webmagic使用手册

> webmagic是一个开源的Java垂直爬虫框架，目标是简化爬虫的开发流程，让开发者专注于逻辑功能的开发。webmagic的核心非常简单，但是覆盖爬虫的整个流程，也是很好的学习爬虫开发的材料。作者曾经进行过一年的垂直爬虫的开发，webmagic就是为了解决爬虫开发的一些重复劳动而产生的框架。
>
> web爬虫是一种技术，webmagic致力于将这种技术的实现成本降低，但是出于对资源提供者的尊重，webmagic不会做反封锁的事情，包括：验证码破解、代理切换、自动登录、抓取静态资源等。
>
> webmagic的架构和设计参考了以下两个项目，感谢以下两个项目的作者:
>
> python爬虫 **scrapy** https://github.com/scrapy/scrapy
>
> Java爬虫 **Spiderman** https://gitcafe.com/laiweiwei/Spiderman
>
> webmagic遵循Apache 2.0协议，你可以自由进行使用和修改。有使用不便或者问题，欢迎在github提交issue，或者在oschina讨论模块提问。

快速开始

使用maven

webmagic使用maven管理依赖，你可以直接下载webmagic源码进行编译：

```
git clone https://github.com/code4craft/webmagic.git
mvn clean install
```

安装后，在项目中添加对应的依赖即可使用webmagic：

```
    <dependency>
        <groupId>us.codecraft</groupId>
        <artifactId>webmagic-core</artifactId>
        <version>0.2.0</version>
    </dependency>
    <dependency>
        <groupId>us.codecraft</groupId>
        <artifactId>webmagic-extension</artifactId>
        <version>0.2.0</version>
    </dependency>
```

项目结构

webmagic主要包括两个包：

- **webmagic-core**

  webmagic核心部分，只包含爬虫基本模块和基本抽取器。webmagic-core的目标是成为网页爬虫的一个教科书般的实现。

- **webmagic-extension**

  webmagic的扩展模块，提供一些更方便的编写爬虫的工具。包括注解格式定义爬虫、JSON、分布式等支持。

webmagic还包含两个可用的扩展包，因为这两个包都依赖了比较重量级的工具，所以从主要包中抽离出来：

- **webmagic-saxon**

  webmagic与Saxon结合的模块。Saxon是一个XPath、XSLT的解析工具，webmagic依赖Saxon来进行XPath2.0语法解析支持。

- **webmagic-selenium**

  webmagic与Selenium结合的模块。Selenium是一个模拟浏览器进行页面渲染的工具，webmagic依赖Selenium进行动态页面的抓取。

在项目中，你可以根据需要依赖不同的包。

不使用maven

不使用maven的用户，可以下载这个二进制打包版本(感谢oschina)：

```
git clone http://git.oschina.net/flashsword20/webmagic-bin.git
```

在**bin/lib**目录下，有项目依赖的所有jar包，直接在IDE里import即可。

第一个爬虫

定制PageProcessor

PageProcessor是webmagic-core的一部分，定制一个PageProcessor即可实现自己的爬虫逻辑。以下是抓取osc博客的一段代码：

```
public class OschinaBlogPageProcesser implements PageProces
sor {

    private Site site = Site.me().setDomain("my.oschina.net
")
       .addStartUrl("http://my.oschina.net/flashsword/blog"
);

    @Override
    public void process(Page page) {
        List<String> links = page.getHtml().links().regex("
http://my\\.oschina\\.net/flashsword/blog/\\d+").all();
        page.addTargetRequests(links);
        page.putField("title", page.getHtml().xpath("//div[
@class='BlogEntity']/div[@class='BlogTitle']/h1").toString(
));
        page.putField("content", page.getHtml().$("div.cont
ent").toString());
        page.putField("tags",page.getHtml().xpath("//div[@c
lass='BlogTags']/a/text()").all());
    }

    @Override
    public Site getSite() {
        return site;

    }

    public static void main(String[] args) {
        Spider.create(new OschinaBlogPageProcesser())
             .pipeline(new ConsolePipeline()).run();
    }
}
```

这里通过page.addTargetRequests()方法来增加要抓取的URL，并通过page.putField()来保存抽取结果。page.getHtml().xpath()则是按照某个规则对结果进行抽取，这里抽取支持链式调用。调用结束后，toString()表示转化为单个String，all()则转化为一个String列表。

Spider是爬虫的入口类。Pipeline是结果输出和持久化的接口，这里ConsolePipeline表示结果输出到控制台。

执行这个main方法，即可在控制台看到抓取结果。webmagic默认有3秒抓取间隔，请耐心等待。

使用注解

webmagic-extension包括了注解方式编写爬虫的方法，只需基于一个POJO增加注解即可完成一个爬虫。以下仍然是抓取oschina博客的一段代码，功能与OschinaBlogPageProcesser完全相同：

```
@TargetUrl("http://my.oschina.net/flashsword/blog/\\d+")
public class OschinaBlog {

    @ExtractBy("//title")
    private String title;

    @ExtractBy(value = "div.BlogContent",type = ExtractBy.T
ype.Css)
    private String content;

    @ExtractBy(value = "//div[@class='BlogTags']/a/text()",
 multi = true)
    private List<String> tags;

    public static void main(String[] args) {
        OOSpider.create(
            Site.me().addStartUrl("http://my.oschina.net/fl
ashsword/blog"),
            new ConsolePageModelPipeline(), OschinaBlog.cla
ss).run();
    }
}
```

这个例子定义了一个Model类，Model类的字段'title'、'content'、'tags'均为要抽取的属性。这个类在Pipeline里是可以复用的。

注解的详细使用方式见后文中得webmagic-extension注解模块。

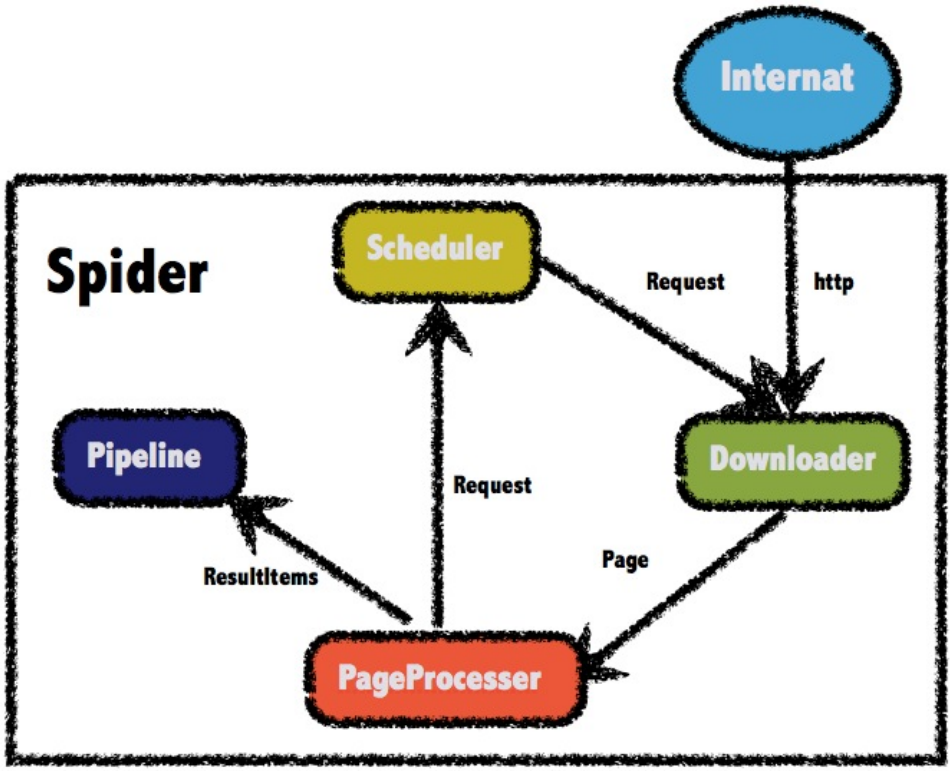
webmagic-core

webmagic-core是爬虫的核心框架，只包括一个爬虫各功能模块的核心功能。webmagic-core的目标是成为网页爬虫的一个教科书般的实现。

此节部分内容摘自作者的博文 webmagic的设计机制及原理-如何开发一个Java爬虫。

webmagic-core的模块划分

webmagic-core参考了scrapy的模块划分，分为Spider(整个爬虫的调度框架)、Downloader(页面下载)、PageProcessor(链接提取和页面分析)、Scheduler(URL管理)、Pipeline(离线分析和持久化)几部分。只不过scrapy通过middleware实现扩展，而webmagic则通过定义这几个接口，并将其不同的实现注入主框架类Spider来实现扩展。



Spider类(核心调度)

Spider是爬虫的入口类，Spider的接口调用采用了链式的API设计，其他功能全部通过接口注入Spider实现，下面是启动一个比较复杂的Spider的例子。

```
Spider.create(sinaBlogProcessor)
.scheduler(new FileCacheQueueScheduler("/data/temp/webmagic
/cache/"))
.pipeline(new FilePipeline())
.thread(10).run();
```

Spider的核心处理流程非常简单，代码如下：

```
<!-- lang: java -->
private void processRequest(Request request) {
    Page page = downloader.download(request, this);
    if (page == null) {
        sleep(site.getSleepTime());
        return;
    }
    pageProcessor.process(page);
    addRequest(page);
    for (Pipeline pipeline : pipelines) {
        pipeline.process(page, this);
    }
    sleep(site.getSleepTime());
}
```

PageProcessor(页面分析及链接抽取)

页面分析是垂直爬虫中需要定制的部分。在webmagic-core里，通过实现**PageProcessor**接口来实现定制爬虫。PageProcessor有两个核心方法：public void process(Page page)和public Site getSite() 。

- public void process(Page page)

  通过对**Page**对象的操作，实现爬虫逻辑。Page对象包括两个最重要的方法：addTargetRequests()可以添加URL到待抓取队列，put()可以将结果保存供后续处理。 Page的数据可以通过Page.getHtml()和Page.getUrl()获取。

- public Site getSite()

  Site对象定义了爬虫的域名、起始地址、抓取间隔、编码等信息。

Selector是webmagic为了简化页面抽取开发的独立模块，是webmagic-core的主要着力点。这里整合了CSS Selector、XPath和正则表达式，并可以进行链式的抽取。

```
<!-- lang: java -->
//content是用别的爬虫工具抽取到的正文
List<String> links = page.getHtml()
.$("div.title")  //css 选择，Java里虽然很少有$符号出现，不过貌似$作为方法名是合法的
.xpath("//@href")  //提取链接
.regex(".*blog.*") //正则匹配过滤
.all(); //转换为string列表
```

webmagic包括一个对于页面正文的自动抽取的类**SmartContentSelector**。相信用过Evernote Clearly都会对其自动抽取正文的技术印象深刻。这个技术又叫**Readability**。当然webmagic对Readability的实现还比较粗略，但是仍有一些学习价值。

基于Saxon，webmagic提供了XPath2.0语法的支持。XPath2.0语法支持内部函数、逻辑控制等，是一门完整的语言，如果你熟悉XPath2.0语法，倒是不妨一试(需要引入**webmagic-saxon**包)。

webmagic-samples包里有一些为某个站点定制的PageProcessor，供学习之用。

Downloader(页面下载)

Downloader是webmagic中下载页面的接口，主要方法：

- public Page download(Request request, Task task)

  Request对象封装了待抓取的URL及其他信息，而Page则包含了页面下载后的Html及其他信息。Task是一个包装了任务对应的Site信息的抽象接口。

- public void setThread(int thread)

因为Downloader一般会涉及连接池等功能，而这些功能与多线程密切相关，所以定义了此方法。

目前有几个Downloader的实现：

- HttpClientDownloader

  集成了**Apache HttpClient**的Downloader。Apache HttpClient(4.0后整合到HttpCompenent项目中)是强大的Java http下载器，它支持自定义HTTP头(对于爬虫比较有用的就是User-agent、cookie等)、自动redirect、连接复用、cookie保留、设置代理等诸多强大的功能。

- SeleniumDownloader

  对于一些Javascript动态加载的网页，仅仅使用http模拟下载工具，并不能取到页面的内容。这方面的思路有两种：一种是抽丝剥茧，分析js的逻辑，再用爬虫去重现它；另一种就是：内置一个浏览器，直接获取最后加载完的页面。**webmagic-selenium**包中整合了Selenium到SeleniumDownloader，可以直接进行动态加载页面的抓取。使用selenium需要安装一些native的工具，具体步骤可以参考作者的博文使用Selenium来抓取动态加载的页面

Scheduler(URL管理)

Scheduler是webmagic的管理模块，通过实现Scheduler可以定制自己的URL管理器。Scheduler包括两个主要方法：

- public void push(Request request,Task task)

  将待抓取URL加入Scheduler。Request对象是对URL的一个封装，还包括优先级、以及一个供存储数据的Map。Task仍然用于区分不同任务，在多个任务公用一个Scheduler时可以此进行区分。

- public Request poll(Task task)

  从Scheduler里取出一条请求，并进行后续执行。

webmagic目前有三个Scheduler的实现：

- QueueScheduler

一个简单的内存队列，速度较快，并且是线程安全的。

- FileCacheQueueScheduler

  使用文件保存队列，它可以用于耗时较长的下载任务，在任务中途停止后(手动停止或者程序崩溃)，下次执行仍然从中止的URL开始继续爬取。

- RedisScheduler

  使用redis存储URL队列。通过使用同一台redis服务器存储URL，webmagic可以很容易的在多机部署，从而达到分布式爬虫的效果。

Pipeline(后续处理和持久化)

Pipeline是最终抽取结果进行输出和持久化的接口。它只包括一个方法：

- public void process(ResultItems resultItems,Task task)

  ResultItems是集成了抽取结果的对象。通过ResultItems.get(key)可以获取抽取结果。Task同样是用于区分不同任务的对象。

webmagic包括以下几个Pipeline的实现：

- ConsolePipeline

  直接输出结果到控制台，测试时使用。

- FilePipeline

  输出结果到文件，每个URL单独保存到一个页面，以URL的MD5结果作为文件名。通过构造函数 `public FilePipeline(String path)` 定义存储路径，**以下使用文件持久化的类，多数都使用此方法指定路径。**

- JsonFilePipeline

  以JSON输出结果到文件(.json后缀)，其他与FilePipeline相同。

webmagic目前不支持持久化到数据库，但是结合其他工具，持久化到数据库也是很容易的。这里不妨看一下webmagic结合JFinal持久化到数据库的一段代码。因为JFinal目前还不支持maven，所以这段代码并没有放到webmagic-samples里来。

webmagic-extension

webmagic-extension是为了开发爬虫更方便而实现的一些功能模块。这些功能完全基于webmagic-core的框架，包括注解形式编写爬虫、分页、分布式等功能。

注解模块

webmagic-extension包括注解模块。为什么会有注解方式?

因为PageProcessor的方式灵活、强大，但是没有解决两个问题:

- 对于一个站点，如果想抓取多种格式的URL，那么必须在PageProcesser中写判断逻辑，代码难以管理。
- 抓取结果没有对应Model，并不符合Java程序开发习惯，与一些框架也无法很好整合。

注解的核心是Model类，本身是一个POJO，这个Model类用于传递、保存页面最终抓取结果数据。注解方式直接将抽取与数据绑定，以便于编写和维护。

注解方式其实也是通过一个PageProcessor的实现--ModelPageProcessor完成，因此对webmagic-core代码没有任何影响。仍然以抓取OschinaBlog的程序为例:

```
@TargetUrl("http://my.oschina.net/flashsword/blog/\\d+")
public class OschinaBlog {

    @ExtractBy("//title")
    private String title;

    @ExtractBy(value = "div.BlogContent",type = ExtractBy.T
ype.Css)
    private String content;

    @ExtractBy(value = "//div[@class='BlogTags']/a/text()",
 multi = true)
    private List<String> tags;

    public static void main(String[] args) {
        OOSpider.create(
            Site.me().addStartUrl("http://my.oschina.net/fl
ashsword/blog"),
            new ConsolePageModelPipeline(), OschinaBlog.cla
ss).run();
    }
}
```

注解部分包括以下内容：

- **TargetUrl**

  "TargetUrl"表示这个Model对应要抓取的URL，它包含两层意思：符合这个条件的URL会被加入抓取队列；符合这个条件的URL会被这个Model抓取。TargetUrl可以**sourceRegion**指定提取URL的区域(仅支持XPath)。

  TargetUrl使用了正则表达式，匹配"http://my.oschina.net/flashsword/blog/150039" 格式的URL。webmagic对正则表达式进行了修改，"."仅表示字符"."而不代表任意字符，而"*"则代表了".*"，例如"http://*.oschina.net/*"代表了oschina所有的二级域名下的URL。

  与TargetUrl相似的还有**HelpUrl**，HelpUrl表示：仅仅抓取该URL用作链接提取，并不对它进行内容抽取。例如博客正文页对应TargetUrl，而列表页则对应HelpUrl。

- **ExtractBy**
  - **用于字段**

    "ExtractBy"可用于类以及字段。用于字段时，定义了字段抽取的规则。抽取的规则默认使用**XPath**，也可以选择使用CSS Selector、正则表达式(通过设置type)。

    ExtractBy还有几个扩展属性。**multi**表示是否抽取列表，当然，设置为multi时，你需要一个List字段去容纳它。**notnull**则表示，此字段不允许为null，若为null则放弃整个对象。

  - **用于类**

    "ExtractBy"用于类时，则限定了字段抽取的区域。用于类时仍支持multi，multi则表示一个页面可以抽取到多个对象。

  - **ExtractByRaw & ExtractByUrl**

    在类使用"ExtractBy"修饰后，字段的"ExtractBy"使用的是其抽取的结果，如果仍然想要抽取原HTML，可以使用"ExtractByRaw"。与此类似的还有"ExtractByUrl"，表示从URL中抽取信息。ExtractByUrl只支持正则表达式。

  - **ExtractBy2 ExtractBy3**

    "ExtractBy"、"ExtractByRaw"支持链式抽取，通过增加注解"ExtractBy2"、"ExtractBy3"实现。

- **AfterExtractor**

  AfterExtractor接口是对注解方式抽取能力不足的补充。实现AfterExtractor接口后，会在**使用注解方式填充完字段后**调用**afterProcess()**方法，在这个方法中可以直接访问已抽取的字段、补充需要抽取的字段，甚至做一些简单的输出和持久化操作(并不是很建议这么做)。这部分可以参考webmagic结合JFinal持久化到数据库的一段代码。

- **OOSpider**

  OOSpider是注解式爬虫的入口，这里调用**create()**方法将OschinaBlog这

个类加入到爬虫的抽取中，这里是可以传入多个类的，例如：

```
OOSpider.create(
    Site.me().addStartUrl("http://www.oschina.net"),
    new ConsolePageModelPipeline(),
    OschinaBlog.clas,OschinaAnswer.class).run();
```

OOSpider会根据TargetUrl调用不同的Model进行解析。

- **PageModelPipeline**

  可以通过定义PageModelPipeline来选择结果输出方式。这里new ConsolePageModelPipeline()是PageModelPipeline的一个实现，会将结果输出到控制台。

- **分页**

  处理单项数据分页(例如单条新闻多个页面)是爬虫一个比较头疼的问题。webmagic目前对于分页的解决方案是：在注解模式下，Model通过实现**PagedModel**接口，并引入PagedPipeline作为第一个Pipeline来实现。具体可以参考webmagic-samples中抓取网易新闻的代码：**us.codecraft.webmagic.model.samples.News163**。

  关于分页，这里有一篇对于webmagic分页实现的详细说明的文章关于爬虫实现分页的一些思考。 目前分页功能还没有分布式实现，如果使用RedisScheduler进行分布式爬取，请不要使用分页功能。

分布式

webmagic-extension中，通过redis来管理URL，达到分布式的效果。但是对于分布式爬虫，仅仅程序能够分布式运行，还满足不了大规模抓取的需要，webmagic可能后期会加入一些任务管理和监控的功能，也欢迎各位用户为webmagic提交代码，做出贡献。

更进一步

如果这篇文档仍然满足不了你的需要，你可以阅读webmagic的Javadoc，或者直接阅读源码。