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

TODO

1.6 特色描述

TODO

1.7 本章小节

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：

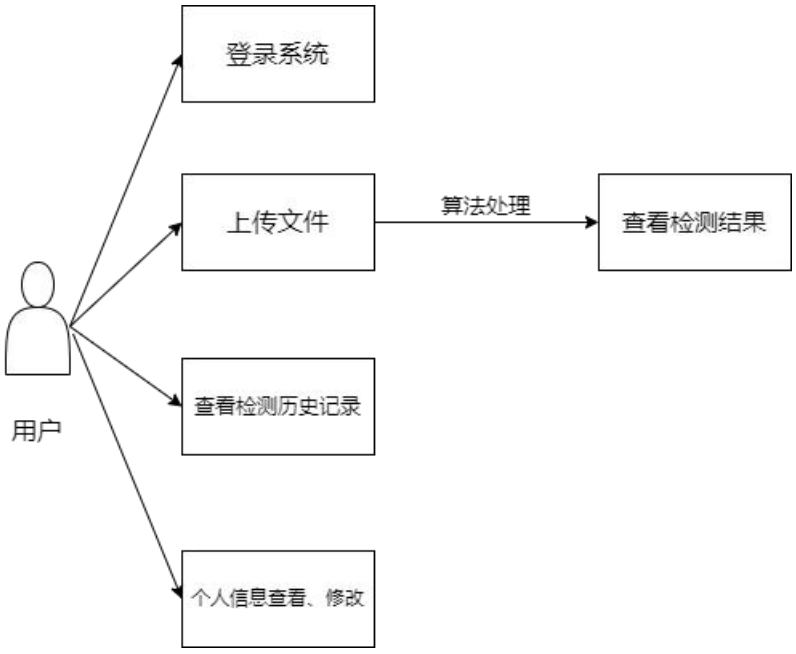

图 2-1

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 本章小节

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，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

TODO

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

TODO

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 TODO 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。

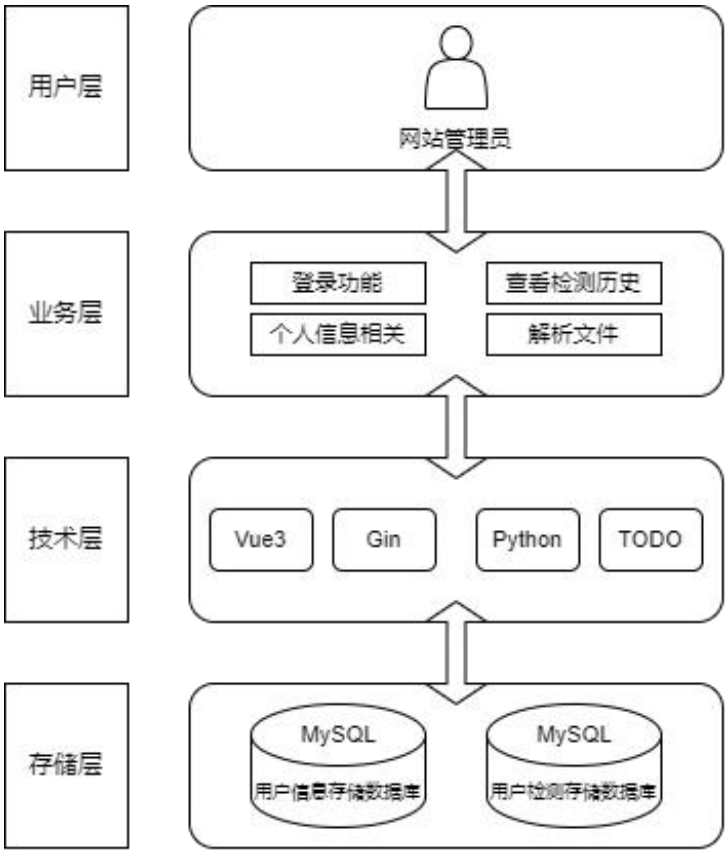

图 3-1

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。

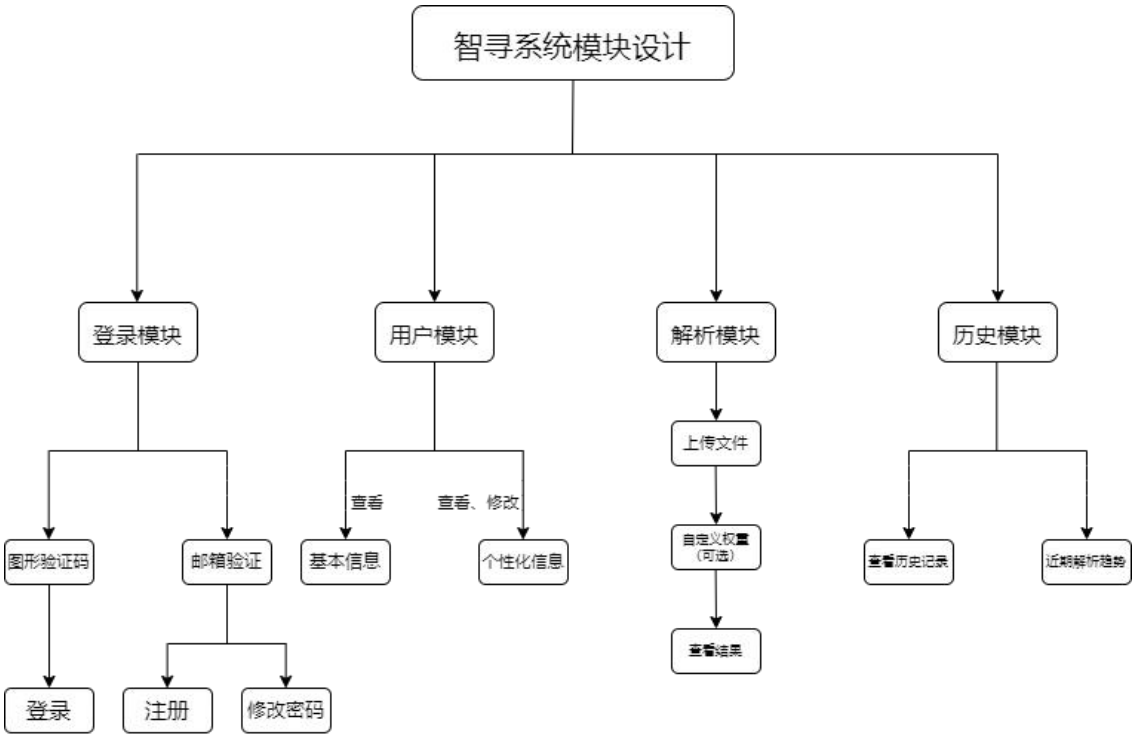

图 3-2

其中解析模块的具体流程如图 3-3 所示：

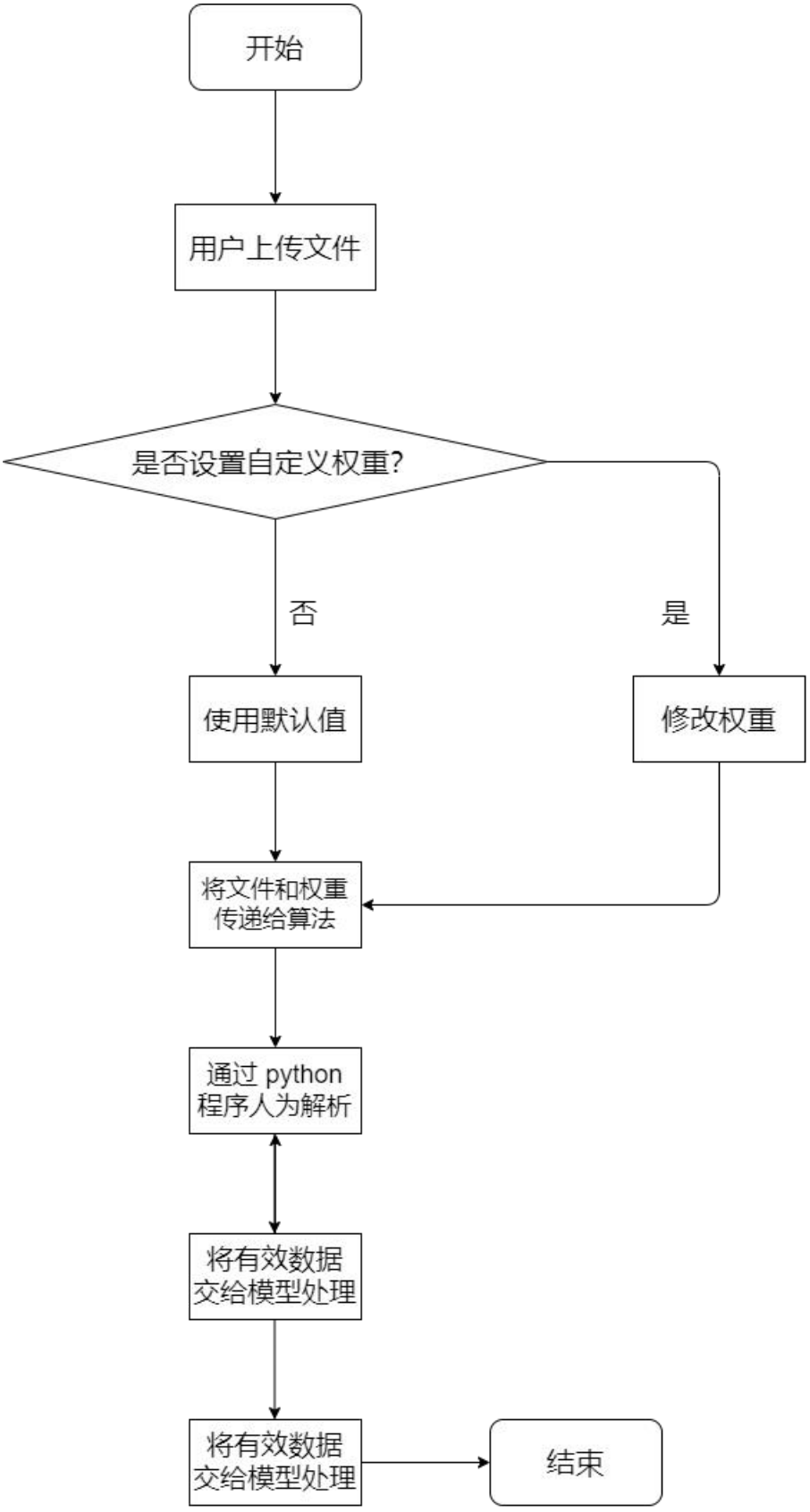

图 3-3

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；

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	
GoLand	2023.0.1
DataGrip	2023.3.4
Vue	3.0
Element-Plus	
flask	3.0.2
openAI	chatGPt3.5-Turbo
Gin	1.91
MySql	5.5.62
Redis	7.2.4
Nginx	1.23.4

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。

2. 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。

3. 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

TODO

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 本章小节

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。

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。

4.2 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()
```

```
const instance = axios.create({
  // 基础地址，超时时间
  baseURL,
  timeout: 10000
})

// 请求拦截器
instance.interceptors.request.use(
  async (config) => {

    // 判断本地持久化仓库是否有数据
    if (!userStore.value) {
      userStore.value = useUserStore()
    }

    // 判断仓库内是否有暂存的 Token
    if (userStore.value.token) {
      // 如果有 Token，则将其配置于请求头的 Authorization
      config.headers.Authorization = userStore.value.token
    }
    return config
  },

  // 如果请求有异常，则拦截此次请求
  (err) => Promise.reject(err)
)
```

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：

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。

```
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。

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

```
1.  <el-form-item>
2.    <div>
3.      <el-select> </el-select>
4.      <el-input> </el-input>
5.      <el-button type="danger" @click="removeItem(index)">删除</el-button>
6.    </div>
7.  </el-form-item>
8.
9.   <el-form-item>
10.    <el-button type="primary" @click="submitForm">保存配置</el-button>
11.    <el-button @click="addItem">新增权重配置</el-button>
12.  </el-form-item>
```

```
13.
14.
15. // 用户配置表单
16. const formRef = ref()
17.
18. // 默认展示两个权重值配置组件
19. const count = ref(2)
20. const formModel = ref({
21.  items: [{
22.   name: '',
23.   word: '',
24.   value: '50'
25.  }, {
26.   name: '',
27.   word: '',
28.   value: '50'
29.  }]
30. })
31.
32. const addItem = () => {
33.  if (count.value >= 9) {
34.   ElMessage.error('最多只能添加 9 个权重配置项')
35.   return
36.  }
37.  formModel.value.items.push({ word: '', name: '', value: '' }
    )
38.  selectedItems.value.push({ word: '' })
39.  count.value ++
40. }
41.
42. const removeItem = (index) => {
43.  formModel.value.items.splice(index, 1)
44.  count.value --
45. }
```

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

```
1.  <el-form-item
2.    v-for="(item, index) in formModel.items"
3.    :key="index"
4.    :prop="'items.' + index + '.word'"
5.
6.   >
7.    <div>
8.     <el-select v-model="selectedItems[index].word"
9.     @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 => ite
m.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.wor
d}))
   }
```

```
53.
54.   // 否则，过滤 ratioList 排除已经被选中的 word
55.   const excludedWords = [...selectedWords.slice(0, index), ...selectedWords.slice(index + 1)];
56.   return ratioList.value.map(option => ({ word: option.word })).filter(item => !excludedWords.includes(item.word));
57. };
58. });
```

4.2.2 后端方面

4.2.3 算法方面

1. 算法流程如何与后端进行协调？

TODO

4.3 本章小节

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

6 结语

7 参考文献