Table of Contents

WebMagic in Action

> Little book of WebMagic.

WebMagic is a simple but scalable crawler framework. You can develop a crawler easily based on it.

This little book will describe how to use WebMagic. It will also show some classic cases of how to develop a crawler for specific site.

This document is being writing currently. It is translated from my Chinese docs http://webmagic.io/docs/zh/, and there may be some mistake. Thanks for Reporting Issue or fork and send a pull request to me.

Docs preview: http://webmagic.io/docs/en/(construct based on gitbook).

License: CC-BYNC.

1. Overview of WebMagic

WebMagic is a simple crawler for Java developer.

WebMagic contains two parts: core and extension. Webmagic-core is a simple and well modulized implemetation of crawler, and webmagic-extension supply some convenient features for crawler depleloping.

The architect of WebMagic-core is refer to Scrapy. It supply simple but flexible API. You can write a crawler just if you are familiar to Java.

Webmagic-extension supply some convenient features, such as writing a crawler only with a POJO and some annotation. There are also some default implementation of the components.

Webmagic also contains some other extensions and an complete product "WebMagic-Avalon".

1.1 Philosophy

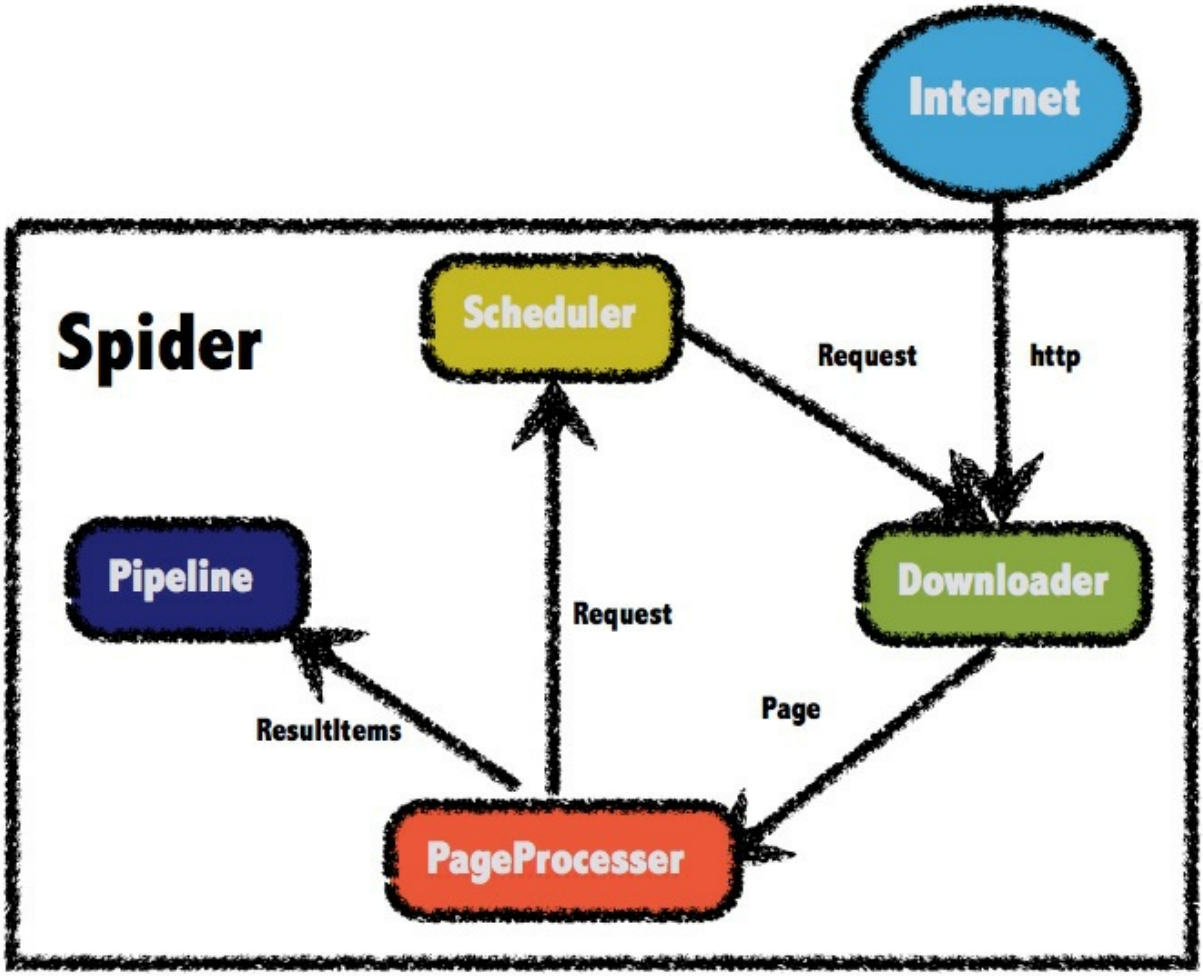
1. Solution for Domain
2. Mirco core with high flexibility
3. Convenient to Use

1.2 overall architecture

WebMagic structured into `Downloader` , `PageProcessor` , `Scheduler` , `Pipeline` four components by Spider will organize them with each other. This component corresponds to the four crawler lifecycle download, processing, management, and persistence capabilities. WebMagic design reference Scapy, but the implementation of some of the more Java.

The Spider will be several components to organize themselves so that they can interact with each other, the process of implementation can be considered Spider is a large container, it is also the core WebMagic logic.

WebMagic overall architecture is as follows:



1.2.1 The four components WebMagic

1.Downloader

Downloader responsible for downloading from the Internet page, for subsequent

processing. WebMagic default of Apache HttpClient as a download tool.

2.PageProcessor

PageProcessor responsible for parsing the page, extract useful information, as well as the discovery of new links. WebMagic use Jsoup as HTML parsing tools, based on its development of an analytical tool XPath Xsoup.

In these four components, `PageProcessor` not the same for each page of each site, the user is required to customize parts.

3.Scheduler

URL Scheduler manages to be crawled, as well as some heavy to work. WebMagic provided by default JDK memory queue management URL, and set to go with the weight. Redis also supports the use of distributed management.

Unless the project has distributed some special needs, you do not need to customize their own Scheduler.

4.Pipeline

Processing Pipeline responsible for taking the results, including calculations, persisted to files, databases and so on. WebMagic provided by default "to the console" and "Save to File" two results treatment program.

`Pipeline` defines the way results are saved, if you want to save to the specified database, you need to write the corresponding Pipeline. For a class generally only needs to write a `Pipeline` .

1.2.2 for data transfer objects

1. Request

`Request` is a URL address layer package, a Request corresponding to a URL address.

It is the carrier PageProcessor interact with Downloader, Downloader is the only way to PageProcessor control.

In addition to the URL itself, it contains a Key-Value Structure field `extra` . You can save some extra special attributes, and then read in other places to perform different functions. For example, some additional information on one page and so on.

2. Page

`Page` representatives from Downloader to download a page - may be HTML, it may be the content of JSON, or other text formats.

Page WebMagic extraction process is the core of the object, which provides methods for extraction, save the results and so on. In the case of the fourth chapter, we will detail its use.

3. ResultItems

`ResultItems` equivalent to a Map, which holds the result PageProcessor processing for use Pipeline. Map and its API is very similar, it is worth noting that it has a field `skip`, if set to true, the Pipeline should not be processed.

1.2.3 Control crawler running engine --Spider

Spider is the core WebMagic internal processes. A property Downloader, PageProcessor, Scheduler, Pipeline is the Spider, these properties can be freely set by setting this property can perform different functions. Spider WebMagic also operate the entrance, which encapsulates the creation of crawlers, start, stop, multi-threading capabilities. Here is a set of each component, and set an example of multi-threading and startup. See detailed Spider setting Chapter 4 - crawler configuration, start and stop.

```
public static void main(String[] args) {
    Spider.create(new GithubRepoPageProcessor())
            // From https://github.com/code4craft began to grasp
            .addUrl("https://github.com/code4craft")
            // Set the Scheduler, use Redis to manage URL queue
            .setScheduler(new RedisScheduler("localhost"))
            // Set Pipeline, will result in json way to save a file
            .addPipeline(new JsonFilePipeline("D:\\data\\webmagic"))
            //Open 5 simultaneous execution threads
            .thread(5)
            //Start crawler
            .run();
}
```

1.2.4 Quick Start

A lot of the components described above, but in fact the user need to be concerned not so much, because most of the module WebMagic already provides a default implementation.

In general, for the preparation of a crawler, `PageProcessor` is part of the need to write, and `Spider` is created and controlled entrance crawlers. In the fourth chapter, we will explain how to write a crawler customized PageProcessor, and by Spider to start.

1.3 Project Components

WebMagic project code consists of several parts, in the root directory to a different directory name separately. They are independent of the Maven project.

1.3.1 The main part

WebMagic includes two packages, both packages through extensive practical, more mature:

Webmagic-core

`Webmagic-core` is WebMagic core part, only contains the basic modules and basic crawler extractor. WebMagic-core goal is to become a textbook pages crawler-like implementation.

Webmagic-extension

`Webmagic-extension` is WebMagic major expansion module that provides some of the more convenient tool written in crawlers. Including annotation format definition crawlers, JSON, distributed and other support.

1.3.2 Peripheral functions

In addition, WebMagic projects in several packages, these are some experimental features, and the purpose is to provide some tools to integrate peripheral sample. Because of the limited energy, these packages have not been widely used and tested, recommended way is to download the source code, then modify encounter problems.

Webmagic-samples

Here are some examples of crawlers author written earlier. Because of the limited time, some of these examples use is still the old version of the API, but also because there may be some changes in the structure of the target page is no longer available. To date, been featured examples, see the `us.codecraft.webmagic.processor.example` webmagic-core package and the `webmaigc-core package of us.codecraft.webmagic.example` .

Webmagic-scripts

WebMagic for crawlers rule scripted some attempts, the goal is to allow developers

from the Java language, for simple, rapid development. While emphasizing the shared script.

Currently the project because the user is not much interested in, on hold, you can look for scripted interest here: webmagic-scripts simple document

Webmagic-selenium

WebMagic and Selenium combined modules. Selenium is an analog browser page rendering tools, WebMagic rely Selenium crawl dynamic pages.

Webmagic-saxon

WebMagic and Saxon binding module. Saxon is a XPath, XSLT analytical tools, webmagic rely Saxon to XPath2.0 parsing support.

1.3.3 webmagic-avalon

`Webmagic-avalon` is a special project, it wants to achieve a product based on WebMagic of tools that covers the creation of crawlers, crawlers and other backend management tools. Avalon Arthurian legend is the "ideal island", `webmagic-avalon` the goal is to provide a common crawler products achieve this goal is not easy, so the name is also a little" ideal "means, but the author has been striving towards this goal.

You can look interested in this project here [WebMagic-Avalon project] (https://github.com/code4craft/webmagic/issues/43).

2. Use WebMagic

WebMagic's core feature requires two jars： `webmagic-core-{version}.jar` and `webmagic-extension-{version}.jar` . Just add them to your classpath and you are good to go.

WebMagic uses Maven to manage its dependencies, but of cause you can use it without Maven.

2.1 Use Maven to manage your dependency

WebMagic is built on Maven, so it is highly recommanded to use Maven to manage your project. You can add WebMagic by appending following lines to your `pom.xml` :

```
<dependency>
    <groupId>us.codecraft</groupId>
    <artifactId>webmagic-core</artifactId>
    <version>0.6.0</version>
</dependency>
<dependency>
    <groupId>us.codecraft</groupId>
    <artifactId>webmagic-extension</artifactId>
    <version>0.6.0</version>
</dependency>
```

If you are not familiar with Maven, Here is an introduction: What is Maven?

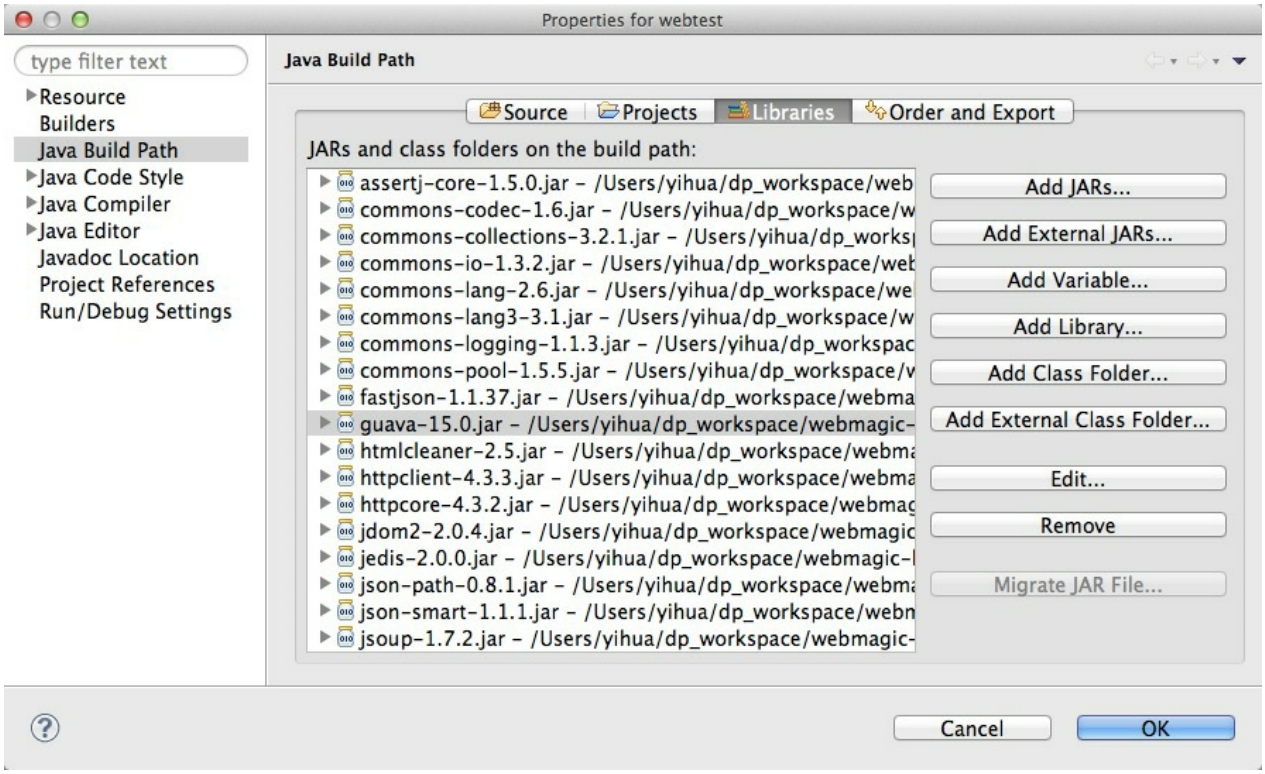
WebMagic uses `slf4j-log4j12` as the implemetation of slf4j. If you have your own choice of slf4j implemetation, exclude the former from your dependency.

```
<dependency>
    <groupId>us.codecraft</groupId>
    <artifactId>webmagic-extension</artifactId>
    <version>0.6.0</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

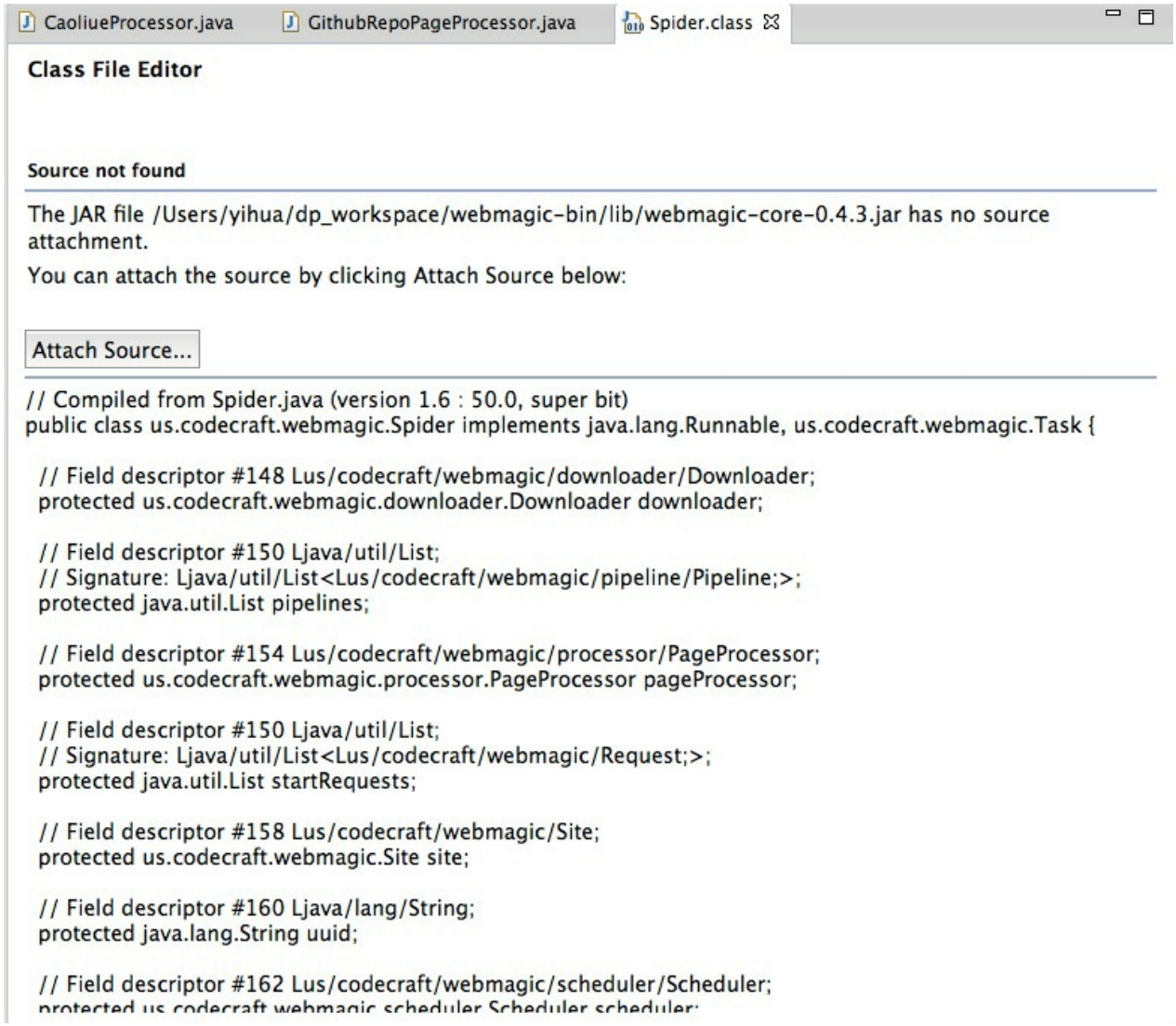
2.2 Without Maven

If you don't know Maven or don't like it, you can download the newest jars from http://webmagic.io. I also wrapped all the jars that webmagic requires, click here to download.

After the download, just unzip the archive and add all jars to your classpath.



Because WebMagic encourages the customization, it is really helpful to see the source code. You can get the newest `webmagic-core-{version}-sources.jar` and `webmagic-extension-{version}-sources.jar` on http://webmagic.io, just click on "Attch Source".

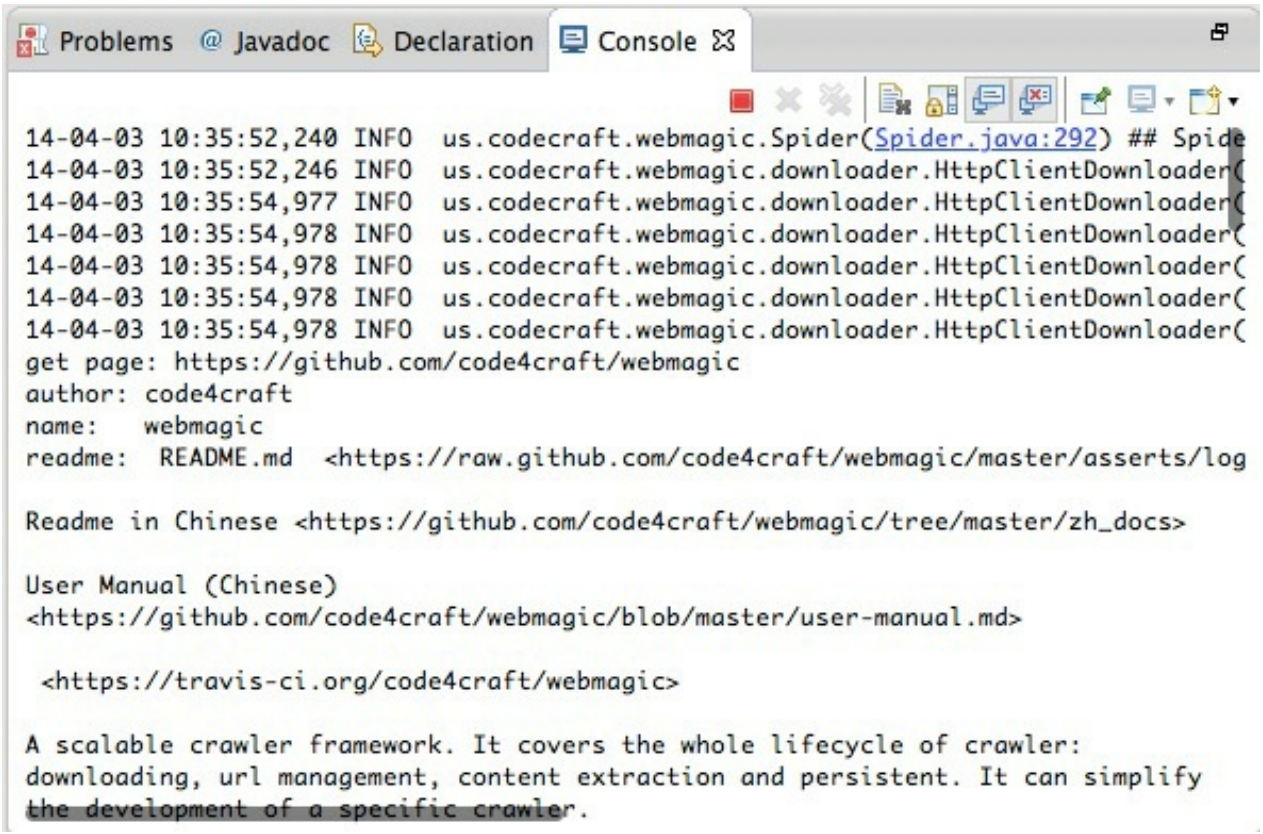

CaoliueProcessor.java
GithubRepoPageProcessor.java
Spider.class
Class File Editor
Source not found
The JAR file /Users/yihua/dp_workspace/webmagic-bin/lib/webmagic-core-0.4.3.jar has no source attachment.
You can attach the source by clicking Attach Source below:
Attach Source...
// Compiled from Spider.java (version 1.6 : 50.0, super bit)
public class us.codecraft.webmagic.Spider implements java.lang.Runnable, us.codecraft.webmagic.Task {
// Field descriptor #148 Lus/codecraft/webmagic/downloader/Downloader;
protected us.codecraft.webmagic.downloader.Downloader downloader;
// Field descriptor #150 Ljava/util/List;
// Signature: Ljava/util/List<Lus/codecraft/webmagic/pipeline/Pipeline;>;
protected java.util.List pipelines;
// Field descriptor #154 Lus/codecraft/webmagic/processor/PageProcessor;
protected us.codecraft.webmagic.processor.PageProcessor pageProcessor;
// Field descriptor #150 Ljava/util/List;
// Signature: Ljava/util/List<Lus/codecraft/webmagic/Request;>;
protected java.util.List startRequests;
// Field descriptor #158 Lus/codecraft/webmagic/Site;
protected us.codecraft.webmagic.Site site;
// Field descriptor #160 Ljava/lang/String;
protected java.lang.String uuid;
// Field descriptor #162 Lus/codecraft/webmagic/scheduler/Scheduler;

2.3 First project

After adding maven dependencies for your project, we are ready to write our very first crawler. Here we take Github for example:

```
import us.codecraft.webmagic.Page;
import us.codecraft.webmagic.Site;
import us.codecraft.webmagic.Spider;
import us.codecraft.webmagic.processor.PageProcessor;

public class GithubRepoPageProcessor implements PageProcessor {

    private Site site = Site.me().setRetryTimes(3).setSleepTime(100);

    @Override
    public void process(Page page) {
        page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/\\w+/\
\w+)").all());
        page.putField("author", page.getUrl().regex("https://github\\.com/(\\w+)/.*").toS
tring());
        page.putField("name", page.getHtml().xpath("//h1[@class='entry-title public']/str
ong/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()"));
    }

    @Override
    public Site getSite() {
        return site;
    }

    public static void main(String[] args) {
        Spider.create(new GithubRepoPageProcessor()).addUrl("https://github.com/code4craf
t").thread(5).run();
    }
}
```

Now run the main()... And voilà.


Problems
Javadoc
Declaration
Console
14-04-03 10:35:52,240 INFO us.codecraft.webmagic.Spider(Spider.java:292) ## Spide
14-04-03 10:35:52,246 INFO us.codecraft.webmagic.downloader.HttpClientDownloader(
14-04-03 10:35:54,977 INFO us.codecraft.webmagic.downloader.HttpClientDownloader(
14-04-03 10:35:54,978 INFO us.codecraft.webmagic.downloader.HttpClientDownloader(
14-04-03 10:35:54,978 INFO us.codecraft.webmagic.downloader.HttpClientDownloader(
14-04-03 10:35:54,978 INFO us.codecraft.webmagic.downloader.HttpClientDownloader(
14-04-03 10:35:54,978 INFO us.codecraft.webmagic.downloader.HttpClientDownloader(
get page: https://github.com/code4craft/webmagic
author: code4craft
name: webmagic
readme: README.md <https://raw.github.com/code4craft/webmagic/master/asserts/log
Readme in Chinese <https://github.com/code4craft/webmagic/tree/master/zh_docs>
User Manual (Chinese)
<https://github.com/code4craft/webmagic/blob/master/user-manual.md>
<https://travis-ci.org/code4craft/webmagic>
A scalable crawler framework. It covers the whole lifecycle of crawler:
downloading, url management, content extraction and persistent. It can simplify
the development of a specific crawler.

3. Download and compile the source code

If you are interested in the source WebMagic, then you can choose to download the source code and compile way to use WebMagic. "Very simple secondary development" is also one of the goals of WebMagic.

WebMagic is a pure Java project, if you are familiar with Maven, then download and compile the source code is very simple. If you are not familiar with Maven does not matter, this chapter describes how to import this project in the Eclipse.

3.1 Get the Source Code

WebMagic currently has two repositories:

*https://github.com/code4craft/webmagic

Github repository on save the latest version of all the issue, pull request all here. We feel that the project is good, then do not forget to give a star!

*http://git.oschina.net/flashsword20/webmagic

This repository contains all compiled dependencies, save only stable version of the project, the latest version is still on github update. Oschina relatively stable in the country, mainly as a mirror.

No matter in which warehouse use

git clone https://github.com/code4craft/webmagic.git

or

git clone http://git.oschina.net/flashsword20/webmagic.git

You can download the latest code.

If you are not familiar with using git itself, we recommend a look @ Huang Yong Download Smart Source from Git OSC

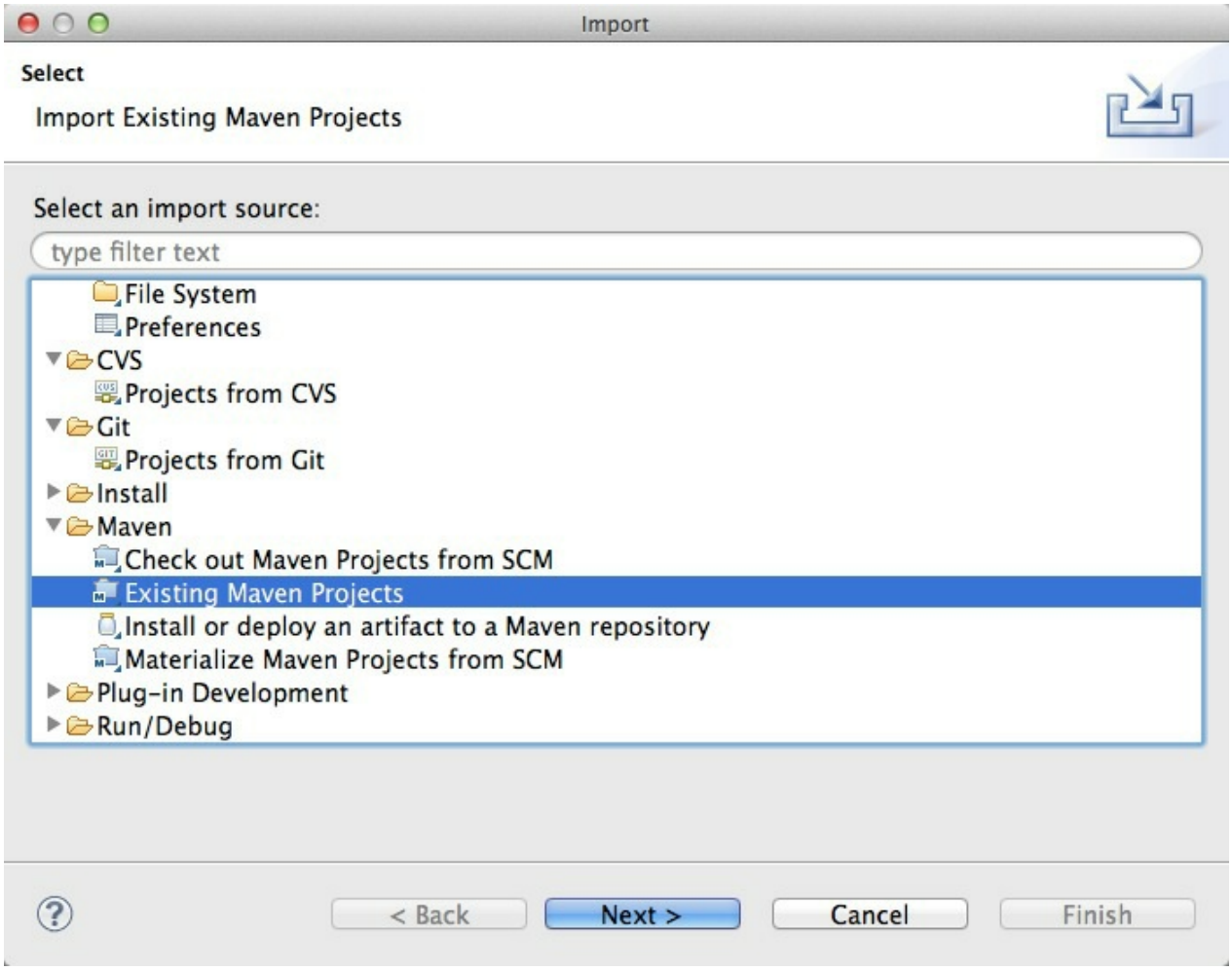
3.2 Import Project

Intellij Idea default comes with Maven support, select items when you can import Maven projects.

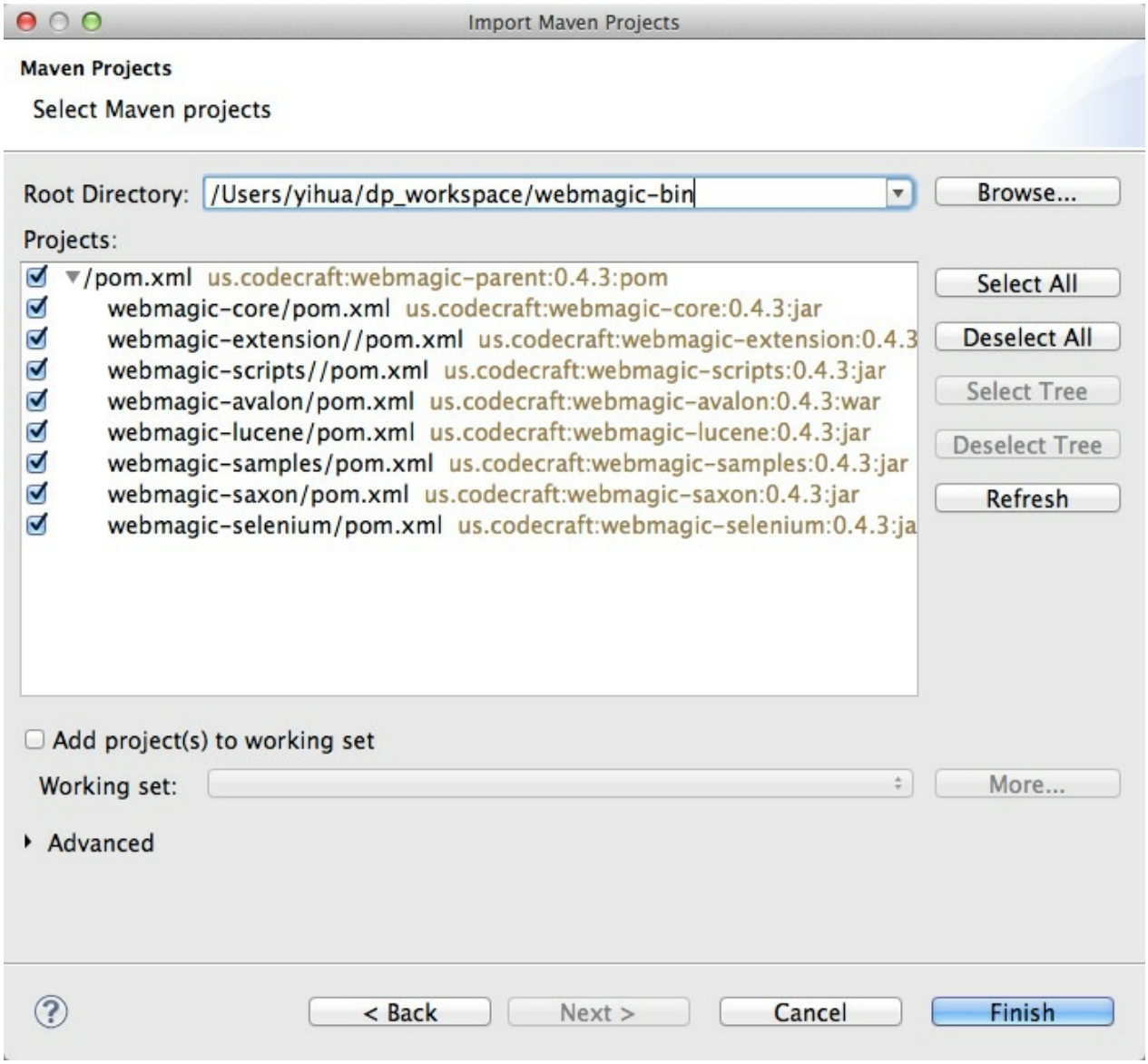
3.2.1 m2e plug-in

Eclipse user, it is recommended to install m2e plug-in installation address: https://www.eclipse.org/m2e/download/

After installation, select File->Import... and select Maven-> Existing Maven Projects and click Next, can be imported folder into the project.



After importing the project to see the selection screen, you can click finish.

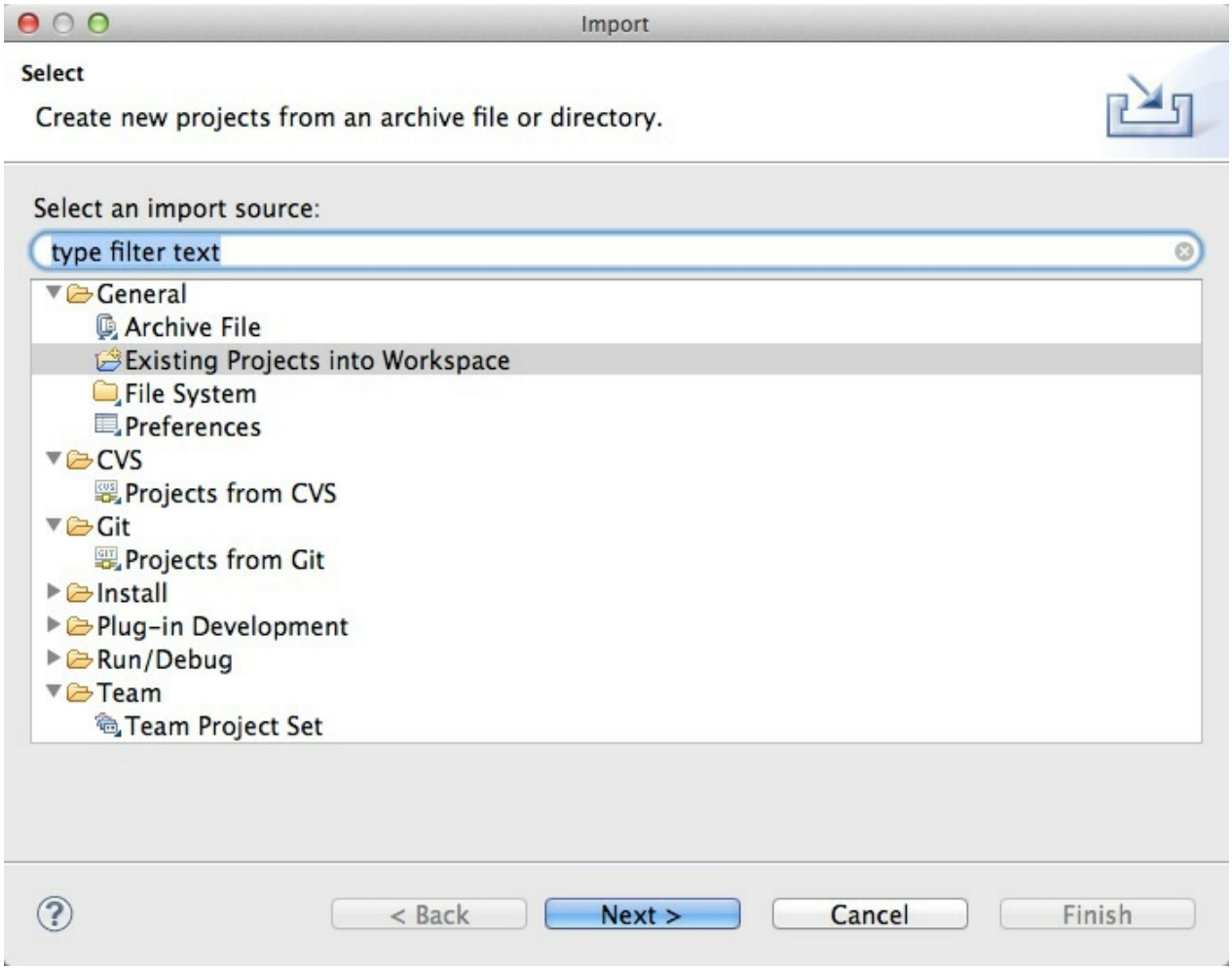


3.2.2 Using Maven Eclipse plugin

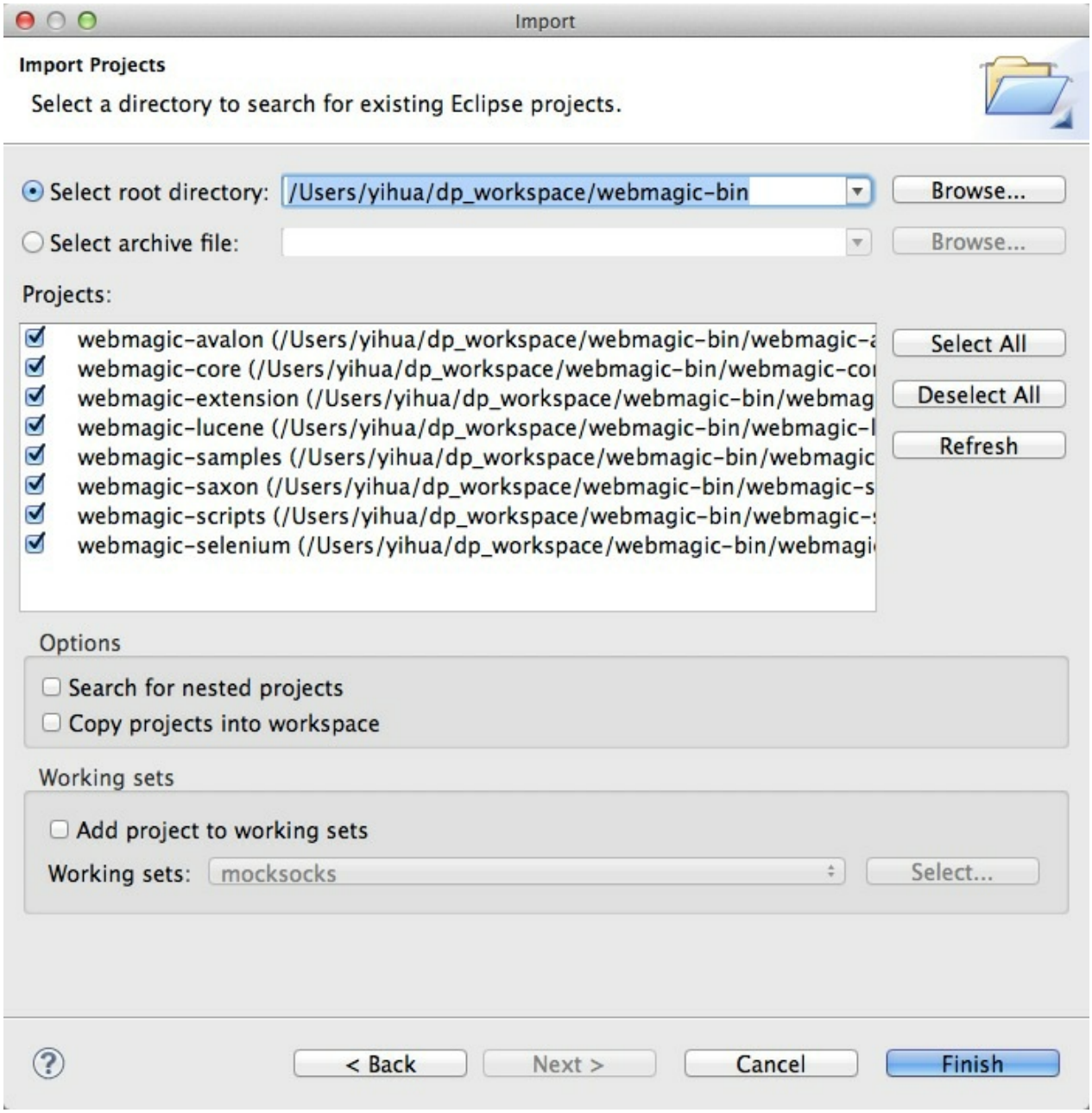
If m2e plug is not installed, but you have install Maven, it is relatively easy to handle. Command in the root directory of the project:

mvn eclipse:eclipse

Generate Eclipse Maven project structure configuration file, then select File->Import and open General->Existing Projects into Workspace for import project.



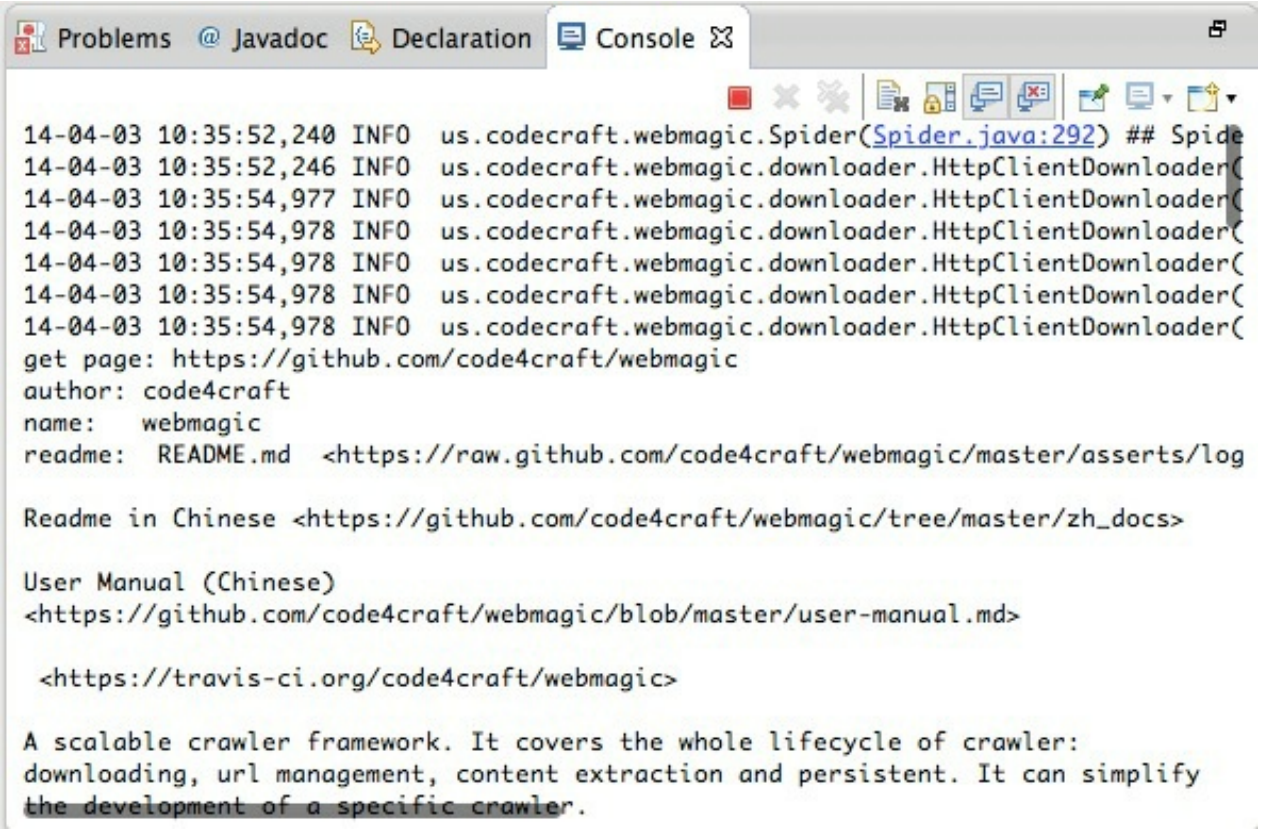
After importing the project to see the selection screen, you can click finish.


Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
/Users/yihua/dp_workspace/webmagic-bin
Browse...
Select archive file:
Browse...
Projects:
webmagic-avalon (/Users/yihua/dp_workspace/webmagic-bin/webmagic-a
webmagic-core (/Users/yihua/dp_workspace/webmagic-bin/webmagic-co
webmagic-extension (/Users/yihua/dp_workspace/webmagic-bin/webmag
webmagic-lucene (/Users/yihua/dp_workspace/webmagic-bin/webmagic-l
webmagic-samples (/Users/yihua/dp_workspace/webmagic-bin/webmagic
webmagic-saxon (/Users/yihua/dp_workspace/webmagic-bin/webmagic-s
webmagic-scripts (/Users/yihua/dp_workspace/webmagic-bin/webmagic-
webmagic-selenium (/Users/yihua/dp_workspace/webmagi
Select All
Deselect All
Refresh
Options
Search for nested projects
Copy projects into workspace
Working sets
Add project to working sets
Working sets:
mocksocks
Select...
< Back
Next >
Cancel
Finish

3.3 source code compilation and execution

After the import is successful, there should be no mistake compilation! At this point you can run the webmagic-core project comes example: "us.codecraft.webmagic.processor.example.GithubRepoPageProcessor".

Also, see the console output is as follows, it indicates that the source code to compile and execute a success!



4. Basic crawler

In WebMagic, the basic realization of a crawler only need to write a class that implements `PageProcessor` interface. This class basically contains all the code to crawl a site, you need to write.

At the same time this section will describe how to use WebMagic extraction API, as well as the most common problem crawl save the results.

4.1 Implement PageProcessor

This is part of our `GithubRepoPageProcessor` directly through this example to introduce the write mode `PageProcessor`. I will PageProcessor customization is divided into three parts, namely, crawlers configuration, extracted page elements and links discovery.

```
public class GithubRepoPageProcessor implements PageProcessor {

    // Part I: crawl the site configuration, including coding, crawler space, retries, et
c.
    private Site site = Site.me().setRetryTimes(3).setSleepTime(1000);

    @Override
    // Process custom crawler logic core interfaces, where the preparation of extraction
logic
    public void process(Page page) {
        // Part II: the definition of how to extract information about the page, and pres
erved
        page.putField("author", page.getUrl().regex("https://github\\.com/(\\w+)/.*").toS
tring());
        page.putField("name", page.getHtml().xpath("//h1[@class='entry-title public']/str
ong/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()"));

        // Part III: From the subsequent discovery page url address to crawler
        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())
                //From "https://github.com/code4craft" began to grasp
                .addUrl("https://github.com/code4craft")
                //Open 5 threads of Crawler
                .thread(5)
                //Start Crawler
                .run();
    }
}
```

4.1.1 Crawler configuration

The first part on the configuration of crawlers, including coding, crawl interval, timeout, retries, but also includes some analog parameters such as User Agent, cookie, and proxy settings, we'll Chapter 5 - "crawlers configuration" in introduced. Here we briefly set about: retries 3 times, crawl interval of 1 second.

4.1.2 Extraction of page elements

The second part is the core of the crawlers: for download to Html page from you how to extract the information you want? WebMagic uses mainly three extraction technologies: XPath, regular expressions and CSS selectors. In addition, the content JSON format, you can use JsonPath resolution.

1. XPath

   Originally XPath is a query language for XML elements acquired, but also more convenient for Html. E.g:

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

   This code uses XPath, it means "find all the class attribute 'entry-title public' the h1 element, and find a child node of his strong child node, a node and extracts text information." Corresponding Html is like this:

2. CSS selector

   CSS and XPath selectors are similar language. If we did front-end development, know for sure that $('h1.entry-title') the wording of meaning. Objectively speaking, it is more than XPath to write simpler, but if you write more complex extraction rules, it is relatively little trouble.

3. Regular Expressions

   Regular expressions are a universal language text extraction.

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

This code uses a regular expression that matches all "https://github.com/code4craft/webmagic" such a link.

4. JsonPath

   JsonPath in XPath is a language very similar to that used to quickly locate a Json from content. WebMagic used JsonPath format can be found here: https://code.google.com/p/json-path/

4.1.3 link found

With the processing logic page, our crawlers will be close to done!

But now there is a problem: a page of the site is a lot of from the beginning we can not all listed, then follow the link to discover how, is an indispensable part of a crawler.

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

This code is divided into two parts, `page.getHtml().links().regex("(https://github\\.com/\\w+/\\w+)").all()` With Get to meet all "(https://github\\.com/\\w+/\\w+)" this regular expression links, `page.addTargetRequests()` these links will be added to the queue to be crawled go.

4.2 Selectable chain API

Chain API `Selectable` relevant is a core function of WebMagic. Selectable interface to use, you can complete extraction chain directly to page elements, and need to take care of details withdrawn.

Can be seen in the earlier example, page.getHtml() returns a `Html` object that implements the `Selectable` interface. This interface contains a number of important ways. I divide it into two categories: partial extraction section and get results.

Extraction section 4.2.1 API:

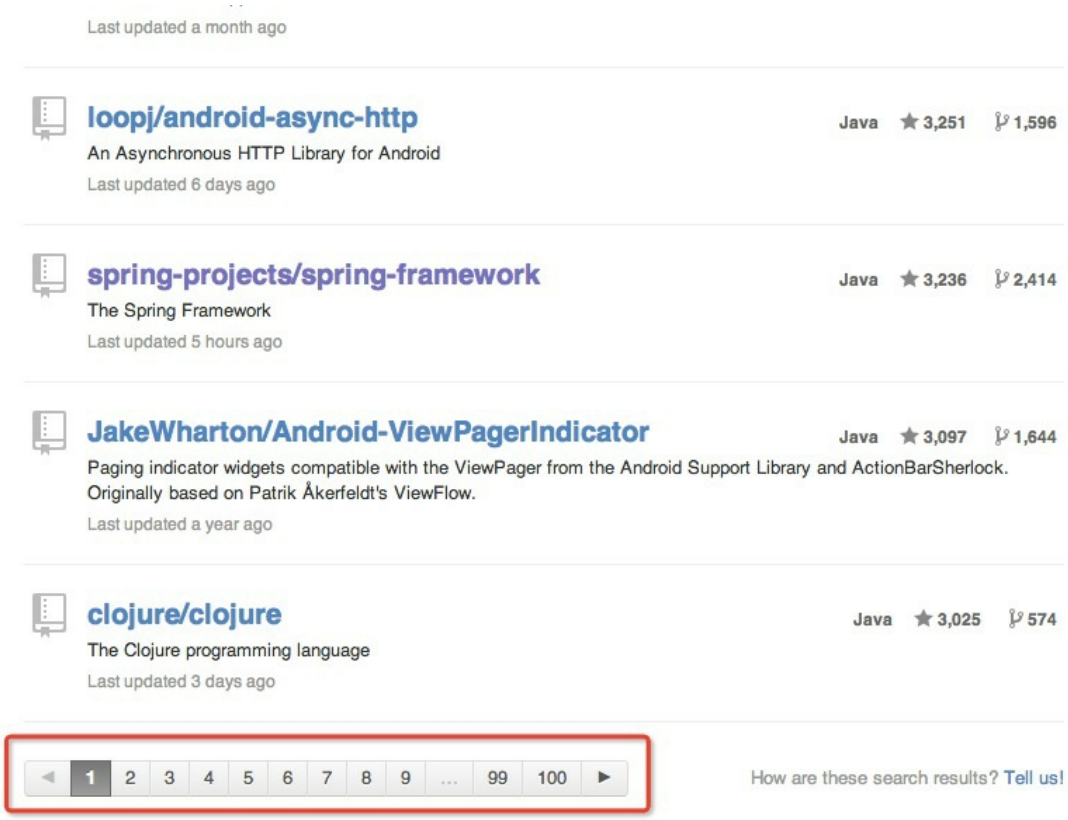
Method	Description	Examples
xpath(String xpath)	Using XPath selectors	html.xpath("//div[@class='title']")
$(String selector)	Use CSS selector to choose	html.$("div.title")
$(String selector,String attr)	Use CSS selector to choose	html.$("div.title","text")
css(String selector)	Function with $(), using CSS selector to choose	html.css("div.title")
links()	Select All link	html.links()
regex(String regex)	Use regular expressions to extract	html.regex("\(.*?)\")
regex(String regex,int group)	Use regular expressions to extract and specify the capture group	html.regex("\(.*?)\",1)
replace(String regex, String replacement)	Replace the contents	html.replace("\","")

This part is extracted API returns a `Selectable` interfaces, meaning that extraction is supported chained calls. Let me use an example to explain the use of chain API.

For example, I now want to grab all Java projects on github, these items can be https://github.com/search?l=Java&p=1&q=stars%3A%3E1&s=stars&type=Repositories

see the search results.

To avoid crawling too wide, I specify to crawl only link from the tab section. The crawl rules are more complex, I would be how to write?



First, see page html structure looks like this:

```
▼<div class="pagination" data-pjax="true">
    <span class="disabled prev_page">◄</span>
    <span class="current">1</span>
    <a href="/search?l=Java&p=2&q=stars%3A%3E1&s=stars&type=Repositories" rel="next">2</a>
    <a href="/search?l=Java&p=3&q=stars%3A%3E1&s=stars&type=Repositories">3</a>
    <a href="/search?l=Java&p=4&q=stars%3A%3E1&s=stars&type=Repositories">4</a>
    <a href="/search?l=Java&p=5&q=stars%3A%3E1&s=stars&type=Repositories">5</a>
    <a href="/search?l=Java&p=6&q=stars%3A%3E1&s=stars&type=Repositories">6</a>
    <a href="/search?l=Java&p=7&q=stars%3A%3E1&s=stars&type=Repositories">7</a>
    <a href="/search?l=Java&p=8&q=stars%3A%3E1&s=stars&type=Repositories">8</a>
    <a href="/search?l=Java&p=9&q=stars%3A%3E1&s=stars&type=Repositories">9</a>
    <span class="gap">…</span>
    <a href="/search?l=Java&p=99&q=stars%3A%3E1&s=stars&type=Repositories">99</a>
    <a href="/search?l=Java&p=100&q=stars%3A%3E1&s=stars&type=Repositories">100</a>
    <a href="/search?l=Java&p=2&q=stars%3A%3E1&s=stars&type=Repositories" class="next_page" rel="next">►</a>
</div>
```

Then I can use CSS selectors to extract this div, and then to take all the links. For insurance purposes, I'll use regular expressions to define what the format of the extracted URL, and then the final wording is like this:

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

Then we can put these URL added to the list to crawl:

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

It is not simple? In addition to finding the link, chain drawn Selectable can also do a lot of work. In Chapter 9, we will re-mentioned examples.

4.2.2 Get Results API:

When the call chain, we generally want to get a string type of result. This time we need to use the API to obtain the results. We know that an extraction rule, either XPath, CSS selectors or regular expression, it is always possible to extract a plurality of elements. WebMagic these were unified, you can get to one or more elements through different API.

Method	Description	Examples
get()	returns a result of type String	String link= html.links().get()
toString()	function with get (), returns a result of type String	String link= html.links().toString()
all()	returns all extraction results	List links= html.links().all()
match()	is there a match result	if (html.links().match()){ xxx; }

For example, we know that the page will have a result, you can use selectable.get() or selectable.toString() to get this result.

Here selectable.toString() uses the toString() this interface, as well as to output and frameworks when combined, more convenient. Because under normal circumstances, we only need to select one element!

selectable.all() will get to all the elements.

Well, up to now, look back at 3.1 GithubRepoPageProcessor, you may feel more clear, right? Specifies the main method, the results can already be seen crawling in the console output.

4.3 Save the results

Well, crawlers written and now we may have a question: If I want to grab the results saved, how to do it? Components WebMagic to hold the result is called `Pipeline`. For example, we adopted "console output" it is through a built-in Pipeline completed, it is called `ConsolePipeline`. Well, I now want to save the results down by Json format, how to do it? I just need to be replaced to achieve Pipeline "JsonFilePipeline" on it.

```
public static void main(String[] args) {
    Spider.create(new GithubRepoPageProcessor())
            // From "https://github.com/code4craft" began to grasp
            .addUrl("https://github.com/code4craft")
            .addPipeline(new JsonFilePipeline("D:\\webmagic\\"))
            // Open 5 threads of Crawl
            .thread(5)
            // Start Crawl
            .run();
}
```

Like this downloaded file will be saved in the disk D: directory webmagic.

By customizing Pipeline, we can achieve save the results to a file, a database and a series of functions. This will be introduced in Chapter 7, "to extract a result Handling".

Thus far, we have completed the basic preparation of a crawler, but also has a number of customization features.

4.4 Crawler configuration, start and stop

4.4.1 Spider

`Spider` is crawler start entrance. Before starting the crawlers, we need to use a `PageProcessor` create a Spider object, and then use the `run()` to start. While other components of the Spider (Downloader, Scheduler, Pipeline) can be set by a set method.

Method	Description	Examples
create(PageProcessor)	Create Spider	spider.create(new GithubRepo
addUrl(String…)	Add initial URL	spider.addUrl("http://webmagic
addRequest(Request...)	Add initial Request	spider.addRequest("http://webr
thread(n)	n threads open	spider.thread(5)
run()	starts, blocking the current thread of execution	spider.run()
start()/runAsync()	asynchronous start, continue with the current thread	spider.start()
stop()	stop crawler	spider.stop()
test(String)	crawl a page to test	spider.test("http://webmagic.io/
addPipeline(Pipeline)	add a Pipeline, a Spider can have multiple Pipeline	spider.addPipeline(new Conso
setScheduler(Scheduler)	Settings Scheduler, a Spider must have at a Scheduler	spider.setScheduler(new Redis
	Settings	

setDownloader(Downloader)	Downloader, a Spider must have at a Downloader	spider.setDownloader(new SeleniumDownloader())
get(String)	synchronous calls, and direct access to the results	ResultItems result = spider.get("http://webmagic.io/d
getAll(String…)	synchronous calls, and direct access to a bunch of results	List<ResultItems> results = spic .getAll("http://webmagic.io/docs "http://webmagic.io/xxx")

4.4.2 Site

The site itself, some configuration information, such as encoding, HTTP headers, timeout, retry strategies, agents, etc., can be configured by setting `Site` object.

Method	Description	Examples
setCharset(String)	set the encoding	site.setCharset("utf-8")
setUserAgent(String)	Settings UserAgent	site.setUserAgent("Spider")
setTimeOut(int)	set the timeout in milliseconds	site.setTimeOut(3000)
setRetryTimes(int)	Settings retries	site.setRetryTimes(3)
setCycleRetryTimes(int)	Setting cycle retries	site.setCycleRetryTimes(3)
addCookie(String,String)	add a cookie	site.addCookie("dotcomt_user","code
setDomain(String)	set up the domain name, the domain name to be set later, addCookie only take	site.setDomain("github.com")

	effect	
addHeader(String,String)	add a addHeader	site.addHeader("Referer","https://gith
setHttpProxy(HttpHost)	Http proxy settings	site.setHttpProxy(new HttpHost("127.0.0.1",8080))

Wherein the loop retry cycleRetry version 0.3.0 is added mechanism.

This mechanism will fail to download url back into the tail of the queue retry until the number of retries to ensure that no leakage grasping for some reason the network page.

4.5 Introduction extraction tools

WebMagic extraction main use of the Jsoup and my own development tools Xsoup.

4.5.1 Jsoup

Jsoup is a simple HTML parser, and it supports the use of CSS selectors way to find elements. In order to develop WebMagic, I Jsoup source conducted a detailed analysis of specific articles see Jsoup study notes.

4.5.2 Xsoup

Xsoup is based Jsoup I developed an XPath parser.

Before using the parser WebMagic HtmlCleaner, there are some problems during use. The main problem is XPath error position is not accurate, and it is not reasonable code structure, it is difficult to customize. I finally realized Xsoup, making it necessary to develop more in line with crawlers. It is gratifying, tested, Xsoup performance than HtmlCleaner faster than doubled.

Xsoup development up to now, has been supported crawler common syntax, the following are some of them have supported syntax table:

Name	Expression	Support
nodename	nodename	yes
immediate parent	/	yes
parent	//	yes
attribute	[@key=value]	yes
nth child	tag[n]	yes
attribute	/@key	yes
wildcard in tagname	/*	yes
wildcard in attribute	/[@*]	yes
function	function()	part
or	a | b	yes since 0.2.0
parent in path	. or ..	no

predicates	price>35	no
predicates logic	@class=a or @class=b	yes since 0.2.0

Also my own definition for several crawlers, it is very convenient XPath functions. Note, however, these functions are not XPath standards.

Expression	Description	XPath1.0
text(n)	n-th child node text directly and 0 for all	text() only
allText()	all direct and indirect text child	not support
tidyText()	all direct and indirect child nodes text, and replace some of the labels wrap, so plain text display cleaner	not support
html()	internal html, html tag does not include itself	not support
outerHtml()	internal html, including tags html itself	not support
regex(@attr,expr,group)	@attr here and can be selected from the group, the default is group0	not support

4.5.3 Saxon

Saxon is a powerful parser XPath support XPath 2.0 syntax. `Webmagic-saxon` integration of Saxon is a tentative, but now it seems, XPath 2.0's advanced grammar, it seems that users are not many crawlers development.

4.6 Monitor with JMX

Crawlers monitor is 0.5.0 new features. Using this feature, you can check the implementation of the crawlers - how many pages have been downloaded, how many pages, how many starts the thread and other information. The functionality through JMX implementations, you can use the tool to see Jconsole etc. JMX local or remote crawler information.

If you will not JMX does not matter, because its use is relatively simple, this chapter will use more detailed explanation. If you want to find out where the principle, you may need some knowledge of JMX is recommended reading: JMX Finishing. I have a lot of part of the article was a reference.

Note: If you define a Scheduler, you need to use this class implements `MonitorableScheduler` interface to view "LeftPageCount" and "TotalPageCount" these two messages.

4.6.1 Add monitoring project

Add monitoring is very simple, get a `SpiderMonitor` singleton `SpiderMonitor.instance()` , and you want to monitor Spider registration go to. You can register multiple Spider in to `SpiderMonitor` .

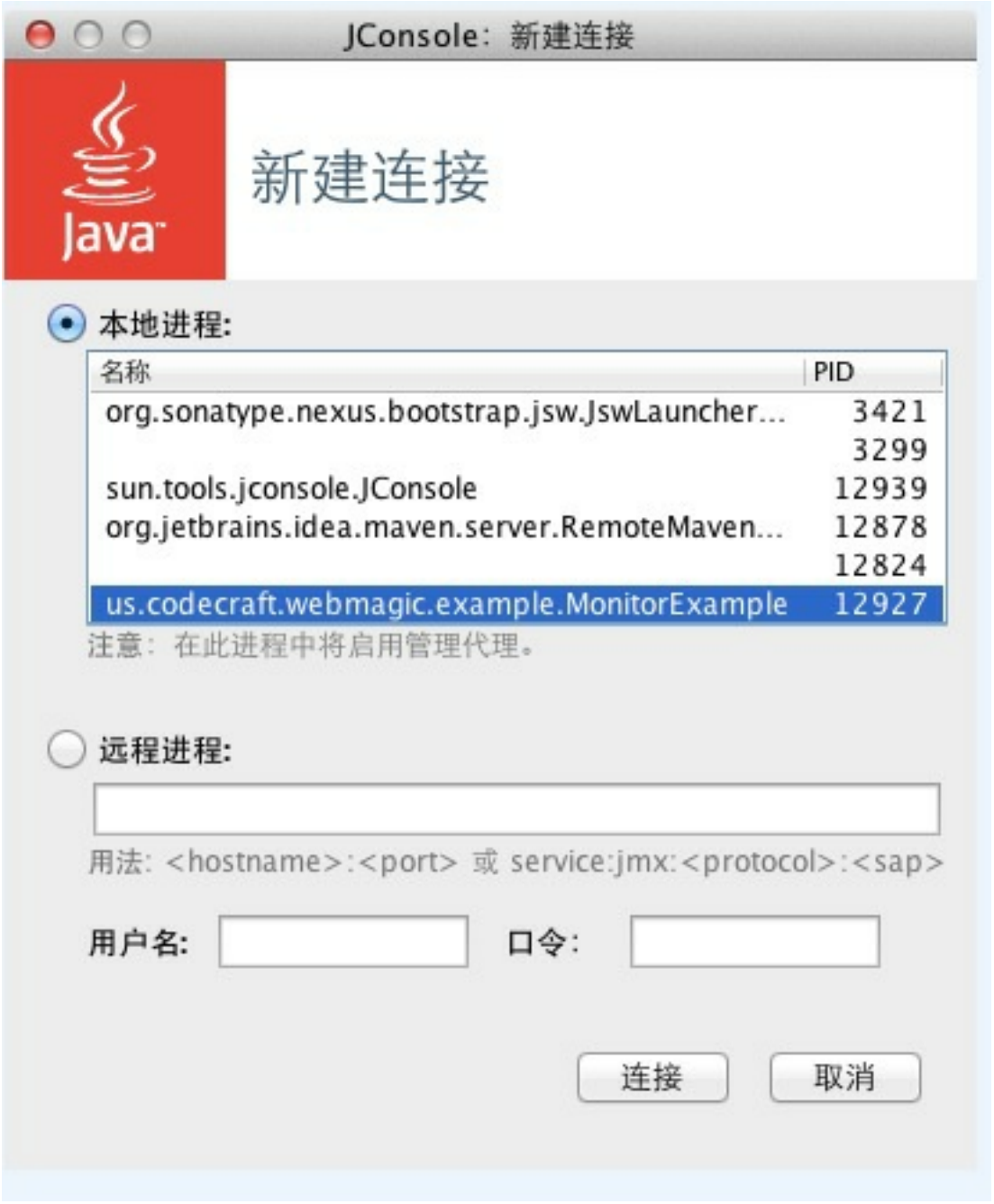
```
public class MonitorExample {

    public static void main(String[] args) throws Exception {

        Spider oschinaSpider = Spider.create(new OschinaBlogPageProcessor())
                .addUrl("http://my.oschina.net/flashsword/blog");
        Spider githubSpider = Spider.create(new GithubRepoPageProcessor())
                .addUrl("https://github.com/code4craft");

        SpiderMonitor.instance().register(oschinaSpider);
        SpiderMonitor.instance().register(githubSpider);
        oschinaSpider.start();
        githubSpider.start();
    }
}
```

4.6.2 View monitoring information

WebMagic monitoring using JMX provides control, you can use any JMX-enabled client to connect. We are here to JDK comes JConsole example. We first start a Spider

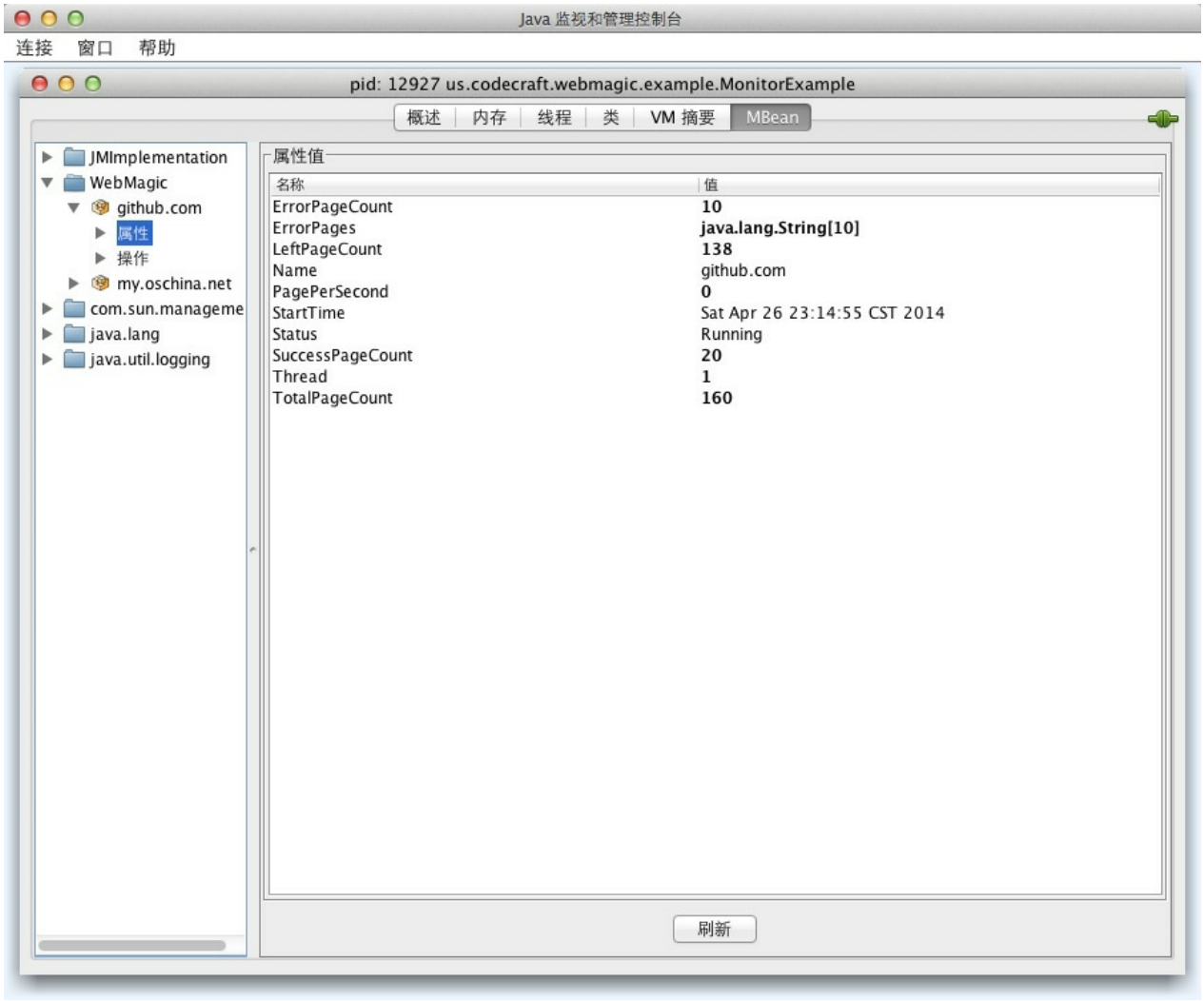
WebMagic and add monitor code. We then JConsole to view it.

We start the program in accordance with the example 4.6.1, and then enter the command jconsole (under Windows Command Line Interface (CLI) is entered in the DOS jconsole.exe) to start JConsole.



Here we have chosen to start a local process WebMagic choose "MBean" After connecting, opening the "WebMagic", will be able to see all the information has been monitoring the Spider!

Here we can also choose "action" in the operation where you can choose to start - start() and termination crawlers -stop(), which will directly call the corresponding Spider's start() and stop() method to achieve the purpose of basic control.



4.6.3 Extended Monitoring Interface

In addition to some of the existing monitoring information, if you have more information needs to be monitored, can also be extended to solve. You can inherit `SpiderStatusMXBean` to achieve the expansion, specific examples can be seen here: Custom extensions demo.

5. Using annotations written in crawler

WebMagic support the use of a unique style to write a comment crawler introduced webmagic-extension package to use this feature.

In annotation mode, using a simple object add comments, use minimal amount of code to complete the preparation of a crawler. For simple crawlers, write simple and easy to understand, and also very easy to manage. It is also a major feature of WebMagic, I called it `OEM` (Object/Extraction Mapping).

Annotation model development approach is this:

1. First you need to define the extracted data, and write a class.
2. On the class stated `@TargetUrl` annotation, define which URL to download and extraction.
3. Add the `@ExtractBy` annotation on the class of the field, this field uses the definition of what way decimated.
4. Define the result is stored.

Here we still examples github chapter IV, to write a similar function crawlers, to explain the use of annotations. Final preparation of good crawler is like this, is not it easier?

```
@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 = tru
e)
    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();
    }
}
```

5.1 Model class write

Examples with Chapter IV, we are here to extract a github project name, author, and three profile information, so we define a Model class.

```
public class GithubRepo {

    private String name;

    private String author;

    private String readme;

}
```

Here are omitted getter and setter methods.

In extracting the end, we will get one or more instances of this class, this is the result of crawlers.

5.2 TargetUrl and HelpUrl

In the second step, we still have to define how to discover URL. Here we first introduce two concepts: `@TargetUrl` and `@HelpUrl` .

5.2.1 TargetUrl and HelpUrl

`HelpUrl/TargetUrl` crawler is a very effective development model, TargetUrl we eventually crawled URL, eventually want the data from here; and HelpUrl is to find the final URL, we need to access the page. Almost all vertical reptile needs, can be attributed to these two types of URL processing:

- For the blog page, HelpUrl is a list of pages, TargetUrl article pages.
- For the forum, HelpUrl is the list of posts, TargetUrl is a post for details.
- For the ecommerce website, HelpUrl is a list of categories, TargetUrl commodity details.

In this example, TargetUrl page is the final project, and the project is HelpUrl search page, which will display links to all projects.

With this knowledge, we have for example defined URL format:

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

5.2.2 TargetUrl custom regular expressions

Here we are using regular expressions to specify the URL scope. Careful friends may be, will know `.` reserved character is a regular expression, this is not it wrong? In fact, here for the convenience, WebMagic own custom regex for URL's, mainly by two changes:

- The URL of characters commonly used default `.` do escape into the `\.`
- The "*" replaced - "*" asterisk may be used directly.

For example, `https://github.com/*` here is a valid expression that all URL under `https://github.com/` .

In WebMagic, a page from the URL `TargetUrl` obtained, provided that they meet TargetUrl format, will also be downloaded. So even if you do not specify `HelpUrl` also

possible - for example, there will always be some blog link "Next" on page , in this case need to specify HelpUrl.

5.2.3 sourceRegion

TargetUrl also supports the definition of `sourceRegion`, this parameter is an XPath expression that specifies the URL from where to get - not sourceRegion the URL will not be extracted.

5.3 `@ExtractBy`

`@ExtractBy` annotation is used to extract a element, which describes an extraction rule.

5.3.1 Intro to @ExtractBy annotation

@ExtractBy Annotation major role in the field, it means "the use of the extraction rules, to save the extracted result into this field." E.g:

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

Here "//div[@id='readme']/text()" is a representation of the XPath extraction rules, and to extract the results will be saved to the readme field.

5.3.2 use other ways to extract

In addition to XPath, we can also use other ways to extract extraction, including CSS selectors, regular expressions and JsonPath, indicated in a annotation after the `type` can.

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

5.3.3 notNull

@ExtractBy Contains a `notNull` property, if familiar with mysql students must be able to understand what it means: This field does not allow empty. If empty, this extract to the result is discarded. For critical attributes (such as the title of the article, etc.) some of the pages, set `notnull` to `true`, you can effectively filter out unwanted pages.

`NotNull` default is `false`.

5.3.4 multi (deprecated)

`Multi` is a boolean attribute that represents this extraction rule corresponding number of records or a single record. Corresponding to this field must be `java.util.List` type. After 0.4.3, when the field is a List type, this property will automatically true, no longer need to set up.

- 0.4.3 before

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

- 0.4.3 and later

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

5.3.5 ComboExtract (deprecated)

`@ComboExtract` is a more complex notes, it can be a combination of a plurality of extraction rules, combinations including "AND/OR" in two ways.

Use the Xsoup 0.2.0 version WebMagic 0.4.3 version. In this version, XPath syntax supported greatly strengthened, not only supports the XPath and regular expressions used in combination, also supports the "|" ORed. Therefore author thinks, `ComboExtract` this complex combination, are no longer needed.

- XPath combination with regular expressions

```
@ExtractBy("//div[@class='BlogStat']/regex('\\d+-\\d+-\\d+\\s+\\d+:\\d+')")
private Date date;
```

- XPath or take

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

5.3.6 ExtractByUrl

`@ExtractByUrl` is a separate annotation, it means "be extracted from the URL." It only supports regular expressions as extraction rules.

5.4 Use on Class ExtractBy

In the previous annotation mode, we have a page corresponds to only one result. If a page has multiple extracted record it? For example, in "QQ food" list page http://meishi.qq.com/beijing/c/all, I want to extract all businesses Get names and information, how to do it?

Use `@ExtractBy` annotation on the class can solve this problem.

Using this annotation on the class meaning is very simple: use this result to extract a region, so that this region corresponds to a result.

```
@ExtractBy(value = "//ul[@id=\"promos_list2\"]/li",multi = true)
public class QQMeishi {
    ……
}
```

Correspondence, then use the `@ExtractBy` in this field in the class, then from this area rather than the entire page extraction. If this time still want to extract from the entire page, you can set the `source = RawHtml` .

```
@TargetUrl("http://meishi.qq.com/beijing/c/all[\\-p2]*")
@ExtractBy(value = "//ul[@id=\"promos_list2\"]/li",multi = true)
public class QQMeishi {

    @ExtractBy("//div[@class=info]/a[@class=title]/h4/text()")
    private String shopName;

    @ExtractBy("//div[@class=info]/a[@class=title]/text()")
    private String promo;

    public static void main(String[] args) {
        OOSpider.create(Site.me(), new ConsolePageModelPipeline(), QQMeishi.class).addUrl
("http://meishi.qq.com/beijing/c/all").thread(4).run();
    }

}
```

5.5 Results of type conversion

Type Conversion (`Formatter` mechanism) is WebMagic 0.3.2 increased functionality. Because the content is always drawn to String, and we want may be other types of content. Formatter can be drawn into the content is automatically converted into a number of basic types without having to manually use the code conversion.

E.g:

```
@ExtractBy("//ul[@class='pagehead-actions']/li[1]//a[@class='social-count js-social-count
']/text()")
private int star;
```

5.5.1 supports automatic conversion type

Automatic conversion supports all basic types and packing type.

Primitive	packing type
int	Integer
long	Long
double	Double
float	Float
short	Short
char	Character
byte	Byte
boolean	Boolean

In addition, it supports `java.util.Date` type conversion. However, when you convert, you need to specify the Date format. Format according to standard JDK defined, specific norms can be seen here: http://java.sun.com/docs/books/tutorial/i18n/format/simpleDateFormat.html

```
@Formatter("yyyy-MM-dd HH:mm")
@ExtractBy("//div[@class='BlogStat']/regex('\\d+-\\d+-\\d+\\s+\\d+:\\d+')")
private Date date;
```

5.5.2 explicitly specify conversion types

Under normal circumstances, Formatter will be converted according to the field type, but under special circumstances, we will need to manually specify the type. This occurs mainly in the field type is `List` time.

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

5.5.3 Custom Formatter (TODO)

In fact, in addition to the automatic type conversion, Formatter also can do some things to process the results. For example, we have a demand scenario, the results need to be extracted as a result of part of the mosaic on the part of the string to use. Here, we define a `StringTemplateFormatter` .

```
public class StringTemplateFormatter implements ObjectFormatter<String> {

    private String template;

    @Override
    public String format(String raw) throws Exception {
        return String.format(template, raw);
    }

    @Override
    public Class<String> clazz() {
        return String.class;
    }

    @Override
    public void initParam(String[] extra) {
        template = extra[0];
    }
}
```

Well, we can, after extraction, to do some of the simple operation!

```
@Formatter(value = "author is %s",formatter = StringTemplateFormatter.class)
@ExtractByUrl("https://github\\.com/(\\w+)/.*")
private String author;
```

This feature in version 0.4.3 BUG, and will be fixed in 0.5.0 are open.

5.6 a complete process

Prior to the date, we know the URL and extract the relevant API, a crawler has been basically completed the preparation.

```
@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 = tru
e)
    private String name;

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

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

5.6.1 crawler creation and start

Entrance annotation model is `OOSpider`, it inherits the `Spider` class that provides special creation method, other methods are similar. Create an annotation mode crawles require one or more `Model` class, and one or more `PageModelPipeline` -- define the results manner.

```
public static OOSpider create(Site site, PageModelPipeline pageModelPipeline, Class... pa
geModels);
```

5.6.2 PageModelPipeline

Under annotation mode, the results of class called `PageModelPipeline`, by implementing it, you can customize your results approach.

```
public interface PageModelPipeline<T> {

    public void process(T t, Task task);

}
```

PageModelPipeline with Model class is the corresponding, may correspond to a plurality of Model PageModelPipeline. Except when you create, you can also

```
public OOSpider addPageModel(PageModelPipeline pageModelPipeline, Class... pageModels)
```

Method, add a Model at the same time, you can add a PageModelPipeline.

5.6.3 Conclusion

Well, now we have to complete this example:

```
@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 = tru
e)
    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();
    }
}
```

5.7 AfterExtractor

Sometimes, annotation mode can not meet all the needs, we may need to write the code to do some things, this time we should use the `AfterExtractor` interfaces.

```
public interface AfterExtractor {

    public void afterProcess(Page page);
}
```

`afterProcess` extraction method in the end, after the fields are initialized to be called, you can deal with some special logic. Like in the example using Jfinal ActiveRecord persistence webmagic crawled blog:

```
// TargetUrl mean only the following URL format will be extracted to generate the object
model
// Here is to do a little positive change, '' The default is no need to escape, and the '
*' will be automatically replaced with '*', as described URL looked a little uncomfortabl
e ...
// Inherited jfinal the Model
// Implement AfterExtractor interfaces can perform other operations after filling propert
ies
@TargetUrl("http://my.oschina.net/flashsword/blog/*")
public class OschinaBlog extends Model<OschinaBlog> implements AfterExtractor {

    // Will be automatically extracted with ExtractBy annotation fields and filling
    // Default xpath grammar
    @ExtractBy("//title")
    private String title;

    //Extract can be defined syntax Css, Regex, etc.
    @ExtractBy(value = "div.BlogContent", type = ExtractBy.Type.Css)
    private String content;

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

    @Override
    public void afterProcess(Page page) {
        //Jfinal property is actually a Map instead of the field, it does not matter, I w
ant to go in filling
        this.set("title", title);
        this.set("content", content);
        this.set("tags", StringUtils.join(tags, ","));
        //save
        save();
    }

    public static void main(String[] args) {
        C3p0Plugin c3p0Plugin = new C3p0Plugin("jdbc:mysql://127.0.0.1/blog?characterEnco
ding=utf-8", "blog", "password");
        c3p0Plugin.start();
        ActiveRecordPlugin activeRecordPlugin = new ActiveRecordPlugin(c3p0Plugin);
        activeRecordPlugin.addMapping("blog", OschinaBlog.class);
        activeRecordPlugin.start();
        //Start webmagic
        OOSpider.create(Site.me().addStartUrl("http://my.oschina.net/flashsword/blog/1457
96"), OschinaBlog.class).run();
    }
}
```

Conclusion

Annotation mode is now regarded as the end of the presentation, in WebMagic, the annotation model is in fact based entirely on `webmagic-core` the `PageProcessor` and `Pipeline` extension implementation, interested friends can go to look at the code.

This is partly achieved but it is still more complex, there is a problem if you find some of the details of the code, welcomed feedback to me.

6.Customize Components

In the chapter One, we reference the components of WebMagic. WebMagic the biggest advantage is that you can customize the function of components flexible to achieve the function you want to have.

In the Spider class, `PageProcessor`、`Downloader`、`Scheduler` and `Pipeline` are the fields of Spider class. Beside the `PageProcessor` have been assigned when the Spider create, the other three components can be changed by the setter method.

Method	Description	Example
setScheduler()	Change the Scheduler	spipder.setScheduler(new FileCacheQueueScheduler("D:\data\webm
setDownloader()	Change the Downloader	spipder.setDownloader(new SeleniumDownloader()))
addDownloader()	Change the Pipeline，one Spider can have a few of pipeline	spipder.addPipeline(new FilePipeline())

6.1 Customize Pipeline

When the extract finished, we use `Pipeline` to persist the result of extract.We can also customize the pipeline to do some common function. In this chapter we will introduce the `Pipeline`, and use two examples to explane how to customize the pipeline.

6.1.1 Introduction of Pipeline

The interface of `Pipeline` define is here:

```
public interface Pipeline {

    // ResultItems persist the result of extract, it is a structure of map
    // The data in the page.putField(key,value) can use the ResultItems.get(key) to get
    public void process(ResultItems resultItems, Task task);
}
```

We can see, `Pipeline` persist the data which was extracted by the `PageProcessor`. This work we can also do in the `PageProcessor`. But why we use the `Pipeline`? There is some reason for this:

1. To separate the modules. The extract of page and persist the data are the to stages of a spider. On one hand, separate the modules can make the structure of the code more clear. On the other hand, we can separate the process, process in another thread or even in another server.
2. The function of `Pipeline` is more stable, it is very easy to make it as a common component. There is a big difference between process of different pages. But the persist of data is almost the same,such as save in a file or persist in the database. It is very commons for almost of the pages. There is lots of common `Pipeline` in the WebMagic, such as write to the console, save in a file, save in a file as a JSON format.

In the WebMagic, a `Spider` can have a lot of `Pipeline`, to use the `Spider.addPipeline()` can add a `Pipeline`. These `Pipeline` can all be process. For example, you can use:

```
spider.addPipeline(new ConsolePipeline()).addPipeline(new FilePipeline())
```

You can write the data on the console and save in the file.

6.1.2 Put the result on the console

When we introduce the `PageProcessor`, we use the GithubRepoPageProcessoras a example. There is a chip of the code:

```
public void process(Page page) {
    page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/\\w+/\\w+)
").all());
    page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/\\w+)").al
l());
    //save the author, the data will be save in ResultItems finally
    page.putField("author", page.getUrl().regex("https://github\\.com/(\\w+)/.*").toStrin
g());
    page.putField("name", page.getHtml().xpath("//h1[@class='entry-title public']/strong/
a/text()").toString());
    if (page.getResultItems().get("name")==null){
        //when we set the skip,this page will not be processed by the`Pipeline`
        page.setSkip(true);
    }
    page.putField("readme", page.getHtml().xpath("//div[@id='readme']/tidyText()"));
}
```

Now we want to write the result in the console. ConsolePipeline can do this.

```
public class ConsolePipeline implements Pipeline {

    @Override
    public void process(ResultItems resultItems, Task task) {
        System.out.println("get page: " + resultItems.getRequest().getUrl());
        //Iterator all the result,and put it on the console,the "author","name","readme"a
re all the key,the result is value
        for (Map.Entry<String, Object> entry : resultItems.getAll().entrySet()) {
            System.out.println(entry.getKey() + ":\t" + entry.getValue());
        }
    }
}
```

To Reference this example, you can customize your own `Pipeline`. Get the data from the `ResultItems` and process as your own method.

6.1.3 persist the result in the MySQL

First, we introduce a examplejobhunter. It's a WebMagic which integrate a spring framework to crawl the job information. This example also show how to use Mybatis to persist the data in the MySQL database.

In Java, we have many methods to save the data in database, such as jdbc、dbutils、spring-jdbc、MyBatis. These tools can do the same things, but their complexity is not

the same. If we use JBDC, we should get the data in the ResulrItem and save it.

If we use the ORM framework to persist the data, we will face a big problem. That is the framework all need a well defined model, but not a Key-Value format ResultItem. We use the Mybatis as a example to define a DAO MyBatis-Spring.

```
public interface JobInfoDAO {

    @Insert("insert into JobInfo (`title`,`salary`,`company`,`description`,`requirement`,
`source`,`url`,`urlMd5`) values (#{title},#{salary},#{company},#{description},#{requireme
nt},#{source},#{url},#{urlMd5})")
    public int add(LieTouJobInfo jobInfo);
}
```

All we need to do is to implements a Pipeline,to combine the `ResultItem` and `LieTouJobInfo`.

Annotation mode

Under the annotation mode, there is a PageModelPipeline in the WebMagic：

```
public interface PageModelPipeline<T> {

    //give the well processed object
    public void process(T t, Task task);

}
```

At this time,we can define a JobInfoDaoPipeline to achieve the function:

```
@Component("JobInfoDaoPipeline")
public class JobInfoDaoPipeline implements PageModelPipeline<LieTouJobInfo> {

    @Resource
    private JobInfoDAO jobInfoDAO;

    @Override
    public void process(LieTouJobInfo lieTouJobInfo, Task task) {
        //call the MyBatis DAO to save the result
        jobInfoDAO.add(lieTouJobInfo);
    }
}
```

Basic Pipeline mode

We have finished the work of save the data! But how to use a original `Pipeline` interface? It's very easy! If you want to save a object, then you should save the data as a object when you extract it from a page.

```
public void process(Page page) {
    page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/\\w+/\\w+)
").all());
    page.addTargetRequests(page.getHtml().links().regex("(https://github\\.com/\\w+)").al
l());
    GithubRepo githubRepo = new GithubRepo();
    githubRepo.setAuthor(page.getUrl().regex("https://github\\.com/(\\w+)/.*").toString()
);
    githubRepo.setName(page.getHtml().xpath("//h1[@class='entry-title public']/strong/a/t
ext()").toString());
    githubRepo.setReadme(page.getHtml().xpath("//div[@id='readme']/tidyText()").toString(
));
    if (githubRepo.getName() == null) {
        //skip this page
        page.setSkip(true);
    } else {
        page.putField("repo", githubRepo);
    }
}
```

In the `Pipeline`, you should use

```
GithubRepo githubRepo = (GithubRepo)resultItems.get("repo");
```

then you can get this object

> PageModelPipeline is also implements from the original `Pipeline` interface. It combine the `PageProcessor`. it use the class name as the key and the value is the object.In detail: ModelPipeline.

6.1.4 The WebMagic has already define some Pipeline

WebMagic can write the result to the comsole,save the data in a file or save as a JSON format.

Class	Description	Remark
ConsolePipeline	write to the console	the result must implements the toString() method
		the result must

FilePipeline	save the result in the file	implements the toString() method
JsonFilePipeline	save the data in the file as JSON format	
ConsolePageModelPipeline	(Annotation mode)write to the console	
FilePageModelPipeline	(Annotation mode)save the result in the file	
JsonFilePageModelPipeline	(Annotation mode)save the data in the file as JSON format	the field which want to be saved must have the getter method

6.2 Customized Scheduler

The scheduler is a components of the WebMagic which manage the url. In general, there are two effect of the scheduler:

1. manage the url which is wait to be crawl
2. filter out the repeat url

WebMagic have some common schedular. If you just want to run some simple sprider in your local, then you needn't to customize scheduler. But it is meaningful for you to know some of them.

Class	Description	Remark
DuplicateRemovedScheduler	a abstract class,it provide some template method	extends it can achieve your own function
QueueScheduler	use the memory queue to save the url	
PriorityScheduler	use the priority mamory queue to save the url	the use of memory is bigger than the QueueScheduler, but whne you set the request.priority. it is necessary to use the PriorityScheduler to take the priority effect
FileCacheQueueScheduler	use the file to save the url, when the program exit and start next time， it can crawl the url which have been saved in the file	it need to set the path of the file. It will create two files .urls.txt and .cursor.txt
RedisScheduler	use the redis to save the queue, it can crawl the internet in a distributed system	need to install redis and start it

In the Version 0.5.1, i have rebuild the scheduler. The duplicated remover have been extract to a independent interface: `DuplicateRemover`. Then you can set a different

`DuplicateRemover` for one scheduler. There are two ways of remove the Duplicate.

Class	Description
HashSetDuplicateRemover	use the HashSet to remove, but it needs a lots of memory
BloomFilterDuplicateRemover	use the BloomFilter to remove, use a few of memory. But it may leave out a few url

All the default scheduler use the `HashSetDuplicateRemover` to remove (except the RedisScheduler). If you have a mount of url to do this, we recommend you to use the `BloomFilterDuplicateRemover`. For example:

```
spider.setScheduler(new QueueScheduler()
.setDuplicateRemover(new BloomFilterDuplicateRemover(10000000)) //10000000 is the estimat
e value of urls
)
```

6.3 Use the Downloader

The default Downloader based on the HttpClient. In general, you don't need to implement the Downloader, but there are some reserve cut point in the `HttpClientDownloader`. These are made to meet some different demand.

In the other hand, you may want to download the webpage in some other way, such as use the `SeleniumDownloader` to render the webpage.

Some written in a style common crawlers

Even if you WebMagic framework has been very skilled, will be prepared for some crawlers and some confusion. Such as how to fetch and update periodically, how to crawl dynamic rendering of pages and the like.

In this section I will sort out some common cases, we hope to help readers.

List details the basic combination of page

We start with a simple example to start. For this example, we have a list of pages, this list tab page to show the form that we can traverse these tabs to find all the target page.

1 Introduction example

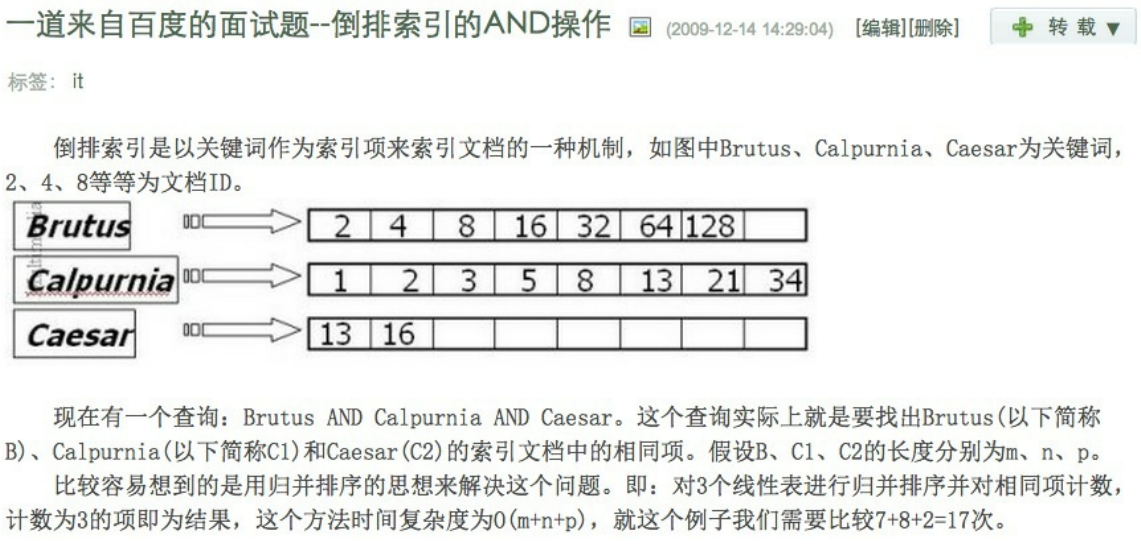
Here we are with the author's Sina blog http://blog.sina.com.cn/flashsword20 as an example. In this example, we want the final blog page article, crawling blog title, content, date and other information, but also to crawl blog links and other information from the list of pages, in order to gain all articles of this blog.

- Page

  Page format is "http://blog.sina.com.cn/s/articlelist_1487828712_0_1.html", where "0_1" in the "1" is a variable number of pages.

- Article Page

  The article on page format is "http://blog.sina.com.cn/s/blog_58ae76e80100g8au.html", where "58ae76e80100g8au" is variable character.



2 article URL found

In this crawler demand, the article URL is our ultimate concern, so how to find all the articles in this blog address is the first step crawlers.

We can use the regular expression `http://blog\\.sina\\.com\\.cn/s/blog_\\w+\\.html` a coarse filter for URL. More complicated here is that this URL is too broad and could crawl to the other blog information, so we must specify the area from the list page Get URL.

Here, we use xpath `//div[@class=\\"articleList\\"]` select all regions, then use links () or xpath `//a/@href` get all the links, and finally the use of regular expression `http://blog\\.sina\\.com\\.cn/s/blog_\\w+\\.html` , a URL filtering to remove some of the" edit "or" more "category links. Thus, we can write:

```
page.addTargetRequests(page.getHtml().xpath("//div[@class=\"articleList\"]").links().rege
x("http://blog\\.sina\\.com\\.cn/s/blog_\\w+\\.html").all());
```

At the same time, we need to find a list of all the pages are added to the URL to be downloaded to go:

```
page.addTargetRequests(page.getHtml().links().regex("http://blog\\.sina\\.com\\.cn/s/arti
clelist_1487828712_0_\\d+\\.html").all());
```

3 Content Extraction

Extracting the article page of information is relatively simple, written corresponding xpath expression to extract it.

```
page.putField("title", page.getHtml().xpath("//div[@class='articalTitle']/h2"));
page.putField("content", page.getHtml().xpath("//div[@id='articlebody']//div[@class='arti
calContent']"));
page.putField("date",
        page.getHtml().xpath("//div[@id='articlebody']//span[@class='time SG_txtc']").reg
ex("\\((.*)\\)"));
```

4 distinguish lists and landing pages

Now, we have defined the target page list and processing the way, now we need to deal with when they make that distinction. In this case, the distinction is very simple, because the list on the page and the destination page URL format is different, so the

URL directly distinguish it!

```
// List
if (page.getUrl().regex(URL_LIST).match()) {
    page.addTargetRequests(page.getHtml().xpath("//div[@class=\"articleList\"]").links().
regex(URL_POST).all());
    page.addTargetRequests(page.getHtml().links().regex(URL_LIST).all());
    // Article Page
} else {
    page.putField("title", page.getHtml().xpath("//div[@class='articalTitle']/h2"));
    page.putField("content", page.getHtml().xpath("//div[@id='articlebody']//div[@class='
articalContent']"));
    page.putField("date",
            page.getHtml().xpath("//div[@id='articlebody']//span[@class='time SG_txtc']")
.regex("\\((.*)\\)"));
}
```

Consider this example the complete code SinaBlogProcessor.java.

5 Summary

In this example, we use several main methods:

- Found a link from a page using regular expressions to specify the location of the filter link.
- PageProcessor deal with two pages, depending on page URL to distinguish between what is required.

Some friends of the reaction, if-else deal with some inconvenience to differentiate #issue83. WebMagic planned future version 0.5.0 added `SubPageProcessor` to solve this problem.

Crawl tip rendered page

With the continued popularity of AJAX technology and the emergence of such AngularJS Single-page application framework now, now js render pages more and more. For crawlers, this page is more annoying: Just extract HTML content, are often unable to get valid information. So how do you deal with this pages? In general there are two approaches:

1. In the crawl phase, crawler built a browser kernel, after performing js rendering the page, and then crawl. This aspect of the corresponding tools `Selenium` , `HtmlUnit` or `PhantomJs` . But these tools there are some efficiency, at the same time it is not so stable. Benefit is to write the rules, like static pages.
2. Because js rendering data page is from the rear end to get, and basically get AJAX, so analysis AJAX request, a request to find the corresponding data is also more feasible approach. And with respect to the page style, this interface is less likely to change. The disadvantage is to find the request, and simulation, is a relatively difficult process, but also requires a relatively large number of analytic experience.

Comparison of two ways, my view is that the demand for one-time or small-scale, with the first approach saves time and effort. But for long-term, large-scale demand, or the second would be ideal. For some sites, and even some js confusing technology, this time, the first way is basically a panacea, while the second will be very complicated.

For the first method, `webmagic-selenium` is one such attempt, which defines a `Downloader` , at the time of the download page, the browser is the core rendering. selenium configuration is relatively complex, but with the version of the platform and, not too stable solution. I can see this blog interested in: use Selenium to crawl dynamic loading pages

Here I introduced a second, and I hope in the end you will find: the original page parsing a front-end rendering, it is not so complicated. Here we AngularJS Chinese community http://angularjs.cn/ as an example.

1 How to determine the front-end rendering

Determine whether the page is rendered js relatively simple way, in a browser to view the source code directly (Windows under Ctrl + U, Mac under the command + alt + u), if no effective information is almost certain to js rendering.

This example, the page heading "Corfu computer network - Front siege division" can not be found in the source code, it can be concluded that js rendering, and this data is obtained AJAX.

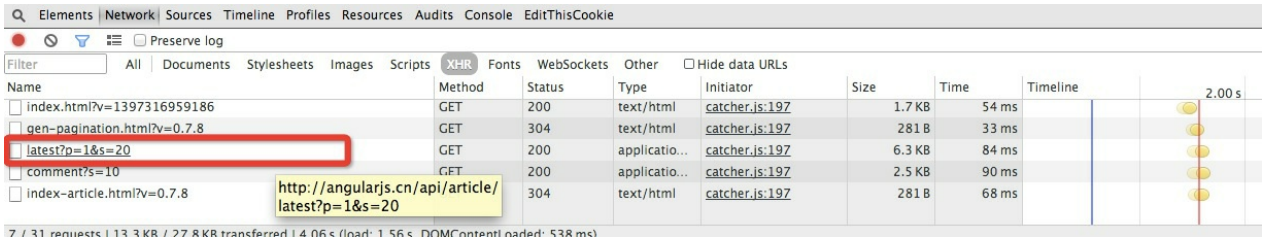
2 analysis request

Here we enter the hardest part: finding the data request. This step helps our tools is the development of tools to view web browser requests.

In Chome example, we open the "Developer Tools" (under Windows is F12, the next Mac is command + alt + i), then refresh the page (there may be a drop-down pages, in short, all that you think might trigger a new data operations), and remember to retain the scene, the one used to request analysis of it!

This step requires a bit of patience, but it is not out of nowhere. First, to help us is above the Sort (All, Document and other options). If it is normal AJAX, appear under `XHR` label, JSONP request under `Scripts` labels, which are two of the more common data types.

Then you can look at the data to determine the size, the results are generally more likely to be larger return data interface. The rest, basically rely on the experience of, for example, where the "latest? P = 1 & s = 20" look very suspicious ...



For suspicious address, this time can look at what is the response body. Here not clear in developer tools, we URL `http://angularjs.cn/api/article/latest?p=1&s=20` copied into the address bar, the request once again (if Chrome recommend installing a jsonviewer,? Show AJAX results easily). See the results, it seems that we want to find.

The same way, we enter into the post details page, find the specific content of the request `http://angularjs.cn/api/article/A0y2` .

3 programming

Recall that before the target page list + example, we will find this demand, with the previous similar, but replaced with a list of ways -AJAX AJAX mode, the data AJAX mode, and returns the data into a JSON. Well, we can still use the way last time, into two pages to be written:

1. Data List

In this list page, we need to find effective information to help us build targets AJAX URL's. Here we see, this is what we should `_id` want id's posts, and posts details of the request, by a few fixed URL plus the id components. Therefore, in this step, we have to manually construct URL, and added to the queue to be crawled. Here we use language JsonPath this option to select the data (webmagic-extension package that provides `JsonPathSelector` to support it).

```
 if (page.getUrl().regex(LIST_URL).match()) {
     //Here we use language JSONPATH this option to select the data
     List<String> ids = new JsonPathSelector("$.data[*]._id").selectList(page.getRaw
Text());
     if (CollectionUtils.isNotEmpty(ids)) {
         for (String id : ids) {
             page.addTargetRequest("http://angularjs.cn/api/article/"+id);
         }
     }
 }
```

2. Objectives data

   With the URL, in fact resolve the target data is very simple, because the JSON data is completely structured, so eliminating the need for our analysis page, write XPath process. Here we still use JsonPath to get the title and content.

```
 page.putField("title", new JsonPathSelector("$.data.title").select(page.getRawText(
)));
 page.putField("content", new JsonPathSelector("$.data.content").select(page.getRawT
ext()));
```

Consider this example the complete code AngularJSProcessor.java

4 Summary

In this example, we analyzed the crawling process a more classic dynamic pages. In fact, the dynamic pages crawled, the biggest difference is: it increases the difficulty of links found. We compare the two development models:

1. The rear end of the rendered page

   Download Helper page => find links => download and analyze the target HTML

2. The front-end rendering pages

   Found auxiliary data => constructing a link => download and analyze the target

AJAX

For a different site, the auxiliary data may have been previously output in page HTML, it may be to request through AJAX, perhaps even process multiple data requests, but the basic pattern is fixed.

However, analysis of these data requests than the page analysis, it is still much more complicated, so this is actually a dynamic page fetch difficult.

The example in this section do hope that in the analysis of the request, provide a model to follow for such crawlers write that `find secondary data => constructing a link => download and analyze the target AJAX` this mode.

PS:

After WebMagic 0.5.0 support will be added to the chain of Json API, later you can use:

```
page.getJson().jsonPath("$.name").get();
```

In such a way to resolve AJAX requests.

Also supports

```
page.getJson().removePadding("callback").jsonPath("$.name").get();
```

JSONP such a way to resolve the request.