《设计开发大赛训练》报告

设计开发类

题　　名：千年舆图——中国历史地理电子地图平台

专　　业：地理信息科学

班　　级：地理信息科学 2021-1 班

姓名学号：马骁 07212393（组长）等级：

　　韦娜 07212415（组员）等级：

中国矿业大学环境与测绘学院

2024.2.

目录

一、开发背景与目的

1.1 开发背景

中华上下五千年，中国有着非常悠久的书面文字记录的历史，并且自古代一直有“左图右史”的传统。1987 年，《中国历史地图集》全部出版完成后，成为迄今为止国内外同类地图集质量最高、内容最详细、印制最精美的地图集之一。但传统纸质历史地图集存在着难以表达非标准年代的政区、使用者再加工的便利性不足、随着新出土资料涌现的地图考订错误而需要修订带来的复杂性的不足。同时历史中绝大部分信息都和空间、地理位置有关，信息化时代对历史地理学提出了新方向、新需求。

2016 年，复旦大学葛剑雄教授团队与哈佛大学合作完成“中国历史地理信息系统”（CHGIS）的研发工作，形成了时间序列数据、1820 年和 1911 年的专题数据和一套时空数据制作的技术标准；首都师范大学张萍教授团队研发出基于 GIS 数据的“丝绸之路历史地理信息开放平台”。GIS 具备对地理数据进行采集、存储、管理、运算、分析等强大功能，把 GIS 应用到历史地理研究引发了历史地理学研究范式的转型，引领了中国历史地理学研究方向近 20 年的发展（图 1-1）。因此，开发集成历史地理数据、多源历史数据和模型分析的中国历史地理电子地图平台（CHGIS），提供丰富的历史地理可视化和历史地理制图技术，可以有效整合管理历史地理数据、推动学科交叉融合和提高历史地理爱好者的学习积极性。

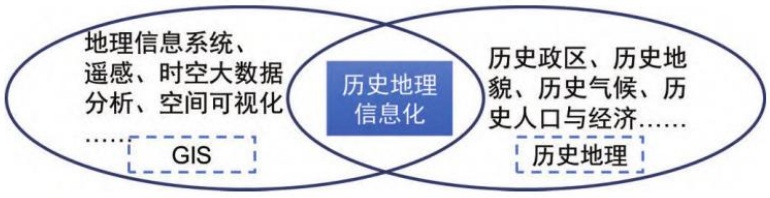

图 1-1 GIS 与历史地理学研究关系示意图

1.2 开发目的

“千年舆图——中国历史地理电子地图平台”以历代中国疆域为研究区域，开发基于知识图谱的历史地理知识问答模型，同时耦合时态分析、热点分析等分析模型，以 GIS 为集成环境的中国历史地理信息系统。其主要功能包括历史地理数据可视化、历史专题地图模型分析和 3D 可视化、历史地理知识图谱、历史地理知识问答模型等。在系统实现上，基于 Springboot3+Vue3+MyBatis 框架，运用 HTML5、CSS、JavaScript、Java 等语言进行整体平台的搭建；利用 Mapbox GL JS 实现地图的基本功能（信息窗口、打印、漫游等）、专题图制作和相关服务的发布、更新与调用，从而为历史地理学研究者和历史地理爱好者提供历史地

图可视化，历史地理数据专题制图、模型分析及 3D 可视化，历史地理知识图谱的查询、历史地理知识问答等，是进行有效历史地理数据管理和应用的技术支持。

二、开发流程

2.1 需求分析

2.1.1 功能需求分析

1、功能需求：

（1）多源数据管理：以数据库管理系统为依托，存储历史地理数据（如 等）和历史知识信息。

（2）历史地理数据可视化：能够提供历史地理数据的条件查询并通过表格、统计图、专题图等多种形式展示查询结果。

（3）历史地理数据模型分析：通过缓冲区分析、热点分析、方向分布分析和时态分析等分析模型，挖掘历史地理数据的信息。

（4）历史地理专题制图：实现基于地理数据可视化和模型分析结果的专题制图，为历史地理数据的丰富应用提供技术支持。

（5）历史地理知识图谱展示：实现基于知识图谱的历史地理知识的展示，方便用户了解历史地理知识。

（6）历史地理知识问答：调用 GPT 接口实现对历史地理知识的问答系统，高效解答用户提出的历史知识问题。

（7）其他辅助功能：提供历史知识、帮助文档及相关外部链接等人性化服务，更方便用户使用。

“千年舆图——中国历史地理电子地图平台”包括四个子系统：用户信息管理子系统、历史地图子系统、专题地图子系统、历史走廊子系统。每个子系统包含相应的功能与模块，如下图所示：

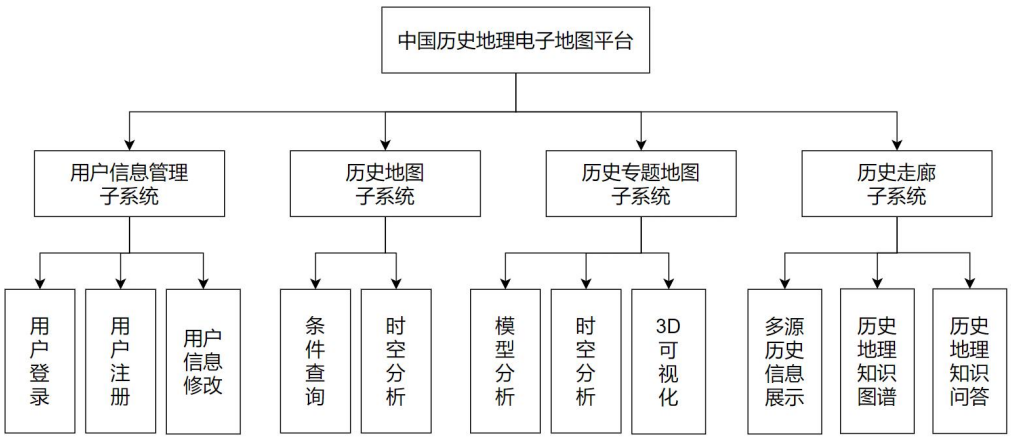

图 2-1 系统功能模块图

2、数据流图

（1）顶层数据流图

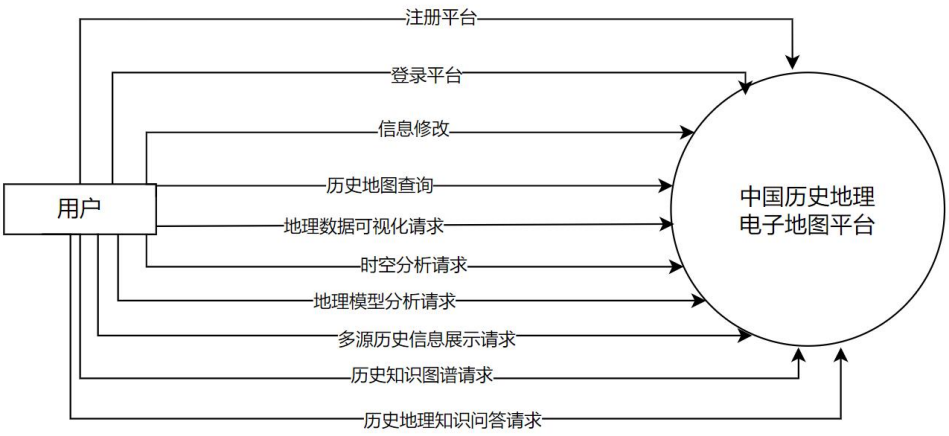

图 2-2 顶层数据流图

顶层数据流图说明：

用户对系统发出各类请求，系统对发出的请求进行接收与分析处理，并将分析结果呈现给用户。

（2）第一层数据流图

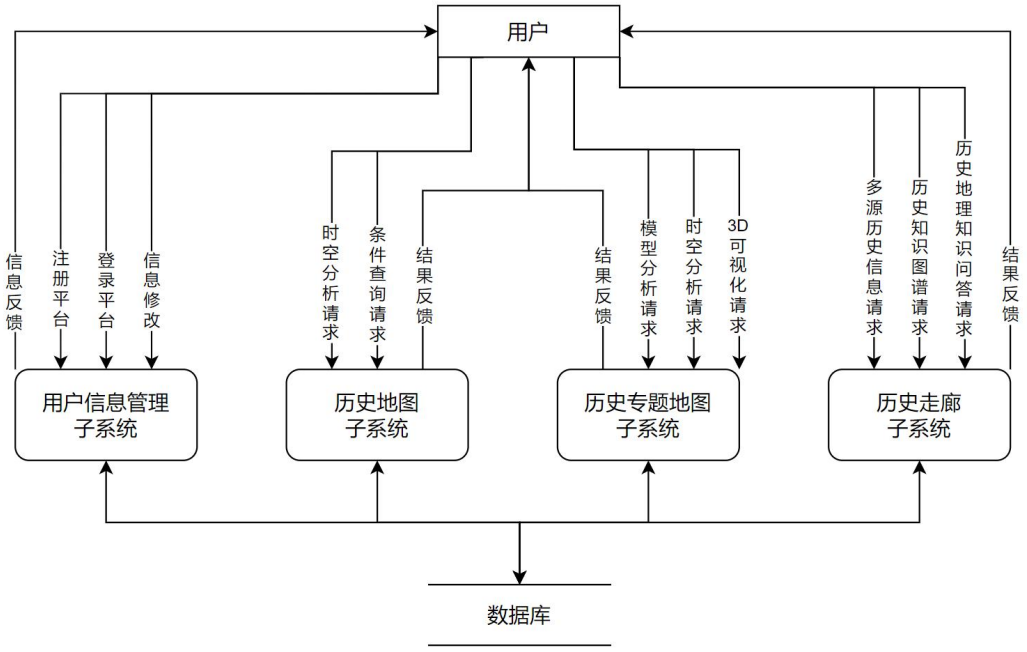

图 2-3 第一层数据流图

第一层数据流图说明：

将系统分为用户信息管理、历史地图、历史专题地图和历史走廊四个子系统，四个子系统都依赖于数据库，处于同一地位。

（3）第二层数据流图

由于用户信息管理子系统的功能比较复杂，其子系统的层次结构还可以进一

步深入分析，如下图所示。

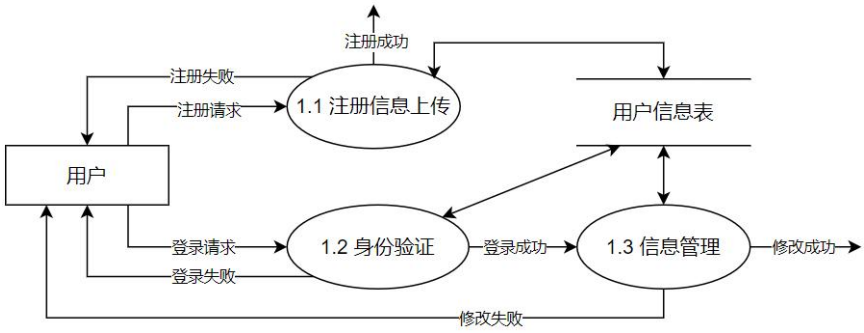

图 2-4 用户信息管理子系统二层图

3、用例图

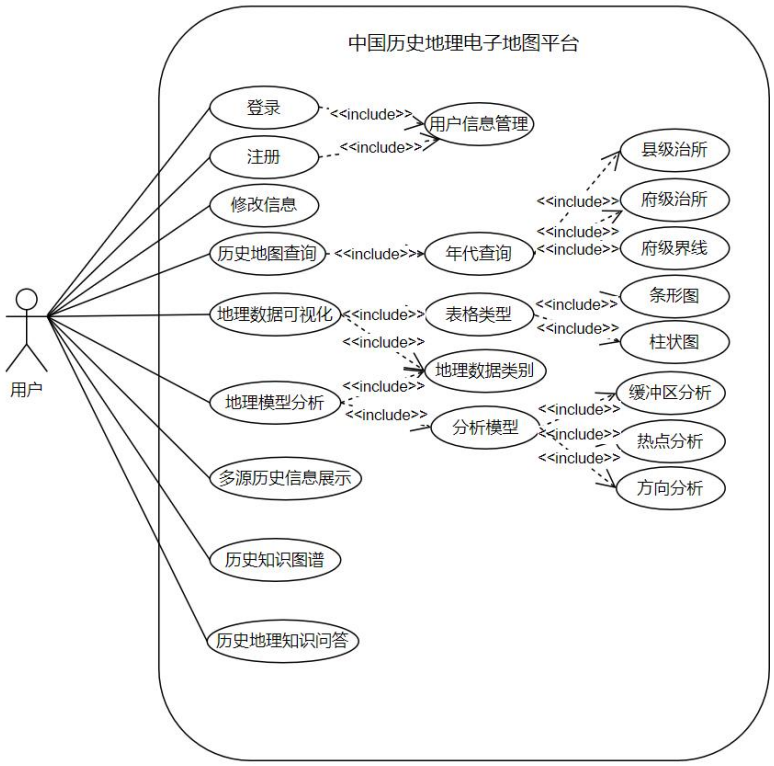

图 2-5 用例图

2.1.2 系统界面需求

本系统在界面方面的目标是人性化的设计和便捷的可操作性，确保首次使用该系统的用户能很快熟练掌握并操作。力求系统界面简洁美观；主要、重要功能的按钮突出显示；功能分类、分区组织；用户界面响应速度较快，同时占用较少的系统资源。

2.1.3 应用领域/实用性分析

“千年舆图——中国历史地理电子地图平台”的应用领域和实用性有以下几

点：

（1）历史地理数据资料的积累与管理：平台能够存储历史地理数据（如 等）和历史资料，方便数据的更新与共享，为历史地理相关科学研究人员及爱好者后续使用相关资料提供数据基础。

（2）历史地理数据查询与可视化：提供按条件查询历史地理数据和丰富的历史地理数据可视化方法，为历史地理相关科学研究人员及爱好者分析相关资料提供技术支持。

（3）历史地理专题制图：用户可以基于不同的地理数据可视化和模型分析结果生成并制作历史专题图，对历史地理分析提供数据基础。

（4）历史知识图谱与问答：方便用户清晰直观地了解历史地理知识；高效及时解决用户提出的历史知识问题。

2.2 总体设计

2.2.1 系统架构描述

为使整个系统具有较高的运行效率、较强的灵活性和可扩展性，系统采用多层架构模式，整个系统体系分为表现层、业务逻辑层、持久层和数据层。系统架构图如图 2-5 所示。

（1）表现层

表现层使用 Vue3 统一组织页面的结构，提高用户可操作性，同时便于系统改版、维护。使用 Mapbox GL JS、HTML5、CSS、JS、Cesium 等展示地图的基本功能，呈现历史地理数据、历史资料、知识图谱及问答系统等。

（2）业务逻辑层

主要使用 Springboot3 框架，进行包括用户管理模块、历史地图模块、历史专题地图模块、历史走廊模块的业务处理。

（3）持久层

使用 MyBatis 框架调用、存储数据，加速加载驱动、创建连接的过程，完成数据持久化。

（4）数据层

本系统的数据包括历史地理数据和用户信息数据。地理数据利用 File Geodatabase 进行存储，通过 Mapbox GL JS 发布为地图服务、几何服务等，历史地理数据和用户数据则是利用 PostgreSQL 和 Neo4j 数据库进行组织存储。

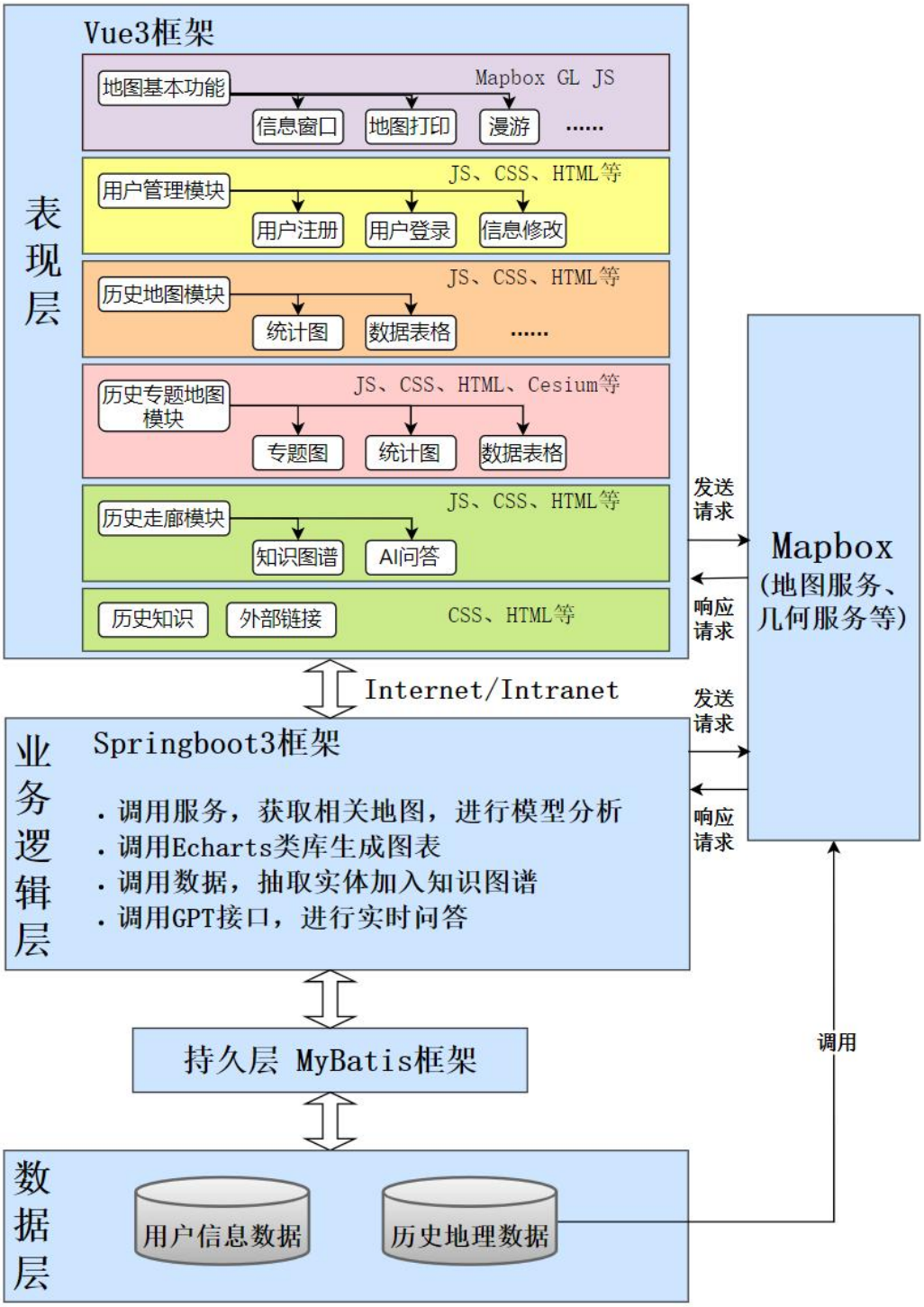

图 2-6 系统架构图

2.2.2 功能设计

1）地图基本功能

利用 Mapbox GL JS 实现图层的基本控制，如地图的缩放、漫游、地图打印输出、在地图上点击站点显示站点信息等。

2）用户管理模块

用户完成注册后，通过账号和密码登录，可进行用户信息的管理，包括：修改密码、更换头像等。

3）历史地图模块

该模块由条件查询模块、时空分析模块等子模块组成。

（1）条件查询模块

用户通过树形勾选框选择历史年代，地图显示相应的府级治所、府级界线和县级治所数据，点击站点弹出信息框显示站点信息；用户通过时间轴选择历史年代，页面跳转至地图，显示相应的时间序列府级治所、时间序列府级界线和时间序列县级治所数据，点击站点弹出信息框显示站点信息

（2）时空分析模块

用户点击时空分析按钮利用 echarts 图表库生成统计中国历史时期朝代的各要素数量条形图。

4）历史专题地图模块

该模块由模型分析模块、时空分析模块、3D 可视化模块等子模块组成。

（1）模型分析模块

用户点击缓冲区分析、热点分析、方向分布分析按钮，弹出相应业务窗口，用户选择业务数据后点击完成，地图响应显示相应分析结果。

（2）时空分析模块

用户点击时空分析按钮，在弹出窗口中选择地理数据和图表类型，利用 echarts 图表库生成相应图表。

（3）3D 可视化模块

用户点击 3D 可视化按钮，在弹出窗口中选择 3D 模型（如长城模型），点击完成后，地图中加载相应 3D 模型。

5）历史走廊模块

该模块由历史地理知识图谱模块、历史地理知识问答模块等子模块组成。

（1）历史地理知识图谱模块

用户点击知识图谱按钮，跳转至知识图谱页面，选择主题，点击生成按钮，页面显示相应主题知识图谱。

（2）历史地理知识问答模块

用户点击知识问答按钮，跳转至知识问答页面，在输入框中输入问题，点击发送，机器人回答用户的问题。

6）其他辅助功能

本平台提供历史知识和相关外部链接等辅助功能，更方便用户使用。

2.2.3 数据库设计

平台采用 PostgreSQL 数据库进行数据组织存储，数据库中表结构设计如下

表 2-1、表 2-2、表 2-3、表 2-4 所示，包括用户信息表、府级治所表、府级界线表、县级治所表等。

（1）用户信息表（user）：用于存放用户信息。

表 2-1　用户信息表

字段名称	数据类型	数据长度	说明
id	int	8	ID
username	varchar	20	用户名
password	varchar	25	密码
nickname	varchar	20	昵称
email	varchar	25	邮箱
user_pic	varchar	100	头像

（2）府级治所表（v6_time_pref_pts_gbk_wgs84）：用于存放府级治所信息。

表 2-2　府级治所表

字段名称	数据类型	数据长度	说明
id	serial	/	ID
geom	geometry(point,4326)	/	geometry 对象
name_py	varchar	40	县级治所
name_ch	varchar	30	简体中文名称
name_ft	varchar	30	繁体中文名称
x_coor	double precision	/	经度
y_coor	double precision	/	纬度
pres_loc	varchar	40	现所在地
type_py	varchar	15	建制类型拼音
type_ch	varchar	10	建制类型简体中文

lev_rank	varchar	1	建制等级
beg_yr	bigint	/	建制开始时间
beg_rule	varchar	1	开始时间精度
end_yr	bigint	/	建制结束时间
end_rule	varchar	1	结束时间精度
note_id	bigint	/	系统 id
obj_type	varchar	7	geometry 对象类型
sys_id	bigint	/	系统 id
geo_src	varchar	10	geometry 数据来源
compiler	varchar	12	编辑人员
gecomplr	varchar	10	绘制人员
checker	varchar	10	审核人员
end_date	varchar	10	结束时间
beg_chg_ty	varchar	20	建制开始原因
end_chg_ty	varchar	20	建制结束原因

（3）府级界线表（v6_time_pref_pgn_gbk_wgs84）：用于存放府级界线信息。

表 2-3　府级界线表

字段名称	数据类型	数据长度	说明
id	serial	/	ID
geom	geometry(multipolygon,4326)	/	geometry 对象
name_py	varchar	40	拼音名称
name_ch	varchar	30	简体中文名称
name_ft	varchar	30	繁体中文名称

x_coor	double precision	/	经度
y_coor	double precision	/	纬度
pres_loc	varchar	40	现所在地
type_py	varchar	15	建制类型拼音
type_ch	varchar	10	建制类型简体中文
lev_rank	varchar	1	建制等级
beg_yr	bigint	/	建制开始时间
beg_rule	varchar	1	开始时间精度
end_yr	bigint	/	建制结束时间
end_rule	varchar	1	结束时间精度
note_id	bigint	/	系统 id
obj_type	varchar	7	geometry 对象类型
sys_id	bigint	/	系统 id
geo_src	varchar	10	geometry 数据来源
compiler	varchar	12	编辑人员
gecomplr	varchar	10	绘制人员
checker	varchar	10	审核人员
end_date	varchar	10	结束时间
beg_chg_ty	varchar	20	建制开始原因
end_chg_ty	varchar	20	建制结束原因

（3）县级治所表（v6_time_cnty_pts_gbk_wgs84）：用于存放县级治所信息。

表 2-4　县级治所表

字段名称	数据类型	数据长度	说明
id	serial	/	ID
geom	geometry(point,4326)	/	geometry 对象
name_py	varchar	40	拼音名称
name_ch	varchar	30	简体中文名称
name_ft	varchar	30	繁体中文名称
x_coor	double precision	/	经度
y_coor	double precision	/	纬度
pres_loc	varchar	40	现所在地
type_py	varchar	15	建制类型拼音
type_ch	varchar	10	建制类型简体中文
lev_rank	varchar	1	建制等级
beg_yr	bigint	/	建制开始时间
beg_rule	varchar	1	开始时间精度
end_yr	bigint	/	建制结束时间
end_rule	varchar	1	结束时间精度
note_id	bigint	/	系统 id
obj_type	varchar	7	geometry 对象类型
sys_id	bigint	/	系统 id
geo_src	varchar	10	geometry 数据来源
compiler	varchar	12	编辑人员

gecomplr	varchar	10	绘制人员
checker	varchar	10	审核人员
end_date	varchar	10	结束时间
beg_chg_ty	varchar	20	建制开始原因
end_chg_ty	varchar	20	建制结束原因

2.3 系统实现

本系统前端使用 Vue3 技术，后端使用 Springboot3 技术，均是当前主流网页开发技术，利用 Springboot 设计接口将 Postgresql 数据库中的历史地理数据供前端 Axios 库调用。地图库方面使用了 Mapbox GL，与传统的 Openlayers，leaflet 地图库相比，Mapbox GL 基于 WebGL 开发，利用 WebGL 的高性能图形渲染能力，相比 Canvas 和 SVG，可以支持更加复杂和动态的三维地图效果。项目中还利用 Echarts 进行历史地理数据的图表分析。此外，项目中利用 GPT3.5 接口开发了历史聊天机器人，能够进行对话等功能。但目前由于历史地理数据有限和开发时间较短，实现复杂地理场景较少。

2.4 系统功能介绍

图 2-7 系统初始界面

首先进入初始界面，用户需要进行登录/注册，顶部导航栏可选择进入地图展示、历史讲解、历史 AI、我的用户界面。

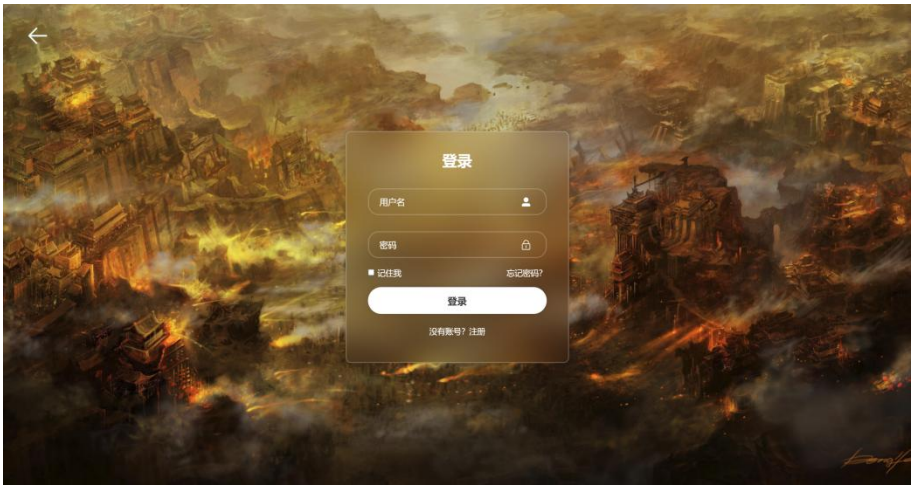

图 2-8 系统登录界面

进入系统登录界面，输入用户名密码进行登录，如果没有账号，选择注册账号。

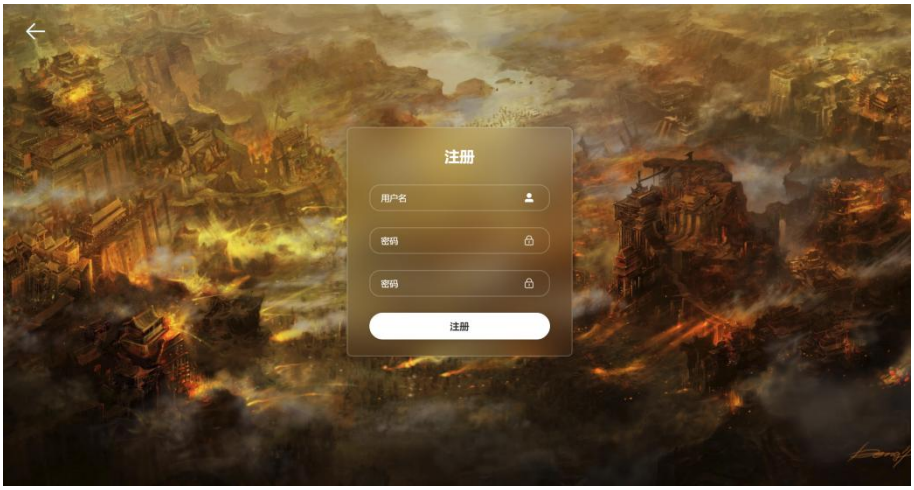

图 2-9 账号注册界面

没有账户先进行注册，输入用户名和密码，点击注册完成注册。

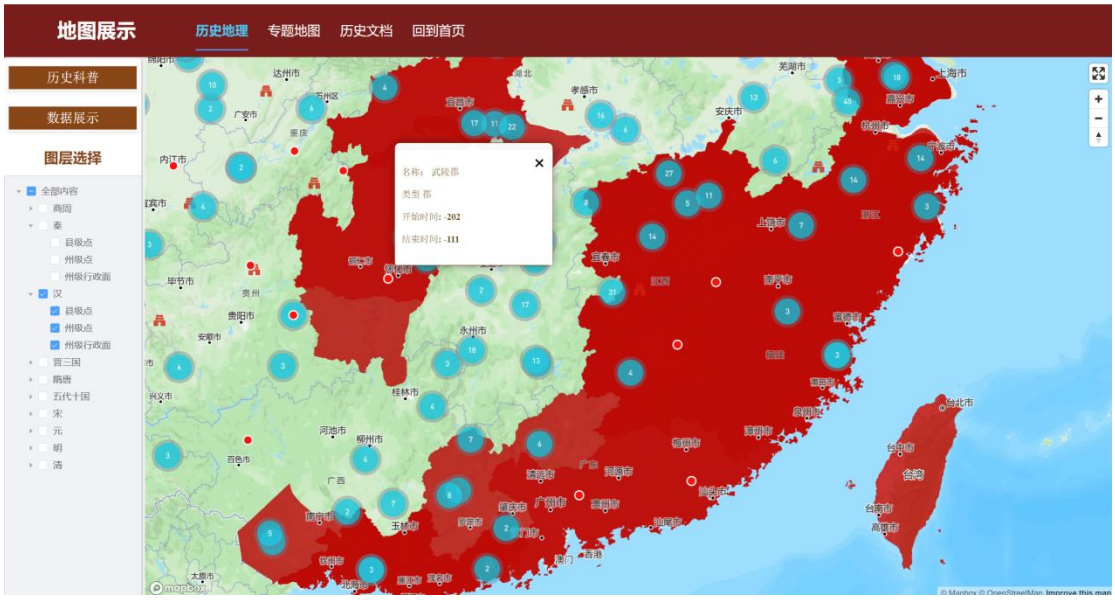

图 2-10 地图展示—历史地理界面

进入地图展示模块中的历史地理界面，系统提供了商周至清朝的县级点、州级点、州级行政面要素的历史数据加载，点击要素出现弹窗展示要素信息。

图 2-11 历史地理—历史科普界面

系统在左侧边栏提供了历史科普按钮，点击按钮，屏幕右侧出现信息框，介绍中国历史朝代的信息。

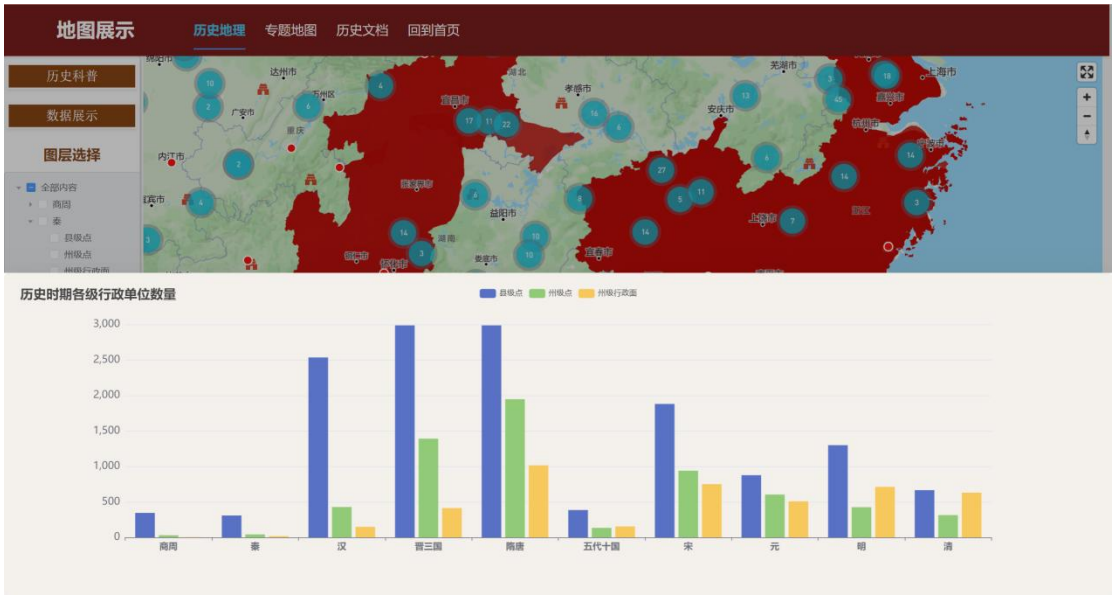

图 2-12 历史地理—数据展示界面

点击数据展示按钮，页面底部出现图表，统计中国历史时期朝代的各要素数量。

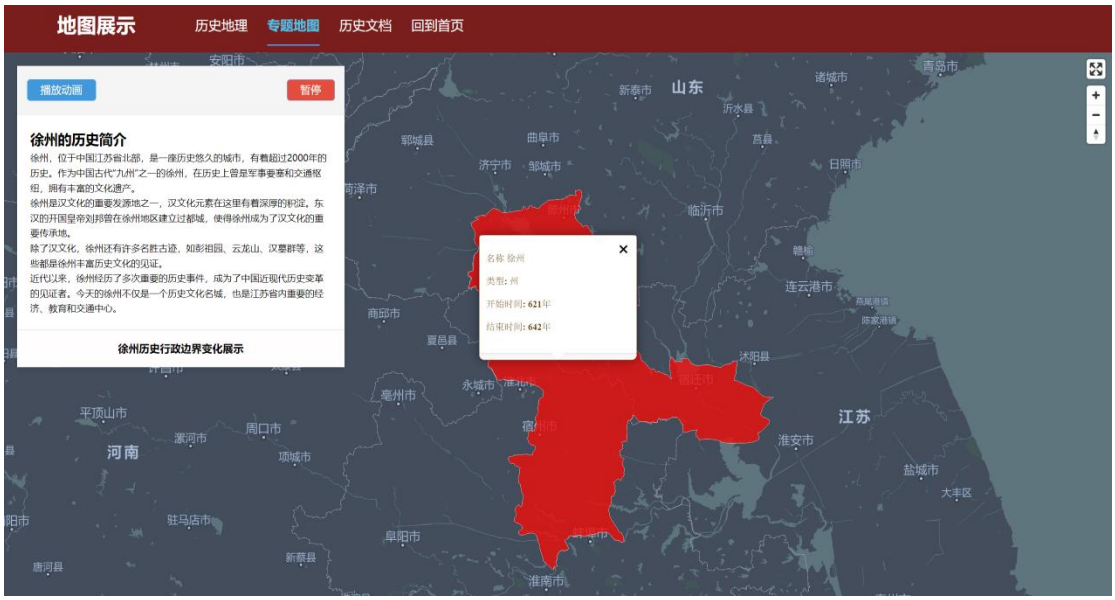

图 2-13 地图展示—专题地图界面

点击顶部导航栏专题地图按钮，进入专题地图界面，该专题地图介绍了徐州的历史信息，点击播放动画，可展示徐州历史行政边界变化，点击图层要素可显示相应信息。

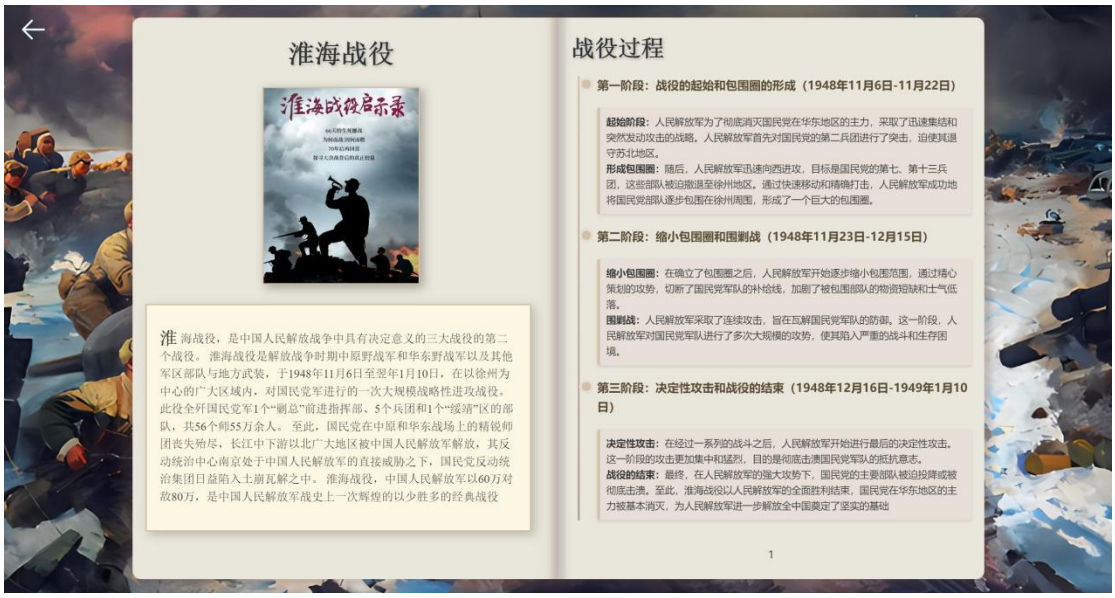

图 2-14 历史文档界面

历史文档界面主要讲解淮海战役相关信息和战役过程。

图 2-15 历史讲解界面

历史讲解界面以时间轴形式展示中国各个朝代，点击顶部按钮进入各个朝代的地图界面。

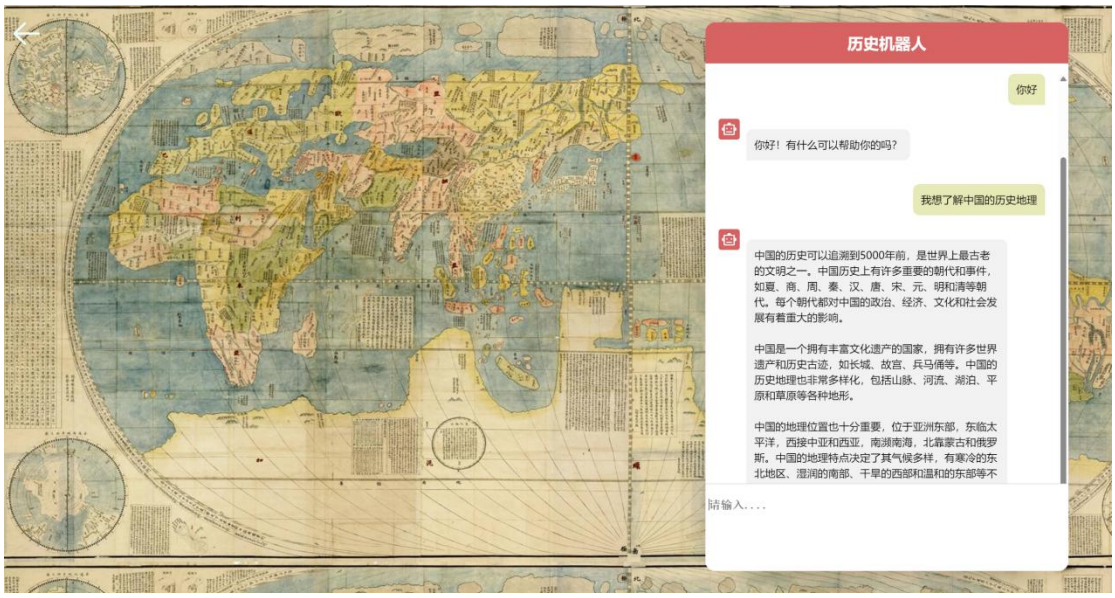

图 2-16 历史 AI 界面

历史 AI 界面提供了历史机器人，可以在线回答用户的问题。

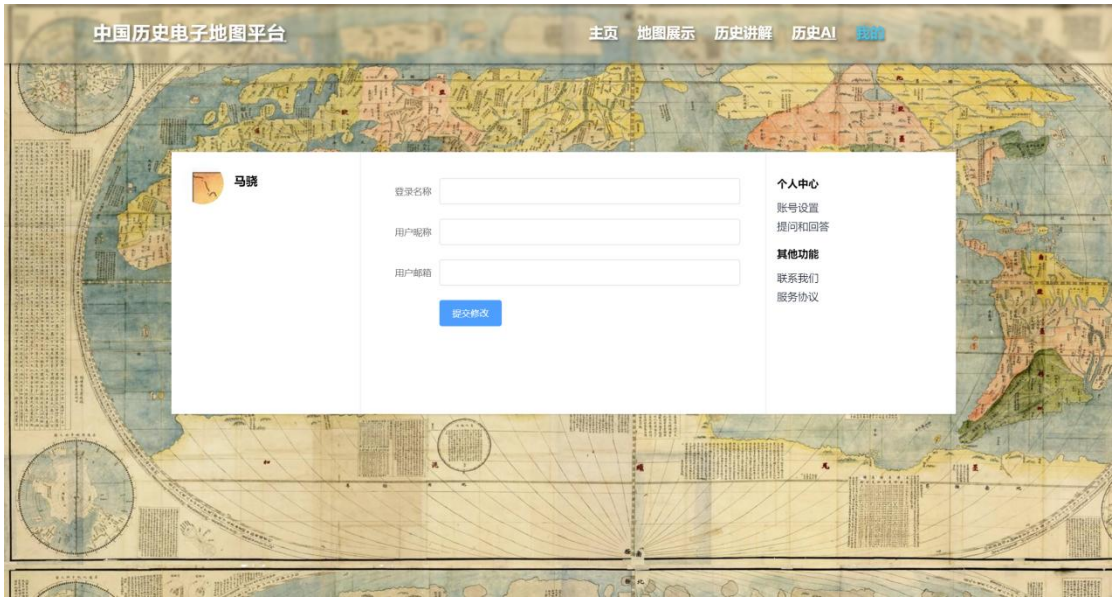

图 2-17 “我的”界面

“我的”界面提供用户相关信息的设置和修改功能。

2.5 系统未来开发方向

1. 历史数据的收集与整合

收集历史矢量数据：深入收集关于黄河流域的历史变迁、长城的历史沿线变化等矢量数据，包括不同朝代的边界变化、历史事件的地点等，为用户提供丰富的历史地理信息。

专题历史地图开发：开发以特定历史事件、文化遗产和自然地理为主题的地图，如丝绸之路的贸易路线、各朝代首都的变迁等，通过地图形式展现历史的发

展脉络。

2. 利用动画和 3D 技术增强表现力

动态地图讲解：利用动画技术讲解重大历史事件，如淮海战役的过程，通过动态变化的前线、进攻和防守方向等，使用户能直观感受战役的发展过程。

3D 地图特性：开发 3D 特性的历史地图，如三维重建的长城、古城遗址等，让用户可以在虚拟环境中探索，体验沉浸式的历史学习。

3. 知识图谱的整合，实现多源信息融合

建立历史知识图谱：整合历史人物、事件、地点等信息，构建知识图谱，通过关联分析，为用户提供丰富的背景知识和深入的历史解读。

多源数据融合：结合历史文献、考古发现、民间传说等多源数据，提供更全面的历史视角，增加学习的深度和广度。

4. 建立众包地理数据平台、地理数据社区

平台构建：开发一个众包地理数据平台，允许用户上传自己的地理历史数据。

社区参与：鼓励用户参与到历史地理信息的收集和验证过程中来，通过社区的力量，可以极大地丰富和扩展现有的历史地理数据集。

三、技术路线与关键技术

本系统基于 WebGIS，前端使用 Vue3，后端使用 Springboot3，使用 Mapbox 地图库加载地图，通过 Axios 获取接口数据，并引入 GPT 接口开发历史机器人。

3.1 系统使用流程图

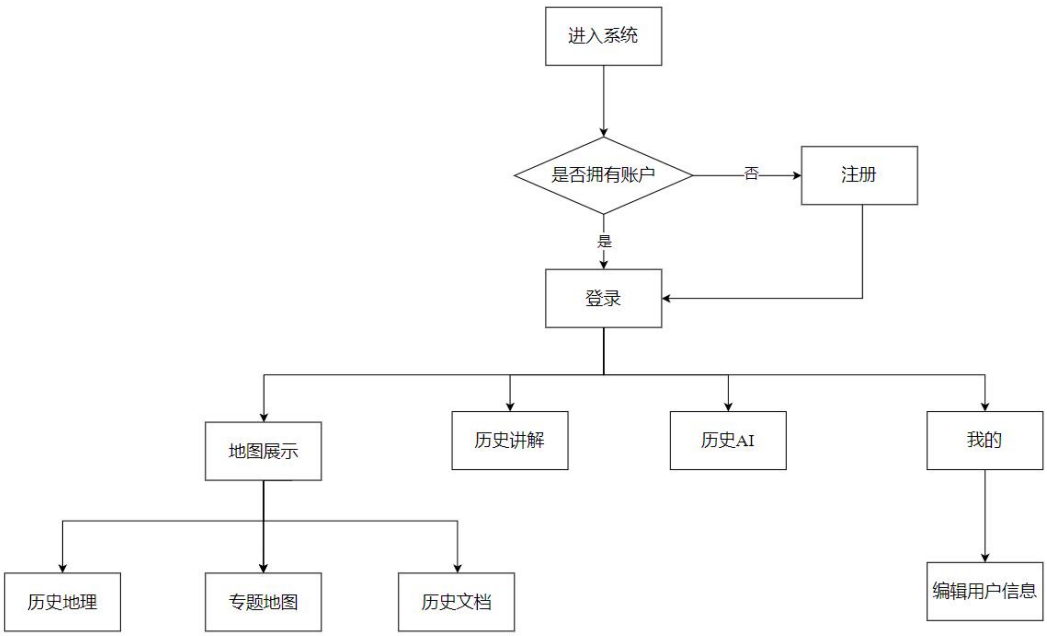

图 3-1 系统使用流程图

3.2 WebGIS 关键技术

WebGIS 是 Internet 技术与地理信息系统（GIS）结合的产物，通过互联网发布和应用地理空间数据，实现空间数据共享和互操作性。它允许用户通过 Web 浏览器（如 IE，Firefox）访问 GIS 信息，进行在线查询和业务处理。作为一种新技术，WebGIS 通过嵌入 HTTP 标准应用体系到 GIS 中，在 Internet 环境下支持空间信息管理和发布。

WebGIS 采用浏览器/服务器（B/S）架构，可以进行多主机、多数据库的分布式部署。这种架构下，服务器端向客户端提供信息和服务，而客户端能够获取和应用各种空间信息。

作为 Web 技术在 GIS 开发中的应用，WebGIS 使 GIS 变得更加普及，用户可以从任何地点通过 Internet 浏览空间数据、制作专题图、进行空间检索和分析。它是基于网络的客户机/服务器系统，利用 Internet 实现客户端与服务器之间的信息交换，是一个分布式系统，支持用户和服务器在不同地点和平台上工作。

WebGIS 的主要功能包括空间数据发布、查询与检索、空间模型服务和 Web 资源组织。它的基本特点包括使用 HTTP 协议、主要运算任务在服务器端执行（如地图绘制、空间数据库查询、空间分析）、用户端使用通用浏览器、以及 WWW 服务器作为地理信息服务的关键通道。在浏览器与服务器之间的 GIS 信息通讯中，WWW 使用的通用标记语言占据重要地位。

3.2.1 前端技术

Vue 3 是一个流行的前端 JavaScript 框架，用于构建用户界面和单页应用程序（SPA），于 2020 年发布，带来了许多新特性和改进，以提高性能、可用性和可维护性。

组合式 API：Vue 3 引入了组合式 API，与 Vue2 的选项式 API 相比，允许开发者更灵活地组织和重用代码。通过使用 setup 函数，开发者可以更好地控制响应式数据和生命周期钩子，从而使组件逻辑更加清晰和模块化。

性能提升：Vue 3 在性能方面做了大量优化，包括更小的包大小、更快的虚拟 DOM 重写和更新机制、以及更高效的组件初始化。这些改进使得 Vue 3 应用程序运行更快，同时减少了内存占用。

响应式和可观察性的改进：Vue 3 重写了其响应式系统，采用了 Proxy API 作为其响应式机制的基础。这使得 Vue 能够更直接和高效地跟踪数据变化，同时也使得 API 更简单、更易于使用。

本项目使用 Vite 构建工具+Vue3 创建前端应用，与 Webpack 相比有更快的程序启动，使用 Vue3 框架开发 WebGIS 应用是当下市场环境的主要选择。

3.2.2 后端技术

Spring Boot 3 是 Spring Boot 框架的最新版本，作为 Spring 框架的扩展，它继续简化 Spring 应用的创建和开发过程。Spring Boot 是一种 Java 基础的框架，用于创建独立的、生产级别的 Spring 应用。它致力于简化开发过程，其自动配置原理，使开发者能够快速启动和运行新的 Spring 项目，并且减少了项目的初始配置。

Spring Boot 提供了一个内置的 Apache Tomcat 服务器，使得开发者能够非常容易地开发和部署 web 应用。内置 Tomcat 的支持意味着开发者可以创建一个独立的 Spring 应用，该应用可以直接运行，无需部署到外部服务器。简化了开发和部署过程，因为它减少了与服务器配置和管理相关的复杂性。

目前主流 GIS 服务器如 ArcGIS Server，SuperMap iServer 均采用 Java 开发，掌握 Java 技术对 WebGIS 开发至关重要。

主要代码：

将历史地理数据按照数据类型，起始年份分开，基于请求参数 category，start，end 检索地理数据，创建一个 JSONObject 来构造响应体，这个 JSON 对象代表了一个 GeoJSON FeatureCollection。创建一个 GeometryModel 列表，为每个元素创建一个 GeoJSON Feature，包括 type、properties 和 geometry。

将每个 Feature 添加到 featuresArray 中，然后将这个数组添加到最初的 JSON 对象中，作为 features 属性的值。

最后，返回构造好的 JSON 对象，Spring Boot 会自动将这个 JSONObject 序列化为 JSON 字符串，并作为 HTTP 响应返回给客户端，返回数据满足 GeoJSon 格式，得以被地图库渲染在界面上。

```
1.      @ResponseBody
2.      @CrossOrigin
3.      @GetMapping("/geometry")
4.      public JSONObject getGeometry(@RequestParam("category") String category, @R
equestParam("start") Integer start, @RequestParam("end") Integer end) {
5.          List<GeometryModel> result = geometryService.getDynastyGeom(category, s
tart, end);
6.          JSONObject jsonObject = new JSONObject();
7.          jsonObject.put("type", "FeatureCollection");
8.          JSONArray featuresArray = new JSONArray();
9.          for (GeometryModel g : result) {
```

```
            JSONObject feature = new JSONObject();
            feature.put("type", "Feature");
            JSONObject properties = new JSONObject();
            properties.put("gid", g.getGId());
            properties.put("namech", g.getNameCh());
            properties.put("presloc", g.getPresLoc());
            properties.put("typech", g.getTypeCh());
            properties.put("begyr", g.getBegYr());
            properties.put("entdate", g.getEndDate());
            feature.put("properties", properties);
            JSONObject geometry = geometryToJson(g.getGeometry());
            feature.put("geometry", geometry);
            featuresArray.add(feature);
        }
        jsonObject.put("features", featuresArray);
        return jsonObject;
    }
```

由于前端接收的是 Json 格式数据，而后端存储的地理数据为 WKT 需要将其转换为geometry格式数据再转为Json，前一步已经在Mapper层转换，再Controller层中需要将 geometry 转为 Json，其中需要判断 geometry 的类型，对 MultiPolygon 和 Point 进行不同的处理。

```
private JSONObject geometryToJson(Geometry geometry) {
    Map<String, Object> jsonMap = new LinkedHashMap<>();
    jsonMap.put("type", geometry.getGeometryType());
    JSONArray coordinates = new JSONArray();

    if (geometry instanceof MultiPolygon) {
        MultiPolygon multiPolygon = (MultiPolygon) geometry;
        for (int i = 0; i < multiPolygon.getNumGeometries(); i++) {
            Polygon polygon = (Polygon) multiPolygon.getGeometryN(i);
            JSONArray polygonCoordinates = processPolygonCoordinates(polygon);

            coordinates.add(polygonCoordinates);
        }
    } else if (geometry instanceof Point) {
        Point point = (Point) geometry;
        Coordinate coordinate = point.getCoordinate();
        JSONArray coordinateArray = new JSONArray();
        coordinateArray.add(coordinate.x);
        coordinateArray.add(coordinate.y);
        coordinates.add(coordinateArray);
    } else {
```

```
            coordinates.addAll(processCoordinates(geometry.getCoordinates()));
        }

        if (coordinates.size() == 1) {
            jsonMap.put("coordinates", coordinates.get(0));
        } else {
            jsonMap.put("coordinates", coordinates);
        }
        return new JSONObject(jsonMap);
    }
```

第一个方法接收一个 Polygon 对象作为参数，其目的是处理多边形的外部环（外边界）和任何内部环（洞）的坐标，然后将它们以 JSON 数组的形式返回。

第二个方法接收一个坐标数组（Coordinate[]），用于处理单个环（无论是外部环还是内部环）的坐标，然后将它们转换为 JSON 格式。

```
    private JSONArray processPolygonCoordinates(Polygon polygon) {
        JSONArray polygonCoordinates = new JSONArray();
        polygonCoordinates.add(processCoordinates(polygon.getExteriorRing().getCoor
dinates()));

        for (int k = 0; k < polygon.getNumInteriorRing(); k++) {
            Coordinate[] interiorCoordinates = polygon.getInteriorRingN(k).getCoord
inates();
            polygonCoordinates.add(processCoordinates(interiorCoordinates));
        }

        return polygonCoordinates;
    }

    private JSONArray processCoordinates(Coordinate[] coordinates) {
        JSONArray coordinateList = new JSONArray();
        for (Coordinate coordinate : coordinates) {
            JSONArray coordinateArray = new JSONArray();
            coordinateArray.add(coordinate.x);
            coordinateArray.add(coordinate.y);
            coordinateList.add(coordinateArray);
        }
        return coordinateList;
    }
```

下面是用户注册、登录、获取信息、更新的相关代码，其中使用了 JWT 令牌进行登录校验，MD5 密码加密，redis 存储 token 令牌等。

```
1.  @PostMapping("/register")
2.      public Result register(@Pattern(regexp = "^\\S{5,16}$") String username, @P
attern(regexp = "^\\S{6,20}$") String password) {
3.
4.          User u = userService.findByUserName(username);
5.          if (u == null) {
6.              //没有占用
7.              //注册用户
8.              userService.register(username, password);
9.              return Result.success();
10.         } else {
11.             return Result.error("用户名已经被占用");
12.         }
13.     }
14.
15.     @PostMapping("/login")
16.     public Result<String> login(@Pattern(regexp = "^\\S{5,16}$") String usernam
e, @Pattern(regexp = "^\\S{6,20}$") String password) {
17.         User loginUser = userService.findByUserName(username);
18.
19.         if (loginUser == null) {
20.             return Result.error("用户名不存在");
21.         }
22.
23.         //比较密码，生成 token
24.         if (Md5Util.getMD5String(password).equals(loginUser.getPassword())) {
25.             Map<String, Object> claims = new HashMap<>();
26.             claims.put("id", loginUser.getId());
27.             claims.put("username", loginUser.getUsername());
28.             String token = JwtUtil.genToken(claims);
29.
30.             //把 token 存储到 redis 中
31.             ValueOperations<String,String> operations = stringRedisTemplate.ops
ForValue();
32.             operations.set(token,token,2, TimeUnit.HOURS);
33.             return Result.success(token);
34.         }
35.         return Result.error("密码错误");
36.     }
37.
38.
39.     //获取用户信息
```

```
40.     @GetMapping("/userinfo")
41.     public Result<User> userInfo() {
42. //        Map<String, Object> claims = JwtUtil.parseToken(token);
43. //        String username = (String) claims.get("username");
44.         Map<String, Object> map = ThreadLocalUtil.get();
45.         String username = (String) map.get("username");
46.         User user = userService.findByUserName(username);
47.         return Result.success(user);
48.     }
49.
50.     //用户更新
51.     @PutMapping("/update")
52.     public Result update(@RequestBody @Validated User user) {
53.         userService.update(user);
54.         return Result.success();
55.     }
56.
57.     //更新用户头像
58.     @PatchMapping("/updateAvatar")
59.     public Result updateAvatar(@RequestParam @URL String avatarUrl) {
60.         userService.updateAvatar(avatarUrl);
61.         return Result.success();
62.     }
```

与编写地理数据处理接口相比，用户接口编写相对容易，利用 Java 处理地理数据需要对地理数据格式十分熟悉。Java 提供了多种库和工具来效地处理 GIS 数据，如 Geotools 和 JTS。在 Java 中处理地理信息系统（GIS）数据是一个涉及读取、处理、分析和展示地理空间数据的复杂过程，需要不断学习。

3.3 Mapbox 关键技术

Mapbox 是一个提供地图和位置服务的平台，它允许开发者在网页和移动应用中集成高度可定制的地图和位置功能。Mapbox 使用了向量地图技术，这意味着地图可以在客户端动态渲染，提供了平滑的缩放和导航体验，同时支持强大的样式和可视化定制。Mapbox 提供地理编码（将地址转换为坐标）和逆地理编码（将坐标转换为地址）服务，支持全球范围内的位置搜索和地址解析。

数据可视化和分析：Mapbox 支持丰富的数据可视化功能，允许开发者在地图上展示复杂的数据集和分析结果。

与传统的 Openlayers 和 Leaflet 地图库，Mapbox 有更多的地图样式，而且开发者可以通过 Mapbox Stduio 自定义地图样式并发布。此外，Mapbox 基于 WebGL 开发，具有强大的地理数据渲染能力，能够带来更好的视觉效果。

本项目采用 Mapbox 地图库进行地理数据的加载和渲染，部分代码如下：

初始化 Mapbox 底图，并选用中文语言包，删除道路等图层，设置初始化位置和缩放等级。

```
import mapboxgl from 'mapbox-gl';
import MapboxLanguage from '@mapbox/mapbox-gl-language'
export const Mapbox = () => {
    mapboxgl.accessToken = 'pk.eyJ1IjoibXh4eG9yejEyIiwiYSI6ImNsa2MyMjBvaTBhaHEzcnM0YmQ1d2Y2Y3EifQ.1Ez5HwcosSSBihZN23s9ZA';
    const map = new mapboxgl.Map({
        container: 'map', // container ID
        style: 'mapbox://styles/mapbox/streets-v12', // style URL
        center: [116.397128, 39.916527],
        zoom: 5 // starting zoom
    });
    map.addControl(new MapboxLanguage({
        defaultLanguage: 'zh-Hans'
    }));
    map.addControl(new mapboxgl.FullscreenControl({ container: document.querySelector('body') }));
    map.addControl(new mapboxgl.NavigationControl());
    map.on('load', function () {
        // 获取所有图层的列表
        const layers = map.getStyle().layers;
        // 遍历图层列表
        layers.forEach(function (layer) {
            // 根据图层 id 或其他属性判断是否为路网图层
            if (layer.id.includes('road') || layer.id.includes('highway')) {
                // 隐藏路网图层
                map.setLayoutProperty(layer.id, 'visibility', 'none');
            }
        });
    });
    return map;
};
```

这段代码加载了县级类型的历史点到底图上，首先要指定数据源，数据源来自于 Axios 通过接口从后端获取，为避免大量点渲染到地图上，造成卡顿和可视化效果差，我们使用了 Mapbox 提供的聚合点加载，通过设置聚合半径和数量，当缩放地图时，动态显示地图上的点，通过 map.addLayer 加载聚合点，未聚合

点，并通过 paint 设置点样式。设置点击事件，点击聚合点时调整地图缩放等级，点击未聚合点时，展示弹窗信息。加载州级点同理。

```
export const clusterCntyPoints = (map, url) => {
    function fetchData(url) {
        return axios.get(url).then((res) => {
            return res.data;
        });
    }
    fetchData(url).then((point) => {
        console.log(point);
        var cntyptslist = point['features'];
        map.addSource("cntyPoints", {
            type: "geojson",
            data: point,
            cluster: true,
            clusterMaxZoom: 14, //最大缩放到群集点
            clusterRadius: 50 // 每一组点的半径（=50）
        });
        map.addLayer({
            id: "clusters",
            type: "circle",
            source: " cntyPoints ",
            filter: ["has", "point_count"],
            paint: {
                // 修改颜色方案
                "circle-color": [
                    "step",
                    ["get", "point_count"],
                    "rgba(124, 210, 144, 0.8)", // 淡绿色，点数小于 100
                    100,
                    "rgba(34, 139, 34, 0.8)", // 深绿色，点数在 100 到 750 之间
                    750,
                    "rgba(160, 82, 45, 0.8)" // 棕色，点数大于或等于 750
                ],
                "circle-radius": [
                    "step",
                    ["get", "point_count"],
                    20, // 当点数小于 100 时为 20px 圆
                    100, // 100 以内
                    21, // 点计数在 100 到 750 之间时为 21px 圆
                    750, // 750 以内
                    22 // 点计数大于或等于 750 时为 22px 圆
                ],
                // 修改外边框颜色以匹配新的颜色方案
```

```
                "circle-stroke-color": [
                    "step",
                    ["get", "point_count"],
                    "rgba(144, 238, 144, 0.4)", // 淡绿色边框
                    100,
                    "rgba(34, 139, 34, 0.4)", // 深绿色边框
                    750,
                    "rgba(160, 82, 45, 0.2)" // 棕色边框
                ],
                // 调整外边框晕染的范围以匹配新的颜色方案
                "circle-stroke-width": [
                    "step",
                    ["get", "point_count"],
                    5, // 当点数小于 100 时为 5px 晕染
                    100, // 100 以内
                    6, // 点计数在 100 到 750 之间时为 6px 晕染
                    750, // 750 以内
                    7 // 点计数大于或等于 750 时为 7px 晕染
                ]
            }
        });

        //聚合图圆圈中的数字
        map.addLayer({
            id: "cluster-count",
            type: "symbol",
            source: "cntyPoints",
            filter: ["has", "point_count"],
            layout: {
                "text-field": "{point_count_abbreviated}",
                "text-font": ["DIN Offc Pro Medium", "Arial Unicode MS Bold"],
                "text-size": 12
            },
            // 添加这个就可以改变圆圈内字样式，这里我改变了他的颜色
            paint: {
                "text-color": "#fff",
                "text-opacity": 1
            }
        });
        // 聚合图中没有数字的显示小圆点
        map.addLayer({
            id: "unclustered-point",
            type: "circle",
```

```
            source: " cntyPoints ",
            filter: ["!", ["has", "point_count"]],
            paint: {
                "circle-color": "#ff0000",
                "circle-radius": 6,
                "circle-stroke-width": 2,
                "circle-stroke-color": "#fff"
            }
        });

        // 单击时检查群集
        map.on("click", "clusters", function (e) {
            var features = map.queryRenderedFeatures(e.point, {
                layers: ["clusters"]
            });
            var clusterId = features[0].properties.cluster_id;
            map
                .getSource("cntyPoints ")
                .getClusterExpansionZoom(clusterId, function (err, zoom) {
                    if (err) return;
                    map.easeTo({
                        center: features[0].geometry.coordinates,
                        zoom: zoom
                    });
                });
        });
        // 单击时检查未聚类的点
        map.on('click', 'unclustered-point', (e) => {
            const clickedFeature = e.features[0];
            const coordinates = clickedFeature.geometry.coordinates.slice();
            const namech = clickedFeature.properties.namech;
            const typech = clickedFeature.properties.typech;
            const presloc = clickedFeature.properties.presloc;
            const begyr = clickedFeature.properties.begyr;
            const endyr = clickedFeature.properties.endyr;
            while (Math.abs(e.lngLat.lng - coordinates[0]) > 180) {
                coordinates[0] += e.lngLat.lng > coordinates[0] ? 360 : -360;
            }
            new mapboxgl.Popup()
                .setLngLat(coordinates)
                .setHTML(
                    `<p>名称： ${namech}</p>
                    <p>类型： ${typech}</p>
                    <p>位置: ${presloc}</p>
```

```
                    <p>开始时间: ${begyr}</p>
                    <p>结束时间: ${endyr}</p>`
                )
                .addTo(map);
        });
        map.on("mouseenter", "clusters", function () {
            map.getCanvas().style.cursor = "pointer";
        });
        map.on("mouseleave", "clusters", function () {
            map.getCanvas().style.cursor = "";
        });
    })
}
```

加载州级行政区，与加载点同理，通过接口获得地理数据，点击图层展示面要素信息。

```
export const prefpgn = (map, url) => {
    function fetchData(url) {
        return axios.get(url).then((res) => {
            return res.data;
        });
    }
    fetchData(url).then((pref) => {
        console.log(pref);
        map.addSource("prefpgn", {
            type: "geojson",
            data: pref,
        });
        map.addLayer({
            id: "prefpgn",
            type: "fill",
            source: "prefpgn",
            paint: {
                "fill-color": '#C00000',
                "fill-opacity": 0.8
            }
        });
        map.on('click', 'prefpgn', (e) => {
            const clickedFeature = e.features[0];
            let coordinates;
            if (clickedFeature.geometry.coordinates[0][0][0].length === 2) {
```

```
                coordinates = clickedFeature.geometry.coordinates[0][0][0].slic
e();
            }
            else {
                coordinates = clickedFeature.geometry.coordinates[0][0].slice();

            }
            const namech = clickedFeature.properties.namech;
            const typech = clickedFeature.properties.typech;
            const presloc = clickedFeature.properties.presloc;
            const begyr = clickedFeature.properties.begyr;
            const endyr = clickedFeature.properties.endyr;
            new mapboxgl.Popup()
                .setLngLat(coordinates)
                .setHTML(
                    `<p>名称： ${namech}</p>
                    <p>类型 ${typech}</p>
                    <p>开始时间: ${begyr}</p>
                    <p>结束时间: ${endyr}</p>`
                )
                .addTo(map);
        });
    })
}
```

专题地图中展示了徐州历史行政边界的变化，首先获取数据，数据中含有多个面，我们通过设置定时器实现面要素的动态加载，同样点击要素出现弹窗展示信息。

```
const playAnimation = () => {
    // 确保在动画已暂停且定时器存在时恢复动画
    if (isPaused && interval) {
        isPaused = false;
        return;
    }
    // 如果定时器不存在，则开始或重新开始动画
    if (!interval) {
        index = 0;
        previousSourceId = null;
        previousLayerId = null;
        map.value.flyTo({ center: [117.1113, 34.1554], zoom: zoomLevel });
        interval = setInterval(function () {
            if (isPaused) { // 如果动画暂停，则跳过此次迭代
```

```
                return;
            }
            if (index < xuzhou.features.length) {
                // 在添加新图层之前删除上一个图层和源
                if (previousSourceId && previousLayerId) {
                    map.value.removeLayer(previousLayerId);
                    map.value.removeSource(previousSourceId);
                }
                const featureToLoad = xuzhou.features[index];
                const sourceId = 'xuzhou' + index;
                const layerId = 'xuzhouLayer' + index;
                map.value.addSource(sourceId, {
                    type: 'geojson',
                    data: {
                        type: 'FeatureCollection',
                        features: [featureToLoad]
                    }
                });
                map.value.addLayer({
                    'id': layerId,
                    'type': 'fill',
                    'source': sourceId,
                    'layout': {},
                    'paint': {
                        'fill-color': '#FF0000', // 鲜艳的红色
                        'fill-opacity': 0.75,
                        'fill-outline-color': '#FFFFFF'
                    }
                });
                map.value.on('click', layerId, (e) => {
                    const clickedFeature = e.features[0];
                    const namech = clickedFeature.properties.name_ch;
                    const typech = clickedFeature.properties.type_ch;
                    const begyr = clickedFeature.properties.beg_yr;
                    const endyr = clickedFeature.properties.end_yr;
                    new mapboxgl.Popup()
                        .setLngLat([117.1113, 34.1554])
                        .setHTML(
                            `<p>名称 ${namech}</p>
                            <p>类型: ${typech}</p>
                            <p>开始时间: ${begyr}年</p>
                            <p>结束时间: ${endyr}年</p>`
                        )
                        .addTo(map.value);
```

```
                });
                previousSourceId = sourceId;
                previousLayerId = layerId;
                index++;
            } else {
                setTimeout(() => {
                    clearInterval(interval); // 所有要素都已加载，清除定时器
                    interval = null;
                    if (previousSourceId && previousLayerId) {
                        map.value.removeLayer(previousLayerId);
                        map.value.removeSource(previousSourceId);
                    }
                }, 0); // 使用 setTimeout 确保这段代码在当前执行栈之后运行
            }
        }, 2000);
    }
};
```

3.4 Axios 技术

Axios 是一个基于 Promise 的 HTTP 客户端，用于浏览器和 node.js 环境，广泛用于前端开发中进行异步 HTTP 请求。它是一个轻量级的库，提供了简洁的 API 用于处理 XMLHttpRequests (浏览器) 和 http 接口 (Node.js)。Axios 的主要特点包括：简单易用的 API，基于 Promise 的异步处理，请求和响应的拦截器。和 Ajax 相比，Axios 为处理 HTTP 请求提供了一个现代化和高度可配置的接口，它提供了比传统 AJAX 方法更高级、更灵活的解决方案。

通过 Axios 创建一个 HTTP 请求实例，并通过拦截器（interceptors）处理请求和响应，以及集成 Vue 3 的状态管理和路由：

```
import axios from "axios";
import { ElMessage } from 'element-plus'
const baseurl = "/api";
const instance = axios.create({
  baseURL: baseurl,
});
import { useTokenStore } from "@/stores/token";
import router from "@/router";
instance.interceptors.request.use(
  (config) => {
    const toeknStore = useTokenStore();
    if (toeknStore.token) {
      config.headers.Authorization = toeknStore.token;
```

```
    }
    return config;
  },
  (err) => {
    return Promise.reject(err);
  }
);
instance.interceptors.response.use(
  (result) => {
    if (result.data.code === 0) {
      return result.data;
    }
    ElMessage.error(result.data.message ? result.data.message : '请求失败');
    return Promise.reject(result.data);
  },
  (err) => {
    //判断响应状态码
    if (err.response.status === 401) {
      ElMessage.error('登录状态失效，请重新登录');
      router.push("/");
    } else {
      ElMessage.error('服务异常');
    }
    return Promise.reject(err);
  }
);
export default instance;
```

3.5 GPT 接口

开发基于 ChatGPT 接口的应用涉及到与 OpenAI 提供的 GPT（Generative Pre-trained Transformer）模型进行交互，以实现自然语言处理任务，包括文本生成、对话系统、内容摘要、语言翻译等。这些应用可以广泛应用于客服自动化、教育辅导、内容创作、娱乐互动等多个领域。

本项目使用 GPT3.5-turbo 开发历史聊天机器人，实现聊天对话功能，代码如下：

```
export function chatBot() {
    const sendChatBtn = document.querySelector(".chat-input span");
    const chatInput = document.querySelector('.chat-input textarea');
    const chatbox = document.querySelector('.chatbox');

    let userMessage;
```

```
7.      const API_KEY = "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";
8.      const inputInitHeight = chatInput.scrollHeight;
9.
10.     const createChatLi = (message, className) => {
11.         const chatLi = document.createElement("li");
12.         chatLi.classList.add("chat", className);
13.         let chatContent = className === "outgoing" ? `<p></p>` : `<span class="
material-symbols-outlined">smart_toy</span><p></p>`;
14.         chatLi.innerHTML = chatContent;
15.         chatLi.querySelector("p").textContent = message;
16.         return chatLi;
17.     }
18.     const generateResponse = (incomingChatLi) => {
19.         const API_URL = "https://api.openai.com/v1/chat/completions";
20.         const messageElement = incomingChatLi.querySelector("p");
21.
22.         const requestOption = {
23.             method: "POST",
24.             headers: {
25.                 "Content-Type": "application/json",
26.                 "Authorization": `Bearer ${API_KEY}`
27.             },
28.             body: JSON.stringify({
29.                 model: "gpt-3.5-turbo",
30.                 messages: [{ role: "user", content: userMessage }]
31.             })
32.         };
33.         fetch(API_URL, requestOption).then(res => res.json()).then(data => {
34.             messageElement.textContent = data.choices[0].message.content;
35.         }).catch(err =>
36.             messageElement.textContent = "服务异常，请稍后再试").finally(() => {
37.                 chatbox.scrollTo(0, chatbox.scrollHeight);
38.             });
39.     }
40.     const handleChat = () => {
41.         userMessage = chatInput.value.trim();
42.         if (!userMessage) return;
43.         chatInput.value = "";
44.         chatInput.style.height = `${inputInitHeight}px`;
45.
46.         chatbox.appendChild(createChatLi(userMessage, "outgoing"));
47.         chatbox.scrollTo(0, chatbox.scrollHeight);
48.         setTimeout(() => {
49.             const incomingChatLi = createChatLi("思考中", "incoming");
```

```
            chatbox.appendChild(incomingChatLi);
            chatbox.scrollTo(0, chatbox.scrollHeight);

            generateResponse(incomingChatLi);

        }, 500)
    }
    chatInput.addEventListener("input", () => {
        chatInput.style.height = `${inputInitHeight}px`;
        chatInput.style.height = `${chatInput.scrollHeight}px`;
    });
    chatInput.addEventListener("keydown", (e) => {
        if (e.key === "Enter" && !e.shiftKey && window.innerWidth > 800) {
            e.preventDefault();
            handleChat();
        }
    });

    sendChatBtn.addEventListener("click", handleChat);
}
```

四、作品特色（特点）

千年舆图—中国历史地理电子地图系统，采用前后端分离设计模式，高性能地图库 Mapbox，将历史地理数据通过网页直观的展现出来，帮助用户通过地图的形式了解中国历史，本系统有以下特点：

1、具有良好的网页界面：

系统前端采用 Vue3 框架，利用其响应式和组合式 API 特性，提高了开发效率和应用性能。ElementPlus 作为 UI 组件库，使界面设计不仅美观而且用户友好。整合这些前端技术，确保了网页界面的现代化和流畅性，同时也保证了良好的用户体验和交互设计。

2、响应速度较快：

Mapbox 相较于 Openlayers 和 Leaflet，Mapbox 能够更快地渲染大规模的地理数据和复杂的地图样式，其先进的 GPU 加速渲染技术和高效的数据处理能力。通过使用 Mapbox，本系统能够实现快速加载和平滑的地图浏览体验，即便是在数据量较大的情况下也能保持良好的性能。

3、丰富的地理数据展示：

数据库中收录了从商周到清朝的历史地理数据，这个时间跨度覆盖了中国的主要历史时期，丰富了用户的知识体验，提供了一个独特的视角来理解中国的历史演变。

4、良好的交互体验：

通过整合 GPT 接口开发的历史机器人，系统提供了一个交互式的学习工具。用户可以通过自然语言查询具体的历史事件、人物或地点，历史机器人则能够提供详细的背景信息、相关的历史数据和地图展示。这种交互方式大大降低了用户获取信息的难度，提高了学习的效率和趣味性。同时，这种智能化的交互体验也使得用户能够更加深入地探索中国丰富的历史文化。

五、小结

本系统为千年舆图—中国历史地理电子地图平台，由于历史地理数据获取困难，且开发难度较大导致目前历史地图科普网站较少，但是传承和科普中国历史十分重要，因此我们基于 WebGIS 开发该系统，帮助用户更便捷的了解历史地理知识。

我们采用前后端分离的开发模式，后端选择了 Postgresql 数据库来存储历史地理数据，并利用 SpringBoot 3 框架设计地理数据接口和用户数据接口。前端则使用 Vue 3、ElementPlus 及 Echarts 构建了直观且响应迅速的用户界面，而 Mapbox 地图库的引入则进一步丰富了地图展示功能。通过该项目的开发，我们不仅深化了对 WebGIS 开发技术的理解，也提升了自身的开发技能。

尽管项目的开发周期相对较短，并且功能界面尚待丰富，历史地理数据的获取难度也限制了系统内容的多样性，导致目前展示的历史数据相对单一。我们期望未来能够持续对该项目进行深入开发，计划引入三维历史场景、知识图谱、地图动画等创新功能，以期不断完善系统的构建，使之成为一个更加全面且生动的历史地理学习平台。

六、分工

姓名	负责内容的标题号	负责开发界面
马骁	2.3 【系统实现】	系统初始界面
	2.4【系统功能介绍】	系统登录界面
	2.5【系统未来开发方向】	账号注册界面
	三【技术路线与关键技术】	“我的”界面
	四【作品特色（特点）】	地图展示—历史地理界面
	五【小结】	历史地理—历史科普界面
		历史地理—数据展示界面
		地图展示—专题地图界面
		历史文档界面
		历史 AI 界面
韦娜	1.1【开发背景】	系统初始界面