附件1：中文封面页

|  |
| --- |
| **学士学位论文** |
| **基于WebGIS的多功能地图在线应用系统设计** |
| |  |  | | --- | --- | | 学 号： | 20171000664 | | 姓　 名： | 朱宇宸 | | 学 科 专 业： | 测绘工程 | | 指 导 教 师： | XXX 职称 | |  | XXX 职称 | | 培 养 单 位： |  | |
| 二○二一年四月 |

附件2：原创性声明

中国地质大学（武汉）学士学位论文原创性声明

本人郑重声明：本人所呈交的学士学位论文《基于WebGIS的多功能地图在线应用系统设计》，是本人在指导老师的指导下，在中国地质大学（武汉）攻读学士学位期间独立进行研究工作所取得的成果。论文中除已注明部分外不包含他人已发表或撰写过的研究成果，对论文的完成提供过帮助的有关人员已在文中说明并致以谢意。

本人所呈交的学士学位论文没有违反学术道德和学术规范，没有侵权行为，并愿意承担由此而产生的法律责任和法律后果。

学位论文作者签名：

日 期： 年 月 日

附件3：目录

**目 录**

图清单 I

表清单 Ⅱ

第一章 绪论 1

第二章 X 3

2.1 X 3

2.1.1 X 3

2.1.2 X 3

……………

2.2 X 4

2.2.1 X 4

……………

……………

……………

……………

……………

……………

第八章 结论 81

致谢 82

参考文献 83

附录 85

附件4：图和表清单

图清单

图2-1 X 4

图2-2 X 6

图3-1 X 10

表清单

表2-1 X 4

表2-2 X 6

表3-1 X 10

**摘要**

地图作为一项极其古老的发明，随着人类文明历史的演变进程也一直发生着进化。而在科技日新月异的今天，它也巧妙地结合了遥感、计算机科学等其他学科的优势，演变出了地理信息系统这一门新兴的综合性学科。而在万物互联互通的互联网时代，地图又能以网页的形式，出现在人们触手可及的各种有联网功能的终端设备之上，形成了WebGIS这一门最新的技术。WebGIS具有传统GIS的特点，可以实现空间数据的检索、查询、制图输出、编辑等GIS基本功能，同时也是在Internet上地理信息发布、共享和交流协作的基础。[1]

本系统就是在结合了目前工业界较为前沿的互联网技术，结合高德地图的实时更新的网页地图功能库，并以校友间的基础社交作为应用场景，开发出了一套兼具社交活动、实时导航、地图信息距离测量、面积测量等功能的在线网页应用。

本系统用的是浏览器/服务器前后端分离的架构（又称B/S架构），并利用Internet的HTTP协议实现在线访问交互，以Golang编程语言和Linux操作系统自行搭建的服务作为服务端，并以浏览器作为客户端载体。旨在利用一些我在实习期间所学到的最新知识，快速开发并搭建出一套可以在线浏览并且性能优越的多功能在线地图应用。

**关键词：WebGIS Golang JavaScript 网页应用 导航 社交**

**Abstract**

As an extremely old invention, maps have evolved along with the history of human civilization. In today's fast-changing technology, it has also evolved into a new comprehensive discipline of GIS by skillfully combining the advantages of other disciplines such as remote sensing and computer science. WebGIS has the characteristics of traditional GIS. which can realize the basic functions of GIS such as spatial data retrieval, query, map output, and editing, and is also the most advanced technology in the Internet. WebGIS has the characteristics of traditional GIS and can realize the basic functions of GIS such as spatial data retrieval, query, cartography output, editing, etc. It is also the basis of geographic information dissemination. Sharing, communication and collaboration on the Internet.

This system is to develop a set of online web application with functions of social activities, real-time navigation, distance measurement of map information, area measurement, etc. by combining the more cutting-edge Internet technology in the current industry with the real-time updated web map function library of Gaode Map and the basic social interaction among alumni as the application scenario.

The system uses a browser/server front and back-end separation architecture (also known as B/S architecture), and uses the Internet's HTTP protocol to achieve online access interaction, using the Golang programming language and a service built by the Linux operating system itself as the server, and a browser as the client carrier. The aim is to develop and build a multi-functional online map application that can be browsed online with superior performance, using some of the latest knowledge I learned during my internship.

**Keywords: WebGIS Golang JavaScript Web-Application Navigation Social**

# 1绪论

## 研究背景与意义

自从中国改革开放以来，我国社会经济发展的速度越来越快，并且伴随的人类的信息革命的完成，21世纪以来全国人民的生活都与以往有了翻天覆地的变化。以往的地图仅仅作为承载有限数量地物地貌信息的载体，供人们出行时提供方便。但是免不了信息更新不及时，不易保存以及使用起来较为繁琐等问题。而自从地理信息系统、计算机科学、遥感科学等众多前沿科学的高速发展并相继取得成功之后，地图已经变成了众多最新技术展示最好呈现方式之一。而地图载体的表现形式也不仅仅拘泥与之前的纸张，书本。网页，手机应用程序也都可以承载地图。并且随着手机的普及程度的提高，越来越多的人在出行的时候会选择在手机应用中利用导航功能查看对应的出行路线，而不是使用纸质地图，这也从侧面佐证了在线地图应用的普及程度。

但是随着社会的高速发展，只含有最基础功能的地图已经无法满足人民日益增长的物质文化需求了，所以现在的在线地图应用的功能也越来越多，同时也有越来越多的其他软件在自己的基础功能之上嵌入了地图，来满足用户越来越多的需求。但是目前市面上的绝大多数应用仍然会有比较多的缺陷，无法满足部分需求。

综上所述，开发一款多功能的地图在线应用，已经是是国家和社会的迫切需要，同时也便于满足人民日益增长的物质文化需求。故本文以高德地图的WebGIS相关API为基础，开发出了一款可以同时满足手机端和个人电脑端的兼备社交、导航、娱乐等功能的在线地图应用。同时此应用采用较为成熟的组件化设计，具有高内聚低耦合的特点，因此还含有高扩展性，便于后期维护以及新增其他的功能。最终本人将该系统命名为“基于WebGIS的多功能地图在线应用”，对外发布版本则命名为CUGMap（地大地图），一来为自己本科四年交一份满意的答案，二来为现有的相关的地图应用软件提供一个可以拓展的方向。

## 国内外研究现状

随着社会科技的不断发展，以及改革开放以来国内的经济建设速度加快，人们的生活变得日新月异的同时城市建设的速度也和以往不可同日而语。我国多地大力推进城市基础设施的建设。以我求学的城市武汉为例，在短短四年建设了八条地铁线路，政府也提出了“武汉每天都不一样”的宣传口号，在这种情况之下如果仍然采用旧的纸质地图出行，必然会遇到地图更新不及时、线路不准确、难以保存等问题。所以国家也在大力建设地理信息基础信息库，以服务器存储地理信息、联网即时更新的方式来应对现有的情况，多款地图软件也斥巨资购置了测绘车辆在全国各地绘制并记录最新的地理信息。同时，高德地图、百度地图等在线地图软件也在假期频频登上各大手机厂商的下载排行榜前列。由此可见，人民对于在线地图应用的需求非常旺盛，国内对于在线地图应用的研究已经开始了很长一段时间，但是至今没有特别好的产品。

而地理信息系统、计算机科学等这些学科本身就是由国外最先提出并传入国内的，所以国外的研究也更领先于国内，例如谷歌地球等大型在线应用已经应用与生活的方方面面，帮助他们解决平时出行、研究等各种方面遇到的困难。但是国外对于地图和其他功能应用的整合却没有国内做得好，如美团、大众点评等借助地图而生的本地生活应用已经在国内非常普及的现在、据我的调查国外许多人在本地生活领域仍然依靠口耳相传，这是我们的优势。

## 本文主要工作

本文主要从研究内容、技术路线和文章结构三个方面来阐述本文的主要工作。

### 研究内容

本文的研究对象借助高德地图WebGIS基础功能，如何实现最基本的地图在线应用，并在此基础之上增设服务器，满足一些社交、娱乐等需求的本人新开发的新功能。本文将对我在开发过程中使用到的最新技术已经解决困难过程中的思考进行描述。

### 技术路线

本系统的主要界面搭建主要是通过由Facebook公司开发的开源Web开发框架React的轻量级版本React Hooks、结合国内大部分公司所采用的集编译检查、开发热更新、并利用TreeShaking将模块依赖打包成静态资源的Web开发工程化工具Webpack进行开发的。其中主要使用了Web技术和服务端开发技术，Web技术主要是通过HTML（超文本标记语言）、CSS（层叠样式表）、JavaScript语言进行开发。而React Hooks + Webpack的开发方式也是我实习所在的公司所采用的大型项目的开发方式、据我所知这种开发方式具有开发效率高、Web性能好、可维护性高等一系列优点、目前国内国外绝大部分技术领先的公司都在采用这种开发方式。服务端开发的技术选型主要是以Gin+ Mysql的技术架构。其中Gin是Google公司在2009年最新开源的Golang（以下简称Go语言）编程语言的一个高性能服务端框架，具有中间件支持、错误恢复、性能监控等功能。而Go语言是一门静态强类型语言，具有内存安全、垃圾回收、支持多线程的高性能服务器的特点，是现在许多大型企业作为服务端语言的首选。其总体技术路线如图1-2所示。

|  |
| --- |
| WBS |
| 图 1‑2 技术路线 |

### 文章结构

第一章是绪论部分。在这一部分先介绍了本文研究的国内外背景与开发这个系统的意义，介绍一下如今国内外在这方面的的研究情况，并在此基础上对本文的主要研究内容进行概述。

第二章是系统总体设计。在这一章节中首先会对本系统的开发环境做个描述，然后对数据库设计中的表结构进行展示，然后对本系统的架构设计进行概述，最后以图文并茂的形式介绍本文的功能设计。

第三章是具体介绍系统所用到的一些比较关键的技术。在简单介绍完构架设计之后，我将在第四章具体介绍我在开发本项目过程之中所用到的一些比较重要的技术点以及使用这些技术所解决的问题以及优缺点的分析。

第四章是系统中遇到的一些难点与痛点，本章中我将详细描述我在开发本系统过程之中遇到了什么问题以及如何将问题解决的过程。

第五章是全文性的一个总结以及期望。在本章中我会首先对全文做一个总结，然后针对本系统的不足做出未来的展望。

# 2.系统总体设计

## 2.1开发环境

本系统是在MacOS系统环境下进行开发的。本系统所使用的前端程序编写工具VS Code，后端程序编写工具是GoLand，通过在Mac开发机上开发完成，将对应服务端代码打包编译成Linux操作系统的Centos7.0版本，并在国内共有云上租用的云服务器部署对应的Mysql数据库环境，连接对应的环境并部署，申请对应的域名www.cugmap.com并进行域名解析，使得对应的请求能被服务器所响应。本系统所使用的网络通信数据为json格式，数据存储在Mysql（innodb存储引擎版）数据库中。本系统所使用的技术为服务端开发与部署技术和Web前端技术。本系统前端部分所使用的开发语言为HTML（超文本标记语言）、CSS（层叠样式表）、JavaScript，所涉及的库为Axios，React Router。所使用的JavaScript开发框架为React Hooks。后端开发语言为Go语言，所涉及的库为jwt-go Token鉴权库、go-sql-driver数据库驱动库，所使用的Go开发框架为Gin框架。

## 2.2数据库设计

本系统的数据库采用的是Mysql（innodb存储引擎版）数据库，Mysql则是一款市场占有率第一名的高性能的数据库，采用InnoDB存储引擎，因此可以对坐标数据添加对应的索引并利用B+树的数据结构达到高效查询的目的，并持久化地存储用户与地物信息的各项数据。在本系统设计中一共有四个表，用户信息表，用户聊天记录表，地物信息表以及地物评论记录表。分别对应下表中的表2-1、表2-2、表2-3、表2-4

|  |  |  |
| --- | --- | --- |
| 表2‑1 用户信息表表结构 | | |
| 字段名 | 数据类型 | 说明 |
| student\_id | 11位变长字符串 | 用户的账号（学号）、主键 |
| username | 20位变长字符串 | 用户姓名 |
| password | 20位变长字符串 | 用户密码 |
| lng | 有效数字为9位的浮点数 | 用户所在位置经度 |
| lat | 有效数字为9位的浮点数 | 用户所在位置纬度 |
| signature | 50位变长字符串 | 用户个性签名 |
| 索引名 | 索引字段 | 说明 |
| PRIMARY | student | Mysql自动生成的主键索引 |
| position\_index | lng,lat | 经纬度联合索引，用于快速匹配附近的人 |

|  |  |  |
| --- | --- | --- |
| 表2‑2 用户聊天记录表表结构与索引 | | |
| 字段名 | 数据类型 | 说明 |
| message\_id | 长整型 | 消息代号 |
| receiver\_id | 11位定长字符串 | 消息接受者账号 |
| sender\_id | 11位定长字符串 | 消息发送者账号 |
| send\_time | 13位定长字符串 | 消息发送时间(时间戳类型) |
| Is\_read | 布尔型 | 消息是否已读 |
| send\_time\_str | 时间日期类型 | 消息发送时间(时间格式类型) |
| message | 50位变长字符串 | 消息内容 |
| with\_place | 长整型 | 所含地点代号(地物信息表外键) |
| 索引名 | 索引字段 | 说明 |
| PRIMARY | message\_id | Mysql自动生成的主键索引 |
| receiver\_index | receiver\_id | 接受者Id索引，用于快速查找聊天记录 |
| sender\_index | sender\_id | 发送者Id索引，用于快速查找聊天记录 |

|  |  |  |
| --- | --- | --- |
| 表2‑3 地物信息表结构 | | |
| 字段名 | 数据类型 | 说明 |
| place\_code | 长整型 | 地物代号 |
| name | 50位变长字符串 | 地物名称 |
| founder | 11位定长字符串 | 地物添加人 |
| founder\_comment | 200位变长字符串 | 添加人评价 |
| score | 有效数字3位的浮点数 | 地物评分 |
| type | 短整型 | 地物类型 |
| phone\_number | 15位变长字符串 | 地物联系方式 |
| address | 50位变长字符串 | 地物详细地址 |
| lng | 有效数字9位的浮点数 | 地物经度 |
| lat | 有效数字9位的浮点数 | 地物纬度 |
| image1\_url | 3000位变长字符串 | 地物图片链接1 |
| image2\_url | 3000位变长字符串 | 地物图片链接2 |
| comment\_number | 长整型 | 地物评价数量 |
| 索引名 | 索引字段 | 说明 |
| PRIMARY | place\_code | Mysql自动生成的主键索引 |
| position\_index | lng,lat | 经纬度联合索引，用于快速匹配附近的地物 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表2‑4 地物评论记录表表结构 | | | | |
| 字段名 | | 数据类型 | | 说明 |
| comment\_id | | 长整数 | | 地物评论代号 |
| place\_code | | 长整数 | | 对应地物代号 |
| commentator | | 11位定长字符串 | | 评论者账号 |
| commentator\_message | | 200位变长字符串 | | 评论内容 |
| score | | 短整型 | | 评论者打分 |
| comment\_time\_str | | 时间日期类型 | | 消息发送时间(时间戳类型) |
| comment\_time | | 13位定长字符串 | | 消息发送时间(时间格式类型) |
| likes | | 长整数 | | 评论点赞数 |
| 索引名 | 索引字段 | | 说明 | |
| PRIMARY | comment\_id | | Mysql自动生成的主键索引 | |

## 2.3系统架构设计

本系统采用四层架构，分别是数据层、服务层、业务逻辑层和接入层。其中数据层采用高性能关系型数据库Mysql来存储和管理数据。服务层通过Go语言的Gin框架提供的高性能服务器，添加Token鉴权、Mysql数据库驱动操控数据库来提供一些接口服务。业务逻辑层主要是该多功能地图应用主题的六大功能模块，具体功能将在下一部分详细介绍。同时由于本系统采用的是Web适应性布局，所以可以同时兼容多种接入方式，包括电脑端浏览器、移动端浏览器以及App内置的浏览器均可以访问本系统，增大了用户接入的渠道，为各种不同的用户提供方便，同时本系统对于不同屏幕的宽度进行了特殊的适配，所以不需要担心特殊机型的客户端上出现适配问题的情况出现，其适配结果如图3-1、3-2所示。而其完整的系统架构设计图如3-3所示。

|  |
| --- |
| Go多线程服务  MySQL  阿里云服务器 |
| 图 3‑1 个人电脑端（宽屏）适配情况 |

|  |
| --- |
| Go多线程服务  MySQL  阿里云服务器 |
| 图 3‑2 移动断（竖屏）端适配情况 |

|  |
| --- |
| 系统架构设计  Go多线程服务  MySQL  阿里云服务器 |
| 图 3‑3 系统架构设计 |

## 2.4系统功能设计

本系统功能共分为六大功能模块，其中地图基础功能包括了地图缩放、平移、切换三维模式显示、展示遥感图像、显示实时路况等基础功能。而注册和登陆则是通过服务层提供的接口来进行用户的基础功能，后续功能必须通过登陆才能使用。而高级功能工具栏则是一些其他地图没有的功能，例如距离测量，面积测算，常用地点快速定位以及校区导航等，该工具栏可以自由关闭或者开启，提供给有特殊需求的用户使用。搜索附近的人和本地生活则是本地图的亮点部分，可以为用户搜索用户附近的人与商铺等地物信息，搜索之后可以与他人进行聊天，社交并会保存用户的好友信息，以便用户更换位置后仍然能取得联系，同时也可以在聊天过程之中给其他用户推荐地点信息，方便一同前往，所有地点均能提供点到点的导航功能。如果进入本地生活模式，则可以自动匹配附近的地物，也可以按类型进行分类，目前有美食，娱乐设施以及基础设施三种分类，点击对应的地物可以查看他人的评论并留下您的评论或者打分，也可以与其他评论人沟通，两个模式相辅相成，共同提供地图的特色功能。具体功能说明如图3-4所示。

|  |
| --- |
| 基于WebGIS的多功能地图在线应用 |
| 图 3‑4 具体功能 |

# 3关键技术

本系统采用B/S模式进行设计，所使用的的关键技术如图3-1所示。

|  |
| --- |
| 基于WebGis的多功能地图 (3) |
| 图 3‑1 关键技术 |

## 

## 3.1 Web前端技术

本系统的前端页面搭建主要采用了Node.js技术，其中具体地是使用了React.js技术、Webpack技术和Koa技术。

React是一套具有声明式、组件化特性、用于构建用户界面的JavaScript框架。与以往的MVC框架不同的是，React是一款使用了MVVM架构的框架，这种架构的好处就是可以将图形用户界面的开发与业务逻辑或后端逻辑（数据模型）的开发分离开来。同时，其利用了虚拟DOM技术以及加入了其特有的Diff算法，使得页面的渲染速度较以往传统的MVC架构的开发框架（如JQuery）有了质的提升，同时，其单向数据流的特点也保证了软件的健壮性，使得其在搜寻问题的时候的便利性要由于Vue.js这样具有双向数据绑定的视图层框架。在该系统之中，我用到了React.js的一个新特性——React Hooks，这种函数式组件与原生的React相比具有更加轻量的特点，同时其特有的Hooks生命周期函数可以加快开发的速度，也是我实习所在公司许多新项目所会选取的开发技术选型。另外由于本应用是一个多页面应用，用户可以在不同的模式之中切换页面，如社交、本地生活以及注册登陆等基础功能页面。所以本次还采用了React Router作为路由组件，这样在页面切换的时候可以保证页面之间的数据传输不会受到影响。

至于Axios库则是对于Ajax网络请求方式进行了一个封装的库，也是目前市面上最受欢迎的网络请求库，本系统中所有前端的请求均是通过Axios库进行的。

Webpack则是目前在前端大型项目市场占有率超过百分之九十的打包构建工具，其最大的特点是可以把ES6以上的语法模块以及某些库的文件类型（如Jsx、Sass类型）打包构建成ES5之前的语法的静态资源（HTML、CSS、JS文件），来兼容一些旧版本的浏览器。同时也具有类似于TreeShaking、分包构建、异步加载等高级特性，可以根据项目的需求开启或者关闭这些功能。同时在开发阶段使用Webpack，也具有编译检查，开发热更新，模块性能分析等功能。其大致功能如图3-2所描述。

|  |
| --- |
|  |
| 图 3‑2 Webpack功能描述 |

而Koa.js，则是一个对Node.js的http库进行封装的框架，具有中间件技术、静态资源访问，路由拦截等大部分http框架所具有的功能，在本项目中利用了Koa框架对于Webpack打包出来的静态资源进行了监听，以便用户在访问对应地址时能即使在Linux服务器端进行监听并将对应的资源进行返回，同时对路由进行了拦截保护，以便用户输入错误的路径时仍然能导向正确的资源。

## 3.2 Web服务端技术

本系统的服务端页面搭建主要采用了Gin+Mysql的相关架构，Gin框架是Go语言目前市场占有率最高的大型项目HTTP服务框架，同时也是我实习所在公司所使用的服务端框架之一，这也是我在该系统中采用了Go语言作为服务端的主要语言的原因。由于本系统是一个需要登陆才能使用绝大部分功能的用户系统，所以如果保存用户的登陆状态就成了非常重要的一件事情。至于本系统如何安全有效的储存用户的登陆状态，采用的则是Json Web Token这种技术手段，关于这种技术手段的将放在后面的技术难点之中详细介绍，在此便不再赘述。

在本系统中，Gin框架中的路由组建的用处则是将不同功能的请求进行分类，并封装成为不同的路由组建，这样在进行代码开发和维护的过程之中就能快速定位到对应的组件之中，而不会产生混乱，使得代码的可读性和可维护性都大大增加，符合高内聚低耦合的程序开发的规范。性能检测则是对与开发过程中不同接口设计方式的最好检测方式，每个请求的具体耗时都可以很清楚的展示在开发机上，由此来选择一种性能最好的接口设计方式，来提高系统的健壮性与稳定性，并优化用户的体验。错误恢复也是同样的作用，通过恢复错误并打印调用栈以及记录错误日志的方式来提高程序的稳定性，保证在个别错误请求的时候不会使得服务器宕机，来影响其他请求。而错误栈和日志则是为了追查错误的时候更加方便快速的定位问题，通过修复零星出现的问题来一步步提升程序的稳定性。

系统中所使用的SQL驱动的时候则是为了对数据进行持久化的操作，在本系统中采用数据库的就是市场占有率第一的Mysql高性能关系型数据库来对数据进行持久化，而SQL驱动则是让Go语言操纵数据库的必备工具。在本系统中采用了Mysql的InnoDB存储引擎，因此可以对一些高频数据添加B+树的索引，大大加快了信息查询的速度，减少了请求的耗时，从而优化用户的体验。

## 3.3 数据可视化

本系统的地图WebGIS地图基础功能是由高德地图的高德开放平台提供的地图JS API所提供的，在本系统中一共引用了API以及插件中的获取用户地点信息、地图缩放、三维建模、点到点导航、测量测距工具、遥感图像等功能来优化地图的功能，使得用户在不登陆的情况之下仍然可以使用绝大部分基础功能满足自己的需求。至于IconPark则是一款免费的图标库，用到软件之中给用户和地物做标准，可以有效的提升软件的美感，使得用户体验更好。

# 4 技术难点

本系统作为一款功能完善、软件健壮性良好的系统，在开发的过程之中必然是会遇到了各式各样的困难。如果说前面的基础功能只是增加我学习以及代码编写过程之中的熟练度，那下述的技术难点则是我自认为在我进行这个项目有所突破的地方，解决这些技术难点过程之中的思考和手段都给我留下了深刻的印象，我相信我以后如果再遇到类似困难的时候就会处理得更加得心应手了。

## 4.1 用户鉴权

由于本系统是一个需要记录用户信息并由此提供更加完善服务的用户系统，所以记录用户的登陆状态以及判断用户就成了非常必要的一个需求。但是本系统是建立在Web之上的一个系统，与需要安装的手机或电脑应用程序不同，Web应用是依赖HTTP请求进行消息的传递的，但是由于HTTP请求本身是一种无状态协议，导致了请求是无法记录用户状态更是无法鉴权的，所以在经过我查阅资料之后，我发现目前市面上主要有两种技术方案可以用于对用户进行安全可靠地鉴权，一种是使用cookie+session鉴权的方式，另一种是利用Json Web Token对用户鉴权的方式（以下简称JWT鉴权）。由于第一种鉴权方式存在cookie的跨域导致CSRF攻击漏洞以及需要将session存储在服务端耗费存储资源等各种问题，目前绝大部分企业都已经采用JWT鉴权的方式保存状态，本系统鉴于上述的考虑也采用了此方案。JWT技术的主要原理就是在用户登陆的时候在服务端通过一个存储在服务器的密钥进行对称加密技术生成一段和用户信息、登陆时间相关的加密字段，并通过请求返回至前端，再由前端存储到cookie当中。当用户进行任何请求的时候，都需要带上这个cookie存储在请求头的某个字段中（本系统是存储在请求头的Authorization字段之中），这样服务端在接收到请求的时候便可以对这个字段用之前生成的对称密钥进行校验，由此解析出用户的信息以及上次的登陆时间，如果出现字段为空、用户信息错误、用户登陆时间过期等多种鉴权失败情况中的任意一种，则令服务端拒绝本次请求，并提示前端用户登陆状态有误，让用户进行重新登陆的情况，其完整流程如图4-1。

|  |
| --- |
| jwt |
| 图 4-1 JWT鉴权流程图 |

上述流程在本系统中的详细表现就是让JWT鉴权模块封装成一个中间件，再利用Gin框架的中间件技术将此中间价放置在所有请求相应之前，来达到用户鉴权的效果。

## 4.2 快速定位附近的人

在开发本系统过程之中，遇到的另一个比较难以解决的问题就是在开发社交模块以及本地生活模块的时候，需要给用户快速匹配附近的用户和地物信息。但是经过我最开始按照一个个点位逐一计算的方式构成系统的方式测试的时候，发现只要一请求附近的人的数据就会令服务器反应变得很缓慢甚至直接宕机。

## 4.2.1 计算过程优化

根据我的分析，我发现其原因有两个，第一是根据经纬度计算两点之间距离的公式非常繁琐，涉及到的运算非常之多，我用Go语言编写的Haversine公式完整函数代码如下

|  |
| --- |
| const (  Pi = 3.1415926535  R = 6367000 // 地球半径（米） )  type User2D struct {  lng float64  lat float64 } func GetDistance1(user1,user2 User2D) (distance float64) {  user1Lng := user1.lng \* Pi / 180  user1Lat := user1.lat \* Pi / 180  user2Lng := user2.lng \* Pi / 180  user2Lat := user2.lat \* Pi / 180  hsinX := math.Sin((user1Lng - user2Lng) / 2)  hsinY := math.Sin((user1Lat - user2Lat) / 2)  h := hsinY\*hsinY + math.Cos(user1Lat)\*math.Cos(user2Lat)\*hsinX\*hsinX  distance = 2 \* math.Atan2(math.Sqrt(h), math.Sqrt(1-h)) \* R  return } |

可以看到，在一次的距离计算之中，涉及到了一共六处三角函数计算和两次开平方计算。而对于计算机来说，浮点数的运算本身就很耗费性能，这样的计算无疑是很慢的，经过我在我的高性能开发机上的测试，计算一对坐标的距离所耗时大约在0.013毫秒，粗略的推算一下，光是计算十万对坐标之间的距离的耗时可能就需要1300毫秒，如果加上操控数据库，排序，以及网络请求的延时，一次请求耗费十几秒甚至数十秒也是非常正常的。更不要说用户量达到百万、千万级别的耗时了，该系统完全无法应对那个数量级的用户。甚至在面对上万用户的时候就已经非常吃力了。

本身该系统在数据库设计的时候有关用户地理位置信息的字段是采取字符串存储的，其具体采取的方式是将用户、地物的经纬度信息转化成字符串，再利用分号作为分隔符的方式存储。例如一位经度为114.345317，纬度为30.521812的用户的坐标信息会被转化为“114.345317;30.521812”的字符串存储。这样存储和提取数据的时候就能减少一个字段，略微提高效率。然后在进入社交模式之后，逐一提取用户信息表中全部用户的坐标信息，再根据登陆用户自身的坐标信息，逐一计算出该用户与其余用户的全部坐标信息，再对距离进行排序，提取距离用户最近的五十位用户的信息并将数据返回，再经由前端将对应坐标的用户的数据渲染到对应的地点，使得用户能与附近的人进行社交。这个功能本身功能正常，但是在我利用自己编写的脚本在中国范围内随机插入了十万个测试用户之后，这个功能就无法正常使用了。一个计算过程很有可能耗时高达十几秒，甚至直接能让服务器宕机，因为内存中无法存储那么多的用户信息。

针对这个问题，我先后想出了一下几个解决方案来应对更大规模的用户量级。

第一个解决方案就是对计算的过程进行优化。优化分成两个过程，第一步就是对距离计算的公式进行优化，因为每一对坐标都需要进行距离运算，所以如果能从这个公式本身上优化，就能成倍数的节省时间。在我在网络上查询类似坐标计算的方式的时候，看到了一篇美团公司技术团队所提出的一种优化计算坐标的思路，他通过对CPU利用率的分析发现了目前计算耗时的主要部分也是计算三角函数的过程，所以他们的优化策略主要放在了减少三角函数的计算上面。但是我们知道，根据经纬度求两点之间距离的公式是固定的，所以如果要做优化肯定是要从存储上面去思考，他们的方式就是通过坐标转化，将经纬度坐标信息(lng,lat)通过三角函数转化成球面模型下的三维坐标(x,y,z)，对应的映射方式如下：

然后通过三维坐标的转化，两点之间距离计算公示就变成了如下公式：

这样在进行亮点之间距离计算的时候就只需要计算一个反三角函数值了，效率大大提高，经过美团专业的数据测试，经过这样的优化可以将计算耗时降低至原来的百分之四十左右。如果仅仅考虑同城的情况，的值实际上是无限趋近于0的，这时候可以利用公式

将对应计算距离公式中进行替换，并化简，最终可以得到

下面是该公式对应的Go语言函数代码:

|  |
| --- |
| const (  Pi = 3.1415926535  R = 6367000 // 地球半径（米） )  type User2D struct {  lng float64  lat float64 } type User3D struct {  x float64  y float64  z float64 } func GetDistance2(user1, user2 User2D) (distance float64) {  // 数据存储过程  newUser1 := &User3D{  x: math.Cos(user1.lat) \* math.Cos(user1.lng),  y: math.Cos(user1.lat) \* math.Sin(user1.lng),  z: math.Sin(user1.lat),  }  newUser2 := &User3D{  x: math.Cos(user2.lat) \* math.Cos(user2.lng),  y: math.Cos(user2.lat) \* math.Sin(user2.lng),  z: math.Sin(user2.lat),  }    // 数据计算过程  AOB := newUser1.x\*newUser2.x + newUser1.y\*newUser2.y + newUser1.z\*newUser2.z  distance = (Pi/2 - AOB) \* R  return} |

这样做的话就能够完全消除三角函数的计算，大大优化计算效率了，经过性能测试，利用这个公示计算耗时可以降低至原来的三十分之一左右。虽然这样做在用户新增、更改数据的时候稍微计算会麻烦一点，但是能将计算过程分散开来，不需要在查询的时候再去计算大量数据的大量三角函数值，还是能够减少很多计算上的耗时的。但是这种问题也是有自身问题所在的，比如增多了一个字段，就增大了索引的开销，以及上述提到的在用户更改数据的时候的速度就会降低。

受到上述仅考虑同一城市内做法的启发，我发现对于同一城市两点之间距离的计算，在小范围内可以认为经线和纬线是垂直的，这样在计算两点之间距离的时候就可以将这个问题转化为平面坐标的问题，如图4-2。在依据对应的公式算出距离

|  |
| --- |
| dis |
| 图 4-2 简化距离计算示意图 |

下面是该公式对应的Go语言函数代码:

|  |
| --- |
| const (  Pi = 3.1415926535  R = 6367000 // 地球半径（米） )  type User2D struct {  lng float64  lat float64 }  func GetDistance3(user1, user2 User2D) (distance float64) {  AM := R \* (Pi / 180) \* (user1.lng - user2.lng)  BM := R \* math.Cos((Pi/180)\*((user1.lat+user2.lat)/2)) \* (user1.lat - user2.lat)  distance = math.Sqrt(AM\*AM + BM\*BM)  return } |

按照这种方法去进行坐标运算的话，也只会有一次三角函数运算。经过性能测试，利用这种方法计算耗时可以降低到原来的二十分之一左右，同时还可以正常记录用户的经纬度，以满足其他的一些需求，可以说是一种比较好的优化方式，但是由于这种方法直接近似地将经度线和纬度线认定成垂直关系了，我们需要验证一下这种计算公示的可靠性。下面是考虑北京范围内四对点位依据上述Haversine公式以及简化公式运算的结果。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试坐标点对 | Haversine公式运算结果（米） | 简化公式运算结果（米） | 差别（米） |
| (39.941,116.45)  (39.94, 116.451) | 140.0285167225230 | 140.02851671981400 | 0.0 |
| (39.96,116.45)  (39.94, 116.40) | 4804.421262839180 | 4804.421153907680 | 0.0 |
| (39.96,116.45)  (39.94, 117.30) | 72444.81551882200 | 72444.54071519510 | 0.3 |
| (39.26,115.25)  (41.04, 117.30) | 263525.6167839860 | 263508.55921886700 | 17.3 |

可以看到，两公式在千米尺度上几乎没有区别，在万米尺度上也仅仅只有分米级别的差别。这对于我们的同城业务来说是完全够用的，这也从侧面印证了在距离较近的情况之下可以将经纬度近似成普通的平面坐标系来对待。

其实对于上述仍然存在少量三角函数的公式，其实也可以采取傅立叶变换、多项式拟合等方式再配合机器学习的训练来达到消除三角函数计算，从而进一步达到优化计算的目的。这种方法在此就不再展开，欢迎有兴趣的读者自行尝试。在美团的团购筛选、商家筛选的距离排序以及只能排序等相关业务之中已经开始使用上述公示优化后的方法了，其可靠性都已经得到了验证。

然后对于排序的过程，由于我们仅仅需要的是前五十至一百位的用户的坐标信息，对于百万的数据来说全部进行排序是很没有必要的一件事情，所以我将原本的快速排序算法（时间复杂度为）替换为了查找第K大元素算法（时间复杂度为）。这一操作也使得整体的性能提升了不少，减少了运算次数。

## 4.2.2 数据库优化

在经过了上述的这些努力之后，我原以为这个查找附近的人的接口的响应速度能有质的突破，但是现实却是仅仅从数十秒缩短到了数秒，经过我的分析，数据库操作中的磁盘读写操作也会耗费非常多的时间，而这一块是上述无论何种方法都无法优化到的，即使相应时间已经大大缩短了，但是按照目前这个相应速度对于用户来说肯定也是不能接受的，我必须在此之上寻找其他的方法。

由此看来，目前最重要的优化方向就在于如何减少数据库的操作次数，只要解决了这个问题，整个问题就迎刃而解了。我最先想到的方法是替换掉本身的关系型数据库，而采用非关系型数据库中的图数据库进行存储，在[计算机科学](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97_(%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6)" \o "计算 (计算机科学))中，图数据库是一个使用[图结构](https://zh.wikipedia.org/w/index.php?title=Graph_(data_structure)&action=edit&redlink=1" \o "Graph (data structure)（页面不存在）)进行[语义查询](https://zh.wikipedia.org/w/index.php?title=Semantic_query&action=edit&redlink=1" \o "Semantic query（页面不存在）)的[数据库](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93" \o "数据库)，它使用[节点](https://zh.wikipedia.org/wiki/%E9%A1%B6%E7%82%B9_(%E5%9B%BE%E8%AE%BA)" \o "顶点 (图论))、[边](https://zh.wikipedia.org/wiki/%E5%9B%BE%E8%AE%BA%E6%9C%AF%E8%AF%AD" \o "图论术语)和属性来表示和存储数据。该系统的关键概念是图，它直接将存储中的数据项，与数据节点和节点间表示关系的边的集合相关联。这些关系允许直接将存储区中的数据链接在一起，并且在许多情况下，可以通过一个操作进行检索。图数据库将数据之间的关系作为优先级。查询图数据库中的关系很快，因为它们永久存储在数据库本身中。可以使用图数据库直观地显示关系，使其对于高度互连的数据非常有用。[[2]](https://zh.wikipedia.org/wiki/%E5%9B%BE%E6%95%B0%E6%8D%AE%E5%BA%93" \l "cite_note-:0-2)图状数据的结构如图4-3所示

|  |
| --- |
| graph1 |
| 图 4‑3 某种图状数据结构图 |

但经过我的简单的调查之后，目前市面上开源的图数据库的稳定性都有待验证，并没有很大范围的应用到具体业务之中，贸然使用很容易出现问题，同时社区不够完善也会导致问题难以得到排查也解决。而且此时我的数据库已经搭建完成，内部也存有了很多数据，如果切换使用到另一种完全不同的数据库，数据的导入导出以及对应接口的修改过程之中必然也会有很多问题需要修复，再加上高昂的学习成本，我最终没有采用这种方式对数据库操作进行优化。但是我相信如果采取这种方式，也能在一定的程序上优化大量数据查找附近的人这一操作的耗时。但是这种方法的缺点也比较明显，就是在插入新用户或者旧更改数据的时候会比传统的关系型数据库要略慢一些，但是这样也算是平摊运算成本的一种方式，但是这种摊平运算成本的方式也同样有着上面通过计算三维坐标优化计算过程的无法及时取得用户确切坐标信息的缺点。

## 4.2.3 划分空间存储

经过了上述几个方法的研究之后，我发现了这个问题的本质还是在于用户数量太大，坐标运算过程又太过于复杂了，坐标运算的优化方式上面已经有所提及，那么用户数量过大的问题有没有解决方式呢？其实也是有的，最简单的方式就是对用户所在区域进行划分，正如上面所形容的，仅仅考虑同城的情况，这样不仅用户数量能下降好几个数量级，坐标计算公式也可以简化许多，看起来只需要在用户信息表里面加入一个分区的字段就可以了。但是实际情况却远没有那么简单，首先是如果仅仅按照我国的行政区域划分，很难做到准确。举一个例子，一位用户在城市的边界上查找附近的人，这时候可能临近城市的人会比他同一个城市的人离他更近。但是如果在查找的时候要把附近的城市加入搜索范围中，那在诸如珠江三角洲、长江三角洲这样人口稠密的地方用户数量也是极其庞大的。如果不按城市划分区域，而是进一步划分到区、县、街道这种更微小的级别，又会存在人口稀疏地区在附近好几个县都查找不到用户的情况，所以如何进行分区是一个比较复杂的问题，下面提供一种我认为比较好的数据结构分区方法——“KD-Tree划分方式”，这种数据结构是一个常用于激光点云编程的工具。简单来说，他就是一种BST树的高维构成方式，通过计算每个维度数据之间的方差来达到科学划分空间的目的。在本系统中可以通过机器学习训练的方式，确定经度和纬度两个特征的权重，再依据权重划分出二维的KD-Tree空间，再把对应位置区域的用户放到合适的空间，再逐步调整树，来达到快速查找的效果。下图是一颗三维KD-TREE的数据结构图。

|  |
| --- |
| kd-tree |
| 图 4‑3 三维KD-TREE数据结构图 |

## 4.2.4地图分区查找

在考虑完了上面那种对信息划分空间进行存储的方法之后，我就开始思考，既然我能够在存储用户坐标信息的时候对用户进行划分，我也可以在查找用户信息的时候在地图上进行分区，然后对分区逐渐缩小，直到最终满足条件查找为止。既然在4.2.1之中已经证明了，在两坐标点距离不远的情况之下我们可以近似的将经纬度信息转化成平面直角坐标系来简化计算，那么这个问题也就变成了如何在平面直角坐标系之中查找与某一点距离最接近的某一数量点的问题了。很快就能想到，可以以某一点为圆心，按照初始值半径进行画圆，然后计算满足要求的点的数量，如果超出范围，则逐渐缩小圆半径的方法。并以此类推，最终找到满足数量条件的点，并再去获取这些点的坐标信息，这样就既减少了坐标运算的过程，又减少了数据库获取用户数据的操作，大大地优化整个系统的计算效率，例如以经纬度分别为(116.15,25.61)的用户USER搜索附近最近的三位用户的全过程为例，其过程图如下图4-4,4-5所示。

|  |
| --- |
|  |
| 图 4‑4 分区搜索流程1图 |

第一步，首先以用户USER为圆心，半径为16进行画圆，得到在此范围之内一共有五位用户。所以需要缩小半径。

|  |
| --- |
|  |
| 图 4‑5 分区搜索流程2图 |

第二步，首先以用户USER为圆心，缩小半径为10进行画圆，得到在此范围之内一共有三位用户。满足条件，获取三位用户的信息并返回数据。那么这种方法在实际系统之中到底是否可行，我尝试进行了编码，以下是上述流程的Go语言代码

|  |
| --- |
| const queryAllUserInfo = "select student\_id,username,lng,lat,signature from cug\_map\_users\_tpl where lng between ? and ? and lat between ? and ?;"  type User struct {  StudentId string  Username string   Password string   Longitude float64   Latitude float64  Signature string  }  func GetAllUserInfo(lng, lat float64, myId string) (userList []\*User, err error) {  number := 0  Lrange := 10.0  userList = make([]\*User, 0)  stmt2, err2 := Db.Prepare(queryAllUserInfo)  if err2 != nil {  fmt.Println("err2", err2.Error())  return  }  defer stmt2.Close()  stmt, err := Db.Prepare(countUserNumber)  if err != nil {  fmt.Println("err1", err.Error())  return  }  defer stmt.Close()  row := stmt.QueryRow(-180, 180, -90, 90)  row.Scan(&number)  for number > 50 {  row = stmt.QueryRow(lng-Lrange, lng+Lrange, lat-Lrange, lat+Lrange)  row.Scan(&number)  Lrange = Lrange / 1.2  }  Lrange = Lrange \* 1.2  var rows \*sql.Rows  if Lrange > 9 {  rows, err = stmt2.Query(-180, 180, -90, 90)  } else {  rows, err = stmt2.Query(lng-Lrange, lng+Lrange, lat-Lrange, lat+Lrange)  }  if err != nil {  fmt.Errorf(err.Error())  return  }  count := 0  defer rows.Close()  for rows.Next() {  count++  if count > 50 {  break  }  user := User{}  rows.Scan(&user.StudentId, &user.Username, &user.Longitude, &user.Latitude, &user.Signature)  if user.Longitude != 0 && user.Latitude != 0 && user.StudentId != myId {  userList = append(userList, &user)  }  }  return } |

在本系统的设计之中，首先会对全球的用户进行检索，计算出一个用户数量，如果得到的数量大于五十位，则会进行范围检索。将最初的查找范围设定为10，然后每次将范围除以1.2再重复去查询用户数量，直到查询到的用户量小于五十为止，然后将该范围内符合要求的用户全部提取出来，返回给前端进行渲染。并且在这一查询过程之中，我们可以对Mysql数据库中的经度纬度两个字段添加联合索引“position\_index”，这样在查询过程之中就可以无需遍历全部数据，而是利用索引构成的B+树进行范围查找，极大的提高了搜索的概率，达到优化数据库查询时间的目的，同时查询出来的数据无需再次进行过程复杂、耗时较长的坐标计算和排序过程，可以直接使用，可以说是从各个角度都有所优化，并且该方法也不会随着用户量级极速扩张导致不可用，反而是用户数量越大这种方法越具有其优势。

从理论上看这种方法的性能应该极好，在实际工作中表现究竟是怎么样的呢？我们将利用这种方法的服务器部署在对应的Linux服务器中，并通过在实际生产环境上谷歌浏览器内置的性能测试工具进行测试，发现利用这种方法可以将该请求的整体耗时降低至一百二十毫秒左右。相较于之前动辄数秒的请求耗时，用这种方式进行处理附近的人用户数据无疑是极为快速的，极大的提高了用户体验。图4-6是利用这种方法获取数据的在谷歌浏览器内置的性能分析图。其中红色框内的部分则是本次请求从请求到返回第一个字段，加上网络传输过程耗时的完整资源请求耗时。

|  |
| --- |
|  |
| 图 4‑6 网络请求性能分析图 |

经过实际生产环境中的验证，该方案是一种性能极高且具有可行性的方法，所以我最终在本系统中采取了这种方案。但是这种利用这种方案查找附近的点位坐标也不是没有缺陷的。首先这种方法只能获取某一用户附近大致数量的点为数据，比较难做到精确个数的数据。因为可能存在比较特殊的情况，用户的分布和圆形并不一直，这就导致了我仅仅设置了一个用户上限而不是精确值。经过实际测试发现每次返回的用户数量实际上在30-50的范围，但是我认为在该场景下，这种动态的用户数量是能够满足用户需求的。如果一个用户在最大搜寻范围内一位用户都没有的话，这种方法也会满足不了条件，但是这种情况可以在实际代码之中进行规避，目前的设置方案是遇到这种情况，会返回用户列表的前五十位用户，防止用户数据过多的情况出现。

## 4.3 聊天记录获取时机

在本系统中由于存在即时通讯的功能，所以势必要做到即时更新聊天数据，但是如果采取正常的轮询服务器获取最新消息的方式必然会给服务器造成极大的压力，如果减少数据请求次数的同时还能保证正常的即时通信成为了又一个难题。在调研过市面上的各种方案之后，我最终决定采用了懒加载的形式。所谓懒加载，就是指的是通过合理的安排资源的请求时机，来减少一次性资源请求的次数，来达到性能优化的目的。在本系统中的具体表现则是用户在发送信息的时候，并不会直接去请求服务器存储聊天信息，而是先把对应的消息渲染到本地的前端页面之上，渲染完成之后再去请求服务器来存储聊天记录。这样做的好处就是用户无需等待网络通信的时间就可以看到自己所发出的信息，让用户感受到消息是及时的。而获取最新消息的时机也需要考量，目前是放置在用户退出与单一用户的聊天，回到聊天列表的时机，这样能极大的减少服务器端的压力，又不至于让用户一直等待消息。具体前端JavaScript代码实现如下

|  |
| --- |
| ...  const [messageList, setMessageList] = useState([])  const [totalNewMessage, setTotalNewMessage] = useState(0)  async function getMyMessage() {  const res = await React.$http.get("/message/mymessage")  setMessageList(res.data.message\_list)  }  ...  const updateMessage = async () => {  if (inputMessage.trim() == "") {  alert('请输入内容')  return  }  let placeCode = 0  let message = inputMessage  if (inputMessage[0] === '@') {  const re = /^@[0-9]+@/  let res = re.exec(inputMessage)  if (res) {  const index = inputMessage.indexOf('@', 1)  placeCode = inputMessage.slice(1, index)  message = inputMessage.slice(index + 1)  }  }  const newInfo = { ...partnerInfo }  const date = new Date()  const sendTime = `${date.getFullYear()}-${String(date.getMonth() + 1).padStart(2, '0')}-${String(date.getDate()).padStart(2, '0')} ${String(date.getHours()).padStart(2, '0')}:${String(date.getMinutes()).padStart(2, '0')}:${String(date.getSeconds()).padStart(2, '0')}`  newInfo.messageList.push({ Message: message, isSendFromMe: true, SendTimeStr: sendTime, SendTime: date.getTime(), placeCode })  setPartnerInfo(newInfo)  setInputMessage('')  const form = {  receiver\_id: newInfo.senderId,  message: message,  sender\_id: studentId,  place\_code: placeCode,  }  const data = React.$qs.stringify(form)  const res = await React.$http.post("/message/leave", data)  }  return (  <>  ...  onKeyDown={e => {  if (e.key === "Enter") {  updateMessage()  }  </>)  ... |

# 5总结与展望

## 5.1总结

本文围绕着“基于WebGIS的多功能地图在线应用系统设计”进行论文的撰写。首先在绪论部分介绍了本文的研究意义和国内外研究现状，然后介绍了本文所涉及到的相关理论，其次对系统的开发环境、数据库设计、系统四层架构、功能设计等以图文并茂的方式进行了描述，最后对本系统开发过程之中的难点详细介绍了思考与解决的过程，最终总结如下：

本系统是基于高德地图提供的地图基础能力，借助于Web前端技术和服务端技术，开发了一款完整的前后端分离的Web系统。本系统的基础界面布局是参照了绝大网站的布局格式，使用上中下三个部分。上部分为导航栏，中间为高德地图结合系统二次开发的功能主题部分，下部分为符合相关要求的网站信息。接下来实现了基本的地图导航功能，包括缩放、遥感图像、移动位置、测距测面积等基础功能。最后React Hooks框架在前端对界面进行搭建，并通过自行搭建的服务器，从中存储、提取数据进行调用和渲染，来完成其附加的社交和本地生活的功能。本文主要实现了以下功能。

1. 地图基础功能。包括最基本的缩放、移动、三维模型展示、遥感图像和道路实时情况展示。
2. 账户信息控制功能。账户信息控制主要是指用户的注册、登陆、信息更改这三项基础功能，注册、登陆和修改信息都是通过服务层提供的接口来进行用户的基础功能，后续功能必须通过登陆才能使用。
3. 高级工具栏。高级功能工具栏则是一些其他地图没有的功能，例如距离测量，面积测算，常用地点快速定位以及校区导航等，该工具栏可以自由关闭或者开启，提供给有特殊需求的用户使用。
4. 社交功能。社交与本地生活则是本地图的亮点部分，可以为用户搜索用户附近的人与商铺等地物信息，搜索之后可以与他人进行聊天，社交并会保存用户的好友信息，以便用户更换位置后仍然能取得联系，同时也可以在聊天过程之中给其他用户推荐地点信息，方便一同前往，所有地点均能提供点到点的导航功能。如果进入本地生活模式，则可以自动匹配附近的地物，也可以按类型进行分类，目前有美食，娱乐设施以及基础设施三种分类，点击对应的地物可以查看他人的评论并留下您的评论或者打分，也可以与其他评论人沟通，两个模式相辅相成，共同提供地图的特色功能。

## 5.2展望

由于时间原因，本系统还存在很多不足的地方，若今后有机会，本系统将会从以下两点进行改进。

1. 性能方面。由于本系统的最终部署环境为阿里云平台提供的单核2GB内存的学生服务器，性能确实比较差。即使我已经使用了各种方式和手段尝试去做优化，但是仍然在某些方面有可以提升的地方，例如页面首帧的渲染时间和服务器的相应速度，都可以再多去思考一些优化的方式，来达到最极致的性能。
2. 功能方面。目前本系统虽然具有社交和本地生活两大特色功能，但是相对来说整体的功能还是比较少，同时这两功能也需要进一步完善，例如可以在后续加入好友关系、群聊、语音、图片聊天等功能。而在本地生活方面则可以进一步加入评论点赞，以点赞来排序、而不是仅仅像现在这样用时间排序。同时还可以加入一些风险控制的基础功能，例如防止对某一地物进行刷分等，这些都是可以进一步提升的地方。

顺序编码制文后参考文献表著录格式示例

(详见《文后参考文献著录规则(GB/T 7714－2005)》)

1．普通图书

[1]孟令奎．网络地理信息系统原理与技术[M]：科学出版社，2010.

．

[2]Yoon, Byoung-Ha; Kim, Seon-Kyu; Kim, Seon-Young. [Use of Graph Database for the Integration of Heterogeneous Biological Data](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5389944). Genomics & Informatics. March 2017, 15 (1): 19–27. [ISSN 1598-866X](https://www.worldcat.org/issn/1598-866X). [PMC 5389944](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5389944). [PMID 28416946](https://www.ncbi.nlm.nih.gov/pubmed/28416946). [doi:10.5808/GI.2017.15.1.19](https://dx.doi.org/10.5808/GI.2017.15.1.19).