# 系统说明

## 爬虫识别

这一部分主要在node.js中实现，在接口中引入爬虫模块。

### 使用

在node.js中，通过以下的代码使用爬虫识别模块。使用的参数为用户的请求。

const recCrawler = require("./crawlers");

recCrawler(req); //接口中

### 数据库

**数据库：**连接ip-request表”reqs”、已识别的爬虫表”craws”。连接方式如下：

const Mongo = require("./mongo");

const reqs = new Mongo("mongodb://localhost:27017/", "user", "reqs");

const craws = new Mongo("mongodb://localhost:27017/", "user", "craws");

两张数据库表的结构如下表1与表2所示。

表1 用户请求时间表reqs

|  |  |  |
| --- | --- | --- |
| 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 |
| Time | Date | 存储接收到客户端请求的时间 |
| Count | Number | 存储每个客户端请求的计数 |

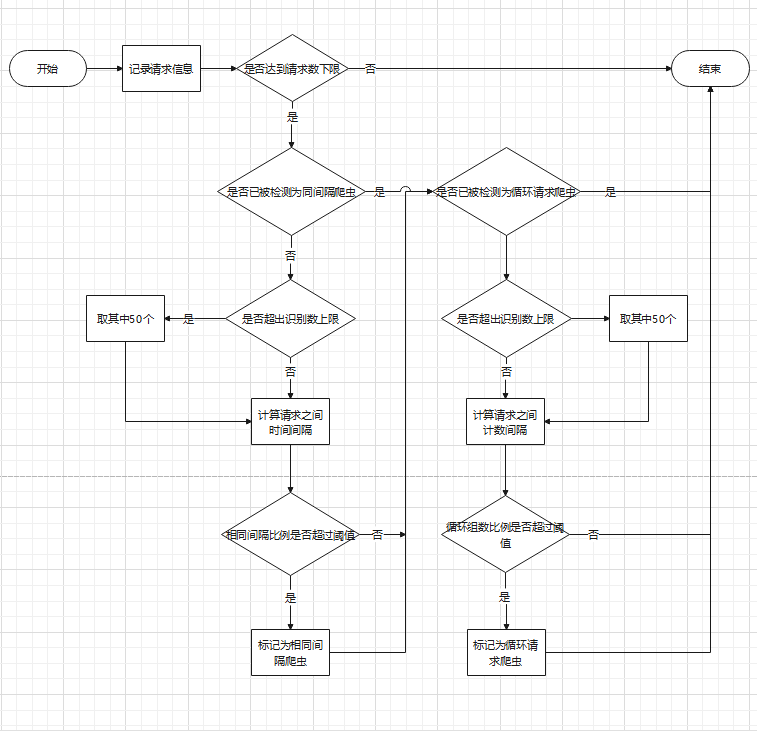
reqs表记录了每一次发送请求的用户的IP地址、所请求的接口和请求的时间，同时对请求进行计数，存储在count中。Count可以帮助判断客户端是否采用循环请求的方式，Time则用于判断请求间隔是否相同。

表2 爬虫表craws

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| IP | String | 被标识为爬虫的客户端IP地址 |
| Reason | Array | 每个reason为String类型，包括三种sameGap、loopApi、highFreq表示时间间隔相同、循环请求接口和高访问频率 |

### 程序流程

爬虫识别部分的流程图如下图所示。



**请求间隔：**对于请求的时间间隔进行计算，取至少20个、至多50个最新请求。当请求间隔的时间（毫秒）相等的个数超过所取样本的50%时，标记为爬虫，标记理由设置为相同间隔sameGap。

**循环请求：**计算请求的计数间隔，取至少20个、至多50个最新请求。当请求间隔的计数相等的个数（也就是成循环组的请求个数）超过所取样本的50%时，标记为爬虫，标记理由设置为循环请求loopApi。

以上两个部分的识别仅当请求个数大于20个，且客户端的爬虫标记理由没有对应的内容时才会启用。

在请求限制部分中识别到的访问频率超过限制的客户端也会被识别为爬虫，标记理由设置为高频率highFreq。

## 单接口1分钟访问频率限制

这一部分主要在node.js中实现，在接口中引入单接口访问限制模块。

### 使用

在node.js中，通过以下的代码使用访问限制模块。使用的参数为用户的请求，返回值lim为被限制客户端剩余需要等待的毫秒数，若不被限制则返回0；

const checkLim = require("./limit");

checkLim(req).then(lim => {

console.log(lim)

}); //接口中

### 数据库

**数据库：**连接ip-接口时间表”apis”、 ip-限制表“lims”、白名单表“whites”、已识别的爬虫表”craws”。 连接方式如下：

const Mongo = require("./mongo");

const apis = new Mongo("mongodb://localhost:27017/", "user", "apis");

const lims = new Mongo("mongodb://localhost:27017/", "user", "lims");

const whites = new Mongo("mongodb://localhost:27017/", "user", "whites");

const craws = new Mongo("mongodb://localhost:27017/", "user", "craws");

另外三张数据库表的结构如下与所示。

表3 ip-接口时间表apis

|  |  |  |
| --- | --- | --- |
| 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 |
| Time | Array | 每一条时间的类型为Date，存储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 |

ip-接口时间表记录了发送请求的用户的IP地址以及所请求的接口，以同一IP和同一接口为一组整合，同时记录了请求的时间，用于计算每分钟用户对同一接口请求的频率。

表4 ip-限制表lims

|  |  |  |
| --- | --- | --- |
| 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 |

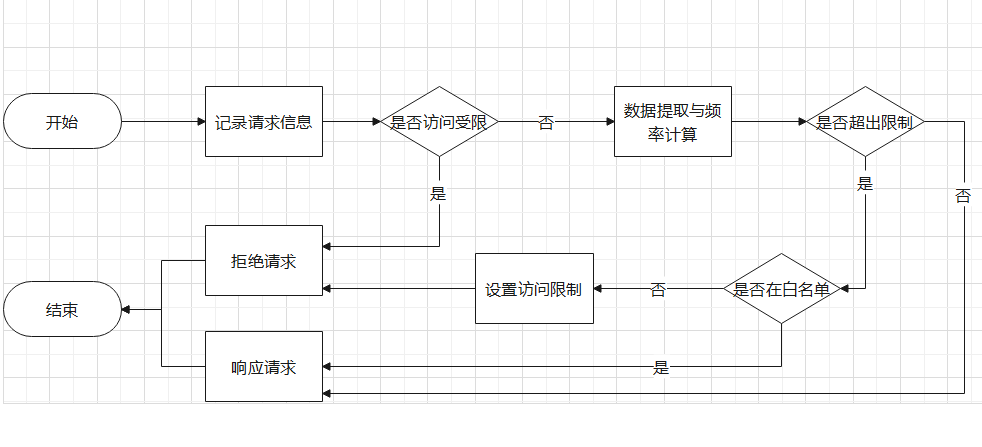
ip-限制表记录了被设限的客户端IP地址，当ip-接口时间表中计算得到的每分钟请求频率高于10次时将会加入ip-限制表，并将Time字段设置为一分钟后的时间，限制来自该IP地址的请求频率为每分钟1次。

表5 白名单表whites

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

### 程序流程

请求限制部分的流程图如下图所示。



当服务器端收到客户端请求时，首先根据lims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apis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whites表中，如果在则响应请求，否则在lims表中设置限制，并拒绝客户端的请求。

这里的拒绝、响应请求分别通过模块的返回值确定，0为响应，其他为拒绝，表示需要等待的毫秒数。

## 2小时总访问次数限制

这一部分主要在nginx中实现，使用了lua来编写判断脚本。

### 使用

在nginx.conf中需要控制访问的站点location段中加入访问控制的代码：

access\_by\_lua\_file ‘/usr/local/nginx/conf/access.lua’

### 依赖

使用openresty来扩展nginx对lua的支持，安装openresty的方式如下：

sudo apt-get -y install openresty

具体的安装可以参考官方文档的下载和新手上路页面。

<http://openresty.org/cn/linux-packages.html>

### 数据库

数据库的表使用一张，记录了ip-限制ban、ip记录周期开始时间time、ip周期内访问计数count、是否白名单white\_list。如表6所示。连接redis数据库的方式如下所示：

local redis = require 'resty.redis'

local cache = redis.new()

local ok ,err = cache.connect(cache,'127.0.0.1','6379')

cache:set\_timeout(60000)

表6 nginx表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 意义 |
| IP | String | 客户端IP地址 |
| count | Number | 记录周期内ip请求次数 |
| time | Time | ip记录周期开始时间 |
| white\_list | Number | 1表示当前ip为白名单用户 |
| ban | Number | 1表示当前ip被限制，10s后过期 |

### 程序流程

程序流程类似单接口限制：

1. 检查是否是白名单
2. 查询ip是否在封禁时间段内
3. 如果ip记录周期不存在或者进入新的记录周期，则重置记录周期开始时间和访问计数
4. 如果处于当前记录周期内，则ip访问计数+1
5. 如果ip计数大于指定ip访问频率上限，则设置ip封禁ban为1，设置ban的过期时间为ip封禁时间（10s）之后