 单位代码 **10006**

学号 **17373281**

分类号 **TP311.5**

****

毕业设计(论文)

网站访客唯一性识别和智能流控方案的设计与实现

|  |  |
| --- | --- |
| 学院名称 | 软件学院 |
| 专业名称 | 软件工程 |
| 学生姓名 | 江一帆 |
| 指导教师 | 吕云翔 |

**2021**年**6**月

网站访客唯一性识别和智能流控方案的设计与实现

学生：

江一帆

指导教师：

吕云翔

摘要

随着互联网的发展与移动设备的普及，一位用户可能使用多台不同的设备或经由不同的浏览器访问相同的网站。同时，网络爬虫也被频繁地用于获取网络数据，可能导致网站核心内容被复制等问题。清华大学出版社作为国内领先的综合性教育与专业出版机构，其网站在用户的唯一性识别以及反恶意爬虫等方面都有较高的需求。本课题的目标是根据清华大学出版社的实际需求，设计并实现一个识别用户身份与爬虫，并进行网站流量控制的方案。通过IP地址、浏览器Fingerprint等信息对访客进行标记，通过接口请求频率以及身份标识一致性对爬虫进行识别，将用户、爬虫、用户表示信息等存储在前后端数据库中。通过对访问频率过高的用户设置请求限制的方式，达到网站流量控制的目的。

关键词：计算机

网站，反爬虫，访客识别，流量控制

The Design and Realization of Website Visitor Recognition and Smart Website Traffic Control

Author :

Jiang Yi-fan

Tutor :

Lu Yun-xiang

Abstract

With the development of the Internet and the popularization of mobile devices, a user may visit a same website through several different devices or different browsers. At the same time, web crawlers are also frequently used to obtain data from websites, which may cause problems such as leakage of the core content of the website. As a leading comprehensive and professional publishing institution in China, Tsinghua University Press has a high demand in unique user identification and anti-malicious-crawler. The purpose of this project is to design and implement a scheme to identify user and crawler, as well as controlling website traffic according to the actual needs of Tsinghua University Press. Users are marked by IP address, browser Fingerprint and other information, crawlers are identified by interface request frequency and identity consistency. User, crawler and user identity information are stored in front-end and back-end database. Website traffic control is achieved by setting request limitation for users with high request frequency.

Key words:

Computer Website, Anti-Crawler, Website Visitor Recognition, Website Traffic Control

目录

[1 绪论 1](#_Toc69416463)

[1.1 课题背景与意义 1](#_Toc69416464)

[1.2 国内外研究现状 1](#_Toc69416465)

[2 系统设计与技术选型 3](#_Toc69416466)

[2.1 系统设计 3](#_Toc69416467)

[2.1.1 需求分析 3](#_Toc69416468)

[2.1.2 系统整体设计 6](#_Toc69416469)

[2.2 技术选型 8](#_Toc69416470)

[2.3 项目任务 11](#_Toc69416471)

[3 项目具体实现 12](#_Toc69416472)

[3.1 用户识别部分 12](#_Toc69416473)

[3.2 爬虫识别部分 14](#_Toc69416474)

[3.3 请求限制部分 15](#_Toc69416475)

[参考文献 16](#_Toc69416476)

# 绪论

## 课题背景与意义

随着互联网的发展与移动设备的普及，互联网用户及终端设备数量呈现爆发式的增长。一位用户可能使用多台不同的设备或经由不同的浏览器访问相同的网站。

除此之外，出于对获取各类网站上大量有价值信息的需要，网络爬虫也被大量地用于获取网络数据。根据统计，2019年，恶意爬虫的流量占所有互联网流量的24.1%，上升到有史以来最高的百分比，而37.2%的互联网流量是非人为带来的[1]。对于网站而言，恶意爬虫可能会导致网站核心内容被复制、注册用户被扫描、网站带宽负担加重等问题。

论文课题由北京航空航天大学软件学院吕云翔老师提出，来源于吕老师于清华大学出版社联系到的对于识别用户唯一性、识别爬虫并进行接口请求限制的实际需求。清华大学出版社成立于1980年6月，是教育部主管、清华大学主办的综合性大学出版社，现年出版图书、音像制品、电子出版物等近3000种。作为国内领先的综合性教育与专业出版机构，清华大学出版社的网站在用户的唯一性识别以及反恶意爬虫等方面都有较高的需求。

本课题的目标是根据清华大学出版社提出的保障网站安全性与性能的实际需求，结合国内外研究现状和发展状态，设计并实现一个识别用户身份与爬虫，并进行网站流量控制的方案，希望通过对访客进行标记以及对恶意爬虫进行识别，对恶意用户与爬虫设置访问限制，达到智能流量控制的目的。

## 国内外研究现状

本课题所涉及到的需求主要可以分为三部分，分别是访客唯一性识别、恶意爬虫识别以及网站流量调控。

在访客唯一性识别方面，服务器产生的Web日志可以成为良好的用户识别数据源，因为Web日志中包含了访问主机IP、访问时的服务器时间、访问页面、请求方式等信息。其中，HTTP协议中包含的User Agent字段能够识别访客使用的操作系统及版本、CPU类型、浏览器及版本等。由于MAC地址是网卡物理地址，与网络无关，通过在Web日志中加入MAC地址，还可以实现用户的唯一性识别，增加用户识别的准确性本课题所涉及到的需求主要可以分为三部分，分别是访客唯一性识别、恶意爬虫识别以及网站流量调控。

在访客唯一性识别方面，服务器产生的Web日志可以成为良好的用户识别数据源，因为Web日志中包含了访问主机IP、访问时的服务器时间、访问页面、请求方式等信息。其中，HTTP协议中包含的User Agent字段能够识别访客使用的操作系统及版本、CPU类型、浏览器及版本等。由于MAC地址是网卡物理地址，与网络无关，通过在Web日志中加入MAC地址，还可以实现用户的唯一性识别，增加用户识别的准确性[2]。

在爬虫识别方面，妥协型的方法有设置Robots.txt协议。设置Robots.txt文件，来告诉一些规范的爬虫使用者页面的爬取规则[3]。其他比较常用的主动型方法有设置单位阈值，倘若某一IP、User Agent标识超出了阈值，对其实施监控；对HTTP请求头的每个属性进行“是否常规访问”的判断[4]；设置cookie检测等。基于滑动时间窗的爬虫实时检测方法提高了对不符合爬虫规则的检测的准确性和效率。此外，还有基于机器学习的爬虫识别，通过提取会话的身份信息、日志的统计信息、日志中的异常信息进行模型训练，并使用训练后的模型通过会话信息识别爬虫[6]。

在智能流控方面，许多爬虫可能会通过使用代理的方式来避开IP识别[7]，当遭受来自无法产生大量流量的国家的访问（通常外国访问者使用一个网站是没有意义的），可以限制来自该国家的流量。重定向到验证页面也是十分有效的方法，常见的验证方式有验证码、短信验证、邮箱验证。使用Honeypot陷阱可以跟踪许多种类的爬虫，并且通过一个简单的规则列表就可以将请求分类为恶意或非恶意[8]。

(undone,待扩展

# 系统设计与技术选型

## 系统设计

### 需求分析

本项目的需求主要集中在数据的收取与处理方面，可分为如下几部分：记录用户标识数据、标记用户、记录用户请求数据、标记爬虫以及设置接口访问限制。项目的用例图如图2.1所示。

（undone，具体说明用例）



图2.1 项目用例图

记录用户浏览器数据的用例说明见表2.1。

表2.1 记录用户身份数据

|  |  |
| --- | --- |
| 用例名称 | 记录用户身份数据 |
| 简要描述 | 该用例描述了系统记录用户身份数据的过程 |
| 参与者 | 网站管理员 |
| 涉众 | 用户 |
| 前置条件 | 用户进入到网站 |
| 后置条件 | 记录成功后进行用户标记 |
| 基本事件流 | 1. 用户进入网站 2. 系统收集用户身份数据，包括useragent、ip地址、fingerprint等。（A-2） |
| 备选事件流 | A-2 存在无法获取的数据   1. 以undefined代替 |
| 补充约束 | 无 |
| 待解决问题 | 无 |

标记用户的用例说明见表2.2。

表2.2 标记用户

|  |  |
| --- | --- |
| 用例名称 | 标记用户 |
| 简要描述 | 该用例描述了系统根据获取的数据标记用户的过程 |
| 参与者 | 网站管理员 |
| 涉众 | 无 |
| 前置条件 | 已经获得用户身份数据 |
| 后置条件 | 标记成功后保存到数据库中 |
| 基本事件流 | 1. 传入收集到的用户浏览器等信息 2. 根据数据区分用户后，存入IndexedDB中，并同步到MongoDB。（B-2） |
| 备选事件流 | B-2 同一用户已存在   1. 检查数据是否重复 2. 添加不重复的新数据 |
| 补充约束 | 无 |
| 待解决问题 | 无 |

记录用户请求数据的用例说明见表2.3。

表2.3 记录用户请求数据

|  |  |
| --- | --- |
| 用例名称 | 记录用户请求数据 |
| 简要描述 | 该用例描述了系统记录用户请求数据的过程 |
| 参与者 | 网站管理员 |
| 涉众 | 用户 |
| 前置条件 | 用户进入到网站 |
| 后置条件 | 记录成功后进行爬虫标记 |
| 基本事件流 | 1. 用户进入网站 2. 系统收集用户请求数据，包括请求的header、请求的频率等。（C-2） 3. 计算请求的频率 |
| 备选事件流 | 无 |
| 补充约束 | 无 |
| 待解决问题 | 无 |

标记爬虫的用例说明见表2.4。

表2.4 标记爬虫

|  |  |
| --- | --- |
| 用例名称 | 标记爬虫 |
| 简要描述 | 该用例描述了系统根据获取的数据标记爬虫的过程 |
| 参与者 | 网站管理员 |
| 涉众 | 无 |
| 前置条件 | 已经获得用户请求数据 |
| 后置条件 | 标记成功后保存到数据库中 |
| 基本事件流 | 1. 传入收集到的用户请求信息 2. 根据信息进行频率、合法性等检查以区分爬虫 3. 区分爬虫后，存入MongoDB中。（D-2） |
| 备选事件流 | D-2 同一用户已存在   1. 检查数据是否重复 2. 更新请求频率、访问限制等信息 |
| 补充约束 | 无 |
| 待解决问题 | 无 |

设置访问限制的用例说明见表2.5。

表2.5 设置访问限制

|  |  |
| --- | --- |
| 用例名称 | 设置访问限制 |
| 简要描述 | 该用例描述了系统根据爬虫标记限制访问的过程 |
| 参与者 | 网站管理员 |
| 涉众 | 无 |
| 前置条件 | 已经完成爬虫标记 |
| 后置条件 | 无 |
| 基本事件流 | 1. 传入需要限制的用户数据。 2. 根据数据对请求进行限制。（E-2） |
| 备选事件流 | E-2 同一用户已存在   1. 更新限制信息 |
| 补充约束 | 无 |
| 待解决问题 | 无 |

### 系统整体设计

用户识别部分采用IP地址与浏览器Fingerprint相结合的方式。将同Fingerprint不同IP地址的字段进行整合，看作是同一用户，用以解决IP池的问题。通过timezone、fonts等其他信息可以做更精确的用户区分。

数据以JSON的形式保存在NoSQL型数据库中，前端为IndexedDB，后端为MongoDB。

在后端采用Node.js，对MongoDB的增删改查操作进行了封装，便于数据的整理。

对于爬虫检测的部分，采用设置阈值、HTTP请求检测的方式。在Node.js端对容易被频繁起请求的接口处接入爬虫检测。通过记录时间段内的请求次数以及最近的访问时间，将访问过于频繁的用户进行标记。对于请求中User Agent、platform等信息与记录不匹配的用户也进行标记。将受标记的用户与白名单用户进行匹配，去除需要放行的用户后，对于其他受标记的用户进行频率与流量速率限制。

为了加快请求响应速度与访问限制的灵活性，将影响严重的用户加入Nginx的blockip配置文件中，并适时进行更新。

在访问限制部分，过时的用户代理、过时的浏览器版本、不合理的海外IP、不合理的Referer等会被特别标记，在需要时可以在这些方面设置更为严格的限制策略。

（undone，细化）

系统整体的流程图如图2.2所示。

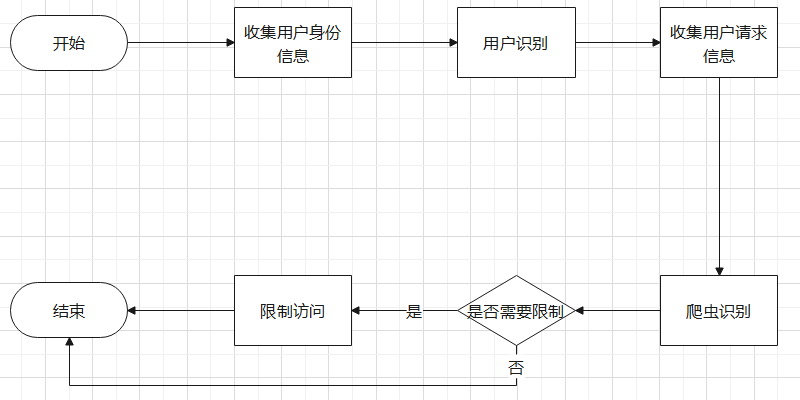


图2.2 系统流程图

系统架构图如图2.3所示。

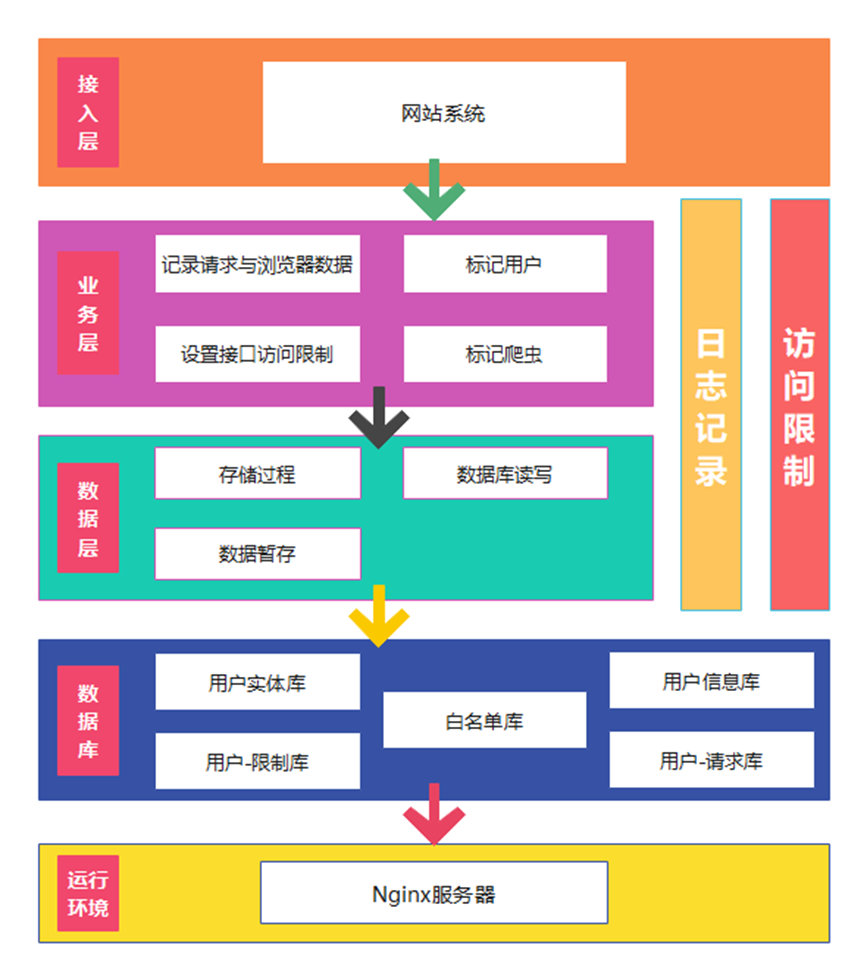


图2.3 系统架构图

## 技术选型

（undone，去重）

在项目过程中使用了一些第三方库，包括ClientJS、Dexie等。主要通过CDN与NPM的方式引入和管理依赖。

CDN 的全称是 Content Delivery Network，即内容分发网络。使用CDN引用的一个好处是可以从缓存中加载 JQuery，减少加载时间。同时，大多数 CDN 都可以确保当用户请求文件时，会从离用户最近的服务器上返回响应，这样也可以提高加载速度。

NPM全称为 Node Package Manager，是随同Node.js一起安装的包管理工具，能解决Node.js代码部署上的很多问题。最初用来作为Node.js的包管理器，但是随着其它构建工具的发展，NPM已经可以用来安装、管理和分享JavaScript包，同时会自动处理多个包之间的依赖。

在区分不同用户时，用到了不同的浏览器指纹作为区分依据。浏览器指纹可以简单分为普通指纹和高级指纹，普通指纹可以理解为容易被发现并且容易修改的部分，例如HTTP的Header。高级指纹则几乎可以直接确定一个独一无二的浏览器身份，例如Canvas指纹、AudioContex指纹。上述指纹都是从同一个浏览器上获得，很多特征值都是不稳定的，例如User Agent、Canvas指纹在相同设备的不同浏览器打开会不尽相同。

跨浏览器指纹Fingerprint就是即便是在不同浏览器上也可以取得相同或者近似值的稳定浏览器特征。Fingerprint是通过获取浏览器具有辨识度的信息，进行一些计算得出的值。特征值可以是UA、时区、地理位置或者是使用的语言等。这些特征值具有不同的信息熵，信息熵大的特征值对于唯一确定一位用户更有作用。而将指纹信息综合起来，可以大大降低碰撞率，提高客户端uuid的准确性。因此往往会在综合特征值计算fingerprint时，给信息熵较大的特征值更大的权重。

选择前端数据存储时考虑了LocalStorage、WebSQL与IndexedDB。虽然LocalStorage是用key-value键值的模式存储数据，但它都是通过字符串形式进行数据的存储，而且LocalStorage的容量 仅在 2.5MB 到 10MB 之间，并且不提供搜索功能，不能建立自定义的索引，因此对于数据的查找并不方便。而WebSQL已经停止维护，只有Chrome还支持WebSQL，因此采用了更为兼容的IndexedDB。IndexedDB为NoSQL型数据库，支持查找、建立索引等。

NoSQL是非关系型的数据库，也称作Not Only SQL，是对不同于传统的关系型数据库的数据库管理系统的统称。NoSQL用于超大规模数据的存储，这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

由于IndexedDB的API十分复杂，并且异常处理并不完善，因此使用Dexie封装的API来简化对IndexedDB的操作。

用户请求数据的获取、整理以及用户请求限制的部分主要在服务端完成。后端采用Node.js，采用CommonJS标准并使用Express框架帮助开发Node.js，服务器使用Nginx。

Nginx是一款是由俄罗斯的程序设计师开发的高性能轻量级Web和反向代理服务器，也是一个IMAP/POP3/SMTP邮件代理服务器。Nginx 可以在大多数 Unix Linux OS 上编译运行，特点是占有内存少、并发能力强。Nginx的并发能力在同类型的网页服务器中表现较好，也因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。在高连接并发的情况下，Nginx是Apache服务器不错的替代品。

Node.js 是一个开源与跨平台的 JavaScript 运行时环境。得益于Node.js，为浏览器编写 JavaScript 的前端开发者除了客户端代码之外还，可以编写服务器端代码而无需学习完全不同的语言。Node.js 可以在一台服务器上处理数千个并发连接而无需引入管理线程并发的负担。在 Node.js 中还可以使用新的 ECMAScript 标准。

Express 是一个保持最小规模的灵活的 Node.js Web 应用程序开发框架，是目前最流行的基于Node.js的Web开发框架，为 Web 和移动应用程序提供一组强大的功能。Express 提供精简的基本 Web 应用程序功能，同时保留了Node.js 功能。

CommonJS 是开发Node.js应用的模块规范。Node.js中每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见。CommonJS规范规定，每个模块内部，module变量代表当前模块。这个变量是一个对象，它的exports属性是对外的接口。CommonJS地优点是所有代码都运行在模块作用域，不会污染全局作用域。模块可以多次加载，加载的顺序就是模块在代码中出现的顺序。但是模块只会在第一次加载时运行一次，然后缓存运行结果，以后直接读取。要想让模块再次运行，必须清除缓存。

在后端的数据库选择上，为了与前端数据库IndexedDB匹配，采用了同为Nosql型数据库的MongoDB。MongoDB 是一个基于分布式文件存储的数据库，旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。通过Node.js控制对MongoDB的增删改查等操作。

使用GitHub完成项目代码、说明文档和版本的管理。

## 项目任务

本项目主要包括如下几个任务：

* + - 1. 分析清华大学出版社提出的网站访客识别和智能流控具体需求。
      2. 进行需求建模，设计方案的架构与数据库
      3. 对网站获取到的访客数据进行提取，整理系统需要的信息。
      4. 在提取的信息中进行访客身份的识别，并在数据库中标记。
      5. 设置合适的规律判断方法，对爬虫进行识别。
      6. 根据流量控制要求，对频繁访问的接口的请求次数与频率进行限制。
      7. 交付清华大学出版社运行测试。

# 项目具体实现

## 用户识别部分

为了实现跨浏览器识别，最为直接的方法是获取到用户设备的唯一标识，这样就可以解决用户在同一台设备的不同浏览器上访问网站的识别问题。MAC地址就是这样一种设备的唯一标识，但是由于MAC地址只能在IE浏览器中的ActiveX控件中获得，IE浏览器的普及性与使用率十分低，并且一般情况下网站并没有方法获取到准确的用户设备的MAC地址，便放弃MAC地址识别的方法，转而使用一些可以通过JavaScript等方式获得的用户特征值作为识别用户的依据，也就是浏览器追踪技术。

（undone，去重）

浏览器追踪技术到目前已经进入3代。第一代是状态化的，主要集中在用户的cookie和evercookie上，需要用户登录才可以得到有效的信息。第二代才有了浏览器指纹的概念，通过不断增加浏览器的特征值从而让用户更具有区分度，例如（UA、浏览器插件信息）。第三代是通过收集用户的行为、习惯来为用户建立特征值或模型，实现真正的追踪技术，这部分目前实现比较复杂，依然在探索中。

由于第三代需要与网站业务有深度的结合，因此选择2.5代追踪技术。2代与3代之间的2.5代浏览器Fingerprint专注于解决跨浏览器识别的指纹。将特征值与权重结合计算后可以生成一个Fingerprint哈希值，用于标记不同的用户，这里采用了ClientJS提供的计算方法，通过clientjs.getFingerprint()方法获取到fingerprint的哈希值。采用的特征值包括：user agent、screen print、color depth、current resolution、available resolution、device XDPI、device YDPI、plugin list、font list、local storage、session storage、timezone、language、system language、cookies、canvas print，共16种。

根据Crossing Browser Tracking的研究，各个特征值在单浏览器下的信息熵和跨浏览器下的信息熵与稳定性如表3.1所示。

表3.1 特征值信息熵与跨浏览器稳定性

|  |  |  |  |
| --- | --- | --- | --- |
| 特征值 | 单浏览器 | 跨浏览器 | |
| 信息熵 | 信息熵 | 稳定性 |
| User Agent | 6.71 | 0.00 | 1.39％ |
| Accept | 1.29 | 0.01 | 1.25％ |
| Content encoding | 0.33 | 0.03 | 87.83％ |
| Content language | 4.28 | 1.39 | 10.96％ |
| List of plugins | 5.77 | 0.25 | 1.65％ |
| Cookies enabled | 0.00 | 0.00 | 100.00％ |
| Use of local/session storage | 0.03 | 0.00 | 99.57％ |
| Timezone | 3.72 | 3.51 | 100.00％ |
| Screen resolution and color depth | 7.41 | 3.24 | 9.13％ |
| List of fonts (Flash) | 2.40 | 0.05 | 68.00％ |
| List of HTTP headers | 3.17 | 0.64 | 9.13％ |
| Platform | 2.22 | 1.25 | 97.91％ |
| Do Not Track | 0.47 | 0.18 | 82.00％ |
| Canvas | 5.71 | 2.73 | 8.17％ |
| WebGL Vendor | 2.22 | 0.70 | 16.09％ |
| WebGL Renderer | 5.70 | 3.92 | 15.39％ |
| Use of an Ad blocker | 0.67 | 0.28 | 70.78％ |
| Screen Ratio | 1.40 | 0.98 | 97.57％ |
| List of fonts (JavaScript) | 10.40 | 6.58 | 96.52％ |
| AudioContext | 1.87 | 1.02 | 97.48％ |
| CPU Virtual cores | 1.92 | 0.59 | 100.00％ |

用户识别部分的详细流程图如图3.1所示。

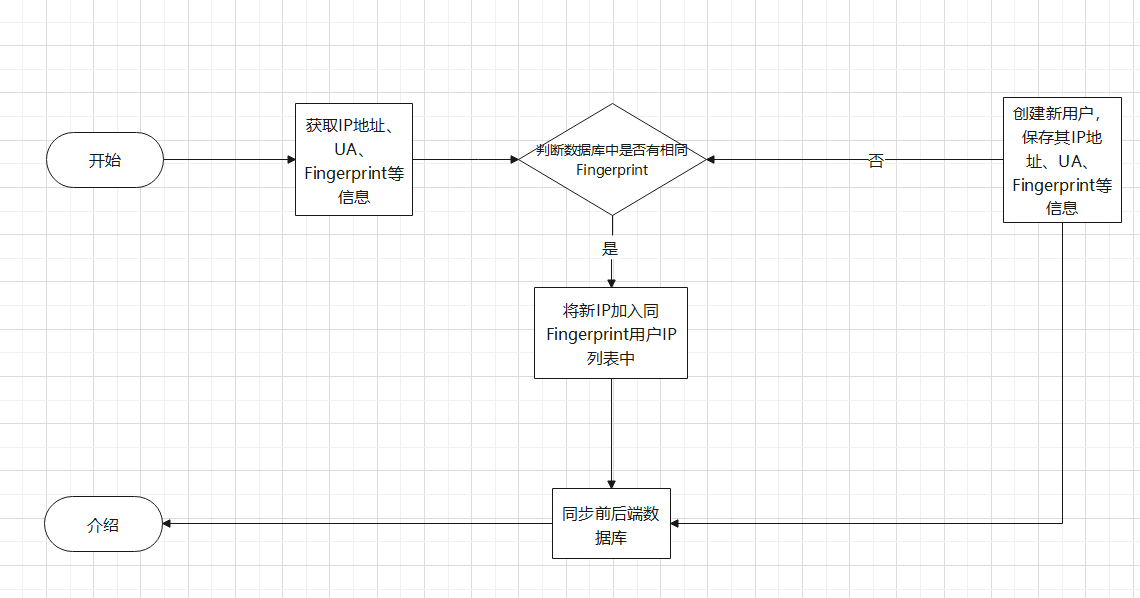


图3.1 用户识别流程图

## 爬虫识别部分

本项目的爬虫识别方式主要为特征识别以及频率控制。特征识别反爬虫是指通过客户端的特征、属性或用户行为特点来区分正常用户和爬虫程序。

（undone，去重）

Navigator对象的cookieEnable、 platform、 plugins等属性和 Screen对象的一些属性可以作为判断依据。通过将浏览器请求头中的User Agent值与navigator. userAgent 属性值进行对比，结合navigator. platform就可以判断客户端是否使用随机切换的User Agent，如果值不同则将客户端进行标记。User Agent中的操作系统显示为 Win32 ，如果 navigator.platfom属性值与此不符，也可以将该客户端进行标记。

另外，User Agent 属性值可以作为客户端特征，屏幕分辨率也可以作为客户端特征。此外，除非客户端的计算机运行在虚拟机中或是年代久远，个人计算机的核心数量一般为 2 个以上。因此，CPU 核心数量同样可以作为客户端特征。不同渲染工具的浏览器插件数量也是不相同的，虽然插件数量与渲染工具关联并不大，但这个属性值可以作为客户端特征。不过，属性值可以作为特征并不代表服务器端通过单个属性值就能确认客户端身份，只是判断客户端身份的依据之一。

由于这些属性的值可以通过 JavaScript进行更改，所以这种特征识别方式得到的结果是不可靠的。因此在编码时仅做标记用于参考。主要限制访问频率过高的爬虫程序。

访问频率指的是单位时间内客户端向服务器端发出网络请求的次数，描述网络请求频繁程度。正常用户浏览网页的频率不会像爬虫程序那么高，可以将访问频率过高的客户端视为爬虫程序。

要限制爬虫程序的请求频率，首先就是要找到并确定客户端的身份标识，这里可以使用ip或者用户识别中的fingerprint。然后根据标识记录该客户端的请求次数，拒绝单位时间内请求次数过多的客户端请求。

这里通过在Node.js端记录用户请求次数以及请求的接口，计算出时间段内请求的频率，对请求过多的用户进行限制。

## 请求限制部分

根据沟通，初步采用的流量控制方案为：某一个接口请求太频繁或者总请求次数过多则限制访问一段时间。每分钟请求接口超过10次则限制其访问1分钟、每2个时总请求数大于1000次则限制其访问频率为每10秒一次等。

（undone，更新最终版，去重）

在Node.js端完成这部分工作，更新了数据库表的设计以及操作，便于统计每个接口的请求次数、总请求次数以及当前的限制情况。更新了一些数据库操作，便于统计接口在一分钟内的访问频率。

尝试将ip信息与此前记录的用户信息进行匹配，尝试对使用ip池的爬虫进行标记，用于更好地区分爬虫与普通用户。

爬虫识别与请求限制部分的详细流程图如图3.2所示。

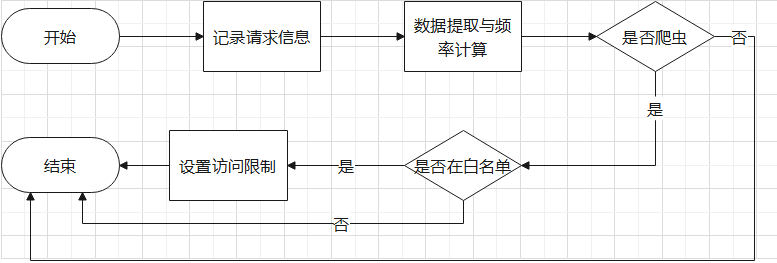


图3.2 爬虫识别与请求限制流程图

参考文献

1. Edward Roberts. Bad Bot Report 2020: Bad Bots Strike Back.[R]. San Mateo: Imperva, 2020.
2. 林中明. 基于Hadoop的Web用户识别与新闻智能推荐算法研究[D]. 郑州：战略支援部队信息工程大学，2018.
3. 伏康，杜振鹏. 网站反爬虫策略的分析与研究[J]. 电脑知识与技术，2019(28)：28-30.
4. 张晔，孙光光，徐洪云，庞婷，曲潇洋. 国外科技网站反爬虫研究及数据获取对策研究[J]. 竞争情报，2020(01)：24-28.
5. Y. Liu, Z. Yang, J. Xiu, C. Liu. Research on an anti-crawling mechanism and key algorithm based on sliding time window[A]. In: 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)[C]. Beijing: IEEE, 2016: 220-223
6. W. Zhu, J. Qin, R. Kong, H. Lin, Z. He. A System Framework for Efficiently Recognizing Web Crawlers[A]. In: IEEE SmartWorld 2018 Organizing and Program Committees. 2018 IEEE SmartWorld [C]. Guangzhou: IEEE, 2018: 1130-1133
7. H. Wang, C. Li, L. Zhang, M. Shi.Anti-Crawler strategy and distributed crawler based on Hadoop[A]. In: 2018 IEEE 3rd International Conference on Big Data Analysis(ICBDA)[C]. Shanghai: IEEE, 2018: 227-231
8. P. Lewandowski, M. Janiszewski, A. Felkner. SpiderTrap—An Innovative Approach to Analyze Activity of Internet Bots on a Website[J]. IEEE Access, 2020, 8: 141292-141309
9. G. Neelima, S. Rodda. Predicting user behavior through sessions using the web log mining[A]. In: 2016 International Conference on Advances in Human Machine Interaction (HMI) [C]. Doddaballapur: IEEE, 2016:1-5
10. 刘洋. 基于网页浏览行为的反爬虫研究[J]. 现代计算机(专业版)，2019(07)：50-60+70.
11. Cao, Y., Li, S., & Wijmans, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features[R]. San Diego: NDSS, 2017.