

本科毕业设计论文

题目：交通大数据网站的设计与开发

|  |  |
| --- | --- |
| **作者姓名**  **指导教师**  **专业班级**  **学 院** | **朱鑫栋** |
| **李永强 教授** |
| **电气工程及其自动化1401** |
| **信息工程学院** |

**提交日期** 2018年6月05日

**Dissertation Submitted to Zhejiang University of Technology**

**for the Degree of Bachelor**

**Dissertation Title (Times New Roman, Bold, Centered, Multi-lines allowed)**

**Student: Zhu Xindong**

**Advisor: Li Yongqiang**

**College of Information Engineering**

**Zhejiang University of Technology**

**June 20xx**

浙江工业大学

本科生毕业设计(论文、创作)诚信承诺书

本人慎重承诺和声明：

1. 本人在毕业设计（论文、创作）撰写过程中，严格遵守学校有关规定，恪守学术规范，所呈交的毕业设计（论文、创作）是在指导教师指导下独立完成的；

2. 毕业设计（论文、创作）中无抄袭、剽窃或不正当引用他人学术观点、思想和学术成果，无虚构、篡改试验结果、统计资料、伪造数据和运算程序等情况；

3. 若有违反学术纪律的行为，本人愿意承担一切责任，并接受学校按有关规定给予的处理。

学生（签名）：

年 月 日

**本科毕业设计的任务书**

（用盖过红章的任务书替换）

交通大数据网站的设计与开发

摘 要

现代化的智能交通系统往往以大数据为基础，不断分析大量的用户、设备数据，从中找寻规律，提升系统服务的准确性和可靠性。本文主要研究了以大数据驱动的交通信号灯配时排查算法，并以该算法为基础，从工程上设计并实现了一个交通大数据平台。

在交通系统中，信号灯时长的计算极其复杂，同时又对道路拥堵情况具有非常重大的影响。因此，本文先根据单车道绿灯浪费时间估计算法，利用停车线过车数据点的密度来估算绿灯浪费时间。接着介绍了单车道绿灯不足时间估计算法。在此基础上，本文继续研究了信号灯配时方案相位设计排查算法，最终通过对各个路口的大量过车数据进行分析，指出问题，并且提供相应的优化建议。

最后，利用Python实现了上述算法，并且在Django框架的基础上，利用可视化技术架构出一个交通大数据分析网站，并且稳定运行于服务器上。

**关键词：**大数据，智慧交通，可视化，网站设计

ABSTRACT

Modern intelligent transportation systems are often based on big data, constantly analyzing a large number of users and equipment data, and looking for patterns to improve the accuracy and reliability of system services. In this paper, we mainly study the algorithm of large-data-driven traffic signal timing and use this algorithm as a basis to design and implement a traffic big data platform from engineering.

In the traffic system, the calculation of signal duration is extremely complicated, and at the same time it has a very significant impact on road congestion. Therefore, in this paper, according to the single-lane green light waste time estimation algorithm, the green light waste time is estimated by using the density of the parking line data points. Then introduced the single-lane green light shortage time estimation algorithm. On this basis, this paper continues to study the phase design and troubleshooting algorithm of the signal lamp timing plan. Finally, it analyzes a large number of passing data of each intersection, points out problems, and provides corresponding optimization suggestions.

Finally, the algorithm is realized by Python, and on the basis of the Django framework, a large traffic data analysis website is constructed by visualization technology, and it runs steadily on the server.

**Key Words**: Big data, intelligent transportation, visualization, website design

目录页预留

参 考 文 献-临时版本

1. J. Zhang et al., “Data-driven intelligent transportation systems: A survey,”IEEE Trans. Intell. Transp. Syst., vol. 12, no. 4, pp. 1624–1639,Dec. 2011.
2. 张昕,曾鹏,张瑞,张帆.交通大数据的特征及价值[J].软件导刊,2016,15(03):130-132.
3. 舒采焘，张孜 . 新型城市化背景下的先进交通信息服务体系构建 [J]. 交通科技与经济，2016，18（5）：21-25.
4. 关积珍 . 对北京奥运公众交通信息服务的探讨 [J]. 交通运输系统工程与信息，2008，8（6）：61-66.
5. 朱昊，王磊，张会娜 . 世博交通决策支持信息服务系统研究 [J]. 城市交通，2010，8（5）：84-88
6. 任广震,侯进,王献.MVC模式在B/S结构政务系统的应用研究[J].计算机应用与软件,2014,31(08):54-58+74.
7. 朱爱红，余冬梅，张聚礼． 基于 B/S 软件体系结构的研究［J］． 计算机工程与设计，2005，26(5):1164-1166．
8. 李德兵,尹战文,王洪明.Java EE基于Hibernate的ORM框架[J].电子技术,2010,37(02):7-8+3.
9. 姜立.动态网页前端开发技术以及优化策略研究[J].数字技术与应用,2018,36(03):188+190.
10. 王国庆.浅谈浏览器兼容统一性问题与解决方案[J].电脑知识与技术,2015,11(07):57-58+62.
11. 绪 论
    1. 课题研究目的和研究意义

现代大城市的交通拥堵是一个世界性难题，即使是在高度现代化的发达国家，城市交通的拥堵问题也没有得到根本的解决。交通问题是一个大城市经济发展的命脉，在现代都市，随着运输系统的快速发展，交通已经成为人类生活的重要组成部分并且对人们的生活质量产生了深远的影响，甚至成为了城市发展的瓶颈。调查显示，大约平均有40%的人每天至少需要花费1小时来通勤。为了改善交通状况，人们不断尝试各种方法，其中，基于大数据的智能交通系统极具前景。[1]

近年来，“大数据”一词被越来越多的提及，其中就包括交通大数据。大数据具有鲜明的“4V”（Volume、Variety、Value、Velocity）特点[2]。由于交通大数据来源、种类的多样性，使其特征更加鲜明且复杂，而且交通大数据需要面对大众，往往具有可视化的需求，一般来说，交通大数据有如下四个特点：

1) 数据量巨大，采集渠道广泛，且部分数据需要长期储存。 不同于其它系统，交通大数据系统的数据来源非常广泛，现代城市道路系统布满了各种传感器、监控探头，它们每天都会采集海量的数据，而且数据种类多样，且需要长期储存。

2) 对数据实时性和处理速度要求极高 交通大数据网站的作用之一就是实时分析城市道路的交通状况，并为用户提供出行建议和指导，因此需要快速处理数据并且实时响应。

3) 数据模态多样化 由于交通大数据来源广泛，因此其类型也极其多样化，这对数据分析系统的兼容性提出了很高的要求

4) 有可视化需求 不同于传统的报表形式，交通大数据的分析结果往往需要以可视化的形式展现给用户。

传统交通系统往往由于历史包袱的沉重，导致了其架构非常重量级，可维护性差，界面难用，交互体验差，而且由于技术的局限性，不能准确有效的分析城市道路交通系统。但如果由传统交通系统辅以大数据技术，就能够推进交通数据采集、分析、管理利用等方面的巨大进步。现代化的智慧交通系统大多采用数据可视化技术，用户可以在界面上清晰的看到实时的系统分析结果，并提供相应的建议，大幅提升了交互体验。

* 1. 课题研究现状
     1. 国外研究现状

日本从 1990 年开始交通信息服务系统（VICS，Vehicle Information and Communication System） 的研究和建设，已覆盖东京等大城市及主要高速公路。自 2002 年 VICS 中心可通过手机终端、掌上电脑、个人电脑、车载终端和电视接收器等多种途径提供交通拥堵、交通事故、道路施工、广域最优路径建议、天气状况及停车场信息等多样化的信息服务。[3]新加坡则拥有出行者信息服务系统，能够为用户提供准确实时的地铁、公交等服务信息。除了车辆的发车时间和预计通勤时间，用户还可查询到任意两地间的最少周转、最低票价或最快抵达的推荐交通路线和相应票价信息。

* + 1. 国内研究现状

我国的交通信息服务系统建设以北京、上海为典型代表。

2006 年，北京市交通委组织实施交通部公众出行信息服务系统示范工程建设，开通北京公众出行网（www.bjjt.cn），在整合交通行业信息资源的基础上，为公交乘客和自驾车出行者提供实时、动态和综合性的交通信息服务。2007 年，在示范工程基础上开展北京市公众交通信息服务系统一期工程建设，为 2008 年奥运会提供公众交通信息服务奠定基础。[4]

2006 年，上海率先建设市级交通综合信息平台，全面、实时整合和处理全市道路交通、公共交通、对外交通领域车流、客流、交通设施等基础信息资源。在此基础上，为保障 2010 年世博交通高效有序运行，建设和完善世博交通决策支持信息服务系统、世博交通网、世博交通指南、电台电视台、世博交通信息咨询服务热线、路侧可变信息标志、手机与车载导航、自助查询触摸屏等多种信息服务方式。为世博交通管理者、决策者及广大公众提供全面、实时、准确的世博交通信息服务，对世博交通安全保障起到关键性作用。[5]

上述都是国内外对智慧交通系统的早期探索。而近年来，数据可视化技术不断发展，也为交通数据分析结果提供了越来越多的展现方式。例如，基于MIT协议开源的图形库chart.js，它使用HTML5 Canvas元素构建图表，性能优越，上手简单，轻量级，无任何第三方依赖。国内厂商近年来也推出了一些可视化产品。例如，百度基于Apache2.0协议开源的图表库echarts.js，该图表库底层基于ZRender（一个轻量级的Canvas类库），提供直观，生动，可交互，可个性化定制的数据可视化图表。这些可视化技术都能方便的构建用户友好型的界面来展现交通大数据的分析结果，大大提升用户的交互体验。

* 1. 本文主要章节概述

章节概述预

第2章 理论基础

2.1 单车道绿灯浪费时间估计算法

单车道绿灯浪费时间估计算法是数据驱动信号配时排查算法的基础。其基本思想是：利用停车线过车数据点的密度来估计绿灯浪费时间。利用该算法，我们可以有效的估算单车道的绿灯浪费时间，基本算法如下：

***输入：***感兴趣时间段内，车道停车线处的过车数据；

***输出：***车道的绿灯浪费时间；

***可调参数***：绿灯时间网格化的网格长度，数据点密度阈值。

***具体步骤：***

1. 将所属相位的绿灯时间以固定长度等分，得到如图2-1所示的红色矩形（高为，宽为感兴趣的时段长度）
2. 计算每个红色矩形中的数据点个数，其中对应绿灯结束时的第一个矩形，以此类推；
3. 若，则绿灯浪费时间为

图2-1 单车道绿灯浪费时间估计算法原理

注1：算法中步骤c可以理解为：从绿灯结束时对应的矩形开始查找，如果连续i个矩形中的数据点个数小于阈值，则说明这些时间段内过车数量明显不足，所以不妨认为这i个秒的绿灯时间都是浪费的。

注2：算法中的可调参数网格长度建议选取为3秒。该经验值通过一个交叉口的小样本数据分析得到，还需要更多数据的验证。

注3：算法中的可调参数数据点密度阈值的选取需要非常仔细慎重。图2-2显示了一个交叉口所有车道的网格化绿灯时间通过的车辆数(选取为3秒)。该交叉口为传统四相位配时。由图2-2可以看到如下结论：

1. 不同车道的通行能力不同(东西向直行车道的车辆数明显多于其他车道)，所以需要对不同车道选取不同的，而且的值应当由过车数据来决定。由于是一个经验值，因此需要对大量过车数据进行分析来得到。经过大量实验模拟，得出一种可行的选取为：



即选取绿灯开始时矩形中的车辆数乘以0.4作为阈值。系数0.4是为了使得绿灯浪费时间的判断更为严格。图2-2中，所有车道的为横坐标为2的点。

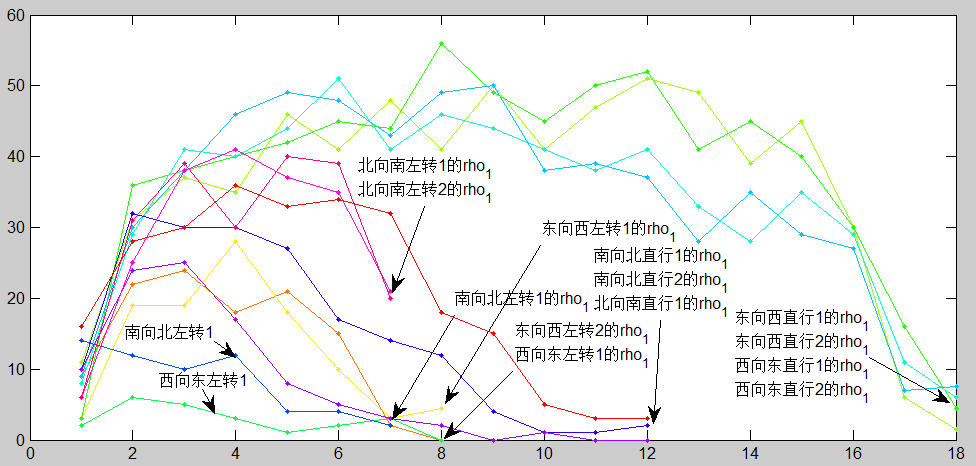
1. 对于过车数量非常少的车道，即平均每个周期通过的车辆数少于3辆(如南向北左转1和西向东左转1)，上述算法并不适用，因为该种情况下的完全不能体现排队消散时的流率。此时，可以直接利用绿灯时间减去最小绿灯时间作为绿灯浪费时间。

图2-2 某交叉口所有车道的

得到车道的绿灯浪费时间后，就可以利用其判断该车道的绿灯时间是否合理，判断依据如下：

* 若绿灯浪费时间等于，则合理；
* 若绿灯浪费时间等于0，则不合理，建议增加绿灯时间；
* 若绿灯浪费时间大于，则不合理，建议减少绿灯时间；

2.2 单车道绿灯不足时间估计算法

理想状况下过车/旅行时间预测（绿灯时间增加）

***假设 1：*** 理想情况下车辆的驾驶行为满足如下假设：所有车辆行驶速度相同。因此，所有车辆在不遇到红灯时，旅行时间都为，称为*标称旅行时间*。

图2-3显示了理想情况下绿灯时间增加引起的车辆行驶过程变化。图中，红色线段表示原来的红灯时间，桔色线段表示绿灯增加后的红灯时间，14条黑色实线轨迹表示在原来配时下14辆车辆的时空轨迹，14条蓝色虚线轨迹表示在绿灯增加后14辆车辆的时空轨迹。图底部的标明了车辆时空轨迹的起点；图顶部的数字标明了车辆时空轨迹的终点，括号外的数字标明了绿灯增加后车辆时空轨迹的终点，括号中的数据标明了原来配时下车辆时空轨迹的终点。

图2-3 理想情况下绿灯时间增加引起的车辆行驶过程变化

通过对图2-3的分析可知，当绿灯时间增加时，新的过车时间和新的旅行时间的变化可以分为如下三类：

1. 若前面有n辆车提前通过，则第辆车的新过车时间和新旅行时间可以用下面的方法计算：

 （1）

其中和为第辆车在原配时下的过车时间和旅行时间。例如：

图2-3中的v1：；车辆v2：。

1. 若由（1）式得到的新旅行时间小于标称旅行时间，则说明提前通过的车辆较多，故更新提前通过的车辆数，再由（1）式重新计算和。
2. 若由（1）式得到的新时空轨迹可以在增加的绿灯时间通过，则说明提前通过的车辆数过少，故更新提前通过的车辆数，再由（1）式重新计算和。

结合以上结论，在理想情况下（假设1满足的情况下），若绿灯时间增加，则过车时间/旅行时间的预测算法如下：

|  |  |
| --- | --- |
| function [,] = estPtTt() | |
| 输入: | 某车道的过车时间数据, 为数据点个数  某车道的旅行时间数据, 为数据点个数  获得数据时的周期  获得数据时的绿灯时间, 为绿灯开始时刻，为绿灯结束时刻  增加的绿灯时间  标称旅行时间 |
| 输出: | 新配时方案下，过车时间的预测值  新配时方案下，旅行时间的预测值 |
| % 过车时间提前的车辆数  % 上一辆车的轨迹是否是新产生的  for  % 更新过车时间  % 更新旅行时间  while  and  % 旅行时间必须大于等于标称旅行时间    % 更新过车时间  % 更新旅行时间  end    % 判断更新后的过车时间能否在增加的绿灯时间内通过  if      else      end    if  and  and          else    end  end | |

表2-1 过车时间/旅行时间预测算法(绿灯时间增加，周期不变)

2.3 配时方案相位设计排查

第3章 网站基础架构

3.1 传统C/S体系架构

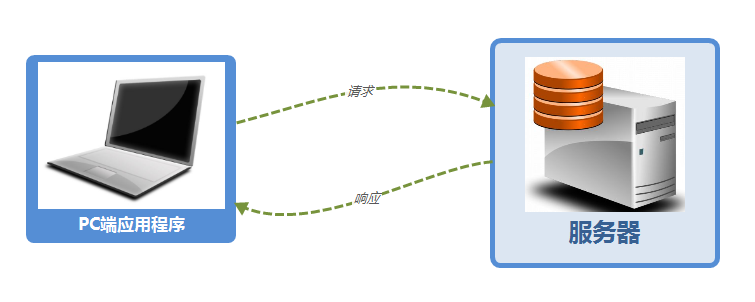
传统的C/S体系架构即客户端/服务端架构，如图3-1所示。这种体系架构

图3-1 传统C/S架构体系

在20世纪80年代由美国的Borland公司设计开发[6]，结构清晰简单。其中，PC端应用程序负责展示用户界面，完成交互逻辑，并向服务器端发送相应的请求。服务器端则负责业务逻辑的处理与数据的持久化，并向客户端返回对应的结果。

这种架构的好处和坏处都显而易见，好处有：应用程序直接访问服务器，可以自定义网络协议，运行速度快，服务器端负载小。不足之处在于：这种架构跨平台能力差，需要为不同的终端专门编写不同的客户端程序；扩展性低，维护困难。此时，所有应用程序，数据库，文件等资源全部在同一台服务器上，系统耦合较为严重。随着现代互联网技术的不断发展，这种架构逐渐无法满足用户的需求，于是，出现了B/S架构。

3.2 B/S三层架构

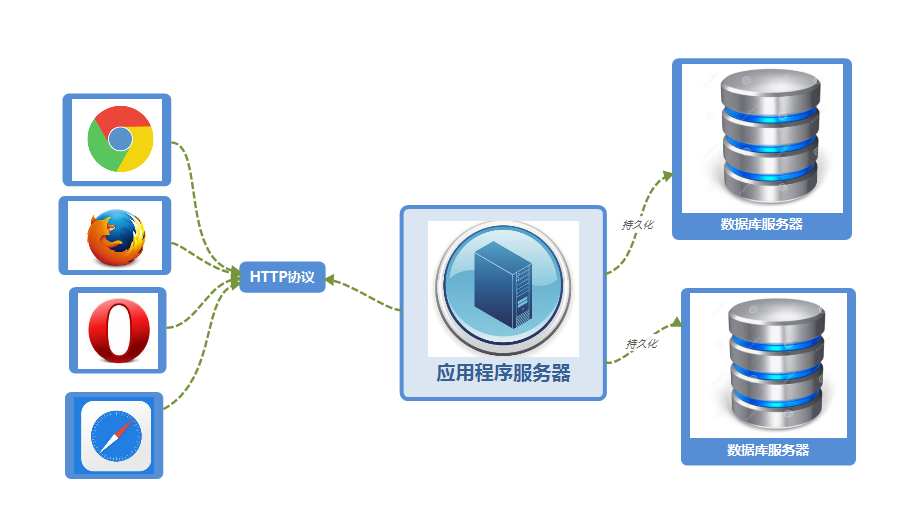
B/S架构体系[6]即浏览器/服务器体系结构，最早由美国微软公司(Microsoft)设计研发。不同于传统的C/S架构，该架构不依赖于专门编写的客户端应用程序或者APP，而是将大部分终端设备都有的网页浏览器作为用户界面的展示窗口。这是一种具有划时代意义的系统架构，B/S架构的出现，大大减少了客户端的维护与升级成本，并且在大局意义上统一了客户端的表现结构（浏览器），为后来的互联网时代奠定了基础。B/S架构的基本原理如图3-2所示。

图3-2 B/S架构基本原理图

图3-2中，第一层是用户浏览器层，主要负责展示界面和交互逻辑。中间层为应用程序服务器，专门负责处理业务逻辑，浏览器通过HTTP协议与浏览器进行通信。以Java为例，常见的应用程序服务器有Tomcat、Jboss等。需要注意的是，应用程服务器专门用于处理业务逻辑，而不负责数据持久化，这样做的好处是可以专门为服务器进行参数调优和硬件适配。底层就是数据库服务器，依据用户规模的大小可以做成分布式，由专门的数据库管理员(DBA)进行管理，负责用户数据的存储和维护。由于分层体系架构的原因，数据库服务器也可以专门进行优化，同时也更方便进行数据备份，主从复制，读写分离，异地热备等工作。

基于B/S架构的应用，有效弥补了C/S架构的不足，能够让开发人员更方便的开发出交互性强、跨平台的应用，同时也大大降低了日后的维护和扩展成本。

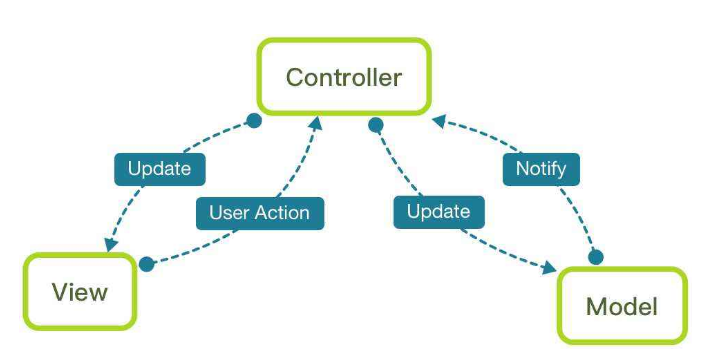
3.3 MVC设计模式

3.3.1 MVC模式简介

MVC的概念是在20世纪70年代由Xerox PARC的Trygve Reenskaug[8]提出。

MVC即“Model-View-Controller”的缩写，中文一般称为“模型-视图-控制器”。

每个部分分别负责不同的功能，其基本原理如图3-3所示。

图3-3 MVC模式基本原理

3.3.2 MVC基本原理

模型（Model）是整个MVC模式的核心，在一个设计合理的软件架构体系中，模型层中应当封装了系统的业务逻辑（关于用户数据持久化的部分，将在本章的下一节介绍）。在经典的MVC模式中，事件由控制器（Controller）处理。每当服务器接收到HTTP请求，控制器将会根据请求路径与请求方法，交给对应的Controller来处理，当Model处理完业务之后，再将结果转发给对应的视图。

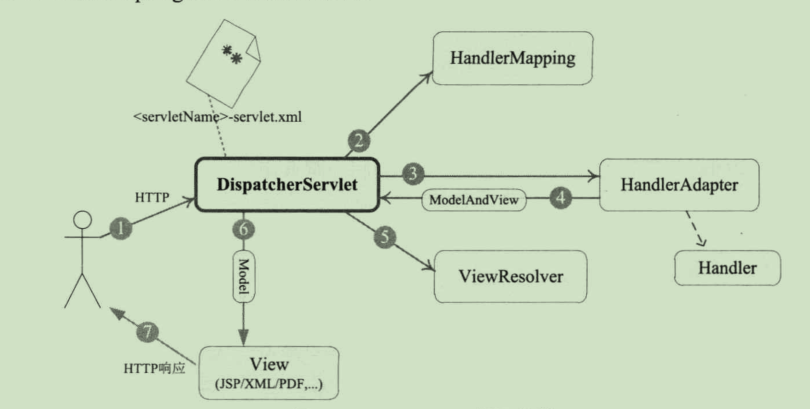
以SpringMVC为例，从接收到请求开始，框架内各个模块组件通力合作，有条不紊的完成各自的工作，直到返回对应的结果，如图3-4所示。

图3-4 SpringMVC框架模型

DispatcherServlet是整个MVC框架的核心，它负责整个框架的各个组件之间的协同配合，下面将详细分析整个流程：

1. 当一个客户端（浏览器）发出HTTP请求，应用程序服务器（通常是指Tomcat）接收到该请求，如果该请求匹配相应的规则，则由服务器将该请求转交给DispatcherServlet处理。这相当于MVC框架的Controller。
2. 当DispatcherServlet接受到该请求后，会根据请求的请求信息（请求路径、请求方法、请求参数、Cookie等），来匹配对应的请求处理器（Handler）。其实这就相当于一个Model。
3. 当请求处理器完成了业务逻辑之后，将返回结果给DispatcherServlet，返回结果是一个包含了视图（View）信息，并根据对应的数据进行视图渲染。

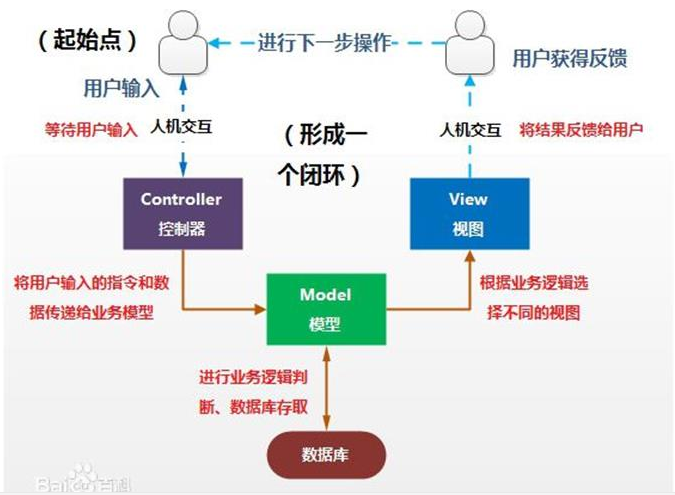
由SpringMVC的工作流程可以看出，HTTP请求不会直接交给model，而是先同意将请求交给Controller，再由Controller根据预先定义的规则转发给Model，如果此时业务逻辑的需求发生变化，我们只需要将Model重新实现即可。本质上来看，Controller的作用就是用来协调Model和View。最终，结合B/S三层架构，可以得到如图3-5所示的架构模型：

图3-5 MVC模式闭环模型

3.3.3 MVC的优势与不足

从上一小节对SpringMVC运行流程的分析者我们可以发现，基于MVC模式的优势有：

* 可复用性高，易维护易扩展：多个View可以对应同一个Model，可以有效增加代码的可复用性，减少代码的冗余。同时，一旦需求发生改变，独立于其它模块的Model也更易于维护和扩展。
* 耦合性低：我们可以发现，Model返回的数据与View层的显示逻辑分离，任一模块的改动都不需要修改别的模块的代码，例如当展示界面的需求发生变化时，我们仅需要修改View层的代码即可。更重要的是，由于数据与显示逻辑的分离，我们可以根据自己的需求采用第三方的显示技术，例如，Java界常用的Freemarker、Velocity和Themeleaf模板引擎。
* 更符合软件工程的精神：不同的层相互独立，各司其职，每一层之间又具有极高的内聚性，这使得开发和维护接口的工作量大大降低，能够使后端开发者更加专注于业务逻辑，前端开发者更加集中于表现形式的开发上。同时也有利于通过工程化和工具化的方式来管理代码。

与此同时，虽然MVC模式有这么多好处，但也带来了一些负面的问题，其缺点主要是：

* 完全理解MVC的设计模式比较复杂：我们并不能保证所有开发者都具有良好的实践经验。使用MVC模式需要大量进行的设计与计划，这才一定程度提高了准入门槛。
* 不适合中小型的应用程序：对于简单的应用程序，如果强行使用MVC设计模式，使得各个层之间分离解耦，反倒增加了结构的复杂性，使得测试困难，增加过多的冗余代码，降低系统运行效率。

3.4 ORM

3.4.1 关系模型与对象模型的阻抗不匹配问题

目前为止，大部分网站还是以传统关系型数据库（如MySQL，Oracle，SQLServer等）为主，存储的是结构化的关系数据。一般来说，这些关系数据的静态结构由一下内容组成：

* 一个或多个数据表
* 表的索引
* 视图和触发器
* 表与表之间的完整性参照（即通常说的外键）

例如我们要保存一个用户信息，往往需要保存用户名、密码、年龄、性别等信息，这体现在数据库中就是一张数据表，对应的各个属性就是表的字段。如图3-6所示。

图3-6 用户信息表

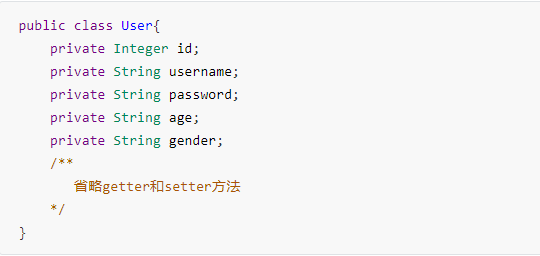
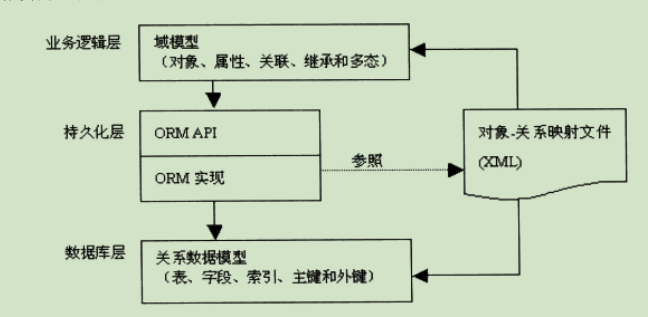
而目前为止，大部分软件的开发工作都是以面向对象（OOP）为主，对象是对真实世界的实体的软件抽象，比如上例中的员工，在高级语言中一般都表述为一个实体类，并且它的成员变量对应相应的属性。以Java为例：

图3-7 Java中的User对象

显然，我们需要持久化的对象，与最终需要存放到关系型数据库里面的结构是不匹配的，这就是阻抗不匹配问题。上述示例只是最简单的情况，如果对象内还含对别的对象的引用，情况将变得更加复杂。于是为了解决阻抗不匹配问题，出现了所谓的ORM（即Object-Realation Mapping，对象-关系映射）。

3.4.2 对象-关系映射

对象-关系映射指的是在3.2节中所述的三层架构中，由数据持久层负责所有数据域实体对象的持久化，封装具体的访问细节。并且为对象，关系型数据库的数据表之间提供映射，让开发人员可以像操作对象一样来操作数据表。其基本原理如图3-8所示。

图3-8 ORM充当业务层和数据库层之间的桥梁

ORM解决的主要问题就是对象-关系的映射。开发人员在开发软件时是面向对象的，而关系型数据库是建立在关系表之上的，一般来说，一个实体类对应关系型数据库表中的一条记录，表3-1阐述了面向对象和面向关系之前的基本映射。

|  |  |
| --- | --- |
| **面向对象概念** | **面向关系概念** |
| 类 | 表 |
| 对象 | 表的行（即记录） |
| 属性 | 表的列（即字段） |

表3-1 对象-关系的基本映射

市面上常见的ORM产品大多采用元数据来描述对象-关系映射的具体细节，例如Java届的全自动ORM框架Hibernate[8]，半自动框架Mybatis。也有部分框架提供了开箱即用的特性，比如Django框架的ORM映射，无需自己编写对象-关系映射文件。相比较而言，这些类型的ORM中间件各有各的好处。需要开发人员自己编写ORM映射文件的产品，往往是大而全，事务管理、缓存控制等一应俱全。而且由于ORM映射文件的存在，开发人员可以在极小的粒度上精确控制SQL语句，有利于性能优化，唯一的缺点就是需要编写复杂的映射文件。而对于Django之类的框架，能够自动映射对象-关系模型，无需配置，开箱即用，缺点在于对SQL语句的控制粒度比较差。

3.5 网页前端

3.5.1 前端技术栈概述

现代主流WEB前端开发主要由HTML5、CSS3、JavaScript技术组成。各个部分各司其职，一同协作才呈现出最终的网页效果[9]。这三个部分承担着不同的职责，比如HTML负责维护网页的元素，控件；CSS负责维护各个元素的外观样式；JavaScript则是运行在浏览器上的编程语言，通常负责交互逻辑，页面DOM操作、浏览器BOM和前后端异步通信（AJAX）。

3.5.2 响应式页面设计

响应式页面的概念最早是由Ethan Marcotte提出的：WEB页面能够根据不同设备显示屏的尺寸自动适应和调整。传统的网页设计，对于不同尺寸的设备，可能需要单独重新设计一套页面，但是如果使用响应式页面，就可以使用一套页面来应对不同的终端设备，大大减少了开发工作量，同时也提升了可维护性。

有别于前几代HTML标准，HTML5标准的提出，让前端网页技术发生了翻天覆地的变化。HTML5和CSS3提供了诸多强大的新特性，如更多语义化的描述性标签、良好的多媒体支持、更强大的文档间消息通信、WebSocket和浏览器本地存储等。这些新特性给网页开发带来了极大的便利，以响应式布局为例，CSS3提供了所谓的媒体查询，可以方便的获取设备的尺寸和分辨率，允许开发者为不同的设备提供对应的样式：

图3-9 CSS3媒体查询

与此同时，谷歌发布了基于Chrome V8引擎的Node.js，使得JavaScript可以在非浏览器环境下运行，因为其事件驱动、非阻塞I/O模型的特性，NodeJS得到了很快的发展，也几乎是在一夜之间，各个前端框架层出不穷，大大提升了前端的开发效率，为开发者带来了福音。

现代主流的前端框架（如Bootstrap、Foundation等），在对响应式页面的开发上都非常友好。在响应式布局方面，大多采用栅格布局系统，能够同时兼容不同的屏幕尺寸、分辨率，甚至对移动端的横屏、竖屏都提供了友好的支持。

3.5.3 浏览器兼容问题

现在市面上有多种不同类型的主流浏览器，如Google Chrome，Firefox，Opera，Safari等。以W3C（World Wide Web Consortium，万维网联盟）标准来说，上述浏览器都是标准型浏览器，符合国际标准。但由于历史原因，市面上仍存在非标准型浏览器（IE浏览器），对W3C标准的支持非常差。因此，我们在用不同浏览器浏览同一个页面时，网页呈现的效果会大相径庭，这种情况就是所谓的浏览器兼容问题[10]。

在前端开发中，浏览器兼容问题非常常见，主要有：

1. 高度标签，在IE6/IE7等浏览器中的高度会不受控制。
2. 图片默认间距问题，部分浏览器会存在默认的图片间距。
3. 内间距（padding属性）和外间距（margin属性）在不加样式控制的情况下差距较大。
4. 浮动的块级元素，在IE6中显示的外间距margin比设置的大。
5. IE浏览器有序列表号显示无效。
6. IE浏览器中实际显示的列表宽度与设置的不同。
7. IE浏览器在解析列表内嵌套的块级元素时，会出现错行问题。

针对这些不兼容问题，开发人员提出了多种解决方案，常见的有：采用CSS过滤器，IE条件语句过滤等。

3.6 技术选型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 软件 | | 可选项 | 最终选择 | 选择理由 |
| 开发语言 | Java，  GO，  Python3 | Python3 | Python语言语法简洁灵活方便，非常适合项目开发初期的快速版本迭代，第三方库丰富，能够满足大部分需求。 |
| 服务端开发框架 | Django，  Flask | Django | Django是一个开源的MVC框架，同时提供轻量级的ORM功能，简单的路由配置和易用的模板引擎，能够有效提升开发效率。 |
| 前端开发框架 | Bootstrap，  LayUI，  ElementUI，  Vue，  React | Bootstrap，  LayUI | Bootstrap是由Twitter开源的响应式前端框架，能够简化前端开发。LayUI是一款国产的经典模块化前端框架，主要面向后端开发者，是一个零门槛，开箱即用的前端UI解决方案。 |
| IDE或编辑器 | Vim，  SublimeText，  PyCharm | PyCharm | PyCharm对Python语言支持良好，能够简化环境配置与依赖管理，并且提供静态代码检查。 |
| 版本管理工具 | Git，  SVN | Git | Git是如今最主流的版本管理工具之一，也是笔者最熟悉的版本管理工具。 |

表3-2 技术架构表

第4章 系统分析和设计

4.1 系统实现目的

本论文设计的是一个交通大数据网站，其理论基础是数据驱动的信号配时排查算法，基于Python的Django框架，实现一个易扩展，高可用的交通大数据分析平台。系统底层从MySQL数据库中获取交通数据进行分析处理，借助百度地图开放API，最终呈现可视化分析结果。

4.2 系统功能需求分析

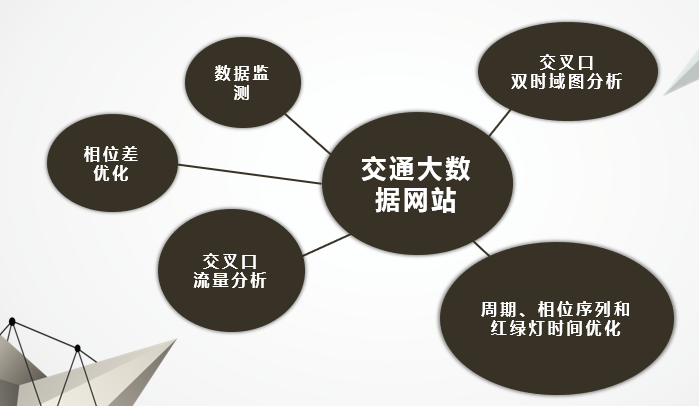
本系统从工程上实现了数据驱动信号配时排查算法，分模块实现算法，呈现结果页面。一共有五大模块，如图4-1所示。

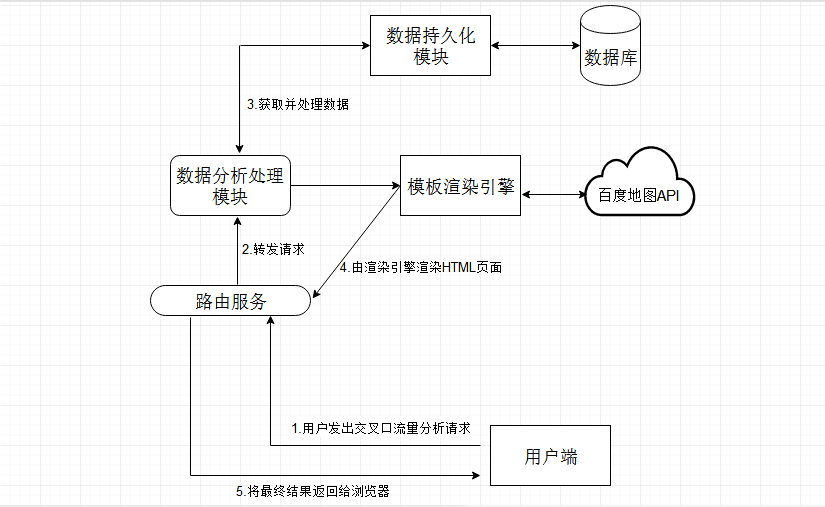
图4-1 交通大数据网站模块分析

1. 数据监测。当用户需要知道某个时段内某个路口的车流情况，本系统接受用户输入，系统从数据库中获取该时段的数据，分析得到结果后，由可视化模块生成图形结果，并在服务器端渲染成HTML页面。
2. 交叉口双时域图分析。在数据驱动信号配时排查算法中，各个路口时域的相位图非常重要。本系统可以指定路口与时间段，获取对应的数据后计算出相位数据，由图片生成模块生成可视化结果，并传送给前端。
3. 交叉口流量分析。在本系统中需要实现一个可视化的流量分析模块，能为用户在地图上以直观的形式呈现路口流量数据，拟采用百度地图开放API。
4. 周期，相位，序列和绿灯时间优化。此模块是系统的核心，系统根据用户指定的时间段和路口信息，依据配时方案相位设计算法，给出优化建议，在提供文字信息的同时，由图像模块生成绿灯优化建议信息，直观的展现给用户。
5. 相位差优化。

4.3 系统模块设计

4.3.1 系统架构

本论文设计的系统主要由4部分组成：数据持久化模块、数据分析处理模块、路由服务和模板渲染引擎。以交叉口流量分析为例，系统监听8000端口，获取HTTP请求，由路由服务转发给对应的数据分析处理模块，调用数据持久化模块获取数据并分析出结果，最终由模板渲染引擎渲染成HTML页面，返回给用户端，如图4-2所示。

图4-2 交叉口流量分析 数据流图

4.3.2 数据分析处理模块

数据分析处理模块接受由路由服务转发过来的请求，从请求中提取出相关的业务数据，根据预先定义的规则，调用模块中封装好的方法。该模块基于Django框架实现，使用ORM中间件从数据库中获取原始数据并进行转换，转换后由数据驱动信号配时算法得到最终结果，再将结果封装后交由模板引擎进行渲染，当模板引擎渲染完成后，相应请求方。

4.3.3 数据持久化模块

本系统的数据持久化模块基于Django的ORM，可以较为容易地实现一对多，多对多关联查询和反向查询。为了对查询结果的控制粒度更加细致，本系统绕过了Django对CRUD方法的封装，自己编写SQL语句以实现更加高效、精准的查询。

4.3.4 路由服务

系统的路由模块基于Django的路由分配系统。与一般的MVC框架类似，Django也在urls.py中记录了系统的路由列表，当请求到达系统时，系统会先遍历该路由列表，匹配对应的处理函数。

4.3.5 模板渲染引擎

本系统采用Django自带的模板引擎。模板从内容上说只是文本内容，用于分离文档的表现形式和内容。Django自带的模板提供了占位符、逻辑处理等基本特性，能够满足项目开发需要。

第5章 系统实现

5.1 整体概述

本系统作为B/S架构的交通大数据平台，基于数据驱动信号配时排查算法，为用户提供可视化的交通信息。利用城市交通流大数据，实现道路交通信号配时方案智能化、精细化管理。通过对路网产生的海量交通流大数据进行自动挖掘和分析，对信号配时方案进行自动排查、优化、评价和监控，做到辖区内道路口全覆盖与时间全覆盖，准确、持续发现信号控制管理问题并给出改进方案，以提高道路交叉口的通行能力和通行效率，减少交通延误和资源浪费，有效缓解城市交通拥堵。

为了便于管理人员操作，本系统基于Windows10操作系统开发，集成开发环境为PyCharm，采用pip来管理依赖，具体环境如下：

Windows10操作系统

mysql5.7；

acodana3；

python3.6；

pycharm；

chrome浏览器；

acodanna3基本库；

django1.11.4以上；

pymysql 0.7.11以上；

5.2 数据持久化模块实现

5.2.1 表结构

本系统采用的数据库是MySQL5.7，数据库引擎采用InnoDB，默认编码格式均为UTF-8。由于需要对大量数据进行分析，故系统的数据量较为庞大，表架构复杂，有14张关键表，分别为：

* 日期表chart\_chartdate，建表语句为：

CREATE TABLE `chart\_chartdate` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`year` int(11) NOT NULL,

`month` int(11) NOT NULL,

`day` int(11) NOT NULL,

`direction` int(11) NOT NULL,

`Xaxisa` int(11) NOT NULL,

`Xaxisb` int(11) NOT NULL,

`Yaxis` int(11) NOT NULL,

`Alane` int(11) NOT NULL,

`Blane` int(11) NOT NULL,

`Clane` int(11) NOT NULL,

`Dlane` int(11) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

* 数据监测表datacheck，建表语句为：

CREATE TABLE `datacheck` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`originyear` int(11) DEFAULT NULL,

`originmonth` int(11) DEFAULT NULL,

`originday` int(11) DEFAULT NULL,

`endyear` int(11) DEFAULT NULL,

`endmonth` int(11) DEFAULT NULL,

`endday` int(11) DEFAULT NULL,

`intersectionid` int(11) DEFAULT NULL,

`time\_lenth` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

* 路口表i023，建表语句为：

CREATE TABLE `i023` (

`id` int(11) NOT NULL,

`inteid` int(11) DEFAULT NULL,

`direction` int(11) DEFAULT NULL,

`lane` int(255) DEFAULT NULL,

`carplate` varchar(11) DEFAULT NULL,

`passtime` datetime DEFAULT NULL,

`traveltime` int(11) DEFAULT NULL,

`upinteid` int(11) DEFAULT NULL,

`updirection` int(11) DEFAULT NULL,

`uplane` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `vd\_index` (`carplate`,`passtime`) USING BTREE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

剩余的表结构与上述大同小异，限于篇幅，不再赘述。

5.2.2 ORM实现

本系统的ORM基于Django的ORM模块实现。Django官方文档中指出，模型（Model）是关于数据源单一、明确的信息来源。它包含了数据的关键字段和行为，通常，每个模型映射到一张数据表。文档中指出关于Model的几个关键点：

* 每个Model都是一个Python的类，这些类全部继承自django.db.models.Model。
* Model类的每个属性都相当于数据表的一个字段

基于上述两点，本系统为关键的14张数据表都做了ORM映射，以5.2.1节中举例的chart\_chartdate表、datacheck表和i023表为例，其ORM映射实现如下：

* chart\_chartdate表：

class chartdate(models.Model):

year=models.IntegerField()

month=models.IntegerField()

day=models.IntegerField()

direction=models.IntegerField()

Xaxisa=models.IntegerField()

Xaxisb=models.IntegerField()

Yaxis=models.IntegerField()

Alane=models.IntegerField()

Blane=models.IntegerField()

Clane=models.IntegerField()

Dlane=models.IntegerField()

* datacheck表：

class datacheck(models.Model):

originyear = models.IntegerField()

originmonth = models.IntegerField()

originday = models.IntegerField()

endyear = models.IntegerField()

endmonth = models.IntegerField()

endday = models.IntegerField()

intersectionid= models.IntegerField()

time\_lenth=models.IntegerField()

class Meta:

managed=False

db\_table='datacheck'

* i023表：

class I023(models.Model):

id = models.IntegerField(primary\_key=True)

inteid = models.IntegerField(blank=True, null=True)

direction = models.IntegerField(blank=True, null=True)

lane = models.IntegerField(blank=True, null=True)

carplate = models.CharField(max\_length=11, blank=True, null=True)

passtime = models.DateTimeField(blank=True, null=True)

traveltime = models.IntegerField(blank=True, null=True)

upinteid = models.IntegerField(blank=True, null=True)

updirection = models.IntegerField(blank=True, null=True)

uplane = models.IntegerField(blank=True, null=True)

class Meta:

managed = False

db\_table = 'i023'

5.3 数据分析处理模块实现

5.3.1 数据监测

数据监测模块首先接受由路由服务转发过来的HTTP请求，从请求域中获取需要监测的路口ID和起止时间，由此得到对应路口的渠化信息（包含渠化列表和路口ID），并将其存入一个列表中以供后续使用。

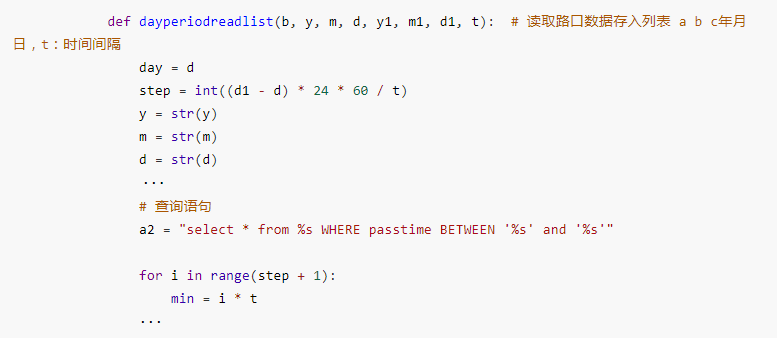
在得到路口渠化信息后，系统从数据库中查询该路口在指定时间段内的车流信息，如图5-1所示。

图5-1 读取路口数据

在得到数据并分析处理后，封装成预先定义好的数据格式，系统将根据数据生成对应的图像，图像生成模块封装了可供直接使用的函数：

picture\_make(time\_lenth, picture\_idtsclane, picture\_makelist) //生成图像

图像生成后，系统将其封装在一个列表中，并交由渲染引擎渲染:

**return** render(resquest, **'datacheck.html'**, context)

上述流程中，最关键的部分就是对从数据库中获取的数据进行处理封装，为了更加方便直观和易于维护，本系统直接采用了Python提供的列表作为数据格式，不仅提升了运行效率，也方便了多个模块之间的数据传输。

5.3.2 交叉口双时域图分析

交叉口双时域图分析是数据驱动信号配时排查算法的基础，也是本系统中最重要的模块。该模块先从路由服务中接受HTTP请求，从请求域中获取需要分析的路口ID，时间段和相位。

系统会先将时间点按方向和转向分类，并由此得到路口的渠化信息并存入列表sidlanelist。该列表的长度是路口总id长度，里面包含了对应sid的列表。为了能够准确的画出最终的图形，我们先定义了一个画点的函数，并在此基础上封装了根据列表内的点自动画出图形的函数。如图5-2所示。

图5-2 绘图函数

5.4 路由模块实现

本系统的路由模块基于Django的路由调度实现，Django官方文档中指出，Django允许开发者自由地设计你URL，不受框架束缚。而且由于路由列表是纯粹的Python代码，因此可以动态构造。其URL调度的基本过程为：

1. Django确定要使用的根URLconf模块。 通常，这是ROOT\_URLCONF设置的值，但是如果传入的HttpRequest对象具有urlconf属性（由中间件设置），则将使用其值替代ROOT\_URLCONF设置。
2. Django加载该Python模块并查找变量urlpatterns。 这应该是django.urls.path（）和/或django.urls.re\_path（）实例的Python列表。
3. Django 依次匹配每个URL 模式，在与请求的URL 匹配的第一个模式停下来。
4. 一旦某个URL模式匹配，Django就会导入并调用给定的视图，该视图是一个简单的Python函数（或基于类的视图）。 该视图通过以下参数传递：
5. 一个 HttpRequest 实例。
6. 如果匹配的URL模式没有返回任何命名组，则来自正则表达式的匹配作为位置参数提供。
7. 关键字参数由路径表达式匹配的所有命名部分组成，并由django.urls.path（）或django.urls.re\_path（）的可选kwargs参数中指定的任何参数覆盖。
8. 如果没有URL模式匹配，或者在此过程中的任何点发生异常，Django将调用适当的错误处理视图。

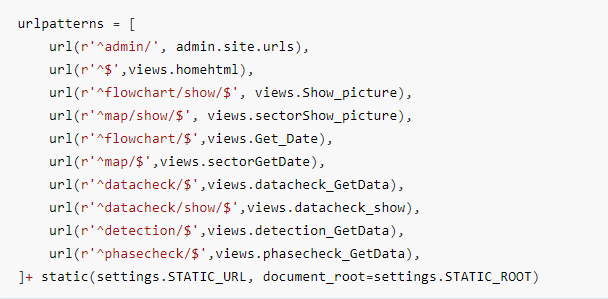
基于上述原则，本系统的路由映射列表如图5-3所示。

图5-3 本系统的路由列表