

## 时空大数据的快速渲染

### 摘要

本课题主要完成的是针对时空大数据的快速渲染工作，分为数据处理、数据可视化和前后端开发三个部分。旨在通过对已有的出租车时空大数据的分析与渲染，探索出租车的时空数据对城市规划、交通分析的意义，以不同方式展示并渲染处理所得数据，从而能够以不同角度分析并探寻构建智慧城市的方法，为研究街区的可达性提供数据基础和重要依据。

**关键词：**时空大数据，智慧城市，数据可视化

装  
订  
线

## Spatial-Temporal Big Data Fast Rendering

### ABSTRACT

The main task of this thesis is to solve the problem of data processing, data visualization and development of three parts. The purpose of this paper is to analyze and render the existing large-time data of taxis, to explore the significance of the space-time data of taxis to urban planning and traffic analysis, to display and render the data in different ways, so as to analyze and explore the method of constructing intelligent city from different angles, providing the data basis and important basis for the study of the accessibility of the block.

**Key words:** Spatial-Temporal Big Data, Wisdom City, data visualization

装  
订  
线

## 目 录

1 絮 论 .....	1
1.1 课题背景及意义 .....	1
1.2 项目背景和主要工作 .....	2
1.2.1 项目背景 .....	2
1.2.2 主要工作 .....	3
1.3 相关技术现状和研究趋势 .....	3
1.3.1 数据可视化 .....	3
1.3.2 数据聚类 .....	4
1.3.3 智慧城市 .....	4
1.4 本章小结 .....	5
2 开发环境和技术 .....	6
2.1 开发语言 .....	6
2.1.1 JavaScript 语言 .....	6
2.1.2 Linux shell 语言 .....	6
2.2 开发工具 .....	7
2.2.1 vim .....	7
2.2.2 Visual Studio Code .....	7
2.2.3 Chrome Devtools .....	7
2.3 渲染引擎与框架 .....	8
2.3.1 WebGL .....	8
2.3.2 Leaflet.js .....	8
2.3.3 OSMBuilding.js .....	8
2.4 数据存储概述 .....	9
2.4.1 Hadoop 介绍 .....	9
2.4.2 WebHDFS 介绍 .....	10
2.4.3 Hive 介绍 .....	10
2.5 服务器框架 .....	11
2.5.1 Node.js 介绍 .....	11
2.5.2 Express 介绍 .....	11
2.6 本章小结 .....	11
3 数据处理部分 .....	12
3.1 数据情况 .....	12
3.1.1 数据量级 .....	13
3.1.2 数据问题 .....	13
3.2 问题数据的处理措施 .....	13
3.3 处理过程的实现 .....	13
3.3.1 热力图数据计算 .....	13
3.3.2 街区热力图计算 .....	19
3.3.3 街区 OD 矩阵计算 .....	19
3.3.4 街区吞吐量计算 .....	21
3.3 本章小结 .....	21
4 数据可视化部分 .....	22
4.1 引言 .....	22
4.2 课题需求 .....	22
4.3 热力模型 .....	23
4.3.1 网格数据热力模型 .....	23

装  
订  
线

---

4.3.2 聚类数据热力模型 .....	29
4.3.3 街区数据热力模型 .....	35
4.4 街区吞吐量模型 .....	39
4.5 街区 OD 矩阵模型 .....	41
4.5.1 街区 OD 矩阵渲染 .....	42
4.5.2 街区 OD 矩阵效果图 .....	43
4.6 街区 OD 矩阵分析与应用 .....	45
4.8 可达性量化模型的分析与应用 .....	46
4.8.1 距离模型 .....	46
4.8.2 机会模型 .....	46
4.8.3 潜能模型 .....	47
4.8.4 时空约束模型 .....	47
4.8.5 效用模型 .....	47
4.8.6 比较 .....	48
4.7 本章小结 .....	48
5 应用开发部分 .....	49
5.1 数据计算 .....	58
5.2 后端开发 .....	49
5.2.2 服务器静态资源映射 .....	49
5.2.3 跨域和同源策略 .....	50
5.3 前端开发 .....	50
5.3.1 Leaflet 开发 .....	50
5.3.2 街区数据渲染 .....	54
5.4 本章小结 .....	64
5 总结和展望 .....	65
5.1 总结 .....	65
5.2 展望 .....	65
参考文献 .....	66
谢 辞 .....	68

装

订

线

## 1 絮论

### 1.1 课题背景及意义

时空大数据的难题不仅仅在于分析处理，也在于如何在可视化过程中尽可能多的将信息表达在图表中。海量数据的可视化除了要解决视觉复杂度问题之外，对计算能力也是很大的挑战。而时空大数据的可视化不仅要解决时间要素在时空中的情景，还要显示他本身的多维信息。

在地理学研究中，传统遥感技术长于自然地理特征的获取，却无法有效感知社会经济环境。大数据的出现，为人们定量理解社会经济环境提供了新的契机。

近年来，传感器网络、移动互联网、射频识别、全球定位系统等设备的快速发展和广泛应用，造成数据量的爆炸式增长，数据增加的速度远远超过现有的处理能力，虽然以 MapReduce 和 Hadoop 为代表的大规模并行计算平台的出现，为学术界提供了一条研究大数据问题的新思路，但这些技术也有其固有的局限性。一方面，时空数据本质上是非结构化数据，不仅包含时间序列模型，还存在地图模型，例如城市网络、道路网络等。

基于地图模型的算法时间复杂度通常比较大，对时空数据的存储管理和索引技术要求比较高。另一方面，MapReduce 计算模型的组织形式和数据处理方法不适合处理时间数据模型；Hadoop 技术也无法有效支持数据挖掘中监督学习所用的迭代式计算方法，因而无法完全满足时空数据分析的需要。这些对学术界和工业界来说都是一项巨大的挑战。因此，为了分析处理时空大数据，迫切需要更可靠、更有效和更实用的数据处理和可视化技术。

社交网络、遥感和传感器等设备的普遍应用产生了海量的时空数据，且每种设备生成的数据和数据形势各不相同，形成了时空数据结构复杂且来源多样的特性。此外，互联网的蓬勃发展，在蚊子、音频和视频等多媒体数据中同样包含了丰富的时空数据。例如，广泛覆盖城市的监控摄像头，纪录了道路车辆的轨迹信息，从视频中可以还原出被监控车辆的移动轨迹。所以，对时空数据进行有效整合、清洗、转换、提取和可视化是时空数据处理面临的重要问题。

时空数据中的时间关系和空间关系通常比较复杂，尤其很多可度量的和不可度量的时间关系和空间关系都是隐含在时空数据中，这就需要在数据挖掘系统中结合时空推理加以考虑这些复杂的时空关系。因此，结合时空推理和数据可视化需要适当折中模型表达能力和时空推理能力。而合理的大数据可视化也正是我们用来分析和挖掘时空大数据中所潜在的信息的重要方法。

## 1.2 项目背景和主要工作

### 1.2.1 项目背景

由于浏览器的性能提升，传统的时空数据可视化由客户端渐渐转向轻便的网页端。许多开源地图网站均使用 WebGL 对时空数据进行渲染，相应的技术虽然已经成熟，但在大数据量下的数据可视化仍有效率提升的空间。

时空数据分析常常伴随快速的数据需求、数据特征、数据模型的更改，传统的时空数据浏览方法明显受到数据量的效率限制。对于大数据量的时空数据加载，渲染方法的优劣足以影响用户浏览体验。

对数据进行特征提取后，数据的维度会超过可视范围，拓展性和动态分析是可视化研究中两个最主要的挑战。可视化每个数据点都可能导致过度绘制而降低用户的辨识能力，通过抽样或过滤数据可以删去离群值，而查询大规模数据库的数据可能导致高延迟，降低交互速率。

本课题属于长三角智慧城市协同中心下的项目，同级大学软件学院为技术负责单位，该大数据平台搭建的主要目的是为了发掘智能城镇或城镇群落关键数据的融合机制，实现生活、自然、社会、人文和环境等基础数据的融合和联动，并研发涵盖城市内部动态数据，城市群落间联动实时数据，以及城市时空数据的分析框架，同时研发城镇群落指标评价的分析与评估模型，提供实时可靠的数据分析引擎，最终能够为城市规划研究人员提供计算平台，提供所需的数据等。

该平台面向长三角区域城市规划，整合并储存了经济、社会、生态、交通与基础设施等五个领域 51 项近 120 亿条城市数据，为数据分析和可视化提供了完备的数据储备，为大数据分析提供了充足的真实数据，以 ArcGIS Server 作为时空数据分析与地图可视化平台，并结合 Spark、ElasticSearch 等大数据处理及应用等前沿技术进行实时数据处理与分析，充分提高了时间数据分析效率。

平台总体架构如图 1.1

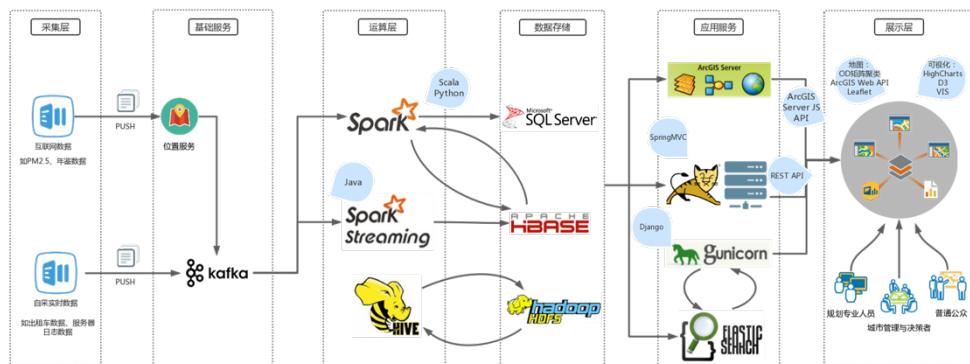


图 1.1 平台架构图

一共分 4 个主要部分

(1) 数据收集与基础服务，包括数据的采集、清洗与归一以及空间化处理

- (2) 运算与数据存储服务，即对数据做初步的整理、分析，并根据应用数据模型，把数据存入数据库或相应的 HDFS
- (3) 应用服务，即提供常用时空数据的分析模型和工具，并暴露为外部 API。
- (4) 数据展示服务，即提供数据的展示以及时空要素的结合、比较、图表等。

### 1.2.2 主要工作

在该数据分析平台上，我负责的工作是数据处理，数据可视化和数据渲染，因此，本文主要从数据处理，数据分析和数据渲染进行描述。以下是我做的主要工作：

(1) 数据处理：原数据为一张大表，包括的信息有车辆的 id、经纬度、时间、方向等，需要根据这张大表里的数据进行处理，得到用于展示的数据。例如 O-D 矩阵数据，根据时间、是否有人、经纬度、车辆 id 算出各个街区间的吞吐量、实时热力、街区间的 O-D 值，以及整个矩阵的极值来确定可视化的系数。

(2) 数据分析：依靠原数据得到新的目标数据后，还需要进一步对数据进行聚类和处理，一方面为了优化可视化的效果，一方面也为了优化网络传输的性能。由于该数据分析平台是基于 web 展示，因此，对数据在网络过程中的传输时间有更为严苛的要求，对性能和时间方面的优化是重中之重。

(3) 数据渲染：这里主要使用了一些 Javascript 的框架，并且比较了相似框架的优缺点，底层地图还是使用与平台相集成的 leaflet 框架，在展示时，热力图使用了 leaflet-heatmap 插件，三维的展示使用了 OSMBuilding.js 框架。在使用和改进框架的过程中也遇到了很多问题，后面会详细介绍所遇到的问题以及解决方案。

### 1.3 相关技术现状和研究趋势

#### 1.3.1 数据可视化

所谓数据可视化是对大型数据库或数据仓库中的数据的可视化，它是可视化技术在非空间数据领域的应用，使人们不再局限于通过关系数据表来观察和分析数据信息，还能以更直观的方式看到数据及其结构关系。数据可视化技术的基本思想是将数据库中每一个数据项作为单个图元元素表示，大量的数据集构成数据图像，同时将数据的各个属性值以多维数据的形式表示，可以从不同的维度观察数据，从而对数据进行更深入的观察和分析。

现在较为流行的数据可视化工具包括 excel、传统 BI 分析工具、以及第三方可视化插件等。这一技术现在正处于不断演变之中的概念，其边界在不断地扩大。主要指的是技术上较为高级的技术方法，而这些技术方法允许利用图形、图像处理、计算机视觉以及用户界面，通过表达、建模以及对立体、表面、属性以及动画的显示，对数据加以可视化解释。与立体建模之类的特殊技术方法相比，数据可视化所涵盖的技术方法要广泛得多。

当前，在研究、教学和开发领域，数据可视化乃是一个极为活跃而又关键的方面，并且随着计算机运算能力的发展，数据的爆炸性增长，数据可视化势必会迎来它的迅猛发展浪潮。

### 1.3.2 数据聚类

数据聚类是对于静态数据分析的一门技术，在许多领域受到广泛应用，包括机器学习，数据挖掘，数据可视化，模式识别，图像分析以及生物信息分析。

所谓聚类，将物理或抽象对象的集合分成由类似的对象组成的多个类的过程被称为聚类。由聚类所生成的簇是一组数据对象的集合。因此聚类分析又称群分析，它是研究（样品或指标）分类问题的一种统计分析方法。聚类分析起源于分类学。

在传统的案例中，聚类已经比较成功的解决了低维数据的聚类问题。但是由于实际应用中数据的复杂性，在处理许多问题时，现有的算法经常失效，特别是对于高维数据和大型数据的情况。因为传统聚类方法在高维数据集中进行聚类时，主要遇到两个问题。高维数据集中存在大量无关的属性使得在所有维中存在簇的可能性几乎为零。

因此，作为聚类技术的难点，高维聚类分析已成为聚类分析的一个重要研究方向。随着技术的进步使得数据收集变得越来越容易，导致数据库规模越来越大、复杂性越来越高，高维聚类也越来越有挑战性。

就目前而言，高维数据聚类分析已经成为聚类分析中一个非常活跃的领域，同时它也是一个具有挑战性的工作，在市场分析、信息安全、金融、娱乐、反恐等方面都有很广泛的应用。研究高维聚类对本课题的后续工作有很大帮助。

### 1.3.3 智慧城市

智慧城市的概念最早源于 IBM 提出的“智慧地球”这一理念，具体地说，“智慧”的理念就是通过新一代信息技术的应用使人类能以更加精细和动态的方式管理生产和生活状态，通过把传感器嵌入和装备到全球每个角落的供电系统、供水系统、交通系统、建筑物和油气管道等生产生活系统的各种物体中，使其形成的物联网与互联网相联，实现人类社会与物理系统的整合，而后通过超级计算机和云计算将物联网整合起来，即可实现。此后这一理念被世界各国所接纳，并作为应对金融海啸的经济增长点。同时，发展智慧城市被认为有助于促进城市经济、社会与环境、资源协调可持续发展，缓解“大城市病”，提高城镇化质量。

狭义上说是使用各种先进的技术手段尤其是信息技术手段改善城市状况，使城市生活便捷；广义上理解应是尽可能优化整合各种资源，城市规划、建筑让人赏心悦目，让生活在其中的市民可以陶冶性情心情愉快而不是压力，总之是适合人的全面发展的城市。可以说，智慧城市就是以智慧的理念规划城市，以智慧的方式建设城市，以智慧的手段管理城市，用智慧的方式发展城市，从而提高城市空间的可达性，使城市更加具有活力和长足的发展。

可达性，是指从给定地点到其他地方工作、购物、娱乐或就医的方便程度，可达性决定于供需分布以及二者在空间上的联系，可达性的计算对于资源或服务设施的空间分布均衡有重要意义，下面介绍可达性在本课题中的计算。

#### 1.4 本章小结

本章首先介绍了关于本课题的背景和意义，然后介绍了项目的背景和功能，最后介绍了一些本项目涉及到的知识的研究现状和发展趋势，以及自己所需要完成的三项任务。接下来将介绍我在项目中使用到的开发环境和工具。

装  
订  
线

## 2 开发环境和技术

### 2.1 开发语言

在本课题中，主要使用的开发语言是 Javascript 以及 Linux 脚本语言。Javascript 语言主要用途有：Node.js 搭建本地服务器，处理数据；Leaflet.js 与地图交互，OSMBulding.js 展示三维信息，Linux 脚本主要用来对文本数据的一些处理操作。

#### 2.1.1 JavaScript 语言

JavaScript (JS) 是一种轻量级解释型的，或是 JIT 编译型的程序设计语言，有着函数优先 (First-class Function) 的编程语言。虽然它是座位开发 web 页面的脚本语言而出名的，但是在很多非浏览器环境中也使用 JavaScript，例如 node.js 和 Apache CouchDB。JS 是一种基于原型、多范式的动态脚本语言，并且支持面向对象、命令式和声明式（如：函数式编程）编程风格。

在 WEB 数据渲染领域，JS 语言有很多独特的优势：

- (1) 友好的开发体验，JS 是一种解释型语言，提供了一个非常方便的开发过程。  
JavaScript 的语法基本结构形式与 C、C++、Java 十分类似。但在使用前，不需要编译，二十载程序运行过程中被逐行的解释，方便开发者开发。
- (2) 庞大的开源社区，JS 提供了许多开源的框架，以及丰富的代码库，在 web 页面渲染方面有很多现成的框架可以使用。
- (3) 良好的兼容性，ECMAScript 是 Javascript 的标准，截至 2012 年，所有的浏览器都完整支持了 ECMAScript5.1，只要计算机能运行浏览器，并支持 JS，就可以正确运行相应的应用，实现了一次构建，多次运行的目标。

因为 JS 语言在数据的渲染和展示等方面的功能很强大，而且在课题中关于数据可视化的结果需要不同的框架来处理和操作数据，所以采用了 JS 语言来完成对数据的渲染，并对可视化方法进行改进。此外，在渲染结果的展示上也利用了 JS 语言优质的函数式编程思想。

#### 2.1.2 Linux shell 语言

Linux shell 语言有一些命令非常适合用来处理文本数据，这一点在我处理各种数据的时候起到了很大的作用。

比如，在对文本数据处理方面，Linux 脚本语言有很多得天独厚的命令可以使用，例如 awk、sed、perl，切割文件的命令，例如 split、cat，能够简单快速的完成处理并 debug 的工作。

## 2.2 开发工具

### 2.2.1 vim

vim 是一个类似 vi 的功能强大、高度可定制的文本编辑器，在 vi 的基础上改进和增加了很多特性。Vim 的理念是组合：vim 的普通模式下的命令支撑起了它很大一部分的强大的编辑能力，它的设计理念即为命令的组合。例如：普通模式下“dd”为删除并将当前行拷贝到剪切板，“dj”则为删除到下一行，这其中的原理是第一个“d”所代表的含义是删除，而“j”代表移动到下一行，将“d”和“j”组合后的命令所代表的操作是删除当前行和下一行；还可以指定次数，“4dd”代表重复“dd”4次，即连续删除4行。将这些命令灵活组合，就可以很高效的进行文本编辑。并且在 vim 的设置中，很多快捷键的设置和正则表达式类似，极大的方便了程序员的使用。

### 2.2.2 Visual Studio Code

Visual Studio Code 是一个运行于 OS X, Windows 和 Linux 之上的，针对于编写现代 Web 和云应用的跨平台编辑器，相比同类型的编辑器来说，它比 sublime 凯源，比 atom 速度更快，比 webstorm 更轻量级。

主要有以下优点：

- (1) 文件目录管理很强大。
- (2) 自定义配置，主题，自动保存，可以设置延迟毫秒后保存，也可以设置文件失去焦点时自动保存。
- (3) 集成 Git，快速 diff,而且修改文件后会左边会显示指示器，比如删除会显示红色，增加显示绿色。
- (4) 智能提示很强大，作为一款编辑器这是 sublime 和 atom 没法比的。
- (5) Go to Definition 很方便,按 F12 自动跳转到方法定义处，如果不想跳转，直接 shift+F12 实现 Peek 功能。
- (6) 自带 emmet 支持 html/jade/css/less/sass/xml，对前端很友好。
- (7) 自带强大的调试功能，对 node 程序调试有很大帮助。
- (8) 插件支持十分丰富，安装插件简单便捷。

### 2.2.3 Chrome Devtools

Google 浏览器基于 Chromium 开源项目，其内核采用的是 webkit.Chrome。有很多优点，包括简洁的界面，很高的稳定性，速度和安全性。时至今日，Chrome 发布已经超过 7 年，凭借其良好的性能，其市场占有率已经跃居第一位。

Chrome 开发者工具是一套内置于 Google Chrome 中的 Web 开发和调试工具，可用来对网站进行迭代、调试和分析。

在使用渲染框架对我们的数据进行渲染时，可以用 Devtools 做很多优化，例如：

- (1) 使用 Elements 面板查看页面的样式，并且能够实时编辑这些代码。
- (2) 可以用 console 测试网页和代码；可以用断点调试来 debug JavaScript。
- (3) 可以通过 Network 面板查看资源的请求和下载时间来调试网络性能。
- (4) 可以通过 Profiles 面板来记录和分析时间和内存使用情况，更好的优化代码和资源。
- (5) 通过 Resources 面板来查看加载页面所存储的数据信息，包括缓存，cookies 等等。

### 2.3 渲染引擎与框架

#### 2.3.1 WebGL

WebGL（全写 Web Graphics Library）是一种 3D 绘图标准，这种绘图技术标准允许把 JavaScript 和 OpenGL ES 2.0 结合在一起，通过增加 OpenGL ES 2.0 的一个 JavaScript 绑定，WebGL 可以为 HTML5 Canvas 提供硬件 3D 加速渲染，这样 Web 开发人员就可以借助系统显卡来在浏览器里更流畅地展示 3D 场景和模型了，还能创建复杂的导航和数据视觉化。显然，WebGL 技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂 3D 结构的网站页面，甚至可以用来设计 3D 网页游戏等等。

本课题中使用的渲染框架底层都是通过 canvas 来操作 WebGL 的 API 来实现对 3D 模型、2D 平面模型的渲染、绘制等操作。

#### 2.3.2 Leaflet.js

Leaflet 是面向移动设备的交互式地图的领先开源 JavaScript 库。轻量级，Gzip 压缩后仅 33 KB，它具有大多数开发人员需要的所有功能。以简洁，高性能和易用性为理念。它可以在所有主流的桌面和移动平台上有效地工作，可以扩展大量的插件，拥有优美，易于使用和整理良好的 API 以及简单可读的源代码。

我们只需要在页面上建立一个 map 的容器，就可以在上面自定义瓦片图层，通过添加不同的图层来实现不同的地图展示。

本课题将使用 JavaScript API 来完成对 Leaflet map 的操作，并实现热力、od 矩阵等不同效果的实现。

#### 2.3.3 OSMBuilding.js

OSMBuilding 是一个用于在 2D 和 3D 地图上为可视化 OpenStreetMap 来构建几何模型的 JavaScript 库。其与 leaflet 的良好结合是选择它来渲染模型的重要原因，在 leaflet 的众多插件中，OSMBuilding 是在渲染 3D 建筑物方面最为性能优良的插件。

它的优点如下：

- （1） 良好的设备兼容性。  
（2） 与大部分的硬件工作有着不错的表现。  
（3） 与 Leaflet 和 OpenLayers 完成集成。  
（4） 支持现代 3D 的各种模型。  
（5） 组合各种数据源。  
（6） 适合模型复杂、有大量数据的地图渲染。  
（7） 能够为多个模型建立响应事件。  
（8） 支持多种数据格式。  
（9） 对模型的创建、销毁、修改等操作十分方便。  
（10） 能够保持地图数据的缩放一致渲染。

因此，OSMBuilding 引擎库很适合本课题研究，表 2.1 为 OSMBuilding 的各个版本对比。

表 2.1 OSMBuilding 版本对比表

版本	名称	是否支持热力	是否支持 3D	是否支持倾斜
WebGL	OSMWebGL	否	是	是
Leaflet	OSMLeaflet	是	是	否
official	OSM3	否	是	是

由于尝试展示 3D 的模型，因此能否倾斜展示图层也作为考察的一项，最后确定以功能优先，选择 OSMBuilding 的 Leaflet 版本作为开发版本。

在本课题中，数据处理后会先放到服务器上，然后在渲染时，每次都会从服务器上请求数据，数据在 http 传输过程中的格式为 JSON，这样便于 JavaScript 直接解析，将数据解析为相应的 feature 后，通过 Leaflet 和 OSMBuilding 的 API 来将数据通过 WebGL 渲染到浏览器上，并根据需要绑定相应的响应函数，完成与用户的交互。

## 2.4 数据存储概述

### 2.4.1 Hadoop 介绍

Hadoop 是一个由 Apache 基金会所开发的分布式系统基础架构，用户可以在不了解分布式底层细节的情况下，开发分布式程序，充分利用集群的威力进行高速运算和存储。

Hadoop 实现了一个分布式文件系统（Hadoop Distributed File System），简称 HDFS。HDFS 有高容错性的特点，并且设计用来部署在低廉的（low-cost）硬件上；而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。HDFS 放宽了（relax）POSIX 的要求，可以以流的形式访问（streaming access）文件系统中的数据。在该框架中，最核心的

设计就是：HDFS 和 MapReduce。HDFS 为海量的数据提供了存储，则 MapReduce 为海量的数据提供了计算。

架构如图 5.1：

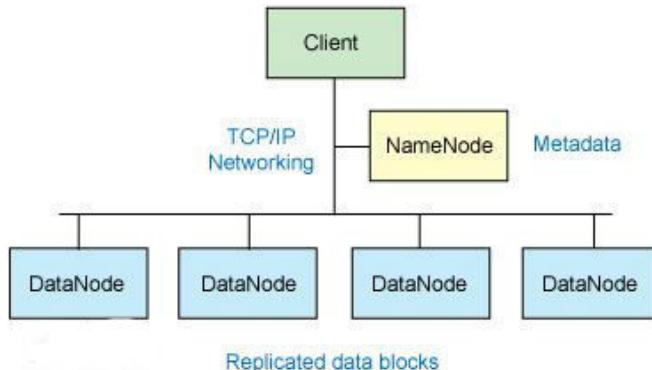


图 5.1 Hadoop 架构

#### 2.4.2 WebHDFS 介绍

WebHDFS 是 Hadoop 提供的对 web 方式访问集群的支持，包括读写和文件管理，用户可以使用 curl 模拟 http 消息进行请求，需要注意的是 Hadoop 配置参数中界定了是否支持身份认证，并设定了默认的用户名 web user，用户可以自行修改是否启用和更改默认的用户，如果不作处理，有时候会出现，用户权限不够，无法执行某些操作的问题，目前的 WebHDFS 主要支持功能见表 5.1：

表 5.1 WebHDFS 支持的功能

请求方式	操作	实现的类名
HTTP GET	OPEN	FileSystem.open
HTTP GET	GETFILESTATUS	FileSystem.getFileStatus
HTTP GET	LISTSTATUS	FileSystem.listStatus
HTTP PUT	CREATE	FileSystem.create
HTTP PUT	MKDIRS	FileSystem.mkdirs
HTTP PUT	RENAME	FileSystem.rename
HTTP POST	APPEND	FileSystem.append

当我们把数据处理好并写入 hdfs 后，即可通过 http 的方式请求到我们的数据，而不需要使用 Java API 来完成对数据的读写。

#### 2.4.3 Hive 介绍

Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供类 SQL 查询功能。它的本质是将 SQL 转换为 MapReduce 程序，有良好的可扩展性，可以自由的扩展集群的规模，一般情况下不需要重启服务。它有良好的延展性，Hive 支持用户自定义函数，用户可以根据自己的需求来实现自己的函数。还有良好的容错性，当节点出现问题时 SQL 仍可完成执行。

Hive 与 Hadoop 关系如图 5.2

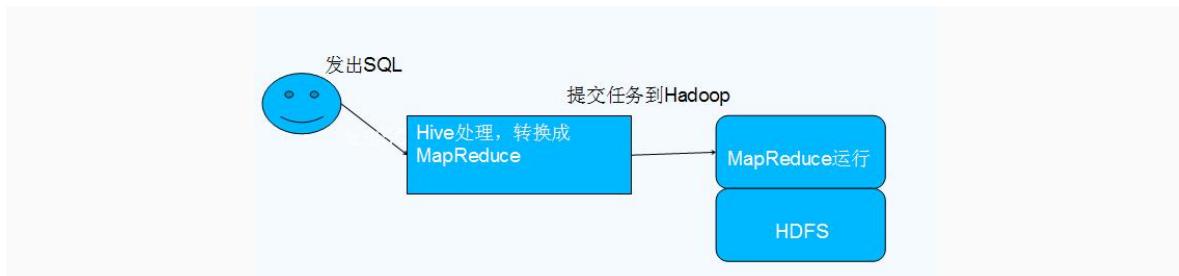


图 5.2 Hive 与 Hadoop 关系

## 2.5 服务器框架

服务器使用基于 Node.js 的框架 Express，下面将介绍具体概念。

### 2.5.1 Node.js 介绍

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境，其使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。而且 Node.js 的包管理器 npm，是全球最大的开源库生态系统。作为一个服务器端 JavaScript 解释器，Node.js 将改变服务器应该如何工作的概念。它的目标是帮助程序员构建高度可伸缩的应用程序，编写能够处理数万条同时连接到一个（只有一个）物理机的连接代码。

### 2.5.2 Express 介绍

Express 是一个基于 Node.js 平台的极简、灵活的 web 应用开发框架，它提供一系列强大的特性，能够创建各种 Web 和移动设备应用，自带丰富的 HTTP 快捷方法和任意排列组合的 Connect 中间件，这使得创建健壮、友好的 API 既快速又简单。在 Node.js 之上扩展了 Web 应用所需的基本功能，保证了性能的高效。为开发和部署应用节省了大量的时间同时又保证了高效运作。

## 2.6 本章小结

本章主要介绍了课题中主要使用到的开发语言、开发工具和渲染引擎。简要介绍了两种开发语言的实际使用场景，最后还介绍了 Leaflet 和 OSMBuilding 的优点，以及本课题同时采用这两种不同类型的渲染引擎的原因。接下来的一章，将详细介绍数据处理的工作。

### 3 数据处理部分

#### 3.1 数据情况

时空大数据的预处理包括时间数据的预处理和空间数据的预处理，在时间数据的预处理中，主要问题在时间序列数据结构复杂且来源多样，如何将不同来源的时间序列数据合并并用于时空数据分析，是时空数据预处理面临的首要问题。解决方案主要为构建时间间隔：不同的时间序列数据的时间起点及时间间隔都不尽相同，想要合并多个时间序列数据就必须要求每个时间序列数据具有相同的时间间隔，这就需要对各个时间变量的时间间隔进行构建，本课题中设定的时间间隔即为30分钟。

在空间数据的预处理中，主要问题在于空间坐标有不同的表达方式，有些空间数据的坐标是多个单维坐标形式，如用三个单独的字段x、y、z分别表示三维坐标；有些空间数据的坐标是一个多位坐标形式，如用一个列表[x、y、z]来表示三维坐标，而一旦涉及到坐标，就必然会关系到坐标系，而不同的空间数据的坐标系往往不同，此外，和时间数据一样，空间数据来源多样，每个空间数据的基本数据结构都不尽相同，如何关联多种类型的数据结构同样是空间数据预处理的重要问题。在本课题中，原始数据只考虑经纬度格式正确的数据，错误数据不予以计算。

数字来源是出租车的原始数据，每个单独的数据为一个小的JSON，共包含出租车所携带的所有信息，包括出租车id、出租车当前行驶速度、出租车当前GPS信息、出租车方向等信息，详细字段见表3.1。

表3.1 原始数据字段说明

字段名称	字段值示例	字段取值	字段意义
carId	“28443”	String	出租车的id，用来区分出租车个体。
isAlarm	“0”	String（“0”或“1”）	
isEmpty	“0”	String（“0”或“1”）	“0”时代表为空车，“1”时代表车上有人。
topLight	“0”	String	
receiveTime	“2015-04-01 12:46:37”	String	代表该条数据产生时的时间
GPSTime	“2015-04-01 12:46:31”	String	代表GPS数据生成的时间
Longitude	“121.586462”	String	代表该车此时的经度
Latitude	“31.213995”	String	代表该车此时的纬度
Speed	“29.6”	String	代表车运行的速度
Direction	“284.0”	String	代表车所处的方向

### 3.1.1 数据量级

目前，经过讨论，暂时选择了 2017-04-01 这天的数据作为数据处理分析的数据源，这一天的数据一共有 3 百万条。

### 3.1.2 数据问题

主要有以下几种主要数据问题：

- a. 很多数据存在空值；
- b. 时间字段问题：关于时间的字段是 String 类型，不是时间戳，还存在少量数据是无意义的数值，无法正常转化为时间戳来比较；
- c. 数据不是完整的 JSON，每一行为一个单独的 JSON，无法将整个文件直接读入使用，需要将原始数据进行处理，得到标准的 JSON 文件后即可读入内存使用。

### 3.2 问题数据的处理措施

存在问题的数据很有可能会成为噪音数据，影响算法分析的结果，甚至影响算法的正常运行，因此对数据进行数据清洗是势在必行的。在我的整个项目过程中，大量用到了 Linux 脚本来完成对文本数据的操作，例如每行加逗号等操作。

下面是针对上述的主要数据问题，最终确认了相对应处理措施，具体的处理措施见表 3.2。

表 3.2 问题数据的处理措施表

数据问题	问题处理措施
数据存在空值	暂不做处理，在具体算法中进行不同处理
时间字段问题	通过读入 String 的不同位置的值来比较
数据格式不正确	通过 Linux 脚本将数据格式化为标准 JSON

其中，使用脚本对时间的具体处理思路如下：

时间字段：统计字段中经常出现的几种日期格式，使用正则匹配来检测时间字符串，如果通过了正则匹配，就通过读时、分、秒对应的位数来得到时间，并转化为可以直接比较的 int 值，如果未通过，就放弃这条数据。

### 3.3 处理过程的实现

#### 3.3.1 热力图数据计算

目前，本课题所使用的数据处理方案主要为利用 node.js 原生读写文件 API 来完成对数据的读写，计算车辆热力图所需要的数据需要对原始数据进行聚类，目前可行的聚类主要有 2 种方法，一种是在原始数据上按经纬度用网格进行划分，在同一网格内的车辆数即为当前单元的热力值，一种是用聚类算法对原始数据进行聚类，

聚类依据为经纬度，聚类值为车辆数量。这一过程全部用 node.js 的原生 API 完成，所有数据的操作都放在内存里，处理完成后由文件读写的 API 写入文件中。

第一种，利用网格对原始数据点进行聚类，每个网格所包含的车辆总数为该网格在热力显示时所需的依据 count 值，按网格的边长为变量，对不同的网格边长得到不同的数据结果。计算步骤如下：

- (1) 原始数据（假设当前需要计算的是  $t_1-t_2$  时间间隔内的热力数据）见表 3.3。

表 3.3 热力网格数据计算第一步

car id	receiveTime	Longitude	Latitude
0	$t_1-t_2$	a1	b1
1	$t_1-t_2$	a2	b2
2	$t_1-t_2$	a3	b3
3	$t_1-t_2$	a4	b4
4	$t_2-t_3$	a5	b5

- (2) 对原始数据进行时间过滤，得到新数据，见表 3.4。

表 3.4 热力网格数据计算第二步

car id	receiveTime	Longitude	Latitude
0	$t_1-t_2$	a1	b1
1	$t_1-t_2$	a2	b2
2	$t_1-t_2$	a3	b3
3	$t_1-t_2$	a4	b4

- (3) 按网格过滤原始数据，每个网格有一个中心点，按网格的边长得到该网格的覆盖区域，然后对每个数据进行过滤时，根据它的经纬度判断出它属于哪个网格，对那个网格的 count 值加 1，以此类推。见图 3.1：

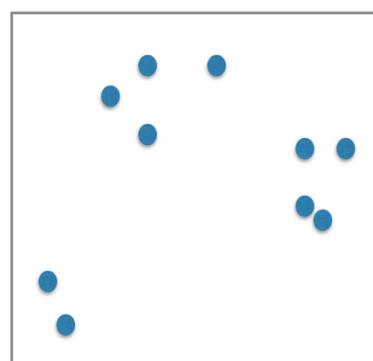


图 3.1 网格原始数据

经过网格聚类后如图 3.2:

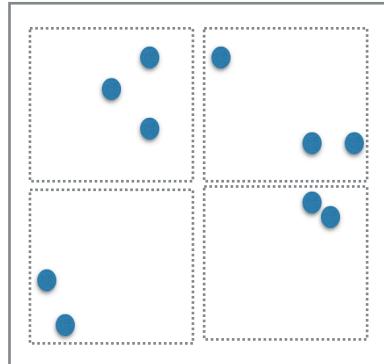


图 3.2 网格聚类数据

经过计算得到  $(a_1, b_1)$ 、 $(a_2, b_2)$  在网格 1 里， $(a_3, b_3)$ 、 $(a_4, b_4)$  在网格 2 里，则网格 1 的车辆数为 2，网格 2 的车辆数为 2，结果见表 3.5。

表 3.5 热力网格数据计算第三步

网格中心点经度	网格中心点纬度	网格 count 值
lng1	lat1	2
lng2	lat2	2

(4) 最后将此结果以 JSON 格式保存并放倒服务器上。

根据不同的数据参数，整理得表 3.6。

表 3.6 热力网格数据计算第四步

网格边长	网格个数	单个网格最大值（车辆数）
0.003	15822	26952
0.005	7713	21791
0.01	2777	49528
0.02	993	66330

第二种，利用聚类算法对原始数据进行聚类，聚类依据为经纬度，相近的数据会被聚类为一个单元，并且该单元的所有车辆数会作为该单元的 count 值，作为热力图的渲染依据。

在选择聚类算法的过程中，对几种知名算法进行了调研和比较，在我们的需求中，要求聚类算法的高度可伸缩性，由于许多聚类算法在小于 200 个数据对象的小数据集合上工作得很好，但是，对于一个大规模数据，如我们的原始数据，包含数百万个对象，在这样的大数据集合样本上进行聚类可能会导致偏差的存在。因此需

要容错率好的聚类算法，该算法需要在有噪声数据的时候正常运作，同时，也能够处理大数量级的数据。

并且，由于错误数据的不可避免，需要该算法能合理处理“噪声”，即缺失或者错误的数据，在调研过程中，针对不同的算法，整理了各自的优缺点见表 3.7。

表 3.7 聚类算法比较

算法	优点	缺点
k-means 聚类	分布较均匀 聚类效果好	取决于起始取点的优劣 初始点选择不稳定 (随机选取)会引起聚类结果的不稳定
层次聚类	比较适合地理数据 按层级划分，可以用多维划分	一旦一个分裂或者合并被执行，就不能修正，聚类质量受限制
SOM 聚类	可以对高维数据进行降维 不需要太多人工干预	每换一种数据，就需要学习一次 处理时间较长，不一定适应大型数据库
FCM 聚类	模糊相似矩阵	需要大量迭代 对初始聚类中心敏感，需要人为确定聚类数，容易陷入局部最优解

综合考虑，选择 K-Means 聚类算法，主要因为其简单、快速的优点，以及对大数据集有着较高的效率并且是可伸缩性的，而且时间复杂度近于线性，适合挖掘大规模数据集。

具体计算步骤如下：

(1) 原始数据（假设当前需要计算的是 t1-t2 时间间隔内的热力数据）见表 3.8。

表 3.8 热力聚类数据计算第一步

carId	receiveTime	Longitude	Latitude
0	t1-t2	a1	b1
1	t1-t2	a2	b2
2	t1-t2	a3	b3
3	t1-t2	a4	b4
4	t2-t3	a5	b5

(2) 对原始数据进行时间过滤，得到新数据见表 3.9

表 3.9 热力聚类数据计算第二步

carId	receiveTime	Longitude	Latitude
0	t1-t2	a1	b1
1	t1-t2	a2	b2
2	t1-t2	a3	b3
3	t1-t2	a4	b4

(3) 对原始数据通过聚类算法进行聚类，算法的原理为不停迭代，计算每一次结果的偏移量，当偏移量最小时，当前结果为最优解。

k-means 聚类的目的是把  $n$  个点（可以是样本的一次观察或一个实例）划分到  $k$  个聚类中，使得每个点都属于离他最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。具体步骤如下：

1. 随机选取  $k$  个聚类质心点（cluster centroids）为  $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ 。
2. 重复下面过程直到收敛：

对于每一个样例  $i$ ，计算其应该属于的类

$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

对于每一个类  $j$ ，重新计算该类的质心

例如，原始数据如图 3.3：

装  
订  
线

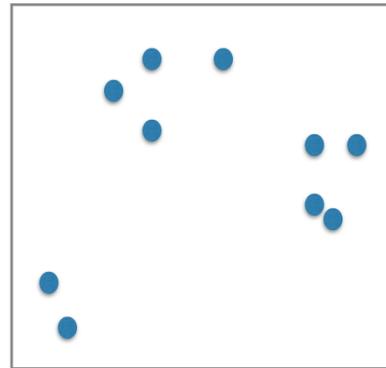


图 3.3 聚类原始数据

经过聚类后如图 3.4：

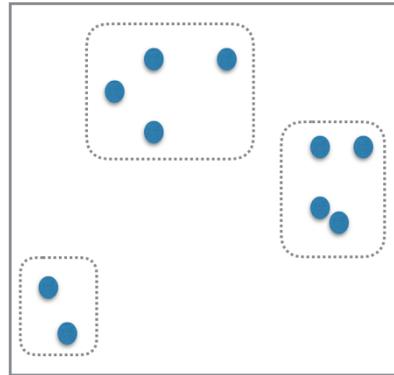


图 3.4 聚类结果数据

K-means 面对的第一个问题是保证收敛，前面的算法中强调结束条件就是收敛，可以证明的是 K-means 完全可以保证收敛性。

定义畸变函数 (distortion function) 如下：

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c(i)}\|^2$$

$J$  函数表示每个样本点到其质心的距离平方和。K-means 是要将  $J$  调整到最小。假设当前  $J$  没有达到最小值，那么首先可以固定每个类的质心，调整每个样例的所属的类别  $c^{(i)}$  来让  $J$  函数减少，同样，固定  $c^{(i)}$ ，调整每个类的质心  $\mu_j$  也可以使  $J$  减小。这两个过程就是内循环中使  $J$  单调递减的过程。当  $J$  递减到最小时， $c$  和  $\mu$  也同时收敛。（在理论上，可以有多组不同的  $\mu$  和  $c$  值能够使得  $J$  取得最小值，但这种现象实际上很少见）。

由于畸变函数  $J$  是非凸函数，意味着我们不能保证取得的最小值是全局最小值，也就是说 k-means 对质心初始位置的选取比较感冒，但一般情况下 k-means 达到的局部最优已经满足需求。但如果怕陷入局部最优，那么可以选取不同的初始值跑多遍 k-means，然后取其中最小的  $J$  对应的  $\mu$  和  $c$  输出。

经过计算得到  $(a_1, b_1)$ 、 $(a_2, b_2)$ 、 $(a_3, b_3)$  的经纬度较为相近，聚为一类  $m$ ， $(a_4, b_4)$  单独为一类  $n$ ，则  $m$  的车辆数为 3， $n$  的车辆数为 1，结果见表 3.10。

表 3.10 热力聚类数据计算第三步

聚类中心点经度	聚类中心点纬度	聚类 count 值
lng1	lat1	3

lng2	lat2	1
------	------	---

(4) 最后将此结果以 JSON 格式保存并放倒服务器上。

根据不同的数据参数整理得表格如下，聚类目标值为 K-means 算法中给定的 k 值，车辆数最大值为每个聚类比较后最大的 count 值，该值大致随 k 值增大而减小，结果见表 3.11

表 3.11 热力聚类数据计算第四步

聚类目标值	车辆数最大值（第 一次）	车辆数最大值（第 二次）	车辆数最大值（第 三次）
200	312098	302060	188653
300	180969	228644	150493
500	144908	114639	96329
800	114639	80303	114286
1000	116077	90165	116077

装  
订  
线

### 3.3.2 街区热力图计算

街区热力的区别在于，不需要网格或者算法来聚类，而是等同于用一个街区来作为聚类的依据，街区热力图的计算的关键在于如何判断一个点是否在一个街区内部，而判断依据是一个街区由一组点组成，要判断一个点是否在一个街区内部，等同于判断该点是否在某多边形内部，而给定的数据里对每个点包含了这个点所属的街区信息，这里要做的工作就是统计每一个街区的车辆数，并将 count 值相加。

最终数据为每个街区对应一条数据，包括这个街区的用来画出该多边形的点值，以及这个街区的热力值。

### 3.3.3 街区 OD 矩阵计算

街区 OD 的计算是本课题的重点，在传统调查中，由于成本高、工作量大等原因，OD 的计算往往很难精准，这使得传统的调查结果在准确性和代表性上有一定的局限性，而基于采集的数据，通过计算来得到 OD 矩阵的推导，是一种新的切实可行的方案。

在本课题的 OD 矩阵计算中，考虑到数据量和展示效果的需要，由于原始数据为 2017 年 4 月 1 日的全天数据，共 24 小时，因此将 OD 矩阵计算的时间间隔定位半小时，分时间段计算 OD 矩阵，并且计算的依据为，对某辆车，以 id 为固定量，而时间和所处街区为自变量，以该时间段内，该车第一次出现的街区为 0，最后一次出行的街区为 D，所得结果综合极为街区 OD 矩阵。

具体计算过程如下：

- (1) 根据原始数据进行时间过滤，每次计算时间间隔为 30 分钟，数据见表 3.12  
(stObjectId 为街区的 id)

表 3.12 街区 OD 矩阵计算第一步						
carId	isEmpty	ReceiveTime	longitude	latitude	stObjectId	
1	1	2015-04-01 23:44:46	121.461124	30.902831	156	
1	0	2015-04-01 23:57:12	121.466088	30.903425	99	
2	0	2015-04-01 23:34:41	121.462091	30.904532	142	
2	1	2015-04-01 23:42:01	121.464240	30.902003	142	

- (2) 得出车辆的移动 OD 矩阵，见表 3.13。

表 3.12 街区 OD 矩阵计算第二步

carId	origin	destination
1	156	99
2	142	142

- (3) 根据车辆 OD 算出街区的 OD 矩阵，见表 3.14（由于 id 为 2 的车一直在 142 街区内，等于这个街区的 OD 没有改变）。

表 3.14 街区 OD 矩阵计算第三步

origin block id	destination block id	count
156	99	1

- (4) 根据街区 id 与中心点对应表，将原街区 id 数据替换为街区中心点经纬度，然后将结果写入文件。数据见表 3.15。

表 3.15 街区 OD 矩阵计算第四步

id	name	lng	lat
221	01001 外滩街道	121.482311761	31.2372704462
156	16021 南桥镇奶牛场	121.277470822	30.7510147032
231	20802 新海农场	121.285064456	31.8130903558

- (5) 最后得到街区 OD 矩阵见表 3.16。

表 3.16 街区 OD 矩阵计算第五步

origin block	destination block	count
[30.9442771665, 121.456620224]	[31.0069872924, 121.394565008]	2

### 3.3.4 街区吞吐量计算

街区吞吐量基于街区 OD 矩阵的计算，街区 OD 矩阵记录了街区之间车流量的移动，记录了各个街区之间车辆的净流量，而街区吞吐量计算的是对每一个街区，将它所有的流入和流出相加，得到该街区的净吞吐量。

各个街区间的车辆净流量记录在 OD 矩阵中，有正负之分，正代表由 o 流向 d 的值更大，净值即为所记录 count 值，负代表由 d 流向 o 的值更大，净值亦为所记录 count 值。

### 3.4 本章小结

本章主要介绍了数据处理部分的工作。首先统计了目前使用的原始数据的数据情况和存在的一些问题。然后将出现的问题进行统计，分为若干种类型的问题，并针对不同的问题选择不同的处理措施；最后介绍了两种不同的处理数据的方法，一种是使用 Linux shell 脚本，另一种是使用 Java 进行编程，并给出了几种处理数据的流程。在下一章中，将详细介绍数据分析的过程和可视化的实现。

装

订

线

## 4 数据可视化部分

### 4.1 引言

在本课题中，在数据分析阶段，我负责研究的是数据可视化。由于在实际可视化场景中，数据的大小、分布、颗粒度，都极大影响着可视化结果的优劣，数据是否合理，处理方式是否正确，是否真实的展示出数据的特点，都是要考虑的因素。

因此，为了提高数据分析的效果，我们就需要确定什么是影响因素，也就是对数据进行处理的同时有权重的分析更为重要的信息。在数据中，往往有一些对后期分析（分类、聚类等）没有影响或者影响很小的属性，或者影响效果类似的属性，也就是噪音数据和冗余数据。按常识来说，在分析之前我们会将没有影响或影响很小的属性剔除，在影响效果相同或者类似的属性中选择一个，其他的剔除，从而使数据的属性变少、维度变低。

在城市诊断所涉及的各个系统中，以城市交通为主要对象的城市流动性诊断由于其量化特征明显，数据获取便利等特点，近几十年来一直是城市各系统中量化研究成果最为丰富，应用最为广泛的一项。而随着大数据时代的到来，新的数据类型及其处理技术的诞生为城市流动性研究提供了新的视角和方法，使得城市流动性诊断在整体性和动态实时性等方面具有了显著的提升空间。

随着城市的发展，城市规模逐渐扩大，其内部的交通方式日趋多元，交通网络日趋复杂，尽管可达性始终是影响城市内部功能布局的关键要素，但是对于城市内部可达性的定量研究却变得越来越困难。接下来我们将介绍本课题如何计算并渲染数据，以及关于可达性的研究。

### 4.2 课题需求

在本课题中，主要的数据为出租车数据与街区数据，因此在对出租车数据进行分析、可视化时，主要侧重点在于利用现有的出租车数据来分析城市里不同时间段内出租车的分布，以及与街区的信息关联后，街区内部出租车的数量，并且利用出租车的活动源自不同客户的需求这一特性，意味着出租车本身没有主观性，其活动路径可以较为客观的展示城市内人个体的流动，来制作城市的OD矩阵，分析不同街区之间车流量的流动情况，以及每个街区自身的车流动情况，用出租车的数据来一览整个城市的人流动情况，并能够用来对交通、城市规划等起到启发式的效应。

因此，我们的重点是热力图的展示和OD矩阵的展示，用上文计算好的数据来将结果可视化，并从中得到我们所需要的信息，如可达性等，正是本课题的研究重点，接下来会详细介绍可视化的过程及要点难点。

### 4.3 热力模型

热力图其实是最简单的数据可视化模型，原理也很简单，每一条数据都是一个点，包括（x、y、weight），而一条数据代表着一个热力点，也就意味着每一个热力点的数据都包含一个位置和权重，权重越大，这个热力点就要越明显，其实这也代表了渐变的一个衰变因素，而之所以会有不规则的区域，其实原理上就是各管各自的，然后各个热力点相互叠加影响导致，最终形成具有真实意义的热力图，制作热力图还需要关注的一点是半径属性，也就是缓冲区的半径，表示一个热力点的影响范围，在通常情况下，所有热力的影响范围都是一样的，只是权重不同，主要也是出于实现的效率考虑。

其实还有一点是数据的真实值问题，因为在实际中，很有可能不同的热力点差距很大，有可能有的热力点数据量非常大，而有的热力点数据为零，所以在不同级别的下的数据会有优化，比如数据量级差距较大的数据会缩小差距，而差距较小的数据可以直接用原始数据作为权重。

#### 4.3.1 网格数据热力模型

网格数据计算结果在上面已经介绍过，初步计算的网格热力权重即为网格的 count 值，目前的数据里，大致数据量见表 4.1。

表 4.1 网格数据数量级

网格边长	网格个数	单个网格最大值（车辆数）
0.003	15822	26952
0.005	7713	21791
0.01	2777	49528
0.02	993	66330

以半径为 0.02 画热力图时，效果如图 4.1：

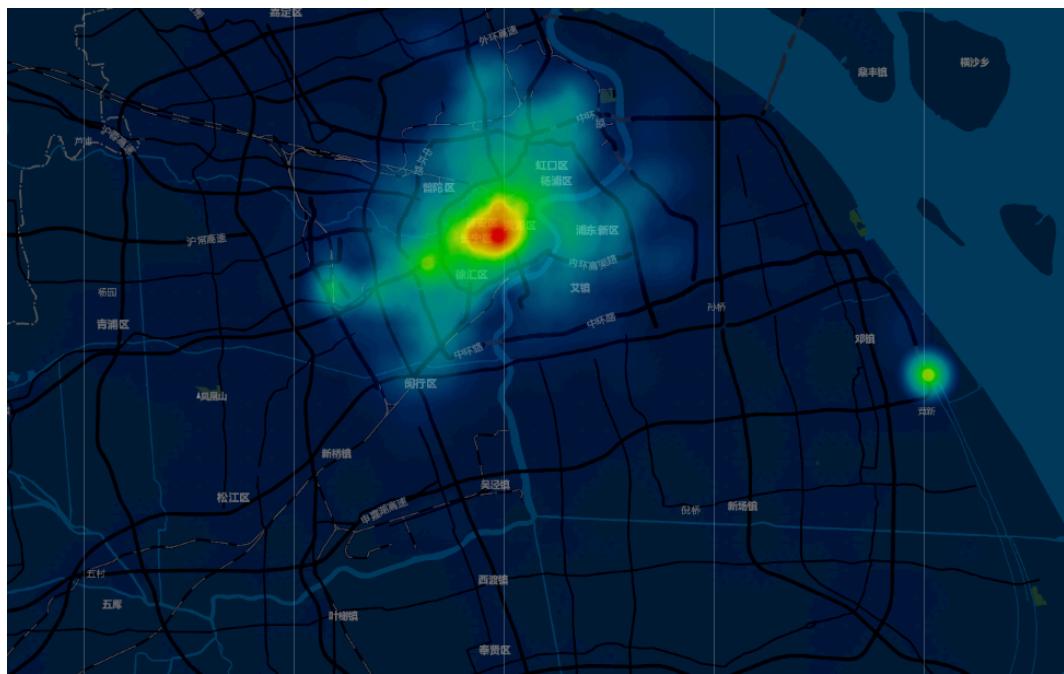


图 4.1 热力图 0.02 半径

可见，虽然网格很多，但由于少部分地区数据较大，导致这部分地区的集中区域颜色较深，而其他地方的热力效果不明显。同样的数据以 0.01 再画，效果如图 4.2：

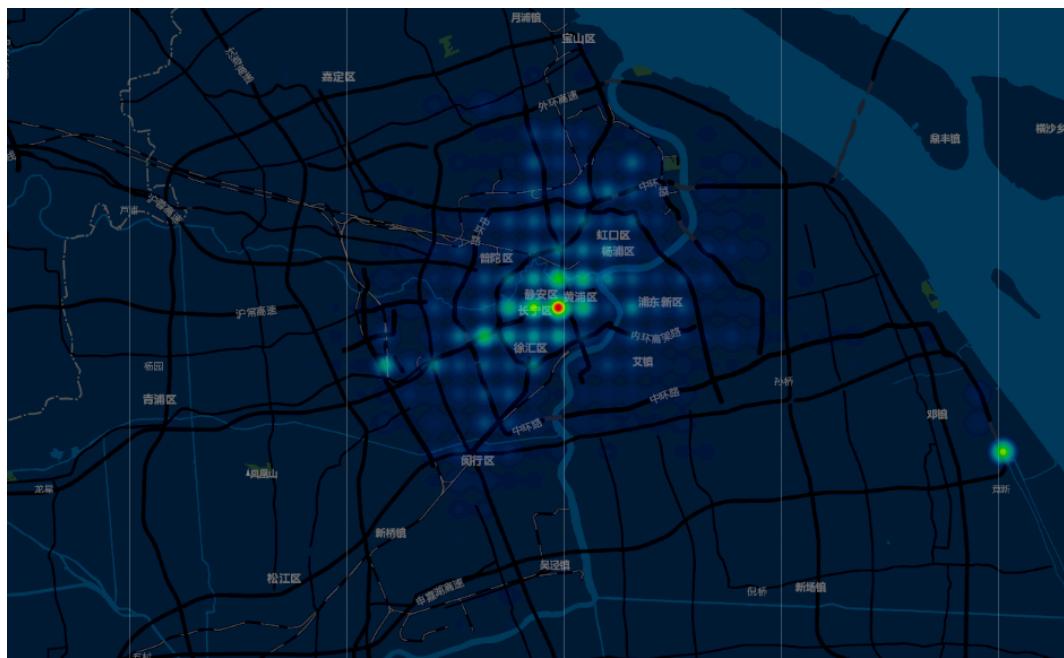


图 4.2 热力图 0.01 半径

原因很明显，因为个别网格的热力值太大，剩余网格的热力值太小，导致整热力图的失衡。

从数据文件中拿出几个热力点来，尝试看一下这时的 JSON 数据情况：

```
{"lat":31.12000000268221,"lng":121.39999999687075,"count":11773},
 {"lat":31.12000000268221,"lng":121.420000000149,"count":10334},
 {"lat":31.12000000268221,"lng":121.43999999970197,"count":2224},
 {"lat":31.12000000268221,"lng":121.45999999925493,"count":3388},
 {"lat":31.12000000268221,"lng":121.479999998079,"count":292},
 {"lat":31.12000000268221,"lng":121.49999999836086,"count":2638},
 {"lat":31.12000000268221,"lng":121.51999999791383,"count":222},
 {"lat":31.12000000268221,"lng":121.5399999974668,"count":871},
 {"lat":31.12000000268221,"lng":121.55999999701976,"count":2034},
 {"lat":31.12000000268221,"lng":121.58000000029801,"count":2313},
 {"lat":31.12000000268221,"lng":121.6999999976158,"count":182},
 {"lat":31.12000000268221,"lng":121.71999999716877,"count":1},
 {"lat":31.12000000268221,"lng":121.75999999999999,"count":514},
 {"lat":31.12000000268221,"lng":121.77999999955296,"count":3},
 {"lat":31.12000000268221,"lng":121.79999999910592,"count":2},
 {"lat":31.12000000268221,"lng":121.81999999865889,"count":50169},
```

可见，count 值的差异非常大，最小的为 1，而最大的有 50169，这就导致在渲染热力图的时候各个热力点间非常不平衡，导致整个图在渲染的时候很多热力值比较小的点直接看不出来热力，而某几个点会变得异常的明显，超过了我们所需要的效果，因此首先对数据做出修改，这里第一次尝试时，设想用热力值的数量级来代替热力值，见表 4.2。

表 4.2 数量级转换

原始值	转换值
10	1
50	2
100	3
200	4
500	5
800	6
1000	7
2000	8
5000	9
10000	10
20000	11
50000	12

将全部数据按此对应关系进行转换后得到新数据，将其用半径为 0.02 时的热力图

画出来后如图 4.3：

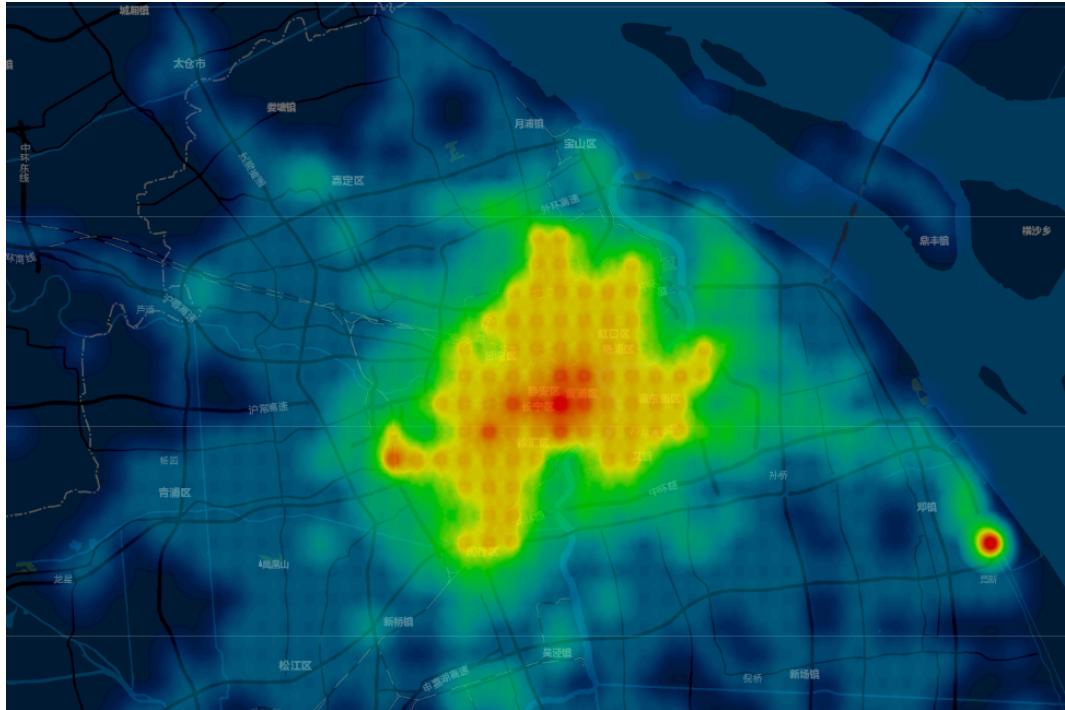


图 4.3 转换后数据热力图 0.02 半径

可以看出，所有区域都有自己的热力值显示，且中心地区颜色较深，复合预想。但是现在的问题是按这个方法得到的热力图，容易陷入局部最优解问题，因为不同时段的峰值不同，如果要按这个方法，意味着每次都要对数据按这个数据的情况进行一次预算，而且这样计算的主观性影响太大，当一个数据里的 1000~5000 的数据较多时，就意味着要在这个区间内多设置几个值，例如：1000、2000、3000、5000 分别各为一个数量级，如果没有这样做，而是将这部分的数据全部作为一个数量级，就意味着看不到这部分的数据情况，而解决方案是手动计算出最值，然后用最值来设定数量级的大小。

但是很多时候热力图不够直观，因为一些距离较远而颜色相近的点无法精确的体现差别，于是开始尝试加一种新的模型，立方体来配合热力的显示，由于原来的画面是二维的画面，因此尝试加的三维的立体模型需要适中并且能够较好的渲染三维模型。

而所谓的三维模型其实相当于原本是热力点的地方换成了一个立方体，并且立方体有自己的数值，这时的数值就不能用数量级，而要用原始数值，意味着要同时准备 2 份数据来分别渲染热力图和热力立方图。因为数量级的话，不同的热力点之间差别太小，并且不同的数值会得到相同的数量级也就意味着原本不同的数值会导致相同的立方体，所以，这里使用 2 份数据。

可以看一下立方体的效果和同样数据的热力点效果如图 4.4:

装  
订  
线

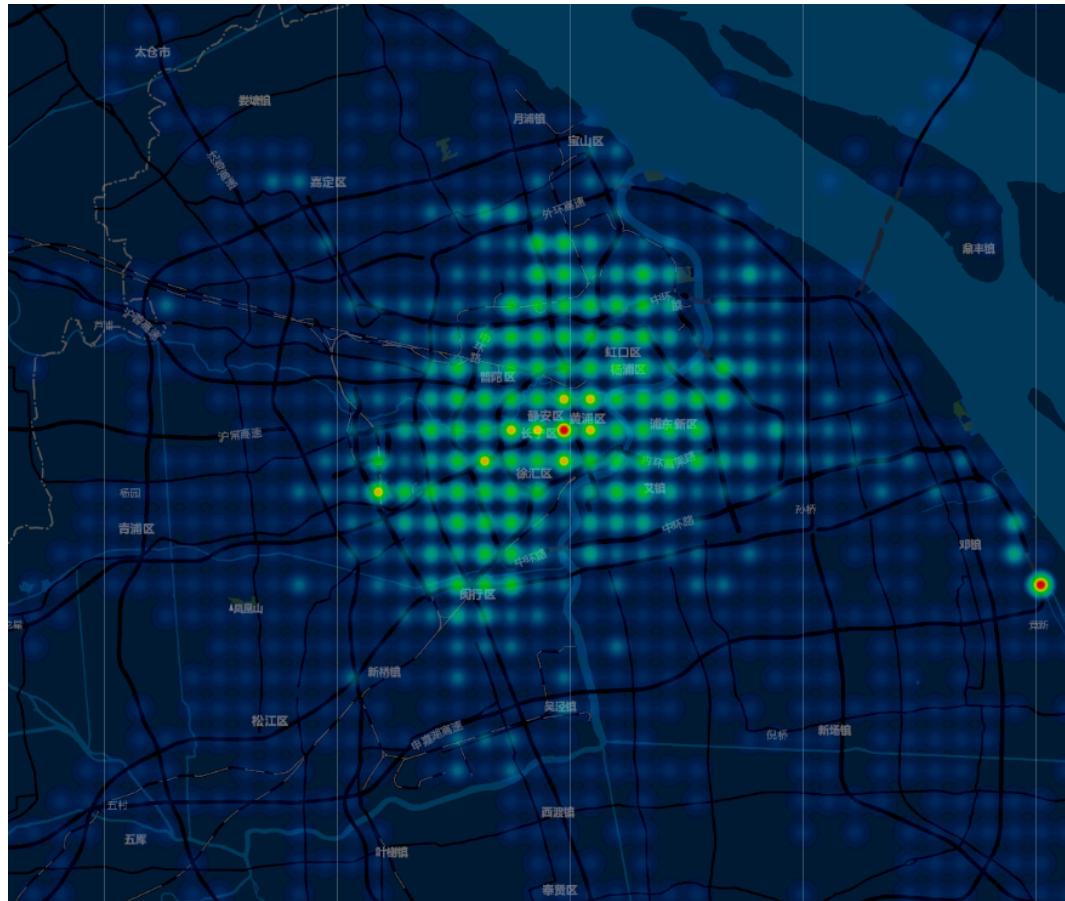


图 4.4 立方体底层热力图

相同数据的立方体效果如图 4.5:

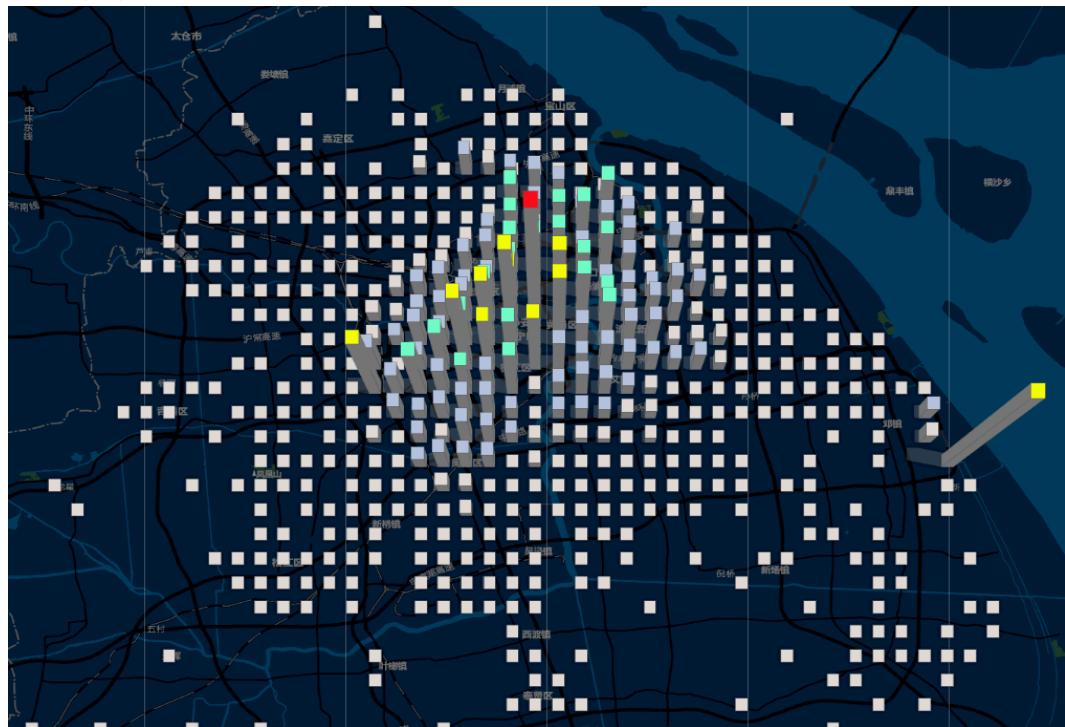


图 4.5 立方体图

可以看到，等价于立方体的效果知识把热力点纵向拉伸了一个纬度，但是效果好于热力，可以更直观的看到不同网格之间的差距。

在本课题中，基于热力图和热力立方做了集成，这一点需要注意的是，在视觉冲击上，肯定是立方体需要更“高”，意思是离用户的眼球距离更近，也就是立方体需要离摄像机的位置更近，而原本热力图在视觉上等于是“叠”在地图上，也就是位置更低，那么就需要在编写渲染代码的时候，先渲染热力图，再渲染立方体，由于我们需要 2 份数据，也就是说先请求热力图所需的数据，然后请求立方体所需的数据，并在数据拿到后渲染，要确保渲染的顺序。

在此还做了一部分工作是，对热力立方的显示与它本身的热力值做关联，热力立方的高度肯定是热力值的原始数值，在这个基础上尝试更改热力立方的立方体颜色为底层热力图的颜色，如图 4.6：

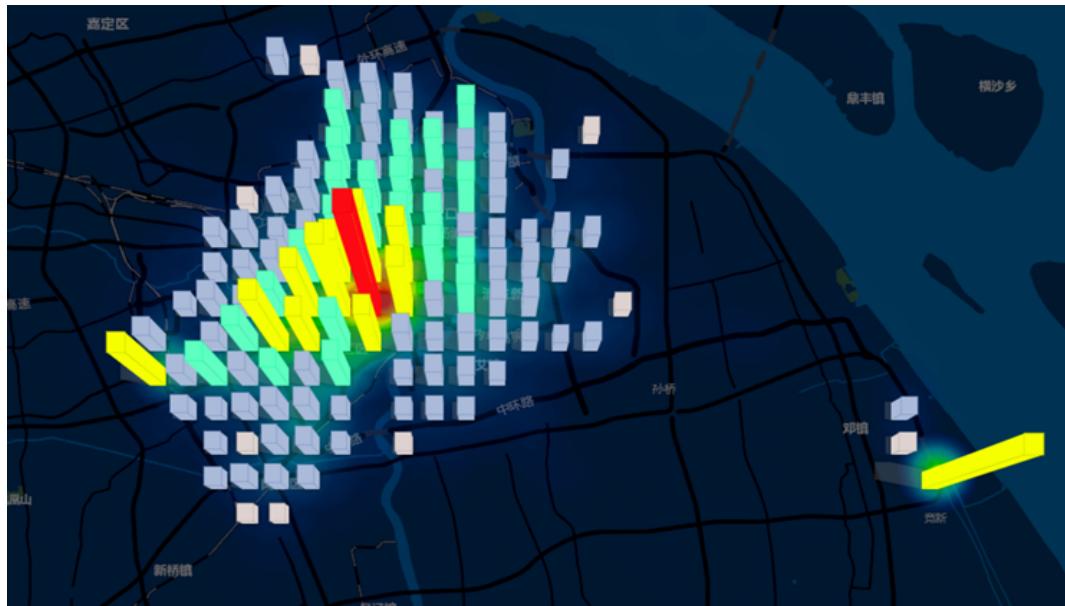


图 4.6 变色立方体

我们发现这样的效果会造成整个画面太过混乱，因为颜色太杂会导致画面的失衡，让用户无法将注意力集中到热力本身，最后的解决方案是保持热力立方体的四个柱面不变，而将热力立方体的顶层颜色改成与底层的热力图的颜色保持一致，计算关系是根据整个数据的最值来计算，按最值来判断每一个热力原始值相对最值的百分比，然后根据百分比来得到对应的颜色，对应关系见表 4.3。

表 4.3 立方体顶部颜色对应表

百分比	颜色码	颜色
0~1/6	#DCD2D0	
1/6~2/6	#67FEC1	

2/6~3/6	#AABAD7	
3/6~4/6	#F4FE00	
4/6~5/6	#F57D0D	
5/6~6/6	#FA0E1B	

最后效果如图 4.7:

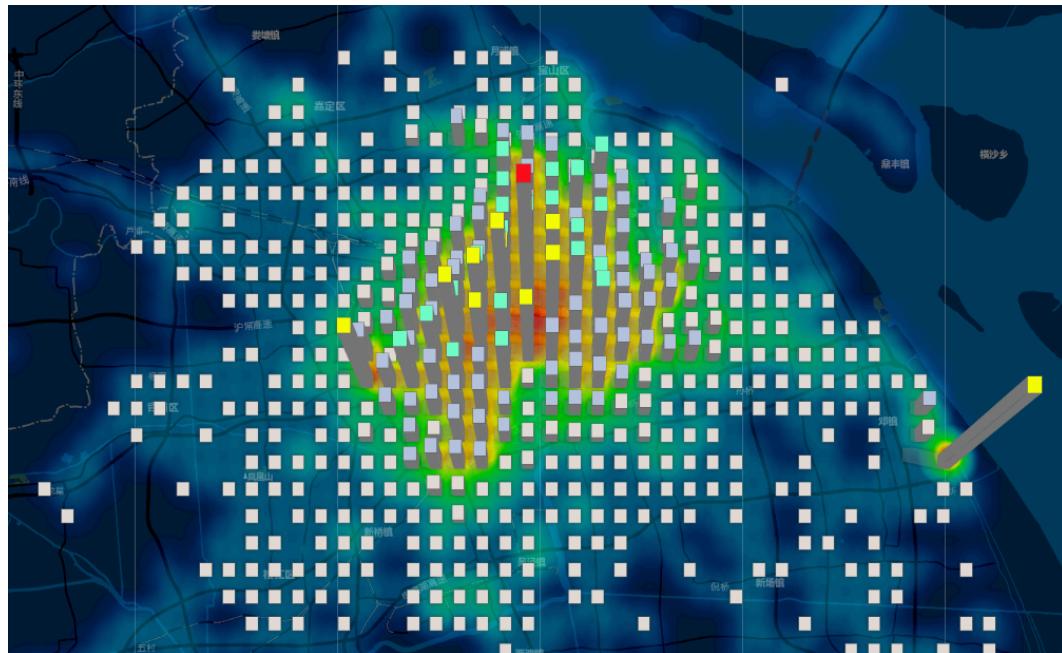


图 4.7 立方体加底图

可以看到底部有热力的显示，可以大致判断该区域的车辆密度，并且结合热力立方的显示，能够较为清晰的比较不同点及临近点的区别。

#### 4.3.2 聚类数据热力模型

聚类数据热力的展示与网格数据热力的展示类似，所不同的是，由于聚类的数据每次结果不同，因此无法确定到底哪一次的效果最好，只能多次尝试，以达到最好的效果。

聚类的标准是经纬度的信息，也就意味着理想情况下，距离较近的点会被聚为一类，车辆数相加，但是由于每次的数据不同，每次的起始点不同，导致每次的结果也不同。

由于我们采用的聚类算法为 K-Means 算法，需要提前给定 K 值，而在不知道最优值为多少的情况下，提前给定 K 值不一定准确，而且，考虑到热力图的渲染效果，如果聚类的结果太少，即可以用来渲染的热力点太少，是无法有较好的热力效

果的，因此，根据经验，将聚类的 K 值最小定为 200，并且逐渐增加，来测试效果，可得数据，如前文所提的见表 4.4。

表 4.4 聚类测试数据表

K 值	最大值（第一次）	最大值（第二次）	最大值（第三次）
200	312098	302060	188653
300	180969	228644	150493
500	144908	114639	96329
800	114639	80303	114286
1000	116077	90165	116077

依次效果如图 4.8~4.16:



图 4.8 K 值为 200 时第一次

装  
订  
线

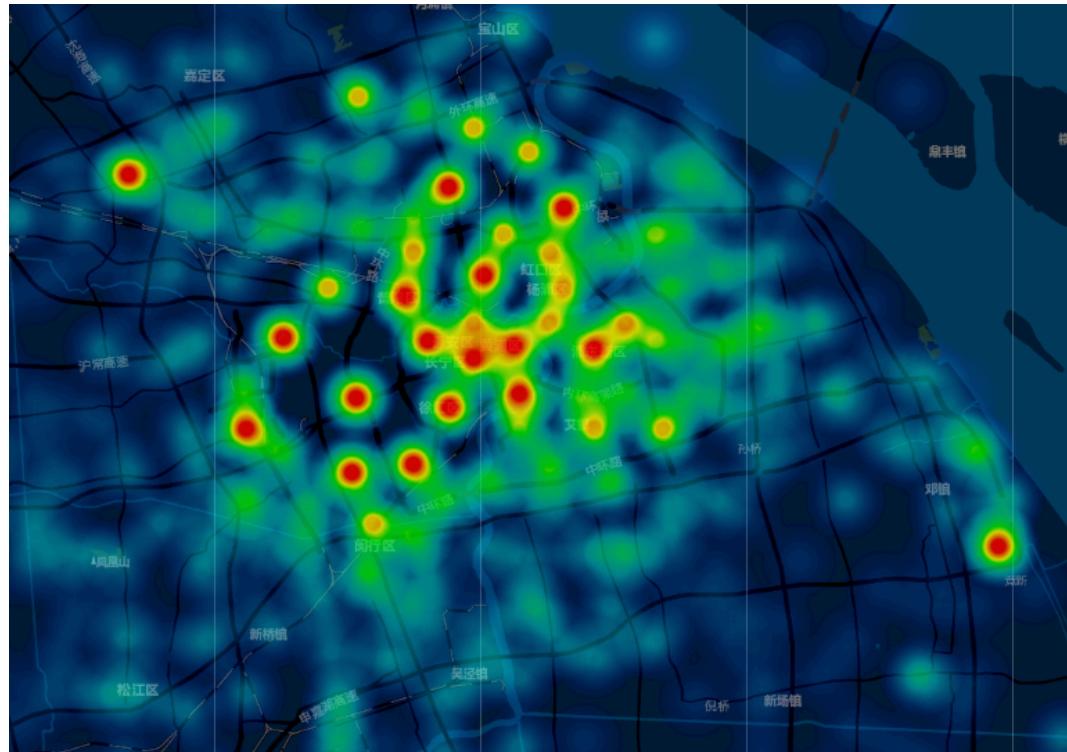


图 4.9 K 值为 200 时第二次



图 4.10 K 值为 200 时第三次

装  
订  
线



图 4.11 K 值为 300 时第一次



图 4.12 K 值为 300 时第二次

装  
订  
线

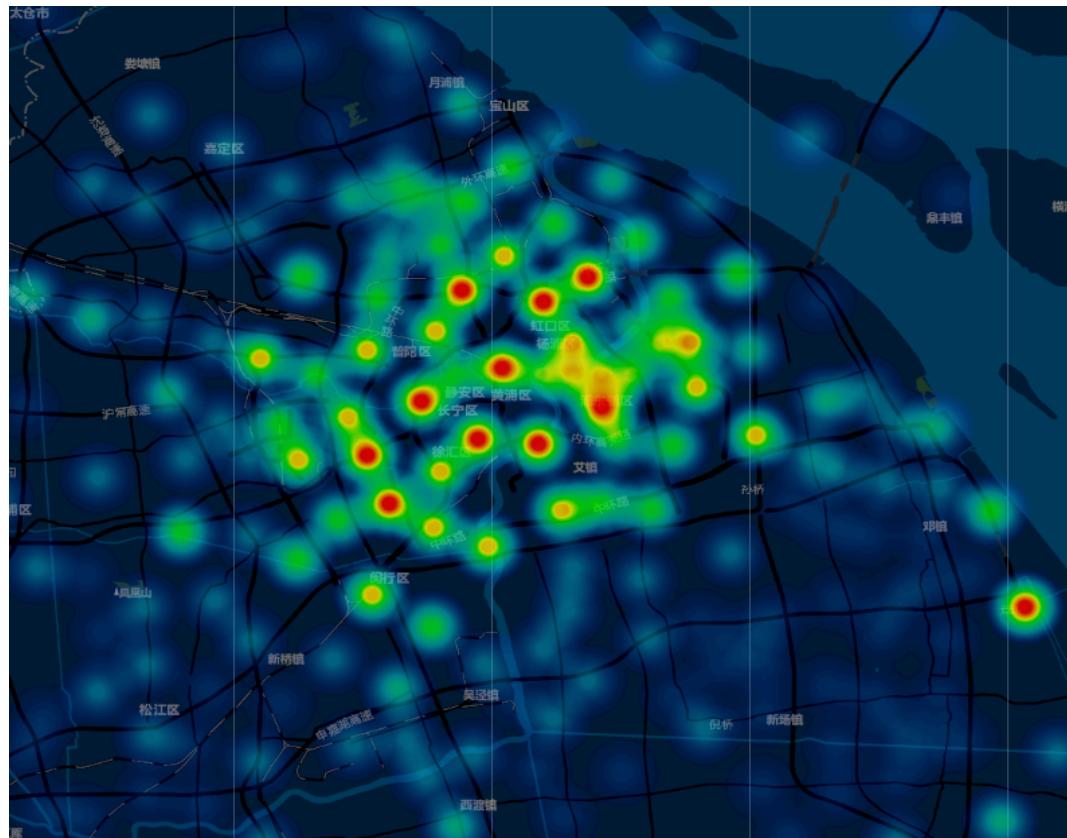


图 4.13 K 值为 300 时第三次

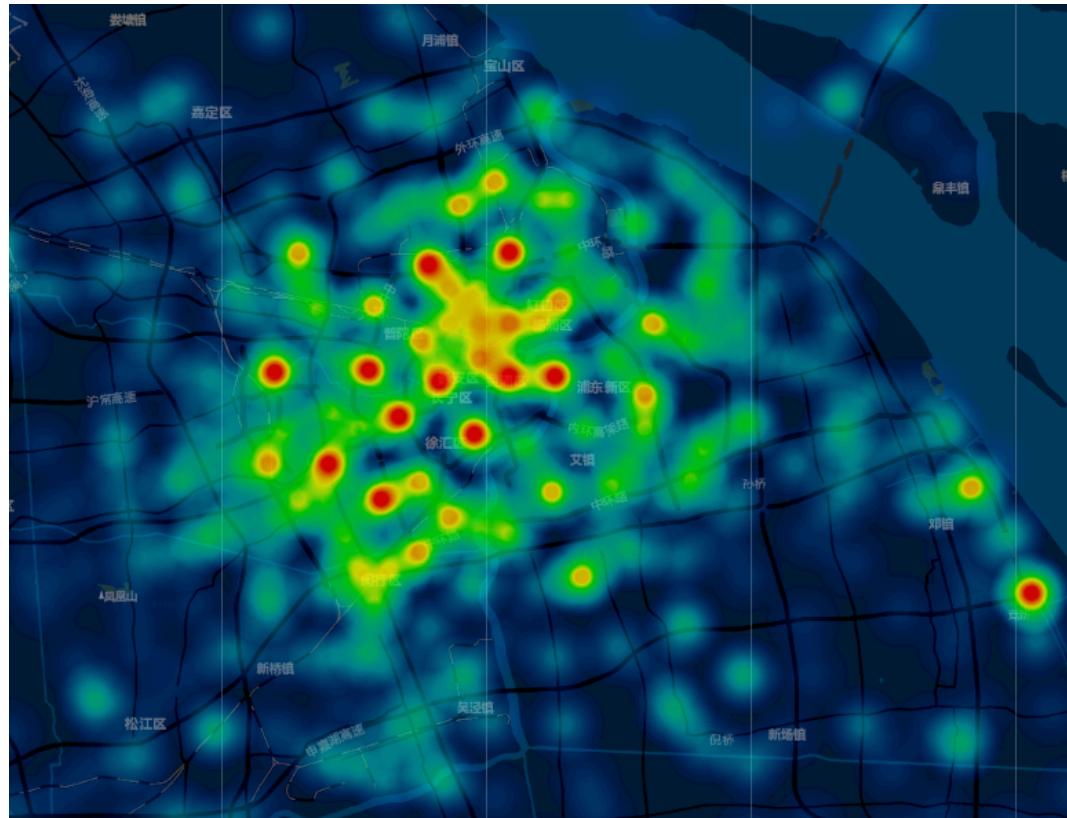


图 4.14 K 值为 500 时第一次

装  
订  
线

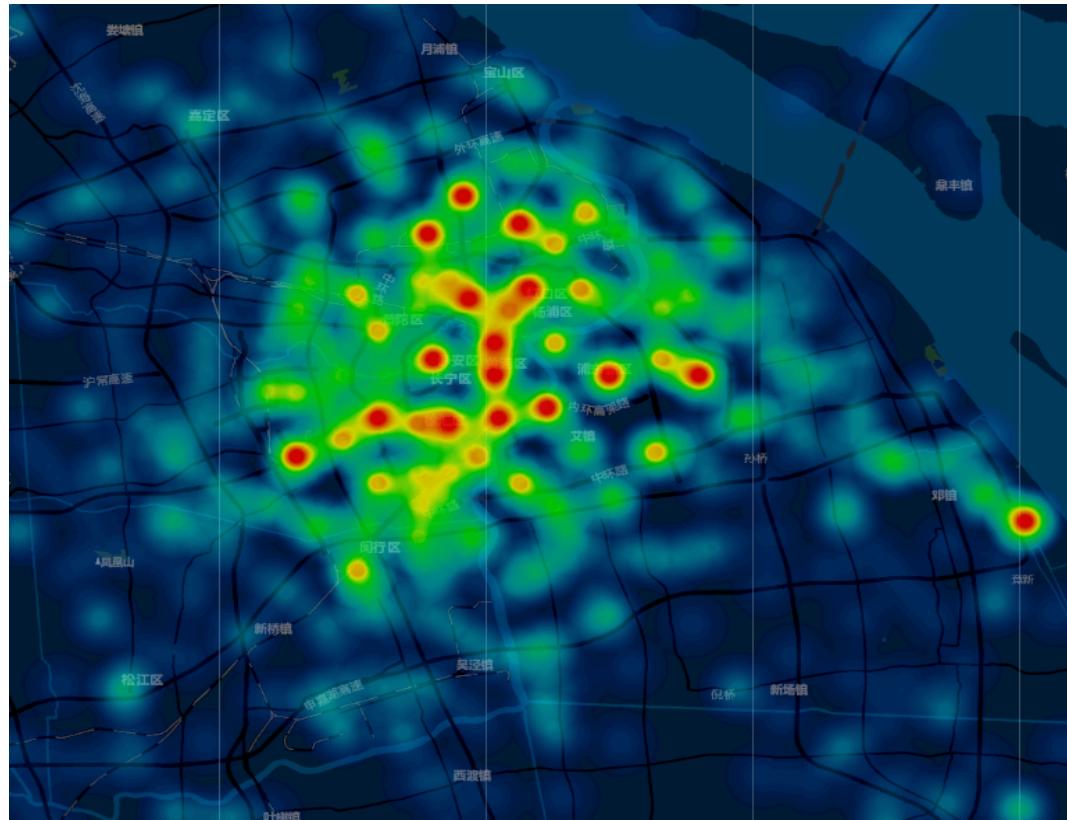


图 4.15 K 值为 500 时第二次

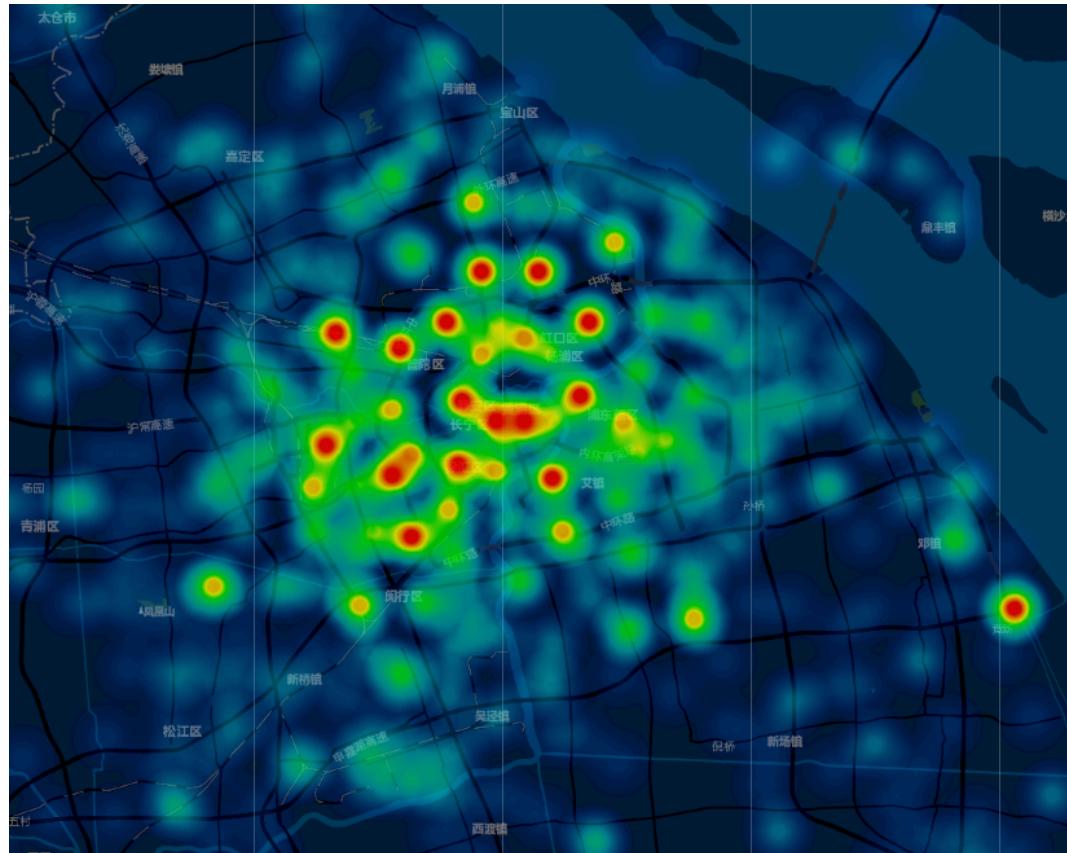


图 4.16 K 值为 500 时第三次

可见，当  $K$  值为 300 或 500 时效果较好，且较清晰的展示出热力分布，不足的地方仍然是不确定性，只能说，随着  $k$  值增大，聚类增多，最后结果相对趋于稳定，但是仍有较大差距，且随着  $K$  值的增大，每一次聚类的时间和性能的要求都会加倍，这对数据实时处理也提出了挑战，只能说，聚类相对来说，是个不错的选择，但如何合理应用，在  $K$  值大小和热力点的多少之间，即运行效率和渲染效果之间取一个平衡，是仍待探索的问题。

#### 4.3.3 街区数据热力模型

街区热力模型与传统热力图有所不同，街区热力模型很大部分取决于街区的划定，在街区边界内部的即属于街区的热力值，不同街区之间没有联系，也不需要聚类，因为街区的设定就等于划定了聚类的依据，所以说，街区热力图可以看作是聚类热力的一种延伸，属于特殊的聚类。但是街区热力在实际应用中其实很广泛，最为经典的就是那张出名的美国的失业率分布。

如图 4.17：

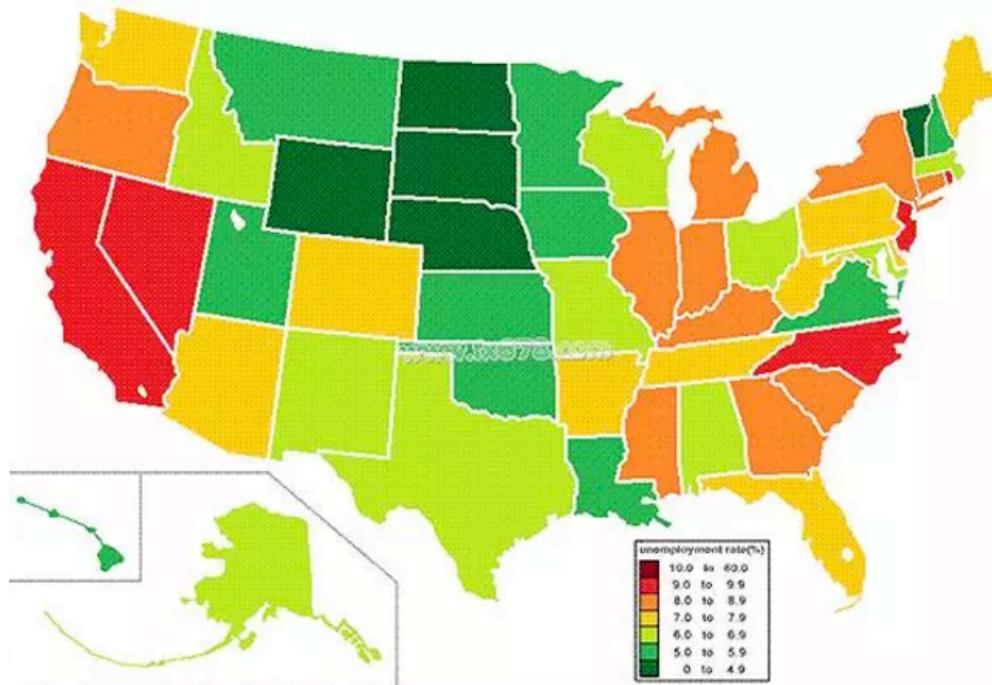


图 4.17 美国失业率分布

而我们所要完成的街区热力与之类似，不过做法不一样罢了，首先确定时间间隔，然后按时间间隔计算该时间段内每个街区出现过的车辆数量，统计并相加每个街区中的车辆数，得到这个街区的热力值，这时重点就放在了渲染策略上，用什么样的色阶来映射效果较好是需要关注的地方。尝试过的配色见表 4.5：

表 4.5 黄绿色系配色

热力值	颜色码	颜色
0~500	#e2f10a	黄色

500~1000	#e1ec3e	
1000~2000	#e5ef63	
2000~3000	#eef58f	
3000~5000	#f6fbae	
5000~10000	#f1f5bf	
10000~20000	#e9ecc2	
20000~50000	#eff1dd	
50000~无穷	#f0f1e9	

效果如图 4.18、4.19:



图 4.18 黄绿色系街区热力

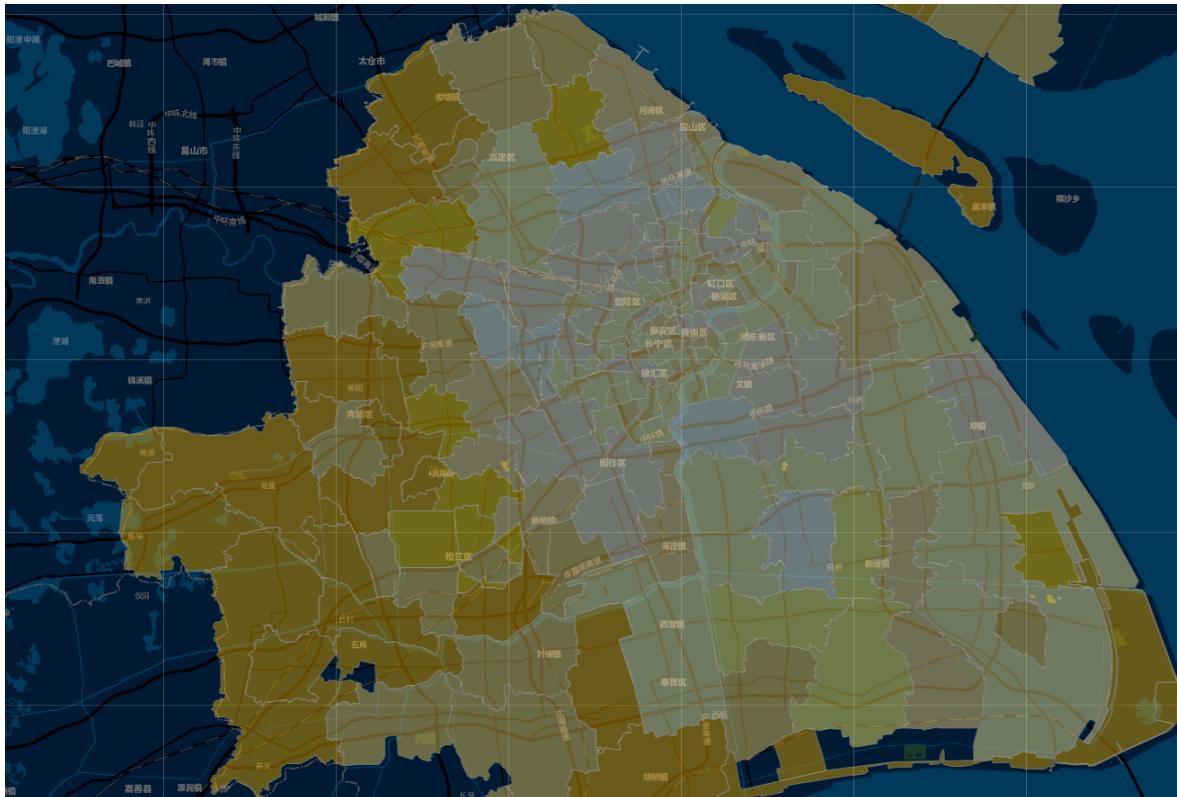
装  
订  
线

图 4.19 黄绿色系顺序倒置

黑灰色系配色见表 4.6:

表 4.6 黑灰色系配色

热力值	颜色码	颜色
0~500	#1C1C1C	
500~1000	#363636	
1000~2000	#4F4F4F	
2000~3000	#696969	
3000~5000	#828282	
5000~10000	#9C9C9C	
10000~20000	#B5B5B5	
20000~50000	#CFCFCF	
50000~无穷	#E8E8E8	

效果如图 4.20、4.21：



图 4.20 黑灰色系



图 4.21 黑灰色系顺序倒置

最后发现，由于底图为暗色系，采用黑灰色系容易和底图混淆，上线版本中采用黄绿色系，但是具体色值还有待改进。

#### 4.4 街区吞吐量模型

OD 矩阵的计算在上文中已经提到，计算好的 od 矩阵主要包含街区的信息和 od 值，因为 OD 矩阵的单位是街区，也就意味着要以街区为最小单元来渲染 OD 矩阵，这意味着在渲染 OD 矩阵的同时，要先渲染街区的信息，于是，每一个街区的边界就成了需要的数据，也就是说，在画 OD 线之前需要先把街区的区块信息画出来。

这一步的初步想法是在画街区区块信息的时候可以直接以街区的吞吐量作为街区 OD 的底图，街区吞吐量的计算在上文也已提到，因此，这里先介绍街区吞吐量的渲染方法。

吞吐量主要来自于街区 OD 矩阵中的值相加，对每一个街区，都有各自的吞吐量的值，有正有负，这里尝试是吞吐量为正时，用红色系，吞吐量为负时，用蓝色系，且随着值的增大而加深，随着值的减小而变浅，借鉴了街区热力的理念，但表示的是不同的数值，也代表了不同的意义。具体计算的值依据是每当有这个街区的 OD 发生时，这个街区的净吞吐量就会被改变，每个街区都有自己的净吞吐量，这个吞吐量计算的也只是这段时间内的吞吐量，所以这幅图所代表的也就是某一时间段内各个街区所发生的车辆流动信息，在赋予颜色时的依据是，找出所有街区的净吞吐量的最值，对每个街区，它的吞吐量占最值的百分比，按百分比来划分颜色区块，所用的颜色表整理得表格。

红色系见表 4.7。

表 4.7 红色系配色

数值	颜色值	颜色
0~1/8	#3cb0e8	
1/8~2/8	#48b4e8	
2/8~3/8	#5abbea	
3/8~4/8	#6ec2ea	
4/8~5/8	#89cff1	
5/8~6/8	#a4d9f3	
6/8~7/8	#b6e1f7	
7/8~8/8	#c4e3f3	

蓝色系见表 4.8。

表 4.8 蓝色系配色

数值	颜色值	颜色
0~1/8	#d45050	
1/8~2/8	#d25e5e	
2/8~3/8	#d26f6f	

3/8~4/8	#d48181	
4/8~5/8	#d08d8d	
5/8~6/8	# d09999	
6/8~7/8	# d0a5a5	
7/8~8/8	# ceb0b0	

用不同深浅的蓝色系和红色系分别代表不同程度的净吞吐量，颜色越深，说明在该时间段内这个街区的车辆流动越大，颜色越浅，说明该时间段内，这个街区的车辆流动性较小。

最后结果如图，分别有 2 种方案，第一种颜色较为饱满，更引人眼球，并且透明度更低，且边界明显，第二种，透明度高，颜色更淡，边界的线较细，分别有不同的展示效果，用第一种可能更直观的展示吞吐量的大小，但是过于明亮的色值会导致整个图的不均衡，以及遮盖住大部分底图的信息，第二种可能没那么显眼，但对用户更友好，也更多的保留了底图的信息。

亮色系见图 4.22，暗色系见图 4.23：

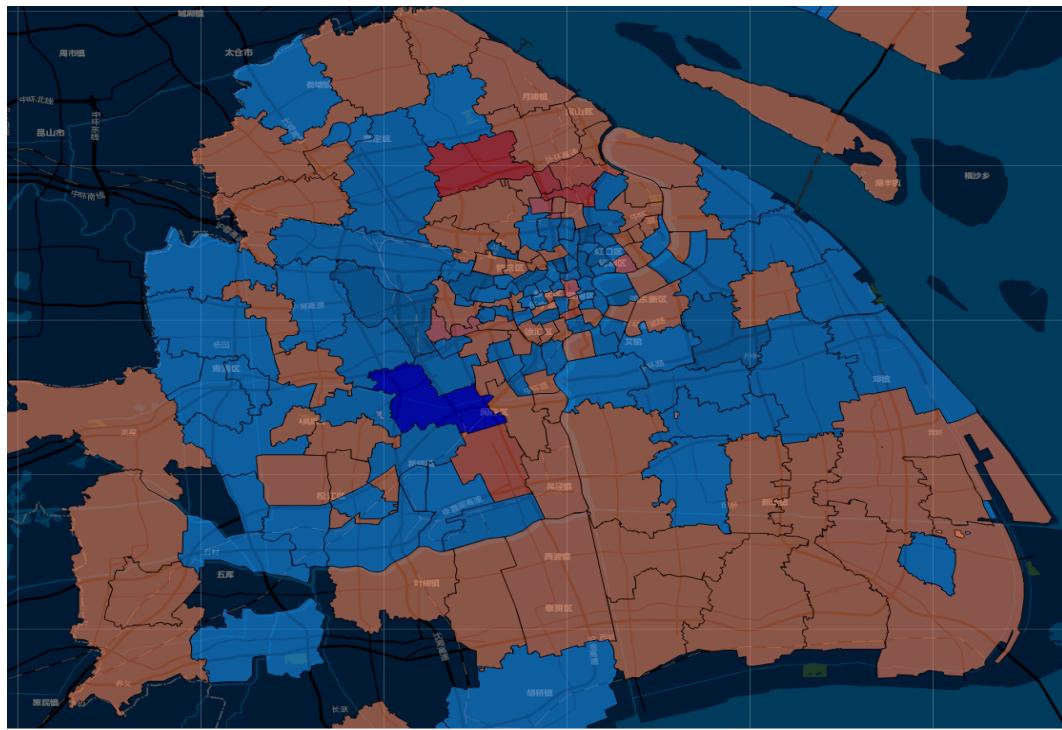


图 4.22 亮色系

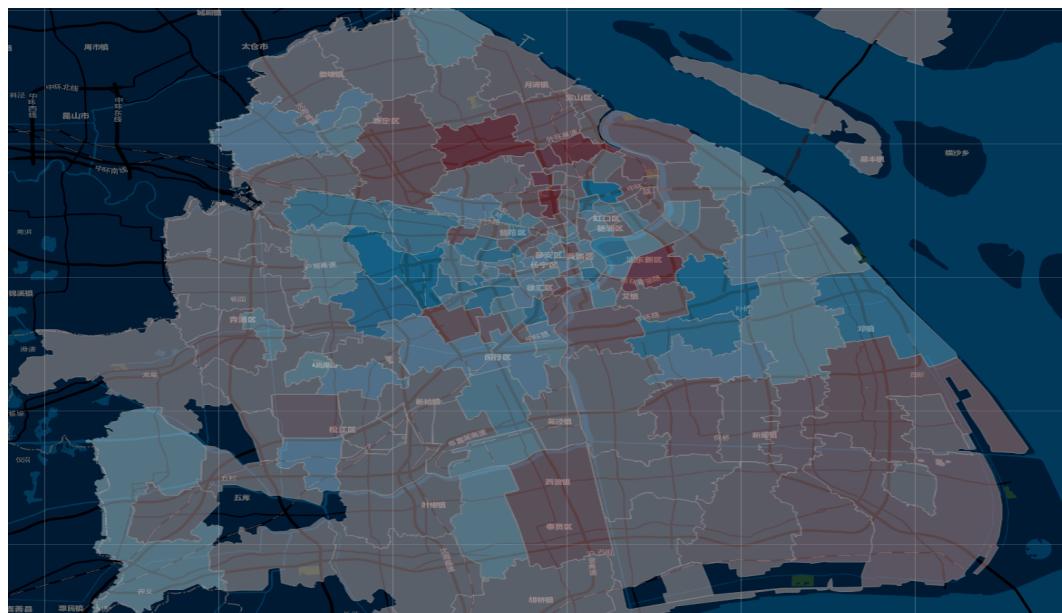


图 4.23 暗色系

可以看到，图中不同的街区有不同的颜色，分别有红色块和蓝色块，颜色越深，代表这个街区的净车流动越大，越小，代表这个街区的净流量越小。

#### 4.5 街区 OD 矩阵模型

OD 矩阵的重点在于画出 OD 的线并表示方向及权重，OD 权重的计算已在上文中说明，关键在于用什么样的形式来表示权重以及不影响 OD 本身的效果，由于街

区之间的距离取决于街区的信息，因此，od 线的长度无法更改，只能修改线的样式来达到预期的效果。

#### 4.5.1 街区 OD 矩阵渲染

这里的渲染方案包括 3 点。

一是修改线的颜色，按照 od 值的大小来控制 od 线的颜色的明暗程度，越大的 od 值，那么这条线的颜色就要越亮，如果 od 值越小，那么这条 od 线的颜色就越暗，也就是说，将 od 矩阵的线的亮度与 od 值成正比。而在设定 OD 线的颜色时，依据取决于当前所有 OD 线的最值，拿到所有 OD 值的最大值后，每条 OD 线都会有一个对应的百分比值，根据这个值可以得到对应的 OD 线所要赋予的颜色。

OD 线的颜色对应色系表见表 4.9。

表 4.9 OD 线配色

百分比	颜色码	颜色
0~1/9	#E8A600	
1/9~2/9	#F5AB00	
2/9~3/9	#F2CA5C	
3/9~4/9	#FCD200	
4/9~5/9	#FFEF61	
5/9~6/9	#FFDF78	
6/9~7/9	#FFEF94	
7/9~8/9	#FFEBB5	
8/9~9/9	#F7F7D2	

也就意味着，越是数值大的 OD 线，会变的越亮，而越为数值小的 OD 线，会变得越暗。

二是修改线的粗细，与颜色同理，按照每一条 OD 线的 OD 值占所有 OD 线中的最值的百分比作为依据，值越大的线，宽度越粗，而值越小的线，宽度越细，这样 OD 值较大的线，会变得既宽又亮，而 OD 值越小的线，会变得既暗又细。

第三点也是最重要的一点，修改渲染顺序，如果按照原始数据的顺序，直接渲染，很可能本来是最亮最粗的 OD 线被其他的线所遮挡，这样既不利于效果展示，也不是良好的用户体验。因此，还需要做的一步就是按照 OD 值的大小，依次渲染

OD 线，值越大的线，越放在最后才渲染，这样画出来的 OD 图，能够清晰的看到 OD 值比较大的线，而不被不重要的线所挡住。

并且，在编写这部分代码的时候，出于代码耦合度考虑，特意留了一个变量，为 OD 线的最小值，默认是 0，当小于这个值的时候，不渲染这条 OD 线，这样，就可以根据需要，来渲染 OD 矩阵，并保证渲染的效果，可以根据需要而调整。

#### 4.5.2 街区 OD 矩阵效果图

结果如图 4.24、4.25，分别过滤了 od 值在 3 以下的 OD 线，即最小 OD 值为 3

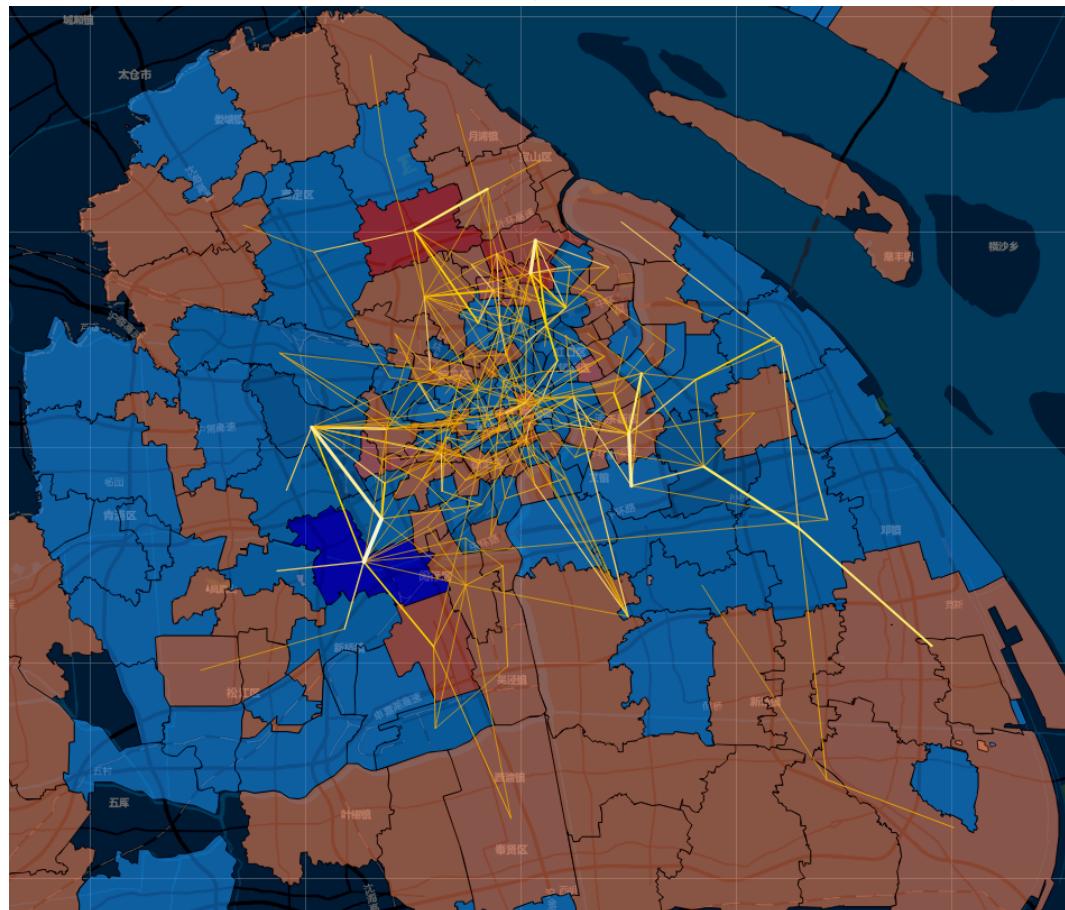


图 4.24 最小 OD 值为 3 亮色版

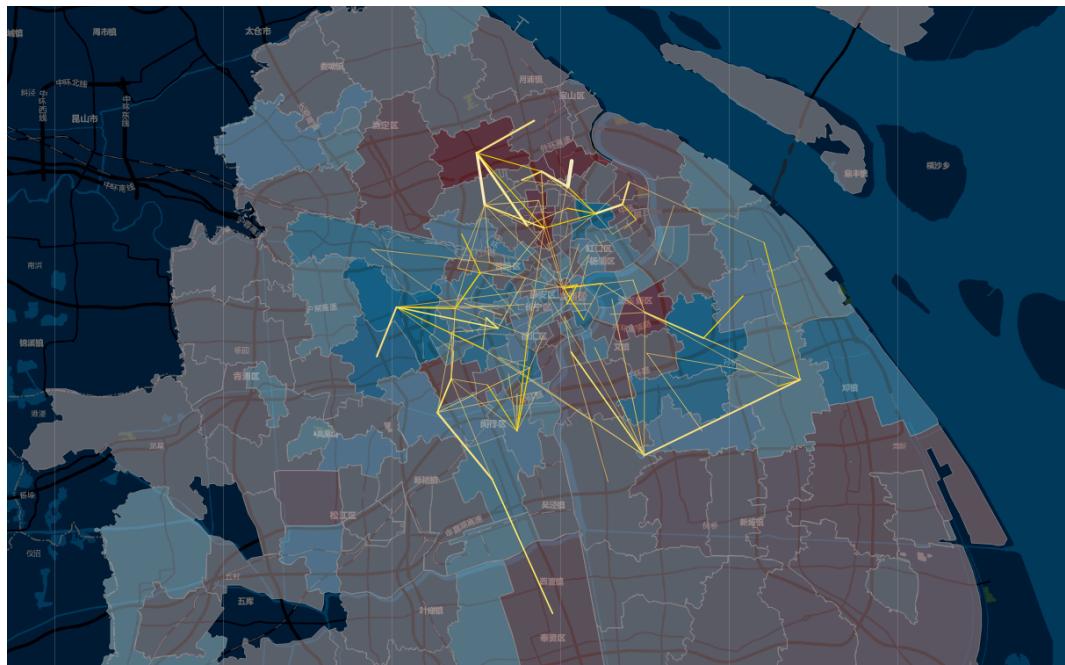


图 4.25 最小 OD 值为 3 暗色版

将最小 OD 值设为 0 时效果如图 4.26、4.27:

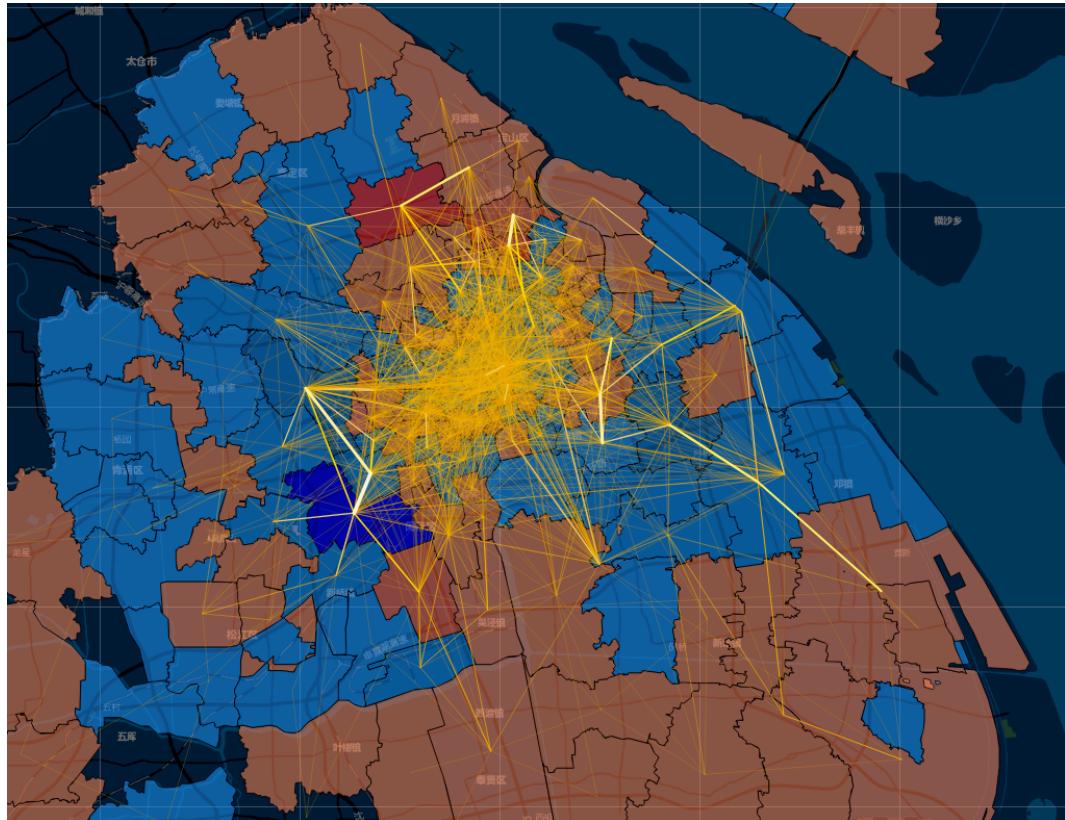


图 4.26 最小 OD 值为 0 亮色版

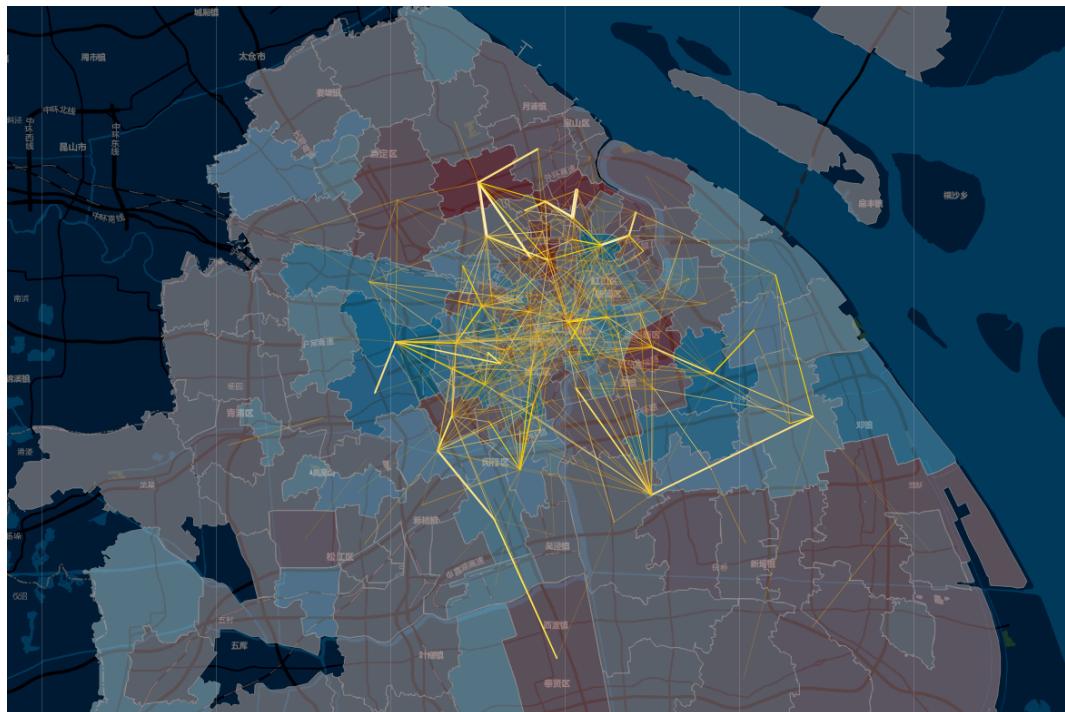


图 4.27 最小 OD 值为 0 暗色版

装

订

线

可以看到基本所有的街区之间都有 OD 线段。

#### 4.6 街区 OD 矩阵分析与应用

OD 矩阵的意义首先在于交通分析，切实可行的途径是基于采集数据，完成 OD 矩阵反推，进而获取一般路网或局域路网的交通信息。OD 矩阵的结果应用方面很广，包括公路网规划，新建或改建项目可行性研究、设计、交通组织及管理等各方面，大量的 OD 调查数据，对远景交通量的预测、道路类型及等级的确定、互通立交的设置、道路横断面的设计、交通服务设施的配置、交通管理与控制、规划方案和建设项目的国民经济评价、以及财务分析等提供了定量依据，进而为交通规划的完善和建设项目的科学决策奠定了基础。

OD 矩阵还可用于分析出行成本，即在网络中查找和测量从多个起始点到多个目的地的最小成本路径，可以指定要查找的目的地数目和搜索的最大距离，尽管 OD 矩阵不输出沿网络的线，但是存储在“线”属性表中的值却反映了网络距离，而不是直线距离 OD 矩阵分析的结果通常会成为其他空间分析的输入，在这些空间分析中，网络成本比直线成本更适合分析。例如，预测建筑环境中的人员流动更适合采用网络成本模型，因为人们一般在道路和人行道上行走，通过对人流量 OD 的矩阵分析即可得到人流量的预测。

#### 4.7 可达性量化模型的分析与应用

从可达性的量化方法上来看，现有研究常用的模型包括了距离模型、机会模型、潜能模型、效用模型和时空棱柱模型等，这里简要介绍几种常用模型。（智慧城市，可达性）

##### 4.7.1 距离模型

距离模型将任意两点之间的距离(可以表达为空间距离、时间成本等)作为单一要素来评价两点之间的相对可达性，而某一点在特定区域范围内的综合可达性则是该点与区域范围内其他点之间距离的集合或均值。

在本课题研究中，这种方法的优点在于简单直观，应用方便，受条件制约较少，根据前文所描述的 OD 矩阵可视化，每条 OD 线的长度即可用来类比距离，根据不同 OD 线的长度，可以直接映射为不同的距离模型，在这种模型下，意味着位置较为中心的街区，可达性必然高于位置较为偏僻的街区，可以利用现有数据直接计算的是，街区中心点之间的距离，并将之作为距离模型的结果。

而其不足则在于过于简化影响要素，忽略了每个点本身的引力差异、距离衰减特征、土地使用功能等因素对可达性的影响。例如有些位置较中心的地方，但是实际上的车辆活动比较少，但得益于它的位置优势，由距离模型计算得出的可达性必然高于一些更为偏远但是实际上车流量更大的地方，这就出现了偏差。因此如果只是简单的距离模型，并不能完美的计算可达性，还需要与其它模型相结合。

##### 4.7.2 机会模型

机会模型主要考察从某一点出发，在一定出行成本约束下所能够到达的其他需要到达的点(机会点)的数量，数量越多则意味着该点的可达性越强，出行成本可以由空间距离，交通时间或者交通经济成本等条件进行控制。

该模型的优点在于直观地将可达性转化为机会点的数量，易于比较和量化分析，同时对研究范围内的点进行了甄别，仅对考察点有吸引力的点(机会点)才参与运算，在本课题中，这个模型的利用场景更大，本课题的全部分析都以半小时为时间约束，意味着我们可以直接计算在半小时时间成本下从某街区所能到达的其它街区数量，数量越多，意味着这个街区的可达性越强。

但模型对于不同点的吸引力还是未加考察，此外约束范围的给定具有较大的主观性，不同范围下的可达性差异较大。如 10 分钟、30 分钟、1 小时、2 小时所能画出来的 OD 模型肯定是差别很大的，不同时间间隔下，所能到达的距离本就不同，因此，取值的合理与否是这个模型的关键所在，约束条件的合理控制能够保证计算的准确性。

#### 4.7.3 潜能模型

潜能模型提出可达性概念是指机会相互作用的潜力，而非相互作用的难易程度。

这种方法认为可达性作为衡量到达特定活动目的地的难易程度的指标，不仅受到两点之间空间阻隔的影响，还受到该点对活动吸引力大小的影响，前者的影响是负向的，后者的影响是正向的。这种方法最大的优点在于将土地使用(就业机会或公共服务类型及其规模)作为影响可达性的重要评价要素纳入了考虑，并且考虑了距离衰减的影响，而其不足之处在于没有考虑服务供给方的容量以及需求方之间的竞争，因此更多适用于简化竞争关系的宏观层面研究。这需要与街区的信息相关联。

在本课题中，街区的信息全部来自原始数据，等于要人为维护一份街区权重表，不同类型的街区所占的权重不同，在计算可达性的同时也需要考虑街区本身类型即权重值的大小。

#### 4.7.4 时空约束模型

时空约束模型是在时空地理学的基础上提出的，其所建立的可达性研究框架的核心是时空约束和时空棱柱，时空约束是个人活动的时空特性所导致的活动选择约束，而时空棱柱则是个体在特定时空约束下可能的活动空间。

基于时空棱柱的可达性计算主要包括两个方面：一是时空棱柱的体积，时空体积越大说明个体参与活动的空间越大，可达性水平越高；二是考察时空棱柱范围内的活动目的地数量(就业岗位、公共服务设施、商业网点等)，数量越多，可达性越高。该方法最大的特点在于将个体因素纳入了研究，使模型的精度大大增加，适用范围也实现了从宏观向微观的发展。但是这与本课题的研究内容不合，我们所计算与分析的是街区之间的活动，研究单位为街区，如果要以个体为研究单位，意味着要把原始数据全部计算出来，然后根据街区来划分结果，这样得到的数据必然精确于街区OD计算的模型，但是对性能和时间也有了更多的要求。

而其局限在于当运用于宏观研究时需要大量的数据加以支撑，也就是刚刚提到的要求大量的精确数据且有足够的运算力。

#### 4.7.5 效用模型

效用模型的理论基础是随机效用理论，模型假设出行目的地对出行人具有一定的效用，并且人们在出行选择时优先选择效用最大的目的地。在这一前提下，可以认为可达性就是出行选择的期望效用，期望效用越大的个体其可达性就越大。

这一模型意味着对本课题的意义在于，用已有的数据进行逆推，现在的数据可以看到一个街区到不同的街区的真实数据，而用这个数据的百分比即可推出个体从该街区出发时，对剩余每个街区的期望权重。在本课题研究中，与机会模型有相似之处。

### 4.7.6 比较

上述几种模型是较为常见的可达性量化方法，这些模型在对距离影响衰减、土地使用类别、个体选择意愿和设施竞争性等方面各有特点，空间阻隔模型直观易懂，却过于简化，精确度有所不足；机会模型考虑了设施类型且量化明确，但忽略了约束范围内不同距离对设施吸引力的影响；潜能模型体现距离衰减。

### 4.8 本章小结

本章主要数据分析的主要工作，主要介绍了如何基于现有的数据进行数据可视化并给出了解决方案，包括配色、色阶以及参照依据，如何基于已有的理论完成新型的数据可视化，这也是本课题的重点，下一章将介绍具体的开发流程与如何实现上文所提到的效果。

装

订

线

## 5 应用开发部分

在本课题中，应用实现主要以 web 形式存在，包括前端的实现，数据原本存在 Hadoop 中，由 Hive 批量计算后将数据暴露到 WebHDFS 上，这样可以通过 http 方式直接获取到所需的数据。

在之前的计算步骤中，属于本课题前半部分开发过程，由于只有原始数据，因此只能用 Node.js 原生读写文件流来处理数据并将新数据写到本地文件系统，然后用于开发和调试，这部分工作只选取了 30 分钟内的数据，在上线的过程中，为了提高性能，并且将数据做成自动化处理的流式读入读出作准备，将数据处理工作交给了 Hive 来完成，利用 Hive 的 sql 操作完成数据的处理并写入 hdfs，以 WebHDFS 的形式暴露为 API，可以直接用 http 的方式获取。

目前的成果集成在所属项目的智慧城市决策大数据支撑平台项目中，以 web 的形式访问，所有数据均放在服务器上，原本设计为，数据流入数据库，自动触发流式数据处理，处理后的结果存在 webhdfs 的路径上，而 web 平台的 App 在请求数据时直接访问 hdfs 的数据，但在实现过程中，遇到了浏览器同源跨域访问限制的问题，出于安全考虑，将所有数据下载到本地后，放到了阿里云的服务器上，这样所有数据都可以由 web 应用安全访问。

### 5.1 后端开发

后台开发主要内容为构建基于 Express 的 web 应用，将前端页面与数据文件映射到外网的路由并做 http 的缓存优化处理，下面将介绍具体部分。

#### 5.1.1 服务器静态资源映射

静态内容是指应用程序不会基于每个请求而去改变的资源，包括多媒体：图片、视频和音频文件、CSS、JavaScript、二进制下载文件：（这包含所有种类：PDF、压缩文件、安装文件等类似的东西）、以及一些 JavaScript（浏览器可以使用未经编译的 LESS。这种方式会影响性能，所以不推荐使用），将静态资源映射到服务器路径上主要出于性能考虑，这样可以减少请求次数，合理的合并资源，并且尽可能多地将小图片合并到一个子画面中。然后用 CSS 设定偏移量和尺寸只显示图片中需要展示的部分，并能够利用浏览器缓存，加快再次打开页面的时间，并缩减内容的大小等。

具体映射代码如图 5.3：

```
app.use('/data', data);
app.use('/static', express.static('public'));
```

图 5.3 映射代码

### 5.1.2 跨域和同源策略

同源策略是浏览器上为安全性考虑实施的非常重要的安全策略。而所谓同源，意思是：URL 由协议、域名、端口和路径组成，如果两个 URL 的协议、域名和端口相同，则表示他们同源。浏览器的同源策略，主要用来限制来自不同源的 "document" 或脚本，对当前 "document" 读取或设置某些属性。即从一个域上加载的脚本不允许访问另外一个域的文档属性。

而之前所说的跨域问题便由此而来，web 应用主要通过 Ajax 获取数据，而 Ajax 通过 XMLHttpRequest 能够与远程的服务器进行信息交互，由于 XMLHttpRequest 是一个纯粹的 Javascript 对象，这样的交互过程，是在后台进行的，用户不易察觉。

因此，XMLHTTP 实际上已经突破了原有的 Javascript 的安全限制。但是当我们需要 XMLHTTP 的无刷新异步交互能力，又不愿意公然突破 Javascript 的安全策略，可以选择的方案就是给 XMLHTTP 加上严格的同源限制，也就是说从根本上限制了跨域请求的提交。

因此，如上所述，我们无法在 web 应用中去请求存储在 webhdfs 上的数据，只能将数据上传到服务器并作为静态资源映射到路径上，这样就可以通过文件路径直接请求访问并使用。

服务器开启 CORS 允许代码如图 5.4：

```
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "X-Requested-With");
  next();
});
```

图 5.4 开启 CORS 代码

## 5.2 前端开发

### 5.2.1 Leaflet 开发

正如前文所提到，本课题的数据可视化部分底图全部由 Leaflet 渲染完成，不得不提 Leaflet 是一个非常成熟好用的地图库，整个配置过程非常简单、快速。

如图 5.5，即可建立一个完整的地图，并且切换为我们所需的主题：

```
// init map object
var majorMap = L
    .map("majorMap")
    .setView([
        block.config.map.initLat,
        block.config.map.initLng
    ],block.config.map.initiRoom);

// map theme
L.tileLayer.chinaProvider('Geoq.Normal.PurplishBlue',{
    maxZoom:block.config.map.maxZoom,
    minZoom:block.config.map.minZoom})
    .addTo(majorMap);
```

图 5.5 初始函数

上文所提到的热力图源自一款 Leaflet 的插件：leaflet—heatmap.js。使用比较方便。在用它来画热力图的时候，需要指定的参数如表 5.2。

装  
订  
线

表 5.2 热力图配置参数

参数名	取值	意义
radius	0.02	热力点的半径
maxOpacity	0.8	热力图的透明度
scaleRadius	true	是否平滑过渡
latField	lat	纬度的 key 值
lngField	lng	经度的 key 值
valueField	count	热力值的 key 值

接下来，需要我们处理好的参数，并符合以下格式。

```
{"lat":30.7000000089407,"lng":120.8399999821185,"count":1},
 {"lat":30.680000001341106,"lng":120.7599999999999,"count":33},
 {"lat":30.680000001341106,"lng":120.7799999955296,"count":7},
```

....

这样就可以直接导入热力图中。

函数如图 5.6：

```
osmb.setHeatMap = function(heatmapData){
    if(osmb.heatmapLayer != null){
        osmb.map.removeLayer(osmb.heatmapLayer);
    }
    var config = { //热力图的配置项
        radius : osmb.radius, // 设置半径大小
        maxOpacity: .8, // 设置最大的不透明度
        scaleRadius: true, // 设置热力点是否平滑过渡
        useLocalExtrema: true, // 使用局部极值
        latField: 'lat', // 维度
        lngField: 'lng', // 经度
        valueField: 'count', // 热力点的值
        gradient: { "0.99": "rgba(255,0,0,1)", // 渐变色
                    "0.8": "rgba(255,255,0,1)",
                    "0.7": "rgba(0,255,0,1)",
                    "0.4": "rgba(0,255,255,1)",
                    "0": "rgba(0,0,255,1)" },
    };
    osmb.heatmapLayer = new HeatmapOverlay(config); // 重新创建热力图对象
    osmb.map.addLayer(osmb.heatmapLayer); // 将热力图图层添加在地图map对象上
    osmb.heatmapLayer.setData(heatmapData);
}
```

图 5.6 热力图函数

而在画热力立方的时候，使用的是一个用于绘制三维模型的框架

OSMBuilding.js，使用他的时候，需要将要画的三维模型的长宽高都指定，并与原地图的坐标系相对应。在使用此框架的时候遇到了一个有趣的问题，框架默认在比较大的缩放等级下不显示所有的三维模型，这与我们的需求刚好相背，于是，我尝试与开发人员联系解决这个问题，但是他们表示无能为力，Github 上的 [issue 地址](#)。

问了 2 个问题，但是都没有正面回应，第一个的解决方案就是放弃了这种方式，而第二个问题必须解决，于是被迫去看了源码，然后找到了那个配置参数并改正，困难在于源码是混淆过的，所以比较困难去找变量，还好最后顺利解决。

Issue 内容截图如图 5.7：

装  
订  
线

Tokkiu commented on Apr 8

hi~  
I'm using the OSMBuildings for my school work about map&Visualization. :-P  
Your project is so awesome!  
But i faced a problem now.  
I found when the map's property `zoom >= 16`,the sky layer will show up automatically.  
I want to remove the sky layer so that the map left only.  
I search and read all the docs onsite but found nothing.  
So can u help me to solve this ? :-)

kekscom commented on Apr 9

Hi,  
I'm very glad you like OSM Buildings!  
I can just assume you've meant `zoom <= 16`. Then we show a sky in order not to load an infinite amount of buildings.  
This can't be turned off. You might limit zoom and tilt though.  
Regards,  
Jan  
...

Tokkiu commented on Apr 9

ok, I get what u mean.  
So i can't remove the sky.  
Then i want to try the OSMBuildings 2.5 with leaflet.  
Because it has no sky on my layer.  
But a new problem came:  
if the `zoom <= 14`,my buildings will be disappear!  
Then i read the source code and found:  
there is a constant variable named G whose value is 15 always!  
I guess u define it so that when the zoom smaller than 15,buildings won't show up.  
Can u tell me how to get my buildings show up when my zoom not so big?

kekscom commented on Apr 11

Hi,  
For OSM Buildings 2.5D it's the same reason. If you zoom out too far, your loading a massive amount of tiles.  
And still objects wouldn't look that good anymore. You may change the limit in source code and try.  
Regards,  
Jan

图 5.7 issue 详情

在渲染热力立方的时候有一些参数需要指定，然后要按照设定好的数据格式写入数据即可。格式如下

```
{"lat":30.70000000089407,"lng":120.83999999821185,"count":1},
{"lat":30.680000001341106,"lng":120.75999999999999,"count":33},
 {"lat":30.680000001341106,"lng":120.7799999955296,"count":7},
....
```

Count 值即为热力立方的高度，而长宽由事先给好。函数如图 5.8

```
var colorIndex = Math.round((config.colors.length - 1) * precent);
var Building = {
  "type":"Feature",
  "properties":{
    "color": osmb.getColor(tmp.count),
    "height": height ,
    "roofColor" : config.colors[
      osmb.connect(colorIndex, config.colors.length)],
    "wallColor" : "#696969"
  },
  "geometry":{
    "type":"Polygon",
    "coordinates": [
      [tmp.lng, tmp.lat],
      [tmp.lng - radius, tmp.lat],
      [tmp.lng - radius, tmp.lat - radius],
      [tmp.lng, tmp.lat - radius],
      [tmp.lng, tmp.lat]
    ]
  }
};
dataArr.features.push(Building);
```

图 5.8 热力立方函数

### 5.2.2 街区数据渲染

街区的渲染主要使用 Leaflet 的 API，每个街区为一个多边形，并由颜色进行填充，包括一系列配置参数见表 5.3。

表 5.3 多边形参数

参数	取值	意义
Color	Grey	多边形每条边的颜色
fillOpacity	0.3	多边形的透明度
Weight	0.1	多边形的边的宽度
fillColor	取决于使用场景	多边形内部的颜色

在不同的使用场景下，多边形内部的颜色填充有不同的意义，甚至多边形的边也有不同的表示方式，在前文中，街区分别有 3 种着色方式，街区区块的吞吐量的蓝色系、红色系以及街区的热力显示，不同的着色方式意味着不同的颜色取值，而所调用的函数都是一样的，只需要改变参数的取值即可。

由于街区的多边形包含很多点，如果每次都去向服务器请求数据，并且数据里包含了每个街区的点的信息的话，这样会导致每次的数据都会非常巨大，测试发现包含街区的数据平均每个 JSON 文件有 17MB，这在用户体验上非常不友好，导致

如果网速不够快的话，往往一个图就要等 10s 以上的时间，而在使用场景中，每个 30 分钟就会有 2 张关于街区的图，根本无法做到流畅的浏览数据。

因此，性能的优化尤为重要，首先，由于街区的数据本身是不会变的，因此，每次的数据只需要包含街区 id 即可，浏览器可以在一开始就将街区的边界信息缓存下来，这样就可以很快的完成渲染，但是这样还有一个问题，在第一次拿到街区数据的时候还是很慢，观察数据发现，每个点都精确到了小数点后很多位，然而事实上，只需要精确到第 5 位就足够满足本课题的需要，因此，这一步处理是将所有坐标点都去小数点后 6 位以后的数值，这样整个数据体积减小了一半，时间也缩短至原来的一半。

函数主体如图 5.9：

```
var config = block.config;
var color = count > 0 ? block.getRed(count) : block.getBlue(count);
for (var j = 0; j < rings.length; j++) {
  var tmp = block.transform(rings[j]);
  var polygon = L.polygon(tmp, {
    color: config.od_block.strokeColor,
    fillOpacity: config.od_block.fillOpacity,
    weight: config.od_block.weight,
    fill: config.od_block.fill,
    fillColor: color
  }).addTo(block.map);
}
polygon.bindPopup("color:"+color+", count:"+count);
```

图 5.9 街区多边形函数

在画出多边形后，便可以将其作为底图，来画街区之间的 OD，街区的 OD 即为线段，并传入参数来修改线段的样式，参数列表见表 5.4。

表 5.4 OD 线参数表

参数	取值	意义
fillOpacity	0.8	线的透明度
Weight	0.1	线的宽度
color	Grey	线的颜色

函数主体如图 5.10:

```
// in O-D graph, show o-d lines
block.showOdLines = function(limit){
  var config = block.config;
  var filterNum = limit;
  var arr = config.od_line.line_color_arr;
  var arrows = [];
  var odData = block.odData;
  for (var i = 0; i < odData.length; i++) {
    var width = Math.abs(odData[i].c);
    var latlgs;
    var color = block.getLineColor(width, arr);
    if (width >= filterNum) {
      latlgs = [[odData[i].y, odData[i].x], [odData[i].j, odData[i].i]];
      var polyline = L.polyline(latlgs, {
        weight: width * config.od_line.line_weight_zoom,
        fillOpacity: config.od_line.line_fillOpacity,
        color: color
      }).addTo(block.map);
    }
  }
  // showOdArrows(arrows);
}
```

图 5.10 画 OD 线函数

最后将代码封装为 2 个依赖文件，只要引入该依赖，传入参数，便可新建一个对象，并使用全部方法，使用方法如下

文件一：osmb.min.js

所需要的依赖如图 5.11:

```
<link rel="stylesheet" href="css/OSMBuildings.css">

<script src="js/leaflet.js" charset="utf-8"></script>
<script src="js/china.js" charset="utf-8"></script>
<script src="js/heatmap.js" charset="utf-8"></script>
<script src="js/leaflet-heatmap.js" charset="utf-8"></script>
<script src="js/jquery.js" charset="utf-8"></script>
<script src="js/OSMBuildings-Leaflet.js" charset="utf-8"></script>
<script src="js/osmb.min.js" charset="utf-8"></script>
```

图 5.11 osmb 所需依赖

README 如图 5.12:

```
// 新建osmb对象
// 参数 map:Object  (生成好的leaflet map对象)
// 参数 url:String  (数据所在路径)

var osmb = new Osmbuilding(map, "./data/osmb/");

// 画热力
// 参数 time:String  (时间,示例：20:00 - 200、20:30 - 205)

osmb.heatmap("200");

// 画立方体
// 参数 object:{
//   time : String,      (时间,示例：20:00 - 200、20:30 - 205)
//   min : int,          (低于该值的立方体不显示, 不填默认为100)
//   distance : double  (立方体间距离, 不填默认为0.01)
// }

osmb.building({
  time : "185",
  min : 100,
  distance : 0.01
});
```

图 5.12 画 OD 线函数

新建 osmb 对象所需参数见表 5.5

表 5.5 osmb 对象所需参数表

参数	类型	是否必须
map	Leaflet map	是
url	String (数据地址)	是

使用画热力函数所需参数见表 5.6

表 5.6 热力函数所需参数表

参数	类型	是否必须
Time	String (时间间隔)	是

使用画热力立方所需参数见表 5.7。

表 5.7 热力立方函数所需参数

参数	类型	是否必须
Time	String (时间)	是
Min	int (最小立方值)	否
Distance	Double (立方体间距)	否

文件二：block.min.js

所需要的依赖如图 5.13:

```
<script src="javascritps/leaflet.js" charset="utf-8"></script>
<script src="javascritps/china.js" charset="utf-8"></script>
<script src="javascritps/jquery.js" charset="utf-8"></script>
<script src="javascritps/block.min.js" charset="utf-8"></script>
```

图 5.13 block 所需依赖

使用说明如图 5.14:

```
// 新建block对象
// 参数 map:object (生成好的leaflet map对象),
// 参数 url:String (数据所在路径)

var block = new Block(map, "./data/block/");

// 画街区热力
// 参数 time:String (时间, 示例 : 20:00 - 200、20:30 - 205)

block.heatmap("235");

// 画街区od与吞吐
// 参数 time:String (时间, 示例 : 20:00 - 200、20:30 - 205)
// 参数 min:int (低于该值的od线不显示)

block.odMap("235",7);

// 画吞吐量
// 参数 time:String (时间, 示例 : 20:00 - 200、20:30 - 205)

block.showOD2Block("235");
```

图 5.14 block 使用说明

新建 block 对象所需参数见表 5.8。

表 5.8 block 所需参数

参数	类型	是否必须
map	Leaflet map	是
url	String (数据地址)	是

使用画街区热力的函数所需参数见表 5.9。

表 5.9 街区热力所需参数

参数	类型	是否必须
Time	String (时间间隔)	是

使用画街区 OD 的函数所需参数见表 5.10。

表 5.10 街区 OD 线所需参数

参数	类型	是否必须
Time	String (时间)	是
Min	int (最小立方值)	否

使用画街区吞吐量的函数所需参数见表 5.11

图 5.11 街区吞吐量所需参数

参数	类型	是否必须
Time	String (时间间隔)	是

### 5.3 数据计算

数据计算流程已在第三章中详细说明，因此此处只介绍代码实现。由于数据从文件中读入并直接写入文件，因此实现过程中包含大量的文件读写操作。

#### 5.3.1 热力数据

热力数据主要重点在于优化数据的体积，以尽量小的参数值来传递，因为热力的数据必然会有许多点，如果能让不必要的数据清除掉，只解析真正重要的数据，对性能优化有很大帮助。优化包括减少小数点位数吧，去掉不必要信息等。具体步骤如图 5.15：

```
var fs = require("fs");

// 获得原数据
fs.readFile('../public/tmp/block/enCounty.json',(err, data) => {
  if(err) throw err;
  var inputs = JSON.parse(data);
  for (var i = 0; i < inputs.features.length; i++) {
    var attributes = inputs.features[i].attributes;
    attributes.cy = new Number(attributes.cy.toFixed(5));
    attributes.cx = new Number(attributes.cx.toFixed(5));
    inputs.features[i].attributes = attributes;

    var rings = inputs.features[i].geometry.rings;
    for (var j = 0; j < rings.length; j++) {
      for (var k = 0; k < rings[j].length; k++) {
        var tt = rings[j][k][0];
        rings[j][k][0] = new Number(rings[j][k][1].toFixed(5));
        rings[j][k][1] = new Number(tt.toFixed(5));
      }
    }
    inputs.features[i].geometry.rings = rings;
  }

  fs.writeFile('../public/tmp/block/newEnCounty.json', JSON.stringify(inputs), (err) => {
    if (err) throw err;
    console.log("saved");
  })
});
```

图 5.15 热力数据计算

装  
订  
线

### 5.3.2 街区数据

这里街区数据分别包括街区热力、街区吞吐量、街区 OD，共三种数据。街区的信息来自 ArcGis 的服务器，包括街区 id、街区信息、街区多边形数据等，下面将一一介绍每种数据的计算代码。

街区热力数据，主要包括街区的 id 以及街区的热力值，计算过程如图 5.16：

装  
订  
线

```
var fs = require("fs");

fs.readFile('../public/tmp/block/od/carsOD.json',(err, data) => {
    if(err) throw err;
    var inputs = JSON.parse(data);
    var len = inputs.length;
    var arr = [];
    // 遍历车辆od数据 根据相同的街区id 合并为街区的od信息
    for (var i = 0; i < len; i++) {
        if (inputs[i] != null) {
            searchArr(inputs[i][0], inputs[i][1], arr);
        }
    }
    fs.writeFile('../public/tmp/block/od/blockOD.json', JSON.stringify(arr), (err) => {
        if (err) throw err;
        console.log("saved");
    })
});

function searchArr(o, d, arr) {
    if (o == d) {
        return;
    }
    var f = true;
    var i = 0;
    var len = arr.length;
    console.log("arr.length:"+len);
    while ( f && i < len) {
        //匹配到起点街区
        if (arr[i][0] == o || arr[i][1] == o) {
            if (arr[i][0] == d || arr[i][1] == d) {
                arr[i][2] += o < d ? 1 : -1;
                f = false;
            }
        }
        i++;
    }
    //未匹配到
    if (f) {
        var n = o < d ? 1 : -1;
        arr.push([o, d, n]);
    }
}
```

图 5.16 街区热力计算

街区 OD 数据，计算过程中主要解决的问题为从 String 类型的数据中读取并比较时间的前后，对车辆的 id 以及街区的 id 建立索引的大小。计算过程如图 5.17：

装  
订  
线

```
fs.readFile('../public/tmp/block/od/carsOD.json',(err, data) => {
    if(err) throw err;
    var inputs = JSON.parse(data);
    var len = inputs.length;
    var arr = [];
    // 遍历车辆od数据 根据相同的街区id 合并为街区的od信息
    for (var i = 0; i < len; i++) {
        if (inputs[i] != null) {
            searchArr(inputs[i][0], inputs[i][1], arr);
        }
    }
    fs.writeFile('../public/tmp/block/od/blockODColor.json', JSON.stringify(arr), (err) => {
        if (err) throw err;
        console.log("saved");
    })
});

function searchArr(o, d, arr) {
    if (o == d) {
        return;
    }
    var of = true;
    var df = true;
    var i = 0;
    var len = arr.length;
    while ( (of || df) && i < len) {
        //匹配到起点街区
        if (arr[i][0] == o) {
            arr[i][1] -= 1;
            of = false;
        }
        if (arr[i][0] == d) {
            arr[i][1] += 1;
            df = false;
        }
        i++;
    }
    //未匹配到o
    if (of) {
        arr.push([o, -1]);
    }
    //未匹配到d
    if (df) {
        arr.push([d, 1]);
    }
}
```

图 5.17 街区 od 计算

街区吞吐量，基于街区 OD 的计算结果，主要参考值为街区 id，用 id 来计算每一个街区的净吞吐量，并将结果写入文件。计算过程如图 5.18：

```
var fs = require("fs");

// 获得街区id与中心经纬度map
fs.readFile('../public/tmp/block/od/name.json',(err, data) => {
    if(err) throw err;
    var inputs = JSON.parse(data);

    // 获得以街区id显示的od信息
    fs.readFile('../public/tmp/block/od/blockODColor.json',(err,odata)=>{
        var od = JSON.parse(odata);
        var len = od.length;
        // 将街区id替换为rings
        for (var i = 0; i < len; i++) {
            var tmp = od[i];
            var a = getRing(tmp[0], inputs);
            tmp = [tmp[1],a];
            od[i] = tmp;
        }

        fs.writeFile('../public/tmp/block/od/blockColorLat.json', JSON.stringify(od), (err) => {
            if (err) throw err;
            console.log("saved");
        })
    })
});

function getRing(d, arr){
    for (var i = 0; i < arr.length; i++) {
        if (arr[i].attributes.objectid == d) {
            return arr[i].geometry.rings;
        }
    }
}
```

图 5.18 街区吞吐量计算

### 5.3.3 聚类算法

本课题中使用的聚类算法为 K-Means 聚类，这里使用了开源框架“kmeans-js”的 API，聚类数据后将结果转换为定义好的数据解析格式并直接写入文件中，代码如图 5.19：

装

订

线

装  
订  
线

```

var kMeans = require('kmeans-js');
var fs = require("fs");

fs.readFile('../public/tmp/rect/km.json',(err, data) => {
  var input = JSON.parse(data);
  var km = new kMeans({
    K: 200
  });
  km.cluster(input);
  while (km.step()) {
    km.findClosestCentroids();
    km.moveCentroids();
    if(km.hasConverged()) break;
  }
}

var maxCount = 0;
fs.readFile('../public/tmp/rect/18520n.json',(err, data) => {
  if(err) throw err;

  var srcData = JSON.parse(data);
  for (var i = 0; i < km.centroids.length; i++) {
    var tmp = {
      "lat": km.centroids[i][0],
      "lng": km.centroids[i][1],
      "count": 0
    };
    var tmpCount = 0;
    for (var j = 0; j < km.clusters[i].length; j++) {
      var index = km.clusters[i][j];
      tmpCount += parseInt(srcData[index].count);
    }
    if (tmpCount > maxCount) {
      maxCount = tmpCount;
    }
    tmp.count = getCount(tmpCount);
    km.centroids[i] = tmp;
  }
  fs.writeFile('../public/tmp/rect/kmReslk_3.json', JSON.stringify(km.centroids), (err) => {
    if (err) throw err;
    console.log("saved");
    console.log(maxCount);
  });
}
});

```

图 5.19 聚类数据后计算

### 5.3.4 数据传输

数据由 Hive 计算后保存在 Hadoop 的 hdfs 文件系统中，可通过 WebHDFS 的方式访问并持久化到服务器上，代码如图 5.20

```

var fs = require("fs");
var request = require('request');

var url = "http://10.10.240.118:50070/webhdfs/v1/taxish20150401_";

var arr1 = ["sttp","stod","stagg","agg1","agg2"];
var arr2 = ["./block/square","./block/od","./block/heat","./osmb/heat","./osmb/build"];

for (var i = 0; i < arr1.length; i++) {
  var type = arr1[i];
  var dir = arr2[i];
  for (var j = 0; j < 24; j++) {
    if (j < 10) {
      j = "0" + j;
    }
    query(j + "0", type, dir);
    query(j + "5", type, dir);
  }
}

function query(time, type, dir){
  request({
    method: 'GET',
    uri: url + type + time + ".json?op=OPEN",
  },
  function (error, response, body) {
    if (error) {
      return console.error('upload failed:', error);
    }
    fs.writeFile(dir + time + ".json", body, (err) => {
      if (err) throw err;
      console.log(dir + time + "saved");
    })
  })
}

```

图 5.20 数据传输

## 5.4 本章小结

本章中，介绍了前后台开发的主要工作。本章开始简略介绍了 Hadoop 生态系统和 Hive 的使用；之后介绍了所用到的渲染框架的用法和具体实现，并介绍了本课题的最终代码成果与使用方法。

装  
订  
线

## 6 总结和展望

### 6.1 总结

随着技术的发展和进步，随着运算能力的不断提升，数据的增长呈指数级。在这样一个大数据的时代，任何数据都有它独特的价值，不断挖掘数据背后的秘密，挖掘数据下隐藏的信息，是未来商业的重中之重，合理的数据可视化，能让 BI 的效果事半功倍，而且随着数据的增加，量级的增大和纬度的增大，数据越来越复杂，越来越难以捉摸，面对错综复杂的数据，要学会应对不同类型的数据、不同类型的需求，在可视化的过程中，面对不同的数据，所要侧重的重点也不尽相同，要记住，技术永远是为业务服务的，要明白，怎样去可视化，怎样去渲染数据，才能让结果一目了然，让用户清晰的看到、观察到隐藏在数据下的秘密，这些都是要认真研究的重点。

用哪种类型的聚类算法，完全取决于业务需求适合哪种，用哪些颜色来渲染，用深色还是浅色，都没有固定的答案，一切为最终结果服务，只有适合这个案例的，才是最好的方案。

其实，无论是数据清洗、数据处理还是数据可视化，都是为了能让用户直观、便捷的体验数据的奥秘，可以让用户不用接触数据的细枝末节就可以对整个数据一探究竟，保证最终能够从海量的数据中获取我们觉得有价值的信息，让他人和社会受益，这也正是整个业界在一直努力的方向和美好愿景。

### 6.2 展望

(1) 本文中介绍了热力图的渲染与计算，但是在热力色阶的选择上一直在改进，综合了很多现有的案例，也查了一些资料，但这部分还可以进一步加强，正确的色阶选择对热力图意义重大。

(2) 在 OD 矩阵中，我们计算的是街区的 OD 值，以半小时为间隔，而接下来可以尝试以出租车为个体，来计算车的 OD，相比街区，计算车的 OD 矩阵可能更为复杂，计算量也较大，但是数据肯定更精确，效果也优于街区的 OD 矩阵图。

(3) 有了 OD 矩阵，计算可达性就有了部分数据依据，但如何确定模型，或者如何将几种模型取长补短，新建一种适合本课题的模型，还需要继续探索和研究。

由于掌握知识的有限性，可能还有更多的错误和功能等待我们去发现和实现。我会不断学习研究新的知识，增强自己的能力，完善现在已经完成的工作，解决可能随时出现的问题。

## 参考文献

- [1] D.L.Donoho, High Dimensional Data Analysis: The Curses and Blessings of Dimensionality[R], presented at American Mathematic Society Conference: Math Challenges of the 21st Century, Los Angeles, USA, 2000
- [2] Sunden E, Ropinski T. Efficient volume illumination with multiple light sources through selective light updates[C]// Visualization Symposium (PacificVis), 2015 IEEE Pacific. IEEE, 2015:231-238.[pdf]
- [3] Ropinski T, Doring C, Rezk-Salama C. Interactive Volumetric Lighting Simulating Scattering and Shadowing[C]// IEEE Pacific Visualization. 2010:169-176.[pdf]
- [4] Ropinski T, Doring C, Rezk-Salama C. Interactive Volumetric Lighting Simulating Scattering and Shadowing[C]// IEEE Pacific Visualization. 2010:169-176.[pdf]
- [5] Hermann M, Schunke A C, Schultz T, et al. Accurate Interactive Visualization of Large Deformations and Variability in Biomedical Image Ensembles[J]. Visualization & Computer Graphics IEEE Transactions on, 2016, 22(1):708-717. (SciVis 2015) [pdf]
- [6] Kurzhals K, Hlawatsch M, Heimerl F, et al. Gaze Stripes: Image-Based Visualization of Eye Tracking Data[J]. IEEE Transactions on Visualization & Computer Graphics, 2015, 22(1):1-1. (SciVis 2014) [pdf]
- [7] David G, Joseph S, Julien T. Mandatory Critical Points of 2D Uncertain Scalar Fields[C]// Computer Graphics Forum. 2014:31–40. (EuroVis 2014) [pdf]
- [8] Kwan-Liu Ma, In Situ Visualization at Extreme Scale:Challenges and Opportunities. IEEE Computer Graphicsand Applications, 29(6):14-19, 2009.
- [9] Caroline Ziemkiewicz, and Robert Kosara, The Shaping of Information by Visual Metaphors [J]. IEEE Transactions on Visualization and Computer Graphics,14(6):1269-1276, 2008
- [10] D. J. Lehmann , H. Theisel. Optimal Sets of Projections of High-Dimensional Data[J]. IEEE Transactions on Visualization and Computer Graphics , 2016, 22:609 - 618. (InfoVis 2015) [pdf]
- [11] A. Ottley, E. M. Peck al. Improving Bayesian Reasoning: The Effects of Phrasing, Visualization, and Spatial Ability[J]. IEEE Transactions on Visualization and Computer Graphics , 2016, 22:529 - 538. (InfoVis 2015) [pdf]
- [12] Netzel R, Burch M, Weiskopf D. Comparative Eye Tracking Study on Node-Link Visualizations of Trajectories[J]. Visualization & Computer Graphics IEEE Transactions on, 2014, 20(12):2221-2230. (InfoVis 2014) [pdf]
- [13] Yangxin Zhong, Shixia Liu al. Tracking Idea Flows between Social Groups. AAAI 2016. (accepted) [pdf]
- [14] Tom White. Hadoop 权威指南[M]. 北京: 清华大学出版社, 2014
- [15] Scholkopf B. Amola A. Muller K. Nonlinear component analysis as a kernel eigenvalue problem [J] Neural Computation, 1998, 10 (5), 1299–1319

- 
- [16] Roweis S T. Saul L K. Nonlinear dimensionality reduction by locally linear embedding [J]. *Science*. 2000, 290(5500):2322–2326
  - [17] Tenenbaum J B. Silva V. Ungar J C. A global geometric framework for nonlinear dimensionality reduction [J]. *Science*, 2000, 290(12), 2912–2923
  - [18] Belkin M. Problems of learning on manifolds[D]. Chicago: The University of Chicago, 2003
  - [19] Belkin M. Niyogi P. Laplacian eigenmaps for dimensionality reduction [J]. *Neural Computation*, 2003, 15(6):1373–1396
  - [20] Lafon S. Diffusion maps and geometric harmonics [D]. USA: Yale University, 2004

装

订

线

## 謝 辭

在开始这个课题之前，我对数据可视化的了解很有限，能够完成这个课题，不仅因为自己的学习研究，更离不开老师、学校和同学的帮助和支持。如果没有他们的帮助，我很难顺利如期完成这篇论文。

首先我要由衷地感谢我的导师曹布阳老师。曹老师在课题开始时就给我提供了很多参考文献，在课题进行的这段时间内，老师也会经常分享好的文章或书籍给我，是我在课题的研究中少走弯路。当课题遇到瓶颈时，曹老师给我提供了很好的解决思路。如果没有曹老师的悉心指导，我想我的论文不可能顺利地完成。

其次，还要感谢同组的饶依文同学给我提供了很多数据支持，并很有耐心地给我解决数据的问题，让我能够有完好的数据研究完成课题。

最后，还要感谢同项目组的学长、学姐们，他们给我提供了很多处理问题的思路，并且监督我能够按时或者提前完成需要完成的任务，不会因为懒惰而浪费时间。还要感谢我的队友们，我们一起讨论了很多知识，解决了很多问题，给予我很大的帮助。

裝  
訂  
線