|  |
| --- |
| 电子科技大学  UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA  硕士学位论文  MASTER THESIS |

论文题目 基于实时流数据平台的车联网数据监控系统

学 科 专 业 软件工程

学 号 201321220110

作 者 姓 名 杨乐

指 导 教 师 顾小丰 高 工

分类号 密级

UDC注1

学 位 论 文

基于实时流数据平台的车联网数据监控系统

（题名和副题名）

杨乐

（作者姓名）

指导教师 顾小丰 高工

电子科技大学 成都

（姓名、职称、单位名称）

申请学位级别 硕士 学科专业 软件工程

提交论文日期 2016.3.18 论文答辩日期 2016.4.19

学位授予单位和日期 电子科技大学 2016 年 6月

答辩委员会主席

评阅人

注1：注明《国际十进分类法UDC》的类号。

|  |
| --- |
| **The Vehicle Networking Monitoring System Based On Real-time Streaming Data Platform** |

A Master Thesis Submitted to

University of Electronic Science and Technology of China

|  |  |
| --- | --- |
| Major: | **Software Engineering** |
| Author: | **Le Yang** |
| Supervisor: | **XiaoFeng Gu** |
| School: | **School of Information and Software Engineering** |

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名： 日期： 年 月 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名： 导师签名：

日期： 年 月 日

摘 要

近年来，随着社会的不断发展，车联网作为物联网在交通方面上的分支迅速发展，各类汽车的数量也在不断增长。不论是汽车租赁公司还是其他各类车辆管理公司，对车辆的统筹管理开始变得越来越重要。因此，一个行之有效的车联网监控系统便变得十分有必要。本论文所述项目跟当前现状进行系统开发，优先阐述了车联网监控系统的研究意义和背景，并且对当前车联网监控统的国内外现状进行了简单的分析。并以此为根据，提出了车联网监控系统的总体设计方案。本文所研究系统主要可分为两个部分：实时流数据处理平台和数据监控平台，其中，实时流处理平台主要负责系统的业务逻辑，数据监控平台主要负责监控数据的显示。

实时流处理平台主要分三个模块，分别负责接收车载终端以及其他终端发送过来的数据的模块，实时数据分类转发以及数块，基于平台的服务扩展模块，数据的存储则使用非关系型的数据库MongoDB。论文主要完成了基于实时流处理平台的车联网监控系统，根据系统高并发接入I/O以及实时性的需求，将数据接收部分和数据处理部分分离开来，系统开发采用node.js的异步编程语言，并通过redis的pub/sub机制进行通信，非常有效的解耦了接收端和处理端，除此之外，系统将根据具体需求进行应用开发，如车辆状态监控、位置信息查询，用户服务等。实时流处理平台根据数据的去向分类，对不同的数据分开处理，对需要实时显示的数据不会做复杂的计算，对需要存储落地的数据则进行相应的按特征存储，对请求的服务状态码则调用服务模块相关方法实现。

数据监控平台基于轻量级的网络框架flask搭建，采用JavaScript作为前端网页脚本语言，并使用backbone.js的MVC（Model-View-Controller）框架来管理JavaScript文件。数据监控平台主要面向监管人员开发，需要完成状态监管、位置跟踪以及路径规划等基本功能,系统开发过程中调用高德api作为地图页面，并融合使用ajax异步请求加载技术实现监管人员与页面的实时互动。

基于上述的功能需求，本文一一阐述了系统的详细设计与实现，并对其功能和性能进行了测试，测试结果均达到要求。

**关键字：**车联网，流数据处理，监控系统，实时显示，模块

ABSTRACT

With the development of society last several years, Internet of Vehicles(IOV) get the rapid development as the branch of Internet of Things, the number of cars become more and more. It becomes more and more important for companies of Automotive Management to monitor the cars, as a result, a effective surveillance system for Internet of Vehicles is very necessary. The project stated by this thesis exploit the surveillance system according to the [present situation](http://www.baidu.com/link?url=rOoQOQMDBxg9-ugMi5f3erR9Ag4cRHW5KmmG4Ea9q3OuApsGlTfyUKVHOAXc-dQ8cUrrYJIfvVxLZ7WKCINgQDEuAqAkb8kqR2osW7fE_70S1LRDuZd5hwPovyL9SOFI" \t "_blank). The thesis first expounded the background and significance of this project, and then simply analyzed the current status of the surveillance system of IOV in domestic and foreign, the thesis put forward the scheme based on these. The content of research of this thesis can be divided into two portions: Real-time streaming data processing platform and Visualization platform, among them, the Real-time streaming data processing platform mainly process the business logic of the system, and the Visualization platform is responsible for displaying the data.

Real-time streaming data processing platform can be divided into three modules: they are responsible for receiving and forwarding the data from vehicle terminal, processing, storing and forwarding the data, and the service module based on processing platform, the system use the MongoDB to store the data. The project mainly complete the surveillance system of IOV based on the Real-time streaming data processing platform. According to demand of high concurrency and real-time response, the system use node.js programming language and redis’s pub/sub communication mechanism to separate the receiver and sender, and this can be validated very effective. Besides of these, the system will develop the application according to the specific demand, such as vehicle condition monitoring, location information inquiry, user-oriented service and so on. The Real-time streaming data processing platform separately process the data, which classify according to the whereabouts of data: the data to be displayed in real time will not do complex calculations, data needs to be stored will be stored according to their characteristics, and service module will call the corresponding service’s method according to the status code.

System build the Visualization platform based on lightweight web framework flask, and use backbone.js, which is a MVC framework, to manage the JavaScript files, JavaScript is a scripting language of web-front. Visualization platform primarily develop for regulators, it need to complete state supervision, location tracking and route planning and other basic functions, In system development, the system called Gaode map api to load the map page and fusion using ajax asynchronous requests loading technology to achieve real-time interaction regulators and pages.

Based on the above functional requirements, this thesis elaborated on the detailed design and implementation of the system below, and be tested to verify its effectiveness.

**Key words:** Internet of Vehicles, stream data processing, surveillance system, real-time display, module

目 录

[第一章 绪 论 1](#_Toc445582164)

[1.1 车联网监控的研究背景及意义 1](#_Toc445582165)

[1.2 国内外研究现状 1](#_Toc445582166)

[1.3 论文主要研究工作 3](#_Toc445582167)

[1.4 论文章节结构概述 4](#_Toc445582168)

[第二章 系统平台及关键技术 5](#_Toc445582169)

[2.1硬件终端 5](#_Toc445582170)

[2.1.1 车载终端OBD 5](#_Toc445582171)

[2.1.2 信息显示终端 5](#_Toc445582172)

[2.2实时流数据处理平台通信基础 5](#_Toc445582173)

[2.2.1 节点集群 5](#_Toc445582174)

[2.2.2 基于redis的pub/sub通信机制 6](#_Toc445582175)

[2.3 系统软件技术及相关应用框架 7](#_Toc445582176)

[2.3.1 node.js语言 7](#_Toc445582177)

[2.3.2 Redis & MongoDB 9](#_Toc445582180)

[2.3.3 数据监控模块框架技术简介 13](#_Toc445582183)

[2.4 本章小结 15](#_Toc445582186)

[第三章 系统需求分析与总体设计 16](#_Toc445582187)

[3.1 系统需求分析 16](#_Toc445582188)

[3.1.1 车辆状态的实时信息监控 16](#_Toc445582189)

[3.1.2 指向性服务推送 18](#_Toc445582190)

[3.2 系统设计 21](#_Toc445582193)

[3.2.1 数据通信设计 22](#_Toc445582194)

[3.2.2 系统模块详细设计 24](#_Toc445582197)

[3.3 数据库设计 29](#_Toc445582200)

[3.3.1 数据热备份 29](#_Toc445582201)

[3.3.2 数据库访问优化策略 30](#_Toc445582202)

[3.4 本章小结 32](#_Toc445582206)

[第四章 实时流数据处理平台的设计与实现 33](#_Toc445582207)

[4.1 实时流数据处理 33](#_Toc445582208)

[4.1.1 实时流数据采集 34](#_Toc445582209)

[4.1.2 数据过滤 39](#_Toc445582213)

[4.1.3 数据分类处理 41](#_Toc445582214)

[4.2 数据库设计与实现 45](#_Toc445582217)

[4.3面向服务模块 48](#_Toc445582218)

[4.4 本章小结 51](#_Toc445582219)

[第五章 实时监控平台的设计与实现 52](#_Toc445582220)

[5.1 实时监控平台概述 52](#_Toc445582221)

[5.2 实时监控平台设计与实现 53](#_Toc445582222)

[5.2.1 地图加载 53](#_Toc445582223)

[5.2.2 数据交互 56](#_Toc445582224)

[5.2.3 主要功能实现 58](#_Toc445582227)

[5.3 本章小结 64](#_Toc445582228)

[第六章 系统测试 65](#_Toc445582229)

[6.1 测试条件准备 65](#_Toc445582230)

[6.2 实时流平台功能测试 66](#_Toc445582231)

[6.3系统性能测试 68](#_Toc445582232)

[6.4本章小结 69](#_Toc445582233)

[第七章 总结与展望 70](#_Toc445582234)

[7.1 本文总结 70](#_Toc445582235)

[7.2 对未来工作的展望 70](#_Toc445582236)

[致 谢 72](#_Toc445582237)

[参考文献 73](#_Toc445582238)

[攻读硕士期间取得的学术成果 75](#_Toc445582239)

第一章 绪 论

1.1 车联网监控的研究背景及意义

车联网，是应国家智能交通的号召将物联网技术应用于交通领域的一种具体的形式，它拥有物联网很多的当代通信技术，基于全网互联的形式来实现车人和车车的实时通信[1]。车联网已经经过很长的一段时间的发展，当前车联网通信基本上已经拥有一种统一的通信规范，可以将车辆的各类信息通过无线的方式接入到互联网，这些数据对于车联网的发展具有举足轻重的作用，其中车联网监控技术便是这些数据的一种具体应用形式[2]。当前在汽车消费不断增长的情况下，对车辆的统一监管已经成为一种车联网应用的需求，它能有效的实时的对车辆的状态和位置信息进行监控并根据当前的状况采取相应的措施。因此，研究车联网监控相关关键技术，并基于其关键技术开发相应的核心产品，必定可以给当前社会带来巨大的社会效益[3]。

当前，随着互联网和车辆技术的不断发展，我国已跻身世界汽车消费大国，车辆数的不断增长，则需要与之配套的监管设施来对车辆资源的合理调度[4]。现阶段，人为的对车辆进行监管已经很难控制当前不断增长的车辆数，最好的方法便是将车辆与互联网联合起来，通过车联网终端技术接入互联网，车载终端会将车辆的属性信息发送到互联网并由控制中心单元进行接收，这样监管人员便通过显示终端对多数车辆进行状态监控。一般情况下，信息来的来源是通过车载终端以及基站、GPRS定位产生的信息来进行，并且随着互联网的发展，车辆与车辆、车辆与人之间的交互变得更加灵活。总之，车联网监控系统具有深远的研究意义，可以有效的调配交通资源，提高车辆监管的效率[5,6]。

1.2 国内外研究现状

车联网实际是要构建一个智能交通网络，这种交通网络旨在能够实时准确的实现车辆的调度，应急管理以及一些其他方面的应用。智能交通系统（Intelligent Transport System，ITS）则是一个建立于这个智能交通网络基础之上的服务管理系统，通过为车辆配置车载终端来实现车辆数据信息的采集，并通过这个只能交通网络将数据发送到互联网，那么具体的智能交通系统便可以通过获取这些网络上的交通数据来定制相关的服务，如实时监控、调度管理等[7]。在这样的大环境下，智能交通网络应对的重点便是完善这种实时的准确的网络通信传输。目前，国际上主要有美国、日本和欧洲的三大组织在推动 ITS 的发展[8]。

在国际上，日本是最早开始研究车联网通信技术的国家，美国尤其重视车联网技术的发张，欧盟等很多发达国家都也都已经建立了较为稳定的车联网系统，实现了车车之间，车路之间以及车人之间的稳定通信。其中美国和日本研究相对比较靠前，其智能交通系统已经可以基本实现车辆的监管以及合理调度。随着WAVE无线通信技术在全球范围的广泛运用，车联网系统又得到了更加稳定的发展空间[9]。在美国，2013年下半年硅谷技术界将具有联网功能的通信装备配备到汽车上，各大汽车企业都在硅谷拥有自己的研究所，开始倾向投资联网汽车方向；此外，无人汽车也渐渐进入研究环节，在富士通美国实验室于2013年6月举行的研讨会上，位于硅谷的奇点大学的拉德·邓普顿（Brad Templeton）教授演示了机器人汽车，这是一次机器汽车驾驶的尝试，这对于车辆状态的监控以及网络通信的需求都很高，研讨会上指出，机器可以迅速对低位计算作出响应。随后一个月，福特公司发表联网汽车的战略，期望能缓解饱和道路的压力，并且随着汽车技术和IT技术的不停发展，汽车行业已经盯上了互联网。从全球看来，2016年2月22日，在巴塞罗那召开的世界移动通信大会上，豪华车商保时捷、奥迪和中国的汽车厂商吉利在同一天发布了有关“[车联网](http://auto.china.com/baike_6L2m6IGU572R.html" \t "_blank)”的最新合作协议，显示出汽车公司加速布局车联网领域的迫切心情，这种迫切推进车联网合作的压力主要来自2月4日谷歌发布的无人驾驶技术，这意味着互联网汽车技术将会达到一个新的高度，在市场上，福特等大型汽车企业也加紧进军P2P、分时租赁等共享业务新模式，这些新兴的汽车服务模式，都必须依赖于当前的车联网技术。

我国车联网相关技术虽然起步较晚，与其他发达国家相比，相对比较落后，但随着近年来我国城市化进程加快，中国的汽车消费已经成为全球第一大国，并且仍然处于快速增长的状态[10]。伴随着互联网技术的不断发展，对车辆管理的需求日益增强。2010年7月北京召开的第九届中国信息港论坛“车联网”产业链合作研讨会以及2012年7月有中国交通运输部主办的第三届节能运输大会发布的《交通运输行业智能交通发展战(2012—2020年)》都充分体现了我国正在讲车联网应用系统逐渐提上日程，尽管起步相对于其他发达国家较晚，但是我国是一个庞大的发展中国家并且拥有最大汽车消费市场，这些都将刺激我国车联网相关技术的蓬勃发展。2016年世界移动大会期间，中兴通讯宣布与北欧领先的运营商TeliaSonera签署战略合作协议，正是进军车联网；国内各大互联网公司也相继进军车联网，目前阿里巴巴以Yun OS操作系统为基础，携汽车电商以及高德地图等，在互联网汽车方面布局。腾讯以车联开放平台为基础，通过车载ROM、车载APP和“我的车”在车联网方面布局。而百度，除了携百度地图和之前推出的Carlife，通过百度大脑，直接进军自动驾驶[11]。

本文就当前发展形势，就车联网监控管理方面提出可能可行的车联网监控系统设计方案并予以实现。

1.3 论文主要研究工作

本文主要研究基于实时流数据平台的车联网监控系统的设计与实现，并基于我们科研团队与四川欧润特软件科技有限公司具体合作项目进行阐述，主要面向车辆监管人员对系统进行开发，考虑到各类需求，系统主要研究工作如下：

1. 研究分析如何满足可能会面对的高并发连接请求，在系统的应用过程中，监管的车辆可能会从最开始的几十辆到后面的几百辆甚至更多，面对潜在的高并发连接需求，系统在开发过程将接受连接的模块单独划分出来，基于node.js异步编程原理以及事件循环的调度机制，采用master进程和worker进程的任务分配方式，master负责监听连接请求，worker负责具体的I/O工作，master会时刻监听着worker的状态。
2. 对于可能的潜在的异常数据，流处理平台在对数据分类处理之前需要对接收到的数据进行过滤，这种过滤可以保证最基本的数据可用性，提高系统运行效率。系统将基于一定的可扩展的过滤规则，对接收的数据进行过滤。
3. 面对杂乱的数据，系统需要对其按特征和需求进行分类：对于需要及时反馈的信息会通过网络或者信息平台反馈驾驶员；对于需要落地存储的统计类数据，按照一定的特征以及功能需求进行存储；对于需要实时转发显示的数据，如GPRS，系统将不会进行复杂计算以提高实时性，通过事件捕获机制进行转发。
4. 对于信息显示端，系统将基于轻量级前端网络框架flask开发一个独立的数据监控平台。系统将使用高德地图api实现地图的加载，并通过jQuery和ajax技术实现监管人员和页面的实时互动。
5. 系统在开发过程中需要满足模块与模块之间、平台与平台之间的数据通信。系统将使用两种通信方式：一种是基于redis的pub/sub通信机制，有效的降低了模块之间的耦合度；另一种是基于稳定连接的socket机制，保证了平台之间通信的实时性。

综上所述，系统将主要基于以上五个部分进行开发，并给出详细的设计与实现方式，最后对系统的功能和有效性进行总的测试。系统已经经过实际应用测试，并且已经初步应用到了实际生活环境中。

1.4 论文章节结构概述

本文的章节组织大致如下：

第一章，主要介绍了车联网的背景，车联网在国内外的发展现状概况，对将车联网相关技术引入到汽车监控行业的研究及应用现状进行了分析，然后简要介绍了论文的主要工作，最后对文章的结构和主要研究内容进行了总结。

第二章，主要介绍工程项目过程中所需要的软硬件平台以及所用到的关键技术进行介绍，主要的关键技术有：实时通信基础，实时流数据处理相关软件技术以及系统相关应用框架。

第三章，系统的总体设计方案，包括对当前状况的需求分析以及系统各部分功能模块的设计。

第四章，实时流数据处理平台的详细设计与实现，主要从数据接收模块、数据处理模块以及服务扩展模块进行详细分析。

第五章，主要针对监控系统的显示页面进行详细设计与分析，包括基于flask的系统的设计与实现，基于ajax的数据传输的实现以及页面效果展示的实现。

第六章，对系统分别进行功能和性能测试，功能测试主要针对系统是否能运行，运行效果是否满足需求的方面进行；性能测试则主要针对数据监控平台的响应速度以及实时流处理平台的数据处理速度方面进行测试。最后以展示效果和表格结果的形式来验证系统有效性。

第七章，对项目的各部分内容的总结以及对项目前景的展望。

第二章 系统平台及关键技术

2.1硬件终端

2.1.1 车载终端OBD

项目使用的车载终端是OBD车载诊断系统，用来监控车辆行驶过程中的各类状态信息，OBD已经历经了三代的发展[12]。

OBD开发之初，各个厂商各自拥有自己的软硬件技术，在相互独立的开发过程中，虽然功能基本能够达到标准，然而厂商硬件之间无法互通，并且调用接口以及各类状态码没有一个统一的规范，因此第一代的OBD不是真正意义上的车联网终端设备。直到第二代的出现，各厂商生产的OBD的通信协议以及调用接口和状态码的规范得到了统一，初步实现了现实意义上各个车载终端的互联，然而功能仍然欠缺，功能完备性有待扩展。第三代OBD的出现，是只能车载终端出现的标志，它在第二代的基础之上非常有效的扩展了服务应用，不仅拥有统一的通信协议和编写规范，还大大减小了车载终端的大小，更为方便的应用到了车联网终端设备中，有效的推动了智能交通网络在实际交通中的应用。

总之，本文所述的工程项目中将会用到现阶段最新的OBD硬件终端，本文所做实验中所有数据的来源也都是通过OBD获取的。

2.1.2 信息显示终端

主要的硬件显示终端就是显示器，在工程项目里，使用监控大屏幕作为显示终端，在实验中用PC机的显示器来实现。

2.2实时流数据处理平台通信基础

作为本文的核心部分，实时流数据处理平台实现了对大量无法预测的数据的实时计算以及实时入库，将数据以流的形式进行处理并通过流的形式进行转发，主要包括的技术有：nodejs节点集群[13]和基于redis的pub/sub通信机制[14]。

2.2.1 节点集群

本文所述系统中主要用来处理I/O密集型业务，因此使用的的信息处理方式属于异步I/O的方式，node.js作为一个天然的基于异步I/O的编程语言便成为了开发系统的首选，但node.js本身也有自己的缺陷，其中为了实现系统的高响应度，在大量数据流输入的情况下必须有即时的数据获取以及处理模块，这就需要集群的支持，然而在node.js中使用集群还是不容易的，JavaScript是一种单线程编程语言[15]，这让基于其规范结构开发的node.js很难充分利用现阶段多核的硬件系统，为了克服这种困难，nodejs内部使用了自己的集群cluster技术。

Node.js继承了JavaScript的单线程特性，因此，对于开发人员来说所开发程序的所有操作都是基于一个主线程的，然而这并不是意味着node.js只能进行单线程操作，我们需要明白的是node.js非常适合处理庞大的I/O密集型事务并且有很高的响应度，很显然这不是一个单线程可以完成的。Node.js不仅继承了JavaScript的单线程特性，同时内部也封装了多线程I/O的实现，其底层基于libuv的原理，依靠事件循环来驱动运行，这种异步机制可以有效的处理I/O密集型业务[16]。

Cluster模块之所以叫做cluster，是因为node.js内部维护了一个节点集群的实现，cluster会将线程分为两类，一个是master线程，另一个是worker线程。同时，cluster模块在设计过程中分别对两种线程封装了不同的方法，比如master拥有fork()方法来创建worker线程，worker可以监听message事件来获取传入的数据信息，充分利用node.js的cluster模块可以有效满足高并发连接请求的需求。

2.2.2 基于redis的pub/sub通信机制

Redis是一个内存数据库，其优缺点显而易见，优点在于存取速度快，缺点是空间内存受限[17]。基于它的优缺点，一种基于redis的通信机制充分利用了它的优点并且有效的避免了它的缺点，redis的pub/sub机制。



图2-1 redis的pub/sub通信机制示意图

Redis的pub/sub通信机制是一种类似观察者模式的通信方式，发布端和订阅端形成一种多对多的关系，而redis所充当的角色是一个消息服务器，发布者可以往这个消息服务器发布消息，而订阅者可以从这个消息服务器中获取需要的消息，订阅者和发布者的角色也可以随时互换，结构示意图如图2-1所示。

这种基于redis的pub/sub通信机制可以非常有效的解耦发布端和订阅端，通过基于redis内存的消息服务器，发布者和订阅者之间不需要有稳定的连接，发布者通过连接redis消息服务器来往服务器发布消息，而订阅者同样通过连接消息服务器来订阅相关的数据，订阅方式一般是基于内容或者基于主题的。

Redis的pub/sub通信机制的低耦合特性主要体现在两个方面：一方面是空间非耦合特性，发布者和订阅者不需要实现知道对方是否在同一台机器或者不同机器，发布者仅仅通过redis维护的时间服务将数据发布到redis的通道中，同样订阅者可以通过事件通知服务来获取需要的数据；发布者和订阅者之间都没有对方的直接引用，订阅者不需要知道消息是谁发布的，同样发布者也不需要知道有多少订阅者在订阅该消息；另一个方面是同步非耦合特性，发布者和订阅者都是基于异步的方式进行通信的，发布者可以不停的发出不同的事件服务，而订阅者则是通过回到的方式来获得事件服务通知，并且调用相应的回调函数。结构示意图如图2-2所示：



图2-2 redis事件服务结构示意图

除此之外，redis的pub/sub机制还提供时间非耦合的特性，这需要对数据的持久化存储，因为本文讨论的系统对通信的实时性有较高的要求，因此在这不予以讨论起时间非耦合特性。

2.3 系统软件技术及相关应用框架

2.3.1 node.js语言

Node.js是一个基于chrome V8引擎的JavaScript服务器端语言，基于事件循环机制编程，拥有异步非阻塞的特性，这些特性使得Node.js这门语言非常适合I/O密集型业务的应用场景。

2.3.1.1 chrome V8高性能引擎

V8是google公司开发的一个高效的JavaScript引擎，其开发之初便旨在提高JavaScript的运行效率[18]。当前，在各大网站所使用的引擎中，V8的运行效率比起其他引擎，速度快上数倍。V8非常适用于那些函数程序会被重复调用的应用，其原因主要有以下三点。

1. 快速属性访问

我们知道，java中对象成员变量的访问是通过jvm内部维护的指针直接指向内存的位置，而JavaScript在存储属性的时候一般都是按照字典的方式存储的，这种存储在获取属性是有一个致命的缺点，就是每次访问都需要进行一次字典的动态查找，这种查找会随着属性数的增多而导致效率的急剧下降，因而效率回避java里面的成员对象效率会差很多[19]。

V8对于JavaScript这种缺陷，使用快速属性访问的方法来解决，这种方法的原理是动态的为对象创建隐藏类，对象每次添加属性都会在这个隐藏类中添加一个属性，隐藏类的这个属性便是对象在内存中的位置[20]。示意图如图2-3所示：



图2-3快速访问原理示意图

1. 动态机器码生成

V8没有解释器的存在，通常JavaScript的运行代码将会被直接编译成本地机器码，而属性访问则是基于隐藏类并由一种内联缓存的优化机制来实现。每一次访问到某个对象属性时，V8会找出其对应的隐藏类，同时，V8会预测在未来的对象访问中都是基于这个隐藏类的，于是修改内联代码将其加入内联优化。如果预测正确，只需要一条简单的指令便可以实现属性值的存取；反之，如果失败V8就会修改相应的内联代码。

显而易见的是如果一个隐藏类被许多对象共享，那么V8所带来的性能提升是非常显著的，这也非常切合当前的编程原则（对象重复利用），因此V8在整体上的性能是优于其他浏览器引擎的。

1. 高效分代垃圾回收

V8在垃圾回收过程中主要会遵守三个原则：垃圾回收过程中会中断当前程序的运行；一般V8只会对堆的部分内存进行回收，这样缩短了程序中断的时间，避免长期中断带来的响应延迟等不好的体验；V8总会知道对象和指正在内存中的位置，有效的避免的错误回收带来的内存溢出错误[21]。

这种高效的分代垃圾回收机制原理取自于Java虚拟机新老生代内存分配回收机制[22]，新创建的对象会被移动到新生代的堆内存空间中，而那种过大的或者达到一定年龄的对象将会被移动到老生代堆内存空间中，相应的V8会更新所有指向这个对象的指针。

2.3.1.2 事件循环机制

Node.js是一个基于事件循环机制的编程语言，使用一种基于事件驱动的一种编程方式。在进程刚开始启动的时候，node 便会创建一个不停运行的循环体，每循环一次，我们称之为一个周期。每个周期的循环过程过程其实就是不断检查事件队列的过程：从事件队列中查看是否有待处理的事件，如果有，则取出事件并且在处理事件的过程中不会再调用其他事件。假如事件有其相应的回调函数，则取出回调函数执行；进入下一个循环，如果进程正在占用处理机，则不检查事件队列，如果没有占用处理机，则检查事件队列中是否有待处理事件。倘若不再产生事件，则终止进程运行。

2.3.2 Redis & MongoDB

面对系统高并发I/O以及读写的需求，系统使用Redis和MongoDB两种非关系数据库。两种数据库各自拥有各自的用途，其中，Redis主要负责数据通信与缓冲，MongoDB主要面向数据的存储与读取。

2.3.2.1 基于内存的数据库Redis

系统应用Redis数据库并非面向数据存储的，而是基于Redis的Pub/Sub机制来进行通信的，此外，因为redis是基于内存的数据库，它拥有很高的存取效率并且支持各类数据结构，系统便利用了这一特性，通过其订阅/发布的通信机制实现模块之间的通信，大大降低了模块之间的耦合度，提高了系统的灵活性和可扩展性。

1. Redis常用数据类型

Redis数据库支持各种数据结构类型，这给我在系统开发过程中减少了许多不必要的麻烦，常用的数据类型有String，Hashmap，Set，List，Sorted set。Redis内部实现了这些数据结构的存储方式，除了使用内存提高存取效率外，redis在存储数据定义数据结构的时候也使用的相应的技术来提高检索效率，比如List中的跳表数据结构[23],如图2-4所示：



图2-4 redis跳表数据结构

这种数据结构非常适合redis这种内存数据库，在这种内存环境紧张的情况下系统对它的存储并不需要消耗太多的内存，相对红黑树实现较为简单并且在面向大量并发操作的环境下，这种跳表的数据结构在不损失性能的情况下可以在性能上有很大的提升，因为对跳表这种数据结构的修改都是局部性的，比红黑树相比它在对结构的修改过程中可以仅仅修改局部的结构而不像红黑树那样需要调用旋转操作调整整棵树。

redis这种基于内存的数据库支持的各类数据类型是系统开发变得非常方便，在性能方面因为基于内存也能表现出非常优异的效果，也正因为基于内存，内存空间的限制便成了它应用于系统的瓶颈，因此系统在开发过程中并没有使用redis数据库进行数据的存储。

1. Pub/Sub机制

Pub/sub机制是redis一个非常有用的功能，这种类似于观察者模式的通信方式，可以有效的解耦通信双方，是系统具有更好的可扩展性。

Redis的pub/sub机制使用起来非常简单，并且可以通过各类语言封装使用。redis的pub/sub机制中redis是作为服务器的形式存在的，客户端可以通过连接到服务器后进行订阅或者发布操作，一个客户端既可以是订阅端也可以是发布端。在这个消息通信过程中，redis服务器充当一个类似图书馆的角色，每当有新的消息publish进来的时候，我们便将其放到对应的书架（通道）上，而订阅端只需要从指定书架（通道）中获取相应的信息即可。

1. Redis使用场景

Redis是基于内存的数据，虽然有时候会使用硬盘进行持久性存储，但本文中使用redis主要是利用其实时性特征。Redis所支持的数据结构可以满足许多现有的需求，如字符串可以用来存储普通的key-value对，hash可以存储较为复杂的key-value对，list以及sortedList可以用来存储有序列表等；基于redis内存存储的特征，我们可以存储那些常用的但数据量并不大的数据，有效的提高存取效率；redis还适用于那些实时性较强的系统，比如电商的PV和UV的实时统计等；本文使用的redis的应用便是其pub/sub的通信机制，它能够非常有效的解耦发布端和订阅端使得系统具有很高的灵活性[24]。

此外，上一部分说到的pub/sub的消息通信机制非常适合实时流数据处理系统模块之间的通信，因为是实时流数据，即使数据量很大，我们在做完传输之后便可以将数据丢弃，并不需要对数据进行存储，因此这种数据通信方式并不会占用态度内存，不会给系统内存造成压力，自然也不会成为系统的瓶颈。因此，redis可以充当实时流数据通信的通道，这种基于内存的高速缓存会给实时流数据通信带来很好的性能提升。

2.3.2.2 数据库MongoDB

MongoDB是非关系数据库的一种，因为其开源特性以及很高的存取效率而得以广泛应用于企业数据存储[25]，它拥有非关系型数据库操作简单，代码开源的特点。MongoDB设计之初便是面向大数据存储的，非常适合那些关系不强的数据的存储，同时为了适应关系型数据的存储需求，MongoDB开发了二级索引的方法，更加的完善了MongoDB的存储系统。MongoDB的在应用开发过程中，使用热数据和冷数据的概念，将数据分为两类，热数据一般作为常用数据一般存储在内存中，便于实时存取，冷数据作为不太常用的数据会被存储在本地硬盘上，具体热数据的多少取决于系统内存的大小以及MongoDB的存储算法。除此之外，MongoDB使用独特的分片机制，将数据分块分别放在多个机架上，用来提高数据的索引效率以及安全性[26]。

MongoDB的设计理念是基于分布式来进行扩展的。MongoDB作为数据存储在高并发系统应用中有着非常优秀的表现，如图2-5所示与传统关系型数据库MySQL在并发测试过程中的比较[27]，并可以从中得出相应的结论：

1. 随着并发数的增加，对mongoDB的各项操作，其吞吐量都会有或多或少的降低，但相对于MySQL关系型数据库而言，在并发数不断增加的过程中,MongoDB仍然表现出比MySQL优秀很多的性能。
2. 在相同并发数的情况下，MongoDB的insert操作和update操作的吞吐量大约是MySQL的2-3倍，select操作也不差于MySQL的select操作。

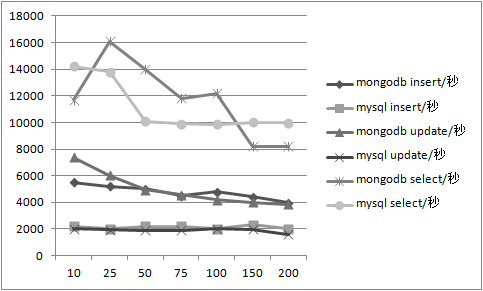


图2-5 MongoDB和MySQL并发测试对比图

因此，MongoDB在高并发应用场景中，性能优于MySQL关系型数据库，这也是现阶段大部分商家喜欢使用MongoDB这个菲关系型数据库的原因之一。



图2-6 MongoDB分片存储机制

如图2-6所示，MongoDB在各个规模企业中广泛应用的另一个重要的原因是MongoDB的分片机制，这是MongoDB基于分布式的一种数据存储方式，是在不影响系统整体性能的情况扩展MongoDB负载的方法[28]。MongoDB分片机制参照当前流行的HDFS文件存储系统原理，是一种安全的高效的并且可扩展数据存储方式，MongoDB是对现有的经典HDFS存储系统的一种改进。MongoDB将数据集合（collection）分解成较小的块数据（chunks），分别存储在每一个分片上，每一个分片相当于HDFS中的一个机架，用来承载这些分解后的数据块，同一个分片中块数据不会重复，不同分片上的块数据按照一定策略进行重复存储以防止万一哪个分片出现故障了而导致的数据丢失，这便实现了MongoDB的单点无故障。此外，这些分片并不需要我们手动控制，我们也不知道具体数据存储在哪个分片上，MongoDB为我们维护了一个总控制节点（mongos），类似GFS中的master节点，这个节点实时观察各个分片的情况，实时了解数据在分片中的分布情况，当应用程序请求需要的数据时，mongos节点便会收集相应的数据并响应应用程序。

分片机制是MongoDB具有了很好的可扩展性和容错性，并且大大提高MongoDB的负载能力。当数据量和负载平衡发生变化的时候，分片机制会自动平衡各分片的数据块量，根据一定的平衡算法重新平衡负载各个分片；当数据量过于庞大超出当前数据库负载能力时，可以实时的添加新的硬件节点，扩大Mongod的负载能力，使得MongoDB能够重新负载数据的存储；由于一块数据会按照一定算法在多个分片上都有备份，所以当一个分片出现故障的时候，分片机制会重新分配数据块，并不会导致数据丢失；控制节点mongos可以拥有多个，分别会控制自己范围内的分片。

2.3.3 数据监控模块框架技术简介

2.3.3.1 Flask轻量级框架

Flask是基于python的WSGI开发的一个轻量级网络应用框架[29]。之所以称之为轻量级，是因为其本身核心很简单，然而简单并不代表不实用。Flask是一个非常灵活易扩展的网络框架，虽然本身不包含数据库抽象层，表单验证等功能，但是你却可以通过扩展选择何种引擎来实现这些功能，完全满足应用系统开发的需求。

Flask作为一个轻量级应用框架，主要依赖两个库：如图2-7所示，一个是jinja2模板引擎，用于加载template模板文件；另一个Werkzeug库，它是[Python](http://baike.baidu.com/view/21087.htm" \t "_blank)基于[WSGI](http://baike.baidu.com/view/1660037.htm" \t "_blank)规范的实用函数库，这个库里面包含了许多flask应用框架中使用的到服务，比如途中的路由机制，由Werkzeug.route维护，连接外部的服务引擎如pymongo也都是由Werkzeug库维护实现。



图2-7 Flask结构示意图

Flask框架主要提供了路由控制，静态资源加载以及模板引擎的功能，除此之外Flask也可以通过Werkzeug库拥有许多额外的功能，如http头部的解析和封装，封装好的request和response对象，兼容各类浏览器，支持与WSGI相关的各类服务器使用工具，会话管理等。这些各种各样的功能除了Flask自带的几个部分以外，其他的都可以在有需求的时候再安装，因此，系统选取了这种轻量级的灵活的网络框架。

2.3.3.2 Backbone.js

Backbone目的在于有效管理复杂的JavaScript应用程序，它是一个经典的MVC框架，为JavaScript应用程序提供了model、collection以及view的结构，使得JavaScript应用程序的更新修改不再会大动干戈[30]。原理图如图2-8所示。



图2-8 backbone.js原理图

Backbone.js是前端的一个MVC框架，主要用来规范管理杂乱的js文件。Backbone使用经典的MVC模式，让html文件和js文件形成了良好的分割，并且规范了数据模型的规范，基于JavaScript基于事件的基础，通过事件驱动应用的运行，同时也提供了丰富的API和内部事件，使得开发变得更加规范有效。

2.4 本章小结

本章主要就系统所用到的硬件平台和相关技术框架进行了研究，首先简单介绍了硬件终端，然后讨论了系统进行实时数据通信的通信基础，再对当前现有的流计算相关技术进行了对比，找出最适合车联网系统的流处理技术，最后我们对系统应用开发过程使用到的软件数据库技术进行了相关的研究。

第三章 系统需求分析与总体设计

论文完成了基于实时流数据处理平台的车辆网监控系统的设计，包括的车辆信息的实时监控以及位置查询的功能。整个系统主要分为三个部分功能模块：硬件终端对接模块，实时流数据处理转发模块，实时数据数据监控模块。其他部分，包括模块通信机制、数据库优化以及基于系统的服务应用，则围绕这个三个部分为中心扩展而来。系统以多进程交互的方式进行实现功能需求，以进程间异步通信的方式进行交流，在保证系统效率的前提下，为系统带来了极大的灵活性。下面主要从需求分析到设计方案再到主要基于的软件平台阐述整个系统的总体设计。

3.1 系统需求分析

车联网实时监控系统，即掌握车辆的实时监控信息，了解车辆的实时路况，接收车载单元传输过来的实时数据，对实时数据进行加工处理转而发送到web前端，监控模块根据获得的数据信息做出相应的行为。

系统主要需要完成的功能是对车辆状态的实时信息的监控，这部分功能主要面向监控端，监控端需要可以随时获取车辆的当前状态信息以及位置信息，除此之外，监控端还需要可以操作调用观察车辆的历史轨迹以及部分关键数据。系统需要满足的另一个功能点是简单的拓展服务，可以通过互联网与驾驶员进行简单的交互，主要包括状态提醒、路线规划以及消息服务等。

在满足这些基本功能点的前提下，系统还需要保证数据的时效性和准确性，主要包括实时数据处理平台对数据的处理转发速度要求、数据监控平台对多请求的响应速度以及对错误数据的丢弃，避免显示平台以及数据库中出现错误数据或者过期数据等情况。因此，在性能上，系统主要需要两个部分的性能要求：第一，在监控页面方面，需要控制监控页面的响应时间在1s以内，平均请求响应时间控制在15ms左右；第二，在一定范围内，实时流数据平台的数据处理时间满足随着数据量的指数级增长，处理时间的增长速度低于线性增长速度。

系统从以上功能需求点出发来开发实时流数据处理平台和数据监控平台，下面对具体的功能需求进行阐述。

3.1.1 车辆状态的实时信息监控

通过GPS全球定位系统确定车辆当前位置以及状态信息，将车辆位置及运行状况信息通过GPRS/CDMA无线通讯网络传至调度管理中心，实现对车辆的实时监控。主要需求设计如图3-1所示：

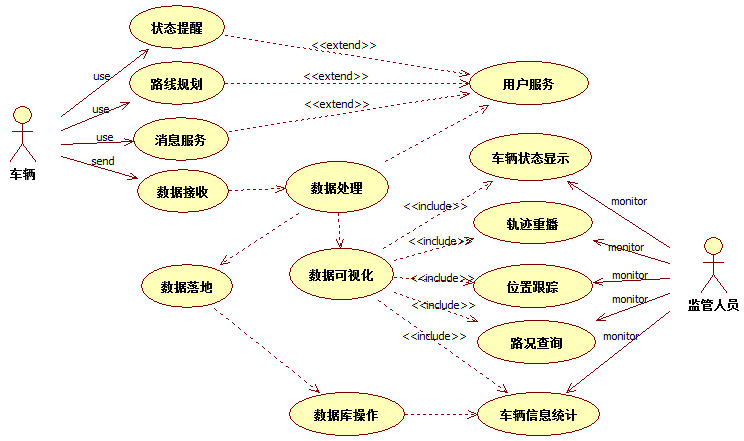


图3-1系统需求设计用例图

1. 车辆监控数据上传接收

数据是系统运行的关键，没有数据的系统没有任何意义，数据是系统所必须实现的最基本的需求。

1. 车辆状态监控及位置跟踪

对行驶中的车辆的位置（经度纬度）进行跟踪监控，除经纬度以外，还需要将车辆的状况信息上传到相应的平台，平台经过处理转发到显示终端，可显示选定车辆的速度、转速、里程和油量等状态信息，如图3-2所示。

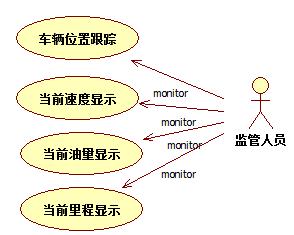


图3-2车辆状态信息监控及位置跟踪

1. 车辆行驶轨迹重播

监控终端可以查询车辆的历史运行轨迹，包括已经行驶的时间，运行过程中停止的地点，行驶速度，行驶距离等各类历史信息，在整个地图上动态显示出历史行车过程，并可以根据设定时间来保存轨迹。

1. 车辆状态实时提醒

对车辆的实时状态进行监控，以状态码的形式发送到监控中心，主要包括车辆超速报警、脱离预定轨迹报警、车辆状态不佳提醒以及车辆紧急情况警告等信息，如图3-3所示。

根据需要对车辆设置限速，车辆行驶过程中一旦超出设置限速，将发送信息到监控中心，监控中心接受到信息后会发送警报信息到车载终端以提醒驾驶员；同样的对于车辆脱离预定轨道监控中心也会发给驾驶员相应的警报信息。

车辆状况不佳时，车载终端会将相应的状态码发送到监控中心，监控中心会根据车辆状态不佳的信息产生相应的提醒信息返回给车载终端并提示驾驶人员。

车辆遇到危险或者紧急情况时，可以触动车内的报警开关，车载终端会立即向监控终端发送求救信息，并且报告车辆的实时状态，监控中心则会显示醒目的提示信息，提醒值班人员该车遇险以实施及时的救援行为。

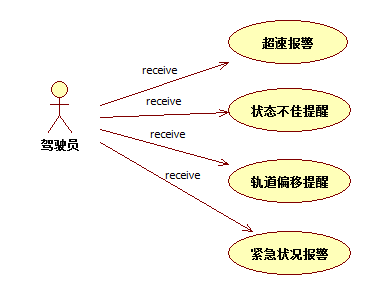


图3-3状态提醒用例图

1. 实时路径规划

可以根据行驶车辆所上传的出发地和目的地，规划出当前的行驶的路线。

1. 实时响应需求

不管对于驾驶员还是对于监控中心，对于车辆状况的实时把握以及中心消息的及时推送对响应度的需求都非常高，因此实时响应也是该系统完成的重要功能之一。

3.1.2 指向性服务推送

作为一个应用级系统，服务信息推送是系统的一个扩展功能，系统主要有两类服务功能：交通信息类服务和网络应用类服务。

3.1.2.1 交通信息类服务

交通信息类服务主要表现为监控中心对实时路况的掌握以及监控中心对驾驶员提供的 及时提醒信息，因此系统的需求大致可以分为以下部分：

1. 交通数据采集

在现阶段车联网系统中，车辆行驶过程中搭载的车载终端OBD可以通过关联车辆各种传感器设备实时获取车辆状态（包括车速，油量等信息），并通国路边单元设施将获取的数据传送的Internet，监控系统通过获取互联网数据实时监控车辆状况，同时路边单元设施可以实时监测固定地点的实时状况，包括收集车辆信息，监控道路拥堵状况以及紧急状况等，路边单元设施RSU会将获得的实时数据一并发送到中心控制单元，由中心控制单元一并处理，如图3-4所示：

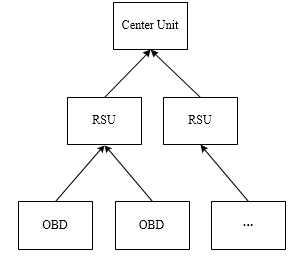


图3-4数据采集示意图

其中车载单元（OBD）和路边单元（RSU）是数据产生节点：OBD节点通过RSU设施连接到中心单元，并通过RSU将数据传输到中心单元处理，等待实时反馈，如果OBD区域范围内无法连接到RSU，将缓存数据，到达指定地点范围后会将数据再发送至RSU；RSU作为固定路边设施可以实时连接到中心单元，监控范围内交通环境并且接受范围内OBD的连接，将路况信息数据以及OBD发送的车辆数据一起发送到中心控制单元。

1. 交通数据分析处理

中心控制单元，也就是本文所开发的应用系统主要的工作便是对数据的分析处理，对交通数据的分析处理也是其功能的一部分，如：监控当前路段的拥堵状况；获取当前行驶车辆状态，包括油量，速度以及车温等；通过统计当前道路车流状况，实施合理调度。

1. 交通数据分析结果决策与执行

中心交通单元会将分析处理的结果封装成决策指令，将指令发送决策受访单位节点，其中决策分为全局决策和区域决策，顾名思义，全局决策会将决策的内容指令以广播的形式发送给全部的RSU，而局部决策则是针对部分区域的RSU发送决策内容；决策受访单位会根据接收到的决策内容执行相应的行为，其中决策受访单位公共交通设备以及RSU等交通管理相关的设备。

3.1.2.2 网络应用类服务

网络应用类服务主要可以分为两类，一类是自主服务，另一类是用户选择服务。第一种属于系统主动推送的服务，比如道路安全信息提醒，旅游地点推荐等；第二种属于用户自主选择系统服务，比如广播订阅，车载wifi等。

1. 系统自主推送服务

系统自主推送类服务主要指广播式的提醒或者警示信息，主要包括道路安全提醒，车辆状态不佳提醒以及旅游景点推荐等，如图3-5所示。



图3-5 系统推荐服务示意图

道路安全提醒，主要表现为三个方面：安全信息公告，周围车辆状态信息，车间间距防碰撞。安全信息公告，如前方事故多发地段，前方道路出现状况等，简单流程介绍，假设车辆行驶过程中遇到道路缺口或者维修等状况，此时车辆会将当前路况传送到中心控制单元，中心控制单元会根据部分配置参数确定广播发送范围，通知附近车辆该路段道路状况，其他驾驶员可以根据获得的路段信息合理安排规划路线。

车辆状态不佳提醒，简单举例来说，如果当前车辆油量低于设定的阀值的话，车载装置OBD会将油量低的信息反馈到中心控制单元，中心控制单元会根据车辆当前位置以及从区域RSU接收到附近路况信息合理规划到附近加油站的路径，以防止车辆中途无法行驶，影响交通管理。

旅游景点推荐，该项功能主要面向自驾游的群体，当车辆行驶过程中，中心单元会获取当前车辆的位置信息，通过地图定位坐标定位，使得当汽车如果行驶到指定范围内时，中心单元会主动向车载设备推送相关旅游景点信息，用户可以根据相关设置选择是否需要服务等。

1. 用户自主选择服务

用户自主选择服务，主要指用户可以根据需要订制自己所需要的服务，主要包括车载小型WiFi源，订阅公告信息服务以及自组小型局域网服务。这部分内容主要属于车载终端的范畴，不是本文主要研究的对象，在此不予以多说，后面内容关于这方面的内容也只会简单介绍下。

3.2 系统设计

本文所研究应用系统主要是面向监管人员实现的监控系统，在实现需求的监控功能的前提下，我们会基于开发的流数据处理平台做一些面向服务的扩展以及数据的落地存储，系统架构图如图3-6所示：



图3-6系统架构图

论文主要完成了基于Internet的实时车辆监控系统的设计，实现了对车辆状态的实时监控以及位置信息的实时查询的功能，此外，对基于该系统的车辆网服务也是系统完成的功能之一。系统主要分为三个部分：车载终端部分，实时流处理平台，数据监控平台。其中本文主要研究的内容是实时流处理平台和数据监控平台，实时流处理平台又可以细分为数据接收，数据处理，车联网服务扩展。系统使用多进程交互的方式，采用异步通信的方式，采用pub/sub以及面向流数据平台的socket通信机制保证了整个系统的通信效率，同时也扩展了系统的灵活性。

3.2.1 数据通信设计

各模块之间数据的交互是建立在数据通信的基础之上的，系统使用数据通信主要有两种方式，一种是基于redis通道的pub/sub机制，也就是数据接收模块和数据处理模块之间的通信方式；另一种是基于面向流计算的socket通信机制，也就是数据处理模块和数据监控平台的通信方式。下面详细介绍两种通信方式：

3.2.1.1 Redis的pub/sub机制

Pub/Sub是一种基于消息服务器的通信机制，消息公布方和消息订阅方可以通过连接中心消息服务器来获取相应的消息服务。这种通信机制旨在解耦消息发布端和消息订阅端，双方可以部署在同一台物理机器上也可以部署不同物理机器上，因此，这种通信机制大大增加了系统的灵活性和可扩展性。

系统在设计过程中便使用到了这种消息通信机制，主要用在了数据接收模块和数据处理模块之间的通信。由于系统需要接收处理的数据量可能会比较大，为了不影响通信模块之间的通信效率，系统在设计过程中需要一个基于内存为通道缓存的消息服务器，而系统的数据接收端和数据处理端分别作为消息服务器两边的消息发布者和消息订阅者，Redis所维护的pub/sub通信方式便适合充当这个基于内存的pub/sub消息通信服务器。



图3-7 redis消息服务器pub/sub示意图

系统之所以使用到这种通信方式，有以下几个原因：

1. 发布订阅双方低耦合

这种消息通信方式可以非常有效的解耦发布订阅双方，甚至在一些特定情况下发布端和订阅端的角色可以互换，发送端内部修改不会造成接收端相应的修改，系统具有很好的灵活性。

1. 具有有较高的可扩展性

既然发布端和接收端双方被有效的解耦，那么系统便可以有多个发布端以及多个接收端，这种高度的灵活性带来的是系统高度可扩展性，系统在有新的数据需求的时候，我们只需要开发一个新的接收模块来接收它然后发布到相应的通道中便可，同时，我们也可以使用多个订阅端订阅同一个或者多个通道的消息。这样，在以后系统扩展应用中便有可能形成多对多的关系。

1. 具有较快的通信效率

系统基于redis的pub/sub机制来进行模块间的通信，redis是基于内存的数据库，内存是所有存储中速度最快的，但空间是最大的限制，系统使用pub/sub机制传输的流数据在redis服务器仅仅暂存一段时间，因为存储时间不长，短时间内数据就会被丢弃，这样便避免了由于内存空间不足所带来的性能瓶颈。

Redis的pub/sub通信机制是将内存作为通道缓存，内部维护一套事件机制来实现实时的订阅发布。内存的存取效率是非常高的，因此，在系统数据流处理的过程中，可以始终保持较高的通信效率。

1. 支持多种数据格式

Redis本身支持多种数据格式，如字符串、键值对、有序集合以及无序集合等，系统开发过程中大量json的数据格式，这是一种典型的key-value数据格式，redis本身也是一种非关系型的数据库，面向集合存储。在通信过程中，只需要对要发布的数据进行简单的转换（json转换成json字符串）便可以发布到redis通道里，数据处理端接收到数据后通过JSON.parse(jsonstr)的方法便可以轻易获取json对象直接使用。

3.2.1.2 Socket通信方式

实时流处理平台是使用node.js的异步编程语言来是实现的，这是一种基于JavaScript语法规范结构扩展的服务器端的JavaScript，不同的是底层拥有自己的线程库，前端我们使用JavaScript作为页面的脚本语言。

数据监控平台本身也是一个基于flask的经典的B/S架构，当数据监控平台需要像统计数据等静态数据时，我们可以通过ajax的方法来请求数据监控平台的server端来操作MongoDB来获取需要的数据，但是，如果我们需要一些实时性比较强的信息，如车辆位置信息，我们则需要更直接有效的方法而不是来回存取MongoDB。系统选择一种面向实时通信的Socket通信方式，这是一种稳定的双向的并且基于事件的实时通信方式，这两种数据获取方式如图3-8所示。

Controller是数据监控平台中管理路由的控制器，可以通过调用pymongo的mongo数据库引擎来对mongo数据库进行操作，pymongo是python封装的一个用来连接操作mongoDB的数据库引擎。而这种socket.io的通信方式是基于事件的通信方式，数据处理模块会以socket.emit(“event”, data)的方式发送一个event事件，而对应的JavaScript部分可以以socket.on(“event”, function(data){})这种回调的方式来等待数据发送完成。因此，socket.io的通信方式提供了客户端和服务端一致的编程体验，不仅如此，socket.io还兼容现阶段大部分的浏览器和客户端设备，在很多情况下，避免了很多不必要的麻烦。



图3-8数据监控平台数据获取方式

3.2.2 系统模块设计

系统整体主要分为两大部分，分别是实时流处理平台和数据数据监控平台。实时流处理平台可以划分主要的四个模块：数据接收模块，数据处理模块，基于数据处理模块的服务扩展模块和数据落地模块 ；数据数据监控平台则是基于经典的MVC结构，由中心控制模块统一控制静态资源、模板引擎以及数据库引擎的加载，根据参数和数据的类型来控制视图的显示。

整个系统结构主要有两个部分：一个是实时流处理平台，主要面向数据的处理操作；另一个是数据监控平台，主要面向数据的显示。实时流处理平台数据处理的逻辑主要包括两个模块：数据接收模块和数据处理模块，数据接收主要是处理并发请求，并向处理端转发数据，数据处理模块主要负责具体的数据处理逻辑，数据的存储及转发等；数据监控平台则主要负责数据处理模块发送数据的显示逻辑，实时的显示数据的变化及动向。其中数据接收模块和处理模块之间使用到的通信方式是redis的订阅发布机制，处理模块和数据监控平台之间使用面向流数据处理的socket.io来实现通信。系统模块图如图3-9所示：



图3-9系统模块设计图

3.2.2.1 实时流数据处理平台设计

1. 数据接收模块

数据接收模块主要负责接收车载单元以及路边单元发送过来的数据，该部分并不对接收的数据进行过滤，这个模块的主要作用是面对每天大量车辆的连接请求，系统必须有一个支持高并发连接的模块来应对这些请求，数据接收模块的作用便体现在这个地方。

如图3-10所示，车载终端和路边单元RSU通过互联网接入数据接收模块，数据接收模块会维护一个连接请求队列，面对高并发连接的需求，模块在开发过程中调用node.js的cluster模块，这是node.js为了充分利用现代处理机多核的特性而开发出来的模块，底层是基于libuv来实现的。Master进程的主要工作有三点：第一，根据CPU核心数来创建worker进程的数量；第二，会不停的循环读取连接请求队列，并将连接请求按照一定的负载均衡算法分配给worker线程；第三，时刻监控worker线程的状态，正在运行或者异常退出，如果有异常退出情况，将会重启worker线程。



图3- 10数据接收模块结构示意图

除此之外，数据接收模块还要面向数据处理模块进行信息转发，这种转发机制是通过每一个worker进程来实现的，如图3-11所示。



图3-11 worker进程与数据处理端通信示意图

每个worker线程会将自己完成的I/O数据进行处理然后发布到redis的通道中，数据处理端通过订阅这个channel获取需要的数据。此外，这种基于redis的通信机制同时可以充当一个缓存的作用，可以有效减缓数据处理模块的压力。

1. 数据分析处理模块

这个模块主要是负责接收对接模块publish到redis通道中的数据，对这些已经做过简单处理的数据进行分析、分类，按照一定的系统需求将数据转发到数据监控平台或者存入mongoDB亦或者部分反馈给用户，如图3-12所示数据处理模块的结构图。



图3-12实时流数据处理模块结构示意图

实时流数据处理模块是一个中间层模块，一方面会订阅数据接收端发布到redis消息服务器的数据，另一个方面会将分类处理好的数据按照需求做相应的转发或者存储。对于从redis消息通道订阅得到的数据，系统在设计流数据处理模块的过程中，又设计了一个小的数据过滤模块，基于一定的过滤规则，过滤掉异常数据或者于系统无用的数据，然后才会由模块的分类处理程序来对数据进行分类。

分类处理模块根据系统的当前设计需求，主要按照数据的三个流向来对数据进行分类处理，主要有三个流向需求：服务扩展需求、实时监控需求和数据落地需求，分别对应着用户得信息终端、系统数据库MongoDB以及系统另一个重要部分实时数据监控平台。因实时流数据处理平台是本系统核心部分之一。

3.2.2.2 数据监控平台设计

实时数据监控平台主要负责接收数据处理模块发送过来的已经经过分类处理的数据流，然后将不同分类的数据以不同的方式在地图页面上展示，前端主要使用到的框架有flask, javascript, backbone.js等，其中backbone.js用来管理杂乱的js文件，是一个比较经典的前端MVC框架，数据监控模块的结构示意图如图3-13所示。

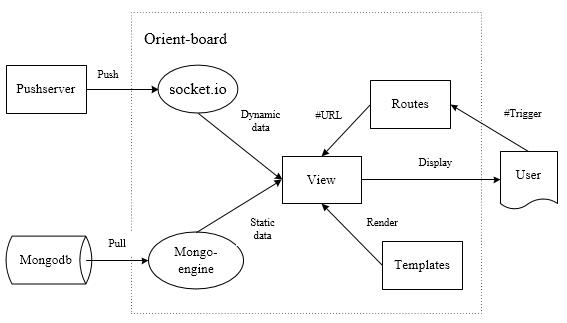


图3-13数据监控模块示意图

OrientBoard是公司项目的名称，这个数据监控平台是一个通用的数据监控平台，可以接收不同平台模块的数据，只要这类数据满足这个数据监控模块所规范的格式。下面根据上面的示意图简单介绍下这个数据监控模块的工作流程：

实时数据数据监控平台模块有两种数据获取方式：一种是push方式（图3-13中的pushserver节点的推动数据的方式），这是一种被动接受的方式，基于一种面向流计算平台的socket通信方式，实现数据的动态展示，可以对一些动态的实时变化的数据进行展示，并可以保证展示界面上数据的变化频率与数据变化速率相当；另一种是pull方式（图3-13中表示可以从mongoDB中拉取数据），相对的这是一种主动拉取数据的方式，图中对应mongoDB我们采用一种叫做MongoEngine的数据库引擎来保证平台与数据库之间的连接以及存取数据，如果使用其他数据库则可以使用其他相对应的数据库引擎，总之，这种拉取数据的方式适合哪种实时性不强，又需要定时更新观察的数据，比如车辆行程等统计信息数据。

除了通信方式以外，该模块自己也是一个完整的系统，flask提供了完整的服务器路由机制以及静态资源路径方式等，它有自己的服务端和浏览器端，拥有自己的页面模板，同时可以集成各类前端框架，这样便保证了前端平台和数据处理平台的低耦合。

这里对前面所述的主要的三个部分作一个简单的总结，这三个部分在车联网监控系统中是相互通信的，他们都以独特的通信方式降低了模块之间的耦合度，使得系统具有高度的可扩展性，三个模块完全可以独立开来应用于其他系统。

3.3 数据库设计

系统开发过程中使用MongoDB这种非关系型数据库，其优点是开源，操作简单并且拥有很高的索引效率。系统主要通过内存映射以及优化索引策略来提高MongoDB的应用特性。

3.3.1 数据热备份

系统在开发过程中所涉及到的数据一般都会有常用数据和不常用数据之分，我们这里分别称之为热数据和冷数据。面对这种数据划分，系统在设计数据库过程中使用热数据备份的形式将其映射存储到内存中，如图3-14所示。

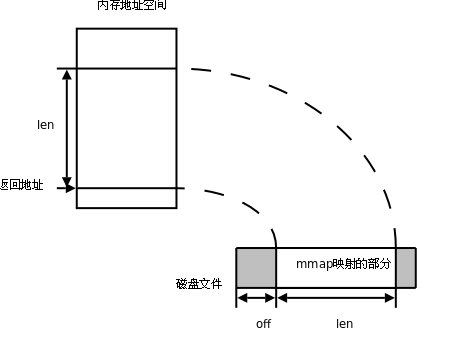


图3-14 MongoDB内存映射示意图

Mmap是操作系统底层所提供的内存映射机制，MongoDB在其开放api上封装了这层功能。系统在开发过程中按时间将数据分类，每当有新的数据注入时，我们将判断它是近日内将会被经常使用的数据，因此将其列入热数据范畴在内存中备份以提高其索引效率。设定阀值为一个月，一旦数据在内存中存储超过设定阀值，系统便会将数据备份从内存备份中删除，减轻内存压力，为新的数据注入提供空间。此外，阀值会按照每月数据量大小的不同，设定不同的阀值，比如，如果当月数据较多，会缩小阀值或者扩大内存；同理如果连续多月数据较少，可以相应的提高阀值。

3.3.2 数据库访问优化策略

数据库对系统的整体性能是不言而喻的，本文在对数据库设计的时候主要是利用MongoDB的基本性能按照需求存储数据来提高MongoDB的使用性能，对MongoDB底层性能优化不作研究。这种提高MongoDB使用性能的策略需要根据具体的需求进行设计。

3.3.2.1 范式与反范式

系统设计初期，明确集合的用途是非常重要的，因为我们需要基于集合数据的用途来设计集合的存储方式。集合的设计则需要根据具体的数据有哪些常用操作，比如对于一个日志文档集合，我们常用的操作便是查询，那么在集合的设计过程中则注重在数据库查询的优化。

1. 范式化设计

这是一种完全分离化的文档集合设计方式，如表4-2和表4-3所示，这两个集合的value都是唯一标识车辆的id信息，我们把速度区间和具体车辆信息完全分隔开来，虽然这种方式会给查询造成额外的开销，但是由于这种小型的细化的文档集合在今后的开发中还会有新的集合加入，如果哪天有车辆信息需要修改，那么我们可以直接修改车辆的信心而不需要修改各个集合文档，因为各集合文档里面只保存了一个不变的唯一标识车辆的id。速度区间对应的车辆id数组分别对应着维护车辆信息的表里的项，而每月总里程同样如此，正是因为车辆的信息被多个集合关联着，我们才使用分离化的文档设计方式，以减少因为车辆信息的改变而带来系统的开销。单条区间对应的存储记录如下所示：

{

“id”: ObjectId(“56d2962fddda1f20567b110c”),

“record”:“speed-based”,

“ids”:[

6214005119007,

6214005119020,

…]

}

1. 反范式化设计

范式化设计比较适合那些被多处关联并且可能经常被更新的数据，与范式化相反，反范式化设计比较适合那些并不会被多处关联且不会有太多更新操作的集合，这些集合数据量往往都比较庞大，需要进行许多的检索操作，如果使用范式化设计存储这些数据会降低检索效率，给系统无故平添很多额外的开销。我们将这些数据通过一个集合来进行存储，一个集合便包括id对应的所有信息，这便是反范式化设计。系统在设计统计数据的存储的时候便使用这种方式。

1. 部分嵌入

部分嵌入的设计方法根据集合的用途需求结合前两种方法使用，对需要被关联较多且需要经常修改的数据集采用范式化的设计方式，而对于那些不经常修改的数据我们采用反范式化的设计方式，取两者的折衷。

3.3.2.2 填充因子

在理解填充因子之前，我们先了解下MongoDB文档的存储方式。MongoDB是文档的存储方式是顺序存储的，这样可以使存储变得更加紧凑，但是如果存储的集合需要经常修改呢？我们知道往顺序表中插入一个新的元素需要将后面的每一个元素后移，如果MongoDB在集合中插入一条新的记录或者修改一条记录也会这样，那带来的系统开销是很大的甚至是毁灭性的。

为了解决对数据修改产生的巨大开销，MongoDB引入了填充因子的概念，填充因子就是预留部分空间以减少文档修改过程中文档移动带来的系统开销，当修改元素移动后，后续插入的文档都会提供一定的填充因子，以便于文档频繁的修改，如果没有不再有文档因增大而移动的话，后续插入的文档的填充因子会依此减小。

系统开发过程中，可能会遇到用户行为习惯的变化，如一位驾驶员从新手变得熟悉开车了，那么他的速度区间可能会提高，这时候我们需要修改两个速度区间所对应的的值，也就是数组，因此这个保存车俩sn号的数据会经常被修改，这时候就需要填充因子的帮助。我们的做法是将集合中usePowerOf2Sizes属性的值设置成true,这样系统为后续文档所分配的空间都是2的N次方，会使后续文档都有一定的额外空间。

3.3.2.3 索引策略

除了集合的存储策略，我们在程序中所进行的索引策略同样会影响到整个系统的效率。假设我们在建立索引时，数据库无法匹配我们的索引条件，那么系统便会进行全集合扫描，这种全集合的扫描对于系统的性能是灾难性的。当然，这种说法过于夸张，但在系统实时应用中，我们还是要规划好我们的索引策略。

1. 减少索引次数

索引能够极大的提高查询的效率，但是如果你过多的使用索引，会给系统造成很多的开销，因为，每当你使用一次索引，系统便会创建相应的索引表，用于索引到指定的位置。而当你对已经建立的索引进行修改的时候，数据库便需要对原来的索引表作出相应的改变，这种改变往往是重新排序，重新排序是非常消耗性能的过程，因此系统只能应对少量的索引。

系统在开发过程中使用复合索引，尽量减少索引的次数，增加索引的效率，在满足索引需求的情况下，使用最少的索引。

1. 减小索引粒度

颗粒越小越好。在索引列中每个数据的重复数量称为颗粒，也叫作索引的基数。如果数据的颗粒过大，索引就无法发挥该有的性能，减小索引的范围。例如，随着注册车辆的数量越来越多，可以根据数据分布特征重新划分速度区间。

3.4 本章小结

本章首先讨论了开发车联网实时流数据监控平台的功能需求，详细阐述了车辆实时信息监控功能，过期数据过滤功能以及指向性的车联网服务功能，简要介绍了如何基于这个实时流数据平台实现这些功能；然后分析了整个系统详细设计方案，包括整个系统的模块化，数据通信和数据处理方式。整体从这两个方面阐述了系统的设计方案。

第四章 实时流数据处理平台的设计与实现

上一章阐述了基于实时流数据处理平台的车联网监控系统的总体设计方案，本章将详细分析流数据处理平台的设计与实现，并给出相应的功能测试和性能测试结果。

4.1 实时流数据处理

实时流数据处理平台是车联网监控系统最重要的部分，是整个系统的中间环节，需要充分考虑系统的实时数据处理，满足高响应特性，可扩展特性以及可维护特性。

我们知道，中心控制单元接收的信息往往规模庞大并且种类繁多，这种具有流式特点的数据一般都未经深度处理就发送到了云端，对此，可以通过我们的实时流数据处理平台来对它们进行实时处理，而主要需要解决的功能有上下文数据的收发、过滤、分类聚合处理以及中间结果捕获。

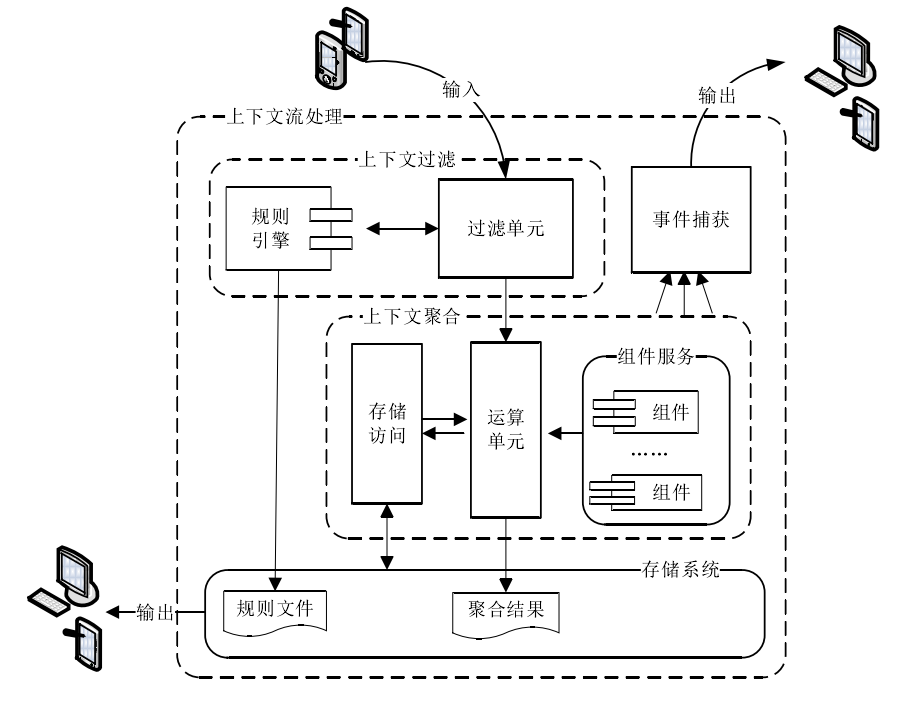


图4-1 实时流处理平台结构图

在整个流处理过程中，对上下文数据的分类聚合处理是整个实时流数据平台的核心功能，负责实时流数据的处理，并且同时与过滤模块、中间结果捕获模块以及存储模块进行交互；上下文收发模块主要负责对到来数据的接收与转发工作；下面就数据流处理的模型一一详述。

4.1.1 实时流数据采集

整个系统作为一个中心处理单元，除了自己内部数据通信以外，还需要对车载单元以及外围路边单元发送过来的数据进行采集，这种数据并不像处理模块和前端数据监控模块那样具有一对一并且稳定的连接，并且车辆产生的数据可以说是无限的。

面对每天可能产生的大量的源源不断的数据，系统将产生大量的数据I/O，而对于接收到的车联网数据，本身并不需要复杂计算，因此对CPU的负荷相对于I/O来说则轻松很多，这是一个典型的CPU使用率较轻、IO使用率较重的应用系统。

试想一下，可以把车载终端每一次数据发送当作一次服务请求，假如我们使用非阻塞的I/O方式，首先创建一个监听进程，每当接收到一次服务请求我们就需要创建一个进程，而当连接断开的时候，我们则需要销毁该进程，退一步来说，我们使用线程池的技术来减少线程创建和销毁所带来的开销，但是这种开销总是无法避免，当然这也并不是使这种开销最小化的方法。

4.1.1.1 基于异步非阻塞I/O通信方式的设计实现

本文讨论系统使用的是一种异步非阻塞I/O的通信方式，使用nodejs这种异步编程语言,可以将线程创建销毁的开销降到很低的程度，同时使系统具有很高的响应度。Node.js非阻塞模式的IO处理给Node.js带来在相对低系统资源耗用下的高性能与出众的负载能力，非常适合用作依赖其它IO资源的中间层服务。

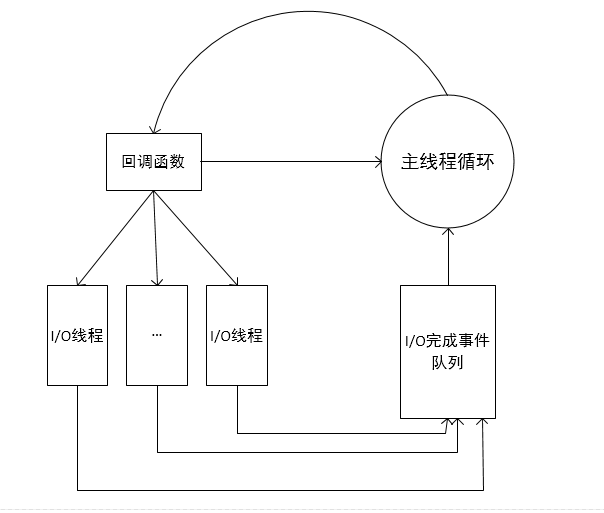


图4-2异步非阻塞通信模式图

关于nodejs，它是一种单线程基于事件循环的异步编程语言，如图4-2所示，主线程也就是nodejs中始终占用处理机的线程会不停的检查事件完成队列，检测是否有完成的I/O操作，如果有则返回该I/O线程的回调部分，恢复现场并运行后续程序。主线程监听连接关闭并将相关的套接字信息输入到日志对象的代码示例如下：

sock.on(‘close’, function(){

…;

io.sockets.emit(‘log’, {sock关闭套接字先关信息});

} )

sock表示连接套接字，sock可以监听许多事件，包括数据I/O完成，错误信息等。当主线程运行到这段回调函数时，如果触发这个close时间的I/O并没有完成，主线程并不会因此而阻塞，此时如果有其他的I/O完成，主线程便去执行其他I/O触发事件的回调函数，若没有其他的I/O完成，那么主线程将会继续空运行，直到有一个I/O线程完成后触发事件为止,这种方式虽然可能会导致主线程空转，但是避免了服务线程创建和销毁的时候的开销，并且在面向这种源源不断的大量的数据的情况下，CPU空转的时间带来的系统损失基本可以忽略不计。

4.1.1.2 node.js多核利用

第三章介绍系统的数据接收模块中充分利用到了硬件的多核特性。虽然node.js语言特性可以很好的处理高并发I/O的应用需求，鉴于现代硬件多核的特性，如果仅仅使用单线程的方式将是对硬件资源的极大浪费，系统在设计数据接收模块过程中调用node.js的cluster模块， 使得硬件环境得到充分利用。

为了满足高并发的应用需求，系统使用node.js的cluster模块，其底层是基于libuv实现的多线程应用模块，这是一种基于事件循环的多线程应用，node.js为我们封装好了关于进程调度以及各进程负载均衡的实现，系统在开发应用过程中需要做的便是使用它，并开发对应的worker进程的数据接收转发的逻辑，代码实现如下：

var cluster = require(‘cluster’);

if(cluster.isMaster){

for (var i = 0; i < require('os').cpus().length; i++) var worker = cluster.fork();

cluster.on(‘exit’, function(worker, code, signal){

console.log('http\_trace worker ' + worker.process.pid + ' exit. restarting');

cluster.fork()

});

…；//其他事件，如listening，death，fork等

}else{ //如果是worker线程则调用具体的数据处理逻辑

console.log('http\_trace worker is running');

require('./http\_trace\_pubredis');

}

在决定线程做什么事情之前，先确定线程的身份，如果当前线程是master线程，那么当前机器的CPU核心数创建工作线程个数，worker线程的个数为（require('os').cpus().length-1）个,同时master线程也会监听worker线程的退出和死亡，如果有意外退出或者死亡的线程，cluster将会重新fork一个工作线程；如果当前线程是工作线程，那么我们将运行require内部的模块的内容，也就是负责数据对接以及转发给处理模块的具体逻辑。

4.1.1.3 数据接收具体逻辑

前面提到过数据接收模块和数据处理模块是通过redis的pub/sub机制实现，这是一种基于内存为缓冲的数据传输方式，这样有效的避免了因为数据量国语庞大可能造成的系统延迟甚至瘫痪，下面撇开这种通信方式的优点，详细介绍下数据接收部分的具体逻辑实现。我们使用express的nodejs框架搭建web服务器（忽略express的静态资源以及路由配置部分，这是项目本来打算开发一个浏览器页面来观察接收数据的，后来没有开发）。

第一步，我们初始化redis服务器和网络服务事例server，初始化如下所示：

var redis = new RedisStream(REDIS\_HOST, REDIS\_PORT);

var server = net.createServer();

其中，RedisStream是通过

var RedisStream = require('./lib/framework/redisstream');

语句获取的模块，需要提供主机名HOST和端口号PORT来初始化，具体RedisStream模块如何实现是基于redis本省提供的方法进行一层适用于该系统的封装，主要是对redis数据库本身方法以及部分自定义方法的扩充。关于server的创建，是通过nodejs内置的net模块创建的，该模块与nodejs的http模块类似，主要就是创建一个服务端服务，并提供相应的方法。

第二步，服务线程初始化完成之后，服务线程便开始监听网络中的连接，监听的方式是，当有一个连接请求过来时，将会触发一个名为connection的事件，当事件触发服务线程将生成连接套接字sock为到来的连接服务，代码事例如下所示：

Server.on(“connection”, function(sock){

… ；//处理过程

});

同理，套接字对象sock会监听data事件并且采取对应的逻辑处理过程，

sock.on(‘data’, function(data){

var result = cmdUtil.unpack(data.toString().trim());

if(result){

…; //sock类型以及数据命令字检查

redis.publish('OBDSERVER\_PUB',JSON.stringify(result));

io.sockets.emit('log',{'sn':result['SN'],'msg':result});

}else{

…; //往日志对象发送unknown message信息

}

});

cmdUtil是自己定义的模块，功能在于按照自己的规则对data进行kev-value的方式重新对应，并且对返回的result进行检查，如果result不为空则进行括号内部分的代码逻辑，否则向日志发送错误信息。括号内部逻辑主要表现两个方面，一个字段检查和一些非数据相关的封装赋值，在判断数据没有问题之后会通过redis对象将数据发布到OBDSERVER\_PUB的通道中供数据处理模块来接收。

数据被发布到redis通道之后，我们的数据处理模块便可以通过订阅这个名为OBDSERVER\_PUB的通道来获取所发送的数据了。这部分就是简单的订阅通道信息，但考虑数据发过来基本信息字段不是同一个终端，所以数据处理模块在接收数据之前需要自己先维护一个client对象，用来保存当前正在通信的终端的信息。这部分功能由RedisStream对象提供initSubChannel方法，应用程序在发开过程中如果需要想要订阅某个通道的信息，可以通过initSubChannel(subChannel, p)方法获得，其中p是系统逻辑核心文件vrsscore.js定义的一个对象，对象结构初始化定义如下：

var Process = function(){

this.env={};

this.env[‘tuple’]={};

this.env[‘stream’] = {};

this.env[‘status’] = {};

this.env[‘result’] = {};

this.env[‘dispather’] = new VRSSEventHandler();

}

initSubChannel初始化的主要功能是维护一个redis的订阅客户端，并且根据给定channel名将具体process对象绑定到这个redis订阅客户端的指定通道上，这样process对便可以订阅到redis通道里面的数据并且根据通过vrsscore模块文件里面定义的其他方法来处理这些数据。主要代码逻辑如下：

RedisStream.prototype.initSubChannel = function(channel,p){

var fn = buildRS (p);//往p中注册抽象的方法，具体实现在core实现

if(!this.client[channel]){ //如果当前为首次初始化

var client = redis.createClient (this.port,this.host);

…; //相关认证信息

this.client[channel] = {}; //

this.client[channel].client = client;

this.client[channel].client.subscribe(channel);

this.client[channel].process = [];//定义一个数组保存p中注册绑定的函数方法

this.client[channel].channel = channel;

if(fn){

this.client[channel].process.push(fn);

}else{绑定一个输出debug信息的函数}

var that =this;

that.client[channel].client.on('message', function(channel,msg){

…； //相关健壮性判断

for(var i=0;i<that.client[channel].process.length;i++){

that.client[channel].process[i](

{"source":channel,"msg":msg}

);

}

}

}else{

…; //否则使用当前的订阅客户端

}

return this;

}

我们看到var fn = buildRS (p)方法，这个方法的返回值是一个函数，根据一些初始化和条件判断过后会将它push到一个this.client[channel].process数组中，最后在接受到数据是会依次调用数组里面的函数，buildRS()代码如下：

function buildRS(p){

return funcition(data){

p.env[‘dispather’].write(‘data’,data);

}

}

p.env[‘dispather’]是在vrsscore模块中定义继承自node.js的events模块的对象，这个对象拥有node.js管理事件的功能，可以在其中注册事件上述代码中的write方法在vrsscore的具体实现就是注册一个data事件并且发送相关的数据。

4.1.2 数据过滤

面对大量的种类繁多的流数据，为了减少数据处理核心部分的压力，使系统拥有更快的处理速度，对原始数据流的过滤十分必要。系统基于一定的过滤规则，对接收的流数据进行过滤，如图4-3所示：

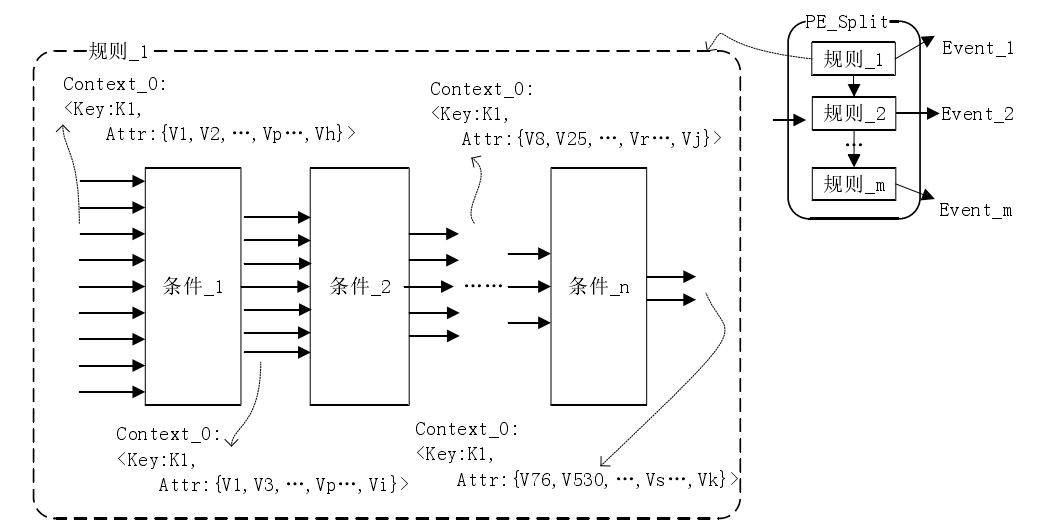


图4-3数据过滤规则示意图

PE\_Spilt是系统定义的一个逻辑处理单元，主要负责对接入流数据的过滤工作，其中定义许多过滤规则，流数据会逐一进行规则匹配，不满足规则的数据将被丢弃。

数据过滤规则模块主要有三个部分组成，分别是规则的操作接口，规则执行引擎和调用引擎的规则逻辑控制单元。如图4-4所示，开发人员对规则机制的操作，首先调用封装好的接口方法操作，然后具体的实现类会根据传入的参数进行相应的方法实现操作，操作过程中会调用规则引擎中的数据结构或者参数对象等，最后会对获取根据相应规则所产生的结果，如我要添加一个新的规则，会在规则队列里面添加一条新的规则。

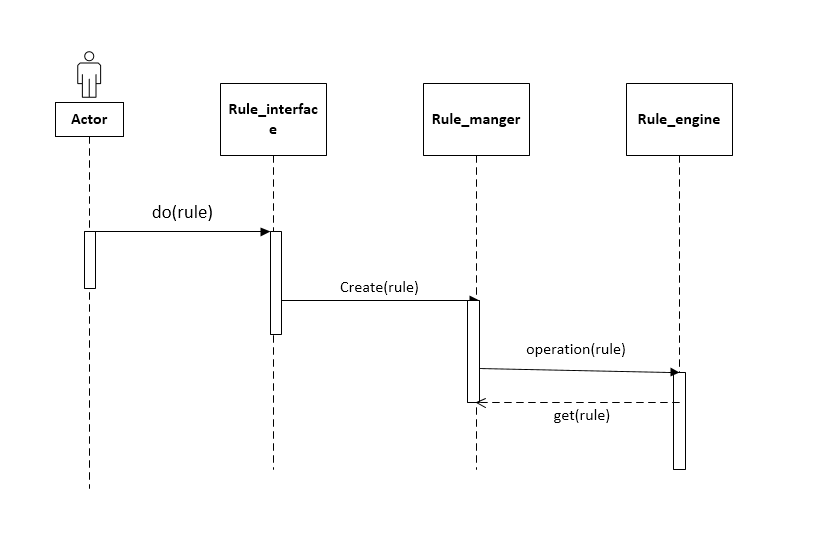


图4-4规则机制操作时序图

对于本文所述的实时流处理平台也根据需求设定了一部分的过滤规则，并且如之前所述，平台支持规则的修改和扩展。下面介绍下系统所基于的基本过滤规则。

1. 车辆状态信息异常

这部分异常主要原因可能在于车载终端发送数据时候出现问题，误将空数据或者异常数据发送到终端，比如车辆刚开始启动时传感器各项数据都处于初始化状态，这时候车载终端获取到数据便是空的数据，数据示例如下所示：

2015-01-15 17:35:22,$OBD-RT,0.0,0,0,0.00,0.00,0,0.00,0.00,0,0,0.00,0.00,

0,0,0,0,0,0,0,$OBD-HBT,0,0.00,0.00,0,0,0,0,0,0,0,0,0,0

此外，如果获取数据中部分稳定数据跳动过大，或者出现异常，如行驶距离字段，一旦发现行驶距离字段出现减少情况，我们则先通过检查数据时间戳，判断是否是因为网络延迟发送的过期数据，如果是则对本次收到的数据进行排序后再发送，否则便丢弃该数据。

1. GPRS位置信息跳跃

这部分异常主要原因可能是GPRS定位信息有误或者车辆行驶到一段没有信号的道路导致信息缺失，面对前者，我们需要对其数据进行丢弃，后者则保留。判断这种异常的区别也在于地理位置的跨度，给予一定的阀值，跨度一旦超过阀值便判定为前者，予以丢弃，若在可控制阀值以内，则予以保留。那么这些保留的数据可能会造成车辆节点在地图的“漂移“现象，面对这种无法避免的漂移现象，后面系统作了两项处理：将定点显示改为定时显示；去除单点跳跃数据记录。

4.1.3 数据分类处理

数据处理这部分过程是整个系统的核心逻辑处理过程，实现这个功能的关键在于基于事件编程的node.js语言和核心文件对接收数据的处理策略。Node.js这个编程语言在第二章软件技术中已有介绍，下面我们介绍下流数据处理策略及实现。

4.1.3.1 基于事件捕获的数据转发机制

首先，我们对需要实时展示的数据以发送事件的方式发送到前端，实现方式图如图4-5所示：

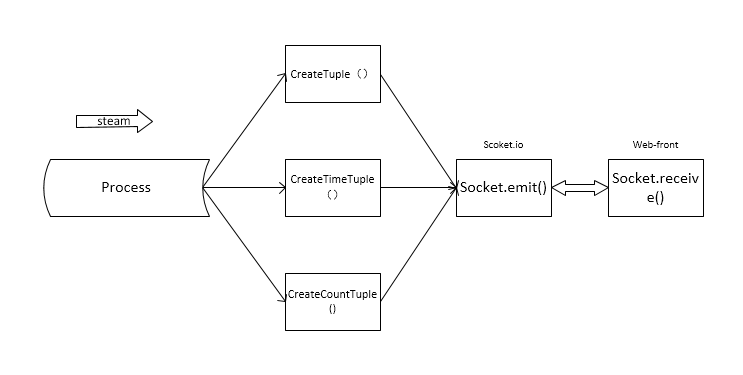


图4-5数据发送到web-front

核心逻辑处理文件vrsscore.js自己定义维护了一个json对象process，后面以p来表示。上文提到过在服务初始化的时候我们曾经使用到过这个p，当时是以参数的形式传入到InitSub()的方法中，这个初始化方法将数据流对象封装到p的这个对象中，并且以value的形式暂存，在p对象中的key值为data。

让我们先看下这个process对象，参见第一节p对象初始化的截图，任意对象初始化p这个对象的时候会有一个dispatcher属性，详细的初始化语句是this.env['dispather'] = new VRSSEventHandler(); 这个对象是node.js的EventEmitter模块的一个实现类对象，继承自node.js的events模块，它的对象具有addListener，on，once，removeListener，removeAllListeners，emit等基本的事件监听模式的方法实现，拥有注册或者监听事件等功能。

实时流数据处理核心逻辑文件在初始化process对象时同时还给process对象定义了一些方法属性，撇去格式转换以及程序启动关闭的基本方法不谈如图4-11所示主要定义了三种数据处理转发的方式，这便是基于事件捕获然后转发到前端数据监控页面的核心逻辑。在介绍图中三种方法之前，我们还要介绍process对象的一个属性，this.env['tuple']={};tuple是process对象中用来暂存处理流对象的关键字，在后面所述的三种方法中，会将监听data事件而读取的流数据按需求暂存到这个对象属性中，data是process对象dispather对象属性中注册的事件，我们可以抽象的把他理解为一个通道，这是一个本地维护的redis客户端，可以缓冲订阅得到的数据，每次需要数据时候我们可以通过监听data事件从这个通道中订阅数据，而我们定义的三种方法便是从redis管道中读取并将数据暂存到这个对象中的策略。

1. 全部读取

每次通过本地redis客户端订阅得到的数据我们会将它封装到process的tuple属性中 ，全部读取则是程序会不停的从通道里获取数据，每次讲当前process订阅到的所有数据都取出来，每当一次I/O完成，就立马去读取当前通道的数据并转发。代码实现如下：

Process.prototype.createTuple = function(id, fn){

var that = this; var \_fn = check(fn);

if(!this.env[‘tuple’][id]){

this.env['tuple'][id]={};

this.env['tuple'][id]={type:"NORMAL", key:null, tmp:[],

func: function(data){

\_fn.apply(that,[data]);}

};

}

return this;

}

1. 按时间戳读取

按指定的时间间隔来接收转发一次tuple中暂存的数据，每隔一段时间会将通道中订阅到的数据进行一次处理，并转发相应的数据。

Process.prototype.createTimeTuple = function(id, fn){

…//同上

if(!this.env[‘tuple’][id]){

this.env['tuple'][id]={};

this.env['tuple'][id]={

type:"TIME",

…

func: function(){

return setInterval(function(){

if(that.env['tuple'][id]){

if(that.env['tuple'][id].tmp.length>0){

var tuple=that.env[‘tuple’][id].

tmp.splice(0,that.env[‘tuple’][id].tmp.length);

\_fn.apply(that,[tuple]);

}

}

},time);

}

};

}

return this;}

1. 按数据量读取

disther属性中额外注册一个事件，一旦tuple中已经暂存了达到阀值量数据，就会发送一个表示到达阀值的事件，处理逻辑部分接收到这样的事件便调用相应的数据处理程序。

4.1.3.2 基于数据特征的分类存储

在许多应用数据中，有一些数据是暂时的，转发后便可以直接丢弃。但是，从运算的角度来看，有一些数据是不能丢弃的，我们需要将这些数据进行落地存储，为那些需要基于历史数据的运算提供必要的数据。

系统基于storm流计算模型设计数据分类存储策略，Spout表示数据源，Bolt表示数据处理单元。首先，前面已经说到，在获取数据源之后我们需要经过第一阶段的过滤阶段，这一阶段除了将数据过滤后做简单处理并转发到前端予以显示以外，我们还需要将这些数据进行分类存储到数据库；其次，我们对存储进数据库的流数据也需要进行分类处理的，根据一定的特征提取进行分类，否则，将影响到后面数据索引的效率；最后，我们将经过特征提取后分类的数据存入数据库，如图4-6所示，我们以汽车行驶速度特征为例进行特征提取并分类。

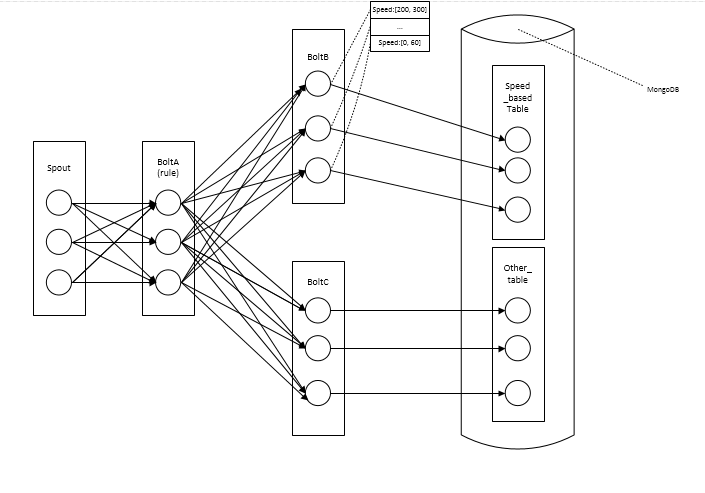


图4-6基于车速区间特征的数据分类存储图

Spout所有的车载终端都可以作为模型中的数据源，我们通过一定的过滤规则将异常数据或者不需要的数据进行过滤，图4-6中BoltA便充当过滤单元模块的角色，此时我们从BoltA中输出数据，输入BoltB和BoltC消息处理单元，其中我们假设BoltC属于其他消息处理单元，如上文中提到的消息转发单元或者按区域划分等消息处理单元，BoltB消息处理单元便是我们的以平均速度为特征的数据分类处理策略，我们将平均速度按照不同的区间分类，统计满足速度区间的车辆，以key-value（速度区间为key，车辆id数组为value）的形式最终存储的数据库中。按照平均速度分类的功能伪代码如下：

引用模块导入及对象初始化；

function classifyDataByAvgspd(sn, avgspd){

mongohelper.find('iov', 'vehicleinfo', {"sn": sn},function(data){

//helper创建连接

var current-avgspd = data.other.avgspd;

if(avgspd not in current-avgspd){

…; //更新车辆信息数据集合

var currentstr = arrtostr(current-avgspd); //转换成字符串作为键值

mongohelper.find('iov', 'vehicleinfo', {"spdregion": currentstr},function(data){

var newdata = data.remove(sn);

update … mongodb;

})

get … newspdregion;

var newstr = arrtostr(new-spdregion);

mongohelper.find('iov', 'vehicleinfo', {"spdregion": newstr},function(data){

var newdata = data.remove(sn);

update … mongodb;

})

}

});

}

其他函数及模块导出；

4.2 数据库设计与实现

传统的关系型数据库虽然能满足大规模数据存储的需求，但是它在面向密集型I/O系统应用的时候略显乏力，MongoDB使用文档结构存储数据，虽然不能像传统数据库那样能够明确的表示表之间的关联关系，但在很多场景下可以替代关系型数据来表示数据之间的关系，并且MongoDB在处理高并发I/O的情况拥有自己很好的处理机制，可以有效处理系统开发过程中对高效存取的需求。

系统开发过程将数据库命名为iov，使用MongoDB进行数据存储。系统在数据存储过程中将数据优先根据特征分类，在存储到数据库中，因此在MongoDB中维护了多个表，以文档的形式存放在数据库中。

系统在过滤异常数据后，我们首先需要将数据存到一个总表中，这个表我们称之为统计数据的表格，里面保存有我们接收到的所有需要用到的车辆相关信息，按照MongoDB文档形式的存储结构以及bson数据存储格式的要求，我们需要将我们提取的数据以key-value的形式组装一下，我们对获取的value给予一定的key相对应。表4-1是获取的一条原始数据

1421313948103|119.4.252.45|$EST627,6214005119007,01,303,13,2015-01-1517:3 2:11,$OBD-RT,14.4,1458,32,18.04,22.75,85,8.77,13.30,0.52,4,0.09,1.68,0,0,0,0,0,0, 0,$OBD-HBT,4,0.28,0.20,10,16,47,15359,0,0,0,0,0,0,$GPS,(1,104.048927,30.64761 0),(1,104.048851,30.647579),(1,104.048767,30.647539),(1,104.048683,30.647499), (1,104.048599,30.647461),(1,104.048523,30.647430)

进行处理后以表格的形式显示存储结构：

表4-2统计数据表格

|  |  |  |  |
| --- | --- | --- | --- |
| 键 | 值 | | 备注 |
| \_id | ObjectId("56d2962fddda1f20567b110c") | | MongoDB基于数据存储策略生成的唯一ID |
| SN | 6214005119007 | | 代表单个车辆的唯一ID |
| Detail | Base | 基础信息，消息时间，命令字等 | 封装车辆具体信息的字段属性 |
| OBD-RT | 车辆实时数据，如当前速度，发动机转速等 |
| OBD-HBT | 车辆统计数据，如总行程，最高速度等 |
| GPRS | 车辆定位信息，包括GPRS定位和基站定位 |

具体在数据库存储方式如图4-7所示，Detail部分主要包括基础信息、实时数据、统计数据以及GPRS数据，基础信息主要包括车辆id、命令字以及日期时间等信息；实时数据主要包括当前时速、当前发动机转速以及当前耗油量等；统计数据主要包括总行程、最高时速以及急转弯次数等信息；GPRS数据存储的结果主要是通过一定校准将基站定位和GPRS定位信息结合计算得出的位置信息。

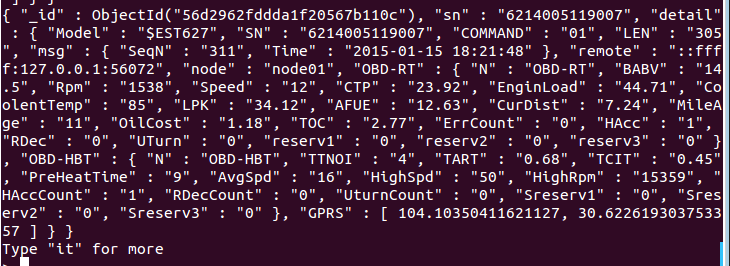


图4-7单条统计数据MongoDB存储示意图

除了统计集合文档，系统还定义了车辆相关信息以及驾驶员相关信息的集合，详见表4-2，表4-3所示。

表4-3车辆信息集合单条记录

|  |  |  |  |
| --- | --- | --- | --- |
| Key | 字段类型 | Value | 备注 |
| SN | Integer | 6214005119007 | 车辆id |
| Level | Integer | 2 | 车辆等级 |
| CarNum | String | … | 车牌号 |
| OBDNum | Integer | 19007 | 车载终端编号 |
| Type | String | 小型汽车 | 车辆类型 |
| Status | String | 良好 | 车辆状态形容 |
| other | Object | {avgspd:[0,60],…} | 车辆其他行驶信息，如当前所在速度区间等 |

表4-4驾驶员信息集合单条记录

|  |  |  |  |
| --- | --- | --- | --- |
| Key | 字段类型 | Value | 备注 |
| driverNum | Integer | 01 | 驾驶员编号 |
| Base | Object | {“name”: “测试人员”， “phone”: “182\*\*\*\*\*\*\*\*”,… } | 驾驶员基本信息，以key-value方式存储 |
| Licence | String | … | 驾照编号 |
| Type | String | 小型汽车 | 驾驶车辆类型 |
| e-Record | Array | [object，object，…] | 违章记录 |
| Status | Array | [良好，一般，…] | 当月驾驶状态记录 |

系统在设计数据库过程中为了方便其他数据的分析或者其他操作，还设计了其他细化的集合类型，如表格4-4所示便是基于速度区间的文档存储结构。除了以速度为特征以外，系统还以每月总路程为特征细化的文档集合，见表4-5。

表4-5基于速度区间的分类存储文档结构

|  |  |  |  |
| --- | --- | --- | --- |
| Key | 字段类型 | Value | 备注 |
| spdregion | Array | [0, 60] | 速度区间 |
| sns | Array | [sn1, sn2,…] | 车辆id数组 |

表4-6基于每月车程的分类存储文档结构

|  |  |  |  |
| --- | --- | --- | --- |
| Key | 字段类型 | Value | 备注 |
| miles | Array | [600, 1000] | 里程区间 |
| sns | Array | [sn1, sn2,…] | 车辆id数组 |

这些细化的文档集合对我们的系统具有各自的作用，例如，我们在需要这些数据做对比的时候，不需要特地的检索统计数据库然后分类统计，这样会给系统的运行效率造成极大的负面影响，然后我们还可以根据这种分类区间数据分析用户的驾驶习惯，分析用户类型等。

4.3面向服务模块

系统在应用开发的过程中，除了关注与监控平台的交互以及数据的落地存储，还做了一些基于实时流数据平台的应用服务扩展，在满足监控需求的前提下可以实时的提供部分相关信息服务。系统设计初期主要做了以下几部分的服务应用，简单看下系统对这几个需求的应用设计，见表4-6：

表4-7实时流处理平台面向服务扩展设计

|  |  |
| --- | --- |
| 服务模块 | 备注 |
| require(“Status-based-service”).getCurrentStatus(data); | 车辆状态实时监控，会根据收到的车辆状态码数组，将向数据源端注册的终端设备发送提醒信息。 |
| require(“GPRS-based-service”).getCurrentLocation(data); | 车辆位置信息实时监控，对进入指定范围内推送相应的服务信息。 |
| require(“other”); | 其他潜在的需求 |

这些具体的服务模块基于一个总的服务模块Services-app，其功能是通过判断用户接入信息的类型，调用具体的服务模块以及具体的方法以反馈用户。这种即时通信的方式需要基于一个有效的通信平台，服务模块功能实现的核心代码如下：

var status/gprs/path = require(“具体服务模块路径”);

…; //服务依赖模块以及相关初始化

function service(sn, data){

var weixin = mongohelper.getWeiXinBySn(sn); //调用数据库获取注册微信号

var type = data.type; //判断数据类型

if…else if…else{ //根据type的值调用具体的服务模块,status为例

var status-data = status.getCurrentStatus(data.msg);

var push-url = "http://www.centibox.io/weixin/push?token=

voyager\_abcd1234&p="+weixin+"&msg="+status-data;

require('nodegrass').get(push-url, function(data){

logger.info(data );

}, null, ‘utf-8’).on(‘error’, function(e){

logger.info(e); //logger是nodejs管理日志模块log4js

})

}

}

…; //其他方法以及export导出模块等其他内容

上述代码中，我们会调用具体的服务子模块来获得返回的数据，不同的模块返回的数据不同，最后通过nodegrass模块将数据推送到微信服务器上,订阅微信公众号的便有具体的信息提醒。具体的子模块结构图如图4-8所示：



图4-8 实时流处理平台面向服务扩展图

1. 车辆状态实时提醒

车载终端OBD会不停的向服务器发送车辆状态信息，服务器会实时地接收车辆状态信息，接收端会将数据转发到实时流数据平台。系统在设计实时流数据平台过程中维护了一个总服务应用模块，它像一个有自己功能的接口，可以随时调用子服务模块，根据获取的数据类型，切分相应的数据，并将相应的数据传入具体的子服务函数以获取返回的状态码数组信息。其主要方法getCurrentStatus()实现如下：

var standard = require(‘config路径’).statusStandard；

…;//初始化及判断语句

function getCurrentStatus(data){

… //data数据检查

var status = []; //定义一个数组来存储所有的状态码，0为正常，1为异常；

jsontoarr(data); //自定义方法，将data按照一定字段顺序转换成数组

for(int i=0; i<data.length; i++){

if(data[i].compare(standard[i])){ //自定义方法，比较阀值

status.push(0);

}else{

Status.push(1);

}

}

return status;

}

…；//其他方法及模块导出

1. 实时信息服务推送

这类服务主要是基于车辆位置信息来推送服务的，系统会在地图上规定每个坐标范围内会主动向范围的车辆推送先关信息服务，比如当有车辆在东郊记忆附近一公里以内时，我们可以向车辆注册信息终端推送东郊记忆的相关旅游信息。地图中选取的范围可大可小，也可以限制对同一个用户的推送次数。我们以旅游为例，这里我们使用到了百度的地图api来获取附近的景点的地址以及距离信息，实现核心代码如下：

…; //模块导入及相关初始化

//旅游

function getCurrentLocation-travel(gprs){

var travel-near = [];

var baidulbsurl="http://api.map.baidu.com/telematics/v3/local?location=

"+gps[0].toFixed(6)+','+gps[1].toFixed(6)+"&keyWord=景点&

output=json&ak=23d9ab5254157436939cb75c129483d0";//百度api-key

require(‘nodegrass’).get(baidulbsurl, function(data,status,header){

var loc-json=JSON.parse(data);

if(数据有效){

for(遍历loc-json.pointList){

var msg =loc-json.pointList[i].name+','+loc-json.pointList[i].address;

if(loc-json.pointList.distance<阀值){

travel-near.push(msg);

}

}

}

}, null, ‘utf-8’).on(‘error’, function(e){

…;

})

}

…; //其他方法及模块导出

4.4 本章小结

本章主要介绍了系统核心部分实时流数据处理平台的设计与实现，详细介绍了实时流处理平台各部分模块的功能的设计和实现，分别是高并发数据I/O应用场景数据的接收和基于数据特征的分类处理；后面我们就数据如何存储详细阐述了数据库的设计以及系统开发过程中相关的数据库访问优化策略；最后我们基于现有的实时流数据处理平台详细介绍了系统对于车辆网服务方向的扩展。

第五章 实时监控平台的设计与实现

第四章主要介绍了实时流数据处理平台的详细设计与实现，本章将对面向管理端的实时监控平台功能的设计与实现作出详细的介绍，并给出最后的功能测试结果。

5.1 实时监控平台概述

实时监控平台主要是面向管理端开发的，用来实时观测车辆实时状态以及道路拥堵情况，当然在面向服务开发的时候也有部分功能面向车载信息终端开发的，如用户可以通过车载信息终端获取当前道路是否有拥堵发生等。平台基于Flask的网络框架设计，使用pymongo的数据库引擎来连接mongoDB，显示界面使用html5和JavaScript配合的方式实现界面的实时交互，并通过调用高德地图API来加载地图，使用highcharts和dataTable来实现统计图表的显示，并且使用ajax的load()方法实现部分页面局部加载。



图5-1实时监控平台结构图

如图5-1所示，pymongo是用来连接mongoDB的数据库引擎，从mongoDB中获取数据属于拉取的方式，主要是通过ajax技术来实现交互的，用来统计类图表显示；相对的，当我们需要表示车辆在地图显示的实时数据，则需要与数据处理平台实时交互，这是一种实时push的方式，通过Socket.io来实现；当我们需要将统计类图表以前显示在地图上时，就需要ajax局部加载的技术，通过这种方式，我们可以将图表嵌入到地图页面一起返回给B端；此外，Flask框架提供了模板渲染以及静态资源加载的功能，地图的加载则是通过调用高德API的方式加载的。

5.2 实时监控平台设计与实现

实时监控平台采用了经典的B/S架构，利用Flask的网络框架以及各种依赖包实现与各部分模块之间的交互。由于基于B/S架构，因此我们在系统设计过程中除了有基本需求的监控功能，车载信息终端如果B类端的话同样能接收相关的数据在地图中实现，本文测试环境有限，会用本地浏览器来模拟基于其他信息终端的服务扩展。

5.2.1 地图加载

系统使用的地图是高德地图，因此需要调用高德API的地图接口，在此之前，我们需要先获取高德地图的开发者资格，然后才能进行后续的操作。

1. 申请JSAPI的开发者key

为了获得开发者资格，我们需要登录<http://lbs.amap.com/console/key>以申请高德地图开发者的APIKey，系统在开发过程中申请的key值为e0a3f49e0891b45d692cc1b800022343，后面我们可以通过这个key值来加载高德地图文件。

1. 引入高德地图JavaScript Api文件

引用的格式：

<script type="text/javascript" src="http://webapi.amap.com/maps?v=1.3&key=

e0a3f49e0891b45d692cc1b800022343"></script>

其中，src属性是获取地图文件的详细地址，里面传入了两个参数v和key，分别表示地图的版本和之前所申请的开发者秘钥key。

1. 在html中创建地图容器

Html中支持地图元素的标签<body>，在<body>标签中定义一个div层来加载我们的地图，并赋予一个id属性,如下所示：

<body>…

<div id="cont"></div>

…</body>

1. 规定容器基本参数

容器初始化的参数，一般都是指容器的宽度和高度，原本可以在上面div层标签里面添加高度和宽度属性或者通过css来定义，这样设置的话确实可能满足本地浏览器的需求，但是如果别的和本地不同屏幕比例的浏览器来访问的话，会导致很差的用户体验，因此系统将基本参数的初始化过程放到了JavaScript文件里面执行，通过当前文档对象的宽窄度来定义。如下所示：

<script>…

height=window.innerHeight||document.documentElement.clientHeight;

width=window.innerWidth||document.documentElement.clientWidth;

$("#cont").height(height);

$("#cont").width(width);

…</script>

这段程序会获取当前窗口的高度和宽度，并通过jQuery选择器来设置div层的高度和宽度，之所以用“||”符号是因为IE无法识别innerHeight和innerWidth对象，这种方式可以兼容IE浏览器。

1. 创建地图

一般情况下我们会在JavaScript代码中创建地图对象，初始化的时候会设置中心点和区域范围，系统代码如下所示：

<script>

mapObj = new AMap.Map("cont", {

view: new AMap.View2D({

center: position,

zoom: 14,

rotation: 0

}),

layers: [new AMap.TileLayer.Satellite({

zIndex: 5

})],

lang: "zh\_cn"

});

</script>

除此之外，高德API还提供了普通地图和卫星地图之间的切换，在代码里面所需要做的就是添加两个控件，分别是AMap.MapType和AMap.ToolBar，调用高德API的addControl()方法可以实现地图的切换, 使用时可以点击设置一个dom事件来触发addControl()功能。设置插件代码如下所示：

<script>

mapObj.plugin(["AMap.MapType"], function () {

//地图类型切换

var mapType = new AMap.MapType({

defaultType: 0,

showRoad: true //叠加路网图层

});

mapObj.addControl(mapType);

});

mapObj.plugin(["AMap.ToolBar"], function () {

toolBar = new AMap.ToolBar();

mapObj.addControl(toolBar);

toolBar.show();

});

</script>

地图加载图效果如图5-2所示。



图5-2地图加载图

5.2.2 数据交互

数据交互的方式主要有两种：一种是与mongoDB的交互，这一类数据通常是静态的，使用pull的方式来获取，监控平台担当主动端角色；另一种是与数据处理模块的交互，这一类数据一般是动态的实时的，使用push的方式获取，监控平台属于被动端。

5.2.2.1 基于ajax的静态数据交互

Ajax作为一个动态交互式前端开发技术，在网页开发中已经得到了广泛的应用，它终结了只有重新刷新网页才能更新网页内容的时代，使得人机交互变得更加方便。Ajax是基于JavaScript异步编程语言的特性，通过事件机制实现与后台服务器之间的数据交互，从而实现了网页数据的局部更新。

值得一说的是，实时监控平台本身也是一个基于B/S架构的系统，拥有自己的服务器端和浏览器端，而我们使用ajax也正是与实时监控平台的服务器端进行数据通信，通信方式如图5-3所示：



图5-3 Ajax原理图

Ajax主要负责的任务便是brower端和server端之间的数据交互，如图所示，用户通过界面操作会调用相应JavaScript的ajax操作，此时ajax会根据获取的url参数定位当前服务器地址，并发送http请求向服务器请求数据，此时服务器会调用mongoDB的数据经服务器返回到ajax的回调函数中，最后再基于这些数据通过JavaScript相应的操作来更新当前页面。

我们首先定义数据监控平台的application来实现一个调用mongoDB的python文件，引用代码如下：

import \*

from flask import Flask, request, jsonify, render\_template,make\_response, Module，pymongo

database-module = Module(\_\_name\_\_)

flow.url\_prefix = '/data '

@database.route('/')

def database\_index():

session = request.environ['beaker.session']

… //处数据逻辑

return data

对应的，我们通过前端ajax请求这个路由，代码如下：

$.get('\*/data ', function(data){ //具体的请求路径

if(data.isavailble()){

… //调用highcharts或者dataTable生成具体的图表

}

})

上述brower端和server端都是基于flask框架搭建的，其路由机制以及连接MongoDB的引擎都是依赖于Werkzeug库。那么，既然ajax机制可以实现前后台的数据交互，那么我们能不能使用ajax与实时流处理平台进行数据的交互呢？答案是否定的，主要原因有二：

1. 实时流处理平台转发的数据是以一种流的形式推送的，仅仅是暂存在通道中，如果用ajax的方式去读取，必然会导致数据不连续。
2. Ajax的数据请求方式是一种pull的请求方式，而且一般用来用户与界面的交互，用来接收这种实时数据流的应用很难适应。

因此，鉴于ajax无法满足动态的实时流数据处理，系统使用socket.io模块的通信机制，来处理这种push过来的实时数据流。

5.2.2.2 基于socket.io的实时数据交互

车辆的实时动向以及实时状态是监控平台的基本功能，如何显示这种实时的行为，先存入数据库再从数据库读取显然是很难做到的，原因在于如果通过数据存取操作来实现，那你首先必须有非常优化的数据库存储以及索引策略，并且系统要受得了系统频繁调用数据带来的开销，显然这样做是不合理的。因此，系统需要一种实时通信的机制，socket.io便是基于监听事件的异步通信机制，接收方会一直监听着接收数据的事件，而数据转发端一有数据就会接收方发，接收方接收到数据后会调用相应的回调函数进行处理，如图5-4所示。



图5-4 socket.io通信示意图

这种通信方式同redis的pub/sub通信机制很像，不同的是它不像pub/sub那样拥有一个中间的redis内存服务器来实现通信，因此，这种通信方式是点对点的，流数据转发端和流数据接收端之间需要建立一个稳定的连接，然后通过socket.io维护的事件机制，流数据转发端注册一个事件名，然后发送相应的流数据，接收端采用异步的基于事件循环的方式始终监听着一个事件队列，接收端可以通过监听这个注册事件来获取相应的流数据。简单示意代码如下：

转发端：pushServer.sockets.emit('push-event',data);

接收端：sockets.on('push-event' ,function(data){});

5.2.3 主要功能实现

实时监控平台的主要功能包括：车辆状态实时监控，车辆位置信息实时更新，历史轨迹查询回放，车辆统计数据显示，道路轨迹规划，网络消息推送等。



图5-5实时监控平台流程结构图

1. 车辆状态实时监控

车辆在行驶过程中状态在不停的变化，包括油量、车速以及转速等，实时监控平台基于实时流处理平台发送过来的数据实时监控着车辆的各项数据，这种实时的数据将会以表单的形式显示出来，为了更好的体验，正常情况这个表单是隐藏的，当鼠标事件移动到车辆图标上时，会自动显示这一个动态变化的表单。

由于这是一种动态的实时的数据通信方式，因此系统使用socket.io的通信方式来实现这种流数据的通信方式，接收端socket.io的js代码会控制监听之前注册的事件名，循环读取当前的事件队列，当读取到socket.io的通信事件就绪后，会调用相应的回调函数，回调函数里面实现具体的数据处理方式。如图5-5所示，系统调用一个前端比较通用的js框架库jQuery来实现表单的填充，回调函数获取数据后，会调用jQuery的选择器来选择需要更新的dom元素的位置并且将分类好的数据更新至相应的位置。同时，jQuery还控制表单的隐藏和显示，jQuery内部维一系列的事件机制，比如对外设鼠标的行为判断，当鼠标移动到地图上车辆图标的时候，jQuery会控制表单进行显示，而当鼠标离开图标时，表单就会隐藏。这是一种良好的交互方式，有效了的预防了地图上车辆较多情况下导致form窗口互相遮蔽的情况。



图5-6车辆实时状态监控功能描述图

图5-6显示了车辆实时监控的功能图，车辆实时状态监控的实时效果图如图5-7所示。消息框需要鼠标移动到上面才会显示，实验中消息框只显示了部分数据，具体实行则根据需求进行显示。



图5-7车辆状态实时监控效果图

1. 车辆位置信息实时更新

车辆实时位置信息显示，是为了能让系统能够监控车辆的实时位置，从而根据当前车辆位置提供相应的服务或提醒。那么系统如何定位车辆当前数据，并且让它在地图上显示，流程和图5-5所示是相同的，不同的是车辆的移动的是通过一个汽车图标来始终显示的，不会隐藏，只会根据获取的GPRS信息进行位置移动。

然而车辆图标的移动并不会简单按我们想象的那样非常的平滑并且按照地图规划的路线移动，系统在开发初期在车辆移动方面主要遇到两个问题：一个是车辆移动轨迹并不会按照地图上的道路轨迹行驶；另一个是车辆行走过程中出现位置跳跃现象，并不是一个连续的行驶过程。

究其原因有两个：第一，车辆之所以不按地图的道路轨迹行驶是因为GPRS所检测到返回的坐标是WGS-84原始坐标体系，这是一种国际标准的GPS坐标，而国内地图商使用的是经过特殊封装的国测局02年发布的坐标体系 GCJ-02坐标体系，因此，车辆图标在地图上有位置偏移；第二，车辆位置跳跃现象是因为车辆行驶过程可能出现信号不畅，导致短时间内GPRS无法定位，会出现异常数据，异常数据会在过滤模块中被过滤掉，这中间数据便产生了间断，于是便导致位置跳跃的现象。



图5-8前端处理GPRS流数据处理图

如图5-8所示，系统对于上述两个问题节解决方案：对于坐标偏移问题，使用特定的地图坐标转换算法来纠正车辆位置的偏移量，系统将这个偏移算法封装成了一个JavaScript文件，通过<script>标签的方式导入，在保持html文件简洁的情况下保证坐标转换的功能；关于图标位置跳跃问题，系统采用单点跳跃数据去除和定时显示的方式，单点跳跃数据可能是因为数据重发导致的，这类数据没有被过滤掉，但在车辆行驶行为上没有意义，定时显示策略则是取一小段时间的车辆位置信息，并根据地图路线是坐标密集化来避免车辆位置跳跃问题。最后通过jQuery选择器来控制html中相应的dom元素的显示。展示效果见图5-7中位置信息，图5-8中地图坐标转换算法代码如下：

function WGS-to-GCJ(wgLat, wgLon){

var a = 6378245.0;

var ee = 0.00669342162296594323;

if(outOfChina(wgLat, wgLon)){ //判断是否超出国境

return {Lat:this.wgLat, Lon:this.wgLon};

}

var dLat = transformLat(wgLon - 105.0, wgLat - 35.0);

var dLon = transformLon(wgLon - 105.0, wgLat - 35.0);

var radLat = wgLat / 180.0 \* pi;

var magic = math.sin(radLat);

magic = 1 - ee \* magic \* magic;

var sqrtMagic = math.sqrt(magic);

dLat = (dLat \* 180.0) / ((a \* (1 - ee)) / (magic \* sqrtMagic) \* pi);

dLon = (dLon \* 180.0) / (a / sqrtMagic \* math.cos(radLat) \* pi);

var mgLat = wgLat + dLat;

var mgLon = wgLon + dLon;

return JSON.stringify({"Lat":mgLat, "Lon":mgLon});

}

1. 历史轨迹回放

有时候，我们需要对一辆车的行驶轨迹单独查看，同理我们通过点击图5-7中的行驶轨迹按钮，车辆标记会一段时间内的行驶轨迹重新在地图上行驶一遍，如图5-9所示。



图5-9行驶轨迹效果图

1. 车辆统计数据显示

当我们需要对一辆车一段时间的行驶数据进行调查时，系统使用一个单独的页面图表来记录，记录车辆在行驶过程中产生的各类数据，如图5-10所示，是测试过程中产生的数据。

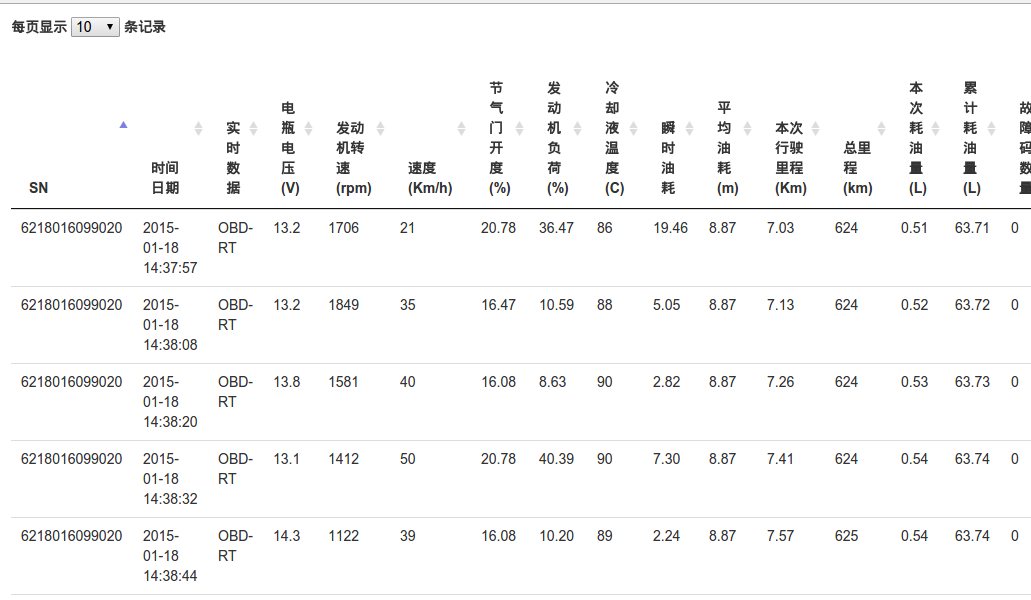


图5-10测试统计数据

1. 道路轨迹规划

这属于面向服务扩展的功能需求，这部分功能系统通过调用高德地图服务api提供的service接口，系统在使用开发过程是分别调用AMap.Transfer，AMap.Walking，AMap.Driving三类服务分别用来规划公交，步行和自驾的路径规划，输入起点、终点信息，返回相关的路径规划信息，包含站点、路径信息。

实现代码如下：

AMap.service(["AMap.Driving"], function() {

var driveOptions = {

map: mapObj,

city: '成都市',

panel:'panel',

policy: AMap.DrivingPolicy.LEAST\_TIME

};

var drive = new AMap.Driving(driveOptions);

drive.search([{keyword:‘电子科技大学（沙河校区）’},{keyword:‘东郊记忆’}], function(status, result){

});

调用AMap.Driving服务实际效果图如图5-11所示：



图5- 11路径规划效果图

5.3 本章小结

本章主要介绍了数据监控平台的设计与实现，分别从地图如何加载，数据交互方式介绍系统平台的实现。此外，还阐述了基于当前数据监控平台实现的一些主要功能需求，并予以相应的显示。

第六章 系统测试

评定一个系统是否可用有两个标准，一种是系统是否实现了需求，另一种是系统在性能上是否满足需求，下面我们基于两个标准来对系统进行测试。

6.1 测试条件准备

在系统测试之前，我们需要为测试过程中所需要的各项条件准备好，以下便是所有测试条件以及准备过程：

1. 实验数据

实验数据是通过租赁的三辆普通汽车，分时间段三天内跑出来的数据，三辆车均搭载了OBD车载装备，系统将获得的数据分别存入MongoDB和日志文件中，日志文件存储的是相对源数据比较接近的数据，因此我们的数据是通过并发读取日志文件然后输入到系统中的，开启的并发线程越多，对系统的压力便越大，并观察此时的系统并发量以及响应时间。考虑到测试平台的局限，实验对并发的线程数做出了一定的控制。日志部分数据样本如图6-1显示：

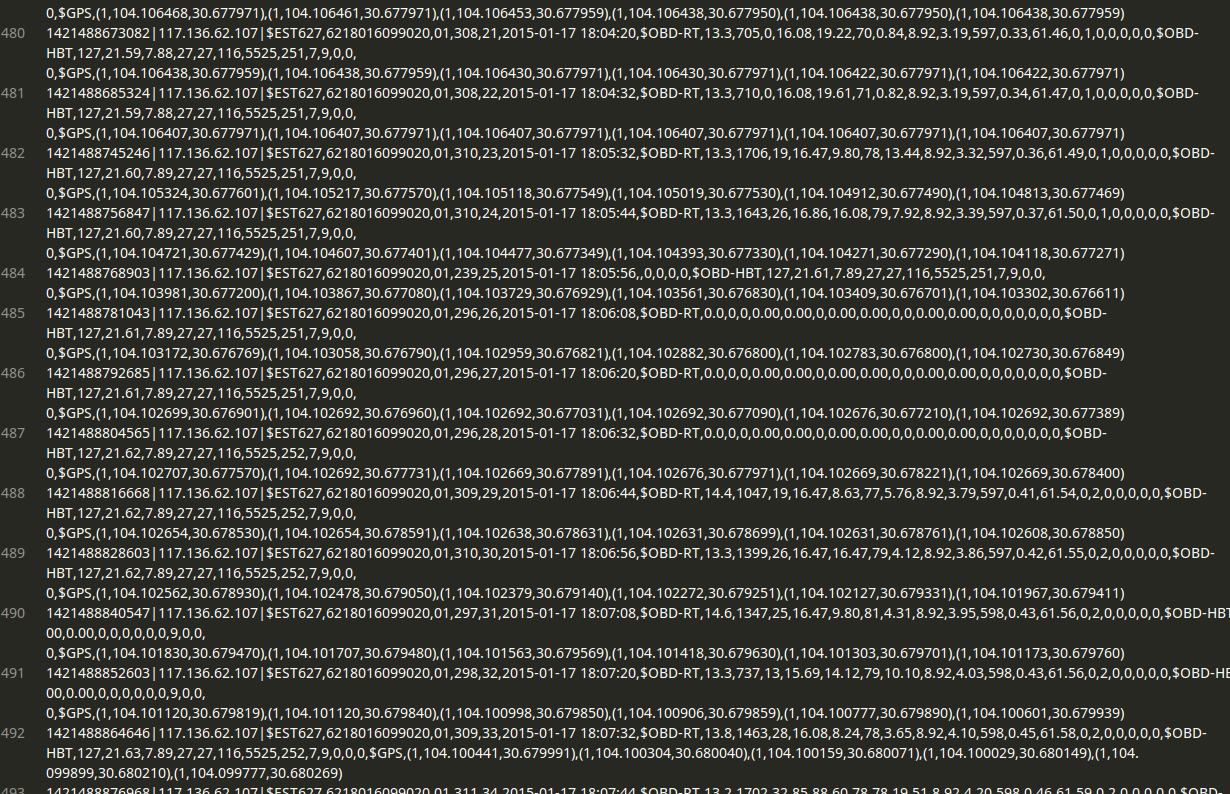


图6- 1部分日志数据样本

1. 测试平台

测试平台也是实验基本需求的，包括软件平台和硬件平台。

硬件平台：一台服务器一台PC机，分别作为并发数据接收端和数据处理端，本节主要针对实时数据处理进行测试，下面是作为实时数据处理端PC的基本配置信息如表6-1所示：

表6-1实时流数据处理端硬件环境

|  |  |
| --- | --- |
| 硬件环境 | 详细 |
| 车载终端 | On-Board Diagnostic |
| 服务端内存 | 16.00 GB ( 1600 MHz) |
| 服务端硬盘 | 32M硬盘缓存1 TB硬盘存储 |
| 服务端处理器 | IntelI CoreI i7-4720HQ 四核CPU @ 2.60GHz(2601 MHz) |
| 服务端网卡 | Killer e2200 Gigabit Ethernet Controller (NDIS 6.30) |
| 测试端 | PC机 |

软件平台：实时流数据处理平台以及相关使用到的数据库，基本信息显示如表6-2所示：

表6-2实验软件环境

|  |  |
| --- | --- |
| 软件环境 | 详细 |
| 操作系统 | Ubuntu Kylin 14.04 |
| Redis | Redis-3.0.7 |
| 数据库 | MongoDB-3.2 |
| Node版本 | V5.0.4 |
| 其他依赖 | {"cheerio": "^0.17.0", "commander": "2.2.0","geoip-lite": "^1.1.3", "heap": "0.2.3","just.randomstring": "0.1.1", "log4js": "0.6.9","mathjs": "^1.0.0", "nodegrass": "0.0.3","ua-parser-js": "^0.7.3", "urlencode": "0.2.0","useragent": "^2.0.9", "zerorpc": "0.9.3","zmq": "2.7.0"} |

6.2 实时流平台功能测试

我们对实时流数据处理平台主要有两个功能需求：第一，需要满足将前端需求数据推送到前端；第二就是将数据落地存储到MongoDB。

在实验的过程中，我们需要写一个读取日志文件的程序来模拟我们的数据流流入系统，我们在写这个程序的时候会把读取到的每条数据显示在终端上，如图6-2所示：

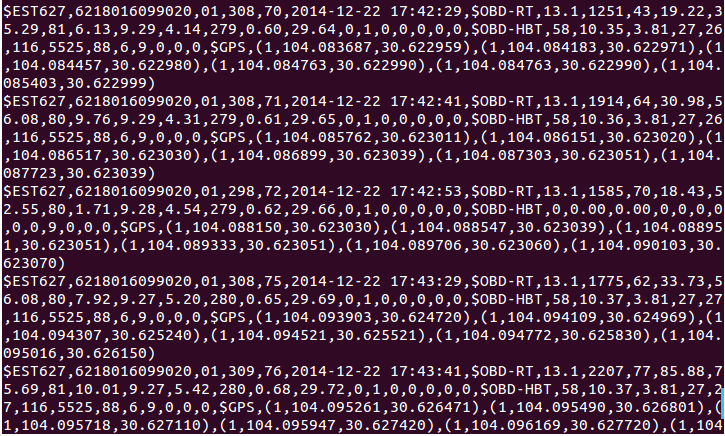


图6-2日志文件读取记录终端显示

第一个功能我们要确定这些信息是否被我们的前端模块收到，这里我们需要核对下数据是否一致，我们使用一台机器模拟了服务端和浏览器端，通过校验时间来验证数据一致性，如图6-3所示：

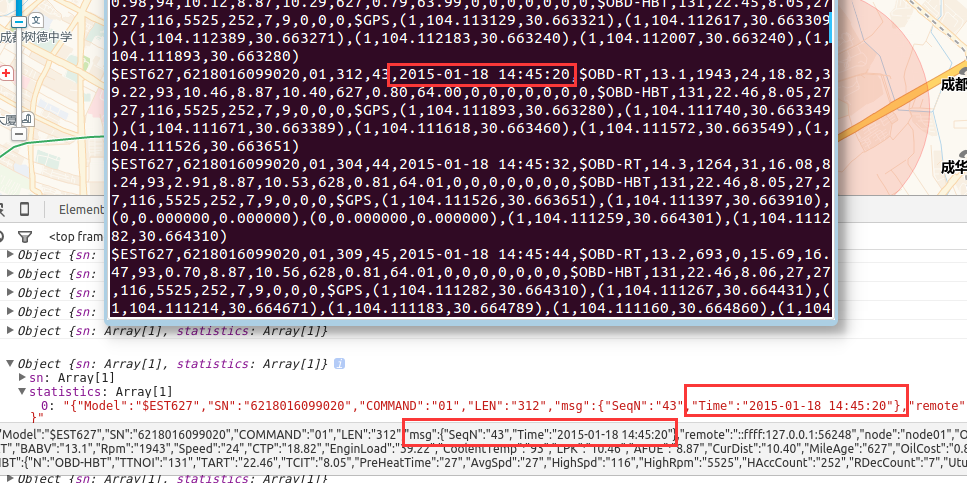


图6-3前端接收数据显示

第二个功能，我们看下系统MongoDB中是否有对应存储记录，通过校验存储记录的时间来进行核对，如图6-4所示：

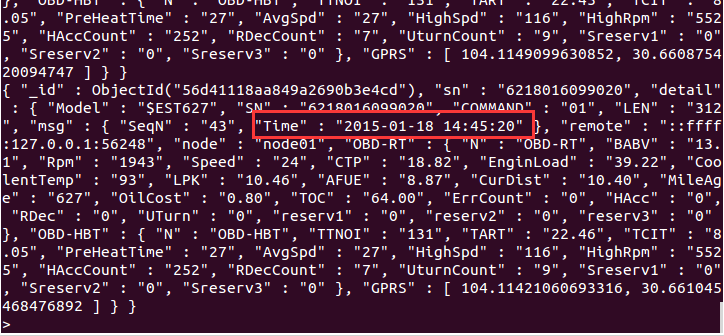


图6-4数据库相应数据存储记录

从上面结果可以看出来，基于实时流数据处理平台的功能基本实现，对数据的转发和落地存储功能均无问题，后面我们需要对系统进行一次性能测试，检测系统在高并发环境下对数据吞吐量的影响。

6.3 系统性能测试

系统性能测试过程中，我们先将源数据以日志的形式存储在本地硬盘里，通过编写日志读取程序来读取日志文件来模拟大量的大量数据流，日志样本如图6-1所示。读取的方式为按条读取，通过控制信息的条数来观察系统的性能。性能测试主要分为两个方面：一方面是管理员访问监控页面的响应时间；另一个方面是数据从接收到存入数据库所消耗的处理时间。

表6-3显示了是关于监控页面响应时间测试的部分详细数据，request表示请求次数，Average表示平均响应时间，Min表示最小响应时间，Max表示最长响应时间，40%表示有又百分四十的请求的响应时间在表格里数值以内，后面也是相

表6-3监控页面响应时间测试数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 编号 | 请求数（个） | 平均响应时间（ms） | 最短响应时间（ms） | 最长响应时间（ms） | 40%（ms） | 70%（ms） | 90%（ms） |
| 1 | 2525 | 15 | 3 | 90 | 13 | 16 | 19 |
| 2 | 3189 | 16 | 5 | 97 | 14 | 15 | 20 |
| 3 | 2722 | 14 | 3 | 120 | 12 | 15 | 22 |
| 4 | 3012 | 18 | 6 | 118 | 13 | 15 | 19 |
| 5 | 3717 | 18 | 8 | 109 | 14 | 16 | 20 |

同。

表6-4显示了实时流处理平台处理数据所消耗的处理时间，messages表示系统模拟读入日志数据到流数据平台的记录条数，time表示流处理平台处理这些数据所消耗的时间。随着数据量的指数增加，平台的处理时间的增长速度低于线性增长速度。

表6-4实时流处理平台处理时间测试结果

|  |  |  |
| --- | --- | --- |
| 编号 | 消息数（条） | 处理时间（ms） |
| 1 | 102 | 22880 |
| 2 | 103 | 23891 |
| 3 | 104 | 26980 |
| 4 | 105 | 34655 |
| 5 | 106 | 39891 |

6.4 本章小结

本章主要介绍了是对系统主要功能和性能方面的测试。首先介绍测试条件以及源数据的获取，然后分别从功能和性能两个方面对系统进行了测试并得出结论。

第七章 总结与展望

7.1 本文总结

汽车在现实生活中的应用越来越多，对于车辆管理公司以及其他车辆相关的行业对车辆的实时监控的需求越来越显著。本文基于科研团队与四川欧润特软件科技有限公司合作开发的项目进行叙述，就实时监控这一主要需求进行展开，并结合当前需求背景，研究开发了一个基于实时流数据处理平台的车联网监控系统。基于当前现有的软硬件技术，本文完成了主要以下几点：

1. 引用当前热门的服务端编程语言node.js，将其应用于车辆网监控系统的实时流处理平台中。基于高性能的I/O的考虑，node.js能够非常有效的处理高性能I/O场景，采用流计算的应用框架，也是node.js和流计算框架的一个集成应用，目的在于构建一个实时的高效的面向I/O密集型应用的流处理平台。这是一次在车辆网应用上的一次创新的尝试。
2. 对于车辆网监控应用，系统主要完成了面向监管人员的基本的车辆监管功能，同时基于流数据处理平台还扩展了相应的服务模块，虽然这个模块的应用还属于测试阶段。
3. 数据监控平台的开发结合了高德地图和JavaScript以及html5技术的配合应用，实时有效的展示了车辆状态以及各类功能的展示效果。
4. 尝试使用redis内存数据库应用于实时通信，基于其pub/sub机制进行模块之间的通信，非常有效的降低了模块之间的耦合度，提高了系统的灵活性和可扩展性。

总之，这是一次许多关键技术应用面向车联网的一次尝试，该项目已经应用实际生产环境中，并得到了充分的检验。在此同时也为其他车联网应用系统提供了部分参考价值，希望在实践中有更好的发展。

7.2 对未来工作的展望

当前，本文研究系统仍然有很多不足，较为复杂的计算会影响的到系统的效率，实时流处理平台和数据监控平台之间的耦合度较高，通信需要建立稳固的连接，面对智能交通的大潮流难免有些笨拙。这是一个长期的需要坚持的应用研究话题。基于本文，对未来有如下展望：

1. 对于当前系统的不完善进行完善，包括面向复杂计算的应用，模块之间的解耦等，使系统具有更高的实时性能以及更高的灵活性。

2. 系统操作比较杂乱，许多应用功能需要单独开启，需要配置一键式应用这种更为之有效的操作，通过脚本来控制整个系统的运行。

1. 数据库方面，MongoDB本身开发是面向大众应用的，因此，基于当前系统需求对数据库的性能方面有待改进和提高。将来面对更大的数据量时会基于MongoDB底层对其性能进行改进以适应系统需求。

总之，系统的功能有待丰富，性能也有待更深的优化，今后的工作会研究更有效更适合的流计算框架应用与系统，更加有效的实时显示平台。

致 谢

论文结束之际，首先，我要向我最尊敬的顾小丰导师表达最真挚的感谢。感谢顾小丰老师在我研究生三年时光里给我的帮助和支持，在三年的学习和生活中，顾小丰老师给了我很大的指导和关心，顾小丰老师工作认真、平易近人，时时刻刻激励着我不断进步，帮助我顺利完成研究生学业并找到一个非常满意的工作。

同时，我还要向科研团队刘玓老师致谢，感谢刘玓老师在我学业过程中给我的指点和帮助，给予了我很大的提点。

感谢与科研团队项目合作的四川欧润特公司的各位经理和同事的指导，在合作期间给予的关心和帮助。

其次，我要感谢科研团队的同学们，同导师下的徐加文同学以及刘玓老师的学生潘冬同学和赵立斌同学，在学习和科研生活中共同监督学习并且共同进步以及共同在生活中开的玩笑，都给予了我很大的帮助。

然后，我要感谢我的室友，曾志强同学、杨眷玉同学和段宇峰同学在生活上给我的宽容和帮助，互相理解互相支持，在学业上互相交流，共同进步。

最后，我要感谢我的家人，感谢家人在我学习期间给予我物质上和精神上的支持，不断鼓励我刻苦钻研、努力学习，未来我将更加努力。

研究生生涯即将结束，感谢电子科技大学三年来给我的支持和帮助，这三年将是我人生中宝贵的三年，在我今后的学习和生活中都将受益无穷，我将会把我三年中学到的知识在工作和生活中得以运用，不断努力奋斗。

参考文献

[1] 钟书华. 物联网演义(一)——物联网概念的起源和演进[J]. 物联网技术, 2012, 05: 87-89.

[2] 王晓欢. 基于车联网的物流车辆监控管理系统设计[D]. 中国地质大学（北京）, 2014.

[3] 刘小洋, 伍民友. 车联网:物联网在城市交通网络中的应用[J]. 计算机应用, 2012, 04:900-904.

[4] 奎丽萍, 罗桂兰. 车联网研究现状及发展[J]. 中国科技信息, 2015, Z4:118-120+113.

[5] Feng Y, Zhao L, Zhao Y, et al. METHOD AND APPARATUS FOR RESOURCE SCHEDULING IN INTERNET OF VEHICLES SYSTEM: , WO/2015/158251[P]. 2015.

[6] Yang F, Wang S, Li J, et al. An overview of Internet of Vehicles[J]. Wireless Communication Over Zigbee for Automotive Inclination Measurement China Communications, 2014, 11(10):1-15.

[7] Kunicina N, Galkina A, Zabasta A, et al. Traffic Route Modelling and Assignment with Intelligent Transport System[J]. Electrical Control & Communication Engineering, 2014, 7(1):34-40.

[8] 曹磊. 全球车联网发展态势研究[J]. 竞争情报, 2014, 04:31-44.

[9] Goos G, Hartmanis J. Internet of Vehicles – Technologies and Services[J]. Lecture Notes in Computer Science, 2014, 8662.

[10] 段玉保. 我国车联网发展探析[J]. 信息技术与信息化, 2015(1):18-19.

[11] 郭华. 智能化信息化引领车联网发展[J]. 移动通信, 2016(1).

[12] 张龙. 基于GPS+北斗+4G的车载终端的OBD诊断系统[J]. 城市建设理论研究:电子版, 2015, 5(12).

[13] Wu J, Zhang L, Bai Y, et al. Cluster-Based Consensus Time Synchronization for Wireless Sensor Networks[J]. IEEE Sensors Journal, 2015, 15(3):1404-1413.

[14] 徐妍. 基于工业组态技术的交通控制系统[D]. 北方工业大学, 2014.

[15] Yu D, Chander A, Islam N, et al. JavaScript instrumentation for browser security[J]. Acm Sigplan Notices, 2007, 42(1):237-249.

[16] 许会元, 何利力. NodeJS的异步非阻塞I/O研究[J]. 工业控制计算机, 2015, 28(3):127-129.

[17] Xu M, Xu X, Xu J, et al. A Forensic Analysis Method for Redis Database based on RDB and AOF File[J]. Journal of Computers, 2014, 9(11).

[18] 刘旭. Chrome V8引擎中的 JavaScript数组实现分析与性能优化[J]. 计算机与现代化, 2014(10):66-70.

[19] TOM BARKER[美]. JavaScript性能优化[M]. 机械工业, 2014.

[20] 谢骋超. Node.js背后的V8引擎优化技术[J]. 程序员, 2014(3):56-60.

[21] Barsness E L, Darrington D L, Peters A, et al. Enhanced garbage collection in a multi-node environment: US, US8868623[P]. 2014.

[22] Xu X, Shen J. Research on stack-allocation based JVM garbage collection[C], Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on. IEEE, 2010:V2-346-V2-349.

[23] Sánchez A, Sánchez C. Formal Verification of Skiplists with Arbitrary Many Levels[M], Automated Technology for Verification and Analysis. Springer International Publishing, 2014:314-329.

[24] 何晓亮, 宗欣, 全喜伟,等. Redis内存数据库在互联网交通安全综合服务平台中的应用一[J]. 道路交通科学技术, 2014(5):7-10.

[25] 王光磊. MongoDB数据库的应用研究和方案优化[J]. 中国科技信息, 2011(20):93-94.

[26] 贺建英. 大数据下MongoDB数据库档案文档存储去重研究[J]. 现代电子技术, 2015, 38(16):51-55.

[27] 周耿达. MongoDB 与MySQL Cluster 效能分析[J]. 台北商业技术学院资历与决策科学研究所学位论文, 2014.

[28] Kookarinrat P, Temtanapat Y. Analysis of Range-Based Key Properties for Sharded Cluster of MongoDB[C], 2015 2nd International Conference on Information Science and Security (ICISS). IEEE Computer Society, 2015:1-4.

[29] 叶锋. Python最新Web编程框架Flask研究[J]. 电脑编程技巧与维护, 2015(15):27-28.

[30] Mardan A. Intro to Backbone.js[M], Full Stack JavaScript. Apress, 2015.

[攻读硕士期间取得的学术成果](#_Toc19205)

Xiaofeng Gu, Le Yang, Shaoquan Wang. A real-time stream system based on node.js[C], 2014 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP, 2014:479 - 482). EI Compendex, Accession number:20151700788241