用户识别部分说明

# 依赖

1. **ClientJS** ——用于fingerprint获取

CDN引入：

<script src="https://cdnjs.cloudflare.com/ajax/libs/ClientJS/0.1.11/client.min.js"></script>

NPM安装：

npm install clientjs

使用：

const ClientJS = require('clientjs');

import CientJS from 'clientjs';

//视情况使用上述引入

const client = new ClientJS();

1. **Dexie** ——用于操作IndexedDB

CDN引入：

<script src=”https://unpkg.com/dexie@latest/dist/dexie.js”></script>

NPM安装：

npm install dexie

使用：

//必要时导入

import Dexie from 'dexie';

//创捷user数据库，存在时打开

var db = new Dexie("user");

//创捷两张表，所标识的名称为索引

db.version(1).stores({

userfps: "++id, \*ip, \*fp", //自增索引id，多目索引ip与fp

fingerprints:

"fp, userAgent, cpu, screenPrint, colorDepth, availableResolution, mimeTypes, fonts, timeZone, language, core"});

1. **IP地址SDK**

CDN引入：

<script language=javascript src="http://pv.sohu.com/cityjson?ie=utf-8"></script>

使用：

let ip = returnCitySN["cip"];

1. **Axios** ——数据传输

NPM安装：

npm install axios

配置：

Config中：

proxyTable: {

"/api": {

target: "http://localhost:3000",

secure: false,

changeOrigin: true,

pathRewrite: {

"^/api": ""

}

}

}

并设置basicURL：

axios.defaults.baseURL = "/api";

使用：

//传输IndexedDB的数据

this.$axios({

method: "get",

url: "/syncufp",

params: {

arr: arr

}

}).then(res => {

console.log(res.data);

});

1. MongoDB ——用于存放用户数据

NPM安装：

npm install mongodb

使用：

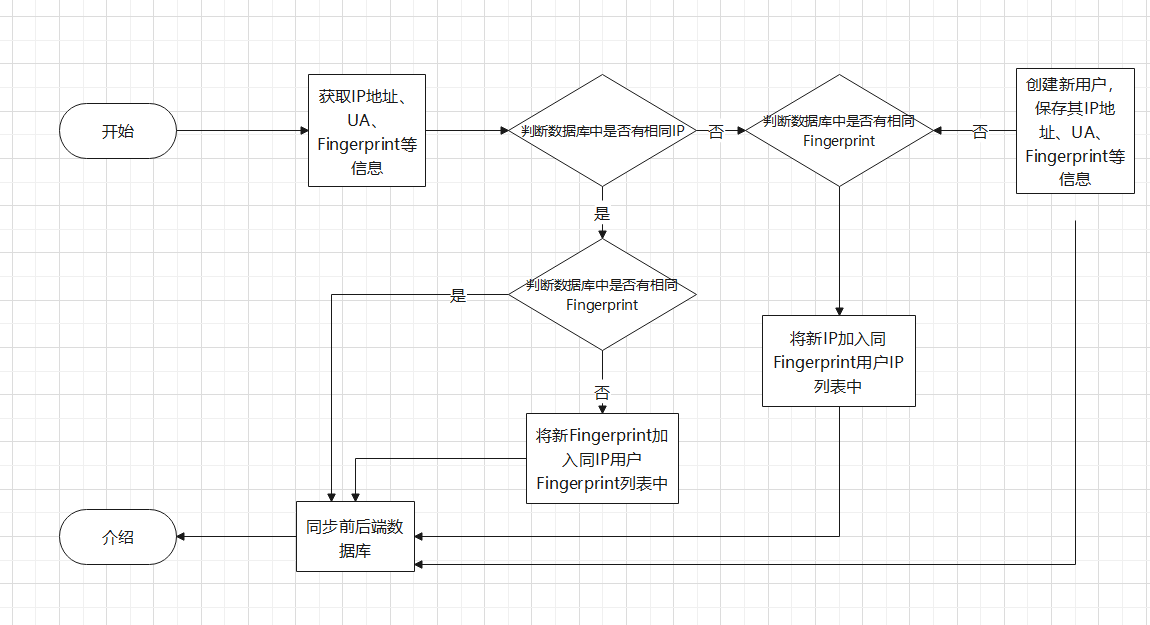
var MongoClient = require("mongodb").MongoClient;

# Fingerprint

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。因此信息熵较大的特征值在计算fingerprint时会有的更大的权重。

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。采用的特征值如下：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。

# 流程



程序流畅如上图所示。

# 代码说明

主要代码集中在usrRec.js中。

saveUser()函数为主要函数，实现了获取用户ip、UA、fingerprint等数据，并判断IndexedDB中是否有相关数据，根据数据情况进行插入、合并与删除。

rmDuplication()函数负责删除单个字段中的重复ip与fingerprint等。

combineUser()函数实习了合并具有相同ip或fingerprint的字段。

可以通过类似如下的方式完成用户识别与存储过程：

mounted: function() {

var usr = new changer([], [], [], {}, {}, {});

saveUser(usr).then(this.syncDB());

},

前后端同步的代码如下：

syncDB: function() {

//打开数据库

var db = new Dexie("user");

db.version(1).stores({

userfps: "++id, \*ip, \*fp",

fingerprints:

"fp, userAgent, cpu, screenPrint, colorDepth, availableResolution, mimeTypes, fonts, timeZone, language, core"

});

//获取userfps表

db.userfps.toArray(arr => {

//发送userfps表数据

this.$axios({

method: "get",

url: "/syncufp",

params: {

arr: arr

}

}).then(res => {

// console.log(res.data);

});

});

//获取fingerprints表数据

db.fingerprints.toArray(arr => {

//发送fingerprints表数据

this.$axios({

method: "get",

url: "/syncfps",

params: {

arr: arr

}

}).then(res => {

// console.log(res.data);

});

});

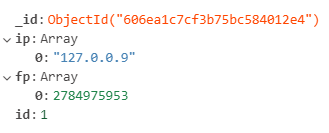
}

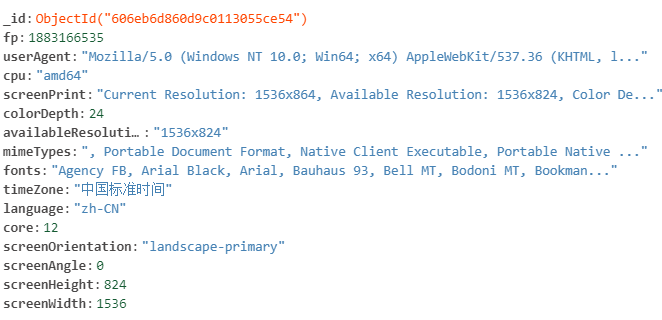
当传输成功时将会返回1001数据。

在mongo.js文件中封装了有关MongoDB的增删改查方法。

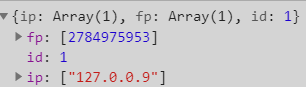
在index.js文件中配置了后端监听与接受前端请求的方法。

MongoDB中的用户与ip、fingerprint表以及fingerprint详细数据表如下图所示。





IndexedDB中的用户与ip、fingerprint表以及fingerprint详细数据表如下图所示。





# 其他说明

关于IndexedDB，由于Websql已经停止维护，因此采用了更为普及的IndexedDB。IndexedDB为Nosql型数据库，因此在后端数据库匹配上采用了同为Nosql型数据库的MongoDB。

关于用户的识别，由于fingerprint并不能完全准确地判断不同浏览器上的同一设备用户，因此还是采用了结合ip地址共同判断用户的方式。将同ip不同fingerprint与同fingerprint不同ip的字段进行整合，看作是同一用户。通过timezone、fonts等其他信息可能可以做更精确的区分。

MongoDB目前的数据存储存在一些重复情况，有待更新。