**4.1 实现PageProcessor**

这部分我们直接通过GithubRepoPageProcessor这个例子来介绍PageProcessor的编写方式。我将PageProcessor的定制分为三个部分，分别是爬虫的配置、页面元素的抽取和链接的发现。

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()").toString());

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\\-]+)").all());

}

@Override

public Site getSite() {

return site;

}

public static void main(String[] args) {

Spider.create(new GithubRepoPageProcessor())

//从"https://github.com/code4craft"开始抓

.addUrl("https://github.com/code4craft")

//开启5个线程抓取

.thread(5)

//启动爬虫

.run();

}

}

**4.1.1 爬虫的配置**

第一部分关于爬虫的配置，包括编码、抓取间隔、超时时间、重试次数等，也包括一些模拟的参数，例如User Agent、cookie，以及代理的设置，我们会在第5章-“爬虫的配置”里进行介绍。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。

**4.1.2 页面元素的抽取**

第二部分是爬虫的核心部分：对于下载到的Html页面，你如何从中抽取到你想要的信息？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。

1. XPath

XPath本来是用于XML中获取元素的一种查询语言，但是用于Html也是比较方便的。例如：

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

这段代码使用了XPath，它的意思是“查找所有class属性为'entry-title public'的h1元素，并找到他的strong子节点的a子节点，并提取a节点的文本信息”。 对应的Html是这样子的：

1. CSS选择器

CSS选择器是与XPath类似的语言。如果大家做过前端开发，肯定知道$('h1.entry-title')这种写法的含义。客观的说，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。

1. 正则表达式

正则表达式则是一种通用的文本抽取语言。

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

这段代码就用到了正则表达式，它表示匹配所有"[https://github.com/code4craft/webmagic"这样的链接。](https://github.com/code4craft/webmagic%22%E8%BF%99%E6%A0%B7%E7%9A%84%E9%93%BE%E6%8E%A5%E3%80%82)

1. JsonPath

JsonPath是于XPath很类似的一个语言，它用于从Json中快速定位一条内容。WebMagic中使用的JsonPath格式可以参考这里：<https://code.google.com/p/json-path/>

**4.1.3 链接的发现**

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。

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

这段代码的分为两部分，page.getHtml().links().regex("(https://github\\.com/\\w+/\\w+)").all()用于获取所有满足"(https:/ /github\.com/\w+/\w+)"这个正则表达式的链接，page.addTargetRequests()则将这些链接加入到待抓取的队列中去。

### 4.2 使用Selectable抽取元素

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，你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。这个接口包含一些重要的方法，我将它分为两类：抽取部分和获取结果部分。

#### 4.2.1 抽取部分API：

| **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("\(.\\*?)\") |
| regex(String regex,int group) | 使用正则表达式抽取，并指定捕获组 | html.regex("\(.\\*?)\",1) |
| replace(String regex, String replacement) | 替换内容 | html.replace("\","") |

这部分抽取API返回的都是一个Selectable接口，意思是说，抽取是支持链式调用的。下面我用一个实例来讲解链式API的使用。

例如，我现在要抓取github上所有的Java项目，这些项目可以在<https://github.com/search?l=Java&p=1&q=stars%3A%3E1&s=stars&type=Repositories>搜索结果中看到。

为了避免抓取范围太宽，我指定只从分页部分抓取链接。这个抓取规则是比较复杂的，我会要怎么写呢？

首先看到页面的html结构是这个样子的：

那么我可以先用CSS选择器提取出这个div，然后在取到所有的链接。为了保险起见，我再使用正则表达式限定一下提取出的URL的格式，那么最终的写法是这样子的：

List<String> urls = page.getHtml().css("div.pagination").links().regex(".\*/search/\?l=java.\*").all();

然后，我们可以把这些URL加到抓取列表中去：

List<String> urls = page.getHtml().css("div.pagination").links().regex(".\*/search/\?l=java.\*").all();

page.addTargetRequests(urls);

是不是比较简单？除了发现链接，Selectable的链式抽取还可以完成很多工作。我们会在第9章示例中再讲到。

#### 4.2.2 获取结果的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，你可以通过不同的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() | 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; } |

例如，我们知道页面只会有一条结果，那么可以使用selectable.get()或者selectable.toString()拿到这条结果。

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。

好了，到现在为止，在回过头看看3.1中的GithubRepoPageProcessor，可能就觉得更加清晰了吧？指定main方法，已经可以看到抓取结果在控制台输出了。

### 4.3 使用Pipeline保存结果

好了，爬虫编写完成，现在我们可能还有一个问题：我如果想把抓取的结果保存下来，要怎么做呢？WebMagic用于保存结果的组件叫做Pipeline。例如我们通过“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Json的格式保存下来，怎么做呢？我只需要将Pipeline的实现换成"JsonFilePipeline"就可以了。

public static void main(String[] args) {

Spider.create(new GithubRepoPageProcessor())

//从"https://github.com/code4craft"开始抓

.addUrl("https://github.com/code4craft")

.addPipeline(new JsonFilePipeline("D:\\webmagic\\"))

//开启5个线程抓取

.thread(5)

//启动爬虫

.run();

}

这样子下载下来的文件就会保存在D盘的webmagic目录中了。

通过定制Pipeline，我们还可以实现保存结果到文件、数据库等一系列功能。这个会在第7章“抽取结果的处理”中介绍。

至此为止，我们已经完成了一个基本爬虫的编写，也具有了一些定制功能。