一、实验介绍

1.1 实验内容

本课程通过 Node.js 实现一个简单的爬虫，爬取豆瓣热门电影。主要会用到的模块（包）有：http，fs，path，cheerio (https://github.com/cheeriojs/cheerio.git)。http 模块用于创建 http 请求，fs 模块用于保存文件，path 模块用于解析路径，cheerio 包是服务器端的 jQuery 实现，这里用于解析 HTML。

1.2 实验知识点

- http 模块
- fs 模块
- path 模块
- cheerio

1.3 实验环境

- Node.js 6.x

1.4 适合人群

本课程难度一般，属于初级级别课程，适合具有 JavaScript 基础的用户，学习 Node.js 基础知识。

二、实验原理

本课程爬虫的基本原理很简单，就是请求待爬取页面的 URL，下载页面内容，然后解析下载的页面内容，处理内容数据。

三、实验步骤

3.1 初始化项目

创建文件夹 `spider` ，然后进入此文件夹，输入以下命令初始化项目：

```
$ npm init
```

初始化项目时需要填一些项目基本信息，如果不想填或不知道填什么，一路回车即可。这一步会生成一个 `package.json` 文件，保存项目的配置信息和模块依赖信息。

添加第三方包（此项目我们只用到了一个三方包，cheerio）：

```
$ npm install cheerio --save
```

安装三方包的时候，添加 `--save` 参数，项目对于此包的依赖就会写入 `package.json` 文件中。

然后创建 data 文件夹和 img 文件夹，data 文件夹用于保存抓取的数据，img 用于保存抓取的图片。

然后创建项目主文件 `spider.js` ，下面写的爬虫程序就放在这个文件了。

3.2 HTTP模块

在这个项目中我们主要用到的是 HTTP 模块，http 模块中共有两个属性，四个 Class，四个方法以及两个对象：

```
http.METHODS // 返回解析器支持的所有HTTP方法
http.STATUS_CODES // 返回所有状态码及其说明
Class: http.Server
Class: http.ServerResponse
Class: http.Agent
Class: http.ClientRequest
http.createServer([requestListener]) // 用于创建 http 服务
http.createClient([port][, host]) // 用于发送 http 请求
http.request(options[, callback]) // 用于发送 http 请求
http.get(options[, callback]) // 用于发送 http get 请求
http.globalAgent
http.IncomingMessage
```

`http.createServer([requestListener])` 方法用于创建**http**服务，返回值是 `Class: http.Server` 的一个实例，如：

```
// 这里 server 就是 Class: http.Server 的一个实例
// 是由 http.createServer 返回的
var server = http.createServer(function(req, res) {
    // 回调函数中的参数 req, res
    // req 是 http.IncomingMessage 对象的一个实例
    // res 是 Class: http.ServerResponse 的一个实例
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('okay');
});

// 另外一种等效的方法
// 通过 Class 创建web服务
var server2 = new http.Server();
server2.on('request', function(req, res) {
    res.writeHeader(200, {'Content-Type': 'text/plain'});
    res.end('okay');
});
```

一个 **http.IncomingMessage** 是由 **http.Server** 或者 **http.ClientRequest** 创建，并且作为第一个参数传递给其 **request** 事件或 **response** 事件，可以用来访问响应状态，**headers** 和数据。**http.ServerResponse** 的实例对象由 **HTTP** 服务端创建（而不是客户端或用户），作为 **request** 事件的第二个参数，用于处理客户端的**http**请求，如上例中发送了**“okay”**字符串给客户端。

`http.createClient([port], [ host])` 方法已经被弃用了，新方法是 `http.request` 和 `http.get` ，用于发送**http**请求， `http.get` 是 `http.request` 的简化版，只能发送**GET**请求。返回的都是 `Class: http.ClientRequest` 的一个实例。在此次项目中，我们主要用到的就是 `http.get` 方法来抓取数据。

3.3 编写爬虫程序

因为豆瓣网使用了 **https**，所以我们需要使用 **https** 模块，**https** 模块的功能和 **http** 模块相同。

打开 `spider.js` 文件，输入如下代码：

```
'use strict';

// 引入模块
var https = require('https');
var fs = require('fs');
var path = require('path');
var cheerio = require('cheerio');

// 爬虫的URL信息
var opt = {
    hostname: 'movie.douban.com',
    path: '/top250',
    port: 443
};

// 创建http get请求
https.get(opt, function(res) {
    var html = ''; // 保存抓取到的HTML源码
    var movies = [];  // 保存解析HTML后的数据，即我们需要的电影信息

    // 前面说过
    // 这里的 res 是 Class: http.IncomingMessage 的一个实例
    // 而 http.IncomingMessage 实现了 stream.Readable 接口
    // 所以 http.IncomingMessage 也有 stream.Readable 的事件和方法
    // 比如 Event: 'data', Event: 'end', readable.setEncoding() 等

    // 设置编码
    res.setEncoding('utf-8');

    // 抓取页面内容
    res.on('data', function(chunk) {
        html += chunk;
    });

    res.on('end', function() {
        // 使用 cheerio 加载抓取到的HTML代码
        // 然后就可以使用 jQuery 的方法了
        // 比如获取某个class：$('.className')
        // 这样就能获取所有这个class包含的内容
        var $ = cheerio.load(html);

        // 解析页面
        // 每个电影都在 item class 中
        $('.item').each(function() {
            var movie = {
                title: $('.title', this).text(), // 获取电影名称
                star: $('.star .rating_num', this).text(), // 获取电影评分
                link: $('a', this).attr('href'), // 获取电影详情页链接
                picUrl: $('.pic img', this).attr('src') // 获取电影图片链接
            };
            // 把所有电影放在一个数组里面
            movies.push(movie);
        });

        console.log(movies);
    });
}).on('error', function(err) {
    console.log(err);
});
```

上面的代码中，我们使用了 `http.get()` 方法来获取页面数据， `http.get()` 是 `http.request()` 方法的简化版，不同的是， `http.get()` 只能发送**get**请求，并且会自动调用 `req.end()` 方法。而 `http.request()` 就需要手动调用 `req.end()` 方法。

运行程序：

```
$ node spider.js
```

运行程序，我们可以看到，抓取到的电影信息都打印出来了，说明抓取成功了。下面我们就把数据保存到本地文件中。

3.4 保存数据到本地

上面的程序只会把抓取到的数据直接打印出来，现在我们需要把这些数据保存下来：

```
/**
 * 保存数据到本地
 *
 * @param {string} path 保存数据的文件夹
 * @param {array} movies 电影信息数组
 */
function saveData(path, movies) {
    // 调用 fs.writeFile 方法保存数据到本地
    // fs.writeFile(filename, data[, options], callback)
    // fs.writeFile 方法第一个参数是需要保存在本地的文件名称（包含路径）
    // 第二个参数是文件数据
    // 然后有个可选参数，可以是 encoding，mode 或者 flag
    // 最后一个 参数是一个回调函数
    fs.writeFile(path, JSON.stringify(movies, null, 4), function(err) {
        if (err) {
            return console.log(err);
        }
        console.log('Data saved');
    });
}
```

如果想要下载电影的图片的话，需要新建一个**http**请求，因为每张图片都是一个新的链接，抓取到图片后我们可以通过**fs**模块保存图片到本地，代码如下：

```
/**
 * 下载图片
 *
 * @param {string} imgDir 存放图片的文件夹
 * @param {string} url 图片的URL地址
 */
function downloadImg(imgDir, url) {
    https.get(url, function(res) {
        var data = '';

        res.setEncoding('binary');

        res.on('data', function(chunk) {
            data += chunk;
        });

        res.on('end', function() {
            // 调用 fs.writeFile 方法保存图片到本地
            // path.basename(url) 可以得到链接指向的文件名
            // 如：path.basename('http://localhost/img/2.jpg') => '2.jpg'
            fs.writeFile(imgDir + path.basename(url), data, 'binary', function(err) {
                if (err) {
                    return console.log(err);
                }
                console.log('Image downloaded: ', path.basename(url));
            });
        });
    }).on('error', function(err) {
        console.log(err);
    });
}
```

然后我们在爬虫程序中调用这两个方法来保存数据，最终 `spider.js` 的代码如下所示：

```
'use strict';

// 引入模块
var https = require('https');
var fs = require('fs');
var path = require('path');
var cheerio = require('cheerio');

// 爬虫的URL信息
var opt = {
    hostname: 'movie.douban.com',
    path: '/top250',
    port: 443
};

// 创建http get请求
https.get(opt, function(res) {
    var html = ''; // 保存抓取到的HTML源码
    var movies = [];  // 保存解析HTML后的数据，即我们需要的电影信息

    // 前面说过
    // res 是 Class: http.IncomingMessage 的一个实例
    // 而 http.IncomingMessage 实现了 stream.Readable 接口
    // 所以 http.IncomingMessage 也有 stream.Readable 的事件和方法
    // 比如 Event: 'data', Event: 'end', readable.setEncoding() 等

    // 设置编码
    res.setEncoding('utf-8');

    // 抓取页面内容
    res.on('data', function(chunk) {
        html += chunk;
    });

    res.on('end', function() {
        // 使用 cheerio 加载抓取到的HTML代码
        // 然后就可以使用 jQuery 的方法了
        // 比如获取某个class：$('.className')
        // 这样就能获取所有这个class包含的内容
        var $ = cheerio.load(html);

        // 解析页面
        // 每个电影都在 item class 中
        $('.item').each(function() {
            // 获取图片链接
            var movie = {
                title: $('.title', this).text(), // 获取电影名称
                star: $('.info .star em', this).text(), // 获取电影评分
                link: $('a', this).attr('href'), // 获取电影详情页链接
                picUrl: $('.pic img', this).attr('src') // 获取电影图片链接
            };

            // 把所有电影放在一个数组里面
            movies.push(movie);
            // 下载图片
            downloadImg('img/', movie.picUrl);
        });

        // 保存抓取到的电影数据
        saveData('data/data.json', movies);
    });
}).on('error', function(err) {
    console.log(err);
});


/**
 * 保存数据到本地
 *
 * @param {string} path 保存数据的文件
 * @param {array} movies 电影信息数组
 */
function saveData(path, movies) {
    // 调用 fs.writeFile 方法保存数据到本地
    fs.writeFile(path, JSON.stringify(movies, null, 4), function(err) {
        if (err) {
```

```
            return console.log(err);
        }
        console.log('Data saved');
    });
}

/**
 * 下载图片
 *
 * @param {string} imgDir 存放图片的文件夹
 * @param {string} url 图片的URL地址
 */
function downloadImg(imgDir, url) {
    https.get(url, function(res) {
        var data = '';

        res.setEncoding('binary');

        res.on('data', function(chunk) {
            data += chunk;
        });

        res.on('end', function() {
            // 调用 fs.writeFile 方法保存图片到本地
            fs.writeFile(imgDir + path.basename(url), data, 'binary', function(err) {
                if (err) {
                    return console.log(err);
                }
                console.log('Image downloaded: ', path.basename(url));
            });
        });
    }).on('error', function(err) {
        console.log(err);
    });
}
```

OK，一个简单的爬虫就实现了，大家可以自行扩展更加丰富的功能，编写一个更加高级的爬虫程序。

四、总结

本课程通过 Node.js 实现一个简单的爬虫，爬取豆瓣热门电影。主要会用到的模块（包）有：**http**，**fs**，**path**，**cheerio**。http 模块用于创建 http 请求，fs 模块用于保存文件，path 模块用于解析路径，cheerio 包是服务器端的 jQuery 实现，这里用于解析 HTML。

课程教师

forever

共发布过**8**门课程

查看老师的所有课程 > (/teacher/45)

前置课程

HTML基础入门 (/courses/19)

Node.js 教程 (/courses/44)

Javascript基础（新版） (/courses/21)

进阶课程

Node.js 经典项目实战 (/courses/455)