

网络爬虫框架

webMagic

什么叫网络爬虫：

网络爬虫（又被称为网页蜘蛛，网络机器人，网页追逐者等），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁、自动索引、模拟程序或者蠕虫。

通俗的讲就是：访问网站并将网站内容下载下来的程序或脚本。

常用框架：Nutch、WebCollector、WebMagic、crawler4j等等

HttpClient:

用来提供高效的、最新的、功能丰富的支持HTTP协议的客户端编程工具包

使用步骤:

- 1、创建HttpClient对象;
- 2、创建请求方法的实例, 并指定请求URL。(HttpGet/HttpPost对象);
- 3、设置请求参数;
- 4、调用HttpClient对象的execute方法发送请求;
- 5、获取Response读取相关内容;
- 6、释放连接;

Jsoup

jsoup 是一款Java 的HTML解析器，可直接解析某个URL地址、HTML 文本内容。

它提供了一套非常省力的API，可通过DOM，CSS以及类似于jQuery 的操作方法来取出和操作数据。

```

86 //使用Jsoup, 获取百度logo图像地址
87 @Test
88 public void getBaiduLogoByJsoup() {
89     try {
90         Document doc = Jsoup.connect("http://www.baidu.com/").get();
91         Element htmlBody = doc.body();
92         Element imgDiv = htmlBody.getElementById("lg");
93         System.out.println("imgDiv内容: ");
94         System.out.println(imgDiv);
95         String regex = "hidefocus.+?src=\"//(.+?)\"\\s";
96         System.out.println("logo地址: "
97             +RegexStringUtils.regexString(imgDiv.toString(), regex));
98     } catch (IOException e) {

```

goByJsoup

self. 780ms
ByJsoup 780ms

1 test passed

/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java .

imgDiv内容:

<div id="lg">

logo地址: www.baidu.com/img/bd_logo1.png

为什么选择WebMagic

- 1、一个框架，一个领域
- 2、微内核和高可扩展性
- 3、注重实用性
- 4、国产，中文文档，文档详细

下载：







最新版本：

<https://github.com/code4craft/webmagic>

稳定版本：

<http://git.oschina.net/flashsword20/webmagic>

maven项目， import项目时选择作为Maven项目

- ▶  webmagic-core
- ▶  webmagic-extension
- ▶  webmagic-samples
- ▶  webmagic-saxon
- ▶  webmagic-scripts
- ▶  webmagic-selenium

两个主要包

webmagic-core: 核心部分, 包含爬虫基本模块和基本抽取器

webmagic-extension: 扩展模块, 包括注解格式定义爬虫、JSON等支持

四个组件

1.Downloader

负责从互联网上下载页面,WebMagic默认使用了Apache HttpClient作为下载工具

2.PageProcessor

负责解析页面, 抽取有用信息, 以及发现新的链接。WebMagic使用Jsoup作为HTML解析工具, 并基于其开发了解析XPath的工具Xsoup。

四个组件

3.Scheduler

负责管理待抓取的URL，及去重url的工作。WebMagic默认提供了JDK的内存队列来管理URL，并用Set集合来进行去重。除非项目有一些特殊的分布式需求，否则无需自己定制Scheduler。

4.Pipeline

负责抽取结果的处理，包括计算、持久化到文件、数据库等

用于数据流转的对象

Request： 是对URL地址的一层封装；

Page： 从Downloader下载到的一个页面（html/json/其他文本）

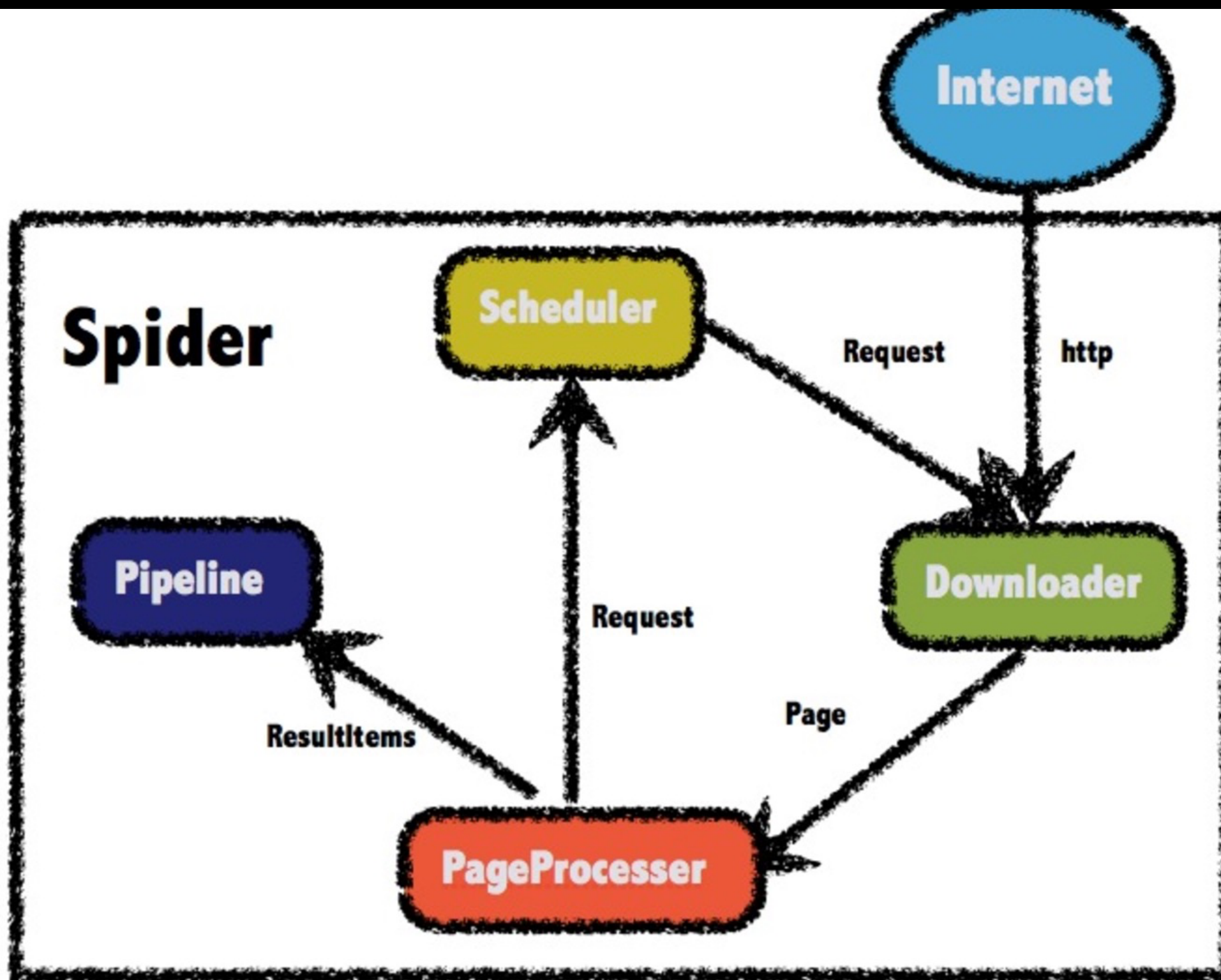
ResultItems： 像Map，保存PageProcessor处理的结果，供Pipeline使用（字段skip，为true，则不被Pipeline处理）。

控制爬虫运转的核心：Spider

1、Downloader、PageProcessor、Scheduler、Pipeline都是Spider的一个属性，通过设置这些属性可以实现不同的功能；

2、Spider也是WebMagic操作的入口，它封装了爬虫的创建、启动、停止、多线程等功能。

webmagic总体架构:



```
public static void main(String[] args) {  
    Spider.create(new GithubRepoPageProcessor())  
        //从https://github.com/code4craft开始抓  
        .addUrl("https://github.com/code4craft")  
        //设置Scheduler, 使用Redis来管理URL队列  
        .setScheduler(new RedisScheduler("localhost"))  
        //设置Pipeline, 将结果以json方式保存到文件  
        .addPipeline(new JsonFilePipeline("D:\\data\\webmagic"))  
        //开启5个线程同时执行  
        .thread(5)  
        //启动爬虫  
        .run();  
}
```

编写基本的爬虫步骤：

- 1、实现PageProcessor
- 2、使用Pipeline保存结果
- 3、爬虫的配置、启动和终止

步骤一：定制PageProcessor

- 1、站点的配置

- 2、页面元素的抽取

- 3、链接的发现


```
public class GithubRepoPageProcessor implements PageProcessor {
```

```
// 部分一：抓取网站的相关配置，包括编码、抓取间隔、重试次数等
```

```
private Site site = Site.me().setRetryTimes(3).setSleepTime(1000);
```

```
@Override
```

```
// process是定制爬虫逻辑的核心接口，在这里编写抽取逻辑
```

```
public void process(Page page) {
```

```
// 部分二：定义如何抽取页面信息，并保存下来
```

```
page.putField("author", page.getUrl().regex("https://github\\.com/(\\w+)/.*").toString());
```

```
page.putField("name", page.getHtml().xpath("//h1[@class='entry-title public']/strong/a/text()"));
```

```
if (page.getResultItems().get("name") == null) {
```

```
    //skip this page
```

```
    page.setSkip(true);
```

```
}
```

```
page.putField("readme", page.getHtml().xpath("//div[@id='readme']/tidyText()"));
```

```
// 部分三：从页面发现后续的url地址来抓取
```

```
page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/[\\w\\-]+)/[\\w\\-]+/"));
```

```
}
```

爬虫的配置:

包括编码、抓取间隔、超时时间、重试次数等;

页面元素的抽取:

XPath: `page.getHtml().xpath("//div[@class='title']")`

CSS选择器: `page.getHtml().$("div.title");`

正则表达式: `page.getHtml().links().regex("(https://github\\.com/\\w+/\\w+)")`

JsonPathSelector: 用于从Json中快速定位一条内容

链接的发现:

`page.addTargetRequests(urls)`: 则将这些链接加入到待抓取的队列中去

`page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/\\w+/\\w+)").all());`

Selectable接口： 抽取 / 获取结果

1、抽取：

page.getHtml()返回的是一个Html对象， 它实现了Selectable接口

XPath: page.getHtml().xpath("//div[@class='title']")

2、获取结果：

方法	说明	示例
get()	返回一条String类型的结果	String link= html.links().get()
toString()	功能同get(), 返回一条String类型的结果	String link= html.links().toString()
all()	返回所有抽取结果	List links= html.links().all()
match()	是否有匹配结果	if (html.links().match()){ xxx; }

步骤二、使用Pipeline保存结果

- 1、实现Pipeline接口就可以使用
- 2、注解模式下可以实现PageModelPipeline接口

```
public interface Pipeline {  
    public void process(ResultItems resultItems, Task task);  
}
```

```
public interface PageModelPipeline<T> {  
    public void process(T t, Task task);  
}
```

WebMagic提供的几个Pipeline

类	说明	备注
ConsolePipeline	输出结果到控制台	抽取结果需要实现toString方法
FilePipeline	保存结果到文件	抽取结果需要实现toString方法
JsonFilePipeline	JSON格式保存结果到文件	
ConsolePageModelPipeline	(注解模式)输出结果到控制台	
FilePageModelPipeline	(注解模式)保存结果到文件	
JsonFilePageModelPipeline	(注解模式)JSON格式保存结果到文件	想要持久化的字段需要有getter方法

步骤三、爬虫的配置和启动

Spider：是爬虫启动的入口。在启动爬虫之前，我们需要使用一个PageProcessor
创建一个Spider对象，然后使用run()进行启动。同时Spider的其他组件
(Downloader、Scheduler、Pipeline) 都可以通过set方法来进行设置。

方法	说明
create(PageProcessor)	创建Spider
addUrl(String...)	添加初始的URL
addRequest(Request...)	添加初始的Request(处理非HTTP GET请求)
thread(n)	开启n个线程
start()	启动
addPipeline(Pipeline)	添加一个Pipeline， 一个Spider可以有多个Pipeline
setScheduler(Scheduler)	设置Scheduler， 一个Spider只能有个一个Scheduler
setDownloader(Downloader)	设置Downloader， 一个Spider只能有个一个Downloader

Site：对站点本身的一些配置信息，例如编码、HTTP头、超时时间、重试策略等、代理等，都可以通过设置Site对象来进行配置。

方法	说明
setCharset(String)	设置编码
setSleepTime(int)	设置休眠时间
setUserAgent(String)	设置UserAgent
setTimeout(int)	设置超时时间，单位是毫秒
setRetryTimes(int)	设置重试次数
addCookie(String,String)	添加一条cookie
setDomain(String)	设置域名，需设置域名后， addCookie才可生效
setHttpProxy(HttpHost)	设置Http代理(已废弃)

配置代理

HttpClientDownloader.setProxyProvider(ProxyProvider proxyProvider)

ProxyProvider有一个默认实现：SimpleProxyProvider

1. 设置单一的普通HTTP代理为101.101.101.101的8888端口，并设置密码为"username","password"

```
HttpClientDownloader httpClientDownloader = new HttpClientDownloader();
httpClientDownloader.setProxyProvider(SimpleProxyProvider.from(new Proxy("101.101.101.101",8888,"username","password"));
spider.setDownloader(httpClientDownloader);
```

1. 设置代理池，其中包括101.101.101.101和102.102.102.102两个IP，没有密码

```
HttpClientDownloader httpClientDownloader = new HttpClientDownloader();
httpClientDownloader.setProxyProvider(SimpleProxyProvider.from(
    new Proxy("101.101.101.101",8888),
    new Proxy("102.102.102.102",8888)));
```


注解的使用：

@HelpUrl：对于博客页，HelpUrl是列表页，TargetUrl是文章页

@TargetUrl：是我们最终要抓取的URL

比如对于博客页，HelpUrl是列表页，TargetUrl是文章页

```
@TargetUrl("https://github.com/\\w+/\\w+")
@HelpUrl("https://github.com/\\w+")
public class GithubRepo {
    .....
}
```

WebMagic自己定制的适合URL的正则表达式，主要由两点改动：

- 1、将URL中常用的字符默认做了转义，变成了\.
- 2、将"*"替换成了".*"，直接使用可表示通配符

例如，<https://github.com/>*在这里是一个合法的表达式，
它表示<https://github.com/>下的所有URL。

@ExtractBy: 主要作用于字段, 它表示“使用这个抽取规则, 将抽取到的结果保存到这个字段中”

```
@ExtractBy("//div[@id='readme']/text()")  
private String readme;
```

@Formatter: 根据字段类型进行转换

```
@Formatter(value = "", subClazz = Integer.class)
@ExtractBy(value = "//div[@class='id']/text()")
private List<Integer> ids;
```

```
@TargetUrl("https://github.com/\\w+/\\w+")
@HelpUrl("https://github.com/\\w+")

public class GithubRepo {

    @ExtractBy(value = "//h1[@class='entry-title public']/strong/a/text()", notNull = true)
    private String name;

    @ExtractByUrl("https://github\\.com/(\\w+)/.*")
    private String author;

    @ExtractBy("//div[@id='readme']/tidyText()")
    private String readme;

    public static void main(String[] args) {
        OOSpider.create(Site.me().setSleepTime(1000)
            , new ConsolePageModelPipeline(), GithubRepo.class)
            .addUrl("https://github.com/code4craft").thread(5).run();
    }
}
```

案例：

- 1、列表+详情页面（含模拟登录）
- 2、抓取前端js渲染的页面

谢谢大家，再见