# WebMagic介绍

昨天完成了爬虫的入门的学习，是一个最基本的爬虫案例，今天我们要学习一款爬虫框架的使用就是WebMagic。其底层用到了我们上一天课程所使用的HttpClient和Jsoup，让我们能够更方便的开发爬虫。

WebMagic项目代码分为核心和扩展两部分。核心部分(webmagic-core)是一个精简的、模块化的爬虫实现，而扩展部分则包括一些便利的、实用性的功能。

WebMagic的设计目标是尽量的模块化，并体现爬虫的功能特点。这部分提供非常简单、灵活的API，在基本不改变开发模式的情况下，编写一个爬虫。

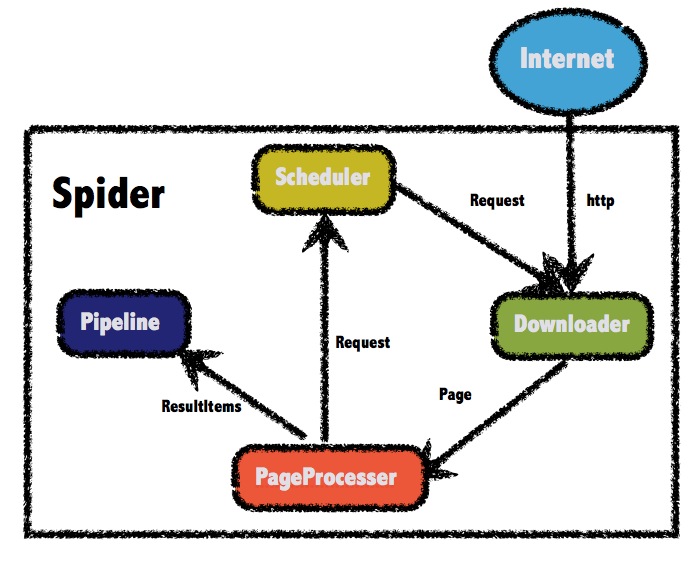
扩展部分(webmagic-extension)提供一些便捷的功能，例如注解模式编写爬虫等。同时内置了一些常用的组件，便于爬虫开发。

## 架构介绍

WebMagic的结构分为Downloader、PageProcessor、Scheduler、Pipeline四大组件，并由Spider将它们彼此组织起来。这四大组件对应爬虫生命周期中的下载、处理、管理和持久化等功能。WebMagic的设计参考了Scapy，但是实现方式更Java化一些。

而Spider则将这几个组件组织起来，让它们可以互相交互，流程化的执行，可以认为Spider是一个大的容器，它也是WebMagic逻辑的核心。

WebMagic总体架构图如下：



### WebMagic的四个组件

1.Downloader

Downloader负责从互联网上下载页面，以便后续处理。WebMagic默认使用了Apache HttpClient作为下载工具。

2.PageProcessor

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。

在这四个组件中，PageProcessor对于每个站点每个页面都不一样，是需要使用者定制的部分。

3.Scheduler

Scheduler负责管理待抓取的URL，以及一些去重的工作。WebMagic默认提供了JDK的内存队列来管理URL，并用集合来进行去重。也支持使用Redis进行分布式管理。

4.Pipeline

Pipeline负责抽取结果的处理，包括计算、持久化到文件、数据库等。WebMagic默认提供了“输出到控制台”和“保存到文件”两种结果处理方案。

Pipeline定义了结果保存的方式，如果你要保存到指定数据库，则需要编写对应的Pipeline。对于一类需求一般只需编写一个Pipeline。

### 用于数据流转的对象

1. Request

Request是对URL地址的一层封装，一个Request对应一个URL地址。

它是PageProcessor与Downloader交互的载体，也是PageProcessor控制Downloader唯一方式。

除了URL本身外，它还包含一个Key-Value结构的字段extra。你可以在extra中保存一些特殊的属性，然后在其他地方读取，以完成不同的功能。例如附加上一个页面的一些信息等。

2. Page

Page代表了从Downloader下载到的一个页面——可能是HTML，也可能是JSON或者其他文本格式的内容。

Page是WebMagic抽取过程的核心对象，它提供一些方法可供抽取、结果保存等。

3. ResultItems

ResultItems相当于一个Map，它保存PageProcessor处理的结果，供Pipeline使用。它的API与Map很类似，值得注意的是它有一个字段skip，若设置为true，则不应被Pipeline处理。

## 入门案例

### 加入依赖

创建Maven工程，并加入以下依赖

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>cn.itcast.crawler</groupId>  
 <artifactId>itcast-crawler-webmagic</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <!--WebMagic-->  
 <dependency>  
 <groupId>us.codecraft</groupId>  
 <artifactId>webmagic-core</artifactId>  
 <version>0.7.3</version>  
 </dependency>  
 <dependency>  
 <groupId>us.codecraft</groupId>  
 <artifactId>webmagic-extension</artifactId>  
 <version>0.7.3</version>  
 </dependency>  
 </dependencies>  
   
</project>

注意：0.7.3版本对SSL的并不完全，如果是直接从Maven中央仓库下载依赖，在爬取只支持SSL v1.2的网站会有SSL的异常抛出。

解决方案：

* + - 1. 等作者的0.7.4的版本发布
      2. 直接从github上下载最新的代码，安装到本地仓库

也可以参考以下资料自己修复

https://github.com/code4craft/webmagic/issues/701

### 加入配置文件

WebMagic使用slf4j-log4j12作为slf4j的实现。

添加log4j.properties配置文件

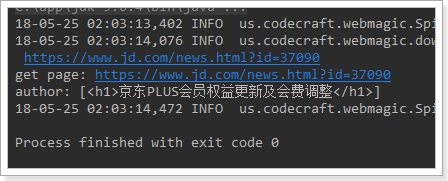
log4j.rootLogger=INFO,A1   
  
log4j.appender.A1=org.apache.log4j.ConsoleAppender  
log4j.appender.A1.layout=org.apache.log4j.PatternLayout  
log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n

### 案例实现

public class JobProcessor implements PageProcessor {  
  
 public void process(Page page) {  
 page.putField("author", page.getHtml().css("div.mt>h1").all());  
 }

private Site site = Site.*me*();  
 public Site getSite() {  
 return site;  
 }  
  
 public static void main(String[] args) {  
 Spider.*create*(new JobProcessor())  
 //初始访问url地址  
 .addUrl("https://www.jd.com/moreSubject.aspx")   
 .run();  
 }  
}

打印结果：



# WebMagic功能

## 实现PageProcessor

### 抽取元素Selectable

WebMagic里主要使用了三种抽取技术：XPath、正则表达式和CSS选择器。另外，对于JSON格式的内容，可使用JsonPath进行解析。

XPath

以上是获取属性class=mt的div标签，里面的h1标签的内容

page.getHtml().xpath("//div[@class=mt]/h1/text()")

也可以参考课堂资料的W3School离线手册(2017.03.11版).chm

CSS选择器

CSS选择器是与XPath类似的语言。在上一次的课程中，我们已经学习过了Jsoup的选择器，它比XPath写起来要简单一些，但是如果写复杂一点的抽取规则，就相对要麻烦一点。

div.mt>h1表示class为mt的div标签下的直接子元素h1标签

page.getHtml().css("div.mt>h1").toString()

可是使用:nth-child(n)选择第几个元素，如下选择第一个元素

page.getHtml().css("div#news\_div > ul > li:nth-child(1) a").toString()

注意：需要使用>，就是直接子元素才可以选择第几个元素

正则表达式

正则表达式则是一种通用的文本抽取语言。在这里一般用于获取url地址。

正则表达式学习难度要大一些，大家可以参考课堂资料《正则表达式系统教程.CHM》

### 抽取元素API

Selectable相关的抽取元素链式API是WebMagic的一个核心功能。使用Selectable接口，可以直接完成页面元素的链式抽取，也无需去关心抽取的细节。

在刚才的例子中可以看到，page.getHtml()返回的是一个Html对象，它实现了Selectable接口。这个接口包含的方法分为两类：抽取部分和获取结果部分。

| **方法** | **说明** | **示例** |
| --- | --- | --- |
| xpath(String xpath) | 使用XPath选择 | html.xpath("//div[@class='title']") |
| $(String selector) | 使用Css选择器选择 | html.$("div.title") |
| $(String selector,String attr) | 使用Css选择器选择 | html.$("div.title","text") |
| css(String selector) | 功能同$()，使用Css选择器选择 | html.css("div.title") |
| links() | 选择所有链接 | html.links() |
| regex(String regex) | 使用正则表达式抽取 | html.regex("\(.\\*?)\") |

这部分抽取API返回的都是一个Selectable接口，意思是说，是支持链式调用的。例如访问<https://www.jd.com/moreSubject.aspx>页面



//先获取class为news\_div的div

//再获取里面的所有包含文明的元素  
List<String> list = page.getHtml()  
 .css("div#news\_div")  
 .regex(".\*文明.\*").all();

### 获取结果API

当链式调用结束时，我们一般都想要拿到一个字符串类型的结果。这时候就需要用到获取结果的API了。

我们知道，一条抽取规则，无论是XPath、CSS选择器或者正则表达式，总有可能抽取到多条元素。WebMagic对这些进行了统一，可以通过不同的API获取到一个或者多个元素。

| **方法** | **说明** | **示例** |
| --- | --- | --- |
| get() | 返回一条String类型的结果 | String link= html.links().get() |
| toString() | 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() |

当有多条数据的时候，使用get()和toString()都是获取第一个url地址。

String str = page.getHtml()  
 .css("div#news\_div")  
 .links().regex(".\*[0-3]$").toString();  
  
String get = page.getHtml()  
 .css("div#news\_div")  
 .links().regex(".\*[0-3]$").get();

测试结果：



这里selectable.toString()采用了toString()这个接口，是为了在输出以及和一些框架结合的时候，更加方便。因为一般情况下，我们都只需要选择一个元素！

selectable.all()则会获取到所有元素。

### 获取链接

有了处理页面的逻辑，我们的爬虫就接近完工了，但是现在还有一个问题：一个站点的页面是很多的，一开始我们不可能全部列举出来，于是如何发现后续的链接，是一个爬虫不可缺少的一部分。

下面的例子就是获取<https://www.jd.com/moreSubject.aspx>这个页面中

所有符合[https://www.jd.com/news.\\w+?.\*](https://www.jd.com/news.\\\\w+?.*)正则表达式的url地址

并将这些链接加入到待抓取的队列中去。

public void process(Page page) {  
 page.addTargetRequests(page.getHtml().links()  
 .regex("(https://www.jd.com/news.\\w+?.\*)").all());  
 System.*out*.println(page.getHtml().css("div.mt>h1").all());  
}  
  
public static void main(String[] args) {  
 Spider.*create*(new JobProcessor())  
 .addUrl("https://www.jd.com/moreSubject.aspx")  
 .run();  
}

## 使用Pipeline保存结果

WebMagic用于保存结果的组件叫做Pipeline。我们现在通过“控制台输出结果”这件事也是通过一个内置的Pipeline完成的，它叫做ConsolePipeline。

那么，我现在想要把结果用保存到文件中，怎么做呢？只将Pipeline的实现换成"FilePipeline"就可以了。

public static void main(String[] args) {  
 Spider.*create*(new JobProcessor())  
 //初始访问url地址  
 .addUrl("https://www.jd.com/moreSubject.aspx")  
 .addPipeline(new FilePipeline("D:/webmagic/"))  
 .thread(5)//设置线程数  
 .run();  
}

## 爬虫的配置、启动和终止

### Spider

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** | **示例** |
| --- | --- | --- |
| create(PageProcessor) | 创建Spider | Spider.create(new GithubRepoProcessor()) |
| addUrl(String…) | 添加初始的URL | spider .addUrl("[http://webmagic.io/docs/](http://webmagic.io/docs/" \t "_blank)") |
| thread(n) | 开启n个线程 | spider.thread(5) |
| run() | 启动，会阻塞当前线程执行 | spider.run() |
| start()/runAsync() | 异步启动，当前线程继续执行 | spider.start() |
| stop() | 停止爬虫 | spider.stop() |
| addPipeline(Pipeline) | 添加一个Pipeline，一个Spider可以有多个Pipeline | spider .addPipeline(new ConsolePipeline()) |
| setScheduler(Scheduler) | 设置Scheduler，一个Spider只能有个一个Scheduler | spider.setScheduler(new RedisScheduler()) |
| setDownloader(Downloader) | 设置Downloader，一个Spider只能有个一个Downloader | spider .setDownloader(  new SeleniumDownloader()) |
| get(String) | 同步调用，并直接取得结果 | ResultItems result = spider  .get("[http://webmagic.io/docs/](http://webmagic.io/docs/" \t "_blank)") |
| getAll(String…) | 同步调用，并直接取得一堆结果 | List<ResultItems> results = spider  .getAll("[http://webmagic.io/docs/](http://webmagic.io/docs/" \t "_blank)", "[http://webmagic.io/xxx](http://webmagic.io/xxx" \t "_blank)") |

### 爬虫配置Site

Site.me()可以对爬虫进行一些配置配置，包括编码、抓取间隔、超时时间、重试次数等。在这里我们先简单设置一下：重试次数为3次，抓取间隔为一秒。

private Site site = Site.*me*()  
 .setCharset("UTF-8")//编码  
 .setSleepTime(1)//抓取间隔时间  
 .setTimeOut(1000\*10)//超时时间  
 .setRetrySleepTime(3000)//重试时间  
 .setRetryTimes(3);//重试次数

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** | **示例** |
| --- | --- | --- |
| setCharset(String) | 设置编码 | site.setCharset("utf-8") |
| setUserAgent(String) | 设置UserAgent | site.setUserAgent("Spider") |
| setTimeOut(int) | 设置超时时间，  单位是毫秒 | site.setTimeOut(3000) |
| setRetryTimes(int) | 设置重试次数 | site.setRetryTimes(3) |
| setCycleRetryTimes(int) | 设置循环重试次数 | site.setCycleRetryTimes(3) |
| addCookie(String,String) | 添加一条cookie | site.addCookie("dotcomt\_user","code4craft") |
| setDomain(String) | 设置域名，需设置域名后，addCookie才可生效 | site.setDomain("github.com") |
| addHeader(String,String) | 添加一条addHeader | site.addHeader("Referer","[https://github.com](https://github.com/" \t "_blank)") |
| setHttpProxy(HttpHost) | 设置Http代理 | site.setHttpProxy(new HttpHost("127.0.0.1",8080)) |