论文草稿

# 摘要

最后写

关键词：公交客车；超速分析；设计；实现

[TOC]

# 第一章 绪论

## 系统开发背景

随着城市建设的飞速发展及公交系统的不断完善，作为低价，便携的出行方式之一的公交车已成为城市居民出行的主要交通工具。据统计，在2018年，仅北京的城市公共交通总客运量就共计747454.78万人次，其中地面公交客运量占比48.51%，总计362612.16万人次。且随着消费水平的增长和生活水平的提升，人们越来越遵循低碳环保出行的健康观，未来的交通模式将是新型的生态环保交通模式，公共交通将作为主要的出行方式。在新的条件下，应当对既有的城市公共交通体系设计，交通理念进行全面优化，不断提高公共交通在城市交通体系中的主导地位。

信息化技术，特别是物联网技术的发展，让公共交通管理的信息化成为必由之路。中国交通运输部在2011年公路水路交通运输信息化十二五发展规划某文书中提出，将信息化技术同相结合城市公共交通结合，能够改变我国的交通运输的发展方式，同时也可以提高交通运输管理能力和服务水平。通过现代化的信息技术对公共交通信息进行管理，能够在很大程度上提高城市交通设施利用率，对以信息化为基础的城市交通秩序进行有效的维护，对于城市交通问题的解决极为有效。

安全是公共交通的首要要求，也是公众最关注的问题。公交车人员承载量大、运行时间长，同时也存在较多的安全问题。近年来，社会舆论媒体所曝光的城市公交车事故屡见不鲜，这些事故不仅威胁到人民的生命财产安全，同时也严重影响到城市交通运输的健康发展。引起交通事故的原因主要有驾驶员主观因素、车速因素、交通条件因素、天气因素等，其中，车速不合理造成的交通事故约占三分之一，是影响道路交通安全的主要因素。

公交车的一个很大的安全隐患在于公交司机的违章驾驶，特别是超速驾驶。根据不完全的GPS统计数据显示, 大部分驾驶员约有19%的行程是超速行驶。公交车一旦超速引发事故，其造成的后果将是不可想象的。由于公交车体积大、重量重，在行车过程中，惯性都会比普通小车大。有些司机想快点结束工作交班，或是因一己私欲超速行驶，自己觉得技术老道无关紧要，但往往事故的发生就从这些疏忽大意开始。

超速行驶过程中，驾驶员遇到紧急情况的时候往往措手不及，容易造成碰撞、翻车等事故，而且由于冲击破坏力大，这些事故多为恶性事故。超速行驶也会使驾驶员视力降低，视野变窄，判断力变差，极易形成“隧道视”，特别是在高速行驶时，长时间的高速运行导致驾驶员疲劳困倦，无法看清道路边的标志物和车况，从而更加容易遭遇事故。并且超速行驶的车辆的制动距离增长，车速每增加一倍，制动距离约增加四倍，特别是在重载情况下或在湿滑路面上，制动距离更长，一旦前车突然减速，便极易造成追尾事故。可见车速的快慢对事故发生的可能性及其严重性有着直接影响。

本系统力图利用信息化技术监控公交车运行位置和速度信息，分析发生超速情况的高频时间及路段以便相关人员进行进一步措施，以保障公共交通安全，提升城市公共交通运输能力，并提升信息管理的自动化水平，缩减信息管理成本。

## 系统开发目的及意义

系统的总体目标是初步建立一套基于车载物联网数据的公交客车超速分析系统。该系统应当能够获取海量的公交车载信息，并通过数据库存储这些车辆的车载数据。通过地图匹配功能，系统可以提供美观易用的前端界面，其能够准确地显示车辆位置信息在地图上，并能够显示当前的实时超速情况，以及统计分析历史时间公交车超速位置以及车速规律。系统同时应当提供对车辆、线路、站点等信息做数据管理操作的功能。本课题的研究对提升城市公共交通运输能力，保障城市交通安全和人民群众生命财产安全有着非常重要的实际意义。

## 国内外研究现状

TODO: 这个真的有必要写吗？还是有必要的，主要专注于地理信息的管理（GIS，GeoJSON……），相关可视化框架，对车辆超速情况的研究……

## 研究内容

TODO：也最后写

## 论文结构

TODO:最后写

# 第二章 系统相关技术

本系统使用前后端分离架构，涉及到多种编程语言，编程范式，开发工具，开发框架。下面对系统所使用的开发框架，编程语言，开发工具进行简单介绍。

## 开发框架

系统后端主要使用SpringSSM框架，即Spring，SpringMVC和Mybatis框架，前端主要使用Vue.js和Element-Plus框架，使用AntV-l7框架实现地图以及超速等信息的的可视化，使用Echarts框架进行统计信息的展示。下面对这些框架进行一些简要介绍。

### Spring Boot框架

Spring框架是一个用来进行企业全栈开发的轻量级框架，其业务范围涵盖云计算，大数据，数据持久化等领域，是JavaEE的轻量级替代品，使用户无需开发重量级的Enterprise JavaBean（EJB），为企业级Java开发提供了相对简单的方法，通过依赖注入（DI），面向切面编程（AOP），使简单的Java对象（POJO, plain old java object）便可实现EJB的功能。Spring Boot则是基于Spring的框架，它提供了强大的自动配置功能，用户只需给定起步依赖，Spring Boot便将能引入需要的库并自动提供相关配置，为系统的开发提供很大的便利性。在编写组件时，只需要对组件类使用Java注解即可让Spring Boot进行配置。

### Spring MVC框架

SpringMVC是Spring提供的展现层框架，用于响应前端的请求。MVC分别代表Model（模型）、View（视图）、Controller（控制器），其中视图代表用户所看到的界面，模型代表驱动视图的数据，控制器是对视图进行管理并同时连接模型和视图的功能模块（TODO：这个描述可能不严谨）。MVC模式是web应用程序的一种典型架构，它将应用按照分层的思想进行组织，其目的是将数据与界面分离以更好地组织和解耦代码。SpringMVC通过一套MVC注解，让POJO即可成为处理请求的控制器，而无需实现任何接口。SpringMVC也提供了对Websocket以及基于Websocket的SocketJS等技术的支持。本系统使用更加现代的REST风格的SpringMVC，即使用前后端分离架构，后端系统不负责对视图进行维护，而是仅通过控制器同前端进行交互并维护模型层。REST风格要求前端仅通过ajax与系统通信，系统将通过Json格式的数据响应前端的请求。视图层由前端框架Vue.js维护。

### Mybatis框架

Mybatis是基于Java的持久层框架，它是后端同数据库间的中间件。Mybatis支持使用XML或Java注解来替代原本编写JDBC代码和手动获取结果集的既有方式，免除了处理加载数据库驱动等过程，且将数据库中的记录映射到POJO对象以抽象对结果的获取和与数据库的交互，从而实现了业务代码和SQL代码的解耦合，方便了系统的开发和模块化。本系统使用Mybatis构建DAO层以与数据库进行交互，为数据库中每个表编写对应该表的实体和执行相关数据库操作的Mapper。

### Vue.js框架

Vue.js是前端的视图层框架，它是一个组件化的，自底向上增量开发的，渐进式的框架。其特点在于容易上手且功能强大，容易与现有项目相结合。Vue采用MVVM模式，它实现了视图和模型的双向绑定功能，使开发者不需要手动设置数据监听和更新视图，从而让开发者完全摆脱操作DOM和维护视图的烦扰。Vue提供了强大的组件化和单文件组件功能，用户可以在单个文件中编写某个组件的模板，逻辑和样式，即将某组件相关的css，html，js/ts代码组织在同一个文件中，且可轻易引入外部css样式等资源文件，这让对组件和代码的复用，让前端项目的系统化，工程化变得易如反掌。Vue官方提供的路由等高级功能更是让Vue成为一个灵活且功能强大的前端框架。本系统使用最新的Vue3.X，其使用Typescript编写，拥有更加友好的类型推断和类型安全功能，Vue3.X新提供的setup生命周期和组合式API颠覆了原有的对象属性式的代码布局方式，使开发者能够分离逻辑关注点到不同函数，文件中，将代码按照功能进行切分以进行解耦以提高维护性和可重用性，并让开发者能够对数据绑定进行更加精细的控制从而获得更大的性能优化空间和自由度。

### AntV-L7框架

AntV是蚂蚁金服所提供的一套的可视化图表库，其应用范围涉及数据可视化的各个领域。其中AntV-L7框架专注地理信息的可视化。L7框架具有使用灵活，性能强大，渲染精美的特点。本系统使用L7框架完成地图，公交线路，车辆位置，超速等信息展示的功能。

### Echarts框架

Echarts是由百度团队编写的一个开源可视化库，它实现简单，设计理念为向开发者提供成品图表而非图形，提供了丰富多样的图表种类以供直接使用，而无需自己定义新的图表种类。本系统使用Echarts框架展示超速信息的各种统计规律，如各时间段超速情况，多日超速情况，各路段超速情况等。

### Element-Plus框架

Element-Plus是适用于Vue3.X的UI框架，其易学易用，界面美观大方，更新积极。本系统使用Element-Plus搭建前端项目界面。

## 开发语言

本系统涉及到多种编程语言，其中后端系统主要使用Kotlin语言，同时也利用Python和Typescript语言进行部分模块的编写。前端主要使用Typescript语言。下面对这些编程语言进行简要介绍。

### Kotlin

Kotlin是一种针对Java平台的，致力于更简洁，高效，安全地替代Java的新编程语言，它在Java语言的基础上提供了更为强大的空指针检查，函数式特性，类型推断，运算符重载，DSL等功能，并且能够无缝地与Java代码、框架进行互操作，并因此可以在任何可以使用Java的地方使用。Kotlin原生支持很多设计模式，如单例模式，建造者模式，委托等。Kotlin所提供的协程为并发编程提供了新的可能性。举例来说，Kotlin具备如下优点——

1. 静态类型和类型推断 Kotlin同Java一样是静态类型的语言，这意味着编译器能够在编译期就确定所有表达式的类型，验证对象是否包含想要访问的方法或域，从而让编译器就能够验证程序的正确性，减少程序出现bug崩溃的风险，静态类型也提高了程序的可维护性，让IDE能够提供更加准确强大的语法提示和代码补全功能。与Java不同的是，Kotlin并不需要显式声明没一个变量的类型，取而代之的是，在很多条件下，变量类型可以根据上下文自动判断出来，从而减轻开发者的编码负担。
2. 多范式编程 Kotlin可以结合使用函数式编程和面向对象编程范式。Java是纯粹的面向对象语言，它的编程范式是受限的，虽然Java8通过对Java原有特性如函数式接口，匿名内部类等进行一定地包装来提供了部分函数式特性，但这依然没有减轻使用Java进行函数式编程的复杂程度。而Kotlin在继承了Java语言丰富的面向对象编程特性的基础上提供了非常丰富的函数式特性。其中包括但不限于——Kotlin语言中的函数是“头等公民”，这意味着它可以被当做值使用，当做变量保存，当做参数传递，当做其它参数的返回值，就像普通的变量一样；推荐使用纯函数，即无副作用的函数，对同样的输入，它必定产生同样的结果，它不修改对象的状态，也不对外界进行任何交互；推荐使用不可变变量。同时kotlin为函数参数提供了非常友好的语法——当最后一个参数是函数时，调用时若通过lambda表达式给定参数，则lambda表达式可写在函数调用的括号外。该功能能够大大提升回调代码的可读性，且结合Kotlin提供的let，apply等方法，能够将所有嵌套调用转化为链式调用，如对f(g(h(x)), y)，使用Kotlin代码可以表述为x.let(h).let{g(it)}.let{f(it, y)}，方便了代码的编写和维护。

* 函数式编程的优势在于其更加抽象，简练，容易阅读，且对并发编程友好。一般来说函数式的代码是声明式的，相较于命令式的代码，它更倾向于要求计算机“去做什么”而不是“如何做”。开发者不必关注实现的细节，只需要将自己的需求通过函数抽象，作为值提供给语言提供的丰富的函数式库即可。并且，函数式编程是多线程安全的，因为函数式编程要求使用不可变数据结构和纯函数，保证线程不安全的修改根本不会发生，从而不需要设计复杂的同步方案。

1. 和Java完全兼容 Kotlin同Java代码是完全兼容的，甚至集合框架都同Java使用的是同样的，Java代码和Kotlin代码可以轻易转换。Kotlin也可以利用Java上既有的框架，无论是继承Java类或是利用Java的注解，Kotlin都不会遇到任何问题。这也让本系统结合使用Kotlin和Spring Boot框架具有现实性和可行性。并且Kotlin提供了很多语法糖，能够减少代码量。比如data class能够自动生成构造器和各个域的getter和setter，使编写对应数据库表中实体对应POJO时十分方便。

本系统使用Kotlin语言和Spring Boot框架编写后端。

### Python

Python是一种面向对象的，解释型的，动态类型的高级编程语言。Python因其开源，跨平台，语法简洁强大，类似自然语言，自二十世纪九十年代初Guido van Rossum创造这门语言起已经越来越广泛地被应用。Python常被用以向初学者介绍编程，多个Linux发行版使用Python完成系统管理任务，Krita，Blender等知名应用使用Python作为脚本语言。Python也因其功能强大的标准库和极其丰富的第三方库从而能够被应用在各种领域，包括但不限于网络爬虫，机器学习，图像处理，游戏制作等。

出于Python功能强大且容易编写，在小型项目的快速开发上有着非常强大的优势，本系统使用Python进行公交车模拟模块的编写。

### Typescript

Typescript语言是微软开发的一门编程语言，它是Javascript语言的超集，能够编译成在任何浏览器和Nodejs上运行的Javascript代码，并致力于修正Javascript中所存在的种种问题，如隐式类型转换，函数传参等，Typescript扩展了Javascript的语法，为Javascript添加了静态类型，编译期类型检查，类型推断，泛型编程，基于类和接口的面向对象等功能，从而帮助开发者编写更加类型安全，更加具有健壮性的代码，并能够为IDE提供更加强大的语法提示和代码补全。

为利用Typescript强大的类型功能以帮助编码，本系统使用Typescript作为应用Vue的语言编写前端页面，并结合Nodejs编写后端的部分模块。同时也大量使用lodash库以辅助编写，这是一个js的工具库，提供了大量的工具函数以方便对js/ts中的对象，数组等数据结构的使用，同时也提供了对对象进行包装以进行链式调用的功能。

## 系统开发工具

本系统的开发涉及到多种开发工具，下面对这些开发工具进行简要介绍。

### MySQL数据库

MySQL是最实用的关系型数据库管理系统之一。其因体积小，安装方便，开源易用，在Web开发领域被广泛应用。本系统使用最新的MySQL8.0作为数据库系统。

### IntelliJ IDEA

IntelliJ IDEA是最好的Java语言的集成开发环境（IDE）之一，它提供了强大的代码提示和分析功能，并能够整合J2EE和Spring Boot框架，同时对除Java以外的其它JVM平台语言如Kotlin，Scala，Groovy，Clojure等语言提供了良好的支持。本系统使用IDEA进行后端系统项目的编写。

### Vite

Vite是由Vue框架的作者尤雨溪所开发的Web开发构建工具，同经典的Webpack等打包工具相比，Vite具有冷启动，热更新的特性，且自带开发所需的大部分插件，从而使开发者基本无需任何配置便可开始进行开发。Vite的缺点是其仅支持Vue 3.x项目。本系统使用Vite构建前端项目。

### Node.js

Nodejs是一个基于ChromeV8引擎的Javascript运行时环境，它的底层同ChromeV8一样使用C++开发的，可以认为是运行在服务端上的Javascript环境。Nodejs框架令使用Javascript进行后端或全栈开发成为可能。Nodejs同样可以结合Typescript使用。本系统使用Nodejs和Typescript编写数据生成模块和公交车模拟模块中的部分功能。同时Vite工具也是运行在Nodejs平台上的。

## 本章小结

本章介绍了系统所使用的编程框架，编程语言，开发工具。根据本系统实际之需求，系统后端主要使用Spring Boot框架和Kotlin语言，并使用Python，Typescript语言进行部分模块开发，使用持久层框架Mybatis与数据库进行交互，数据库使用MySQL。系统前端使用Vue.js框架和Typescript语言开发。

# 第三章 需求分析

本章内容为依据用户的实际需要对系统进行需求分析，根据用户需求详细分析系统所需功能，模块，数据等信息。

## 需求概述

用户的需求是整个系统开发的基础，系统的一切抽象设计和具体实现根本上取决于用户的需求。因此，对用户需求的具体分析是必要的。本系统的重点在于对公交客车的运行数据中超速信息进行分析，其中假定用户的需求主要集中于对某个地区的公交信息进行查询。具体说来可以归结为如下需求——

1. 能够实时接收、显示车辆的位置，速度信息，并存储到数据库中。
2. 能够让信息管理者不需要进行编程操作即对车辆信息、线路信息、站点详情进行数据管理。包括对车辆信息、线路信息、站点详情等信息进行增删改查的数据管理操作。
3. 能够对公交车超速信息进行统计分析并以各种形式进行可视化展示。

本系统只有两种角色——用户和信息管理员。他们的权限具体来说如表TODO所示——

|  |  |  |
| --- | --- | --- |
| 序号 | 角色 | 权限 |
| 1 | 用户 | 查看某地区实时公交车，线路，站点等信息查看某地区实时超速信息的统计分析数据查看某地区历史公交车，线路，站点等信息查看某地区历史超速信息的统计分析数据 |
| 2 | 信息管理员 | 登录对公交车，公交线路，站点等信息进行管理修改自己的信息 |

下面就上表对两种用户角色进行介绍。

1. 用户 用户为系统的主要服务对象，对应于公交车超速信息的信息分析者。用户不需要登录，能够在本系统中进行实时和历史的公交车运行信息，线路信息，站点信息进行查询，用户也能够查看公交车的实时或历史超速信息以及对其统计分析后的信息。用户的用例图如图所示：TODO
2. 信息管理员 信息管理员对应公交系统中对公交信息进行维护的人员。信息管理员需要登录，有权力对公交车，线路，站点信息进行增删改查操作，且能够修改的名称，密码信息。信息管理员的用例图如图所示——TODO

## 功能需求

通过上面的需求分析，能够确定本文将设计实现的公交客车超速分析系统的主要功能将包括公交信息的实时录入，实时展示，统计分析等业务功能和信息管理功能。鉴于这些功能要求，以及无法获取到真实公交车线路和历史、实时运行信息的客观事实，本系统纵向切分应当包括如下子模块：实时信息模块、历史信息模块、信息管理模块、公交数据生成模块、公交车模拟模块。系统的整体功能模块设计如下图所示：TODO

### 超速信息获取模块

超速信息获取模块应当能够实时接受公交车所上传的速度位置信息，将其存储到数据库中并同当前所处路段限速进行分析已确认是否超速，如果超速，则将该信息储存到数据库中。TODO：同时，该模块每一段时间后对会对该段时间内获取的各路段的超速信息进行聚集和分析。该模块同时也要提供信息给实时信息模块以让用户获取相关信息。超速信息获取模块的功能模块图如下图所示：TODO：相较于实时信息模块，该模块偏重于从公交车拿到数据并分析

### 实时信息模块

关于实时信息，用户应当能够获取当前某地区的公交线路，站点等信息且进行展示。通过超速信息获取模块，用户应当能够获取当前在线的公交车的实时信息，包括当前以及之前一段时间的位置速度信息，并可视化地展示在地图上。该地区实时以及最近一段时间的超速信息也应当能够被获取，且以各种方式，如热力图，按速度染色的轨迹等可视化地展示出来。实时信息模块的功能模块图如下图所示：TODO

### 历史信息模块

关于历史信息，用户应当能够获取某地区某历史日期的公交线路，站点等信息并展示。用户也应当能够获取该日任何时刻的公交车位置速度信息。用户应当能够查询某特定时间段的超速信息，单日或多日中各时间段的超速信息的统计分析信息，单日或多日中特定时间段超速信息的统计分析信息等。历史信息模块的功能模块图如下图所示：TODO

### 信息管理模块

信息管理模块主要包括对公交车，线路，站点等信息进行增删改操作，以及信息管理员的登录操作和其他操作。其功能模块图如下图所示：TODO

### 公交数据生成模块

公交数据生成模块应当利用某具体地区的路网或其他形式地理数据随机生成特定数量的公交车，线路，站点，限速信息，并以此信息对数据库信息进行初始化。公交数据生成模块的功能模块图如下图所示：TODO

### 公交车模拟模块

公交车模拟模块应当获取数据库中公交车及线路的信息，模拟一定数量的公交车的运行以对系统进行测试。公交车模拟模块的功能模块图如下图所示：TODO

## 本章小结

本章对公交车超速分析系统的需求进行了具体分析研究，从系统实现之目标出发，划分用户角色为用户和管理者，并详细分析了系统中应有的各种模块以及其应当提供的功能。

# 第四章 概要设计

前文分析了系统需求，确定了系统应有功能，本章就系统的架构，各模块组织，数据库设计等进行分析。

## 系统架构

本文公交车超速分析系统总体上采用MVC分层架构。MVC架构是流行于Web应用开发的典型架构，其通过分层的思想组织代码，从而使各模块代码符合软件工程中一般要求之“高内聚，低耦合”的思想，以提高系统的扩展性，重用性，可维护性。Spring Boot框架为MVC架构提供了非常成熟的解决方案，也是本系统采用MVC架构的原因之一。同时本系统使用更加现代的前后端分离架构，将前后端进行更彻底的解耦以提升扩展性和通用性。

在MVC的分层模式下，本系统从逻辑以及代码组织上具体来说可以大致划分为四层：展现层，控制器层，业务层，持久层。下面对这四层进行简单说明：

1. 展现层 展现层即为通常所称的前端，主要负责界面的展示，同时提供用户对系统进行操作的接口。本系统的展现层通过Vue.js框架+css+Typescript(ts)编写，并最终被编译成html+css+Javascript并运行在浏览器环境上。其中Vue.js框架提供通过组件树形式组织html页面的功能，使本系统能够将前端的实现拆分为一个个组件及其子组件之实现，从而提高系统的可维护性和扩展性。同时Typescript在Javascript的基础上能够提供更多帮助编码和提升健壮性的功能。
2. 控制器层 控制器层是系统后端与前端进行交互的接口，主要负责响应用户特定请求并通过业务层执行相关业务操作和向用户返回数据。传统MVC架构中控制器层同时要负责前端中视图跳转，生成结果页面等功能，而本系统使用前后端分离架构和Rest风格的控制器层，仅通过ajax响应用户请求，不干涉前端的事务。同时，关于接受公交车实时信息和向前端发送实时信息的模块也属于控制器层。
3. 业务层 业务层是系统的核心，主要负责封装各种具体业务操作并暴露给控制器层，并通过持久层操作数据库。以实现用户对系统中数据的操作。
4. 持久层 持久层抽象系统同数据库间的交互，根据用户请求和业务逻辑对持久化存储介质即数据库进行增删改查操作。本系统使用Mybatis框架实现数据持久层。

其中，展现层同控制器层间有一个所谓的过滤器层，它负责筛选合法的请求并转发给控制器。系统后端的总体架构图如图TODO。（那个很简单的）

## 数据库设计

数据库设计对大型应用程序的设计是及其重要的，它决定了对数据的有效访问和存储。考虑到系统需要以单日为粒度查询历史的线路，站点，公交车等信息，本系统应当对数据库进行特殊的设计以支持此功能。

关于历史信息的存储，有三种常用的数据库设计方法：全量表，增量表和拉链表。其中全量表容易设计，但是每日无论数据有无变化都要进行一次全量的保存，对存储空间有严重的浪费；增量表难以设计，应用场景不符合本系统需要，难以保存对数据的删除操作，在本系统的应用环境中性能低下且在多表联合查询时十分麻烦；拉链表对存储空间的浪费少且容易编写。因此拉链表最适合本系统的需要。

所谓拉链表，就是对每一个表中实体都设定该数据的有效时间区间，前闭后闭。数据刚插入时，令数据的起始时间为当前时间，终止时间为一个足够远的值，比如2999年12月31日。对于数据的更新操作，则是将该数据的终止时间设为昨日，同时插入更新后的数据；对于删除操作，则更改数据的终止时间为昨日即可。对拉链表进行查询时，筛选查询日期在有效区间中的数据便可查询该日期的历史数据。

数据库的设计应当始于对系统所涉及信息进行特征之抽象，并据此设计数据库中各表结构，关系等。本系统所涉及的数据实体种类较为单纯，以下列出所有实体及其特征属性：

1. 管理员：管理员ID，名称，密码。
2. 地点：区域代码，名称，经度，纬度，起始时间，终止时间。
3. 公交车：公交车ID，车牌号，车型，起始时间，终止时间。
4. 道路节点：节点ID，经度，纬度，是否站点，区域代码，起始时间，终止时间。
5. 道路路径：路径ID，节点1ID，节点2ID，限速，区域代码，路径名称，起始时间，终止时间。
6. 公交车线路：线路ID，线路名称，起始节点，公交车列表，路径列表。
7. 位置信息：公交车ID，速度，经度，纬度，上传时间。
8. 超速信息：公交车ID，速度，道路路径ID，区域代码，经度，纬度，上传时间。

关于上述实体，有一些要点需要被注意：关于起始时间和终止时间，拥有这些特征的实体是有必要进行历史信息查询，因此是使用拉链表技术构建的；关于公交车线路实体中公交车列表和路径列表，考虑到公交车线路中的路径信息是顺序的，因此在关系型数据库中的存储较为麻烦，这里使用了MySQL所提供的JSON数据类型以存储，公交车列表也使用JSON数据格式，但这里是为了快速查询的需要；关于多个实体中存储冗余的区域代码，这是考虑用户通过地点来查询信息是一个主要的需求，因此这里存储冗余信息便于快速查询。本系统的数据库设计不满足任何数据库范式而是使用反范式设计，试图在性能和冗余程度上取得平衡。

各实体间的关系如下面的ER图图XX所示，考虑到起始时间和终止时间严格来说并不属于实体的属性，在ER图中并未展示这两个属性。

TODO：ER图

## 本章小结

本章确认了系统总体架构，系统使用B/S架构，MVC模式，将系统分为前端和后端，明确了系统的结构层次，然后分析了各功能模块在系统中的层级位置。然后根据系统所涉及数据信息抽象出数据实体及各实体间相互关系，并据此同系统需要进行数据库设计。

# 第五章 系统具体设计与实现

本章主要任务为介绍系统的具体设计与实现，其中包括对系统开发运行环境和系统具体架构的介绍，数据库中各表具体设计以及重要模块和功能，前端页面的设计和实现。

## 系统开发环境

本系统使用的开发环境如下表所示。

|  |  |
| --- | --- |
| 操作系统 | macOS Big Sur |
| 开发工具 | IntelliJ IDEA、VSCode |
| 数据库 | MySQL |
| 服务器 | Tomcat |
| 浏览器 | Firefox |
| 运行环境 | Nodejs、Python3、JVM |

## 系统具体架构

本公交车超速分析系统主要使用Spring Boot和Vue3.0框架，使用Kotlin语言进行开发。关于系统的后端实现架构，本系统主要分为四个层次：Filter层，Controller层，Service层，以及基于Mybatis编写的Mapper组成的DAO层。系统的具体架构图如下图所示。TODO：具体架构图，按层次分级（估计是这里最复杂的图了）

其中，Filter层主要拦截用户不合法的请求，如进行仅管理员才可进行的操作比如对数据的增删改查，从而保证系统的数据安全。通过Filter层过滤的请求将被送往Controller层，Controller层为系统所提供的接口，其将调用相关的Service，获取结果并响应给请求者即前端。Service层封装具体业务代码，其通过DAO层提供之接口操作数据库。DAO层通过Mybatis框架进行编写，负责与数据库进行交互，同时将数据库中实体映射为POJO对象。

关于系统的前端，前端页面使用SPA（单页面应用）。所谓SPA，就是只有一个页面的应用，页面的切换通过使用JS监控URL哈希的改变并修改DOM来实现。vue提供了强大的路由功能vue-router，对这一模式进行了良好的封装。SPA的优势在于其渲染都是局部的，页面的跳转不需要向服务器重新请求资源，同时让前后端更好分离。其缺点在于第一次加载时需要加载所有资源，可能速度较慢，且对SEO（搜索引擎优化）支持不好。

## 数据库设计

数据库中的表主要分为表征实体的实体表以及表征实体间关系的关系表。下面对数据库中较为重要的实体表的设计进行介绍。出于拉链表的性质，即会有多行数据对应一个实体，拉链表的实体表的ID不能设置为自增。

1. 公交线路表，表名为busline，表结构如表XX所示。该表保存公交车线路的各种信息。其中公交车列表和路径列表为JSON数组。因MySQL未提供确保JSON格式中数据约束，需要后端对数据完整性进行维护。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * 属性 | * 字段 | * 类型 | * 长度 | * 约束 |
| * 线路ID | * line\_id | * INT | * — | * 主键 |
| * 线路名称 | * line\_name | * VARCHAR | * 20 | * 非空 |
| * 起始节点ID | * start\_node\_id | * INT | * — | * 外键 |
| * 公交车列表 | * bus\_id\_list | * JSON | * — | * 非空 |
| * 路径列表 | * nodepath\_id\_list | * JSON | * — | * 非空 |
| * 起始时间 | * start\_time | * DATE | * — | * 主键、默认值为当日 |
| * 终止时间 | * end\_time | * DATE | * — | * 主键、默认值为2999-12-31 |

1. 路径表，表名为nodepath，保存公交车线路中各个路段的信息。表结构如表XX所示。在以路径ID和终止时间为主键的前提下，对起始节点ID和终止节点ID施加唯一约束。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * 属性 | * 字段 | * 类型 | * 长度 | * 约束 |
| * 路径ID | * nodepath\_id | * INT | * — | * 主键 |
| * 起始节点ID | * node1\_id | * INT | * — | * 唯一、外键 |
| * 终止节点ID | * node2\_id | * INT | * — | * 唯一、外键 |
| * 限速 | * speed\_limit | * TINYINT | * — | * 非空 |
| * 区域代码 | * adcode | * INT | * — | * 外键、索引 |
| * 路段名称 | * street\_name | * VARCHAR | * 20 | * 非空 |
| * 起始时间 | * start\_time | * DATE | * — | * 主键、默认值为当日 |
| * 终止时间 | * end\_time | * DATE | * — | * 主键、默认值为2999-12-31 |

1. 节点表，表名为node，代表路径上一个具体的位置。表结构如表XX所示。其中为经纬度设置唯一约束。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * 属性 | * 字段 | * 类型 | * 长度 | * 约束 |
| * 节点ID | * node\_id | * INT | * — | * 主键 |
| * 经度 | * lat | * DECIMAL | * (9,6) | * 唯一、非空 |
| * 纬度 | * lont | * DECIMAL | * (9,6) | * 唯一、非空 |
| * 是否站点 | * isStation | * BIT | * — | * 非空 |
| * 区域代码 | * adcode | * INT | * — | * 外键、索引 |
| * 起始时间 | * start\_time | * DATE | * — | * 主键、默认值为当日 |
| * 终止时间 | * end\_time | * DATE | * — | * 主键、默认值为2999-12-31 |

1. 超速表，表名为overspeedPos，代表一个超速信息，表结构如表XX所示。TODO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * 属性 | * 字段 | * 类型 | * 长度 | * 约束 |
| * 公交车ID | * bus\_id | * INT | * — | * 主键 |
| * 速度 | * speed | * TINYINT | * — | * 非空 |
| * 路段ID | * nodepath\_id | * INT | * — | * 外键、索引 |
| * 区域代码 | * adcode | * INT | * — | * 外键、索引 |
| * 经度 | * lat | * DECIMAL | * (9,6) | * 非空 |
| * 纬度 | * lont | * DECIMAL | * (9,6) | * 非空 |
| * 上传时间 | * update\_time | * TIMESTAMP | * — | * 主键、默认值为当前时间戳 |

### 拉链表的操作的实现

出于拉链表的独特性质，对拉链表的操作与对普通表的操作不同，因此不能直接应用SQL中的UPDATE，DELETE等语句来操作拉链表。下面就节点表为例子介绍对拉链表的查询，更新，删除操作的具体实现。

关于查询操作，应当要求查询时间在起始时间和有效时间内，这里可以使用SQL提供的BETWEEN-AND语法。举例来说，查询当前所有公交车站点节点的SQL代码如下：

SELECT \* FROM node WHERE isStation AND CURRENT\_DATE() BETWEEN start\_time AND end\_time;

更新操作分为三步：删除当日的更新，将原数据终止时间设为昨日，插入新数据。要删除当日的更新的原因是本系统使用的拉链表的粒度是日，如果当日对某数据有多次更新，应当取最后一次更新。对应该操作的存储过程（procedure）如下：

CREATE PROCEDURE updateNode (node\_id\_ INT, lat\_ DECIMAL(9, 6), lont\_ DECIMAL(9,6), isStation\_ BIT, adcode\_ INT)  
BEGIN  
 -- 删除可能存在的当日的更新  
 DELETE FROM node WHERE   
 node\_id = node\_id\_   
 AND start\_time = CURRENT\_DATE()   
 AND end\_time = "2999-12-31";  
  
 -- 更改原数据终止时间  
 UPDATE node   
 SET end\_time = DATE\_ADD(CURRENT\_DATE(), INTERVAL -1 day)   
 WHERE node\_id = node\_id\_ AND end\_time = "2999-12-31";  
  
 -- 插入新数据，start\_time和end\_time是自动生成的  
 INSERT INTO node(node\_id, lat, lont, isStation, adcode)  
 VALUES (node\_id\_, lat\_, lont\_, isStation\_, adcode\_);  
END;

删除操作是简单的，将当前数据的结束时间设为当日的前一日即可，其实现与更新操作中第二步相同，此处不再赘述。

## 数据生成模块的实现

数据生成模块的意义在于在无法获取真实公交线路信息的情况下，能够自动生成公交车，公交线路，节点等数据以模拟真实业务环境，测试系统超速分析等业务功能。该模块逻辑上属于后端，使用Typescript编写。模块的主要任务为获取原始线路数据、根据该数据生成节点，路径数据和给定数量的公交车，线路数据、生成最终SQL语句并执行。其中各路径的限速随机给定。

本模块所使用的原始数据来自echarts官网中[北京公交车线路示例](https://cdn.jsdelivr.net/gh/apache/echarts-website@asf-site/examples/data/asset/data/lines-bus.json)（TODO应当插到附录里）中所使用之数据，经转码后其形式为JSON格式的二维数组，其每一个子数组代表一条线路，子数组中各元素为字符串，以半角逗号分隔，代表线路上各点的经纬度坐标。数据中总共有1543条线路，40950个不同的节点。模块的任务流程具体如下：

1. 执行初始化表的sql文件，创建系统所需要的表。
2. 生成地点信息。本模块生成的信息的区域为北京市东城区，其区域代码为110101，中心经纬度假定为(116.397128, 39.916527)，生成对应sql语句。
3. 生成节点信息。具体实现为将二维数组的线路信息展平，唯一化后映射生成所需信息，并设置十分之一的节点为站点。其中，需要维护节点位置同节点ID的对应关系，以在生成路径信息时使用。相关实现代码如下：

* // paths为原始数据的二维数组   
  let poses = \_(paths).flatMap().uniq().value(); // 将线路信息展平，唯一化  
  let ID = 1; // 作为ID的标识，SQL中自增的ID是从1开始的  
  let pos2nodeId: Record<string, number> = {} // 位置（使用半角逗号分隔的经纬度字符串）与节点ID对应的表  
  let stationSet = new Set<string>(\_(poses).shuffle().slice(0, Math.floor(poses.length / 10)).value()); // 标识为站点的节点集合的set  
  let insertNodeSql = // 插入节点数据的SQL  
   `  
  insert into node(lat, lont, isStation, adcode)  
  values   
  ${poses.map(pos => {  
   pos2nodeId[pos] = ID++;  
   return `(${pos}, ${stationSet.has(pos) ? 1 : 0}, 110101)` // 此时的ID为i  
  }).join(',\n')};  
  `;

1. 生成路径信息。具体实现为对每一条线路，利用上一步骤中生成的节点位置同节点ID对应表，将其中节点两两结合创建路径信息。同时维护一个节点对对应路径ID的表，以防止重复插入，以及在生成线路信息中使用。具体实现代码如下：

* // 初始化路径信息  
  let coord2nodepathId: Record<string, number> = {} // 节点对对应nodepathId的表  
  let pairs = paths.map(path => \_.zip(\_.dropRight(path), \_.drop(path))).flat() // 获取所有节点对，这里的dropRight方法获取除尾部的元素，drop方法获取除首部的元素，zip方法将两数组元素一一对应组成新数组，将其展平则为所有节点对  
  ID = 1;  
  let insertNodepathSql = `  
  insert into nodepath  
  (node1\_id,node2\_id,speed\_limit,adcode,street\_name)  
  values  
  ${\_.compact(pairs.map(v => { // compact方法的作用是删除数组中的null，false，undefined  
   // 如果已经存在，则不再重新添加  
   if (coord2nodepathId[`${v[0]}-${v[1]}`])  
   return null;  
   coord2nodepathId[`${v[0]}-${v[1]}`] = ID++;  
   return `(  
  ${pos2nodeId[v[0]]},  
  ${pos2nodeId[v[1]]},  
  ${\_.random(20, 120, false) /\* 速度为20到120 \*/},   
  110101,  
  ${\_.sample(streetNames) /\* 获取随机名称 \*/}  
  )`  
  })).join(',\n')};  
  `

1. 生成其他信息并执行所有SQL语句。接下来的步骤是生成公交车信息，以及线路，公交车同地点对应的信息并执行SQL语句以插入数据到数据库。这里应用前面给定的线路和各线路公交车数量来生成数据。具体实现比较简单，不再赘述。

该模块的流程图如图所示。TODO。

## 公交车模拟模块的实现

公交车模拟模块的意义是模拟真实公交车的运行，从而使系统能够获取实时信息，超速信息以进行分析。本模块使用Typescript和Python语言实现，其中Typescript通过前端所使用的接口向后端发送请求获取所需公交信息并保存，再由Python读取这些信息，模拟一定数量公交车的运行，对每一辆公交车都通过后端中原本假定公交车使用的接口建立websocket连接并间隔1秒发送模拟的信息。本模块的流程图如图。TODO

关于模拟公交车运行功能的编写，本系统利用了闭包，这是现代编程语言大多都会提供的一个语法特性，使函数能够捕获外部作用域中的变量到自己的作用域中，此时该变量同函数的关系可以类比全局变量同函数的关系。面向对象语言如Java可以通过对象实现类似闭包的功能。模块首先根据生成的数据确定每一辆公交车所对应的节点位置序列，是否是站点，各节点之间的限速等信息。然后对每一辆公交车，其将生成运行的初始数据捕获到一个内部函数并返回它，该内部函数利用当前所持有数据计算下一时刻的数据并返回要上传给后端的信息。websocket客户端将间隔时间调用该函数并发送其返回值到后端。

关于计算下一时刻的数据的功能，其假定公交车可能处于三种状态：正常、减速、捣乱，同时定义一个状态的倒计时，为0时切换状态。公交车有99%的可能切换到正常状态，在正常状态下，要求公交车速度最大不超过当前路段限速减去5千米每小时，最小不低于路段限速减去20千米每小时，若速度在这区间之外则将速度向该区间靠近，否则速度随机增减。正常状态倒计时设为30；减速状态在下一个节点是站点且距离小于100米时触发，其会让速度维持在0到15千米每小时，减速状态倒计时设为0，以保证到达站点后正确切换状态；捣乱状态有1%的概率触发，倒计时为5，其对公交车的速度没有任何限制，会在0到120千米每小时这个区间内随机更改速度。

该功能将首先判断公交车是否到了终点，如果到了终点则从起点重新开始，否则计算下一时刻的位置以及距离下一个节点的距离，然后更改当前状态和速度。若距离小于20米则判定它到达了该节点，此时设定当前位置为该节点位置，目标为下一个节点位置，其他属性不变。具体流程图如下TODO。

关于计算下一时刻的位置的算法，本模块使用了比较粗糙的实现。已知数据有公交车的速度，单位为千米每小时，当前的经纬度和下一个节点的经纬度。先求出当前位置同下一个节点的距离fullDist，单位为米，根据当前速度求出一秒内经过的距离dist，单位为米。作当前位置到下一个节点位置的向量，与dist/fullDist相乘并同当前位置相加即为下一个位置。其具体实现如下：

# speed单位为千米/小时，lng，lat代表当前位置的经纬度，nextNodeLng,nextNodeLat代表下一个节点的经纬度  
def nextPos(lng,lat,nextNodeLng,nextNodeLat,speed):   
 dist = speed / 3.6 # 1秒的距离  
 fullDist = getDistance(lng,lat,nextNodeLng,nextNodeLat)  
 scope = distance / fullDistance   
 lngMinus = float(nextNodeLng) - float(lng)   
 latMinus = float(nextNodeLat) - float(lat)   
 # (lngMinus, latMinus)即为所生成的向量  
 return (float(lng) + float(lngMinus) \* scope, float(lat) + float(latMinus) \* scope)

## 超速分析系统服务端的实现

系统的服务端使用Spring Boot框架，使用Kotlin语言进行编写。系统服务端的实现相较于客户端是简单的，其只需要提供返回公交信息，实时接收和获取公交车的位置速度信息并抽取出其中的超速信息并进行存储和分析操作，接收前端请求对数据库进行管理等功能即可。这里所说的公交信息指线路，节点，路径，地区等除公交车速度位置信息外的其它信息。

### 返回公交信息功能的实现

公交信息功能主要任务是向前端返回公交信息，其实现主要分为查看当前公交信息以及历史公交信息两部分，对于当前信息，根据区域代码查询表获取信息，对于历史信息，则根据区域代码和特定日期来查询信息并返回。系统不提供查看多日公交信息的功能。查看当前公交信息和历史公交信息的实现大部分是相同的，差别在于其查询数据库时对应的SQL语句，前者对于日期参数使用SQL中提供的CURRENT\_DATE函数，该函数将返回当前日期，后者需要后端给定日期参数。

该功能的实现所涉及的类图如图XX。TODO。时序图如图XX。该功能首先要求前端向InfoController发送get请求，其参数包括区域代码和日期字符串，日期是可选的，约定如果未给定日期则查询实时信息，否则查询历史信息。InfoController将调用InfoGetterService来获取结果并返回。其中InfoGetterService将依次访问DAO层中的BusMapper，BuslineMapper，NodeMapper，NodepathMapper，根据日期和区域代码信息获取相关的公交车信息，公交线路信息，节点信息，路径信息，并收集结果到一个临时的POJO对象中并返回给Controller层，由Controller将其转换为对应的JSON并返回给前端，由前端进行进一步处理和展示。

### 返回历史超速信息功能的实现

该功能主要为返回历史某时间段特定公交车或特定区域公交车的超速信息以及某时刻的位置速度信息，它将结合返回历史公交信息的功能使用户能够提供对任意历史时间的信息的获取。

### 接收返回实时信息功能的实现

实时信息功能有两个具体要求：实时接受公交车的信息，实时返回公交车的信息到前端。本系统使用websocket实现服务端和客户端，服务端和公交车的持久连接功能。该功能的实现设计类图如图XX所示TODO。其中GetPosWS为前端所使用的websocket服务端，它的实现利用了Kotlin提供的协程（coroutine）以减少线程调度的开销；UploadPosWS是公交车使用的websocket服务端，它仅负责接收公交车发送的信息并发送给Service；这两个websocket服务端都与RealTimePosService进行交互。RealTimePosService负责缓存信息并向数据库中插入公交车的位置速度信息和分析得到的超速信息。RealTImePosService在初始化时将查询数据库，获取数据库中当前所有节点和路径信息并保存到哈希表中以供快速查询。在将公交车发送的信息进行分析并存储进入数据库的同时，该Service同时也将缓存一定时间内接受的信息，并间隔时间发送最近接收到的信息到前端，使前端不需要查询数据库便可获得实时信息，从而减少对数据库进行查询的开销。

在具体实现中，RealTimePosService是注入到GetPosWS和UploadPosWS的伴生对象中的。伴生对象是Kotlin所提供的特性，其性质同Java中静态域或静态方法的性质基本相同。注入到伴生对象是因为websocket服务端是多例的，对每一个websocket客户端，它都将构造对应的websocket对象，而Spring假定使用的Component都是单例的，对其的依赖注入只会执行一次，因此必须注入在单例的伴生对象中。

关于实时接收公交车信息的功能，首先是公交车与系统建立websocket连接，公交车第一次发送位置信息时，系统将对当前公交车数量进行计数，记录公交车的id并将信息上传给RealTimePosService。考虑到公交车同公交线路是多对多的关系，该功能要求公交车同时要上传当前所处线路ID。RealTimePosService将插入位置信息到数据库，并检查公交车所对应线路信息是否在内存中，以及缓存该公交车信息的对象是否被初始化，如果没有，则从数据库中读取该公交车线路信息到内存，将缓存对象进行初始化。然后该Service将插入位置信息到缓存中，并保证对每辆公交车缓存不超过10分钟的信息，如果超过10分钟则删除最早的部分信息。

在接收到公交车所发送消息后，模块将遍历该公交车对应线路中每一条路径，检查其是否处于某路径且速度超过该路径限速。若满足条件，则证明该信息为超速信息，此时将信息插入到数据库并保存入缓存中，同时保证超速信息缓存最久不超过10分钟的信息。该功能的流程图如图XX TODO。其中，关于公交车同路径进行匹配的算法，考虑到公交车是在固定的路径上进行行驶的，本系统使用了比较简单粗糙的实现。对线路上的每一条路径，系统将计算公交车当前位置到该路径的距离，以及垂足的位置。当距离小于一定程度且垂足在路径的线段内时，说明公交车在该条路径上。该算法的具体实现如下。

// point为当前位置，p1和p2为路径上两线段  
fun pointOnSegment(  
 point : Pair<BigDecimal, BigDecimal>,  
 p1 : Pair<BigDecimal, BigDecimal>,  
 p2 : Pair<BigDecimal, BigDecimal>) : Boolean {  
 // x为经度，y为纬度  
 val (x, y) = point // 当前位置  
 val (x1, y1) = p1 // 线段上第一个点  
 val (x2, y2) = p2 // 线段上第二个点  
 val (x0, y0) = getPedal(point, p1, p2) // point到直线p1p2的垂足  
 // 获取到直线的距离  
 val dist = getDistance(x,y,x0,y0) // 点到直线的距离  
 val diff = BigDecimal(0.000001) // 允许垂足位置存在一些误差  
 // 当距离小于5米，且垂足在线段上时则为真  
 return dist < 5 && x0 >= minOf(x1, x2)-diff && x0 <= maxOf(x1, x2)+diff  
 && y0 >= minOf(y1, y2)-diff && y0 <= maxOf(y1, y2)+diff  
}

关于实时返回信息到前端的功能，前端将首先通过前一节中所述的返回公交信息的功能查询某地区的实时公交信息并获取该地区公交车列表，然后同系统建立websocket连接。建立连接后，前端将发送一个特定形式的JSON字符串，其表示前端试图通过区域代码或公交车列表来请求一定的公交车实时信息。系统接受到该信息后获取相应的公交车列表，根据该列表获取RealTimeService中对该列表中公交车的所有缓存信息并发送给前端。之后，系统将为前端创建一个协程，其将间隔1秒发送这些公交车的缓存信息中最近收到的信息给前端。前端仍可重新请求所需公交车列表，这时系统将取消该协程，并重新执行上述步骤。该功能使用协程的原因是其能够对线程池进行有效的重用，从而减少线程创建和切换的开销，提供比线程更高的性能，且不用对应每一个客户端都建立一个新的线程。Kotlin对协程提供了有效的支持，使通过协程的并发代码的编写是容易的。实时返回信息功能的流程图如图XX。TODO

### 信息管理模块的实现

信息管理模块主要负责管理员登录，修改密码，接受前端请求对数据库进行管理等操作。出于系统的性质，系统不提供管理员注册的接口。

关于管理员登录功能，系统不存储密码的明文，而是将其加盐后加密存储到数据库中。管理员首先在前端输入ID和密码，通过POST请求发送到AdminController，AdminController将该请求传递给AdminService，该Service将对密码加盐并使用md5对其进行加密，生成32位十六进制token，将其与数据库中所存储的账号信息进行匹配。若匹配到结果则登录成功，返回结果到Controller，由它发送包括所生成token的相关信息到前端。考虑到系统后端同前端可能并非在同样的域中，可能会有跨域问题，cookie由前端进行设置。这整个操作中需要多次查询数据库和插入信息到数据库，显然是线程不安全的，必须将其转化为同步操作。考虑到管理功能并不需要对性能有特殊要求，可以将最外层即Controller调用的方法使用synchronized锁以保证线程安全。

关于公交信息的管理，系统前端将返回对信息的一个操作序列。该操作序列中的元素就操作的种类来说有三种：插入，更新，删除；就操作的对象来说有七个：公交车，节点，路径，线路，地点，地点中的公交车，地点中的线路。系统应当对该操作序列进行正确的排序和验证，并依序执行以保证在插入过程中维护数据库中约束。其中，对于插入和更新操作，应当按照地点、公交车、节点、路径、线路、地点中的公交车、地点中的线路的顺序进行。操作的顺序按照插入、更新、删除的顺序进行。

对于删除操作则应对各个对象具体分析。对公交车的删除操作应当同时删除其和地点的关系，并保证在任何线路中没有对其的引用；对节点的删除操作应当保证没有引用该节点的路径；对于路径的删除操作应保证没有线路对其存在引用；对线路的删除操作应同时删除其和地点的对应关系；对地点的删除没有需要注意的。因此对于删除操作，对应的顺序是地点中的线路、地点中的公交车、线路、路径、节点、公交车、地点，其顺序与插入、更新操作正好相反。因此得到总的操作顺序：插入地点、插入公交车、插入节点、插入路径、插入线路、插入公交车到地点、插入公交线路到地点、更新地点、更新公交车、更新节点、更新路径、更新线路、从地点删除线路、从地点删除公交车、删除线路、删除路径、删除节点、删除公交车、删除地点。系统将对操作序列按此顺序进行排序并依序执行。

用户首先在前端进行可视化编辑，生成操作序列并通过POST请求发送到后端。该请求将首先经过Filter，检查cookie并查询数据库以判断是否登录，然后传递给AdminController，经由AdminService进行处理。

前端首先向系统发送给定格式的JSON对象，它是一个对象数组，其中每一个对象包括target，type，data三个成员，其中target指定操作对象，type指定操作类型，data则指定操作的值。关于对实体表的操作，对于删除操作，只需要给定实体的ID即可，对于插入操作，需要给定新插入的ID以及其他所有所需实体数据，对于更新操作，需要给定要更新的ID的操作以及所需要更新的数据项。关于对关系表的操作，关系表没有更新操作，对于删除和插入操作操作只需要给定存在关系的两实体的ID即可。上述所谓实体的ID指的并不一定是实体在数据库中的ID，操作序列中新插入的ID是“相对”的，对于每一种实体，它将从-1开始依次递减。系统在对操作序列进行处理前，应当将所有“相对”ID转换为将要存储的数据库中的ID，转换方式是获取每个实体表的最大ID，将其与“相对”ID的绝对值相加即为所需ID。但地点除外，地点的区域代码总是同数据库，同现实的行政区划的区域代码一致的。

在对操作序列进行处理后，将该数组按操作的type进行分类，再按照target进行排序后分发给相应的方法进行处理。这里排序的实现是根据更新和插入操作的顺序定义每种操作对象的权值到一个哈希表中，排序时以target作为key取值即可。考虑到删除操作和更新、插入操作的顺序是相反的，对于删除操作，将该值取反即可。相应方法将按顺序执行每条操作。代码实现如下。

data class Option(  
 val target: String,  
 val type: String,  
 val data: MutableMap<String, Any>  
)  
  
// 控制器执行的方法，应当对各种ID进行转换，排序，分发，这里直接用synchronized，从而避免线程不安全  
@Transactional // 标识这是一个事务，如果任意地方出错，将回退整个修改  
fun execute(json: JSON) = synchronized(this) { // kotlin版synchronized的用法  
 val key2Value = mapOf(  
 "INSERT" to 100,"UPDATE" to 200,"DELETE" to -300, // 弃用，现只用于检测值是否合法  
 "PLACE" to 1,  
 "BUS" to 2,   
 "NODE" to 3,  
 "NODEPATH" to 4,  
 "BUSLINE" to 5,  
 "BUS\_OF\_PLACE" to 6,  
 "BUSLINE\_OF\_PLACE" to 7  
 )  
 val maxNodeId = nodeMapper.getMaxNodeId();  
 val parse = mapOf(  
 "BUS" to ("bus\_id" to busMapper.getMaxBusId()),  
 "BUS\_OF\_PLACE" to ("bus\_id" to busMapper.getMaxBusId()),  
 "BUSLINE" to ("line\_id" to buslineMapper.getMaxBuslineId()),  
 "BUSLINE\_OF\_PLACE" to ("line\_id" to buslineMapper.getMaxBuslineId()),  
 "NODE" to ("node\_id" to maxNodeId),  
 "NODEPATH" to ("nodepath\_id" to nodepathMapper.getMaxNodepathId())  
 )  
 (json.toJavaObject(List::class.java) as List<Option>)  
 .onEach {  
 // 校验数据  
 if (!key2Value.containsKey(it.type) || !key2Value.containsKey(it.target)) {  
 throw Error("JSON格式不正确！") // 这里抛出的异常让调用者（Controller）进行处理  
 }  
 parse[it.target]?.let { (elem, maxId) ->  
 // 更改"相对"ID到绝对ID  
 if ((it.data[elem] as Int) < 0)  
 it.data[elem] = abs(it.data[elem] as Int) + maxId  
 // 当为路径的时候，其两个节点ID也可能是相对的  
 if (elem == "nodepath\_id") {  
 if (it.data.containsKey("node1\_id") && (it.data["node1\_id"] as Int) < 0)  
 it.data["node1\_id"] = abs(it.data["node1\_id"] as Int) + maxNodeId  
 if (it.data.containsKey("node2\_id") && (it.data["node2\_id"] as Int) < 0)  
 it.data["node2\_id"] = abs(it.data["node2\_id"] as Int) + maxNodeId  
 }  
 }  
 }  
 .groupBy { it.type }  
 .let { map ->  
 insertData(map["INSERT"]?.run { sortedBy { key2Value[it.target]!! } }) // 有可能一个都没有，这时候返回null  
 updateData(map["UPDATE"]?.run { sortedBy { key2Value[it.target]!! } })  
 deleteData(map["DELETE"]?.run { sortedBy { -key2Value[it.target]!! } })  
 }  
}

其中insertData，updateData，deleteData方法接受操作的对象的数组，其只需按顺序根据数组中每一个元素的Target属性对该对象调用Dao层中相应Mapper方法即可。举例来说deleteData方法的实现如下。

fun deleteData(options: List<Option>?) =  
 options?.forEach { // 如果为null就什么都不做  
 val data = it.data  
 when (it.target) { // 根据target调用特定方法  
 "PLACE" ->  
 placeMapper.deletePlace(  
 data["adcode"] as Int  
 )  
 "BUS" ->  
 busMapper.deleteBus(  
 data["bus\_id"] as Int  
 )  
 "NODE" ->  
 nodeMapper.deleteNode(  
 data["node\_id"] as Int  
 )  
 "NODEPATH" ->  
 nodepathMapper.deleteNodepath(  
 data["nodepath\_id"] as Int  
 )  
 "BUSLINE" ->  
 buslineMapper.deleteBusline(  
 data["line\_id"] as Int  
 )  
 "BUS\_OF\_PLACE" ->  
 placeMapper.removeBusFromPlace(  
 data["bus\_id"] as Int,  
 data["adcode"] as Int  
 )  
 "BUSLINE\_OF\_PLACE" ->  
 placeMapper.removeLineFromPlace(  
 data["line\_id"] as Int,  
 data["adcode"] as Int  
 )  
 }  
 }

### 超速信息分析聚集功能的实现

TODO：该模块应当间隔30分钟从数据库中查询这段时间的超速情况，并将其进行**聚集**。（当然，如果最终没有时间，这个功能就把它砍掉）

## 超速分析系统客户端的实现

系统的客户端即前端的运行环境是浏览器，其将使用Vue3.0和Typescript语言，围绕AntV-l7地理可视化框架进行编写。客户端需要给用户提供返回某地区当前或历史公交信息，实时获取当前公交车速度位置信息和超速信息，获取历史速度位置信息和超速信息并将这些信息展示在地图上的功能，同时也应提供对历史超速信息的统计数据进行可视化展示和分析功能。客户端也要给管理员提供可视化信息管理的功能，使管理员即使不了解数据库结构也能轻易对数据进行增删改操作。前端界面所使用的各组件在物理上通过Vue提供的单文件组件功能进行组织和编写以充分重用和提高维护性和扩展性。

### 信息获取模块的实现

信息获取模块从功能和实现上可以分为实时获取公交车速度位置信息及超速信息和获取非实时的信息两部分，其中实时获取信息通过原生websocket实现，获取非实时的信息，如公交信息和历史信息则通过对axios库进行实现。该模块将利用websocket或axios库对应后端系统提供的每一获取信息的接口编写对应的函数以对获取信息以及对信息进行的操作进行抽象以供页面进行使用。页面在使用时，只需给定参数和获取信息后执行的回调函数即可，这样抽象的优势在于能够在执行回调函数前对获得数据进行特定的处理和筛选从而使调用者不需要关心同后端交互的细节。并且本模块充分利用Typescript的类型定义功能，限定了所有操作中获取的结果的具体类型，从而使代码更加具有类型安全性。关于获取非实时信息功能，主要可分为获取当日或历史公交信息，获取历史超速信息和超速分析信息，获取某时刻公交车位置速度信息等。下面就获取公交信息的功能进行举例，该函数异步获取某区域的实时或历史的公交信息，这些信息即公交车信息，节点信息，路径信息，公交线路信息是以数组形式存储的，这里在获取后端返回结果后将数组数据转换为按id进行索引的表结构，并将线路信息中的公交车列表和路径列表从字符串转换为number数组，最后将回调函数应用到转换后的数据。在这里，函数的调用者不再需要关心后端发送的原始数据，这个函数在获取信息之余同时起了适配器的作用。

function getAllInfo(  
 adcode: number,  
 date: string | null,  
 callbackFn: (data: AllInfo) => void) {  
 axios.get(`${config.getInfo}?adcode=${adcode}${date ? `&date=${date}` : ''}`)  
 .catch(err => console.log(err))  
 .then((res) => {  
 if (!res) return; // 如果是空的，则直接返回，不调用回调函数了  
 let { buses: buses\_, buslines: buslines\_, nodes: nodes\_, nodepaths: nodepaths\_ }   
 = res.data as BeforeInfo; // 解构后端返回结果到定义的数据类型  
 let buses: Record<number, Bus> = {}  
 let buslines: Record<number, Busline> = {}  
 let nodes: Record<number, Node> = {}  
 let nodepaths: Record<number, Nodepath> = {}  
 buses\_.forEach(bus => buses[bus.bus\_id] = bus)  
 buslines\_.forEach(busline => {  
 buslines[busline.line\_id] = {  
 line\_id: busline.line\_id,  
 line\_name: busline.line\_name,  
 start\_node\_id: busline.start\_node\_id,  
 bus\_id\_list: JSON.parse(busline.bus\_id\_list),  
 nodepath\_id\_list: JSON.parse(busline.nodepath\_id\_list)  
 }  
 });  
 nodes\_.forEach(node => nodes[node.node\_id] = node);  
 nodepaths\_.forEach(nodepath => nodepaths[nodepath.nodepath\_id] = nodepath)  
 callbackFn({ buses, buslines, nodes, nodepaths }); // 结构成AllInfo类型——Typescript判断是否是同一类型只看结构，不看名称  
 })  
}

关于实时获取公交车速度位置信息功能，该功能以要查看公交车的列表作为参数，向后端建立websocket连接，发送该信息到后端，并在每次收到消息时执行回调函数。在应用中系统首先需要获取该地区的所有信息，从中获取所要查看的公交车的列表并使用该功能。流程图如下。TODO

### 信息展示模块的实现

信息展示模块是前端中最重要的部分之一，其负责展示地图以及显示应当显示在地图上的信息如公交车线路，站点信息，实时位置，最近一段时间的按速度染色轨迹，超速密度图等。该模块的诸组件完全围绕AntV-L7框架所提供的地图（Scene）和图层（Layer）对象编写，即为地图和每种所需图层建立相应组件，在组件中编写处理地图、图层对象的逻辑。本系统编写了如下组件：scene，busline-layer，buspos-layer，density-layer，realtime-layer，history-layer，marker-layer，每个组件都通过Vue的单文件组件功能进行组织。scene组件代表地图对象，其将根据经纬度信息进行初始化；busline-layer代表展示公交车线路的图层，对其的初始化需给定scene对象和所需线路信息，同时提供聚焦功能，让用户能够选择查看特定线路；buspos-layer代表展示公交车位置的图层，对每辆公交车的位置，其将放置一个公交车的图标以进行标识，对其的初始化需要给定scene对象和各公交车的位置信息，用户也可以聚焦到特定线路上的公交车；density-layer是展示超速情况的密度图图层，其给定每一个超速数据一个权值并显示在地图上，权值越大，值越密集，则亮度越高，从而标识出超速的情况在地图上；realtime-layer对应展示实时信息的功能，其包括buspos-layer和density-layer并利用信息获取模块对实时信息进行维护；history-layer对应显示历史信息的功能；其同样包括buspos-layer和density-layer，但是根据给定时间段对信息进行维护；marker-layer是用来在地图上进行标注的图层，系统用以显示鼠标所悬浮处的超速位置，公交车，线路的信息。该模块中地图和各图层的加载及加载顺序由使用者进行控制。

### 地点选择页面的实现

考虑到前端主要提供用户对单个地区获取各种信息的功能，选择地点是用户将要进行的最初的操作。用户进入客户端后，将进入选择地点的页面，其包括一幅中国的行政区划地图，用户应通过鼠标点击行政区划按省——市——区/县的顺序逐级选择所要查看的地点，该页面也将提供管理员登录功能。该页面截图如图XX。选择地点后，系统将弹出一个对话框，询问用户将要查看实时信息，历史信息或是超速统计信息并要求用户选择相关日期，然后带着日期和地点参数跳转到相应页面。若选择实时信息，则将跳转到获取实时信息的页面，其将获取当日的公交信息，并实时获取该地区的公交车速度位置信息；若选择历史信息，则要求用户给定要查看的日期并跳转到相应页面，将能够获取某日的公交信息和某时间段的超速信息，用户也可查看某具体时刻公交车的速度位置信息；若选择超速统计信息，则要求用户给定要查看的日期区间并跳转到相关页面，将能够获取多日的超速统计的信息。区分单日和多日的历史信息的原因是原始的超速信息的数据量可能是庞大的，同时获取多日的超速信息是网络所承受不了的，因此只允许获取单日某时间段的原始超速信息，对于更长的时间段则只允许获取对超速信息进行处理后生成的超速统计信息。该功能通过AntV-l7所提供的钻取地图layer进行实现。其流程图如图TODO。

### 实时信息页面的实现

在实时信息页面中，用户能够查看某地区当前的公交信息和实时查看公交车的位置信息和最近一段时间的超速情况，其中超速情况通过聚焦状态下的染色轨迹图（TODO：这个功能还没写呢）和密度图进行展示，这里的聚焦指用户选择查看特定公交线路及其上公交车信息的状态，不直接实时展示所有车辆的染色轨迹图的原因是因为对颜色不同的线段进行实时的更新渲染对性能的要求是很大的，会导致地图刷新帧数极低，影响用户的使用体验。页面截图如图XX。

页面加载时，将首先从URL中获取选择地区的参数，并按此参数对界面和地图进行初始化，调用信息获取模块中提供的函数从后端获取地区的包括公交车，节点，路径，线路的所有信息，之后利用公交车的列表建立websocket连接，实时接受公交车速度位置信息和超速信息并缓存和维护一段时间的信息。

用户在该页面

### 历史信息和超速分析页面的实现

TODO：老子才最重要！

### 信息管理功能和页面的实现

TODO

# 第六章 系统测试

为保证系统质量以及检测能否正确提供系统应有之功能，系统在上线前应当进行充分的测试。系统的性能测试由于业务环境的不确定性无法在当前条件下进行， 因此对系统的测试主要分为功能测试和安全性测试。下面主要对系统的各种功能模块进行测试来验证功能的正确性和系统的稳定性。

## 测试环境

测试环境中服务端和客户端同系统的开发环境相同，均为本机环境。具体环境如表TODO所示。（这个表是复杂表，markdown写不了）

## 功能测试

功能测试主要任务是检查系统前后端各功能是否能够得到正确的结果。其中前端主要测试各种信息的获取和展示，页面的切换等，同时也应测试管理员的登录功能对各种信息的管理功能，后端主要测试Controller所提供之API是否能够返回正确信息，以及对格式或参数错误的请求进行正确处理。

TODO：登录，更改密码，插入各种，更新各种，删除各种，获取历史信息，获取查看实时信息，聚焦，多日超速分析。

## 安全性测试

安全性测试的主要任务是检查前端信息管理功能是否能够正确检查管理员的登录情况，并在未登录的情况下对请求进行拦截。

# 第七章 总结和展望

TODO

## 全文总结

TODO

## 未来展望

TODO：分布式，用户体验，更多数据，多地区，性能（如果最后没写这部分的话），数据库重新设计：删除冗余，区域的层级结构，更多形式的分析

# 参考文献

TODO

# 致谢

TODO

# 附录

TODO