link: null
title: 珠峰架构师成长计划
description: 通过一个实例来介绍如何编写网络爬虫抓去掘金数据，并存储到MySQL数据库中，以及定时任务爬虫来更新内容
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=120 sentences=198, words=993

## 1. 掘金爬虫

通过一个实例来介绍如何编写网络爬虫抓去掘金数据，并存储到MySQL数据库中，以及定时任务爬虫来更新内容

## 2. 核心步骤

## 3. 核心类库

```
npm install --save request
```

```
var request = require('request');
request('http://www.baidu.com', function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log(body);
  }
```

```
const request=require('request');
const options={
    url: 'http://localhost:8080/post',
    method:'POST',
    json: true,
    headers: {
        "Content-Type":"application/json"
    },
    body: {name:"zfpx",age:8}
}
request(options,function (error,response,body) {
    if (!error && response.statusCode == 200) {
        console.log(body);
    } else {
        console.error(error);
    }
});
```

```
const request=require('request');
const options={
    url: 'http://localhost:8080/form',
    method:'POST',
    json: true,
    form:{name:'zfpx',age:10}
}
request(options,function (error,response,body) {
    if (!error && response.statusCode == 200) {
        console.log(body);
    } else {
        console.error(error);
    }
});
```

```
const request=require('request');
const fs=require('fs');
var formData = {
    name: 'zfpx',
    avatar:{
      value:  fs.createReadStream('avatar.jpg'),
      options: {
        filename: 'avatar.jpg',
        contentType: 'image/jpeg'
      }
  }
};
request.post({url:'http://localhost:8080/upload', formData}, function (error, response, body) {
    if (!error&&response.statusCode==200) {
        console.log(body);
    } else {
        console.log(error);
    }
})
```

```
npm install cheerio
```

```
let str=`
Hello world
`;
const cheerio=require('cheerio');
const $=cheerio.load(str);
$('h2.title').text('hello there!');
$('h2').addClass('welcome');
console.log($.html());
```

- 选择器在 Context 范围内搜索，Context又在Root范围内搜索。
- root在右，context在左
- selector 和context可以是一个字符串表达式，DOM元素，和DOM元素的数组，或者chreeio对象。
- root 是通常是HTML 文档字符串。

```
$(selectior,[context],[root])
```

```
let html=`

  Apple
  Orange
  Pear

`;

let cheerio=require('cheerio');
let $=cheerio.load(html);
console.log($('.apple','#fruits').text());
```

- 获得和修改属性
- 在匹配的元素中只能获得第一元素的属性。
- 如果设置一个属性的值为null，则移除这个属性
- 你也可以传递一对键值，或者一个函数。

```
attr(name,value)
```

```
console.log($('ul').attr('id'));
$('.apple').attr('id','favorite').attr('class','favorite');
$('.apple').attr({id:'favorite',class:'favorite'});
console.log($('.favorite').html());
console.log($('ul').html());
```

通过name删除属性

```
$('.favorite').removeAttr('id');
```

```
$('input[type="checkbox"]').prop('checked')

$('input[type="checkbox"]').prop('checked', true).val()
```

```
$('').data()

$('').data('apple-color')

var apple = $('.apple').data('kind', 'mac')
apple.data('kind')
```

```
$('input[type="text"]').val()

$('input[type="text"]').val('test').html()
```

检查匹配的元素是否有给出的类名

```
$('.pear').hasClass('pear')
$('apple').hasClass('fruit')
$('li').hasClass('pear')
```

增加class(es)给所有匹配的elements.也可以传函数。

```
$('.pear').addClass('fruit').html()
$('.apple').addClass('fruit red').html()
```

从选择的elements里去除一个或多个有空格分开的class。如果className 没有定义，所有的classes将会被去除，也可以传函数

```
$('.pear').removeClass('pear').html()
$('.apple').addClass('red').removeClass().html()
```

获得一个在匹配的元素中由选择器滤过的后代

```
$('#fruits').find('li').length
```

获得通过选择器筛选匹配的元素的parent集合

```
$('.orange').parents().length
$('.orange').parents('#fruits').length
```

```
$('.apple').next().hasClass('orange')
$('.pear').next().html()
```

获得本元素之后的所有同级元素

```
$('.apple').nextAll()
$('.apple').nextAll().length
```

获得本元素之前的第一个同级元素

```
$('.orange').prev().hasClass('apple')
```

获得本元素前的所有同级元素

```
$('.pear').prevAll()
```

获得选定范围内的元素数组

```
$('li').slice(1).eq(0).text()
$('li').slice(1, 2).length
```

获得被选择的同级元素(除去自己)

```
$('.pear').siblings().length
$('.pear').siblings('.orange').length
$('.pear').siblings('.pear').length
```

会选择chreeio对象的第一个元素

```
('#fruits').children().first().text()
```

会选择chreeio对象的最后一个元素

```
$('#fruits').children().last().text()
```

通过索引筛选匹配的元素。使用.eq(-i)就从最后一个元素向前数。

```
$('li').eq(0).text()
$('li').eq(-1).text()
```

获被选择元素的子元素

```
$('#fruits').children().length
$('#fruits').children('.pear').text()
```

迭代一个cheerio对象，为每个匹配元素执行一个函数。要提早跳出循环，返回false.

```
var fruits = [];

$('li').each(function(i, elem) {
  fruits[i] = $(this).text();
});

fruits.join(', ');
```

迭代一个cheerio对象，为每个匹配元素执行一个函数。Map会返回一个迭代结果的数组。

```
$('li').map(function(i, el) {
  return $(this).attr('class');
}).join(', ');
```

- 迭代一个cheerio对象，滤出匹配选择器或者是传进去的函数的元素。
- 如果使用函数方法，这个函数在被选择的元素中执行，所以this指向的手势当前元素。

```
$('li').filter('.orange').attr('class');

$('li').filter(function(i, el) {

   return $(this).attr('class') === 'orange';
}).attr('class');
```

在每个元素最后插入一个子元素

```
$('ul').append('Plum')
$.html()
```

在每个元素最前插入一个子元素

```
$('ul').prepend('Plum')
$.html()
```

在每个匹配元素之后插入一个元素

```
$('.apple').after('Plum')
$.html()
```

在每个匹配的元素之前插入一个元素

```
$('.apple').before('Plum')
$.html()
```

从DOM中去除匹配的元素和它们的子元素。选择器用来筛选要删除的元素。

```
$('.pear').remove()
$.html()
```

替换匹配的的元素

```
var plum = $('Plum')
$('.pear').replaceWith(plum)
$.html()
```

清空一个元素，移除所有的子元素

```
$('ul').empty()$.html()
```

获得元素的HTML字符串。如果htmlString有内容的话，将会替代原来的HTML

```
$('.orange').html()

$('#fruits').html('Mango').html()
```

获得元素的text内容，包括子元素。如果textString被指定的话，每个元素的text内容都会被替换。

```
$('.orange').text()
$('ul').text()
```

- 在编写程序的时候，有时候需要输出一些调试信息，以便排查问题。
- 但是在程序运行过程中又不需要这些信息，为了方便切换而且不需要改代码，可以使用debug模块

```
let debug = require('debug')('app:main');
debug('现在的时间是%s',new Date());
```

- Window系统在命令行中执行 `SET DEBUG=app:*`
- Mac系统在命令行中执行 `export DEBUG=app:*`

符号 含义 星号() *代表所有可能的值 逗号(,) 可以用逗号隔开的值指定一个列表范围，例如，"1,2,5,7,8,9" 中杠(-) 可以用整数之间的中杠表示一个整数范围，例如"2-6"表示"2,3,4,5,6" 正斜线(/) 可以用正斜线指定时间的间隔频率，*/10，如果用在minute字段，表示每十分钟执行一次 单位 范围 Seconds 0-59 Minutes 0-59 Hours 0-23 Day 1-31 Months 0-11 Day of Week 0-6

```
var cronJob = require('cron').CronJob;
var job1 = new cronJob("* * * * * *",function(){
  console.log('每秒');
});
job1.start();
```

- 大部分情况下，异步的IO操作发生的错误无法被try catch捕获，如果没有捕获会导致程序退出
- 在Node.js中，如果一个抛出的异常没有被try catch捕获，会尝试将错误交给uncaughtException事件处理函数来进行处理，仅当没有注册该事件处理函数时才会导致进程直接退出。

```
process.on('uncaughtException',function (err) {
    console.error('uncaughtException: %s',erro.stack);
});
```

pm2是一个功能强大的进程管理器，通过 `pm2 start`来启动程序, 当该进程异常退出时,pm2会自动尝试重启进程。

```
npm install pm2 -g
npm2 start
pm2 stop
```

```
var request = require('request');
var iconv=require('iconv-lite');
let cheerio=require('cheerio');
request({url: 'http://top.baidu.com/buzz?b=26&c=1&fr=topcategory_c1'
, encoding: null},function(err,response,body){
    if(err)
        console.error(err);
    body = iconv.decode(body, 'gbk').toString();
    let $=cheerio.load(body);
    let movies=[];
    $('.keyword .list-title').each((index,item) => {
        let movie=$(item);
        movies.push({
            name:movie.text()
        });
    });
    console.log(movies);
})
```

```
const nodemailer = require('nodemailer');
let transporter = nodemailer.createTransport({

    service: 'qq',
    port: 465,
    secureConnection: true,
    auth: {
        user: '83687401@qq.com',

        pass: 'gfndwuvvfpbebjdi',
    }
});

let mailOptions = {
    from: '"83687401" ',
    to: '83687401@qq.com',
    subject: 'hello',

    html: 'Hello world'
};

transporter.sendMail(mailOptions, (error, info) => {
    if (error) {
        return console.log(error);
    }
    console.log('Message sent: %s', info.messageId);

});
```

### 3.9 HTTP代理工具

- Windows 平台有 Fiddler，macOS 有 Charles，阿里有AnyProxy
- 基本原理就是通过在手机客户端设置好代理IP和端口,客户端所有的 HTTP、HTTPS 请求就会经过代理工具
- Tools > Fiddler Options > Connections
    - Fiddler listens on port 8888
    - Allow remote computers to connect
- Tools > Fiddler Options > HTTPS > Decrypt HTTPS traffic
    - Capture HTTPS CONNECTs
    - Decrypt HTTPS traffic

## 参考