5.

由于本项目为接入性质，首先需要模拟接入网站，包括nginx服务器、node.js的后端等。由于这些方面知识较为匮乏，本周主要为知识补充及开发环境搭建。

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，Nnginx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。在我的阿里云服务器上完成了nginx配置。

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 的前端开发者现在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 中可以使用新的 ECMAScript 标准。

6.

上周了解到的内容多为基于后端，发现在与前端通信方面还有空白，这周进行补足。主要搭建了express框架以及了解了CommonJS标准用于导出JSSDK。

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。

Node 应用由模块组成，采用 CommonJS 模块规范。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。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，但是只会在第一次加载时运行一次，然后缓存运行结果，以后直接读取缓存。要想让模块再次运行，必须清除缓存。模块加载的顺序，就是模块在代码中出现的顺序。

7.

本周进行初步的数据库设计，并对前后端数据库进行选型。

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。此外WebSQL也已经停止维护，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。

8.

本周优化了数据库方面的操作。为了与前端数据库IndexedDB匹配，在后端数据库上采用了同为Nosql型数据库的MongoDB。对MongoDB以及IndexedDB的增删改查操作进行了封装，通过Node.js控制对MongoDB的增删改查等操作。

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。

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。

9.

本周进一步调研了用户识别的可行方法并开始实施。

ip地址可以作为一种特征值，但是由于需要在前端中获取，需要借用一些第三方库。经过多方尝试决定采用搜狐提供的sdk，并通过CDN的方式进行引入。

期间考虑到npm引入的必要性，又深入学习了CDN引入与npm引入分别的使用情景。

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。

10.

本周进一步完成了用户识别方面的内容。

为了实现跨浏览器识别，调研了获取MAC地址相关的方法，发现只能在IE浏览器中的ActiveX控件中实现，考虑到IE浏览器的普及性放弃MAC地址识别的方法。

编码实现了User Agent、核心数等其他信息的获取，完成了数据的存储。

进一步调研跨浏览器识别的方法，了解到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代专注于解决跨浏览器识别的指纹。

13.

本周编码实现了跨浏览器识别fingerprint的部分。用户识别大致完成。

浏览器指纹也可以进行简单的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。

但上述提到的浏览器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UA、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。

将特征值与权重结合计算后可以生成一个哈希值，用于标记不同的用户，这里采用了ClientJS.org提供的计算方法，通过clientjs.getFingerprint()方法获取到fingerprint。采用的特征值包括User Agent、时区等16种。

14.

本周尝试优化用户识别部分数据的整理，并开始调研爬虫识别的具体实现。

由于之前的数据库操作较为重复，尝试优化数据整理的操作，但是由于数据库的异步机制，没有达到预期效果，暂时保留原来的方法等待后续改进。

信息校验型反爬虫主要出现在网络请求阶段，尽可能拒绝爬虫程序的请求。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。

Navigator对象的cookieEnable、 platform、 plugins等属性和 Screen对象的一些属性也可以作为判断依据。比如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，如果值不同则将以客户端视为爬虫程序。User Agent中的操作系统显示为 Win32 ，如果 navigator.platfom属性值与此不符，也可以将该客户端视为爬虫程序。

15.

本周继续完成爬虫识别的编码工作。

查阅资料后认为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。一般个人计算机的核心数量为 2 个以上，除非客户端的计算机运行在虚拟机中或是年代久远的。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。

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。

调查发现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。因此在编码时仅做记录参考。主要限制访问频率过高的爬虫程序。

16.

本周继续完成爬虫识别的编码工作。

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。测试时可以直接用 Requests库发起请求。

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。

17.

本周继续完成流量控制的编码工作，主要根据访问频率进行限制。

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。

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。

调查发现请求限制最好放在服务器nginx端，学习nginx端的请求限制。