  data := map[string]interface{}{}
4.  url := viper.GetString("Algorithm.url")
5.  res, err := req.Post(url, data)
```

代码 4-1 后端请求算法 api

iv. 使用 Flask API 可以实现更松散的耦合，因为两个不同的语言之间的交互通过 HTTP 协议进行，这样更容易理解和维护。

v. 通过将算法封装在 Flask API 中，可以更灵活地进行部署和维护。例如，可以将 Flask 服务部署在不同的服务器上，而不影响 Go 客户端的部署。

因此，在需要调用算法时，本系统的流程图如下：

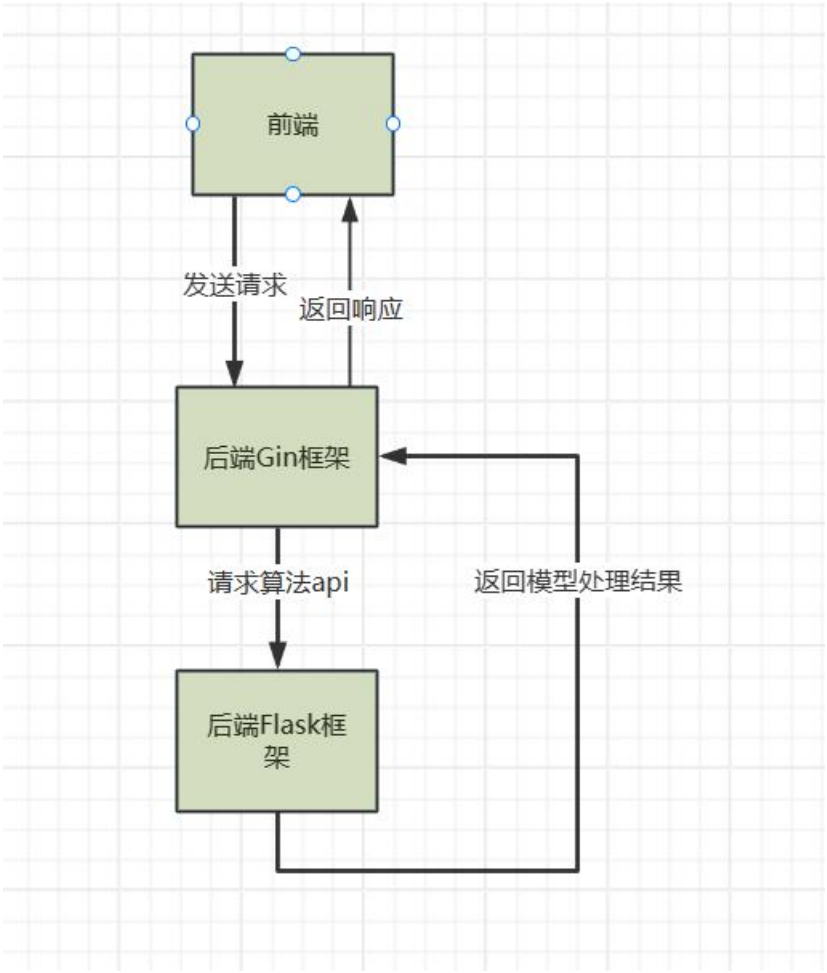

图 4-1 调用算法时流程图

2. 前端和后端交互问题。

为了保证技术栈的灵活性和可扩展性，我们采用了前后端分离的 web 开发思想。前后端分离允许前端和后端使用不同的技术栈。前端可以选择适合其需求和团队技能的技术，而后端也可以独立选择适合的技术。这样可以更好地发挥团队成员的专长，提高开发效率和质量。 因此让前端和后端健康的交互是尤为重要的：前端能够真正确发送请求到后端并且根据后端的响应做出反馈，后端能成功接收到前端的请求并在经过处理后返回正确的响应。为了解决这个问题，我们选择了以下方案：

a) 请求接口的合理设计。

i. 保障数据传输效率。合理设计的接口可以减少数据传输的量和频率，提高数据传输的效率。例如，通过合并接口请求、压缩数据、使用分页和缓存等技术，可以减少网络请求的数量和数据的大小，从而降低系统的负载和响应时长。

ii. 系统可扩展性：良好设计的接口可以提高系统的可扩展性。通过模块化和抽象化的设计，可以降低系统各个模块之间的耦合度，使得系统更易于扩展和维护。

b) 接口管理工具的正确使用。

为了更好的管理接口，并测试接口功能，我们选择了主流开发软件 apifox 作为接口管理工具，使用 apifox 有如下优势：

i. 规范的接口文档。开发者能在 apifox 中规定接口的 url，请求参数以及不同情况下的响应。

ii. 便于测试。apifox 能够直接模拟前端发送请求，能够帮助后端开发者测试自己接口是否符合预期，同时前端开发者通过 apifox 也能查看接口运行情况，方便开发。

c) 合理利用网络状态码。网络状态码是一种标准化的通信方式，通过状态码可以明确地指示请求的结果和服务器端的处理状态。这有助于客户端和服务器之间的有效通信，并提高了系统的可靠性和可预测性。网络状态码提供了诊断错误的重要线索。当请求失败或出现问题时，状态码可以帮助开发人员更快速地定位问题所在，从而加快排查和修复错误的速度。

3. 一级缓存 LRU 算法的实现。

系统使用了 LRU 算法作为第一级缓存，能够减轻服务器压力，减少响应时间，提高用户体验。在本系统中对于 LRU 的设计如下：

a) 使用一个双向链表和一个哈希表，双向链表用于记录缓存数据的访问顺序，哈希表用于快速查找缓存数据。

b) 初始化：初始化一个双向链表和一个哈希表,链表中的每个节点包含键值对（key, value）,哈希表中的键是缓存的键，值是指向链表节点的指针。

c) 获取数据操作：如果缓存中存在键 key，从哈希表中查找 key 对应的节点，并将其从链表中移动到链表头部，返回节点的值。否则返回-1。

d) 插入数据操作。如果缓存中存在键 key，更新该节点的值，并将其从链表中移动到链表头部。否则，创建一个新的节点，将键值对插入到链表头部。如果缓存已满，从链表尾部删除最近最少使用的节点。将键和新节点的指针插入到哈希表中。

根据以上思想流程，最终设计的 LRU 流程图、简单示例代码如下：

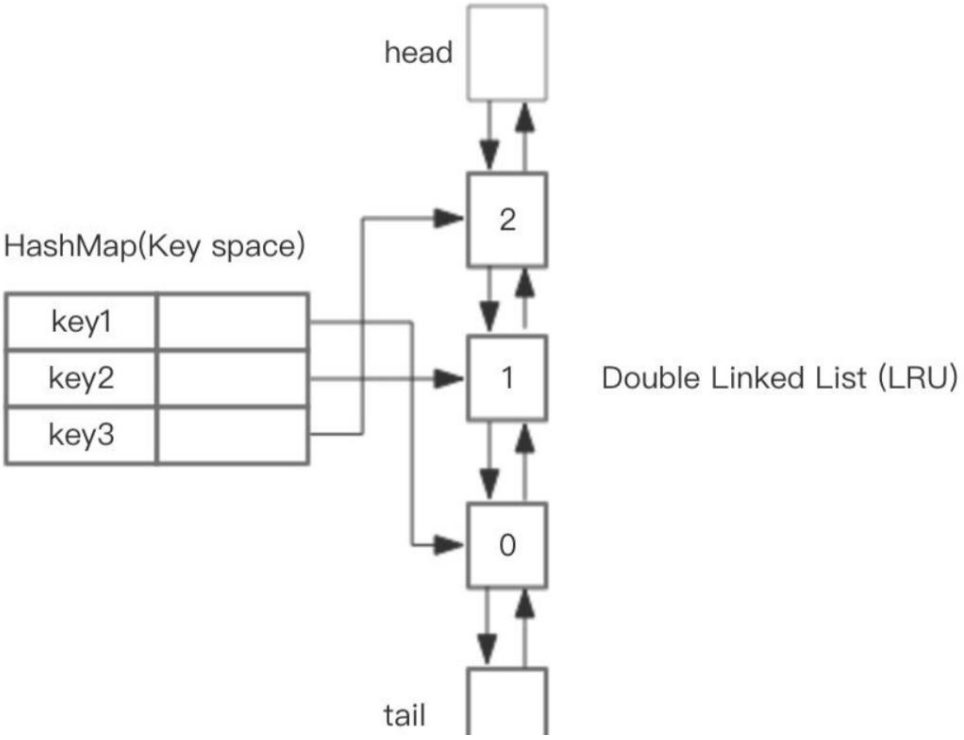

图 4-1 LRU 缓存算法示意图

```
type LRUCache struct {
    capacity int
    cache  map[int]*list.Element
    lruList *list.List
}

type entry struct {
    key  int
    value int
}

func Constructor(capacity int) LRUCache {
    return LRUCache{
        capacity: capacity,
        cache:  make(map[int]*list.Element),
        lruList: list.New(),
    }
}

func (l *LRUCache) Get(key int) int {
    if elem, ok := l.cache[key]; ok {
        l.lruList.MoveToFront(elem)
        return elem.Value.(*entry).value
    }
    return -1
}

func (l *LRUCache) Put(key, value int) {
    if elem, ok := l.cache[key]; ok {
        elem.Value.(*entry).value = value
        l.lruList.MoveToFront(elem)
    } else {
        if len(l.cache) >= l.capacity {
            delete(l.cache, l.lruList.Back().Value.(*entry).key)
            l.lruList.Remove(l.lruList.Back())
        }
        newElem := l.lruList.PushFront(&entry{key, value})
        l.cache[key] = newElem
    }
}
```

4. 缓存的数据同步。

用户在查询获取某种数据时，会优先查询缓存中的数据，但是如果用户想要更改或者删除某条数据，则会优先更改干系数据库中的数据表。因此需要实现缓存和存储中的数据同步，经过调研和商讨，最终确定的数据同步机制如下：

a) 数据查询。用户在获取数据时，如果会优先在缓存中查找数据，如果所查找数据索引没能命中缓存，才会请求到 MySQL 数据库。

   i. 如果命中了缓存。应用程序可以直接从缓存中获取数据，而无需执行数据库查询或其他数据检索操作。这可以大大提高数据访问速度，减少了对底层数据存储的访问压力。并且根据 LRU 算法，将查询数据节点移动到头节点后，刷新了查询数据的时间戳

   ii. 如果没有命中缓存。倘若没有命中缓存，程序将发送请求到 MySQL 数据库进行查询，并且将查询到的数据添加到缓存中，方便下一次可能的查询。

b) 数据更改。当用户发出更改数据的请求时，请求将首先更改 MySQL 数据库中的对应数据，并且触发协程异步任务来更新相关的缓存数据，而不会阻塞当前请求。

c) 缓存数据自动更新。在 Redis 中，通过设置数据的过期时间（TTL，Time To Live）来让数据在一段时间后自动失效，并且可以通过定期刷新缓存来更新数据。在本项目中，设定过期时间为 3600s。

```
1.  SET key value EX 3600
```

d) Redis 订阅机制。Redis 的订阅（Pub/Sub）机制是一种消息传递模式，用于实现发布者（publisher）和订阅者（subscriber）之间的消息通信。以下是订阅机制的示意图：

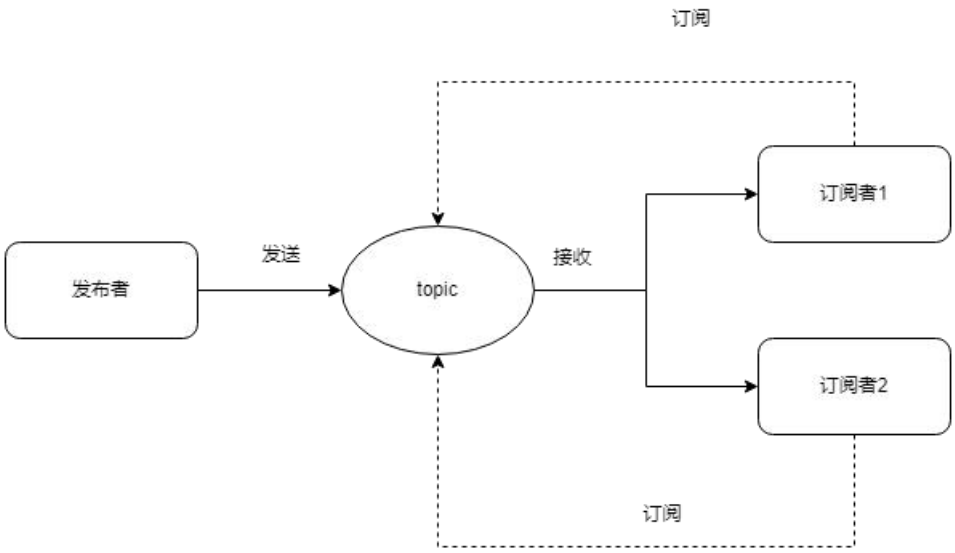

图 4-2 Redis 订阅机制

i. 发布者（Publisher）。发布者负责向指定的频道（channel）发布消息。发布者通过执行 PUBLISH 命令将消息发送到指定的频道。

ii. 订阅者（Subscriber）。订阅者通过执行 SUBSCRIBE 命令订阅一个或多个频道。一旦订阅成功，Redis 就会将订阅者添加到对应频道的订阅列表中。

iii. 当发布者向某个频道发布消息时，Redis 会将该消息发送给所有订阅了该频道的订阅者。如果订阅者同时订阅了多个频道，他们将会接收到每个频道发布的消息。

5. 鉴权、限流中间件的开发。

a) 鉴权中间件。方便权限校验，登陆的用户在访问系统资源时，会在请求头中携带 token 用以校验权限。用用户 token 主要包含用户 id，过期时间以及签名。为了方便权限校验，获取当前登陆用户的状态和信息，后端开发了权限校验中间件，校验用户权限。某些场景需要认证权限（携带正确不过期的 token），某些场景需要非认证权限（不携带 token 或者携带过期 token）。

i. 生成 token。户在登陆时后端会根据 token 生成的算法生成 token 并返回给前端，前端会将 token 存储到用户本地，以便用户后续请求携带。

ii. 校验 token。用户携带 token 请求资源时，后端会解析 token，获取当前登陆的用户，判断 token 是否过期。

iii. 认证权限。对于某些资源的访问需要请求具有认证权限，即携带在有限期内的 token，例如登陆后的一些操作（获取用户个人信息，查看解析历史等），其中对于认证 token 的校验如下：

```
1.  func AuthJWT() gin.HandlerFunc {
2.    return func(c *gin.Context) {
3.      // 从标头 Authorization:Bearer xxxxx 中获取信息，并验证 JWT 的准确性
4.      claims, err := jwt.NewJWT().ParserToken(c)
5.
6.      // JWT 解析失败，有错误发生
7.      if err != nil {
8.        response.Unauthorized(c, fmt.Sprintf("请先登陆再进行操作"))
9.        return
10.     }
11.
12.     // JWT 解析成功，设置用户信息
13.     userModel := user.GetUser(claims.UserID)
```

```
    if userModel.ID == 0 {
      response.Unauthorized(c, "找不到对应用户，用户可能已删除")
      return
    }

    // 将用户信息存入 gin.context 里，后续 auth 包将从这里拿到当前用户数据
    c.Set("current_user_id", userModel.ID)
    c.Set("current_user_name", userModel.Username)
    c.Set("current_user", userModel)

    c.Next()
  }
}
```

iv. 非认证权限。对于一些资源的访问需要请求具有非认证权限，即请求头中不携带 token，对于此类权限的校验如下：

```
func GuestJWT() gin.HandlerFunc {

  return func(c *gin.Context) {
    if len(c.GetHeader("Authorization")) > 0 {

      // 解析 token 成功，说明登录成功了
      _, err := jwt.NewJWT().ParserToken(c)
      if err == nil {
        response.Unauthorized(c, "请使用游客身份访问")
        c.Abort()
        return
      }
    }
    c.Next()

  }
}
```

b) 限流中间件。为了防止“别有用心”的人恶意多次访问网站资源，增大服务器开销，降低其他用户体验，后端做了针对 ip 的访问限制。对于某些开销比较大的操作，会限制用户的访问频率，当然，这个限制不会很低，以免影响用户的正常使用。实现针对 ip 的流量限制，需要经过以下步骤。

i. 获取访问用户的 ip。由于项目基本 web 框架采用 gin 框架，gin 很好地支持了这一操作，对于每一个向服务发送的 ip，都很好地记录在了 gin 的上下文 context 中。因此，获取 ip 的操作是十分便捷的。

```
func GetKeyIP(c *gin.Context) string {
  return c.ClientIP()
```

```
3.  }
```

代码 4-5 获取访问 ip

ii. 在 Redis 中存储 ip 的访问次数，并设定这个键值对的有效时间。

iii. 判定该 ip 的访问次数是否超出限制，如果超出限制，则返回错误码；否则正常接受网络请求。

```
1.  func limitHandler(c *gin.Context, key string, limit string)
    bool {
2.
3.    // 获取超额的情况
4.    rate, err := limiter.CheckRate(c, key, limit)
5.    if err != nil {
6.      logger.LogIf(err)
7.      response.Abort500(c)
8.      return false
9.    }
10.   fmt.Println(rate)
11.   // ---- 设置标头信息-----
12.   // X-RateLimit-Limit :10000 最大访问次数
13.   // X-RateLimit-Remaining :9993 剩余的访问次数
14.   // X-RateLimit-Reset :1513784506 到该时间点，访问次数会重置
    为 X-RateLimit-Limit
15.   c.Header("X-RateLimit-Limit", cast.ToString(rate.Limit))
16.   c.Header("X-RateLimit-
    Remaining", cast.ToString(rate.Remaining))
17.   c.Header("X-RateLimit-Reset", cast.ToString(rate.Reset))
18.
19.   // 超额
20.   if rate.Reached {
21.     // 提示用户超额了
22.     c.AbortWithStatusJSON(http.StatusTooManyRequests, gin.
    H{
23.       "message": "请求太频繁",
24.     })
25.     return false
26.   }
27.
28.   return true
29. }
```

代码 4-6 判断 ip 访问次数是否超出限制

6. 文件系统。

在本项目中，用户主要会上传两类文件：用户个人头像以及需要解析的 json 文件。对于这两类文件都需要提供上传的接口并且合理地保存在系统文件夹中，还需要配置 nginx 使得其能够被访问到。在构建文件系统中，主要有以

下较为重要的问题。

a) 格式校验。用户上传时，前后端都会对文件的格式进行校验，确保上传文件后，前端能够正确地获取到文件并且展示出来。

b) 服务器静态资源获取。对于用户头像这类需要前端展示的文件类静态资源，需要对 nginx 进行路由配置，确保前端能够根据后端返回的路径正确获取到对应文件，项目中 nginx 对于静态文件配置如下：

```
1.    location /avatars {
2.        root /www/wwwroot/backend/public/uploads/;#指定图片存放路径
3.     }
```

4.2.3 算法方面

1. 合理管理 Python 项目

a) 版本问题

TODO

b) 文件结构问题

TODO

c) 路径问题

TODO

2. 用户体验问题的设计和检测

TODO

3. 提供网络接口

算法需要提供网络接口供后端进行调用，合理的接口设计关系到参数的传递以及结果相应的返回。下面将从以下几个方面阐述项目中算法接口的设计。

a) Web 框架。

由于算法部分的网络请求体量较小，不涉及较大数据库的使用和维护，因此我们选择了 flask 框架作为算法部分基本网络框架。Flask 是一个轻量级的框架，它的体积小巧，设计简洁，同时提供了灵活的扩展性，使得开发者可以根据自己的需求轻松添加或修改功能。flask 框架核心代码如下：

```
1.  from web import web
2.
3.  if __name__ == "__main__":
4.    web.app.run(port=8082, debug=True)
```

代码 4-1 flask 框架主函数

flask 框架路由构建如下：

```
1.  from flask import Flask, request, jsonify
2.  from web.json_parse import jsonParse
3.
4.  app = Flask(__name__)
5.
6.
7.  @app.route('/json/parse', methods=['POST'])
8.  def ParseJson():
9.  ...
```

```
10. 省略视图函数实现过程
11. ...
12.   return res
```

代码 4-2 flask 框架路由函数

b) 接口设计。

i. 由于算法模型直接和后端交互，并不会直接暴露在公网供前端调用，因此算法模型的接口可以限制 ip 地址为后端所以云服务器的地址，这样可以保证防止遭到恶意访问。

ii. 为了方便后端存储算法返回的结果，算法模型在返回响应时会将所有结果封装为一个结构体，将这个结构体直接返回。

4. 模型部署

问题：部署模型推理时，在 float16 精度下 13B 的模型参数大约占用 24GB 的内存空间，超过了系统的显存空间，模型无法完全部署在 GPU 设备上。如果采用 llama.cpp 技术，使模型参数在 GPU、内存和硬盘等多设备上部署，会使模型数据在内存和显存之间有频繁的 Copy 操作，占用大量的响应时间，使响应效率下降，并且严重占用了其他的系统资源。

解决：使用 GPTQ 技术对所使用的大模型 Llama2 进行量化转换。在将模型参数从 float16 转换到 int4 之后，模型的显存占用下降到 6GB 左右，在不明显降低模型表现力的情况下提高了模型的性能和响应效率，模型的响应速度提高了约五倍，系统资源的占用减少到原来的四分之一。

量化过程使用 Alpaca 数据集和 AutoGPTQ 工具，量化之后将模型用 safetensors 格式储存，方便部署模型时的加载操作。部分量化代码如下：

```
1.  quantize_config = BaseQuantizeConfig(
2.    bits=4,
3.    group_size=128,
4.    desc_act=False,
5.  )
6.
7.  model = AutoGPTQForCausalLM.from_pretrained(
8.    pretrained_model_dir, quantize_config)
9.  tokenizer = AutoTokenizer.from_pretrained(pretrained_mode
    l_dir)
10. examples = load_data(dataset_dir, tokenizer, n_samples)
11. examples_for_quant = [
12.   {"input_ids": example["input_ids"], "attention_mask": ex
    ample["attention_mask"]} for example in examples
13. ]
14.
15. model.quantize(examples_for_quant)
```

```
model.save_quantized(quantized_model_dir)
model.save_quantized(quantized_model_dir, use_safetensors=True)
```

代码 4-3 模型量化

4.3 本章小结

本章主要讲述了复杂问题归纳和存在问题与解决方案，提出项目难点以及问题，并对存在问题进行分析和解决。分别从前端、后端、算法三个方面简述开发中的问题。

5 项目实现

1. 登录功能

访问项目 url 时，未登陆用户需要先使用邮箱和密码进行登陆认证。登陆部分会有验证码做人机校验，并且对于同一个 ip，限制了其一定时间的访问次数。

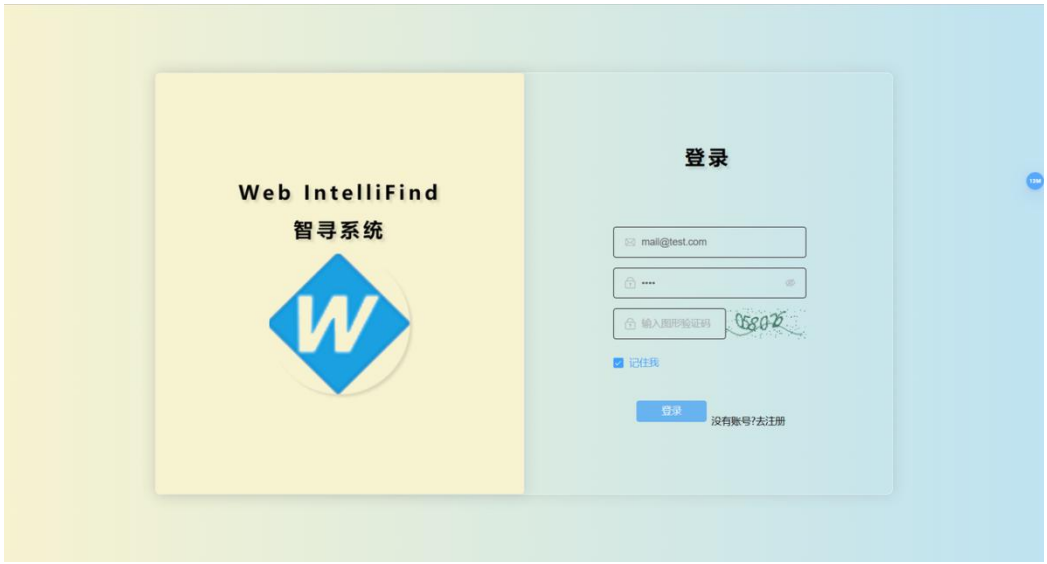

图 5-1 登录界面

2. 注册

倘若是首次使用系统，可以使用邮箱注册账号。为了方便用户管理，防止用户忘记登陆密码后无法找回，我们使用用户邮箱作为主账号，并在注册时要求进行邮箱验证码的校验。

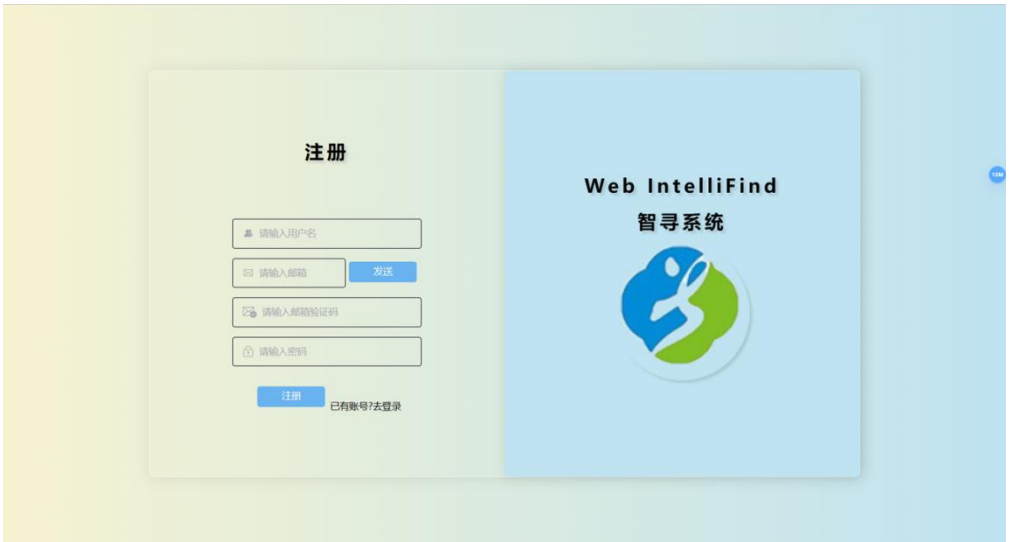

图 5-2 注册界面

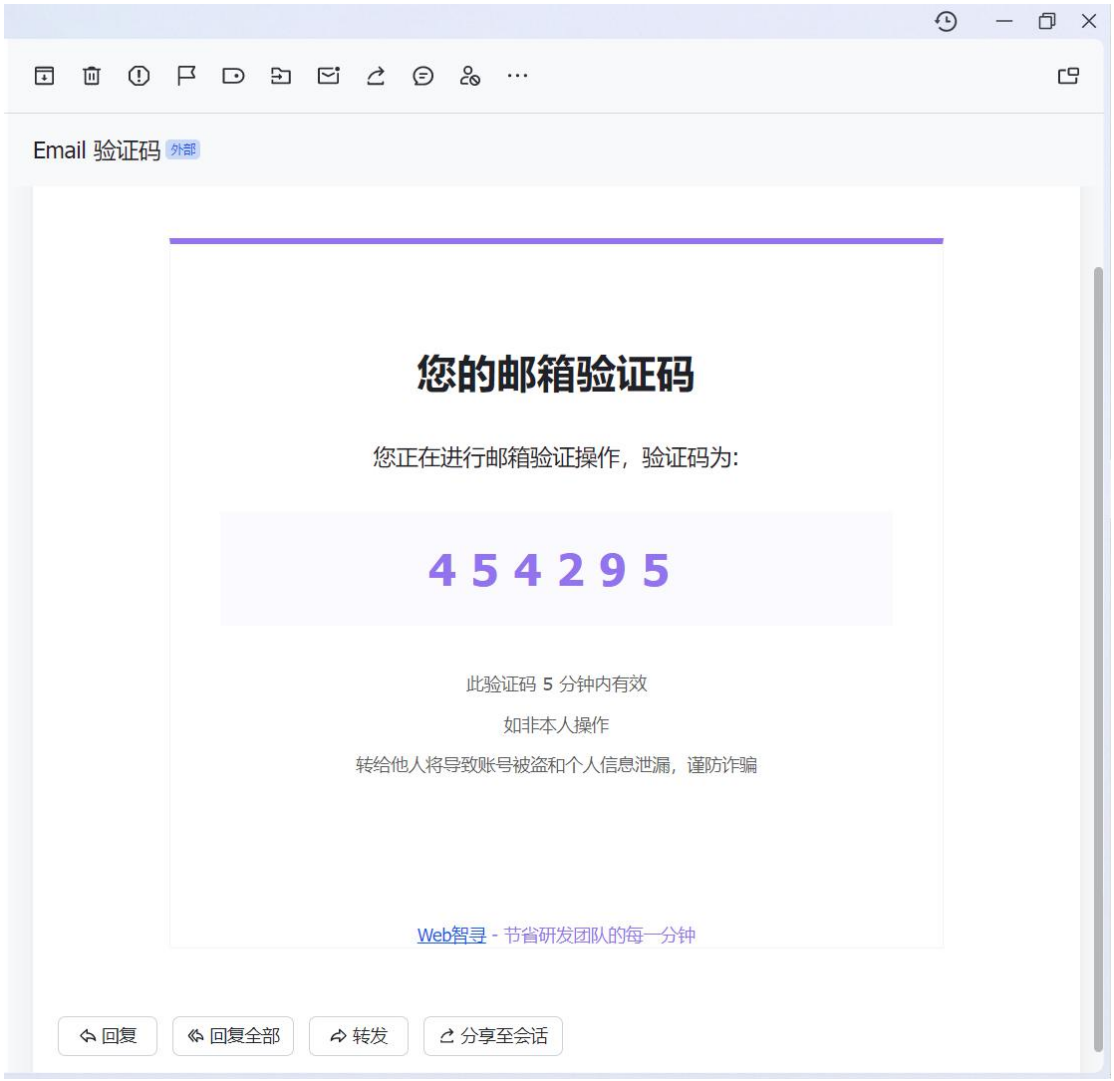
Email 验证码 外部
您的邮箱验证码
您正在进行邮箱验证操作，验证码为:
454295
此验证码 5 分钟内有效
如非本人操作
转给他人将导致账号被盗和个人信息泄漏，谨防诈骗
Web智寻 - 节省研发团队的每一分钟
回复
回复全部
转发
分享至会话

图 5-3 验证码邮箱

3. 更新用户信息

用户登陆后，会跳转到个人页面，这里可以查看用户个人信息，同时可以自由更新用户个人信息，如城市，电话等。

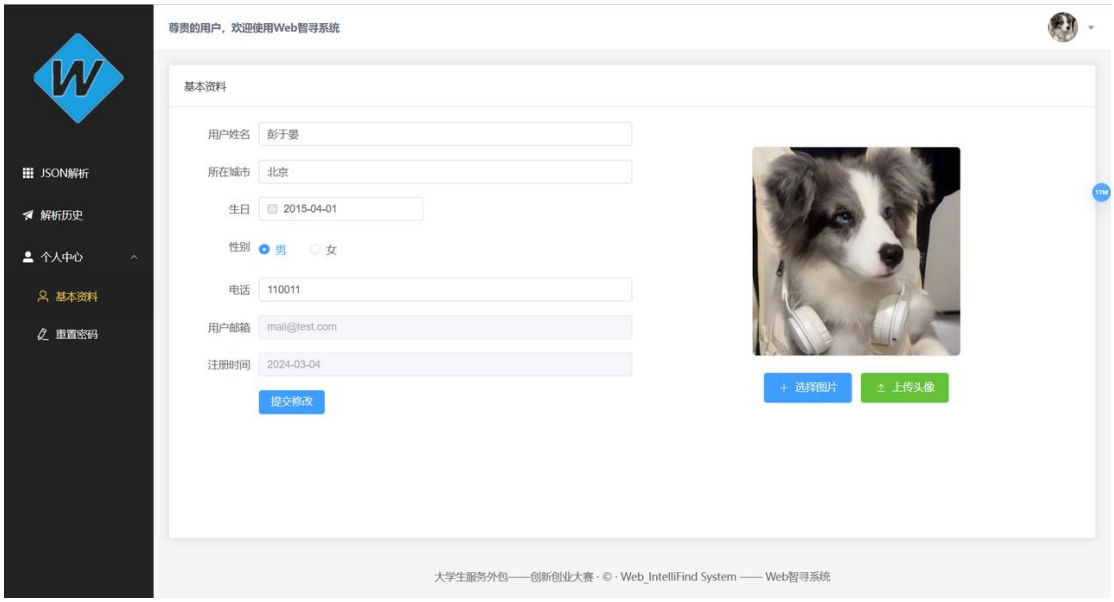

图 5-4 用户界面

4. 上传待解析文件

解析文件是项目的核心功能之一。在进行总分计算时，我们会根据各类问题的打分以及其对应的权重值，输出总分。由于不同网站对不同问题的容忍度存在一定差异，因此，用户在每一次上传文件之前，可以自定义自己想要的各项问题的权重值；如果用户这一次没有设置权重值，将采用用户上一次解析设定的权重值，如果用户是第一次使用，则我们会提供开发者自己定义的默认值。

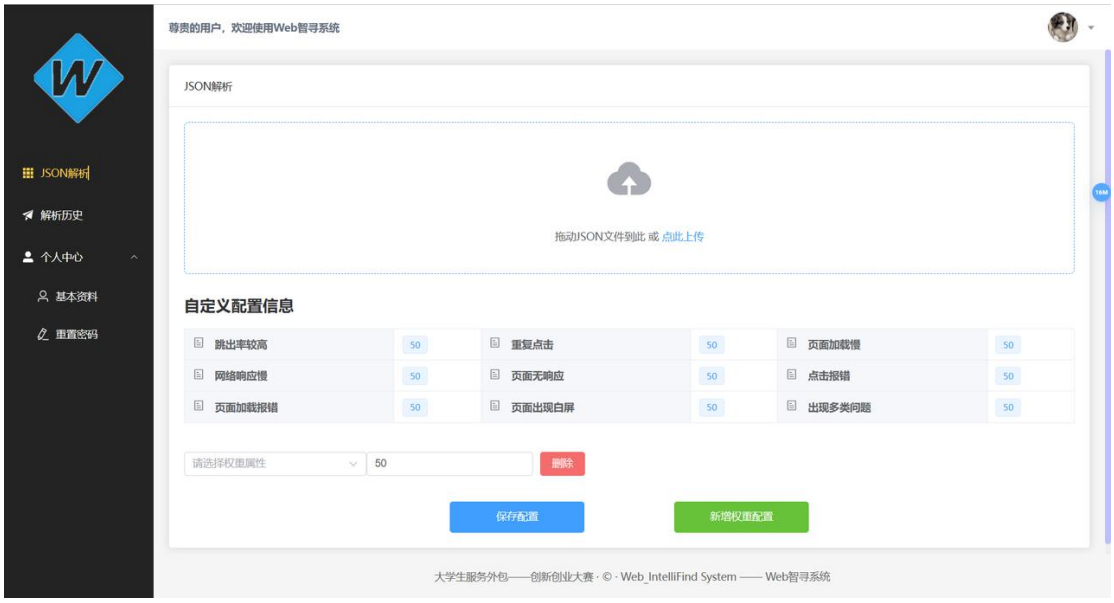

图 5-5 上传待解析文件界面

5. 查看解析详细报告

上传文件之后，后端和算法会对 json 文件进行解析，此时前端会实时轮询后端解析进度，在解析完成后告知用户并展示本次解析结果。解析结果如下图（图片经过处理），解析结果不仅包含了上传 json 文件中网站访问者的基本信息，也包含了这一次解析用户所设定的权重信息，并且将得分等数据可视化为了图表，方便用户直观感受。

图 5-6 解析结果界面

6. 查看解析历史记录

对于用户上传的每一次解析，我们都会将结果保存下来，可以查看某一次解析的详细情况。为了让用户更加直观地感受到近几次解析的变化，我们将近几次解析的得分绘制成了折线图。

图 5-7 解析历史

6 结语

Web 智寻 IntelliFind——基于用户行为数据的网站评分算法系统致力于利用用户行为数据来对网站进行评分，其目的在于帮助用户提高网站质量，调整其市场营销策略，针对用户行为进行更精准的营销活动，提高转化率；通过对网站进行评分和排序，可以鼓励网站运营者遵守良好的网络实践，减少误导性内容和不良信息的传播。此外。学者和市场研究人员可以使用这种基于用户行为的评分系统来分析网络趋势、用户偏好和行为模式，为相关研究提供数据支持。

此外，该项目也响应时代号召，深入贯彻习近平总书记关于科技创新的重要论述和对科技基础能力建设的重要指示，采用先进的技术开发产品，并提供提供给使用人员简单的操作方式，有助于科学教育和科普活动深入开展，创新发展理念深入人心，全民科学文化素质不断提升，尊重创新、鼓励创新、支持创新的社会氛围日益浓厚。

项目实现功能

- 用户管理
    - 用户登录
    - 用户注册
    - 重置密码
    - 更新用户个人信息
    - 注销登陆
- 文件解析
    - json 文件上传和保存
    - json 文件基本解析
    - 根据解析计算各项分值
    - ai 大模型分析 json 数据
    - 可视化解析结果
    - 解析结果存储
    - 历史记录删除

7 参考文献

[1] 支文瑜.Web 性能测试分析[J].信息技术与标准化,2018,No.399(03):41-43.

[2] Touvron H, Martin L, Stone K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint arXiv:2307.09288, 2023.

[3] Dao T. Flashattention-2: Faster attention with better parallelism and work partitioning[J]. arXiv preprint arXiv:2307.08691, 2023.

[4] Frantar E, Ashkboos S, Hoefler T, et al. Gptq: Accurate post-training quantization for generative pre-trained transformers[J]. arXiv preprint arXiv:2210.17323, 2022.