# 网络信息搜集实验报告

1. 实验目的

l 掌握crawler的工作原理及实现方法；

l 熟悉网页搜集的整个流程及操作步骤；

l 掌握crawler应用程序的编写、调试和运行。

2. 实验报告内容

1) 实验目的、内容与要求及实验环境描述；

实验目的：

学习爬虫的实现。

2) crawler设计思路及总体框架:

目前爬虫功能：

从一个根url开始爬取页面所有的url并且发散到整个网站（可以设置爬取的深度）。

设计思路，利用nodejs的puppeteer库，提供了丰富的浏览器操作api,使无头浏览器打开我们队列中第一个url，然后在打开的网页中利用js拿到所有a标签的引用。放入等待队列，宽度遍历url，实现爬取过程，并且拦截request和response ，实现下载资源的分析和请求的伪造。

3) 程序结构及具体实现的流程分析；

我们程序的主要流程

打开根url页面-》拿到页面所有的url-》放进队列-》while循环爬

拦截request和response

伪造ua和解析下载的资源

我们的程序主要实现了一下功能 我们来分析一下每个功能是如何实现的

1. 尽量采集本地域名（[scut.edu.cn](http://scut.edu.cn)）的所有网页，能提取出其中的URL并加入到待搜集的URL队列中；

使用bfs搜索，维护两个队列，当前要搜索的和下一层的

let waitForUrl = ["http://www2.scut.edu.cn/gzic/"];

let waitForUrlChild = [];

在while循环里面取url，解析a拿新的url，并且做去重和生成日志操作

while (

(waitForUrl.length !== 0 || waitForUrlChild.length !== 0) &&

currdeep <= deep

) {

// console.log(waitForUrl);

// 把下面一层等待被爬取的url放进等待队列

if (waitForUrl.length == 0) {

currdeep++;

waitForUrl = waitForUrlChild;

waitForUrlChild = [];

}

count++;

let url = waitForUrl.shift();

processedurl.push(url);

try {

await page.goto(url);

} catch (err) {

data += `时间:${moment().format(

"YYYY-MM-DD HH:mm:ss.SSS"

)} url:${url} 错误原因:${err}\r\n`;

continue;

}

// 拦截页面的资源请求

await page.setRequestInterception(true);

let myvar = [];

let jsHandle = [];

try {

jsHandle = await page.evaluate(() => {

let alist = document.getElementsByTagName("a");

let res = [];

alist = Array.from(alist);

alist.forEach(e => {

if (e.href) {

res.push(e.href);

}

});

// console.log(new Set(res));

res = Array.from(new Set(res));

return Promise.resolve(res);

});

} catch (err) {}

myvar = jsHandle;

myvar = myvar.filter(e => {

return processedurl.indexOf(e) == -1 && waitForUrl.indexOf(e) == -1;

});

waitForUrlChild = waitForUrlChild.concat(myvar);

console.log("waitForUrl", waitForUrl, "waitForUrlChild", waitForUrlChild);

data += `时间:${moment().format(

"YYYY-MM-DD HH:mm:ss.SSS"

)} 页面url:${url}\n`;

data += `当前爬取的url数量 ：${count}`;

fs.writeFileSync("./result.txt", data);

}

1. 使用User-agent向服务器表明自己的身份（“NIR2019S-”+学号）；

拦截request，重置header的ua

page.on("request", request => {

// Override headers

const headers = request.headers();

headers["user-agent"] = "NIR2019S-201633610790";

request.continue({ headers });

if (consolerquest) {

data += `请求报文头部 : ${request.headers()["user-agent"]}\n`;

console.log(request);

consolerquest = false;

}

});

1. 能对HTML网页进行解析，提取出链接URL，能判别提取的URL是否已处理过，不重复下载和解析已搜集过的网页；

在当前网页利用set做一层去重

res = Array.from(new Set(res));

放到队列里面做第二层去重

myvar = jsHandle;

myvar = myvar.filter(e => {

return processedurl.indexOf(e) == -1 && waitForUrl.indexOf(e) == -1;

});

1. 可对crawler的一些基本参数进行设置，包括：搜集深度、文件类型、文件大小等。

拦截响应拿到资源信息

page.on("response", response => {

let head = response.\_headers;

if (head) {

data += `时间:${moment().format(

"YYYY-MM-DD HH:mm:ss.SSS"

)}\n请求的资源类型：${head["content-type"]}\n请求的资源大小：${

head["content-length"] ? head["content-length"] + "bytes" : "暂无数据"

}\n资源路径:${response.\_url}\n

`;

}

});

深度的实现

利用bfs搜索

// 默认深度 -3层

let deep = 3;

// 当前深度

let currdeep = 1;

当深度满足条件退出循环

while (

(waitForUrl.length !== 0 || waitForUrlChild.length !== 0) &&

currdeep <= deep

)

4) 实验结果分析；

实验日志在result.txt。

总体来说爬取了3层的网页，数量高达上百个url，资源更是不计其数，总结出来就是，爬虫的重要性，因为一个网站的资源之多，是人力不能收集的。

5) crawler的优缺点、以及待改进的地方；

优点：

可以实现header伪造

可以控制爬取的深度（设置deep参数）

可以拦截所有响应 查看页面请求了什么资源

缺点+待改进  
没有实现异步 每次只打开一个页面，但其实也可以打开多个页面不影响（但是对电脑cpu的要求较高，并且代码逻辑里面也要增加异步处理）

6) 在实验过程中遇到的问题，实验的心得体会。

问题：

日志格式的排版（如何更清晰的打印）

错误的处理

异步的处理（保证我们能够收集到异步加载的资源和js动态插入网页的东西）

url队列的生成（深度遍历）

加分项及额外的部分

利用async await的特性，在网页完全打开 ，js执行完成后在进行我们的爬取操作，可以保证可采集由JavaScript、jQuery等动态生成的部分能被采集到

await page.goto(url);

}catch(err){

data += `时间:${moment().format("YYYY-MM-DD HH:mm:ss.SSS")} url:${url} 错误原因:${err}\r\n`;

continue;}

// 拦截页面的资源请求

await page.setRequestInterception(true);

let myvar = [];

let jsHandle = [];

try{

jsHandle = await page.evaluate(() => { //页面打开后才执行我们这边的操作

基于所生成的日志文件，生成如下的统计信息（不限于此）：

1. 平均搜集速度（网页数/秒）根据日志打印出来的文件和打印出来的url数量可得 ：

（2）以首页为例，说明网页的覆盖率（发现的网页链接数与实际链接数的比值）。------------------100% 请求的资源和a标签中引用的链接都能捕捉到

4. 提交内容

程序：包括源程序及注释，执行程序及程序安装使用说明；

注释 在代码里面

程序安装 ：

使用node 8.9.0

在当前程序文件夹下面按顺序运行以下命令

Npm install

Npm run

l URL列表：搜集到的所有规范化URL；

url.txt

l 日志文件：搜集过程的详细记录；

result.txt

l 实验报告：阐述程序设计的思路，说明实验内容的各部分要求是如何实现的，并分析实验结果，总结实验心得及体会。

在前面描述了。看看要不要copy过来还是怎么排版~