 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、用户标识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](#_Toc71316639)

[1.1 课题背景与意义 1](#_Toc71316640)

[1.2 国内外研究现状 1](#_Toc71316641)

[2 系统概述及需求分析 3](#_Toc71316642)

[2.1 项目概述 3](#_Toc71316643)

[2.2 功能需求 3](#_Toc71316644)

[2.2.1 记录用户标识数据 4](#_Toc71316645)

[2.2.2 标记用户 5](#_Toc71316646)

[2.2.3 记录用户请求数据 6](#_Toc71316647)

[2.2.4 标记爬虫 7](#_Toc71316648)

[2.2.5 设置访问限制 7](#_Toc71316649)

[2.3 非功能性需求 8](#_Toc71316650)

[2.4 项目任务 8](#_Toc71316651)

[3 相关原理与技术 10](#_Toc71316652)

[3.1 依赖管理CDN与NPM 10](#_Toc71316653)

[3.2 浏览器指纹Fingerprint 10](#_Toc71316654)

[3.3 前端数据库IndexedDB 11](#_Toc71316655)

[3.4 后端Node.js与Express 11](#_Toc71316656)

[3.5 服务器Nginx 12](#_Toc71316657)

[3.6 后端数据库MongoDB 12](#_Toc71316658)

[3.7 其他 12](#_Toc71316659)

[4 项目总体设计 13](#_Toc71316660)

[4.1 系统工作流程 13](#_Toc71316661)

[4.2 系统架构设计 13](#_Toc71316662)

[4.3 数据库设计 15](#_Toc71316663)

[4.3.1 数据库逻辑设计 15](#_Toc71316664)

[4.3.2 数据库表设计 17](#_Toc71316665)

[5 项目具体实现 20](#_Toc71316666)

[5.1 项目实现概述 20](#_Toc71316667)

[5.2 用户识别部分 20](#_Toc71316668)

[5.3 爬虫识别部分 23](#_Toc71316669)

[5.4 请求限制部分 24](#_Toc71316670)

[5.5 数据层的封装 25](#_Toc71316671)

[6 系统测试 27](#_Toc71316672)

[6.1 单元测试 27](#_Toc71316673)

[6.1.1 记录用户标识信息 27](#_Toc71316674)

[6.1.2 标记用户 30](#_Toc71316675)

[6.1.3 记录用户请求信息 30](#_Toc71316676)

[6.1.4 标记爬虫与访问限制 31](#_Toc71316677)

[6.2 集成测试 32](#_Toc71316678)

[总结与展望 33](#_Toc71316679)

[论文总结 33](#_Toc71316680)

[本人工作内容 33](#_Toc71316681)

[展望 33](#_Toc71316682)

[致谢 34](#_Toc71316683)

[参考文献 35](#_Toc71316684)

# 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[5]。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

# 系统概述及需求分析

## 项目概述

本课题的内容是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系统，以接口的形式接入清华大学出版社的网站。本系统通过收集客户端设备、浏览器的特征值、IP地址等信息对用户进行标识，通过请求频率对爬虫进行标识，对爬虫程序设置请求频率、总次数的限制以达到网站流量控制的目的，对网站在安全性、负载等方面提供一定的保护。

本系统的全部设计与开发工作均由本文作者完成，包括需求分析、系统设计、系统编码、测试等。

## 功能需求

本项目的功能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

### 记录用户标识数据

记录用户标识数据的目的是为了在标识用户身份时可以获得更精确的结果，具有高信息熵和高跨浏览器稳定性的用户标识数据越多，就越能唯一确定一名用户。更多有效的用户标识数据也能减少数据之间的碰撞。所记录的用户标识数据将用于计算用户Fingerprint。

数据的收集工作是在用户进入网站时完成的，也就是页面第一次加载时，无法获取到的用户标识数据将会以undefined替代。

记录用户标识数据的用例说明见表2.1。

表2.1 记录用户标识数据

|  |  |
| --- | --- |
| 用例名称 | 记录用户标识数据 |
| 简要描述 | 该用例描述了系统记录用户标识数据的过程 |
| 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. 系统收集用户标识数据，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

标记用户的目的是基于上一步收集到的用户标识数据，将用户标识数据进行加权计算，得到一个唯一的用户Fingerprint值用作区分用户的依据之一。此外，IP地址也将作为次要依据用于区分用户。

在生成Fingerprint后，将会完成前后端数据库的数据存储，并对已经存在用户进行数据的比对与更新。

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 | 已经获得用户标识数据 |
| 后置条件 | 标记成功后保存到数据库中 |
| 基本事件流 | 1. 传入收集到的用户标识信息 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

记录用户请求数据的目的是为了后续判断客户端是否为爬虫程序。需要记录的内容包括请求数据，例如header，并记录请求的时间等，用于计算单位时间内请求的频率。

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、请求的时间等。（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

标记爬虫的目的是基于获取的用户请求数据，对比数据库中记录的用户标识数据、判断请求频率是否超过上限等，将请求过高的客户端标记为爬虫程序。对于爬虫程序，还需要保存在数据库中，并设置访问限制信息。对于已经被标记的爬虫程序的客户端，则需要更新限制信息。

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

设置访问限制的过程是根据上一步标识的爬虫信息，对请求频率过高的客户端设置访问限制，达到流量控制的目的。

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 |

## 非功能性需求

本项目的非功能性需求主要包含一下几方面：

在正确性需求方面，能对跨浏览器的用户做到识别或近似识别，及时更新用户标识数据。能对高请求频率的爬虫程序进行识别，及时更新爬虫标识数据和限制数据，对爬虫程序设置访问限制。

在安全性需求方面，需要保证除前端数据库外的数据不被用户随意更改。

系统由于采用接入的形式，因此对界面没有特殊需求，可以制作简单的网站用于功能展示。

在稳定性方面，该系统接入网站后，如果系统运行的环境没有发生变化，系统需要保持长期稳定地运行。

## 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。

# 相关原理与技术

## 依赖管理CDN与NPM

在项目过程中使用了一些第三方库，包括ClientJS、Dexie等，需要引入外部依赖。在开发过程中，主要通过CDN与NPM的方式引入、管理依赖。

CDN又被称为内容分发网络，全称是Content Delivery Network。CDN的原理是当用户发送请求时，从离用户最近的服务器返回用户请求的内容，以此提高项目的加载速度。它的一个好处是可以通过缓存来加载jQuery，减少加载时间[9]。

NPM的全称是Node Package Manager，它是Node.js包含的依赖包管理组件，在安装Node.js时会一起安装。虽然它最初是用于Node.js的包管理组件，能自动处理Node.js版本或部署等很多问题，但是随着NPM的发展，现在还可以用来自动安装与管理JavaScript依赖包，多个依赖包之间的版本关系等也可以实现自动处理[10]。

本项目使用到了request-ip库、ClientJS库、Dexie库等。

request-ip库用于获取request的IP地址，它会在request中寻找特定的headers，并在内容不存在时设置为默认值。通过12个值依次判断ip地址的值，包括X-Client-IP、X-Forwarded-For、CF-Connecting-IP、Fastly-Client-Ip、True-Client-Ip、X-Real-IP、X-Cluster-Client-IP、X-Forwarded, Forwarded-For and Forwarded、req.connection.remoteAddress、req.socket.remoteAddress、req.connection.socket.remoteAddress、req.info.remoteAddress。

ClientJS库用于查找客户端的设备信息并生成数字浏览器指纹。通过本机的JavaScript函数、方法以及一些开源的资源来查找客户端的信息。由于从不同的浏览器和设备中收集用户设备特征值信息会使处理多平台兼容性变得困难，通过使用ClientJS可以简单地创建一个客户端对象，并生成浏览器或设备的指纹ID[11]。

Dexie库是简单易学的IndexedDB封装API，解决了原生IndexedDB的API过于复杂的问题。Dexie具有近乎原生的表现，也支持了批量操作。

## 浏览器指纹Fingerprint

在选择用于区分用户的设备信息时，用到了不同的浏览器指纹作为区分依据。浏览器指纹可以被分为普通指纹和高级指纹，普通指纹一般比较容易获取，但也往往容易被修改，例如HTTP的Header。高级指纹几乎可以直接确定一个唯一的浏览器标识，例如Canvas、AudioContex等。但是这里的普通指纹和高级指纹都是从同一浏览器上获取的，很多特征值在面对不同浏览器的情况下都是不稳定的，例如User Agent、Canvas在相同设备的不同浏览器中会出现不同的值。

跨浏览器指纹Fingerprint的提出就是为了解决跨浏览器识别的问题，理论上同一设备在不同浏览器上也会生成相同或接近的Fingerprint值。Fingerprint获取浏览器能提供的高信息熵、高稳定性的特征值，计算得到一个哈希值。这里的特征值可以是UA、设备时区、定位信息或者使用的语言等。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，可以极大地降低特征值之间的重合率，提高客户端唯一标识的准确性。因此往往会在计算Fingerprint时，赋予信息熵较大的特征值更大的权重[12]。

## 前端数据库IndexedDB

在选择前端数据存储时综合对比了LocalStorage、WebSQL与IndexedDB。

虽然LocalStorage使用key-value键值对的形式存储数据，但数据的存储格式都是字符串，因此并不能支持数据的检索，也不能自定义创建索引。另外，LocalStorage的容量根据浏览器的不同在2.5MB 到 10MB之间浮动，标准并不统一而且容量也不大。

另一方面WebSQL已经停止维护，只有Chrome还支持WebSQL，在兼容性上并不好。

综上，采用了更为兼容、容量更大且检索更方便的IndexedDB。IndexedDB为NoSQL型数据库，支持数据的查找、索引等。

NoSQL的全称是Not Only SQL，也被称为非关系型数据库，是相对于传统关系型数据库的另一种数据库的统称。NoSQL对于存储大规模的数据有比较好的支持，因为NoSQL的数据存储没有固定模式，可以十分方便地进行横向扩展。

由于IndexedDB的API十分复杂，并且异常处理等机制并不完善，因此使用Dexie封装的API来简化与完善对IndexedDB的操作。

## 后端Node.js与Express

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，依据CommonJS标准并使用Express框架帮助Node.js开发。

Node.js是一个开源的JavaScript跨平台运行时环境。得益于Node.js，使用JavaScript编写网站前端的开发者除了客户端业务，还可以设计服务器端业务，并且不需要学习其他的服务端语言。Node.js可以自动处理并发连接的问题，而不需要开发者花费精力管理多线程并发。在 Node.js中还可以使用新的ECMAScript标准[13]。

Express是当下最流行的，基于Node.js的Web应用开发框架，它有着灵活、小规模的特点。在提供核心的Web应用功能的同时，Express也保留了Node.js的功能。

CommonJS是一种开发规范，对应Node.js的模块化特性。在Node.js中一个文件就是一个模块，其中定义的函数、变量等都是私有的，对其他模块是不可见的。CommonJS规定模块中的module对象代表当前模块，module的exports方法是会将模块暴露给外部。CommonJS的代码是在模块的作用域内运行的，并且模块只会在第一次加载时运行一次，然后缓存运行结果，以后加载时直接读取缓存。模块还可以根据它在代码中的出现顺序进行多次加载。但是想让模块再次运行，就需要清除缓存。

## 服务器Nginx

服务器端使用了Nginx。Nginx是由俄罗斯开发者设计的一款高性能、轻量级的Web和反向代理服务器，也是一款IMAP/POP3/SMTP邮件代理服务器。Nginx可以在大多数 Unix Linux操作系统上运行，它的特点是内存占用少、并发能力强。Nginx在同类型的网页服务器中有比较好的并发能力，它的稳定性、低系统资源消耗等优点也让它变得十分受欢迎。相比于Apache服务器，Nginx在有处理高连接并发的需要时，将会是一个不错的选择。

## 后端数据库MongoDB

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的性质介于关系型数据库和非关系型数据库之间，相比于其他非关系数据库，MongoDB功能十分丰富，也十分接近关系型数据库。MongoDB的数据存储形式采用了分布式文件存储，因此对于Web应用可以方便地进行扩展并保持高性能。MongoDB数据存储的文档类似于JSON对象，数据结构由键值key-value对组成，其中value值可以包含其他文档、数组及文档数组等[14]。

## 其他

使用GitHub完成项目代码、说明文档和版本的管理。

# 项目总体设计

## 系统工作流程

系统的工作流程整体上为功能的顺序执行，在用户开始访问网站时，记录用户的标识信息，并基于用户的身份标识信息，通过计算完成用户身份的识别并完成数据的前后端存储。当用户发送请求时，收集用户的请求信息，包括header、时间、请求的接口等，将请求信息中的身份信息与所记录的进行对比，并根据请求的时间计算单位时间内的请求频率，标记请求频率超过限制的客户端并更新数据库标识、限制信息。根据限制信息对客户端设置访问限制。

系统整体的流程图如图4.1所示。

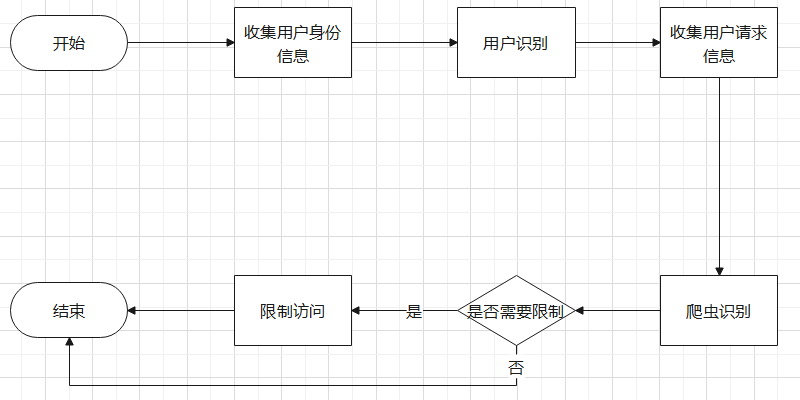


图4.1 系统流程图

## 系统架构设计

本系统在架构设计上采用了三层结构，包括接入层、业务层和数据层。系统架构图如图4.2所示。

由于本系统不需要前端展示层，因此这里用接入层表示。接入层中为系统接入网站的部分。

业务层主要包括记录用户标识数据、标识用户、记录请求数据、标识爬虫、设置访问限制这五项功能的实现。其中，可以进一步区分为在前端通过JavaScript实现的记录用户标识数据和标识用户部分，以及在后端通过Express框架下的Node.js实现的记录请求数据、标识爬虫、设置访问限制部分，其中设置访问限制也用到了部分服务器Nginx的功能。前后端的数据通信则使用了Axios实现，它是一个可以在浏览器和Node.js中使用的HTTP库，基于promise并且可以自动转换JSON格式的数据。主要使用了其中的post与get方法。

数据层主要包括对前端数据库IndexedDB与后端数据库MongoDB封装的增删改查等操作的类，数据的格式也统一为JSON。对MongoDB的操作通过后端的Node.js完成，对IndexedDB的操作在前端通过JavaScript完成。

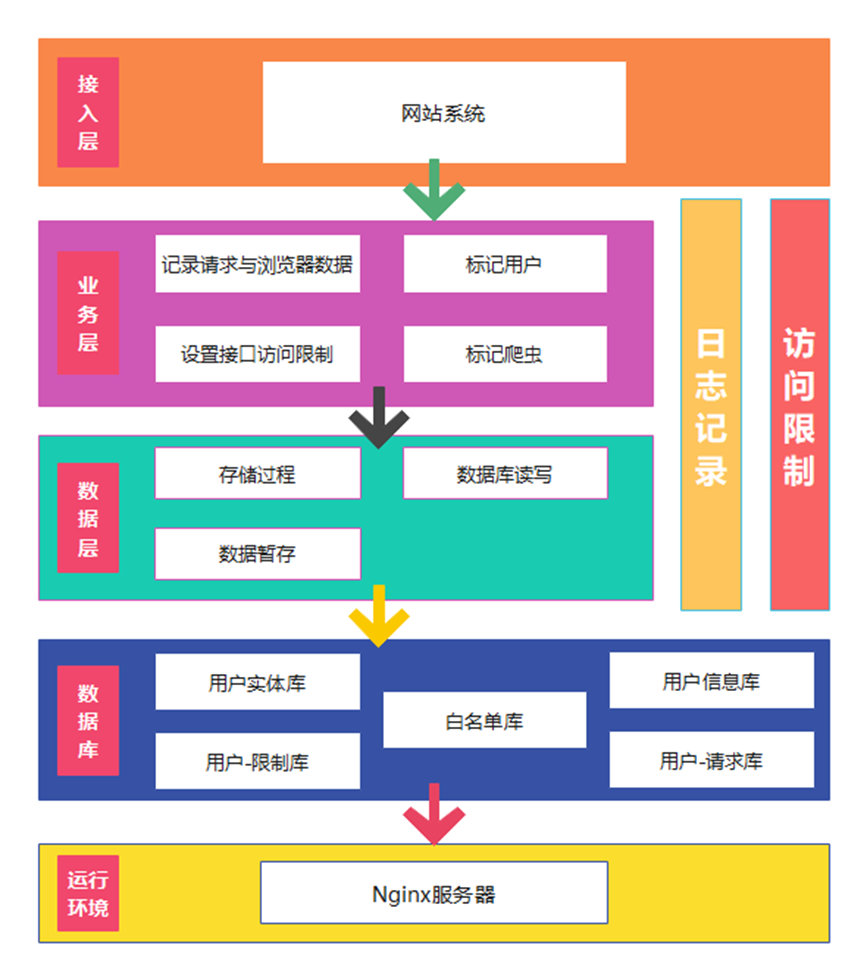


图4.2 系统架构图

## 数据库设计

### 数据库逻辑设计

根据需求提取出了数据库实体包括用户标识、用户信息、用户请求、用户限制，进一步挖掘实体中的关系，可得数据库ER图，如图4.3所示。

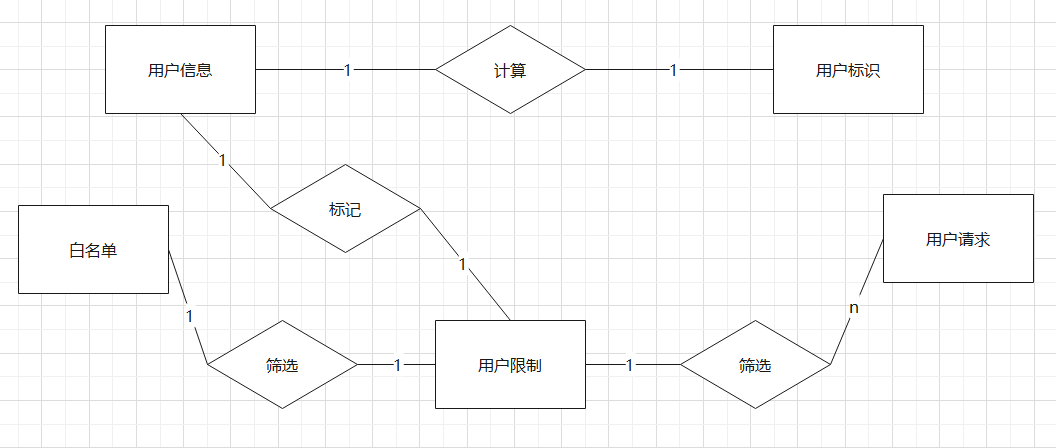


图4.3 数据库ER图

虽然用户限制表和用户信息存在实体联系，但是在实际使用过程中用户信息对用户限制表并非必需，且在使用时仅是辅助检测，因此在具体实现中不必体现两者的联系。经过转化后的数据库类图如图4.4所示，其中，省略了数据库属性内容，将会在后续的数据库表中详细说明。

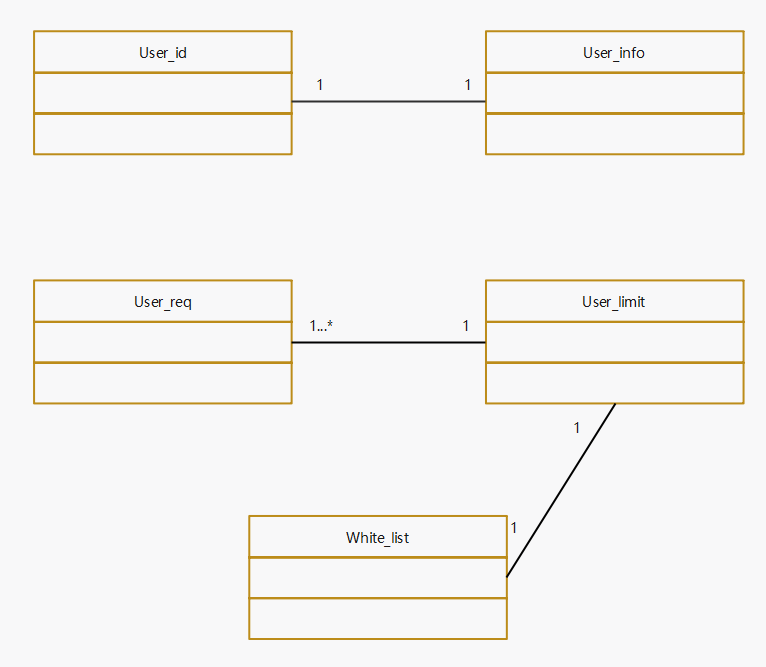


图4.4 数据库类图

### 数据库表设计

根据数据库ER图与数据库类图，设计系统的数据库表，如至表所示。

表4.1 用户标识表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| Fingerprint | Number | 唯一表示用户客户端的至多16位数字 |
| IP | Array | 客户端IP地址池数组，每个IP地址的类型为String |

用户标识表记录了根据客户端信息计算而得出的唯一Fingerprint值，以及该Fingerprint值下所使用到的所有IP地址值，便于在需要时解决使用IP池的爬虫的问题。

表4.2 用户信息表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| Fingerprint | Number | 唯一表示客户端身份的至多16位数字 |
| User Agent | String | 客户端浏览器User Agent字段的值 |
| CPU | String | 客户端CPU类型 |
| screenPrint | String | 客户端屏幕信息，包括分辨率、色深等 |
| colorDepth | Number | 客户端屏幕色深 |
| availableResolution | String | 客户端屏幕可用分辨率 |
| mimeTypes | String | 客户端多功能网络邮件扩展类型 |
| fonts | String | 客户端安装的字体 |
| timeZone | String | 客户端所在时区 |
| language | String | 客户端用户语言 |
| core | Number | 客户端CPU核心数 |
| screenOrientation, | String | 客户端屏幕方向 |
| screenAngle | Number | 客户端屏幕倾斜角度 |
| screenHeight | Number | 客户端屏幕竖向分辨率 |
| screenWidth | Number | 客户端屏幕横向分辨率 |

用户信息表记录了被唯一的Fingerprint所标识的客户端详细信息，在必要时可以结合IP池、Fingerprint数值接近程度等进一步判断是否为同一用户。

表4.3 用户请求表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| IP | String | 发出请求的客户端IP地址 |
| API | String | 客户端请求的接口名称 |
| Time1 | Date | 接收到客户端请求的时间，仅记录一分钟内的最多10次时间 |
| Time2 | Date | 接收到客户端请求的时间，仅记录一分钟内的最多10次时间 |
| … | … | … |

用户请求表记录了发送请求的用户的IP地址以及所请求的接口，同时记录了请求的时间，用于计算每分钟用户请求的频率。

表4.4 用户限制表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| IP | String | 需要设限的客户端IP地址 |
| Time | Date | 客户端访问限制的解除时间 |

用户限制表记录了被设限的客户端IP地址，当用户请求表中计算得到的每分钟请求频率高于10次时将会加入用户限制表，并将Time字段设置为一分钟后的时间，限制来自该IP地址的请求频率为每分钟1次。

表4.5 白名单表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| IP | String | 不受访问频率限制的客户端IP地址 |

白名单表记录了不受访问频率限制的客户端IP地址，在设限前需要过滤掉白名单中的IP地址。

# 项目具体实现

## 项目实现概述

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的格式保存在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端对网站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，将访问过于频繁的客户端进行标记。对于请求中User Agent、platform等信息与记录不匹配的客户端也进行标记。

在访问限制部分，将受标记的高访问频率客户端与白名单客户端进行匹配，去除需要放行的客户端后，对于其他受标记的客户端进行频率限制。在总请求次数方面，使用Nginx的limit\_req和limit\_conn功能限制每小时的总请求次数[15]。

（undone，细化）

## 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。

浏览器追踪技术到目前已经进入3代。第一代主要根据用户状态区分用户，包括对比用户的cookie、evercookie等，这些信息往往需要用户登录后才能被有效地获取。第二代追踪技术出现了浏览器指纹的概念，用户的区分主要是通过使用不同的浏览器特征值，例如UA、浏览器插件信息等。第三代追踪技术主要通过收集用户的行为或习惯特征，并建立用户的特征模型来实现不受设备、浏览器限制的用户人追踪。目前第三代追踪技术的实现比较复杂，还在进一步探索中[16]。

由于第三代追踪技术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代追踪技术浏览器Fingerprint专注于解决跨浏览器识别用户的问题。这就涉及到对于表征用户身份信息的特征值的选择。

在比较特征值唯一确定用户的效果时往往会使用熵作为依据。熵可以用来衡量一个值与唯一确定一个人的身份有多接近，通常以位为单位。直观地说，熵是对一个随机变量不同可能性的数量的概括，每增加一位熵会使可能性的数量加倍。因为地球上大约有70亿人，所以一个人的身份包含不到33位的熵（2的33次幂约为80亿），而User Agent所包含的信息熵约为10.5位。但是需要注意的是，不同的特征值之间所包含的信息可能是重复的，因此如果能同时获取互不相关的多个特征值，就可以增加确定身份的精确性。

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[17]。将浏览器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。

表5.1 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％ |

在本系统中采用了ClientJS提供的Fingerprint计算方法，通过clientjs.getFingerprint()方法获取到Fingerprint的哈希值。ClientJS的计算方法中采用的特征值包括：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。

用户识别部分的详细流程图如图5.1所示。

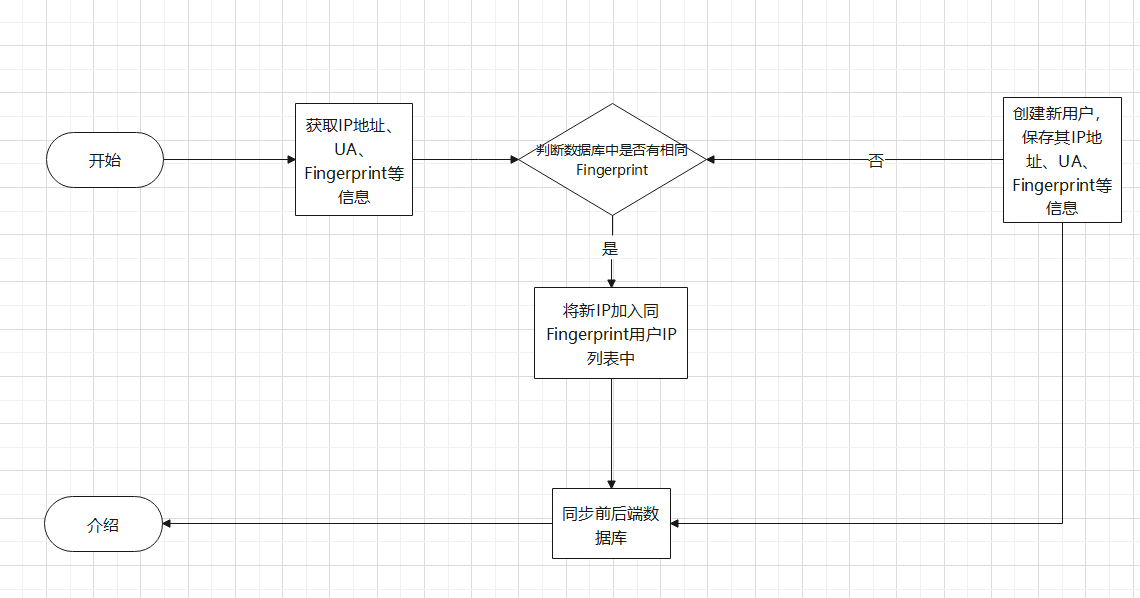


图5.1 用户识别流程图

其中，获取IP地址、User Agent、Fingerprint等特征值的部分是在用户进入网站时完成的。随后，根据是否存在同Fingerprint值的客户端，对前后端数据库进行更新。结合用户第一次进入网站的顺序图如图5.2所示。展示层为网站原有展示层，接入层为本系统接入网站的部分。



图5.2 用户识别顺序图

前后端的数据通信采用了Axios，数据库数据采用JSON格式的方式进行传递，保证了一致性与效率。

为了实现前后端数据的同步，在Node.js端的路由部分向前端提供了操作数据的API，如所示。

表5.2 路由接口表

|  |  |  |
| --- | --- | --- |
| 接口名称 | 方法 | 功能 |
| addUser | get | 增加新的User\_id和User\_info信息 |
| updateUser | get | 更新的User\_id中的IP地址信息 |

## 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、属性来区分正常用户和爬虫程序。客户端的属性值可以作为特征，并不等同于服务端可以通过某一特定的属性值来唯一确定用户身份，属性值只能作为判断用户身份的依据之一。频率控制则是根据同一IP在单位时间内访问单个接口的次数或一段时间内访问的总次数对请求进行限制[18]。

Navigator对象的cookieEnable、 platform、 plugins等属性和Screen对象的orientation等属性可以作为判断依据。通过将浏览器请求headers中的User Agent值与JavaScript获取到的navigator.userAgent 属性值进行对比，可以判断客户端是否随机切换User Agent。如果User Agent值不同，可以对当前的客户端进行标记。User Agent中还包括了操作系统信息，结合navigator. platform提供的操作系统信息，如果两者的属性值不同，也可以标记当前的客户端。

类似地，屏幕分辨率、CPU核心数等也可以作为客户端的特征值。除非客户端计算机运行在虚拟机中或是年代久远，否则JavaScript获取到的hardwareConcurrency，即计算机核心数量一般为2个以上。

在使用不同的渲染工具时，浏览器的插件数量往往也不相同，因此这个属性值也可以作为一种客户端的特征信息。

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，用于表示网络请求的频繁程度。正常用户浏览网页时发出请求的频率不会像爬虫那么高，可以将访问频率过高的客户端标记为爬虫。

限制爬虫请求频率的第一步是要确定客户端的身份标识，这里使用IP地址作为主要依据。IP地址是通过第三方库request-ip库获取的，request-ip库解决了是否有反向代理IP、connection的远程IP、后端的socket的IP等等许多种IP地址的可能性。然后根据客户端的身份标识记录此客户端在单位时间内的请求次数，拒绝请求次数过多的客户端发出的请求。

在项目中通过在Node.js端记录客户端请求的时间以及请求的接口，计算出时间段内请求的频率，对访问频率高的客户端进行标记。爬虫识别部分的顺序图如图5.3所示。



图5.3 爬虫识别顺序图

爬虫一般会采用有规律的方式请求接口完成自动化抓取，如每隔300毫秒请求一次接口或每隔一个随机时间范围请求一次接口，或某几个接口不断循环请求，这里还通过规律识别的方式识别出这些爬虫。

第一种是判断请求之间的间隔时间。当数据库中存储的客户端请求个数超过20个时可以进入识别程序，最多取51个请求进行识别，以保证程序运行的时间不会太长。计算出至多51个请求之间的至多50个时间差，以毫秒为单位，计算出其中时间差重复的时间差个数，当相同时间差的个数占比超过样本总数的50%时，将会把客户端标记为爬虫程序存入数据库，并将原因设置为相同的时间差。

第二种是根据请求的接口是否存在循环请求的情况。同样，会当数据库中的客户端请求个数超过20个时进入识别程序，取最多50个请求进行识别。将相同接口的请求划为一组，判断组内请求的间隔计数以及不同组之间请求间隔的计数是否相同，当判断为循环组请求时，将客户端标记为爬虫程序，在数据库的原因中添加循环请求。

## 请求限制部分

由于通过JavaScript可以更改爬虫识别部分中收集的客户端属性值，所以这种特征识别方式得到的结果也并不是完全可靠的。因此在设置限制时仅做标记用于参考，不会限制标识信息不匹配的客户端，主要限制访问频率过高的爬虫程序[19]。

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）

在Node.js端实现单个接口请求频率限制的这部分工作，连接MongoDB数据库用于统计客户端对每个接口的请求时间以及当前的限制情况。

通过将IP地址信息与此前记录的用户信息进行匹配，可以做到对使用IP池的爬虫进行标记，用于更好地区分爬虫与普通用户。

请求总次数的限制则使用Nginx服务器所包含的limit\_req和limit\_conn功能实现。将limit\_req\_zone中的rate设置为500r/h以限制每个IP每小时的请求个数不超过500个。将漏桶数burst设置为1，当超过了500个请求上限时，缓冲队列中的1个请求会被延迟访问。同时，设置nodelay选项，Nginx将严格使用平均速率限制客户端的请求个数，超过缓冲队列外的请求会直接返回503，如果没有nodelay选项会造成大量tcp连接请求处于等待状态。

爬虫识别与请求限制部分的详细流程图和顺序图分别如图5.4和图5.5所示。当服务器端收到客户端请求时，首先根据User\_limit表判断客户端是否受到访问限制，如果受到访问限制则拒绝请求。否则，在User\_req表中记录请求时间并计算其一分钟内的访问频率，响应请求频率正常的客户端请求。如果频率超过限制，则判断是否在white\_list表中，如果在则响应请求，否则在User\_limit表中设置限制，并拒绝客户端的请求。



图5.4 请求限制顺序图

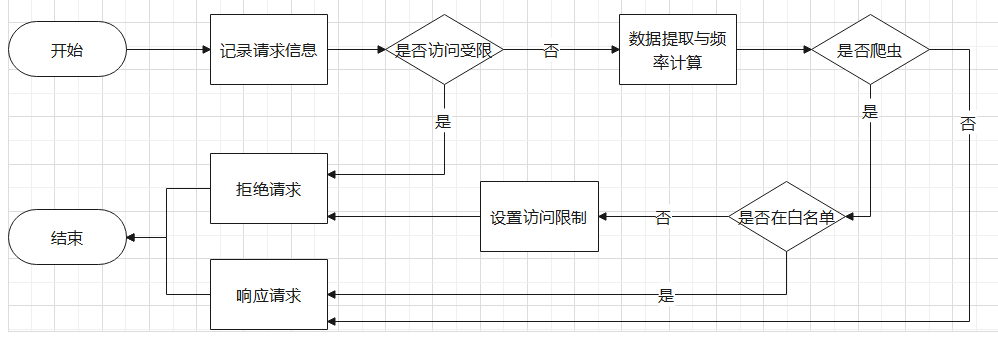


图5.5 爬虫识别与请求限制流程图

## 数据层的封装

对MongoDB数据库的操作封装在Node.js类中，提供的操作接口如表5-3所示。

表5.3 MongoDB操作接口表

|  |  |
| --- | --- |
| 接口名称 | 功能 |
| connect | 连接到MongoDB数据库中的具体表 |
| insert | 向表中插入一条或多条数据 |
| delete | 删除表中符合条件的一条或多条数据 |
| update | 将表中符合条件的数据替换为新的值 |
| find | 查询表中符合条件的数据 |

为了保证数据库数据的整洁、减少数据的冗余，还实现了去重、合并的方法。例如在调用update方法更新Fingerprint对应的IP地址池时，可能会添加重复的IP地址，因此每次在更新数据后会执行一次数据去重。类似地，同样可能出现insert的数据已经存在同一客户端的情况，此时则会合并同一客户端的数据，合并之后也会执行一次去重。通过添加数据、更新数据时执行去重和合并方法可以很大程度上避免数据重复、混乱的情况。

# 系统测试

为了保证本系统在接入网站后能够正常使用，尽可能发现系统中存在的问题，减少实际使用中出错的风险，本系统在开发过程中进行了大量的测试工作，包括各个模块的单元测试和系统集成测试。

## 单元测试

本系统的单元测试按照系统设计中的功能模块进行划分，包括记录用户标识信息、标识用户、记录用户请求信息、标识爬虫、设置访问限制五个部分。

### 记录用户标识信息

记录用户标识信息的测试用例如表6.1到表6.4所示，包含了获取客户端User Agent、屏幕信息、CPU信息以及其他辅助信息的测试。主要为测试信息获取的成功与否，可以获取到的信息保证准确，无法获取到的信息会使用undefined替代。

表6.1 获取客户端User Agent测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 获取客户端User Agent | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 01-01-01 | 正确获取客户端User Agent | * + - 1. 允许浏览器获取User Agent       2. 进入网站 | 获取到客户端User Agent | 通过 |
| 01-01-02 | 无法获取客户端User Agent | * + - 1. 拒绝浏览器获取User Agent       2. 进入网站 | 无法获取到客户端User Agent，用undefined替代 | 通过 |

表6.2 获取客户端屏幕信息测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 获取客户端屏幕信息 | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 01-02-01 | 正确获取客户端屏幕方向 | * + - 1. 允许浏览器获取屏幕方向       2. 进入网站 | 获取到客户端屏幕方向 | 通过 |
| 01-02-02 | 无法获取客户端屏幕方向 | * + - 1. 拒绝浏览器获取屏幕方向       2. 进入网站 | 无法获取到客户端屏幕方向，用undefined替代 | 通过 |
| 01-03-01 | 正确获取客户端屏幕参数 | * + - 1. 允许浏览器获取屏幕参数       2. 进入网站 | 获取到客户端屏幕参数 | 通过 |
| 01-03-02 | 无法获取客户端屏幕参数 | * + - 1. 拒绝浏览器获取屏幕参数       2. 进入网站 | 无法获取到客户端屏幕参数，用undefined替代 | 通过 |
| 01-04-01 | 正确获取客户端屏幕色深 | * + - 1. 允许浏览器获取屏幕色深       2. 进入网站 | 获取到客户端屏幕色深 | 通过 |
| 01-04-02 | 无法获取客户端屏幕色深 | * + - 1. 拒绝浏览器获取屏幕色深       2. 进入网站 | 无法获取到客户端屏幕色深，用undefined替代 | 通过 |
| 01-05-01 | 正确获取客户端屏幕可用分辨率 | * + - 1. 允许浏览器获取屏幕可用分辨率       2. 进入网站 | 获取到客户端屏幕可用分辨率 | 通过 |
| 01-05-02 | 无法获取客户端屏幕可用分辨率 | * + - 1. 拒绝浏览器获取屏幕可用分辨率       2. 进入网站 | 无法获取到客户端屏幕可用分辨率，用undefined替代 | 通过 |
| 01-06-01 | 正确获取客户端屏幕倾角 | * + - 1. 允许浏览器获取屏幕倾角       2. 进入网站 | 获取到客户端屏幕倾角 | 通过 |
| 01-06-02 | 无法获取客户端屏幕倾角 | * + - 1. 拒绝浏览器获取屏幕倾角       2. 进入网站 | 无法获取到客户端屏幕倾角，用undefined替代 | 通过 |
| 01-07-01 | 正确获取客户端屏幕纵向分辨率 | * + - 1. 允许浏览器获取屏幕纵向分辨率       2. 进入网站 | 获取到客户端屏幕纵向分辨率 | 通过 |
| 01-07-02 | 无法获取客户端屏幕纵向分辨率 | * + - 1. 拒绝浏览器获取屏幕纵向分辨率       2. 进入网站 | 无法获取到客户端屏幕纵向分辨率，用undefined替代 | 通过 |
| 01-08-01 | 正确获取客户端屏幕横向分辨率 | * + - 1. 允许浏览器获取屏幕横向分辨率       2. 进入网站 | 获取到客户端屏幕横向分辨率 | 通过 |
| 01-08-02 | 无法获取客户端屏幕横向分辨率 | * + - 1. 拒绝浏览器获取屏幕横向分辨率       2. 进入网站 | 无法获取到客户端屏幕横向分辨率，用undefined替代 | 通过 |

表6.3 获取客户端CPU信息测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 获取客户端CPU信息 | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 01-09-01 | 正确获取客户端CPU类型 | * + - 1. 允许浏览器获取CPU类型       2. 进入网站 | 获取到客户端CPU类型 | 通过 |
| 01-09-02 | 无法获取客户端CPU类型 | * + - 1. 拒绝浏览器获取CPU类型       2. 进入网站 | 无法获取到客户端CPU类型，用undefined替代 | 通过 |
| 01-10-01 | 正确获取客户端CPU核心数 | * + - 1. 允许浏览器获取CPU核心数       2. 进入网站 | 获取到客户端CPU核心数 | 通过 |
| 01-10-02 | 无法获取客户端CPU核心数 | * + - 1. 拒绝浏览器获取CPU核心数       2. 进入网站 | 无法获取到客户端CPU核心数，用undefined替代 | 通过 |

表6.4 获取客户端其他信息测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 获取客户端其他信息 | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 01-11-01 | 正确获取客户端多功能网络邮件扩展类型 | * + - 1. 允许浏览器获取多功能网络邮件扩展类型       2. 进入网站 | 获取到客户端多功能网络邮件扩展类型 | 通过 |
| 01-11-02 | 无法获取客户端多功能网络邮件扩展类型 | * + - 1. 拒绝浏览器获取多功能网络邮件扩展类型       2. 进入网站 | 无法获取到客户端多功能网络邮件扩展类型，用undefined替代 | 通过 |
| 01-12-01 | 正确获取客户端安装的字体列表 | * + - 1. 允许浏览器获取安装的字体列表       2. 进入网站 | 获取到客户端安装的字体列表 | 通过 |
| 01-12-02 | 无法获取客户端安装的字体列表 | * + - 1. 拒绝浏览器获取安装的字体列表       2. 进入网站 | 无法获取到客户端安装的字体列表，用undefined替代 | 通过 |
| 01-13-01 | 正确获取客户端语言 | * + - 1. 允许浏览器获取语言       2. 进入网站 | 获取到客户端语言 | 通过 |
| 01-13-02 | 无法获取客户端语言 | * + - 1. 拒绝浏览器获取语言       2. 进入网站 | 无法获取到客户端语言，用undefined替代 | 通过 |
| 01-14-01 | 正确获取客户端时区 | * + - 1. 允许浏览器获取时区       2. 进入网站 | 获取到客户端时区 | 通过 |
| 01-14-02 | 无法获取客户端时区 | * + - 1. 拒绝浏览器获取时区       2. 进入网站 | 无法获取到客户端时区，用undefined替代 | 通过 |

### 标记用户

标记用户的测试用例如表6.5所示。包含生成用户Fingerprint、正确获取IP地址以及对用户存在与否的数据整合。

表6.5 标记用户测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 标记用户 | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 02-01-01 | 正确生成新的客户端标识 | * + - 1. 使用新的设备与IP地址       2. 允许浏览器获取IP地址和标识信息       3. 进入网站 | 生成新的客户端标识，以新数据形式加入User\_id表和User\_info表 | 通过 |
| 02-01-02 | 无法获取新的客户端IP地址 | * + - 1. 使用新的设备       2. 拒绝浏览器获取IP地址       3. 进入网站 | 无法获取到客户端IP地址，用undefined替代，以新数据形式加入User\_id表和User\_info表 | 通过 |
| 02-02-01 | 正确获取已有客户端的新信息 | * + - 1. 使用原有的设备与新的IP地址       2. 允许浏览器获取IP地址和标识信息       3. 进入网站 | 匹配到已有的客户端标识，将新的IP地址加入User\_id表的IP池中，并去除重复数据 | 通过 |
| 02-02-02 | 正确获取已有客户端的已有信息 | * + - 1. 使用原有的设备与IP地址       2. 允许浏览器获取IP地址和标识信息       3. 进入网站 | 匹配到已有的客户端标识，发现IP地址已存在，不做更改 | 通过 |
| 02-02-03 | 无法获取已有的客户端IP地址 | * + - 1. 使用原有的设备       2. 拒绝浏览器获取IP地址       3. 进入网站 | 无法获取到客户端IP地址，用undefined替代，匹配到已有的客户端标识，不做更改 | 通过 |

### 记录用户请求信息

记录用户请求信息的测试用例如表6.6所示。

表6.6 记录用户请求信息测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 记录用户请求信息 | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 03-01-01 | 正确记录正常用户请求 | * + - 1. 使用新的IP地址       2. 进入网站并请求接口 | 记录客户端IP地址、请求的接口、请求的时间，并返回数据。清除超过一分钟的请求时间。 | 通过 |

### 标记爬虫与访问限制

标记爬虫与设置访问限制的测试用例如表6.7所示。包括在达到频率上限时为客户端设置访问限制、放行白名单客户端、拒绝受限用户请求、解除原有受限用户请求限制。，

表6.7 标记爬虫与访问限制测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例名称 | 标记爬虫与访问限制 | | | |
| 测试编号 | 测试描述 | 测试步骤 | 预期结果 | 测试结果 |
| 04-01-01 | 为新的爬虫程序设置访问限制 | * + - 1. 使用在一分钟内已请求某接口9次的IP地址       2. 进入网站并请求同一接口 | 返回数据。将IP地址加入User\_limit表，并将回复时间设置为发送请求的一分钟后。 | 通过 |
| 04-01-02 | 白名单客户端不受限访问 | * + - 1. 使用在一分钟内已请求某接口9次的白名单IP地址       2. 进入网站并请求同一接口 | 返回数据。 | 通过 |
| 04-02-01 | 受限用户继续请求接口 | * + - 1. 使用在一分钟内已请求某接口超过10次的IP地址       2. 进入网站并请求同一接口 | 请求被拒绝。 | 通过 |
| 04-02-02 | 受限用户解除请求限制 | * + - 1. 使用在一分钟前已请求某接口超过10次的IP地址       2. 进入网站并请求同一接口 | 返回数据。User\_limit表中对应IP地址的限制数据被删除。 | 通过 |

## 集成测试

集成测试部分主要关心各个模块集成之后需要联动的部分或可能互相影响导致的问题。

记录用户标识信息部分，会得到已有用户的新信息，此时需要在标记用户部分中更新用户信息，将新信息加入的同时还需要去除重复信息。IndexedDB与MongoDB中存储的的客户端Fingerprint与IP地址分别如图6.1和图6.2所示。

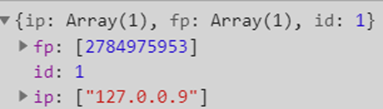


图6.1 IndexedDB中存储的客户端信息

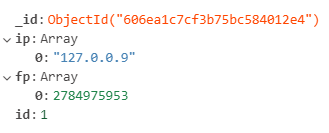


图6.2 MongoDB中存储的客户端信息

IndexedDB与MongoDB中存储的的客户端详细信息分别如图6.3和图6.4所示。



图6.3 IndexedDB中存储的客户端详细信息

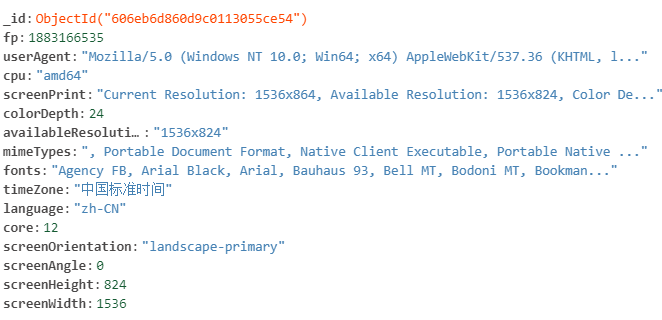


图6.4 MongoDB中存储的客户端详细信息

记录用户请求模块与访问限制则有较多的关联，当记录的用户对某一接口在1分钟内的请求次数超过10次后，需要发送给请求限制模块，对比客户端是否在白名单中后，对客户端设置访问限制。而受限制的客户端在经过限制时间后需要解除限制，同时请求记录模块需要重新开始记录用户请求的接口与时间。

总结与展望

论文总结

随着互联网网站在信息传播方面变得越发重要，网站用户的身份识别与防止网站受到爬虫流量影响也变得十分重要。网站访客的身份一方面可以帮助网站了解用户习惯，一方面也可以更有效地防止爬虫程序的入侵。基于请求频率的访问限制也能对网站流量做到很好的控制。

本论文详细阐述了网站访客唯一行识别与智能流控系统的全部实现过程，包括需求分析、技术选型、系统总体设计、系统数据库设计、系统各功能模块设计与编码实现、系统测试等方面。

本人工作内容

本项目从需求的获取与需求的分析、技术选型、数据库设计、系统的设计与实现、到系统测试的所有工作均由本人独立完成。

展望

本系统所采用的的用户识别技术是基于2.5代的跨浏览器识别技术Fingerprint，而随着第三代基于用户行为与习惯的识别技术不断成熟，使用用户的特征模型对用户进行识别将会是更准确的方法。在用户识别方面，本系统虽然以Fingerprint作为区分用户的依据，但是通过对比User\_info中的其他用户设备信息以及记录到的用户IP池，可以进一步判断相近Fingerprint的用户是否为同一用户。

类似地，通过请求IP查找到Fingerprint后，可以对比得出爬虫程序是否使用IP池、是否使用随机User Agent等，对爬虫程序作更严格的限制。

在访问限制部分，过时的浏览器版本、不合理的海外IP、不合理的Referer等也可以进行标记[20]，在需要时可以在这些方面设置更为严格的限制策略。还可以将影响严重的用户加入Nginx的blockip配置文件中禁止访问，加快请求响应速度与访问限制的灵活性。

致谢

感谢指导老师吕云翔老师对我的悉心指导，在吕云翔老师的课程上和课外共同工作的过程中，我都收获了许多理论知识和丰富的技术经验。

感谢清华大学出版社的相关项目负责人在本课题完成过程中提供的需求上、技术上的帮助与支持。

感谢北京航空航天大学士谔书院和软件学院的老师以及2017级士谔书院的同学，让我在计算机这条道路上收获了许多知识，在课业之余也留下了那么几个不错的宝贵瞬间。

感谢我的朋友们，尤其是小分队和面包的朋友，让我在休息时间的生活中充满乐趣。

感谢我的偶像们，不论是娱乐上的还是技术上的，你们创造出了无数有价值的事物，让我在这条道路上保持热情，不断努力。

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. 赵红. CDN内容分发网络技术原理[J]. 中华建设科技, 2017, (第8期).
10. Dierx Peter. A Beginner's Guide to npm – the Node Package Manager［EB/OL］. sitepoint, 2016-07-22.
11. Jack Spirou. ClientJS Documents［EB/OL］. clientjs.org, 2020-10-22.
12. 程进. 2.5代指纹追踪技术—跨浏览器指纹识别［EB/OL］. paper.seebug.org, 2017-07-10.
13. Ry Dahl. Release v0.0.1 · nodejs/node［EB/OL］. GitHub, 2019-12-24.
14. 张绍华，潘蓉，宗宇伟. 大数据技术与应用大数据治理与服务[M]. 上海：上海科学技术出版社, 2016-01: 72.
15. 恩蒙. Nginx访问频率控制［EB/OL］. https://www.cnblogs.com/zhangeamon/p/9341807.html, 2018-07-20
16. 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
17. Cao, Y., Li, S., & Wijmans, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features[R]. San Diego: NDSS, 2017.
18. 张渊博. 网站反爬虫策略的分析与研究[J]. 电子元器件与信息技术,2021,5(1):14-15.
19. 胡俊潇，陈国伟. 网络爬虫反爬策略研究[J]. 科技创新与应用, 2019,15:137-140.
20. 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),2019,7：50-60+70.