

电子科技大学  
university of electronic science and technology of china

# 专业学位硕士学位论文

master thesis for professional degree



论文题目 基于实时数据流处理的 http 数据分析可视化系统

专业学位类别	工 程 硕 士
学 号	201322220213
作 者 姓 名	潘冬
指 导 教 师	刘玓 教授

分类号\_\_\_\_\_密级\_\_\_\_\_

udc<sup>注1</sup>\_\_\_\_\_

# 学 位 论 文

基于实时数据流处理的 http 数据分析可视化系统

(题名和副题名)

潘冬

(作者姓名)

指导教师

刘均

教授

电子科技大学

成 都

(姓名、职称、单位名称)

申请学位级别 **硕士** 专业学位类别 **工 程 硕 士**

工程领域名称 **信息与软件工程**

提交论文日期 **2016.7.1** 论文答辩日期 **2016.7.7**

学位授予单位和日期 **电子科技大学** **2016 年 12 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。

**research and implementation of  
realtime stream computing data analysis system on  
http messages**

a master thesis submitted to  
university of electronic science and technology of china

major: **software engineering**

author: **Dong Pan**

supervisor: **Di Liu**

**university of electronic science and technology  
school: of china**

---

## 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

互联网时代，数据量飞速增长：城市数据、医疗数据、网站数据等数据不断的产生。这些数据隐含着人们生活活动的规律和社会发展的规律，有很大的分析价值。但在这些数据中，却有大量数据因为难以保存而直接丢弃了。这些被丢弃的数据中隐含着有价值的信息，却没有得到利用。为了挖掘这些信息的隐含价值，可以使用流计算方法分析这些数据。流计算是一种数据分析方法，这种方法是实时的，它在得到数据的同时同步进行数据分析，避免了原始数据的累积。同时实时产生结果，保证了分析结果的有效性和可用性。使用流数据分析方法，可以挖掘那些难以保存的大量数据的隐含价值。

用户访问网站时会产生大量 http 数据包。而因为 http 报文数据量大，冗余信息多，使用传统的存储再分析方法来分析 http 数据代价很大，性价比很低，所以人们访问网站时产生的 http 报文通常用过即丢。本文使用流数据分析方法来处理 http 数据，可以在有限存储容量开销条件下挖掘这些数据的价值。这篇论文的研究方向，是从 http 报文中实时分析网站用户的基础数据，包含用户 pv/uv、访问深度、停留时间、地理位置、搜索关键词等。实时显示分析结果。分析产生的用户数据可以指导网站的决策、引导网站的建设、验证网站的营销成果、支持上层多维用户行为分析。

本文的主要贡献有三个方面：一是通过 nodejs 技术和 redis 技术编码完成了一套流数据分析程序；二是设计并实现了一套从 http 报文中分析用户行为数据的拓扑流程，其中多个分析目标节点（pv/uv、来源网站、关键词、访问深度、停留时间、地理位置）按流水线分布；三是设计了数个用户行为分析算法，算法功能为从输入的流数据中分析计算用户 pv/uv、访问深度、停留时间。

**关键词：**流计算，用户行为分析，访问深度，停留时间，用户地理分布

## ABSTRACT

The internet is developing very fast, the amount of data is growing very fast: urban data, medical data, web data, etc. These datas reflect the laws of people's daily life, they worth being analyzed. However, among all these datas, a large amount of data could not be saved for storage shortage. Http messages for example, when a user visits a web site, if all http messages are to be saved, the storage usage would grow in an amazing speed. Normally, after the user browsed the site, http datas would be discarded. In order to excavate the value of these http datas, this thesis uses stream calculation method to analyze http datas. Stream calculation is a data analyze method, which is real time. In http analysis stream calculation system, http datas would be analyzed immediately after they arrive, results would be saved and results only. The step to save original datas in storage could be avoided. The implied value of the http datas could be significantly taken use of.

This thesis builds a stream calculation system that can analyze http datas in real time. Analysis results include users' pv/uv, stay time, visit deepth, etc. The results could be used to support higher level user behavior analysis, guide the site owners's decision-making, guide the construction of the website, or verify the site's marketing results. Main contributions of this thesis are as follows:

(1) This thesis designs a user-behavior analysis node cluster which can detemines user's information by filtering http messages in real time. Analysis nodes in the cluster work parallely in assembly line. Results include the number of new users, users' access depth, users' retention time, users' coordinates, etc.

(2) This thesis proposes an algorithm which can detemines user's visit times in real time. This algorithm uses set structure, filters http messages, identify user's once operation in many http messages, do calculation.

(3) This thesis proposes an algorithm which can calculates users' access depth in real time. This algorithm uses a hash list to build a circular counting queue. Thus realizes a sliding window and do the calculation.

(4) This thesis proposes an algorithm which can calculates users' stay time in real time. This algorithm uses two hash lists and some timers, builds the process to calculate the residence time of users.

**keywords:** stream calculation, user-behavior, visit depth, residence time

# 目 录

<b>第一章 绪 论</b> .....	1
1.1 论文的研究背景 .....	1
1.2 相关技术国内外研究历史与现状 .....	3
1.2.1 流计算技术 .....	3
1.2.2 网站用户行为分析技术 .....	7
1.3 本文的主要贡献 .....	8
1.5 论文的结构安排 .....	8
<b>第二章 关键技术研究</b> .....	11
2.1 nodejs 技术研究 .....	11
2.2 redis 数据库和 mongo 数据库技术研究 .....	13
2.3 本章小结 .....	16
<b>第三章 流计算框架设计</b> .....	17
3.1 系统功能需求分析 .....	17
3.2 系统总体设计 .....	17
3.3 系统节点部署模块设计 .....	18
3.3.1 master 程序设计 .....	20
3.3.2 daemon 程序设计 .....	22
3.3.3 client 程序设计 .....	22
3.4 系统工作流程 .....	24
3.5 nodejs 多核 cpu 利用方法 .....	24
3.6 vrssstream 和 vrsscore 消息分发组件 .....	25
3.7 系统的容错设计 .....	27
3.8 流数据生命周期 .....	28
3.9 本章小结 .....	30
<b>第四章 用户行为分析节点设计</b> .....	31
4.1 节点集群功能需求分析 .....	31
4.2 节点集群总体结构设计 .....	31
4.3 redis 数据库结果集数据结构设计 .....	33
4.4 dayinfo 分析节点设计 .....	35
4.4.1 用户访问量 pv 的分析依据 .....	37

4.4.2 新用户 uv 的分析依据.....	39
4.4.3 来源网站和访问页面的分析依据.....	40
4.4.4 dayinfo 节点内置函数说明 .....	42
4.5 用户浏览器和设备分析节点设计 .....	43
4.6 用户地点分析节点设计 .....	44
4.7 用户关键字分析节点设计 .....	47
4.8 用户访问深度分析节点设计 .....	48
4.9 用户停留时间分析节点设计 .....	50
4.10 socketio 消息推送节点设计 .....	52
4.11 数据持久化节点设计 .....	53
4.12 本章小结 .....	57
<b>第五章 数据显示模块设计 .....</b>	<b>58</b>
5.1 数据显示功能需求 .....	58
5.2 前端系统总结构设计 .....	58
5.3 图表显示方法设计 .....	59
5.4 mongo 数据推送设计 .....	60
5.5 页面导航菜单设计 .....	61
5.6 抓包模块设计 .....	62
5.7 本章小结 .....	63
<b>第六章 测试 .....</b>	<b>64</b>
6.1 测试需求和计划 .....	64
6.2 测试环境 .....	64
6.3 http 报文模拟模块设计 .....	65
6.4 性能测试 .....	65
6.5 功能测试 .....	66
6.6 本章小结 .....	74
<b>第七章 全文总结与展望 .....</b>	<b>75</b>
7.1 全文总结 .....	75
7.2 后续工作展望 .....	75
<b>致 谢 .....</b>	<b>77</b>
<b>参考文献 .....</b>	<b>78</b>
<b>攻读硕士学位期间取得的成果 .....</b>	<b>80</b>



## 第一章 绪 论

流计算方法可以实时动态的处理连续无结构数据, 适合分析难以保存的数据, 实时提供分析结果。而 http 报文难以大量保存, 其中又有很多值得分析的目标, 适合用流计算方法进行分析。本文尝试结合这两者, 将流计算方法运用到 http 数据分析上, 探索分析目标和分析方法。

### 1.1 论文的研究背景

互联网时代, 网络上的信息量飞速增长, 这些数据不仅有传统的视频/音频文件等结构化的数据, 还有很多来自电子邮件、传感器、网站、实时通信的非结构化数据。对这些数据进行分析可以得到很多有用的信息, 可以用来推测用户的消费趋势、兴趣焦点, 也可以用来研究潜在的社会变化规律。在分析这些数据时先存储这些数据然后进行分析是通常的做法, 存储分析方法可以看到全部数据、可以直接对比数据的特征, 容易分析出科学的结果, 是准确率高的分析方法。但是当数据量越来越大时, 存储分析方法需要的存储空间和计算量也越来越大, 逐渐难以满足, 而且分析周期长, 分析结果难以及时被利用。传统存储分析方法的性价比不能让人满意时, 实时分析处理数据成为新的研究焦点, 流计算方法正是为了处理海量的非结构化数据而产生的。流式计算是一种计算分析模型, 这种模型中各个分析节点可以分布在不同的物理机器上, 整体处理逻辑以流的形式在计算单元之间流转形成。各个节点同步工作, 可以在得到数据的同时展开数据分析。只存储数据分析结果, 实时产生结果。

用户行为分析, 是以统计用户的基本信息<sup>[1]</sup>为基础的数据分析与应用研究。用户基本信息包含用户访问量、新用户数、用户访问深度、用户停留时间等。这些基础信息具有用户行为统计的理论价值, 而在得到基础数据之后, 可以在这些基础用户数据之上分析复杂的高维用户行为信息, 比如用户在不同时间访问不同页面的数量比例、用户访问网站时离开的频率、已经访问过的用户再次访问的概率、新访问者的数量、老用户的回访次数、老用户再次访问相隔的天数、注册用户访问数量和没用注册直接访问的用户数量的比例等等。

用户行为分析具有重要的意义, 统计、分析、对比这用户行为数据能够发现用户访问网站的规律, 能够用以分析社会现象、揭示社会行为变化规律。

目前国内外网站的用户行为信息分析一般包含但不限于以下数据分析项目:

1. 用户的访问的页面计数。

2. 用户使用什么途径（广告，链接）来到网站的最多<sup>[2]</sup>。
3. 用户访问最多的页面。
4. 用户的访问深度（不同页面数量）。
5. 用户在网站的停留时间。
6. 用户搜索的关键词频率统计。
7. 用户使用的设备（手机、平板、电脑）。

如果企业能够通过用户行为信息来推测出用户的喜好、访问习惯。就能做出更好的产品来引起用户的兴趣，设计更加人性化的页面，进而提高企业的业绩。随时关注用户访问数量来源的变动，也可以帮助企业看到他们现行的营销方案的问题所在，帮助他们弥补不足，改正错误。

亚马逊电商网站，作为一个知名的电商网站，就很重视用户行为信息的获取和使用。他们通过分析用户行为信息，取得了很大的商业优势，让自己的网站得到了发展。用户行为分析被他们用在推荐系统上和服务系统上。例如：当客户浏览了多个洗衣机，但是最后没有下单付款时，过一段时间后，亚马逊网站就会周期性的向用户发送邮件和短信，在这些邮件和短信中，会根据用户的浏览历史来推荐一些价位合适，用户喜爱的洗衣机。再例如，用户以前访问过洗衣机的购买页面但是没有买的时候。就把用户访问的商品的价格记录下来，在用户信息中注明用户可能的消费能力，当用户再一次来到亚马逊网站的时候，就根据用户的消费能力推荐相应价格的商品。消费能力高的用户向他推荐高档的洗衣机，消费能力低的用户向他推荐低端的洗衣机。这样一来系统就显得很人性化。低端买家不会因为推荐了买不起的商品感到尴尬。高端买家也不会因为被推荐了低端商品而感到失望。

目前大型网站在统计网站用户行为信息时，一般在网站后台响应 restful 的接口函数中加入统计模块，或在网站 html 中注入 js 代码进行统计。这是传统的统计方法，但是这种方法与网站的结合度非常高，必须对网站的架构流程有深入了解才能使用，而且一旦修改网站，必须重新设计相应的统计模块，这种方法的灵活度和移植性较差，并需要有专业知识的人员来构建和修改。而本论文构建的 http 数据分析系统，与网站本身完全独立，不需要和网站后端结合。可以方便灵活的配置，也容易移植到新的网站上。同时，有部分数据只存于 http 报文中，例如 http 报头中的 user-agent 字段。通过分析 user-agent 字段，可以确定用户使用的设备和浏览器版本。本文构建的系统可以分析这部分数据。

http (hypertext transfer protocol) 是一种网络上交换数据的协议，现在的网页在浏览时大多都是通过 http 协议从服务器下载网页的 html 代码，然后进

行显示。http 协议是一个无状态的协议，分为报头和报体。报体是网站本身的代码。报头中有浏览器之间互相沟通的信息，这些信息对网站用户透明，包含协议版本、错误代码、状态码、浏览器代理等信息。

用户的行为可以通过分析 http 报文得到，但是网络的流量非常巨大，如果将所有 http 报文都存储在数据库中再进行分析，在开始数据分析之前，很快就将累积到巨大的数据量。同时，数据的隐含价值随时间下降，如果不能及时得到有用的数据分析结果，则数据分析的意义也会大大降低。因此大量 http 报文用传统方法分析不仅难以办到，而且会因为周期太长使分析结果意义降低。

为了避免数据大量累积、合理利用存储资源、及时得到可用结果，本文用流计算的方法来分析 http 数据，实时提供分析结果。本文构建的系统相对已有的用户行为分析系统的创新在于：

1. 使用了流计算方法。本文设计并完成了基于流计算的用户行为分析系统。和传统的统计历史数据的用户行为分析系统不同，该系统使用流计算方法，实时分析，实时显示结果。
2. 完全由 http 数据得到分析结果。本系统不需要和网站后台交互。挖掘只能从 http 数据分析得到的用户行为数据，尽量挖掘 http 数据的价值。
3. 设计了一套用户行为分析节点集群，提出了三种实时分析用户行为（pv/uv、访问深度、停留时间）的算法。算法从动态数据中分析并产生结果，和传统的基于静态数据的分析算法不同，是实时分析，累积分析结果的算法。

## 1.2 相关技术国内外研究历史与现状

这一节阐述论文的研究领域在国内外的研究历史与现状。分为流计算技术和用户行为分析技术的国内外研究历史与现状。

### 1.2.1 流计算技术

由于技术的飞速发展，网络接入技术和网络传输技术不断进步，网站的规模越来越大，世界各地越来越多的人将越来越多的时间用在网络世界上。这形成了一个充满真实情感、利益、言论的网络世界。而智能手机的普及，使手机上 GPS 提供的用于测量温度和湿度的传感器遍布四方，加上世界各地使用的所有传感器和仪表，使我们通过数据亦能看到不断演化的真实世界情况。这些构成了应该转换成信息的大量数据。

我们可以利用群众的力量来洞察不断变化的情形。我们可以察觉出可能产生机遇或避免严重灾害的趋势。我们可以看到电信与其他行业中社交媒体观点的实时变化、能量消耗的当前状态、关键性能指标的分析、网络用户的行为趋势。

大数据可以观察世界，可以推演事务发展的趋势，可以用来改变世界。IBM 公司很早就开始提出的“智慧地球”概念，包含了物联化、智能化、互联化三个主要概念。即是一个利用大数据改变生活的理念。

但是在利用这个新的数据机会时，存在几个重大问题。其中一个数据的绝对量，另一个是大量数据的短暂性。为了解决这些问题，出现了流计算方法。以处理大量、快速、时变（可能是不可预知）的数据流为目的的数据分析方法。

流数据有种种不同于常规数据的特点，为了处理流数据，解决流计算领域的特殊问题，各个科技公司和学术机构都对流计算有很多研究。早期流计算系统有 IBM 公司的 System S。System S 是一个完整的计算架构，通过“stream computing”技术，可以对 stream 形式的数据进行 real-time 的分析。最初的 System S 系统可以加载大约 800 个微处理器。系统中最关键的部分是 System S 软件。System S 软件可以分割任务进行处理，然后将处理后的结果碎片组成完整的答案，比如图像识别和文字识别可以被分割开来处理。IBM 有一个高性能流运算项目实验室，其中的负责人是 Nagui Halim，他将 System S 定义为一个全新的运算模式：它的灵活性和速度颇具优势。它的方式比传统方式更加智能化，可以通过适当转变自身来适用需要解决的问题。

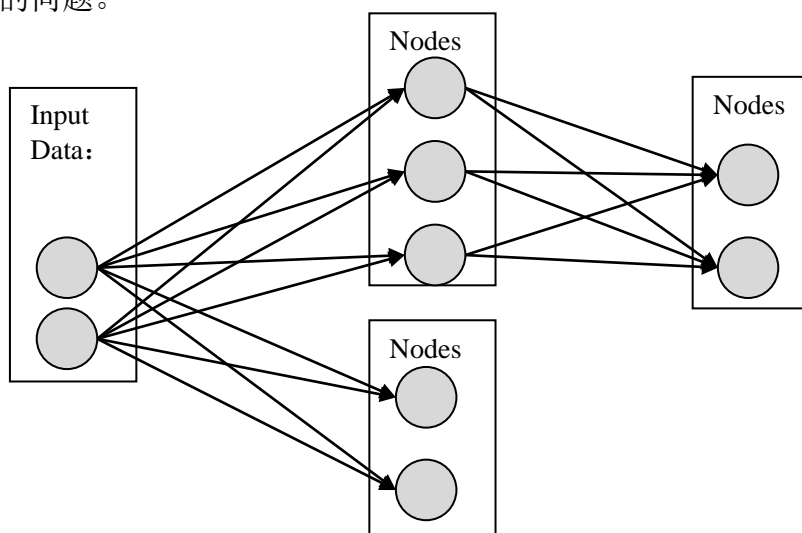


图 1-1 流计算数据流程

如图 1-1 是流计算的数据处理模型。在数据来源处，数据不间断的源源不断的到达。在数据分析节点中，数据随到随处理，中间数据流入下一个分析节点，多

级节点并行工作。动态分析数据，实时产生结果。

在数据流模型中，待处理的数据以“一个或多个数据流”的形式分片到达，数据没有预先存储在硬盘和内存中，无法随机访问。流数据的特性有如下几个方面：

1. 流中的数据元素在线到达。
2. 新到达的数据元素的顺序不能被系统控制。数据元素可能跨越多个流数据片段。同样的数据重新发送一次，可能以不同的顺序到达。

3. 数据流的潜在大小无法得知，也许是无穷的。

4. 数据间关联极小，数据经过时才能看到，不容易被检索。

斯坦福大学将流数据的特征定义为连续、非结构化、不间断。流计算方法是处理分析流数据的分析方法。流计算是一种高效利用并行和定位，使用支持流计算处理的编程语言和处理器，处理连续不间断的非结构化数据的新颖数据分析方法。流计算的实时处理模式有这些特性：

1. 实时性，流计算的数据来源和结果都是实时的。
2. 无先验知识<sup>[3]</sup>，系统对将要处理的数据没有先验知识，无法得知数据的到达时间，无法得知数据量的大小。潜在的数据量可能是无限大的。
3. 数据流不间断，在处理的同时又有新数据到达。
4. 数据的处理不是一次性的，而是一个动态的随到随处理过程。
5. 数据流模型中也引用了一些关系型操作语义<sup>[4]</sup>，如数据流的查找、选择、连接。它们的含义可以进行类推。

传统的数据分析模型中，数据是静态的并和应用分离的。当需要分析这些数据时，可以一次性的取出所有数据，数据持久而稳定。但是在这种数据分析模型中要存储全部的数据。为了满足不断增长的数据分析任务的空间容量需求，出现了分布式存储方法。在分布式存储方法中，将数据存储在许多个不同的物理机器上，需要使用数据的时候，同时在许多机器上进行存储或查询操作。分布式存储系统中比较著名的有谷歌的 gfs 和开源的 hdfs 系统。

分布式存储系统的存储能力非常强大，但是这仍然是对传统数据分析模型的进化和扩展。数据仍然需要全部存储下来才能展开处理，仍然会占用大量存储空间。随着发展，数据的规模逐渐超出传统数据分析方法的极限。即使是分布式的存储系统也将无法承受如此大规模的数据。数据的持久化静态存储在大规模的数据不断产生的情况下越来越难以实现。

流计算的解决方法是不再面向静态数据，而是面向动态数据。在流计算领域中，静态数据是指长期存储在某种介质上的数据，通常指文件和数据库。在静态

数据模型中，无论数据来自何处，都会先存储在磁盘上，然后供用户使用。而动态数据不经过硬盘，数据处于流转状态，每一条数据短暂经过一个程序，很快就消失了。

yahoo 公司开发了现在流行的流计算平台 s4。它们开发 s4 最初的目的是为了处理广告流量的统计，将点击率低的广告统计出来换上点击率更高的广告。yahoo 在处理广告数据这种大规模分布式数据之初，使用的是 hadoop 系统，以 mapreduce 模式处理数据。但后来 yahoo 决心开发 s4。因为 mapreduce 系统主要解决的是对静态数据的批量处理，流计算强调数据的动态流形式和实时性，而广告系统正是重视动态数据实时处理的系统。

下面介绍三个目前比较成熟的流计算分析系统：

(1) s4 流计算平台<sup>[5]</sup>：s4 是一个可插拔的流计算系统。它支持的功能包括分布式、可扩展、分区容错等很有用的功能。s4 系统是 yahoo 开发的。yahoo 开发 s4 系统的目的主要是为了处理用户的点击回馈，和处理广告的数据显示。

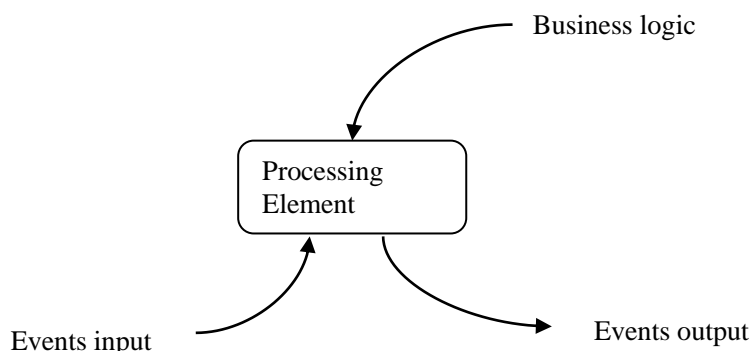


图 1-2 s4 系统处理单元 pe

图 1-2 显示了 pe 的逻辑结构。pe (processing element) 是 s4 系统中的基本处理单元。pe 是直接面向业务方的组件，业务方定义 pe 中对事件的处理方法和处理以后的输出格式，数据分析的其它任务全部由平台完成。

图 1-3 是 s4 中 pn 的逻辑结构。s4 的业务节点是 processing node (pn)，集群中的所有 pn 都是对等的。一个 pn 中可以包含多个 pe。Event Listener 负责监听事件并转交给 pe 容器(processing element container, pec)，由 pec 交由合适的 pe 处理业务逻辑。

(2) twitter 的 storm<sup>[6]</sup>：storm 是一个分布式的实时计算系统。它可以分布式部署。storm 可以支撑并行的操作，如查询的并行，它的查询功能甚至可以在数据流中进行查询。storm 非常善于和消息以及数据库交互。现在 storm 已经是最流行的开源流数据处理系统。

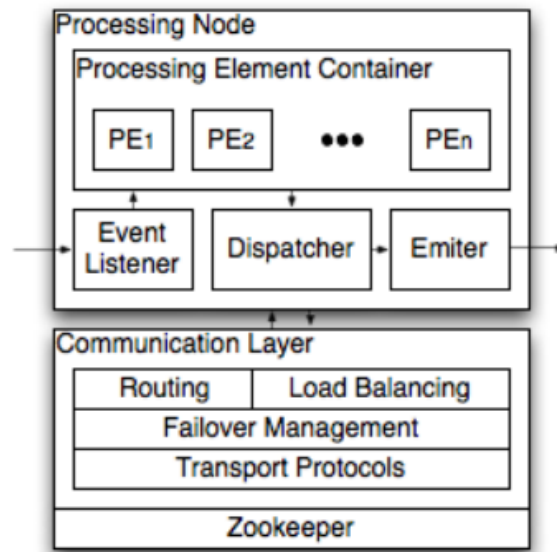


图 1-3 s4 系统处理节点 pn

(3) facebook 的 puma: facebook 使用 puma 和 hbase 相结合来处理实时数据, 另外 facebook 发表过一篇利用 hbase/hadoop 进行实时数据处理的论文 (apachehadoop goes realtime at facebook), 通过一些实时性改造, 让批处理计算平台也可以实时计算。但改造后的批处理系统有许多问题, 流式处理的模式不同于传统方法, 流计算和批处理必须使用非常不同的架构, 想同时满足两方需求, 得到的是一个高度复杂的系统, 并且对两种计算都不理想。

实时计算是分布式计算领域重要组成部分, 各行各业的应用规模都在不断扩大, 它们对实时性的需求也在不断提高。实时数据流计算受到的关注不断加大。改进流计算技术、研究流计算技术的应用是计算机领域发展的一大方向。

### 1.2.2 网站用户行为分析技术

网站用户行为分析, 是指在用户基础数据上展开的用户习惯的分析, 比如关键词分析、数据分析、网站易用性分析。用户在访问网站时的基础数据, 包括用户访问量、新用户数、访问深度、停留时间、跳出率等。通过分析用户行为, 网站管理者可以了解用户关注的重点、用户喜欢的浏览路线、用户搜索的习惯等。这些知识有很大商业价值, 它们能够帮助网站管理者规划网站的建设; 让管理者们对他们的推广手段、营销活动的效果心中有数; 也可以让管理者们在制定新的推广策略时更加精准有效。它们也有很大的社会研究价值, 可以在社会学, 心理学的研究中提供帮助。

用户行为分析的关键概念有下面这些:

### (1) 数据的记录与整理<sup>[7]</sup>

用户基础信息的记录与整理是网站用户行为分析的基础。基础数据记载的越全面详细，后续的分析工作也越容易展开。通常用户行为基础数据来源于网站日志和网站后台统计模块的记录。基础数据有多个维度，在后续分析之前，需要整理基础数据，将多个维度来源的数据整合分类，使后续分析工作得以展开。

### (2) 数据挖掘理论<sup>[8]</sup>

数据挖掘（data mining）是一门交叉学科，近来受到很多关注。它是近年来随着机器学习和人工智能等领域的新的理论和技术突破应运而生的一门技术。数据挖掘的目的是从大量原始数据中，分析出有规律、有价值的信息，探索数据之间的关联，然后用这些知识来帮助理解人们客观世界的内在联系、指导组织结构的进化、帮助企业制定商业决策。这门学科有很大的商业价值。数据挖掘的主要技术包括分类预测、聚类分析、关联规则分析、统计学分析、决策树分析、神经网络和支持向量机等。用户行为分析技术是建立在数据挖掘理论上的。

### (3) 关键词分析

电子商务网站的页面数一般都很巨大，有大量的产品和目录。为了使这些页面带来更大的关键词流量、吸引用户的点击，需要良好的关键词策略。关键词的规划对电商网站的作用不可小觑。在规划商品的关键词时，需要根据用户的关注点，确定一个具有发展性、层次性的关键词序列。所以整理用户关键词，统计用户搜索关键词的排名有着重要意义。

### (4) 数据分析

用户基础数据的获得是用户行为分析的第一步。但用户的行为规律并不足以从简单的访问量判断。应该结合用户的跳出率、停留时间、单个页面流量和总流量比值、在总流量中推广流量的比例等数据，综合分析来确定大量用户的访问习惯<sup>[9]</sup>。

### (5) 网站易用性分析

用户体验是网站建设的重点。增加用户体验是一个综合性很强的工作，需要合理的导航系统、舒适的页面色彩、易用的搜索窗口、合理的页面布局、还需要人性化的引导系统<sup>[10]</sup>。

## 1.3 本文的主要贡献

本文构建了一个实时分析 http 流数据来得到用户行为信息的系统。本文的主要贡献有以下五点：

（1）本文设计并实现了一个流数据分析系统，由三个分布式程序组成。该系



统可以搭载流计算节点，完成流计算分析任务。这套系统使用一个 pub/sub 通道作为流数据的入口，通道接收端的程序接收到数据后，向系统发送消息到达事件。流计算分析节点响应消息到达事件并开始工作。计算节点实时分析，实时产生结果，分析结果实时存入 redis 数据库。计算分析节点可以随时灵活增减，节点的增减不影响其它节点的正常工作。

(2) 本文设计了一个用户行为分析节点集群。可以按照一定流程从 http 数据中过滤出用户访问量、新用户、用户访问深度、停留时间、浏览器、设备、地理位置、来源网站等用户行为数据。集群中设计了 7 个数据分析节点。数据分析多个级别同步进行，多节点并行工作。数据分析节点之间具有关联性，按流水线分布，用户访问深度和用户停留时间分析节点的工作建立在从用户访问量分析节点传递过而来的流数据上。

(3) 本文设计了一种实时统计网站新用户访问量的算法。该算法构建一个 set 结构，实时过滤 http 报文，统计用户访问量。在这个算法的基础上可以改进得到确定新用户数的算法。

(4) 本文设计了一种实时统计网站用户访问深度的算法。该算法构建 hash 数据结构，构建一个循环计数队列，实现一个滑动窗口，可以实时过滤 http 报文，统计过去一段时间内的用户访问深度。

(5) 本文提设计了一种计算网站用户停留时间的算法。该算法构建两个 hash 数据结构，结合定时器，实时过滤 http 报文，可以按给定策略统计用户停留时间。

## 1.4 论文的结构安排

本论文共分为 7 章。

第一章对课题的历史背景、来源、创新点做出了阐述。

第二章简要介绍项目中用到的技术。包含 nodejs 技术、mongo 数据库和 redis 数据库技术。这一章对它们的概念、特点、运行环境、适用领域进行了阐述。

第三章详细剖析本系统的流计算程序的总体设计和模块组成。在这一章主要使用了模块图、数据流程图进行解说。

第四章阐述用户行为分析节点的数据分析逻辑。这一章主要是讲述 7 个数据分析节点的功能和联系，阐述了几种通过分析 http 数据得到用户行为数据的算法。

第五章简述结果显示系统的架构。系统的显示模块是一个 mvc 结构的 web 网页。使用了线图、饼图、地图等开源组件。

第六章记录一份系统测试报告。这一章首先描述测试所用的硬件环境。然后描述测试系统性能和功能的过程。测试程序记录国内某银行电商网站 2015 年某天

24 小时的 http 报文，在开启测试后将之注入流计算分析系统。然后观察系统在数据注入过程中的实时反应。数据全部注入完成后，再综合分析结果，对比不同的用户行为数据之间的数值比例。

第七章对论文的工作做出总结性的描述。对本文的工作未来可能做出的发展和扩展做出猜想和展望。

## 第二章 关键技术研究

搭建流计算环境需要高效的`消息转发模块`和`高效存取的数据库`。本文构建的流计算系统使用 `nodejs` 作为分析节点搭载平台，利用 `nodejs` 高效的事件驱动模块管理节点间数据交互。使用 `redis` 数据库作为中间数据库，利用 `redis` 快速存取的优点保存中间数据。使用 `mongo` 数据库作为结果持久化数据库，利用 `mongo` 分布式、高容错的优点保存历史分析数据。这一章介绍这几种技术。

### 2.1 nodejs 技术研究

`node.js` 是一个基于 `chrome v8` 引擎的 `javascript` 运行环境，它将 `js` 语言的运行环境扩展到后端，使 `js` 语言的优势在后端得到发挥。`node.js` 轻量、高效、事件驱动、使用非阻塞式 `i/o` 模型。

`v8` 引擎是解析运行 `js` 语言的工具，它的技术非常先进。`v8` 引擎本身的执行效率很高，使用了最新的编译技术，这极大的提升了 `javascript` 语言的潜力，使 `js` 语言将有可能应用在更多的场景，发挥更多的功能，承担更多的责任。`v8` 引擎的优秀是 `js` 语言的效率保障的基础。而由于 `js` 语言的易用性，使用 `js` 来开发应用程序可以大大减少开发成本。在这样的情况下，将 `js` 语言迁移到后端的 `nodejs` 技术应运而生。`nodejs` 开发的效率比很多编程语言都要高，开发应用逻辑也比大部分编程语言容易。使用 `nodejs` 编写 `web` 后端，可以得到具有很好的安全性、可用性和可扩展性的 `web` 应用。`nodejs` 的架构使得开发的工作量得以减少，在此同时性能又有所提升。

网页的前端中有大量 `js` 代码，当 `nodejs` 把 `js` 的使用带到后端时，就完成了前后端语言的统一。这大大激发了程序员们的热情。`nodejs` 社区发展日新月异，已经有很多第三方库在 `nodejs` 社区中开源。

`nodejs` 是一种非阻塞的语言。当进行存数据库或读取文件这样的需要同步控制的操作之时。`nodejs` 并不停下来等待结果的产生，而是将处理返回结果的函数放到一个事件响应器中，当操作完成时，通过事件唤醒这个函数进行后续处理。

事件驱动编程是一种以事件为纽带的编程模型。在事件模型中，推动程序向下运行的力量不是传统的顺行逻辑，而是事件。事件原本是一种消息机制。在观察者和发布者之间通过事件来完成通信。在 `nodejs` 中，各种实体都可以发送事件和响应事件。`node.js` 使用基于事件驱动的编程方式，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，称为一个周期。每个周期的

循环过程是一个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，则取出事件进行处理，并且在处理事件的过程中不会再响应其它事件。执行完成后，如果事件有相应的回调函数，则取出回调函数执行。当前任务队列中的所有任务处理完毕后进入下一个循环。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。

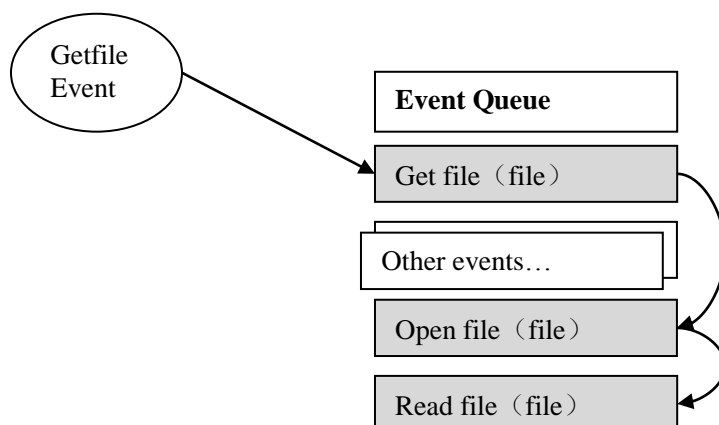


图 2-1 node.js 事件模型

如图 2-1 所示的是 nodejs 事件模型中处理 `getfile` 指令的事件模型。nodejs 程序收到 `getfile` 命令后开始执行，但并不等待读取完毕，而是直接在事件队列中提取下一个事件的代码来执行。`getfile` 命令的读取完成后，在事件队列中插入一个事件等待主程序处理。程序的流程由事件驱动。

nodejs 通过 `libuv` 模块进行事件处理，处理器每次处理事件队列中的下一条事件，在运行时动态添加新事件。在当前任务处理完成后，将 `callback` 中的代码加入事件队列末尾，形成了事件驱动的处理模型。事件随时可能到来，程序不会知道下一个事件会在何时到达。异步并发，是解决这个问题的关键。事件的处理是不能像传统的方式一样同步进行的，因为如果像传统方法一样同步处理事件就会面临复杂的同步问题，需要大量的加锁解锁操作，还要处理争夺锁时可能发生的死锁问题。有极大的概率堆积很多的没有来得及响应的事件请求。因此 nodejs 被设计为单线程结构，采用异步并发模式来处理 io。

用户行为分析系统面对的任务是 i/o 密集型的，使用的信息处理方式属于异步 i/o 模式。nodejs 作为一个基于异步 i/o 的编程语言是开发系统的首选。在 i/o 数量大、单次操作计算量低的情况下，nodejs 的单线程事件循环模式取得了非常优异的效果。而在用户行为分析流程中，各项数据分析以流水线的形式分层次进行，总体分析目标被分为许多小目标的组合，节点之间有着一对多的上下级关系。数据

频繁交互但每一次需要的计算量并不大，正是一种 i/o 数量大但单次计算量低的模型，非常适合使用 nodejs。

## 2.2 redis 数据库和 mongo 数据库技术研究

本文所构建的流计算实时用户行为分析系统使用了两种数据库。这两种数据库分别负责存储数据分析的中间结果，和持久化最终分析结果。它们是 redis 数据库和 mongo 数据库。

### (1) redis 数据库

redis 是一个开源数据库系统。它的基本存储单位是键值。在数据库中，数据的存储形式有词、列表、哈希表、集合、有序集合等几种形式。redis 中数据全部存储在内存中。数据以几种可选的方案备份在硬盘上的一个 dump 文件上。数据存入和取出不直接经过硬盘，它的访问速度相当快。

redis 数据库在内存中组织数据通过一系列对象进行，其中直接表示表的对象是 redisobject 对象。一个数据体的全部信息都记录在 redisobject 对象中。包括数据类型、存储空间、底层编码方式。在 redis 中的不同数据类型（哈希、列表、集合...）都用 redisobject 对象来表示。通过其中的 type 字段标明具体的数据结构的不同。相同的数据结构也可能用不同的编码方法来进行编码。redisobject 中的 edcoding 字段标明编码方式的不同。其中同类的表也可以使用不同的编码，这是为了更有效率的利用存储空间。以哈希表为例，在哈希表的表项数量比较少的时候，使用哈希结构其实是很浪费内存空间的做法。因为哈希表按照关键字的值计算得到内存地址，这种特性使哈希表在访问速度快的同时，实际占用的空间是大于其中的数据大小的。在数据项少的时候，这个比值相当悬殊。而 redis 是一个内存数据库，它的所有数据在运行期都是存储在内存中的。只有少量数据项的哈希表存储在 redis 中是对宝贵内存资源的极大浪费。所以 redisobject 对象中有 encoding 字段来标明底层的编码方法。当哈希表的数据项很少或数据长度比较低时。实际使用 redis\_encoding\_ziplist（顺序表）进行编码。而这个时候，从外界调用哈希表仍然是通过键值的方式，在调用上没有区别。而此时的顺序表比较短，实际存取效率（查询复杂度）也很优秀。当数据项数量增加超过一个值，或数据长度超过一个值的时候，哈希表的编码方式转变成实际哈希结构。这两个值可以自由设定，可以通过 redis 提供的控制命令来修改，也可以直接修改 redis 的 conf 参数控制文件。两种修改方法的效果并无区别。

redis 数据库对内存的使用是通过 `zmalloc.h` 和 `zmalloc.c` 源文件提供的函数来操作的,本质是通过重写 c 语言的 `malloc`, `realloc` 函数,效率和直接调用 c 语言的内存分配函数并无明显区别,只是增加了方便 redis 调用的接口。

redis 主程序是单线程的,它基于事件进行异步处理。在启动 redis 的时候,首先判断编译器的环境变量,根据运行环境的不同,使用不同事件实现方法,会依次判读 `evport`、`epoll`、`kqueue`、`select` 模块是否存在于当前环境。当判断到某一支持模块存在时就使用它,不存在时就进入下一级判断。

redis 支持事物处理。在 redis 提供的命令中,有 `multi`、`exec`、`watch`、`unwatch`、`discard` 这五个命令,它们提供事务控制。可以通过 `multi` 命令向一个事物中添加操作,然后通过 `exec` 命令执行事物中累计的指令。通过事物方法执行的一组命令有两种可能,一是全部正常执行,二是全部不执行。redis 的事物支持使它虽然是单线程,异步处理,但仍然可以胜任复杂的同步任务。redis 数据库支持各种数据结构类型,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 在存储数据的时候也使用的相应的技术来提高检索效率。比如 `zset` 有序集合的底层使用了跳跃表来实现。这种数据结构非常适合 redis 这种内存数据库,在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,这种跳跃表数据结构可以在不明显损失性能的情况下完成任务。对跳表这种数据结构的修改是局部性的,与红黑树相比,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 支持各类数据类型,使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,但是也因为基于内存,内存空间的限制成了它应用的瓶颈。为了节约内存,通常 redis 被用作中间结果的临时存储数据库,而不是大量数据的持久存储数据库。

可以进行消息的发布订阅也是 redis 的一大特点。通过 `publish`、`subscribe` 等命令,redis 可以提供实时消息队列。`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,让系统具有更好的可扩展性<sup>[11]</sup>。redis 的 `pub/sub` 机制使用简单。并且可以在各类语言中使用。redis 的 `pub/sub` 机制实现的过程中,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然后进行订阅或者发布操作。一个客户端既可以是订阅端也能是发布端。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，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。

## （2）mongo 数据库

mongo 数据库是一种非关系型数据库。mongo 数据库支持分布式存储<sup>[12]</sup>。可以在任意一台计算机上连接 mongo 存取数据。在 mongo 数据库中，数据以集合为存储单位，可以直接存储 json 格式的数据。集合的概念和关系型数据库中的表很相似。集合可以看做一组文档。集合中的文档可以是任何数据类型。集合没有模式，不同模式的文档都可以放在同一个 mongo 集合中，即使它们的键和值的类型都不同。mongodb 的设计目标是易部署、高性能、易使用、可扩展。mongo 的主要功能特性如下：

（a）mongo 数据库存储的单位是集合。对象类型的数据可以不经过转化就直接存储在 mongo 数据库中。相异类型的许多个文档也可以同时存储在相同的 mongo 集合中。

（b）采用无模式结构存储。mongo 是模式自由的，无模式的文档是在 mongodb 中集合中存储的数据，集合区别于 rdbms 中的表的特点就是采用无模式结构存储数据<sup>[13]</sup>。

（c）完全索引支持。mongo 可以在包含内部对象的任意属性上建立索引，查询的速度会得到提高。rdbms 的索引可以建立在内部对象、指定属性上面，mongodb 的索引具有类似的能力。mongodb 还提供创建基于地理空间索引的能力<sup>[14]</sup>。

（d）mongo 支持的查询操作很丰富。sql 中的大部分查询 mongo 几乎都支持。

（e）强大的聚合工具。mongo 不仅提供丰富的查询功能，还提供强大的聚合工具，如 count、group 等，复杂的聚合任务能够使用 mapreduce 模式完成。

（f）支持数据恢复和复制<sup>[15]</sup>。mongo 支持主从复制机制，可以实现读扩展、数据备份、故障恢复等功能。而基于副本集的复制机制确保了集群的原始数据不会丢失。mongo 提供自动故障恢复的功能。

（g）包括大型对象（如视频）在内都使用高效的二进制格式存储。mongo 可以保存任何类型的数据对象。不同的对象在 mongo 眼中都是二进制码。

（h）自动处理分片。mongodb 可以让集群自动把数据分成更小的分片。支持云计算层次的扩展。对数据进行分片可以使 mongo 能够负担的任务规模变得更大，也能保证存储的负载均衡。使集群存储更多的数据。

（i）支持 perl、javascript、ruby、php、java、c#、c 和 c++ 语言的驱动程序，开发人员使用任何一种主流开发语言都可以方便的使用 api 访问 mongo。

(j) 文件存储格式为 `bson` (`json` 的一种扩展)。`bson` 支持文档和数组的嵌套。它是二进制格式的 `json` 的简称。

(k) 可以通过网络访问。`mongodb` 数据库可以通过网络在远地存储数据和取得数据。

## 2.3 本章小结

本章介绍了论文涉及的几种关键技术。

`node.js` 是一个基于 `chrome v8` 引擎的 `javascript` 运行环境。`node.js` 使用了事件驱动、非阻塞式 `i/o` 的模型。适用于 `i/o` 密集型任务。

`redis` 是一种内存数据库，它是一个 `key-value` 存储系统。`redis` 支持存储的数据类型包括 `string`(字符串)、`list`(链表)、`set`(集合)、`zset`(sorted set --有序集合)和 `hash` (哈希类型)。数据全部保存在内存中，`redis` 存取十分高效。

`mongo` 数据库是基于分布式文件存储的数据库。是非关系型数据库，支持各种索引和查询操作。数据以集合为单位，以 `json` 形式组织。`mongo` 支持网络存取和数据恢复。



## 第三章 流计算模块设计

流计算模块用来部署用户行为分析节点。论文第三章阐述系统流计算模块的需求分析、设计方法和功能组件。将通过模块图、类图、流程图进行阐述。

### 3.1 系统功能需求分析

流数据分析系统，分为节点部署管理程序和各类计算模块(计算分析节点)，以及辅助库组成。基本框架由控制程序(master、daemon、client)组成，负责各类计算模块的加载。计算模块是包含业务逻辑的代码组成，例如一组 key 值的分组、求和、求均值，或者更为复杂的计算代码。辅助模块提供计算模块所需的工具类库，例如数据单位的转换、特定关键词的检查等。系统的设计目标包括：

1. 低延迟，数据需要实时分析并将结果实时反应到显示系统。
2. 高性能，数据处理速度要高于到达速度，防止数据累积。
3. 可扩展，数据量、计算量会随着业务的发展越来越大，系统需要具有良好的扩展性。
4. 分布式，系统组件需要能分布在不同的机器上。
5. 容错，某一个单独节点崩溃不能影响其它节点的正常工作。

### 3.2 系统总体设计

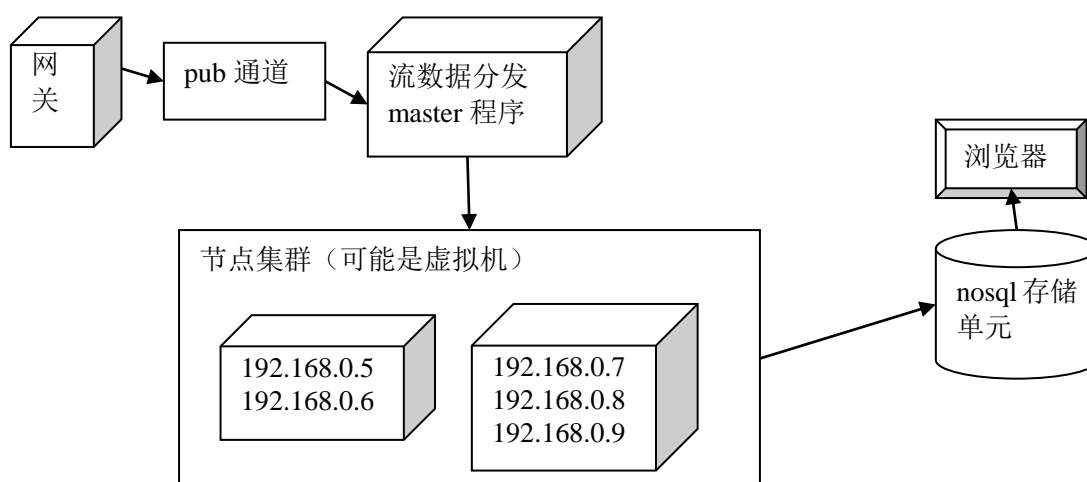


图 3-1 系统架构图

系统架构图如图 3-1。系统的主要组件包括网关上的抓包程序、计算节点资源管理程序 master 程序、计算节点运行程序 client 程序、运算环境管理程序 daemon

程序。负责数据显示的模块包括 web 网站、redis 数据库、mongo 数据库。不同功能模块可以部署在不同的机器环境上，可以是真实的物理机器，也可以是虚拟机。

流计算节点管理程序的主要功能包括了数据分析节点的管理和节点间的数据转发。数据收发的接口封装在 vrsscore.js 和 vrssstream.js 两个文件中。它们分别封装了节点之间的消息接口和节点管理的接口。

整体系统的模块层次结构如图 3-2 所示。流数据子系统完成数据分析节点部署和运行的任务。该子系统的源数据来自网关，数据出口通向三个方向：redis 数据库、mongo 数据库和前端数据显示模块。

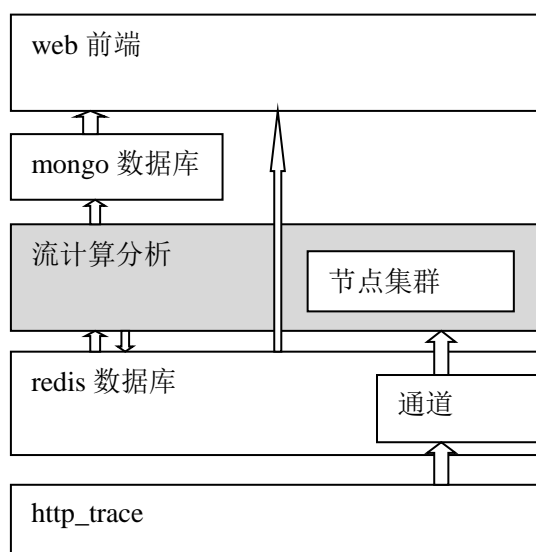


图 3-2 系统模块层次结构图

### 3.3 节点管理程序设计

部署流计算节点的任务由 3 个程序协作完成，这三个程序命名为 master、daemon、client。它们可以分布式的部署。daemon 和 master 是多对一的关系，多个 daemon 程序可以部署在不同的计算机上。master 和 client 是一对多关系。

master 程序是节点管理的接口和界面，从 master 程序中，可以查看其它模块的分布和运行情况。

client 程序是节点代码提交的接口，新的数据分析节点通过 client 程序提交。client 程序和 master 程序通信。

daemon 程序是系统资源的管理者，管理实际运行空间。daemon 程序和 master 程序通信并将可用资源加入 master 中的列表。

三个程序的关系如图 3-3。

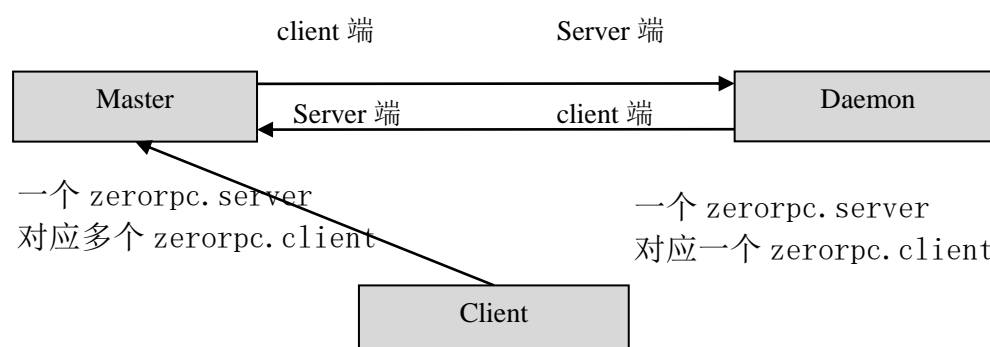


图 3-3 计算节点部署程序

数据分析节点的运行环境由 daemon 程序维持。每一个 daemon 程序实体维护一个单独的运行环境。daemon 程序接收数据分析节点 js 代码后，将代码嵌入本地环境运行。daemon 程序的分布式部署，得到了高效的资源利用率，和可随时增加的空间资源，同时分散了风险。当发现已有的资源不足以运行新的计算节点时，可以在任意一台联网的机器上分配出一部分计算资源，然后通过运行新的 daemon 程序将可用的空间资源注册到 master 所维护的 daemon 程序列表上。可以动态的增加运行空间。这赋予了系统很好的抗压能力、扩充能力，这种能力使系统即使在已经工作的状态下，也可以随时在规模上做出改变，在部署上进行调整。

多运行空间的设计，另一个重要原因是为了分散系统崩溃的风险。noejs 是一个单线程的环境，某一个节点出错，将导致整个系统崩溃。但是如果使用分布式的方法，将不同节点分不到不同的计算机上，或同一台计算机的多个 nodejs 环境，则可大大降低风险。由于 master 和 client 并不实际运行数据分析代码，系统主体部分又经过仔细的理论验证和调试，出错的概率极低。同时使用 try, catch 方法将关键代码可能的错误输出到日志文件中，可保证系统主体的可用性，使系统的 master 程序不会轻易崩溃。在任意时刻，可以通过向 master 进程发送命令来查看系统的 daemon 列表，可以查看每一台运行 daemon 的机器所提供的 ip 地址和端口号。也可以查看已经运行的计算节点的名字。

三个程序的通信是使用建立在 zeromq 开源技术之上的。程序中加入 zeromq 组件完成通信任务。zeromq 的三个基本的通信模型分别是“request-reply”，“publisher-subscriber”，“parallel pipeline”，“rpc (remote procedure call protocol)”<sup>[16]</sup>。三个程序使用远程过程调用协议 rpc 通信，它是一种通过网络从远程计算机程序上请求服务而不需要了解底层网络技术的协议。程序交互使用远程过程调用方法，使 master、daemon、client 程序编写时不需要考虑 tcp/ip 协议交互的细节，直接调

用远地的函数，大大减轻了编码量，增加了可靠性。程序通信部分由开源技术支持，设计这三个程序的重心是数据分析节点的管理。

在 rpc 通信的 server 中，程序通信部分声明一个 zerorpc 实例：

```
var server = new zerorpc.server({
  connect: {函数体}
  submit: {函数体}
  start: {函数体}
  stop: {函数体}
  status: {函数体}
});
```

在上面的代码中，实例化了一个名叫 server 的 zerorpc 对象。在这个函数的初始化参数中传入了一个 json 对象，这个 json 对象按照键值的方式存储函数名和函数体，通过这种方法，将初始化列表中的 json 对象中的函数加入 server 对象中。完成了 server 的初始化。在 client 方面，通过如下代码声明实例：

```
var client = new zerorpc.client();
client.connect(master_host);
client.invoke("report", ..., function(error, ...))
```

在 client 中 invoke 相应的函数并赋予独有的函数名后，在 master 实例中即可向调用本地函数一样调用 client 中的函数，隐藏了网络交互中的底层技术细节，保证可靠的网络交互。

三个程序调用的其它 nodejs 开源模块有 stdio 和 commander 两个。其中 stdio.getopt 模块用于提取命令行参数,生成 help 菜单；commander 模块用于提取命令,建立选择分支。这部分程序功能由开源技术支持完成。

良好的节点部署流程和节点部署模块的设计，保证了系统的高效稳定。系统支持分布式部署并可以随时增加节点和运行资源。

### 3.3.1 master 程序设计

master 程序接受 daemon 程序的注册，维护一个 daemon 程序列表。master 程序接收 client 程序的节点代码上传，为每一个分析节点分配 daemon 机器，节点代码在分配的 daemon 进程中实际运行。master 程序是流计算系统的总控制台，负责维护各种资源的列表，沟通不同的机器。daemon 和 client 程序都只需要记住 master 程序所在的 ip 地址和端口号，和 master 进程进行沟通。

master 程序的初始参数有 3 个，通过 getopt 进行控制，关键代码如下：

```

var ops = stdio.getopt({
  'port': {key: 'p', args: 1, description: 'listen port'},
  'host': {key: 'h', args: 1, description: 'listen host or ip'},
  'help': {args: 1, description: 'print help info'}
});
var nodes_client = {}; //维护 client 程序的集合
function init(ops){
  var host="0.0.0.0";
  var port = default_port+"";
  if (ops['host'])host=ops['host'];
  if(isnan(ops['port'])||parseInt(ops['port'])>65535||parseInt(ops['port'])<1024){
    port = default_port+"";
  }else{
    port=ops['port']
  }
  return "tcp://" + host + ":" + port
}

```

这一段代码控制参数的初始化。在启动 master 程序的时候，通过 port 和 host 参数指定 master 程序将要绑定的 ip 地址和端口号，master 程序绑定地址之后，就会在这个地址监听来自 client 和 daemon 程序的请求。如果在开启程序的时候没有指定需要绑定的 ip 地址和端口号或指定的地址超出正常的 ip 地址和端口号范围，就会监测到非法参数，这时就将启动时的默认地址改为在本机的一个写在配置文件中的默认 ip 和端口。默认 ip 和端口通过设置文件中的 default\_host、default\_port 参数进行设置。在 nodes\_client 变量中维护了连接到 master 的 client 的列表信息。

下面这段代码是 master 程序构建 rpc 函数的主体结构。构建了六个 rpc 函数供 daemon 程序和 client 程序调用。

```

var endpoint =init(ops);
var zerorpc = require("zerorpc");
var server = new zerorpc.server({
  connect: function(node_id,endpoint,process_id, reply) {.....
  },
  submit: function (node_id,app_id ,jsfile, reply){.....
  },

```

```

start:function(node_id,app_id,reply){ .....
},
stop:function(node_id,app_id,reply){ .....
},
status: function (reply){.....
},
report: function (node_id,app_id ,reply){ .....
},
});
server.bind(endpoint);

```

connect 函数提供给 daemon 程序调用，其它函数提供给 client 程序调用，client 程序的调用命令和参数在 3.3.3 小节详述。daemon 程序和 client 程序的函数框架类似 master 程序。

### 3.3.2 daemon 程序设计

daemon 程序提供本机上的 nodejs 运行环境，实际运行代码。

daemon 程序向 master 程序注册，将本机 ip 和 port 加入 master 中维护的机器资源列表。任意一台机器都能运行 daemon 程序，向 master 注册一个可用的运行环境。

daemon 程序的初始化参数如下面代码：

```

var ops = stdio.getopt({
  'port': {key: 'p', args: 1,description: 'listen port'},
  'host': {key: 'h', args: 1, description: 'listen host or ip'},
  'nodeid': {args: 1,mandatory: true,description: 'node id'},
  'master': {args: 1,mandatory: true,description: 'master ip:port'},
});

```

代码声明了 4 个初始参数，host 和 port 是本机绑定的 ip 和端口，可以在程序启动时指定，默认参数由 default\_port 和 default\_host 指定。还有两个参数 nodeid 和 master 是强制参数，在启动 daemon 程序时必须进行设置。它们是 daemon 程序向 master 程序注册的机器名和 master 程序的地址。

### 3.3.3 client 程序设计

client 程序和 master 程序沟通，注册用户。完成注册后，每次通过命令向 master

程序提交一段可以运行的 `nodejs` 程序代码（流数据分析节点）。

`client` 程序可以在任意机器上运行并提交节点代码。通过 `client` 提交的计算节点代码中必须继承和实现 `vrssstream` 和 `vrsscore` 接口。这是代码接受流计算平台数据的接口，也是将数据传到其它分析节点的接口。节点之间通过统一规范的接口完成流数据的传递。

`client` 程序的初始化需要提供本机需要绑定的地址。在启动后，有 4 个命令可以控制 `client` 程序，命令格式和功能如表 3-1、表 3-2、表 3-3、表 3-4、表 3-5。

表 3-1 程序 `client` 中 `status` 命令说明

命令名	<code>status</code>
命令格式	<code>status nodeid all</code>
功能描述	查看节点状态
示例	<code>\$ cli status node01</code> <code>\$ cli status all</code>

表 3-2 程序 `client` 中 `submit` 命令说明

命令名	<code>submit</code>
命令格式	<code>submit &lt;node_id&gt; &lt;app_id&gt; &lt;app&gt;</code>
功能描述	提交功能节点
示例	<code>\$ cli submit node01 app01 app.js</code>

表 3-3 程序 `client` 中 `start` 命令说明

命令名	<code>start</code>
命令格式	<code>start &lt;node_id&gt; &lt;app_id&gt;</code>
功能描述	启动指定节点<app_id>
示例	<code>\$ cli start node01 app01</code>

表 3-4 程序 `client` 中 `stop` 命令说明

命令名	<code>stop</code>
命令格式	<code>stop &lt;node_id&gt; &lt;app_id&gt;</code>
功能描述	停止指定节点<app_id>
示例	<code>\$ cli stop node01 app01</code>

表 3-5 程序 client 中 report 命令说明

命令名	report
命令格式	report <node_id> <app_id>
功能描述	查看指定节点<app_id>状态
示例	\$ cli report node01 app01

### 3.4 系统工作流程

一个 master 可以对应多个 daemon 程序。master 分发工作,daemon 执行工作,client 提交工作。以第一个分析节点的启动和部署为例,列出三个程序的启动和工作顺序:

- 1.master 启动, 建立 zerorpc.server 对象, 建立 node[]、node\_client[]列表。
- 2.daemon 连接 master, 调用函数 connect(nodeid,endpoint,pid), 在 master 资源列表中注册 daemon。
- 3.client 连接 master, 调用 master 的函数 node[nodeid] 和 node\_client[nodeid], 在 master 用户列表中注册 client。
- 4.client 提交节点代码, 调用函数 submit(nodeid,app,file),将节点代码给 master。
- 5.master 提交代码到 daemon 程序, 调用函数 nodeid.submit(app,file)。
- 6.daemon 建立独立运行环境, 调用函数 connect.fork.cryo、submit.start 创建运行环境, 在子环境中运行节点代码, 节点开始工作。

### 3.5 nodejs 多核 cpu 利用方法

nodejs 是单线程的, 虽然系统的分布式设计使不同节点可以部署在不同机器环境上。但如果单个环境上不能利用多核 cpu 的性能也会造成浪费。引用 nodejs 开源组件 clauster 利用多核 cpu 的性能<sup>[17]</sup>。

nodejs 的异步非阻塞模型, 在 io 密集的应用中能够获得非常突出的性能。但是在计算量密集的应用中效果就要差一些。在多核 cpu 中如果可以利用多个处理器的计算能力, 会大大弥补 nodejs 应用的计算力遗憾。引用 child\_process 开源模块, 这个模块可以让开发者同时启动多个线程, 而让每个线程占用一个处理机, 实现了多核的利用。

使用 clauster 的代码结构如下:

```
var cluster = require('cluster');
var http = require('http');
```



```

var numcpus = require('os').cpus().length;
if (cluster.ismaster) {    //主线程运行的代码
    require('os').cpus().foreach(function(){
        cluster.fork();
    });
} else {
    //子线程运行的代码
}

```

使用 `cluster` 模块时，首先声明一个 `cluster` 实例，然后根据该实例的方法 `ismaster()` 判断目前的代码是在主线程运行还是在子线程运行。然后在主线程中进行下一个子线程的创建和子线程运行完毕后的后续处理工作。而在子线程分支中加入子线程需要运行的代码。

在系统运行中。某些节点需要同时开启多个。这时会利用到 `cluster` 模块。多个相同的节点同时开启时，消息不再通过通道进入分析节点。而是通过队列，相同的节点监听同一个队列，每条消息只取出一次，避免同一条消息的多次处理。

### 3.6 vrssstream 和 vrsscore 消息分发组件

`vrssstream.js` 和 `vrsscore.js` 是两个 js 文件。它们是分析节点代码必须包含的库文件。它们包含分析节点代码传递数据的接口。

`vrssstream` 对象是数据流的抽象，内置一组控制字符，包含对 `redis` 对象所有可能的操作的控制。在引用 `vrssstream` 文件后，调用 `initsub` 方法来关联一个 `redis` 通道，同时传入启动参数，然后就可以通过实例化的 `vrssstream` 对象操作数据流。

`vrsscore` 模块的作用是建立流计算处理逻辑，建立 `vrsscore` 对象后，可以通过内置的原型方法向对象中添加节点 `nodejs` 代码，代码以字符串的形式存储在 `vrsscore` 对象的一个子对象中。开始运行的数据分析节点代码称为 `tuple`。

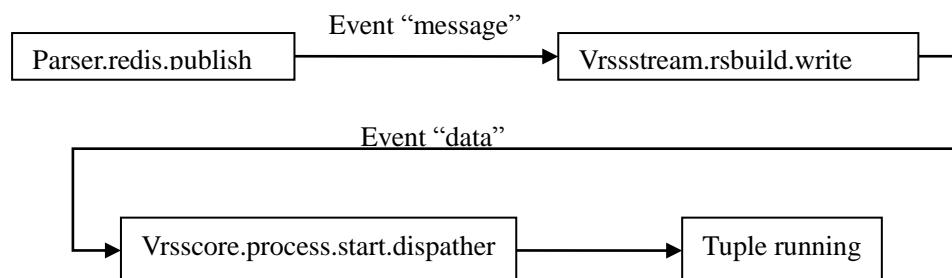


图 3-4 tuple 建立过程

图 3-4 是分析节点代码传入流计算系统到 tuple 建立的过程示意图。vrssstream 对象建立后, 开始对”message”事件的监听。vrsscore 对象建立后, 开始对”data”事件的监听。它们基于事件形成程序处理的流程。这不是一个顺序的处理流程, 示意图中所示的是一个具体 tuple 的建立过程。在多节点具体运行时, 后来的节点可能先于前一个节点建立完毕。每一个部分都只复责对应的事件进行响应处理。

在 vrsscore 中, 建立 process 对象来处理计算节点的加载和运行。process 的初始环境变量设置形式如下。

```
var process = function(){
  this.env={ };
  this.env['tuple']  = { };
  this.env['stream'] = { };
  this.env['status'] = "stop";
  this.env['result'] = { };
  this.env['dispathter'] = new vrsseventhandler();
}
```

precess 维护的 tuple 子对象是计算节点代码的存储空间。可以随时通过 createtuple 方法在 process.env.tuple<sup>[18]</sup>对象中添加新的 tuple。

添加新节点代码的函数如下。新的计算节点代码以函数的形式封装在 fn 对象中。在 createtuple 函数中, 接收 fn 函数, 将它放入 this.env['tuple'][id]变量中。等待启动。

```
process.prototype.createtuple = function (id ,fn){
  var that = this;
  var _fn = checkfn(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;
}

```

在 `vrssstream` 中，通过添加 `prototype` 函数，增加通道消息收发的函数。主要的函数原型有：

```

redisstream.prototype.init = function(id){
redisstream.prototype.initsub = function(channel,p){ }
redisstream.prototype.removesub = function(channel,fn){ }

```

这三个原型函数的功能依次是：初始化对象、增加新的监听通道、停止监听某通道。

在数据分析节点代码中，建立 `vrsscore` 对象后，在其中的 `process` 对象调用 `createtuple` 函数将分析代码封装在 `fn` 函数中，在 `fn` 函数中的节点代码调用 `vrssstream`，完成对流数据的收发。

### 3.7 系统的容错设计

对数据分析平台的容错需求包含两个方面，一是当节点失效时，系统的其余节点安全性保障。二是当系统崩溃时，数据的安全性保障。

为了保证单个节点的失效不会影响到整个系统，系统在设计时就把系统的管理主体和运行部分放在不同的环境中。系统的主体部分 `master` 运行在一个单独的装有 `nodejs` 环境中，在这个 `nodejs` 环境上不再运行其它的 `nodejs` 程序。这样做可以保障系统管理的主体不会因为其它程序的影响导致崩溃。

数据安全性的保障有两个机制。一是定时将内存中的数据备份在硬盘。这样即使 `redis` 崩溃，也可以保障数据分析的中间数据不会丢失。二是将分析结果持久化在 `mongo` 数据库。

`redis` 数据库自带将内存数据定时写入硬盘的功能。在 `redis` 文件夹下的 `redis.conf` 参数配置文件中可以设定硬盘写入频率。参数设置格式如下<sup>[19]</sup>。

```

save 900 1          #900 秒之内有 1 个 keys 发生变化时
save 300 10         #30 秒之内有 10 个 keys 发生变化时
save 60 10000       #60 秒之内有 10000 个 keys 发生变化时

```

数据实时写入硬盘需要检测内存中数据的变动<sup>[20]</sup>。检测的频率越高，安全性越好，但是硬盘读写开销和 `cpu` 使用率也会提高。如果每一个数据都实时写入硬盘，那么 `redis` 作为内存数据库的优势也就荡然无存了。所以在设置存入的频率时需要根据情况考虑好性能和安全性之间的平衡。

数据持久化任务由数据分析节点集群完成，分析得到的结果数据定时存入 mongo 数据库。在 4.11 节中详述。

### 3.8 流数据生命周期

数据在系统中的生命周期包含三个阶段，数据抓取阶段、数据分析阶段和数据展示阶段。数据抓取模块命名为 `http_trace`，抓取某一网关的数据，然后发送到 redis 数据库中的 `http_trace` 通道中。这部分数据传输使用 `publish/subscribe` 机制，数据实时发送不存储。没有收听者存在时，就会丢掉数据。当有多个收听者时，所有监听者收到同样的数据流。

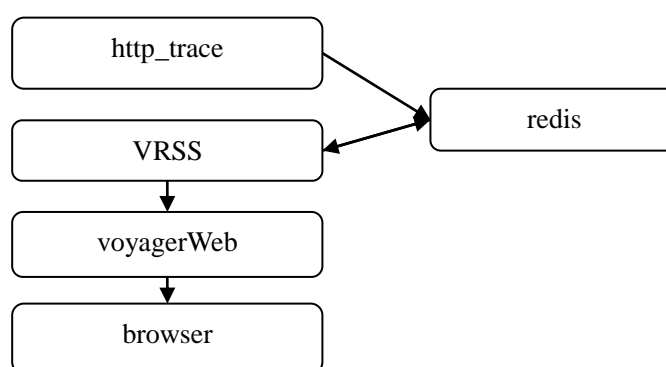
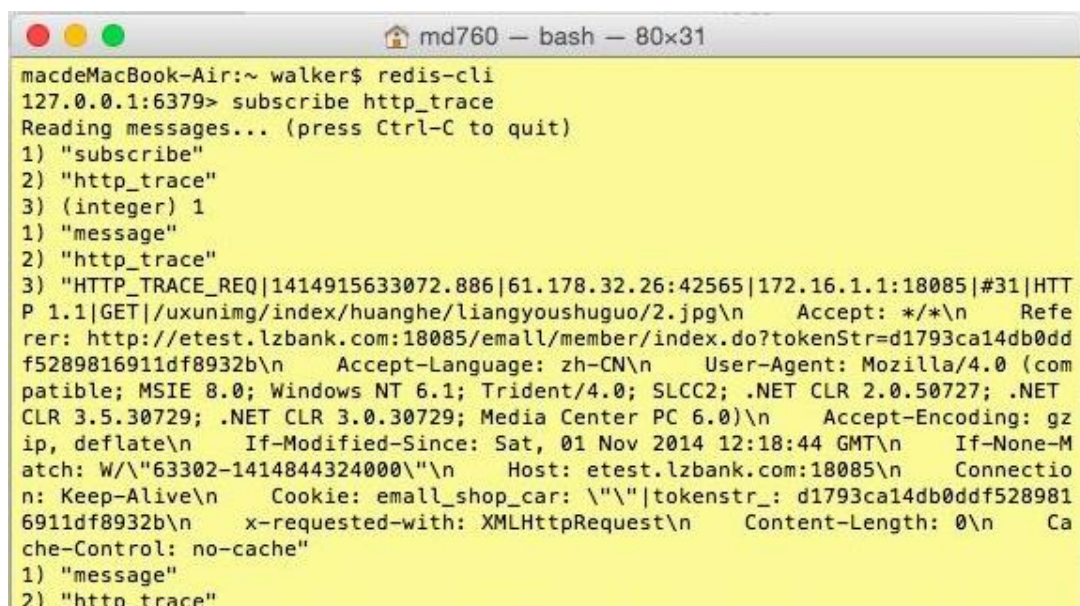


图 3-5 系统数据流程

系统的数据流程如上图 3-5 所示。在本系统中，有 `dayinfo.js`、`staytime.js`、`httperror.js`、`keywordsearch.js`、`averagedeep.js` 五个 nodejs 节点监听 `http_trace` 通道。这五个 node 节点基于流计算系统的 `vrsscore` 和 `vsstream` 模块。`vsstream` 模块提供一套操作 redis 数据库的接口，可以用 `vsstream` 的实例来进行 redis 中的 `key`、`hash`、`zset` 等等数据格式的操作，也可以在一个实例中声明多个 redis 变量，用以监听多个 redis 通道。`vrsscore` 实例可以和 `vsstream` 实例绑定，共同提供一套数据接收然后分片处理的机制。在这五个 nodejs 程序中，使用实时 `tuple` 分片数据流，在其回调函数中用处理 `http` 数据，取得相应的元素，进而分析出用户的 `pv`、`uv`、流量、来源网站、访问网站、停留时间、访问深度、使用浏览器类型、使用终端类型等信息，分析过程中产生的中间结果和最终结果都会以 `set`、`hash`、`zset` 这三种格式存放到 redis 数据库。

图 3-6 是 `subscribe` 原始数据通道的结果。数据流的抓取由网关上的抓包模块完成。抓取的 `http` 数据包发到 `http_trace` 通道中，由系统取用。在通道上可以查看原始的未经分析的数据。除每日清理 redis 数据的定时器模块外，其它模块只读取 redis 结果集的数据，并不对其产生修改。在数据分析节点集合中，有一段单独

的定时器节点代码，celarredis.js，在这段代码中，设置了两个定时器，在每天的 23:58 和 23:59 激发，它们进行数据持久 mongo 的操作和清理 redis 数据的操作。



```

macdeMacBook-Air:~ walker$ redis-cli
127.0.0.1:6379> subscribe http_trace
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "http_trace"
3) (integer) 1
1) "message"
2) "http_trace"
3) "HTTP_TRACE_REQ|1414915633072.886|61.178.32.26:42565|172.16.1.1:18085|#31|HTT
P 1.1|GET|/uxunimg/index/huanghe/liangyoushuguo/2.jpg\n  Accept: */*\n  Refe
rer: http://etest.lzbank.com:18085/emall/member/index.do?tokenStr=d1793ca14db0dd
f5289816911df8932b\n  Accept-Language: zh-CN\n  User-Agent: Mozilla/4.0 (com
patible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET
CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)\n  Accept-Encoding: gz
ip, deflate\n  If-Modified-Since: Sat, 01 Nov 2014 12:18:44 GMT\n  If-None-M
atch: W/\\"63302-1414844324000\\" \n  Host: etest.lzbank.com:18085\n  Connectio
n: Keep-Alive\n  Cookie: emall_shop_car: \\"\\\"|tokenstr_: d1793ca14db0ddf528981
6911df8932b\n  x-requested-with: XMLHttpRequest\n  Content-Length: 0\n  Ca
che-Control: no-cache"
1) "message"
2) "http_trace"
  
```

图 3-6 原始数据进入系统

流数据分析模块和后续的结果显示模块的沟通通过 socketio 开源组件完成<sup>[21]</sup>。在前端接收 socketio 连接请求，监听 50020、50021、50023 等端口，每当有 socketio 请求到来时，就会查询 redis 数据，返回一条消息，消息内包含处理过的格式化信息。前端接收到信息后，就可以进行可视化展示。显示系统数据的直接来源，是 reids 数据库。

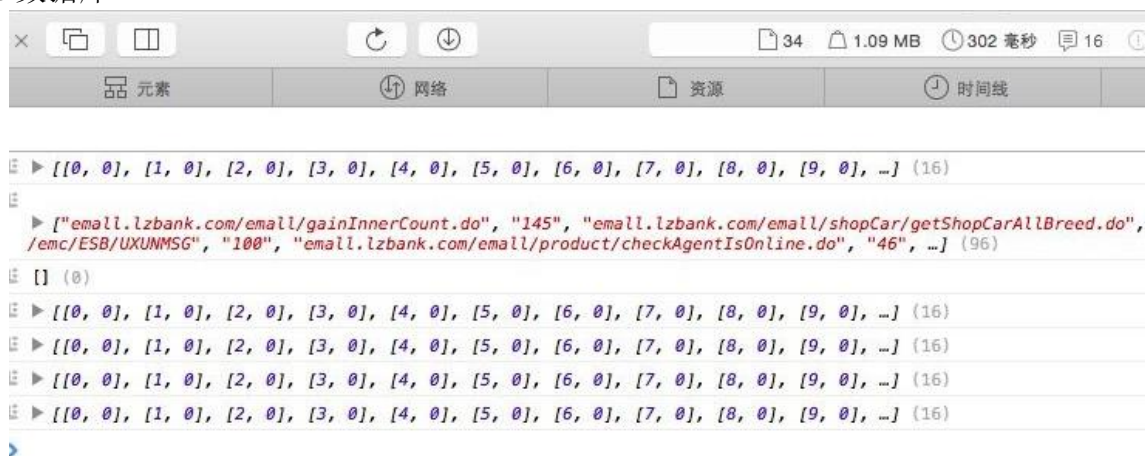


图 3-7 显示模块 socketio 数据接收

在数据显示模块的浏览器调试中，数据在前端测试接收的 socketio 数据如图

3-7。图中的数据是来源网站和当日访问量的 `socketio` 数据。来源网站的数据类型是混合数组，其中单数序号的项是来源网站链接的字符串形式，偶数序号的项是对应的计数。

当日访问量的数据类型是数组，数组的项是二元数组，每一项子数组的两个项的数字分别表示当日的时刻和该时刻的访问量计数。

### 3.9 本章小结

本章介绍了系统的流计算子系统，这是一个基于开源技术 `redis`、`nodejs` 技术构建的流计算系统。

系统编写了 `vrssstream.js` 和 `vrsscore.js` 两个库文件作为事件注册和响应的接口。`vrssstream.js` 负责连通指定的 `redis` 数据库，接收并转发消息。`vrsscore.js` 负责数据分析节点代码的加载，当新的分析节点代码进入系统时，调用 `vrsscroe.js` 提供的函数将代码部署到执行环境。

`master`、`daemon`、`client` 三个程序管理流数据分析节点和运行资源。初次搭建工作环境时，需配置 `master` 中的环境变量，绑定 `ip` 地址和端口。后续增加数据分析节点时，在任意一台计算机上，通过 `client` 上传节点代码，然后通过 `master` 程序将节点代码分发到 `daemon` 程序管理的运行环境中，启动节点开始工作。

程序中使用开源技术 `zerorpc` 支持程序间通信。使用开源技术 `child_clauster` 支持多核 `cpu` 利用。

## 第四章 用户行为分析节点设计

用户行为分析节点集群完成从 http 数据流到用户行为数据（pv/uv、停留时间、访问深度、经纬度等）的实时分析转化功能。

### 4.1 节点集群功能需求分析

用户行为数据分析节点承担着从 http 流数据中分析出具体的用户行为数据的任务。节点集群需要有这些能力：接受 http 流数据、分析数据、实时存取 redis 数据库、定时持久化数据。

需要分析的结果包括：

1. 用户的访问量。
2. 新用户数。
3. 用户来源链接。
4. 用户访问最多的页面。
5. 用户搜索的关键词。
6. 用户的地理位置分布。
7. 用户使用的浏览器型号。
8. 用户使用的设备。
9. 用户访问深度。
10. 用户停留时间。

这些分析节点在性能上的要求必须做到实时性、准确性、一致性、关联性<sup>[22]</sup>。实时性是流数据分析的必要条件。及时处理数据才能防止数据堆积和结果可用。准确性是系统功能正常的基础，只用提供准确的数据分析结果系统才有存在的意义。一致性和关联性是指不同项目的分析结果之间具有内在的联系。分析结果如果不符合客观事物的规律，不同数值之间没有合理的比例关系，则分析出的结果也会失去价值，同时因为不同算法和标准的使用，结果中的正常比值范围会有一些差异，需要根据情况考虑。

### 4.2 节点集群总体结构设计

数据分析节点搭载在论文第三章描述的流计算系统上。数据在不同模块之间的流动通过 redis 的 publish/subscribe 通道完成。位于网关的抓包模块抓取原始的报文信息，然后把数据发布到数据分析系统的 redis 上。数据分析平台接收数据，

将数据交给数据分析节点进行分析。在节点之间的数据沟通通过 nodejs 的 emit/on 事件机制进行。

在设计用户行为分析节点群时，最多达到 11 个节点，在后期优化时，将其中的一部分节点整合，一共得到 7 个节点。这 7 个节点中包含两个功能节点和 5 个数据分析节点，五个分析节点负责用户行为分析，一个功能节点负责与 mongo 数据库沟通，另一个功能节点作为定时器，发送计时信息。本章主要讲述 5 个数据分析节点，这些节点命名为 dayinfo、staytime、averdeep、kwywordsearch、httperror。它们完成对用户的访问量、新用户数、用户来源链接、用户访问最多的页面、用户搜索的关键词、用户的地理位置分布、用户使用的浏览器型号、用户使用的设备、用户访问深度、用户停留时间这 10 个目标的分析。在 dayinfo 节点中，整合了 pv、uv 分析，在 pv 分析的基础之上，通过一些字符串对比，确定 uv 是否增加。

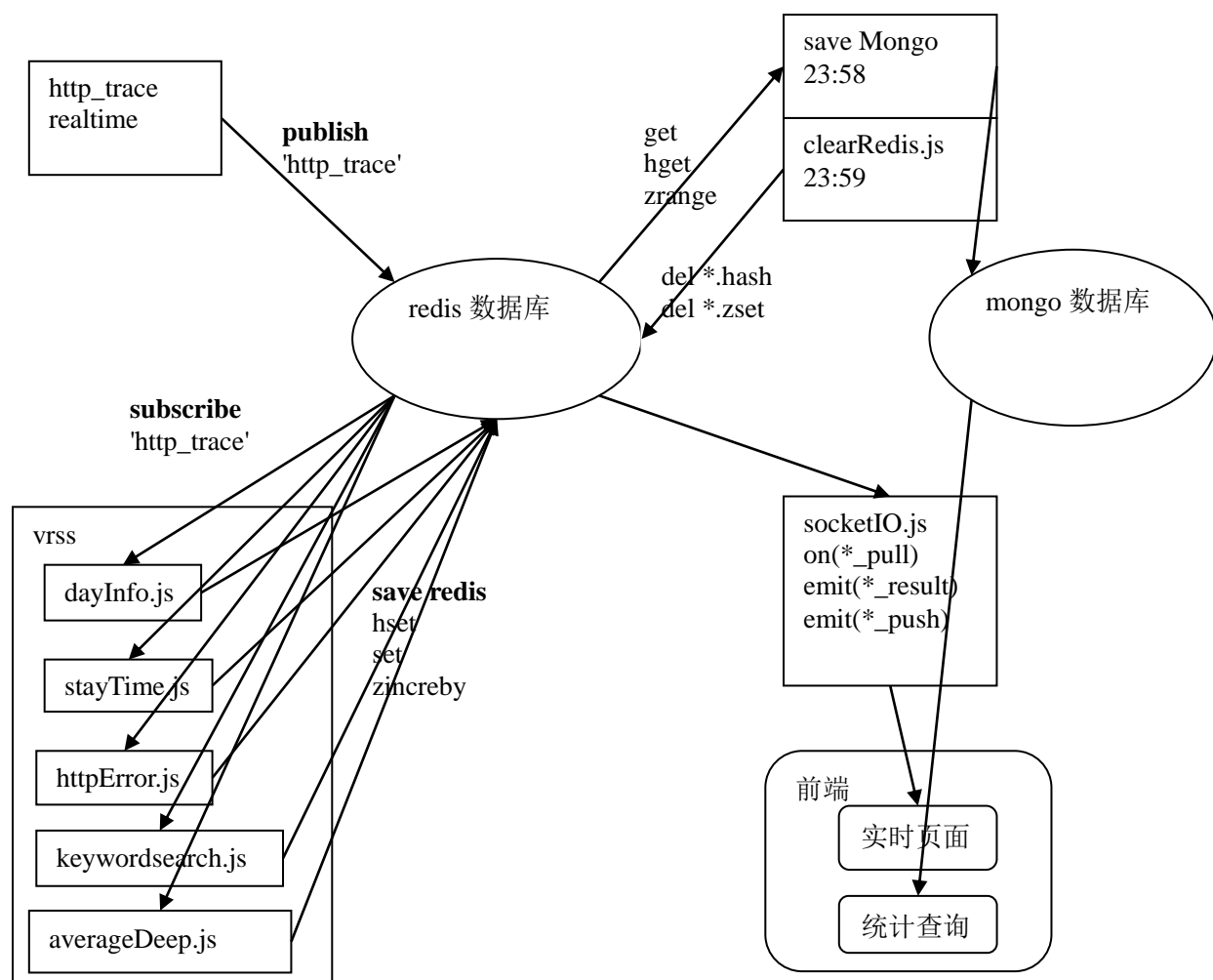


图 4-1 数据分析节点结构



节点总体结构如图 4-1。从中可以看到数据分析节点只和 redis 交互关系。redis 是连接系统不同组件的纽带。原始 http 数据从 http\_trace 模块获取，该模块从网关得到所有通过网关的 http 报文，将这些报文转发到 http\_trace 通道。

流数据分析系统通过 daemon 启动时传入的参数，监听指定的 ip 和端口上的 http\_trace 通道，将数据初步过滤后，以事件的形式传入各个数据分析节点的事件监听器。数据分析节点得到数据后，进行多级分析，产生的结果存入 redis 数据库。

功能节点含有两个特殊的定时器节点，一个每隔 2s 读取指定的表和集合，将数据格式化后通过 socketio 接口，将结果数据推送到前端页面，使页面实时更新。另一个定时将数据存入 mongo 数据库，进行数据持久化，同时每天清空 redis 数据库，确保 redis 中的数据只是当天的数据，不会累加到下一天。

用户行为分析系统本身是一个实时的流数据分析系统，在 redis 当中的数据不会超过一天的数据量。历史数据存储在 mongo 数据库中。mongo 数据库中的数据按天存储。具体时间是每天晚上 23:59，系统将一天统计下来的 redis 结果集存入 mongo 数据库，同时将 redis 数据清零。

查看历史分析数据，可以直接查看 mongo 数据库，或通过前端页面。在前端有一个历史数据查询界面，能够通过选定的时间参数查询 mongo 数据库，进行显示。这些数据有进一步分析的价值，系统仍然有巨大的扩展空间。

### 4.3 redis 数据库结果集数据结构设计

用户行为数据分析节点集群的最终分析结果反应为 redis 数据库中的一系列哈希表、字符串和集合。它们表示着分析出的用户行为数据。

redis 中的哈希表（hash）是一种键值（key-value）结构，一个键对应一个值，根据键计算存储地址，访问速度很快。在用户数据记录时，用户的 ip、sessionid 和网页错误类型等数据存放为 hash 格式。redis 中的有序集合（zset）是以跳跃表为底层结构的键值结构。跳跃表可以高效的对某随机键的值进行加法运算。因此是数据分析中用户 pv、uv 等累加型数据的理想存储结构。在本文构建的用户行为分析系统中使用较多。

这些表中的数据也是下一阶段前端显示模块中的图表（线图、饼图）的原始数据。数据分析模块向结果集中单向写入数据，而数据展示模块从结果集中单向读取数据，两个模块之间并无直接交互，redis 是连接它们的纽带。

对 redis 中的表进行说明如表 4-1。

表 4-1 redis 中的数据结构描述

redis 表名	表类型	功能描述
day.uv.zset	有序集合	存储用户当日 uv
day.pv.zset	有序集合	存储用户当日 pv
day.uv.ipsign.hash	哈希表	存储用户 uv 的 ip 标记, 确保单个用户只统计一次
day.uv.snsign.hash	哈希表	存储用户 uv 的 sessionid 标记, 确保单个用户只统计一次
day.most.visit.zset	有序集合	存储用户访问最多网站统计
day.most.refer.zset	有序集合	存储用户来源地址
day.size.zset	有序集合	存储用户流量当日分段统计
day.user.agent.zset	有序集合	存储用户浏览器计数
day.agent.dev.zset	有序集合	存储用户终端当日分段统计
day.agent.browser.zset	有序集合	存储用户浏览器版本号计数
stay.time.start	哈希表	以 ip@jssessionid 为 key 值, 存储用户到达时间
stay.time.end	哈希表	以 ip@jssessionid 为 key 值, 存储用户最后访问时间
stay.time.ave	哈希表	以 ip@jssessionid 为 key 值, 存储用户平均停留时间
day_access_deep_zset	有序集合	存储用户访问深度
day_ap_id_hash	有序集合	存储用户 id 标记
day_ad_key	字符串	存储用户访问日均时间
urlerror.day.line.zset	有序集合	当日错误分段计数
urlerror.day.zset	有序集合	当日错误计数
urlerror.day.time.hash	哈希表	错误发生时间
urlerror.day.type.hash	哈希表	错误类型
search.day.keyword	有序集合	关键词频率
day.ip.zset	有序集合	用户 ip 计数, 用以生成 map

结果集的数据表以 hash 和 zset 为主。有些表单独承担某一功能, 如 day.uv.zset 表和 day.pv.zset 表直接存储了用户访问计数和新用户计数。有些表联合承担某功能, 如 stay.time.start、stay.time.end、stay.time.ave 这三个表都是用户停留时间统计用到的表, 分别存储了最初到达时间、最后到达时间、平均访问时间。

## 4.4 dayinfo 分析节点设计

dayinfo.js 分析节点集合了 pv、uv、流量、来源网站这四个分析项目。这 4 项统计目标互相之间存在很大的关联性，所以放在一个分析节点之内。

dayinfo 节点中的分析指标都是不需要对 http 报文数据本身做进一步转化就可以直接提取的信息。其中 pv，uv 的分析用到了自定义方法，该方法的有效性在测试网站上通过。

节点结构如图 4-2 所示，可以在图中看到节点内部的功能组成，和该节点与系统上下游模块的关系，节点的数据来自 http\_trace，节点的分析结果显示在前端页面。

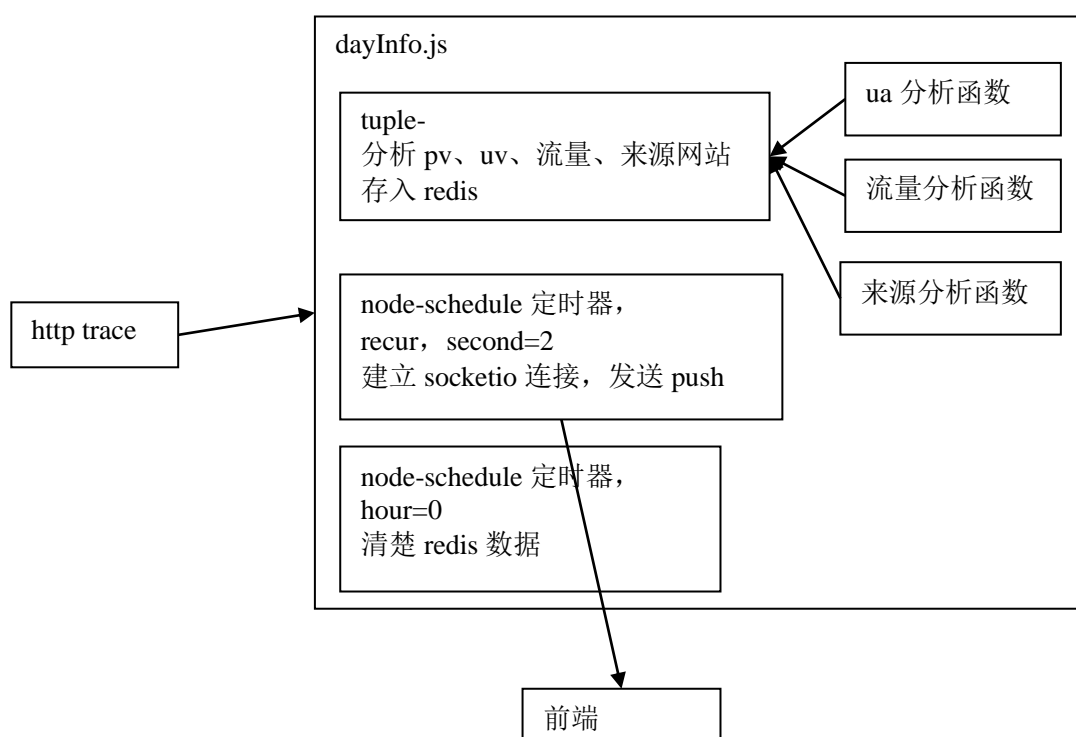


图 4-2 dayinfo.js 结构关系

dayinfo.js 分析节点文件被上传到 daemon 程序管理的运行环境后，从事件监听器中得到 http 消息，分析出用户的 pv、uv、来源网站、流量统计四种信息。

pv（page view）即页面访问量，是用来衡量一个网页状况的常见指标。pv 操作通常是对网站 html 页面的一次访问。用户的一些其它操作，比如 jsp 插件产生的操作或 php 动态产生表格子菜单等页面互动操作也可以被看成一次 pv。网站通常会长期记录它们的页面 pv。网站的管理者会分析 pv 的变化，推究 pv 变化的原因。

uv (unique visitor) 即新的独立访客, 每一个访问网站的新的 ip 和会话编号结合可以确定一个新用户。一天 00:00-24:00 内同一个用户只被计算一次。

来源网站即用户来到页面之前所在的页面, 用户通过这个页面上的一个链接来到所在页面。

流量是一段时间通过网络的数据量大小。

原始 http 报文抱头结构如表 4-2 所示。

表 4-2 原始数据包

```
'http_trace_req|1415089120999.068|220.181.108.174:60003|172.16.1.1:18083|#1|http 1.1
|get|uxunimg/cust/shop/54862/goods/pic/2014/07/30/712508148719313914231163_80_8
0.jpg\n host: emall.lzbank.com\n connection: close\n user-agent: baiduspider-image+
(+http://www.baidu.com/search/spider.htm)\n accept-encoding: gzip\n accept: */*\n ref
erer: http://image.baidu.com/i?ct=503316480&z=0&tn=baiduimagedetail'
```

对 http 数据的解析, 因为 http 协议本身就有明确的结构, 可以直接提取的信息有很多。dayinfo 节点的四个分析项目有一些共同特点, 如一次提取、直接提取, 同时它们具有上下级关系。

用户设备信息也直接来源于 http 报头, 但 user-agent 的分析和 pv/uv 等信息不同, 需要根据不同浏览器的 user-agent 字段的组成和顺序来划分, 而且该字段的格式没有统一标准, 不同的浏览器制作者使用不同的方法定义该字段, 需要凭经验分辨, 增加了分析的难度。在通过 user-agent 分析浏览器型号时, 首先判断某些关键字的有无, 确定一个浏览器版本的范围, 然后再判断关键字的顺序, 判断具体的浏览器型号, 通过其中的一些关键字提取浏览器版本号。用到了正则表达式检索方法。

在数据进入节点之前, 会对 http 报文进行预处理, 将其转换为 json 格式<sup>[24]</sup>, 方便后续的处理。json 格式是一种以对象的形式组织数据的方法, 数据以键-值方式存储, 可以不断嵌套。

转换成 json 格式后的报文数据如所表 4-3 示。在 dayinfo 分析所用到的字段在表中通过注释标明。流量数据 “size: '180.59 kb'”, 来源网站 “host: 'emall.lzbank.com'”, 会话编号 “jsessionId: 'f859049ec236bb83ed6 dcc4844cd531'” 等 dayinfo 分析相关的关键字段用黑体显示。

可以看到, 来源网站、流量、访问网站都可以直接在 json 数据中提取, pv、uv 的分析也有字段标明。dayinfo 节点的流程设计单向向下。和 dayinfo 中项目的分析方法不同, 在后续的访问深度分析中将会有新的结构, 构建滑动窗口来综合分析多个报文的信息。

表 4-3 json 转化的原始报文结构

```
{ base:
  { ts: '1410852492847.014',
    seq: '#4',
    http_version: 'http 1.1',
    http_method: 'post',           //结合 http_method 字段和 accept 字段确定一次 pv
    urlinfo: { url: '/emall/gaininnercount.do' },
    src: '192.168.0.131:38139',    //源 ip, 结合 jsessionid 字段确定一个 uv
    dst: '166:80',
    return_code: '200',
    return_status: 'ok',
    cost: '57.531982421875',
    size: '180.59 kb' },        //计算流量的依据
  headers:
    { host: 'emall.lzbank.com',   //计算来源网站的依据
      'user-agent': 'mozilla/5.0 (x11; ubuntu; linux x86_64; rv',
      accept: 'text/html, */*; q=0.01',    //确定 pv 的依据
      'accept-language': 'en-us,en;q=0.5',
      'accept-encoding': 'gzip, deflate',
      referer: 'http://emall.lzbank.com/emall/default.do',
      cookie:
        { 'jsessionid': ' f859049ec236bb83edf6dcc4844cd531',    //确定 uv 的依据
          pgv_pvi: '2639901696' },
      connection: 'keep-alive',
      'x-requested-with': 'xmlhttprequest',
      pragma: 'no-cache',
      'cache-control': 'no-cache',
      'content-length': '0' } }
```

#### 4.4.1 用户访问量 pv 的分析依据

pv 是用户页面访问量。本文设计了通过同时检索报文中的 `accept` 字段和 `http_method` 字段以确定用户 pv 的方法，该方法在该测试阶段测试成立。

当用户产生网页操作时，通常会在一次点击（一次 pv）中产生多条 http 报文，如何通过过滤这些 http 报文来确定用户的单次 pv 操作，是这部分数据分析的难点，在设计 pv 计算节点时，记录多次点击所生的所用报文，将它们逐条比对，提出了数种过滤规则，然后逐个实验，最后确定了当前方法。在测试中，该统计方法测试成立。

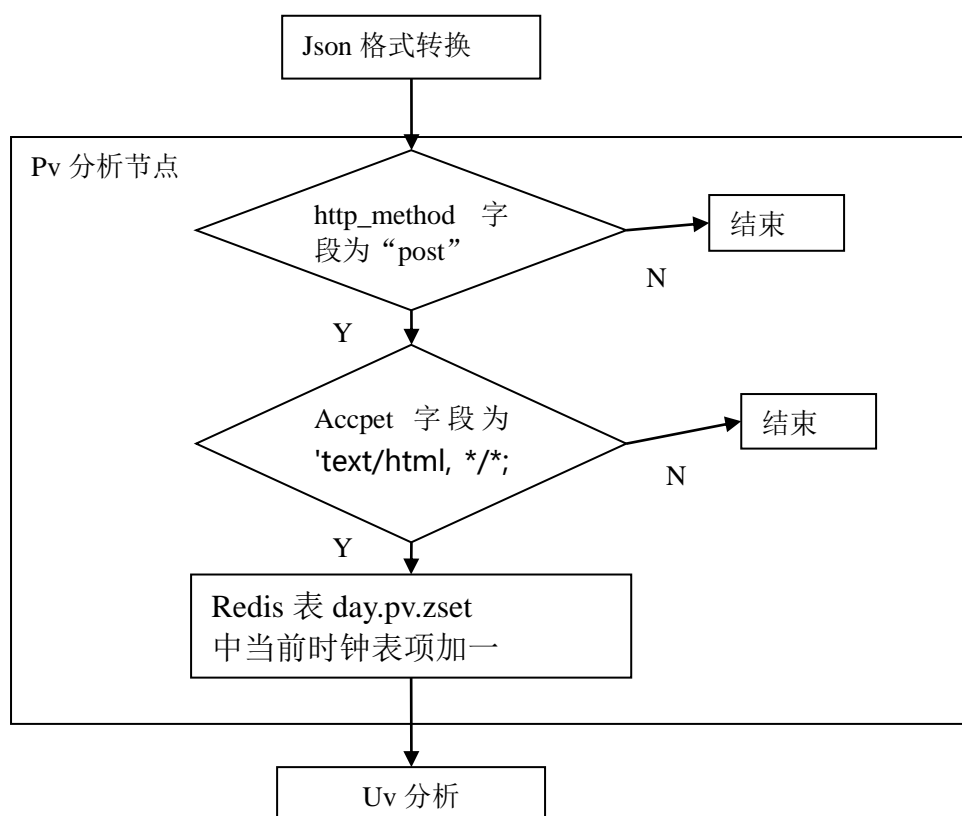


图 4-3 pv 分析流程

用户访问数计数算法流程如图 4-3。该方法的理论基础为，在用户一次操作中，虽然会产生大量 http 报文，但是同时在 http\_method 和报头中 accept 字段都为特定值的报文却只有一个，于是可以用这两个字段的检索确定 pv，流程描述为：

1. 将 http 报文转化为 json 格式。
2. 检测 json 消息中 http\_method 字段的值为 'post'，转到下一流程，否则丢掉报文。
3. 检测 json 消息中 accept 字段值为 'text/html, \*/\*'; 转到下一流程，否则丢掉报文。
4. redis 表 day.pv.zset 中对应表项加一操作。
5. 发送事件到 uv 分析节点。

用户访问量 pv 分析是用户来源网站和用户访问最多页面排名分析的基础，这两个分析节点的数据流来源于 pv 分析过滤出的数据报头。用户访问量的数量应该多于用户数，等于平均访问深度乘以用户数。同时，用户访问量的值等于所有用户访问页面计数值相加的和。

该用户访问量统计算法相对传统算法的区别在于实时性。它每次处理单条数据，将结果累加到历史结果中。而传统统计算法是基于完整数据的。

#### 4.4.2 新用户 uv 的分析依据

uv 是新用户的数量，在计算用户数时，一个用户在一天内只计数一次。在第二天访问时才能重新计算。一个用户如果上午和下午都访问过该网站，则用户访问量每一次都会增加，但新用户数量只在第一次访问时计算一次。

节点中对 uv 的分析是通过结合 ip 和 jsessionid 值来确定的。在同一个路由器 native 网段内的不同机器可以享有相同 ip，所以 ip 地址的相同的两次访问可能来源于不同用户，通过组合 ip 与 jsessionid，可以定位唯一用户。

当在节点中通过对报文的过滤。取得一个标志用户的 ip: sessionid 数据键值时，就把它作为主键将在 redis 中有序集合上的相应的值域进行加一操作。

用户 uv 的计算流程如图 4-4。

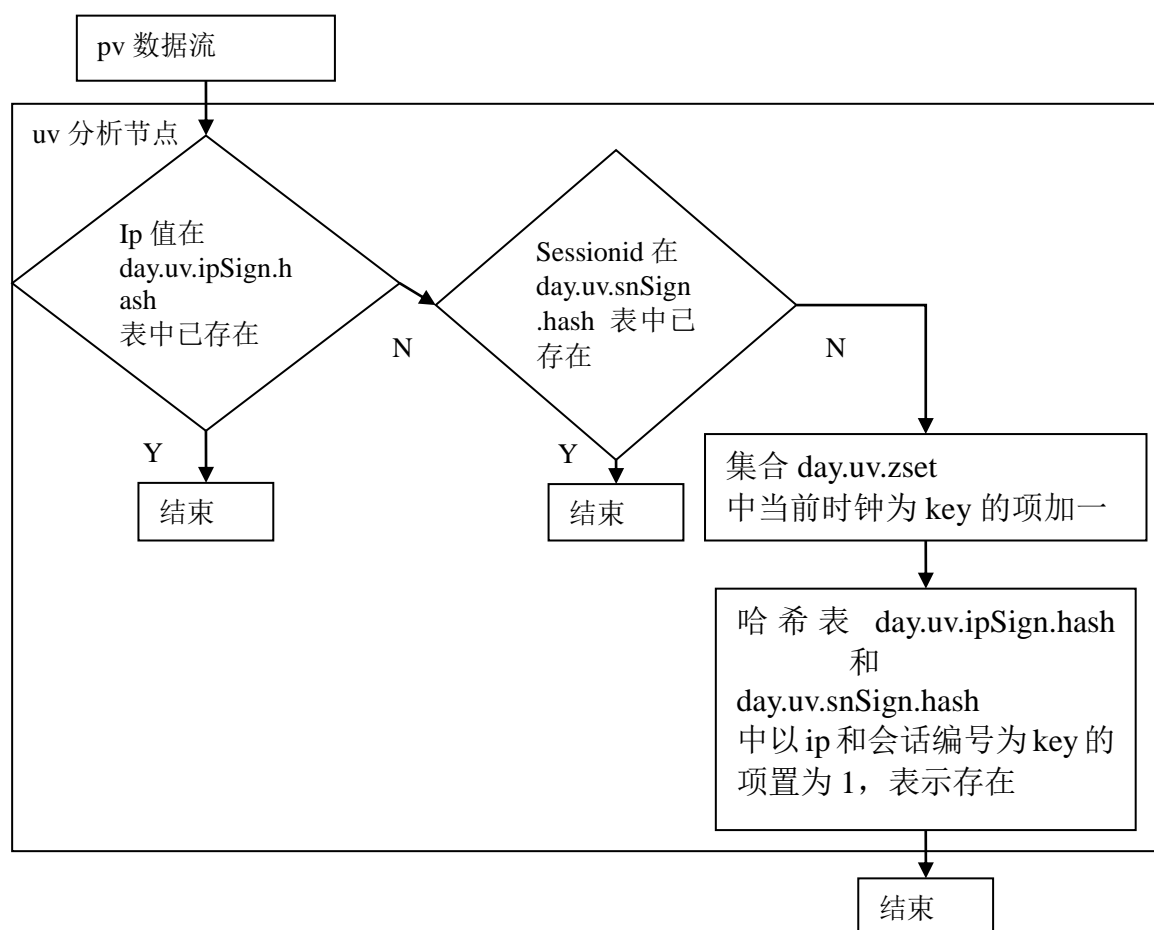


图 4-4 uv 分析流程

### 4.4.3 来源网站和访问页面的分析依据

对于用户行为中来源网站和访问页面的统计。直接对 json 数据中 host, referer 字段进行提取。在分析时先判断该字段存在与否, 再增加相应 redis 表项或增加已有项的值。来源网站的分析流程如图 4-5。

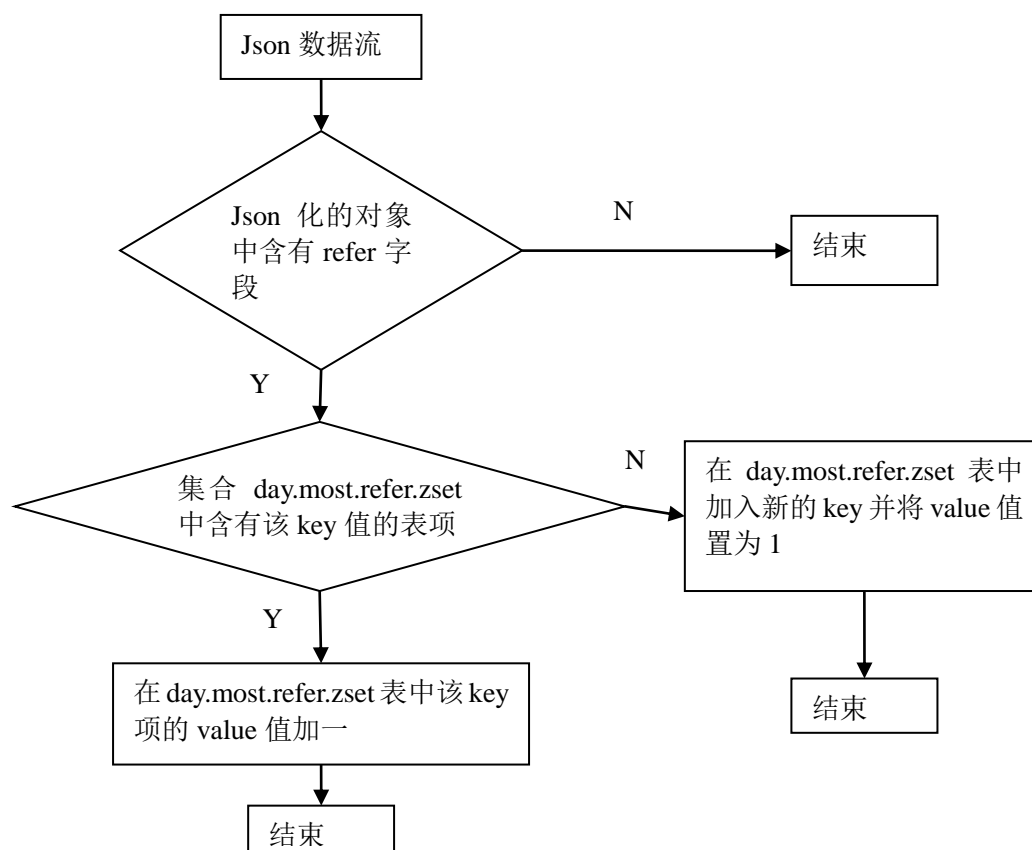


图 4-5 来源网站统计分析流程

用户得知网站的链接并访问网站有许多可能的途径。如果是通过其它网站上的链接到来, 网站的 http 报文中的 refer 字段记录这个来源页面的链接。也有可能是老用户向他周围的朋友推荐了这个网站, 新的用户直接使用 url 访问该网站, 在这种情况下, http 报文中的 refer 字段为空。在扩展分析中, 可以以此定位新用户和老用户。

在来源网站分析节点中, 其它网站上的链接可能是文字链接或图片链接, 可以通过过滤格式关键字区分。

访问页面计数是统计用户访问不同页面的数量, 对比网站上不同网页的访问量可以揭示用户感兴趣的网页内容, 帮助网站安排网页内容分布。

访问页面的计数统计流程如图 4-6。



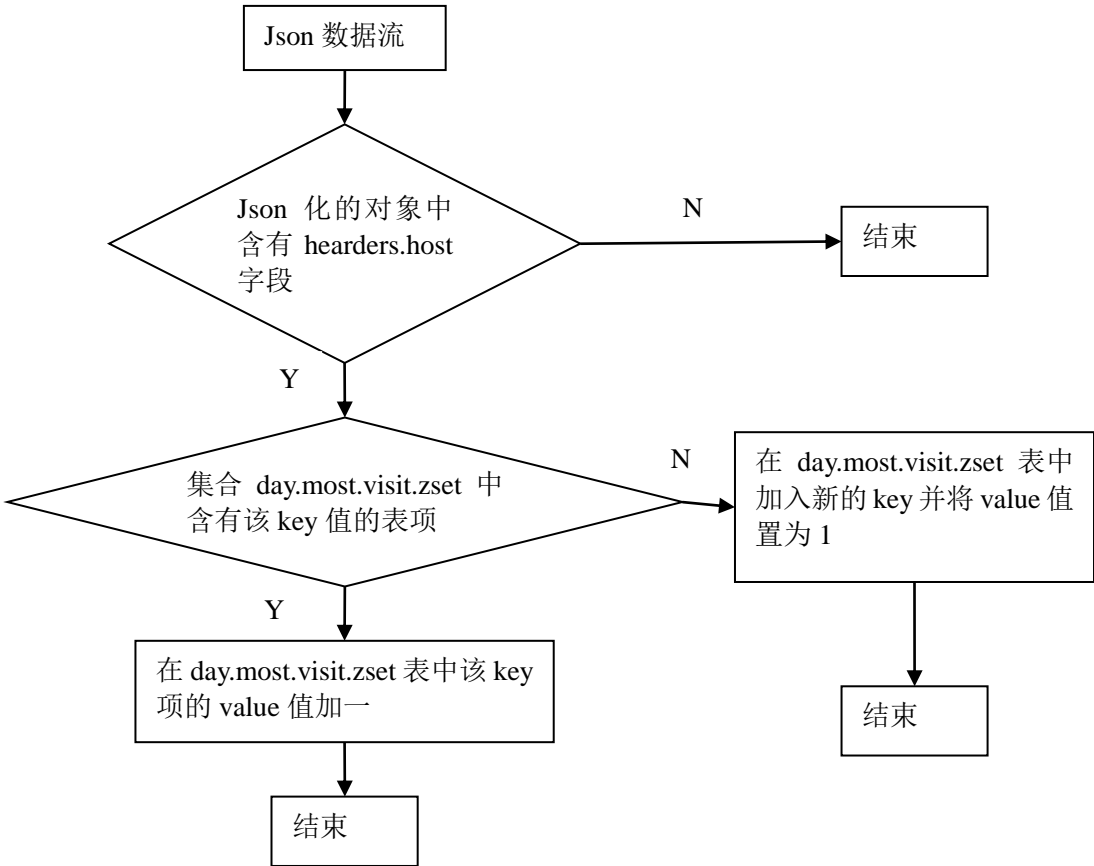


图 4-6 访问最多页面统计分析流程

来源网站和访问页面的统计结果部分列表显示如表 4-4 和表 4-5。

表 4-4 访问页面数据格式

....com/emall/shopcar/gets...	685
....com/emall/gaininnercou...	653
....com/lzb2c/client/clien...	567
....com/emc/esb/uxunmsg	476

表 4-5 来源网站数据格式

nalent.faisco.cn	76
image.baidu.com	27
www.3d100.cn	11
www.so.com	6

表 4-4 和表 4-5 的数据是用真实的 http 报文测试分析得到的结果的一部分。可以看到网站链接跳转的来源中，类似百度这样索引性质的网站排序靠前。用户使

用这索引类站点跳转到目标网站的来源占据的比例最大<sup>[25]</sup>。而在访问页面计数中，链接作为一个字符串被当成键值进行计数。在测试网站中包含商品页面，通过这个链接可以提取商品信息，在链接中还包含商品编号，而商品编号关联到商品名字、图片介绍等各种信息。可以在显示模块中查看商品的这些信息。

#### 4.4.4 dayinfo 节点内置函数说明

为了辅助 dayinfo.js 完成对上述四个分析项目的分析任务，设计了几个辅助分析库函数。它们负责单位转换、字段检索过滤等功能。它们作为节点必要功能的一部分，存放在库函数文件中。对 dayinfo 节点用到的三个函数做出描述如表 4-6，表 4-7，表 4-8。

表 4-6 函数 sizetomb 说明

函数名称	sizetomb
函数原型	function sizetomb(string)
函数功能	将数据大小转化为 mb 为单位，在计算网站的流量时会用到这个函数。
参数	字符串形式： 浮点数+空格+单位 如： '12.3 b' '12.3 kb' '12.3 mb'
返回值	float

表 4-7 函数 sizetoarr 说明

函数名称	sizetoarr
函数原型	function sizetoarr(float)
函数功能	构建 socketio 的推送消息，将输入参数转化为 json 格式。使响应的前端模块能够理解这条信息的含义。
参数	float
返回值	json 对象，包含 js 和 count 关键字 如： {ts: 20141102031232,123.23}

表 4-6 和表 4-7 中描述的函数 sizetomb 和 sizetoarr 承担了在推送消息时的格式转换功能。表 4-8 描述的函数 check 完成特定字符串过滤功能。

表 4-8 函数 check 说明表

函数名称	check
函数原型	function check(name, [])
函数功能	检查 data 的后缀格式是否在[]中，在过滤用户访问最多网站时用到该函数。去除无意义的数据包。
参数	name: 字符串, url 格式 如: https://github.com/tjatse/pm2-gui.html [],条件数组, 如: ['jpg','gif','jpg','css','js','png','ico','jpg','jpg//','html','txt']
返回值	true: data 包含于条件数组 false: data 不在条件数组中

这三个函数完成字段的过滤、单位的转化、格式的转换等辅助功能。这些函数加入流数据数据分析平台的函数库文件。这样一来，由于所有的节点都 import 了 vrsscore 文件，所有节点在部署和运行时都可以直接调用这些函数，在未来的新增节点中，也可以直接使用这些函数。这是代码复用率高的方案，系统已经有的 top, aver 函数等都是直接加在 vrss 平台的库函数中。

这些库函数封装了易变化的部分。当格式转换的规则发送变化时，变化限定在函数中，不影响节点主逻辑代码。

dayinfo 节点分析的结果为表 4-1 中 redis 中的相应集合和哈希表的更新。

## 4.5 用户浏览器和设备分析节点设计

浏览器和设备信息都是从 user-agent 字段分析得出<sup>[26]</sup>。

各种版本的浏览器的 user-agent 字段不同，但没有统一的规定。每个公司都按照自己的习惯排布浏览器版本、编号、引擎等信息。

表 4-9 列出部分浏览器 user-agent 的格式。

表 4-9 部分浏览器 user-agent 信息格式

Chrome 0.2	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML, like Gecko) Chrome/0.2.149.29 Safari/525.13
Opera 8	Opera/Version (OS-or-CPU; Encryption; Language)
IE 3.02	Mozilla/2.0 (compatible; MSIE 3.02; Windows 95)

在通过 user-agent 字段分析用户浏览器和设备类型时。本文研究了主流浏览器的 user-agent 地段的排列方法。然后通过正则表达式匹配 user-agent 字段。根据匹配结果判定用户浏览器信息。用户浏览器和设备判定流程如图 4-7。

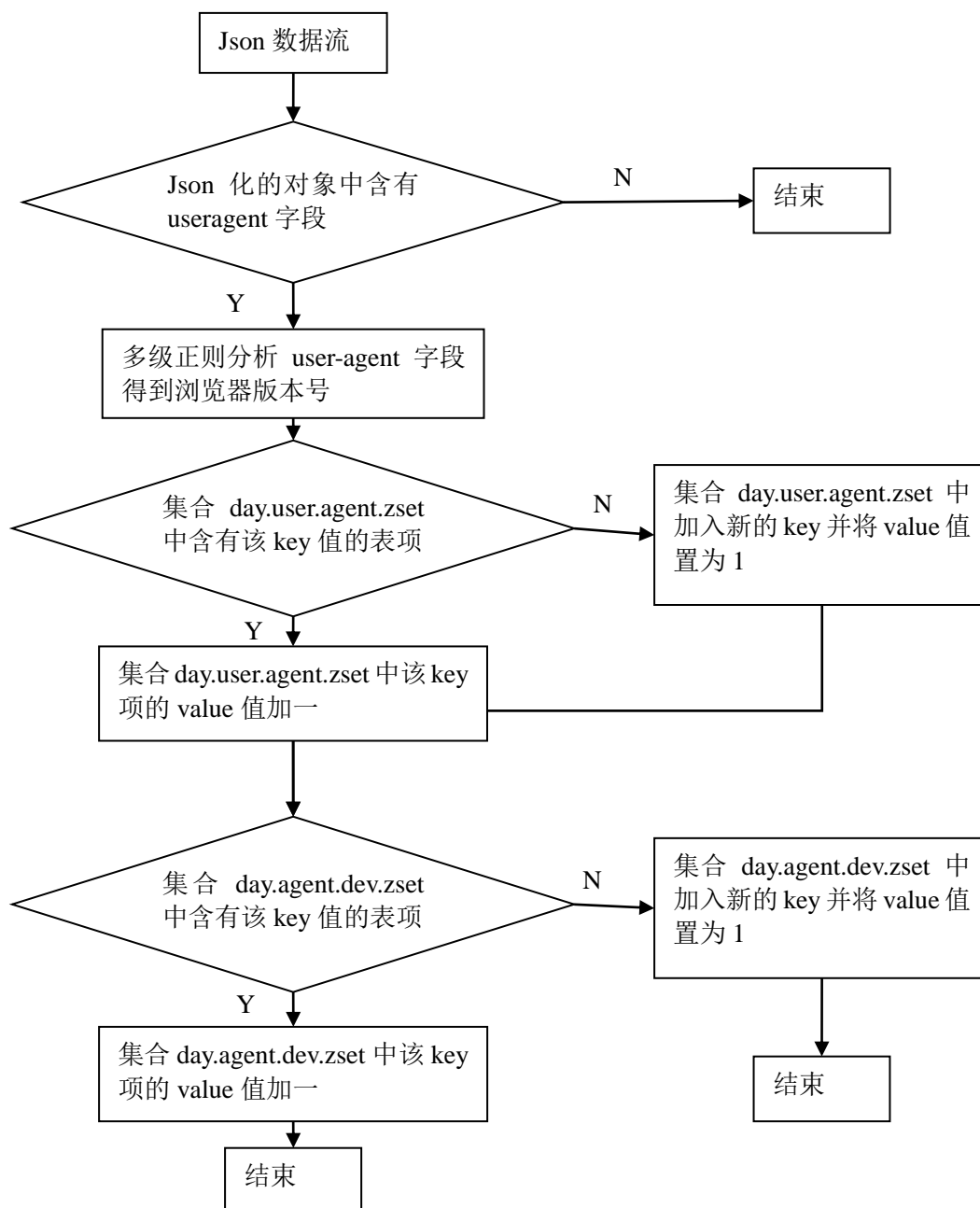


图 4-7 浏览器和设备统计分析流程

## 4.6 用户地点分析节点设计

用户地理分布节点的任务是分析访问网站的用户所在的地点<sup>[27]</sup>（城市，街区），进行统计。为了分析用户的地理分布，在本系统中通过用户的 ip 地址转化到用户的经纬度，再通过经纬度确定城市。

在通过 ip 转化为经纬度的过程中，节点调用了开源插件 `geoip`。  
其过滤算法的流程用图 4-8 来表述。

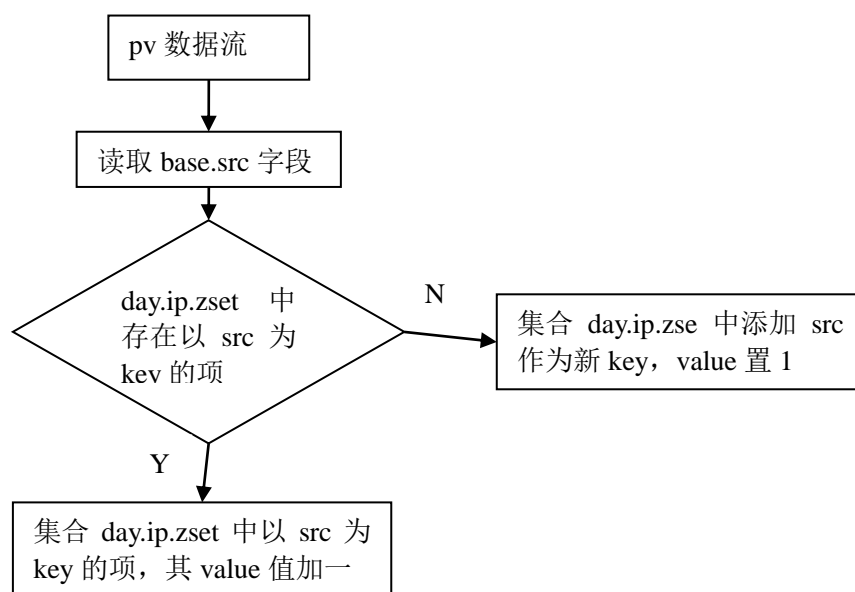


图 4-8 用户 ip 计数流程

ip 地址到经纬度的转化通过 `geoip` 开源插件完成<sup>[28]</sup>。使用 `geoip` 模块分析 ip 地址后，得到 json 格式的地址信息，形式如下：

```

{
    range: [ 1991507968, 1991770111 ],
    country: 'cn',
    region: '15',
    city: 'lanzhou',
    ll: [ 36.0564, 103.7922 ]
}
  
```

这个 json 数据段中含有的信息有坐标范围、国家代码、区域代码、城市的英文名称和经纬度坐标。系统作图需要用到其中的 `ll` 字段的坐标信息，将坐标信息按照地图插件 `e-chart` 需求的标准格式输入 `e-chart` 的地图组件，就可以在地图上打一个点<sup>[29]</sup>。通过 `e-chart` 的配置对象，可以配置点的大小颜色等数据。在数据统计时，记录用户的 ip 地址出现频率，保持数据库的简单。在向前端推送用户地理信息时，再将 ip 信息转化为经纬度、城市名等信息。最终需要传送到前端的数据项有 ip、城市名、经纬度、用户数。

得到兰州某银行的用户数据进行测试后，部分结果数据如表 4-10。

表 4-10 地图绘制信息

序号	ip	城市	计数
0	42.91.49.116	lanzhou	52
1	125.74.78.255	lanzhou	34
2	125.76.61.228	lanzhou	30
3	42.90.131.42	lanzhou	27
4	125.76.35.35	lanzhou	26
5	42.88.137.178	lanzhou	25
6	221.7.41.105	lanzhou	17
7	118.180.123.52	lanzhou	17
8	118.180.137.105	lanzhou	16
9	118.180.89.116	lanzhou	13

从测试网站的地图数据中，可以发现用户量随地点的远离而减少。

使用用户的 ip 分析得到的用户地点有一定的误差，在显示地图时为了有一个直观的印象，将同一个坐标上的多个标记点随机打在地图上该点附近的小片区域。

用户上网时可能使用代理，使用 ip 分析得到的地理坐标也并非精确值，所以该模块的分析结果有一定模糊性，但足以推断不同地区的用户数量分布。

#### 地图散点方法：

在对用户的地理位置在地图上打点的时候，通过 geoip 地址分析出的坐标地点只能精确到地区级，无法更进一步精确<sup>[30]</sup>。当前商业 ip 转化插件可以得到更精确的结果，但仍以地区划分，只是地区划分更加细致，本质不变。这使得同一个地区的人们在地图上被标注成同一个点。在表 4-9 中也可以看到，很多不同的用户的城市都是兰州。它们的坐标在计算模块得到的结果都是一样的。这样一来，如果对它们的坐标直接进行打点作图操作，这些用户在地图上只会有一个点。在全部地图上，也是当某个城市有用户访问目标网站时就有了相应的标记。而无论这个地区的用户数量具体是多少，都只有一个标记。这无疑很不直观。为了使地图的显示更加和真实情况贴合。在处理向前端的数据时引入了两个随机量， $r$  和  $a$ ，分布代表半径和角度。使用下面的公式计算出一个相对随机的坐标值。把一个城市的用户随机分布在以该城市（地区）的坐标为中心的圆形区域上，这样可以使用户多的城市有更多的地图点位分布。和实际情况更加吻合。

$$x = ll[0] + r * \cos(a);$$

$$y = ll[1] + r * \sin(a);$$

这种散点方法中，在不同半径环上落点的概率均等。在不同半径环上落点的总数量的期望相等。所以当落地密集时，距离圆心越近落点密度越大，这符合客观用户分布规律。

地图分析节点的结果表现为表 4-1 中 redis 中相应集合数据项的增加。

#### 4.7 用户关键字分析节点设计

关键词是指用户在由搜索引擎的网站中，在搜索引擎中搜索时输入的字符串。在用户搜索关键字时，通常在 **http** 报文的 **cookies** 字段中对关键字有所反应。

关键词分析节点首先将用户搜索的关键词通过字符判断过滤出来，然后将关键词的权值加一，如果该关键词还没有被搜索过，则将该关键词加入 **redis** 的关键词哈希表。最后得出一段时间内的关键词搜索排名。

最终的分析结果显示为列表，同时提供直观的词云显示<sup>[31]</sup>，使用户有一个直观的体验。词云图绘制使用了 **tag-canva** 开源技术。**tag-canva** 技术可以根据输入的词组绘制词云图。词云图将关键词显示为一个隐约的球体上的一些点，使用户看到不同的关键词。最终的关键字分析结果表现为 **redis** 中相应集合数据项的增加。

关键词云图是动态滚动的，它的静态效果如图 4-9。

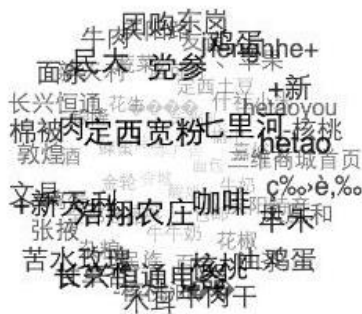


图 4-9 关键词云图

在关键词统计中,数据由json化的http数据字段而来。首先判断是否有keyword字段,然后进入后续流程。这个节点是直接从http数据源接收数据。数据流不从dayinfo节点接收是因为pv分析的后续报文中并没有keyword字段。

用于定位 pv 的字段和定位搜索关键词的字段有一定关联，但不可复用。用户操作会产生多个报文，但在搜索操作中，每次搜索只有一个报文携带关键词字段 keyword。有 keyword 字段的报文可以表示一次 pv。一次 pv 却不一定包含一次关键词搜索。

关键词分析流程如图 4-10。

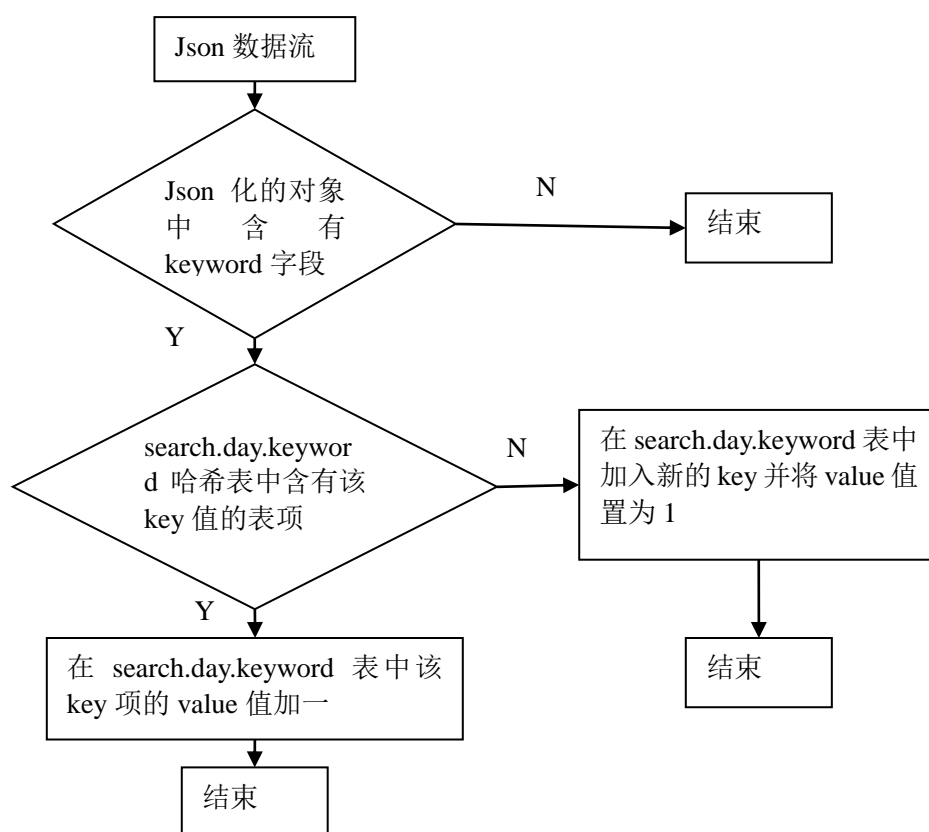


图 4-10 关键词统计流程

## 4.8 用户访问深度分析节点设计

用户的访问深度是用户访问不同页面的个数。当一个用户在一次访问中，访问了网站上的多个页面时，我们可以认为这个网站上有用户感兴趣的信息。可以把用户平均访问深度看作网站对用户吸引力的一个标准。这个标准的计算方法就是计算用户访问页面总量和用户数量的比值。可以认为这个比值大的页面的用户体验更好。

用户实时平均访问深度计算节点计算从当前时刻往前一个小时之内用户平均访问深度。访问深度是指单个用户浏览的不同页面的个数，平均访问深度应该等于用户点击量除以用户个数<sup>[32]</sup>。

为了记录用户的平均访问深度，该节点需要记下过去一个小时的用户访问量和用户个数。而为了使显示效果达到实时，即每过一分钟需要将计算量减去一个小时之前的一分钟的访问量。再加上最近的一分钟的访问量。

在该节点的设计中，需要一个带滑动窗口的缓冲区记录过去一个小时的访问量，该窗口需要 60 个格子，记录每一分钟的具体值。因此，本文设计了使用循环



链表<sup>[33]</sup>来记录数据的方法。循环链表是一种链表结构，链表最后一个节点的指针指向链表的第一个节点。

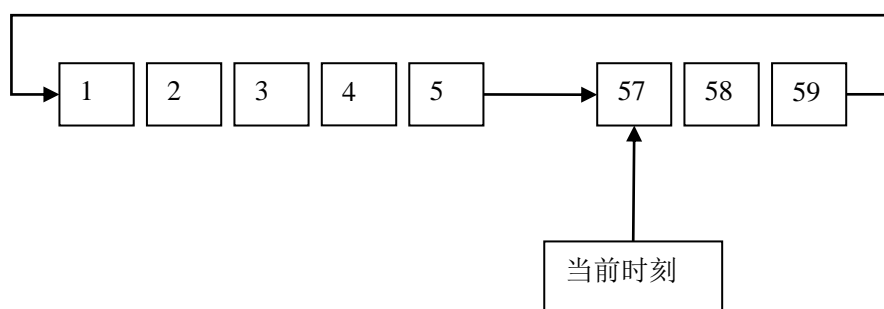


图 4-11 循环链表记录用户 pv

循环链表的结构如图 4-11。当循环链表的构造完成后，对链表的操作限定在相应时间表项的加减之上。对从 pv 分析节点接受到的上一级数据流的操作会变的非常简单。

访问深度统计的算法用流程图如图 4-12。

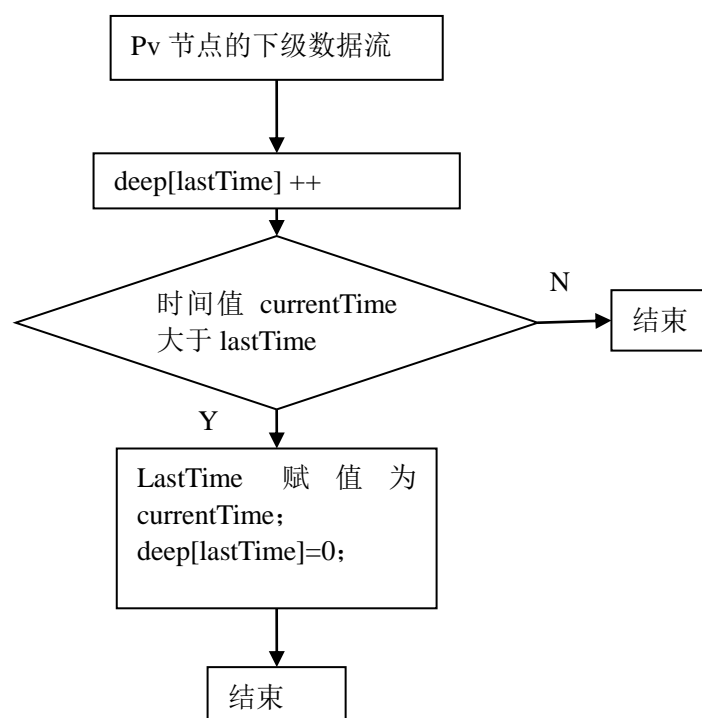


图 4-12 访问深度累加逻辑

该算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

访问深度统计算法相对传统算法的最大优点在于实时性，它是一个长时间运

行的算法，每次运算针对一条数据，结果可以累积；而传统的访问深度统计算法一次针对大量数据，一次得到完整结果。

在最初的设计中，存储每分钟访问数的是一个巨大到  $24 \times 60$  的数组。对数据的计算还涉及到各种变换和同步操作。而改变成循环链表之后，合理的数据结构减少了大量的检索和同步操作的消耗。实际上，此时的流数据本身的内容不再重要。重要的是每一条数据到达的时间。这个到达时间会被访问深度分析节点取得，将其中的分钟数提取为一个整数 `currenttime`。以此作为数组下标来对相应的表项做出操作。

用户访问深度分析节点的数据是来自用户 `pv/uv` 分析的下一级运算，它时刻累积计数 `pv/uv` 到两个循环表中。每次记录新的 `pv/uv` 时都检查当前时间，取分钟数为一个整数，如果这个数值和上一次记录的数据不同，就将积累的数据存入这个整数对应的链表节点中的数据。并且覆盖该节点中原有的数据。使用这种方法，同时记录了新的数据值，和删除了滑动窗口理论上最后一个数据项，操作简单，逻辑清晰。

通过访问深度统计节点分析得到的结果数据格式如表 4-11。

表 4-11 访问深度计算信息

序号	ip	sessionid	计数
1	42.91.49.116	31db441d1b06e9b64d5c53cc31b53386	7
2	221.7.41.105	9b9ad84e91f54a8cd5b10fe957ac5acc	5
3	125.76.61.228	2636cc4bfdde84732deeac95c3fbabf1	2
4	118.180.137.105	fe476f5b5de422cbd8ab02c4fd51413b	1

## 4.9 用户停留时间分析节点设计

用户停留时间是用户到达网站到用户离开网站的时间。

在分析用户停留时间时，用户在最后一次与网站发生操作之后，如果继续浏览页面但不发生操作，在分析模块中是无法感知到的。因此这里统计到的用户停留时间是用户第一次到达该网站到用户最后一次在网站发生操作的时间的间隔。

该节点具体的统计方法是设计两个哈希表，一个记录用户的最初到达时间，一个记录用户的最后一次操作时间。当检测到一个用户的访问时，首先搜索最初到达时间的 `hash` 表，如果该表中存在这个以 `ip: sessionid` 标识的用户，就将当前的时间值存入这个用户的最后操作时间表并覆盖原来的值。如果最初到达时间表中没有该用户的表项，就把当前时间同时加入两张表。

以两张哈希表记录用户访问时间信息使用户停留时间计算简化为表的存取操作。

用户停留时间的过滤算法如图 4-13。

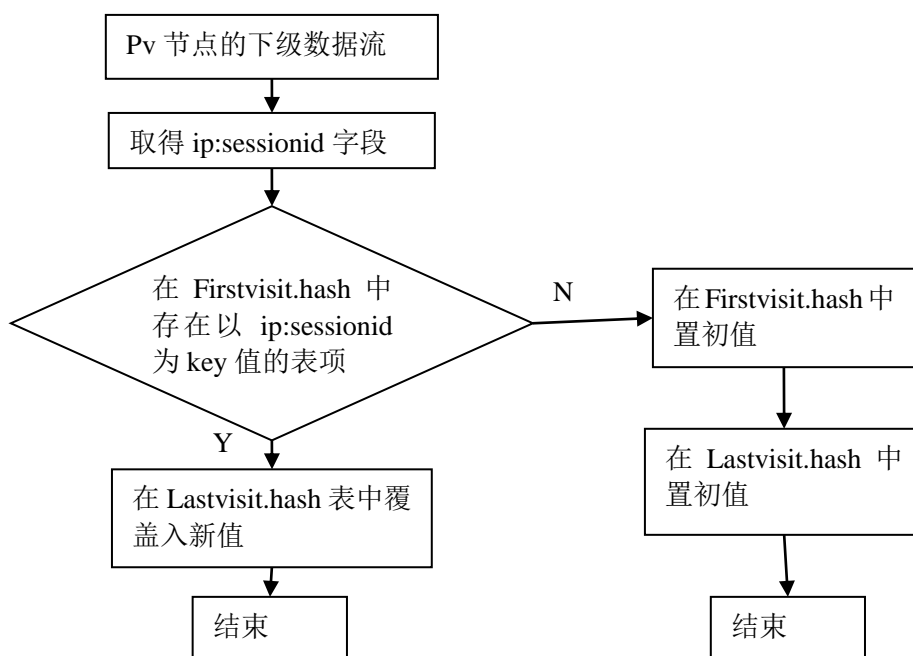


图 4-13 停留时间分析流程

用户停留时间统计算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

用户停留时间统计算法相对传统算法的最大优点在于实时性，它是一个长时间运行的算法，每次运算针对一条数据，结果可以累积；而传统的访问深度统计算法一次针对大量数据，一次运算得到完整结果。

如果一个用户时隔一个小时没有发生任何操作，则认为他的上一次访问已经结束。取出这个用户在两张 hash 表中的时值，计算差值，得到该用户这一个访问的访问时间，把这个时间加入均值计算，清空两个表中代表该用户访问记录的表项。

用户停留时间的分析流程是一个有两个出口点的计算逻辑。在定时器中，每小时检查一次 lastvisit 表单。如果发现其中某一项记录的时间值和当前时间值的差值大于一个小时。就做取数据、清除、计算均值等操作。

停留时间分析节点部分测试结果如表 4-12。这些信息分布在 stay.time.start，stay.time.end，stay.time.ave 这三个 redis 表中。

表 4-12 停留时间结果集信息

序号	ip	到达时间	离开时间	
1	42.91.49.116	sun nov 02 2014 15:38	sun nov 02 2014 15:39	1.186
2	118.180.137.105	sun nov 02 2014 15:38	sun nov 02 2014 15:39	0.74
3	118.181.81.13	sun nov 02 2014 15:38	sun nov 02 2014 15:40	1.693
4	42.90.131.42	sun nov 02 2014 15:38	sun nov 02 2014 15:40	1.647
5	118.180.151.206	sun nov 02 2014 15:38	sun nov 02 2014 15:40	1.652

#### 4.10 socketio 消息推送节点设计

为了实现流计算模块与前端的交互,设计了用户 socketio 消息响应和推送的节点。该节点是数据分析模块和前端显示模块的连接件,存储在 redis 数据库中的用户行为数据经过该节点传输到前端。该模块使用了 socketio 开源技术,可以在两个实现了 socketio 的端口之间通信,通信以事件的形式产生和接受。

socketio 将函数传入通信事件监听器,关键代码如下:

```
//socket.io
pullserver.sockets.on('connection', function(socket) {
  socket.on('day_most_visit_pull', function() {
    rsstream.zrevrange(d_visit,0,-1,'withscores',function(error,data) {
      if(!error)
        socket.emit('day_most_visit_pull_result',data);
    });
  });
  socket.on('day_most_refer_pull', function() {
    rsstream.zrevrange(d_refer,0,-1,'withscores',function(error,data) {
      if(!error)
        socket.emit('day_most_refer_pull_result',data);
    });
  });
  socket.on('day_ip_pull',function() {
    rsstream.zrevrange('day.ip.zset',0,-1,'withscores',function(err,data){
      var msg=[];
      // console.log(data)
      //ip,name,value,geocode
```

```

        for(var i=0;i<data.length;i+=2){
            var geo = geoip.lookup(data[i])
            if(geo){
                // console.log(geo)
                msg.push([data[i],data[i+1],geo])
            }
        }
        console.log(msg[1])
        socket.emit('day_ip_pull_result',msg);
    })
    }));

```

在这段代码中，有一个名为 `pullserver` 的 `socketio` 对象，它调用 `on` 方法添加对事件 `connection`、`day_most_refer_pull` 等的响应函数，函数中经过一些处理（存储数据、整理格式）后发送返回事件。

把模块之间的交互工作设计在一个节点之内，可以减少流数据分析平台的复杂度，增加系统的可移植性，增加系统地灵活度。

当未来开发新的链接模块时，可以添加类似的节点进行交互。在通信响应节点中可以引入不同通信机制。

## 4.11 数据持久化节点设计

数据管理节点每天将 `redis` 中的数据持久化到 `mongo` 数据库中，然后清空当天 `redis` 数据。定时器模块 `node-schedule` 设计为一个库函数。定时器的设置如下：

```

var schedule = require('node-schedule');
var mongorule = new schedule.recurrcerule();
mongorule.hour = 23;
mongorule.minute = 59;

```

在这几行代码中，定义了一个定时任务管理对象 `schedule` 和一个时间规则对象 `mongorule`。在 `mongorule` 的定义中，时间被设置在晚上 23 点 59 分，循环发动。通过下面这行代码，将需要在 `mongorule` 规则启动时执行的代码放在 `job1` 的 `function` 中。

```

var job1 = schedule.schedulejob(mongorule, function() { })

```

`mongo` 中的数据结构为 `json` 格式，`mongo` 数据库中存储了 11 个 `collection`，它们每天各增加一条数据，作为数据持久化空间。

在 mongo 中的数据结构需要包含一天统计下来的必要的用户行为信息。而且需要便于未来的检索，需要有时间和统计目标的标识字段。其数据结构如表 4-13、表 4-14、表 4-15。

表 4-13 所设计的集合包含按小时区分的一天的统计数据的一部分。每一条数据是一个小时中的用户访问量数据和 uv，流量数据。数据通过其中的时间字段检索。

表 4-14 所设计的集合记录用户关键词，count 标明这个关键词被搜索的次数。

表 4-13 电商各小时统计 mongo 数据集合说明

集合名	emall_hour		
功能	电商各小时统计数据		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	datetime	time	时间字符串
	pV	整数	一个小时中的用户访问量
	uv	整数	一个小时中的新用户数
	size	浮点型，两位小数	一个小时的流量，以 mb 为单位
	averagetime	浮点型，两位小数	平均停留时间
	averagedeep	浮点型，两位小数	平均访问深度

表 4-14 电商各天关键词统计 mongo 集合说明

集合名	emall_day_keywords		
功能	电商各天关键词统计		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	keywords	字符串	关键词字符串 eg. “玉米”、“土豆”
	count	整数	关键词计数

表 4-15 所设计的集合记录了网站每天被访问的频率。每一个数据项包含一个被访问链接字符串，key 值 count 记录了被访问的次数。

表 4-16 设计的数据集合记录不同来源网站的来源次数。集合的 key 有 refer 和 count。refer 记录了来源网站页面的链接。count 记录了次数。

表 4-15 电商各天页面访问量 mongo 集合说明

集合名	email_day_urlpv		
功能	电商各天页面访问量, 这个数据集合记录被访问的页面的 pv 数。		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	urlpv	字符串	链接地址 eg. ".../email/gaininnercount.do"
	count	整数	被访问的链接计数

表 4-16 电商各天来源页面频率 mongo 集合说明

集合名	email_day_referer		
功能	电商各天来源页面频率		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	referer	字符串	链接地址 eg. "www.baidu.com "
	count	整数	来源页面的链接计数

表 4-17 电商各天各浏览器访问量 mongo 集合说明

集合名	email_day_browser		
功能	电商各天各浏览器访问量		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	browser	字符串	浏览器名 eg. "chrome39.0 "
	count	整数	浏览器频率计数

表 4-18 电商各天各终端访问量 mongo 集合说明

集合名	emall_day_terminal		
功能	电商各天各终端访问量		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	terminal	字符串	设备类型 eg. "mobile "
	count	整数	设备使用频率计数

表 4-19 错误页面发生频率统计 mongo 集合说明

集合名	emall_day_errordetails		
功能	错误页面发生频率统计		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	errorurl	字符串	发生网页错误的连接 eg. "... /emall/gaininnercount.do"
	errortype	字符串	设备类型 eg. "not found "
	count	整数	错误发生频率计数

表 4-20 电商各天各 ip 访问量 mongo 集合说明

集合名	emall_day_ippv		
功能	电商各天各 ip 访问量		
键说明	键名	数据类型	说明
	_id	objectid	mongo 数据库唯一性标识
	date	time	时间字符串
	ip	字符串	发生网页错误的连接 eg. "192.168.0.183"
	city	字符串	设备类型 eg. "成都"
	count	整数	错误发生频率计数



表 4-17 设计的集合记录浏览器的使用数,每天每种浏览器占用一个条新数据, **date** 字段标明天数、**browser** 字段标明浏览器型号。**count** 字段记录这一天中使用这种浏览器访问网站的用户数量。

表 4-18 设计的集合记录不同终端的使用者的数量。**date** 标明日期,使用 **mongodb** 中默认的时间格式。**count** 记录计数。

表 4-19 所设计的数据集合记录网页上发生的错误的标识和 **count** 计数。

表 4-20 设计的数据集合记录不同 **ip** 的访问量,为用户地点分布的判断依据。每条数据含有四个字段。其中的城市表示使用者所在的城市, **count** 为城市中的用户的计数。

**mongo** 数据和前端的交互不需要通过流数据分析平台,前端可以直接通过 **python** 中的 **mongo** 接口读取。这部分数据相对独立,已经写入的数据不可更改,只能提供读取。**mongo** 是一个跨平台的数据库,对数据的使用方式多种多样,用户行为数据分析结果保存在 **mongo** 中,后续分析工作以 **mongo** 数据为基础展开。

## 4.12 本章小结

本章阐述了用户行为分析节点的设计。

用户行为分析目标有用户访问数 **pv**、新用户数 **uv**、用户访问深度、停留时间、搜索关键词、使用浏览器、用户设备、用户地理分布。

用户行为信息从 **http** 数据包中分析得到,用户的浏览器类型、**ip** 地址信息可以直接从 **http** 报文获取。用户访问量和数据包的数量不具有对等关系,需要过滤特定的字段来取得用户访问量计数。

用户访问深度和停留时间的分析中使用了循环链表建立滑动窗口,过滤一段时间的数据,对比统计结果,进行实时分析。

当天数据保存在 **redis** 数据库中,每天定时将 **redis** 中的数据存入 **mongo** 数据库,进行数据持久化。每天清空 **redis** 数据库。

用户行为分析节点设计中,主要用到两种开源技术:**geoip** 和 **tag-canva**。从 **ip** 地址到经纬度的转化中使用了 **geoip**,该模块可以根据输入的 **ip** 地址得到经纬度。关键词词云图绘制使用了 **tag-canva**,该模块根据输入的词组绘制词云图。

## 第五章 数据显示模块设计

数据显示模块以动态图表的形式实时显示用户行为分析的结果。显示形式有线图、饼图、柱状图、地图等。

### 5.1 数据显示功能需求

在用户行为分析完成后，最后分析结果是代表用户访问量、新用户数、用户地理分布、访问深度的一系统表和集合等结构。显示模块的功能是把这些结果直观易懂的显示出来。显示模块要实时更新，当新的用户行为数据被分析出来后，要能立刻在显示模块中看到。

需要支持的图表形式有：

1. 线图，用以显示流量变化。
2. 饼图，用以显示用户设备和浏览器使用比例。
3. 柱状图，用以显示用户访问深度和用户停留时间。
4. 词云图，用以显示用户关键词分布。
5. 地图，用以显示用户地理位置。

### 5.2 前端显示系统总结构设计

显示模块使用了 mvc 结构<sup>[35]</sup>的 web 网站来实现。mvc 结构是一种使用 mvc (model view controller 模型-视图-控制器) 设计创建 web 应用程序的一种网站结构模式。model (模型) 是应用程序的核心部分 (比如数据库记录列表)。view (视图) 是显示数据的模块 (图表，线图，饼图)。controller (控制器) 处理输入 (响应 restful 请求的接口，接收消息)。

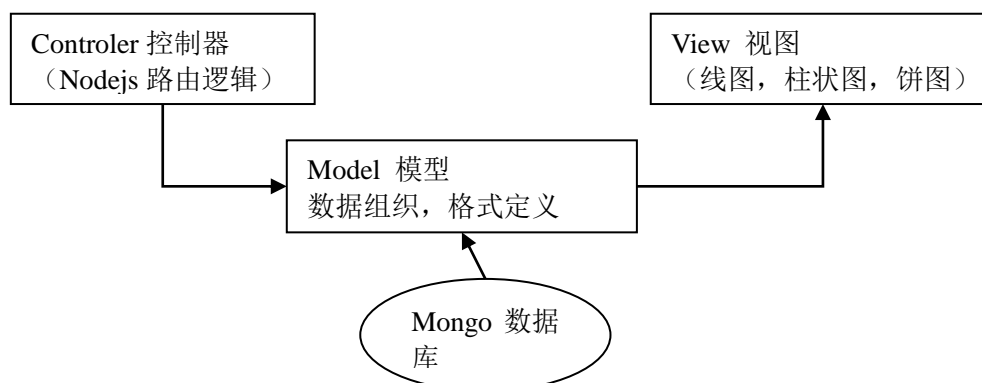


图 5-1 前端 mvc 架构

网站功能结构如图 5-1 所示，分为 control-view-model 三个功能模块。mvc 结构的前端，具有数据格式统一同时显示模式多变的特点，非常适合多样化数据呈现的用户行为数据显示。而将前端显示模块设计为 web 应用，则显示情况可以在任意一台计算机查看，不再需要安装单独的软件，减少了安装/更新软件的麻烦，增加了灵活性。

网站的文件结构如图 5-2。使用了 python 语言实现。

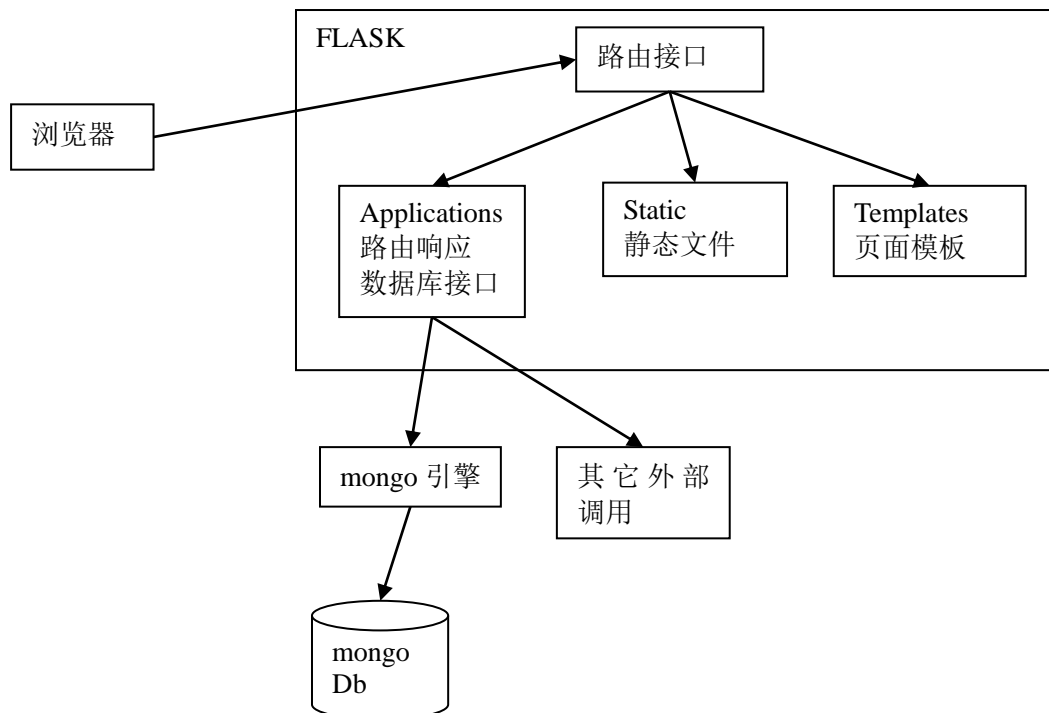


图 5-2 使用了 flask 的网站结构

### 5.3 图表显示方法设计

为使数据显示生动直观，显示系统支持词云图、线图、饼图、双饼图、表格、柱状图显示方法。为了支持这些图表的显示。网站使用了三种开源图表插件：echart、highchart、sparkline。其中，地图模块使用百度开源的 echart 组件。线图、饼图、柱状图使用了开源 highchart 组件。

在 highchart 模块中作图，输入数据结构如下：

```
series: [{
    name: '兰州',
    data: [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9.6]
}, {
    name: '北京',
```

```

        data: [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1, 14.1, 8.6, 2.5]
    }, {
        name: '上海',
        data: [-0.9, 0.6, 3.5, 8.4, 13.5, 17.0, 18.6, 17.9, 14.3, 9.0, 3.9, 1.0]
    }, {
        name: '成都',
        data: [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6, 4.8]
    }
]

```

high-chart 模块在得到这些数据后，就可以在前端绘制图表。

前端监听一个端口，运行 socketio 客户端代码。可以在 socketio 消息通信接口中得到数据，及时的更新到图表上。socketio 响应代码和数据分析节点集群中的 socketio 消息推送节点对应。实时接收消息，可以使前端的图表实时变化。

## 5.4 mongo 数据推送设计

网站 python 文件的启动初始化时，构建了一个 mongohelper 函数来响应网页中的 mongo 数据库读取请求，构建 mongo 连接，关键代码如下：

```

def mongodhelper(dbname,readonly=False):
    global connection
    if len(app_config.mongodb_config) == 0 :
        .....
    try:
        if app_config.mongodb_config['rs'] != "" :
            connection=mongoreplicasetclient(app_config.mongodb_config['url'],
            replicaset=app_config.mongodb_config['rs'])
            .....
    return db

```

当在网页中有某个事件引发了 mongo 操作时，操作路由到该函数代码上执行。mongo 的查询命令通过函数中 db 对象执行，支持普通查询和关联查询。当查询一个区间时，将查询参数设置为类似{'check\_time':{'\$gte':str(start),'\$lt':str(end)}}的格式。下面的代码以用户访问量 mongo 查询响应代码为例，显示关键代码。其它数据的查询代码与此类似。

```

@pvstat.route('/pvstatreport')
def pvstat_stat():

```

```

db=application.mongodbhelper('emall')
if 'end' in request.args and 'start' in request.args:
    result=db.emall_hour.aggregate([{"$match":
        {'check_time':{'$gte':str(start),'$lt':str(end)}}},
        {"$group": { "_id": "$datetime","pv": { "$sum": "$pv" } } },
        {"$sort":{"_id":1}}])
    return jsonify(result)

```

## 5.5 页面导航菜单设计

为了方便查看各个用户行为数据项目，在前端显示网站中设计了导航条。网站的页面分三个主要部分：顶部导航条、侧边导航条、主体显示部分。



图 5-3 显示系统首页结构

如图 5-3 是网站的首页。在进入系统主页面时，会以线图和表格的方式显示用户访问量、新用户数、用户来源网站、用户访问网站频率这几项信息。其它信息在导航菜单中选择查看。顶部导航条有两个选项，分为报告和用户分布，用户分布是用户地理位置的地图显示。

图 5-4 是显示模块的侧边栏结构，侧边栏分为两大部分。命名为受访用户信息和电商统计。受访用户信息是实时分析结果的显示：流量分析、访问深度、停留

时间、地图分布、浏览器、设备、关键词的实时图表。电商统计是查询历史统计结果的菜单。在电商统计子页面中，通过选取一个时间段，作为参数转到 mongo 查询调用，查询 mongo 库中的历史统计信息，将结果显示出来。



图 5-4 侧边栏结构

## 5.6 抓包模块设计

为了获取商品的标题，图片信息，使用 python 编码了一个网页爬虫模块。通过该模块可以在网页上抓去特定的图片。抓去图片的过程如下图 5-5 所示。

通过抓去商品的图片，可以使数据更加丰富，显示更加直观。商品被访问时，通过在访问最多页面统计节点中记载的数据，将商品被访问的情况记录下来。然后在凌晨或晚间网站的流量比较小，用户很少，网站的负载低的时候调用抓包模块，访问网站的商品图片链接，将图片数据缓存下来。这时访问商品图片返回的 http 数据包也会被前面的 pv，uv 等访问统计节点记录下来，作为一个用户的访问被统计。针对这种情况，在系统设计中加入一个特殊用户过滤功能，这个特殊用

户就是显示模块本机的 ip 端口。这个用户的数据在进入流计算系统时就被拦截下来，不会被数据统计节点所统计。这样完成抓取图片操作时，就不会发生无端的用户访问数据增多。

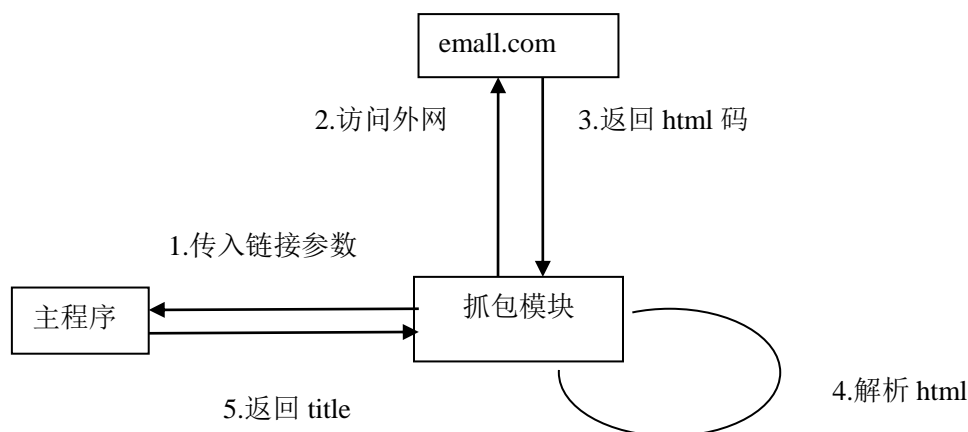


图 5-5 抓取图片流程

## 5.7 本章小结

这一章阐述了用户行为分析系统的数据显示模块的结构。

数据显示模块使用了 mvc 结构来搭建。

数据显示的数据来源于 redis 数据库和 mongo 数据库。redis 中的数据由上一章节中讲述的 socketio 消息推送节点发送而来。

mongo 数据由 flask 中的路由到相应的 python 中的 mongo 引擎模块进行读取。在 python 中编写了 mongohelper 函数作为 mongo 引擎。

数据显示使用了词云图、线图、饼图、双饼图、表格、柱状图等显示方法。用到三种开源插件支持数据显示：high-chart、e-chart、spark-line。其中，high-chart 根据输入数据绘制线图、饼图、柱状图、表格。e-chart 根据输入数据绘制地图。spark-line 根据输入数据绘制特点柱状图。

## 第六章 测试

本章记录系统测试情况，包括性能测试和功能测试。测试数据来源于真实网站访问记录。

### 6.1 测试需求和计划

测试分为两个部分，性能测试和功能测试。

性能测试，通过模拟大量报文，冲入系统，测试系统的稳定性、流畅度、延迟时间、准确性。

功能测试，对比结果的一致性。按照约定的顺序访问网站，点击链接，查询关键字。完成后，观察对比系统的统计结果。测试指标包含以下几项：

1. pv 数和 uv 数的大小对比，任意时段 pv 数必须大于等于 uv 数
2. 来源链接数量和 uv 的关系对比，任意时段，来源网站计数必须小于等于 uv 数。
3. 平均访问深度\*uv 数和 pv 的数值对比，平均访问深度乘以 uv 数必须和 pv 的数值相等。

### 6.2 测试环境

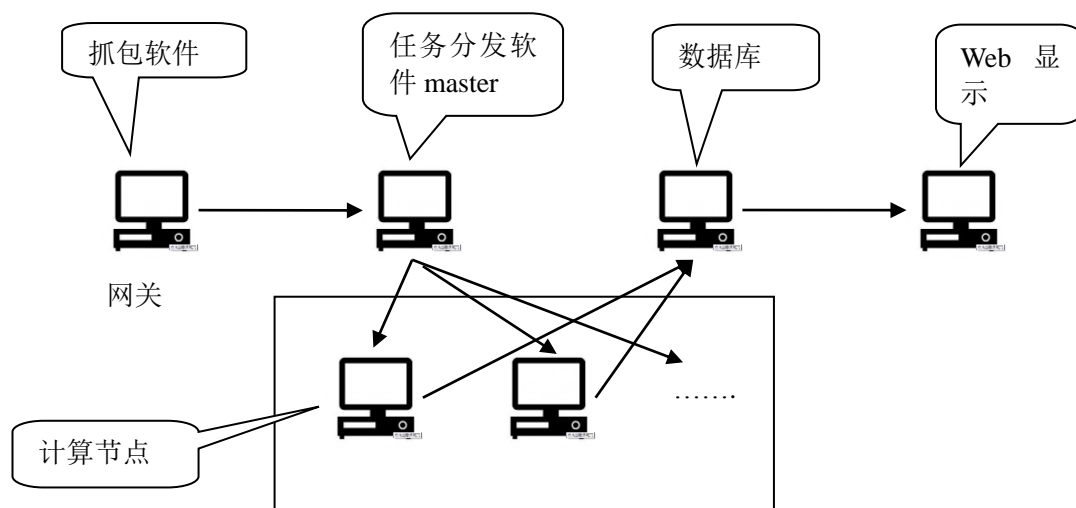


图 6-1 测试环境部署图

测试环境如图 6-1 所示。在测试中搭建了 6 个虚拟机，模拟了如图 6-1 所示的 6 台物理机器分布情况。6 台虚拟机上分别部署了抓包软件、节点管理程序 master 程序、数据库软件、节点运行程序 daemon 程序、前端网站。



每台虚拟机的硬件和软件配置相同，配置参数如表 6-1 和表 6-2。

表 6-1 硬件环境参数

硬件环境:	
内存	4gb
硬盘	20gb
处理器	1.7 ghz

表 6-2 软件环境参数

软件环境:	
操作系统版本	ubuntu kylin 14.04
nodejs 版本	node v0.12.5
redis 版本	redis-3.0.7
mongodb 版本	mongodb 3.2

### 6.3 报文模拟模块设计

为了记录 http 报文并在需要的时候发送报文，以满足系统测试所需。设计并实现了一个 http 报文记录和重发程序。

报文模拟系统是使用 nodejs 编写的，它的功能是记录 redis 通道中的消息，在需要的时候重新发送到指定通道。消息被记录在本地文本文件，使用字符串“(@\_@)”作为报文之间的分隔符号，将接收到的报文续写到文件尾端。

重发报文时，使用 nodejs 的读取文件，单次发送完成后调用 setinterval 命令将下一次发送事件设置为指定值之后触发。可以通过修改参数改变发送频率。

### 6.4 性能测试

在性能测试中，测试结果如表 6-3 所示，系统性能符合预期。在单 cpu 计算机处理时，在表 6-1，表 6-2 配置的计算机上，在 http 报文低于 20000/秒时，系统无明显延迟，前端刷新闻隔为 2s，redis 内存占用量稳定在 40%左右。http 报文达到 23000 条/秒后，系统开始出现较大延迟，redis 数据库占用内存急剧增加，前端显示刷新率超过 10s/次。在报文发送率超过 30000 条/秒后，内存占用达到 80%，前端失去响应。

表 6-3 性能测试结果

报文发送频率 (条每秒)	15000	20000	25000	30000	35000
实时显示数据平均刷新间隔 (s)	2.00	2.00	3.41	5.64	$+\infty$
redis 内存占用率	41%	43%	52%	86%	87%

## 6.5 功能测试

将模拟数据冲入流计算系统后，可以在前端看到统计结果。下面记录在冲入一天量的报文数据之后的系统状态。

(1) 内存数据状态：

图 6-2，显示了内存中所有表和集合的名字。

图 6-3、6-4 取一个哈希表、一个有序集合中的部分数据，查看不同类型的表中数据的状态。

```
127.0.0.1:6379> keys *
1) "stay.time.end"
2) "day.uv.zset"
3) "day.uv.ipSign.hash"
4) "urlerror.day.time.hash"
5) "day.user.agent.zset"
6) "day.most.refer.zset"
7) "search.day.keyword"
8) "day_ap_id_hash"
9) "day.uv.snSign.hash"
10) "urlerror.day.zset"
11) "day.most.visit.zset"
12) "day.agent.browser.zset"
13) "day_access_deep_zset"
14) "day.size.zset"
15) "search.date.list"
16) "day.ip.zset"
17) "search.month.keyword"
18) "urlerror.day.type.hash"
19) "stay.time.start"
20) "day.agent.dev.zset"
21) "end_time"
22) "search.week.keyword"
23) "day_ad_zset"
24) "day.pv.zset"
25) "urlerror.day.line.zset"
```

图 6-2 系统产生的所有内存数据表名字

```

127.0.0.1:6379> zrange day.agent.browser.zset 0 -1 withscores
1) "IE@7.0"
2) "2"
3) "Chrome@31.0.1650.63"
4) "3"
5) "Chrome@38.0.2125.111"
6) "3"
7) "undefined@undefined"
8) "8"
9) "UCBrowser@2.7.0.448"
10) "10"
11) "Chrome@36.0.1985.125"
12) "15"
13) "IE@9.0"
14) "19"
15) "Chrome@30.0.0.0"
16) "23"
17) "IE@8.0"
18) "62"

```

图 6-3 有序集合 day.agent.browser.zset 的状态

图 6-3 截取了部分有序集合 day.agent.browser.zset 的数据，可以看到有序集合在数据库中的状态。该表是浏览器使用计数表。浏览器 key 的格式是“浏览器@版本号”，跟在浏览器 key 后面的是该 key 的 value。

```

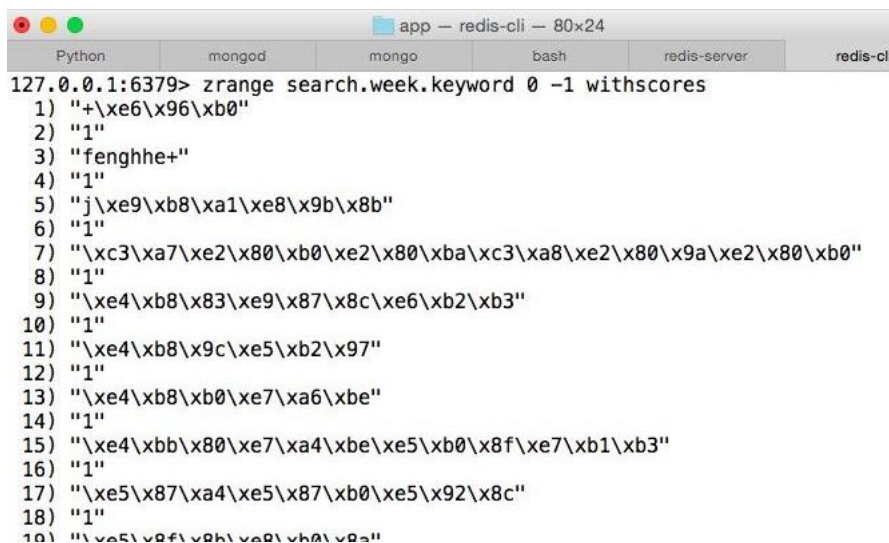
127.0.0.1:6379> hgetall stay.time.start
1) "42.91.49.116@ 31DB441D1B06E9B64D5C53CC31B53386"
2) "1414913912362.642"
3) "118.180.137.105@ FE476F5B5DE422CBD8AB02C4FD51413B"
4) "1414913915109.294"
5) "118.181.81.13@ F55184EA049A466A444F73933F6E1B65"
6) "1414913919803.061"
7) "42.90.131.42@ BEA4B490B7D13C811355BA551467F5D9"
8) "1414913921727.406"
9) "118.180.151.206@ F2793D5DF8106B22F9C8E6AED292887F"
10) "1414913922968.631"
11) "125.76.61.228@ 2636CC4BFDDE84732DEEAC95C3FBABF1"
12) "1414913932069.377"
13) "221.7.41.105@ 9B9AD84E91F54A8CD5B10FE957AC5ACC"
14) "1414913935082.816"
15) "125.74.78.255@ AB6B8AC8905B69673D54FB9A2E38D8D4"
16) "1414913942016.735"

```

图 6-4 有序集合 stay.time.start 的状态

图 6-4 截取了部分哈希表 stay.time.start 的数据。stay.time.start 表是用户停留时间统计的起始时间记录。用户标识的 key 的格式是“ip 地址@sessionid”，跟在用户名后的的是该 key 的 value 值，是一个时间的浮点数格式。

图 6-5 截取了部分关键字记录，中文关键词在内存中使用 gbk 编码存放。在发往网站后端时保持 gbk 字符串格式，在网站前端解码。



```

app -- redis-cli -- 80x24
Python mongod mongo bash redis-server redis-cli
127.0.0.1:6379> zrange search.week.keyword 0 -1 withscores
1) "+\xe6\x96\xb0"
2) "1"
3) "fenghhe+"
4) "1"
5) "j\xe9\xb8\xa1\xe8\x9b\xb"
6) "1"
7) "\xc3\xa7\xe2\x80\xb0\xe2\x80\xba\xc3\xa8\xe2\x80\x9a\xe2\x80\xb0"
8) "1"
9) "\xe4\xb8\x83\xe9\x87\x8c\xe6\xb2\xb3"
10) "1"
11) "\xe4\xb8\x9c\xe5\xb2\x97"
12) "1"
13) "\xe4\xb8\xb0\xe7\xa6\xbe"
14) "1"
15) "\xe4\xbb\x80\xe7\xa4\xbe\xe5\xb0\x8f\xe7\xb1\xb3"
16) "1"
17) "\xe5\x87\xa4\xe5\x87\xb0\xe5\x92\x8c"
18) "1"
19) "\xe5\x8f\x8b\xe8\x8b\xe8"

```

图 6-5 内存数据 keyword 的状态

## (2) 前端显示状态:

下面是显示模块（网站）的显示结果。

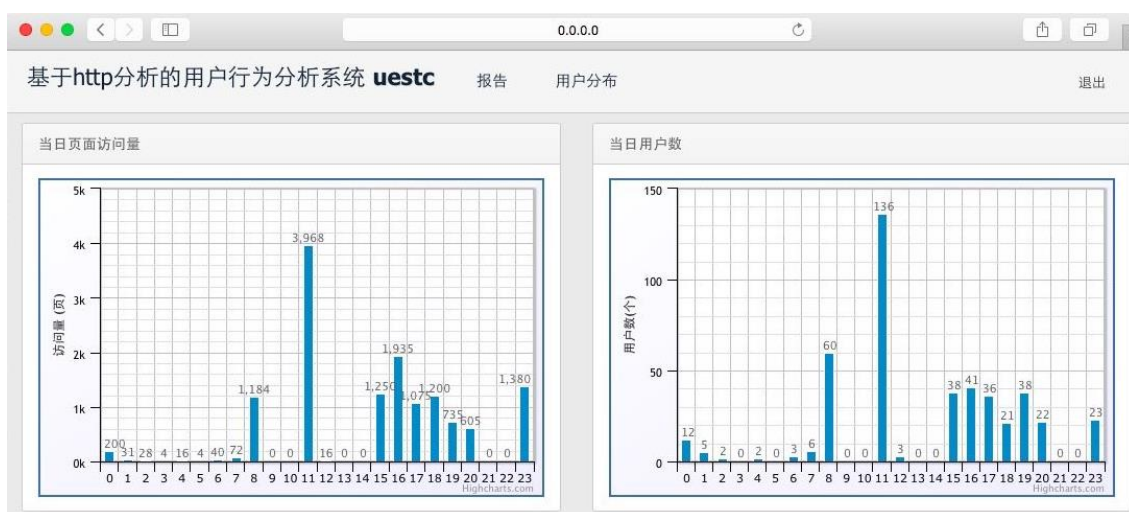


图 6-6 用户访问量与新用户数统计

(左)用户访问量统计; (右)新用户数统计

图 6-6 (左) 为一天的访问量测试图表。使用柱状图显示，每个小时的访问量显示为一个柱面。可以看到，在柱状图中，电商网站这一天的访问量以波浪状变化，有三个波峰，分别在上午的 11 点和下午的 4 点钟以及晚上 11 点。其中，上午 11 点左右的访问量达到一天的最高峰。这一天的访问量统计结果和生活经验比较一致。

图 6-6（右）看到新用户数量的统计结果，用柱状图显示。可以看到，在这一天里，用户数量以波浪状变化，有三个波峰，分别在上午的 11 点和下午的 4 点钟以及晚上 11 点。其中，上午 11 点左右的新用户数量达到一天的最高峰。这个结果和上图的用户访问量结果相吻合。在用户数量的统计逻辑中，一天之内相同的用户只会在第一次访问时统计，下午的总用户数量可能高于新用户数量。

图 6-7 是用户来源网站与用户访问最多页面的统计表。这两项统计结果用表显示。



图 6-7 用户访问最多页面与来源网站统计

(左)用户访问页面频率；(右)用户来源网站

侧边栏菜单的显示效果测试：

图 6-8 是用户访问深度的统计信息，原始信息以列表的形式显示在原页面下方的表格中。同时在折线图中统计用户访问深度日平均值的统计信息。



图 6-8 平均访问深度测试



图 6-9 是用户平均停留时间测试结果线图,可以看到,在测试中用户平均访问深度和用户平均停留时间在相同时段正比关系。



图 6-9 平均停留时间测试

图 6-10 是用户关键词信息。在关键词页面中左下方的表格中做出排序显示,同时在上方的词云图中以直观的词云形式显示。词云图是一个动态旋转的隐约球形。在下方的搜索词排名中可以看到“鸡蛋”的搜索次数最多,这和我们后来所知的当时该电商网站的促销活动相吻合。用户的关键字搜索分布一定程度代表了用户的兴趣分布。在后续分析工作当中可以对关键字的分布做出更进一步的统计。比如关联统计,可以按照地域的划分来统计用户的关键字搜索频率。然后从结果上来推测不同地区的人的关注重点。可以在网站的策略规划中起到方向性的指引。



图 6-10 用户访问关键词统计

图 6-11 是流量统计测试截图。用户的流量信息显示分为两个窗口，都是实时的线图。上方的线图显示当天按小时划分的流量统计，下方的线图显示当前 10 秒的实时数据流量。当系统运行时，上图的坐标系保持不变，始终显示一天的 24 小时的表项，其中当前小时对应的数据实时更新，下图坐标系和数据同步变化，显示当前的流量。

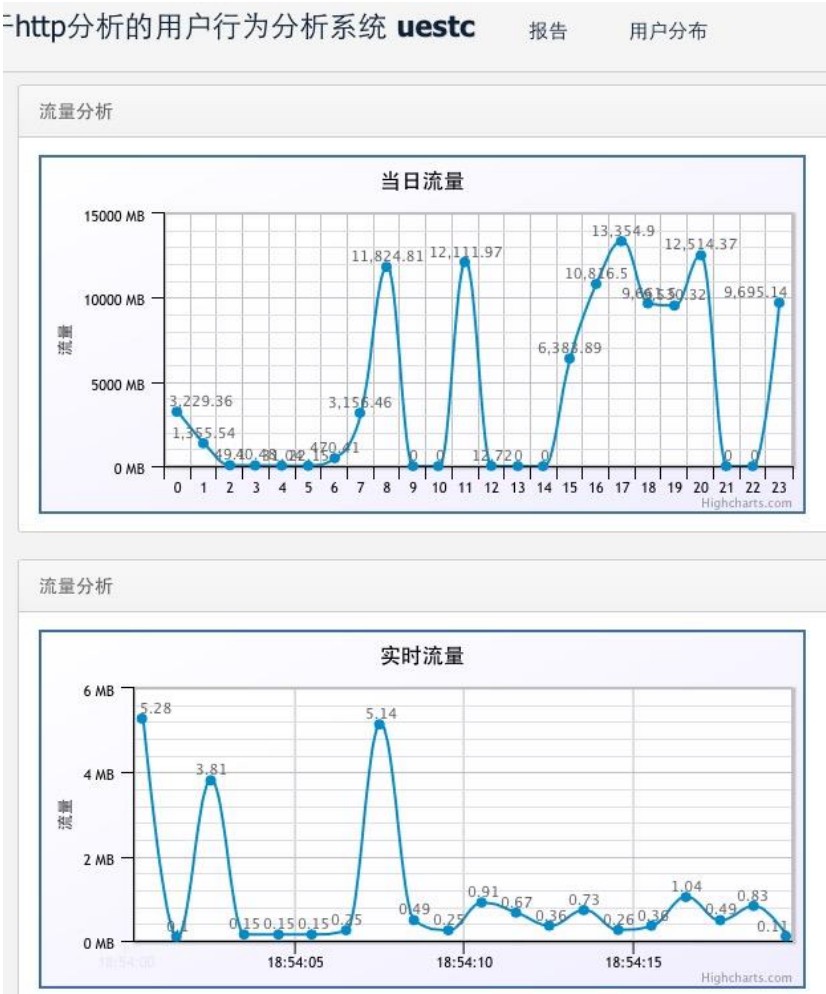


图 6-11 日总流量统计与实时流量统计  
(上)当日总流量统计；(下)实时流量图

如图 6-12 是浏览器使用频率统计。用户浏览器类型是通过分析 http 报文中的 user-agent 字段得到的。用户的浏览器比例分类以双饼图的形式显示，双饼图的内层是该浏览器的品牌，双饼图的外层是浏览器型号的细分。可以看到当天访问该网站的用户使用最多的是 ie 浏览器，其次是谷歌浏览器。在浏览器的版本分布中，

从谷歌浏览器和 ie 浏览器都可以看出发布一段时间的稳定版本使用者最多，最新版本使用者略少，历史版本使用数随时间间隔的增大而减少。



图 6-12 用户浏览器统计

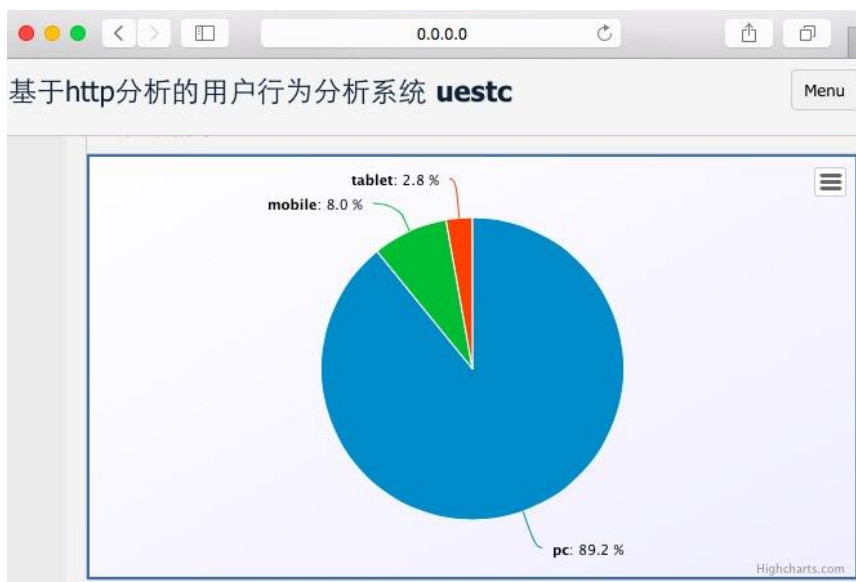


图 6-13 用户设备统计

图 6-13 是用户的设备信息统计，可以看到当天访问该电商网站的用户使用最



多的设备是计算机，其次是手机，平板电脑使用的人最少。



图 6-14 用户地理分布

图 6-14 是用户地理分布的统计结果。使用用户的 ip 分析得到的用户地点有一定的误差，在显示地图时为了有一个直观的印象，将同一个坐标上的多个标记点随机打在地图上该点附近的小片区域。在作图时只做中国版图上的用户统计图。由地图显示可以看出，用户的分布和电商的位置有很大相关性。由于试验时取得的数据是兰州的电商网站数据，可以看到兰州这一片区域的用户数量最多，其次是北京成都等城市。地图部分的显示会加载一个 3.2mb 的 js 文件，根据网络状况的不同，会有不同情况的延迟，在 10m 网速下，延迟在 5s 左右。第二次以后的刷新时不再有延迟。

前端所有页面实时显示正常。同时，各页面上查看到的用户行为数据，经过计算，符合以下关系：

**pv 数和 uv 数：**任意时段 pv 数大于等于 uv 数

**来源链接数量和 uv 数：**任意时段，来源网站计数小于等于 uv 数。

**平均访问深度\*uv 数和 pv 的数：**任意时段平均访问深度乘以 uv 数和 pv 的数值相等。

## 6.6 本章小结

本章列出了本文所构建的用户行为分析系统的测试结果。

测试使用的数据为真实电商网站数据。

内存数据测试中，查看了内存中各种数据表的状态，redis 数据库内存占用量，内存用量和数据准确性符合预期。

网站显示测试的目标有用户访问数量、新用户数量、用户访问深度、停留时间、搜索关键词、使用浏览器、用户设备、用户地理分布的显示状况。

手动点击测试中，系统正确性达到预期。

在压力测试中，系统性能达到预期。

## 第七章 全文总结与展望

### 7.1 全文总结

本文完成了一个基于 http 的流数据分析的用户行为信息分析系统。这套系统可以实时分析传入的 http 数据流，得到用户访问网站的基础数据，进行实时显示。统计结果的历史数据持久化在 mongodb 数据库中。通过前面章节的讲述，可以清楚的看到系统工作的原理，各项用户数据分析的依据和流程。本文最主要的贡献有五个：

(1) 本文设计并实现了一个流数据分析框架。该框架可以搭建流计算节点，完成流计算分析任务。这套框架使用一个通道作为流数据的入口，接收到流数据后，消息到达事件被激活，流计算分析节点开始工作。计算节点事实分析得到的数据，分析产生的结果实时存入数据库。分析节点可以随时灵活的增减，节点的增减不影响其它节点的正常工作。流计算模块之间的信息通过发布/订阅的消息通道进行交互。

(2) 本文设计了一个用户行为分析节点集群，可以从 http 数据中过滤出用户访问量、新用户、用户访问深度、停留时间等用户行为数据。本文一共设计了 7 个数据分析节点。数据分析分多个级别同步进行，多节点并行工作。数据分析节点之间具有关联性，用户访问深度和用户停留时间的分析节点的工作建立在从用户访问量分析节点传递过来的流数据上。

(3) 本文提出了一种结合 set 数据结构，过滤 http 报文，确定用户访问量的算法。在这个算法的基础上可以得到确定新用户数的算法。

(4) 本文提出了一种结合 hash 列表，构建循环计数队列，实现一个滑动窗口，统计一段时间内用户访问深度的算法。

(5) 本文提出了通过两个 hash 列表，结合定时器，统计用户一段时间内平均停留时间的算法。

### 7.2 后续工作展望

http 用户行为分析是一个有价值有潜力的研究领域。在本文完成的系统的基础上进行后续开发，我认为有两个方向：

(1) 一是增加数据分析节点，本文已经构建的流计算系统的框架十分清晰，接口简单易用。可以非常方便的增加流计算分析节点。在未来如果需要更加细致

的分析要求（如精确到分秒的分段分析统计），或增加新的函数，或者增加新的用户行为的分析目标等，都可以通过现有接口，直接将新的分析节点挂在系统上。

（2）二是数据使用方式的扩展。在后续开发中，或可增加交叉分析、多项对比、同项不同时间对比等数据分析项目。可以在现有数据基础上，使用数据挖掘方法，多维分析用户的使用习惯，比如不同地区的用户浏览该网站的时间段（上午，下午，晚上）、停留时间长短、访问深度与用户设备的关系、老用户和新用户的比例等等。

## 致 谢

在攻读硕士学位期间，首先衷心感谢我的导师刘均教授。他严谨的科学态度，认真严肃的治学作风深深感染和激励着我。刘均导师不仅在学业上对我有着巨大的帮助，在工作和生活上也指引着我，给我无微不至的关怀。在此谨向刘均教授致以最真挚的谢意和敬意。

感谢与我同窗的学长学姐们，尤其是高强师兄和朱子博师兄，他们在我开始硕士生涯的第一天起就对我的生活和学习的方方面面提供了无微不至的照顾。为我提出了很多有用的建议。

感谢和我一起工作和研究的伙伴们赵立斌、杨乐、徐加文，我们一起开发项目，攻克难点，互相协作，度过了充实三年。我们互相学习，互相鼓励，是共同的战友。

感谢同寝室一起度过了三年时光的室友刘志伟、蒋志国、胡晓，你们的陪伴让我感到快乐。

感谢这篇论文所引用和涉及到的所有学者，各位学者的研究成果和文献，给了我巨大的启发。

最后，再一次感谢所有在硕士生涯中帮助过我的良师益友，以及被我引用或参考过的论著的作者。衷心的感谢大家。

## 参考文献

- [1] 李秀龙. 基于网络流量监测与预测的用户流量行为分析方法研究[d]. 北京工业大学, 2013.
- [2] 郑纬民.从系统角度审视大数据计算[j];大数据;2015 年 01 期
- [3] 戴声;肖建明;王波;;大规模数据中心监控数据并发处理[j];计算机与数字工程;2014 年 12 期
- [4] 崔星灿;禹晓辉;刘洋;吕朝阳;;分布式流处理技术综述[j];计算机研究与发展;2015 年 02 期
- [5] 张鹏;李鹏霄;任彦;林海伦;杨嵘;郑超;;面向大数据的分布式流处理技术综述[j];计算机研究与发展;2014 年 s2 期
- [6] 黄军;社交网络热点话题公众情感极性实时计算研究[d];杭州电子科技大学;2015 年
- [7] 邵宇. web 数据挖掘技术在电子商务中的应用研究[j]. 中国新技术新产品, 2016(2):21-21.
- [8] 鲍峥嵘,王永成,刘功申,韩客松;一种快速的字串交叉模式匹配算法[j];上海交通大学学报;2003 年 03 期
- [9] 谭敏生;汤亮;基于 http 的网络数据包还原技术研究[j];计算机技术与发展;2007 年 06 期
- [10] 陈潇潇;基于 linux 的虚拟主机信息监控系统的设计与实现[d];哈尔滨理工大学;2005 年
- [11] 邓友良;http 报文监测和过滤技术研究[d];西南交通大学;2007 年
- [12] 马黎;智能 web 过滤系统的研究与设计[d];东华大学;2009 年
- [13] 沈凤仙;一个 web 文本过滤系统设计与实现[d];苏州大学;2009 年
- [14] 乐妍;基于 http 协议面向中文文本的过滤技术研究[d];四川师范大学;2009 年
- [15] 潘拓宇;融入用户行为上下文的个性化推荐模型[d];湘潭大学;2010 年
- [16] 张乐;基于本体的上下文感知计算的研究与应用[d];武汉理工大学;2011 年
- [17] 黄卫平;个性化搜索引擎的研究与实现[d];武汉理工大学;2011 年
- [18] 孙燕花;基于聚类的网络用户行为分析[d];中南大学;2011 年
- [19] 李晓辉;基于用户行为分析的数据挖掘系统研究与设计[d];北京邮电大学;2011 年
- [20] 黄碗明;基于数据挖掘的社区网站用户行为分析系统[d];南京邮电大学;2012 年
- [21] 张乐媛;基于上下文感知的网络用户行为分析[d];北京邮电大学;2010 年
- [22] 王霞;基于 web 浏览的用户行为分析系统的研究与设计[d];北京邮电大学;2010 年
- [23] 周岳;基于兴趣分类的用户行为分析系统的研究与设计[d];北京邮电大学;2010 年
- [24] research of an improved web visit statistical method applying in user's interest degree[a];proceedings 2010 ieee 2nd symposium on web society[c];2010 年
- [25] 徐志明;宋毅;冯子威;李生;一种基于分类的用户兴趣模型[a];第六届全国信息检索学术会议论文集[c];2010 年
- [26] 常光辉;大规模分布式可信监控系统研究[d];重庆大学;2011 年

- [27] m.s.shang,g.x.chen,s.x.dai,b.h.wang,t.zhou,interest-drivenmodelforhumandynamics,chinese physics letters 27(4)(2010)048701.
- [28] grabowski, human behavior in online social systems, the european physical journal b—condensed matter and complex systems 69(4)(2009) 605–611.
- [29] t.zhou,h.a.t.kiet,b.j.kim,b.h.wang,p.holme,role of activity in human dynamics, europhysics letters 82(2)(2008)28002.
- [30] q.yan,l.r.wu,l.l.yi, research on the human dynamics in the mobile communities based on social identity, discrete dynamics in nature and society 2012(2012)672756.
- [31] k.christine,b.martin, identification of influencers—measuring influence in customer networks, decision support systems 46(1)(2008)233–253.
- [32] k.christine,b.martin, leveraging network effects for predictive modeling in customer relationship management, in: proceedings of the 15th annual workshop on information technologies & systems las-vegas usa, 2005, p. 78.
- [33] x.cyang,x.h.zhang,w.h.shi, the impact of relational attributes on mobile phone user's communication behavior, journal of bupt (social science edition) 10(5)(2008)43–47.
- [34] grabowski,n.kruszewska,r.a.kosiński, properties of on-line social systems, the european physical journal b 66(1)(2008)107–113.
- [35] z.d.zhao,h.xia,m.s.shang,t.zhou, empirical analysis on the human dynamics of a large-scale short message communications system, chinese physics letters 28(6)(2011)068901.

## 攻读硕士学位期间取得的成果

- [1] IBM 大型主机竞赛，个人赛一等奖，2013 年
- [2] Liu Di, Pan Dong, 《MESSAGE TRANSPORT SYSTEM OF DISTRIBUTED STREAM COMPUTING》, ei accession no.: 20151700788242, iccwamtip 2014, p 483-486, march 30, 2014
- [3] IBM 大型主机竞赛，个人赛一等奖，2014 年
- [4] IBM 大型主机竞赛，个人赛一等奖，2015 年