

---

# Table of Contents

Introduction	1.1
Огляд WebMagic	1.2
Філософія	1.2.1
Архітектура	1.2.2
Компоненти	1.2.3
Збирання	1.3
з використанням Maven	1.3.1
Без Maven	1.3.2
Перший проєкт	1.3.3
Білд з сирцевого коду	1.4
Отримати сирці	1.4.1
Імпорт проекту	1.4.2
Білд та запуск на прикладах	1.4.3
Основні функції у пошукового робота	1.5
Імплементація PageProcessor -а	1.5.1
Послідовність селекторності з API Selectable	1.5.2
Збереження результатів	1.5.3
Конфігурація пошукача, запуск і зупинка	1.5.4
Знайомство з інструментами вилучення - екстракцією: Jsoup та Xsoup	1.5.5
Монітор з JMX	1.5.6
Використання анотацій при написанні пошукача	1.6
Написання класу Model	1.6.1
TargetUrl та HelpUrl	1.6.2
@ExtractBy	1.6.3
Використання @ExtractBy у класци	1.6.4
Форматування результатів	1.6.5
Життєвий цикл	1.6.6
AfterExtractor	1.6.7

---

Налаштування компонентів	1.7
Кастомизація конвеєру Pipeline	1.7.1
Планувальник Scheduler	1.7.2
Завантажувач Downloader	1.7.3
Навчальні приклади	1.8
Основні комбінації сторінки та список деталей	1.8.1
Обробка пошукачем сторінок на JS рендерингу	1.8.2
Обробка пошукачем сторінки	1.8.3
Періодична обробка пошукачем	1.8.4
Асинхронне завантаження з аjax	1.8.5

# WebMagic у дії

Мала книга WebMagic.



[WebMagic](#) - простий але масштабуємий фреймворк-пошуковий робот (web crawler/spider). Ви легко можете розробити шукач на його основі.

Ця невелика книга розповідає про використання WebMagic. Вона також покаже деякі класичні випадки, як розробити шукач для конкретного сайту.

Цей документ зараз знаходиться у стані написання. Він переведений з оригінальної китайської документації <http://webmagic.io/docs/zh/>, і тому можуть бути деякі помилки. Дякую за [Відгуки до випуску](#) або відгалуження форку чи відправлення запиту pull request до мене.

Попередній перегляд документації: <http://webmagic.io/docs/ua/>(створений на базі [gitbook](#)).

License: [CC-BYNC](#).

# 1. Огляд WebMagic

WebMagic це простий пошуковий робот (краулер, веб-сайдер) розроблений на [Java](#).

WebMagic складається з двох частин: ядро core і розширення extension. Webmagic-ядро являє собою простий і з доброю модульною реалізацію пошукач та у поставці webmagic-розширення додано деякі зручні функцій для розробки пошукача.

Архітектура WebMagic-core заснована та описана в розділі [Scrapy](#). Це предоставляє простий, але гнучкий API. Ви можете написати шукач просто, якщо ви знайомі з Java.

Поставка Webmagic-extension - це деякі зручні функції, такі як написання пошукача тільки з POJO анотаціями. Є також деякі дефолтні реалізації компонентів.

Webmagic також містить деякі інші розширення і комплексний продукт "WebMagic-Avalon".

## 1.1 Філософія



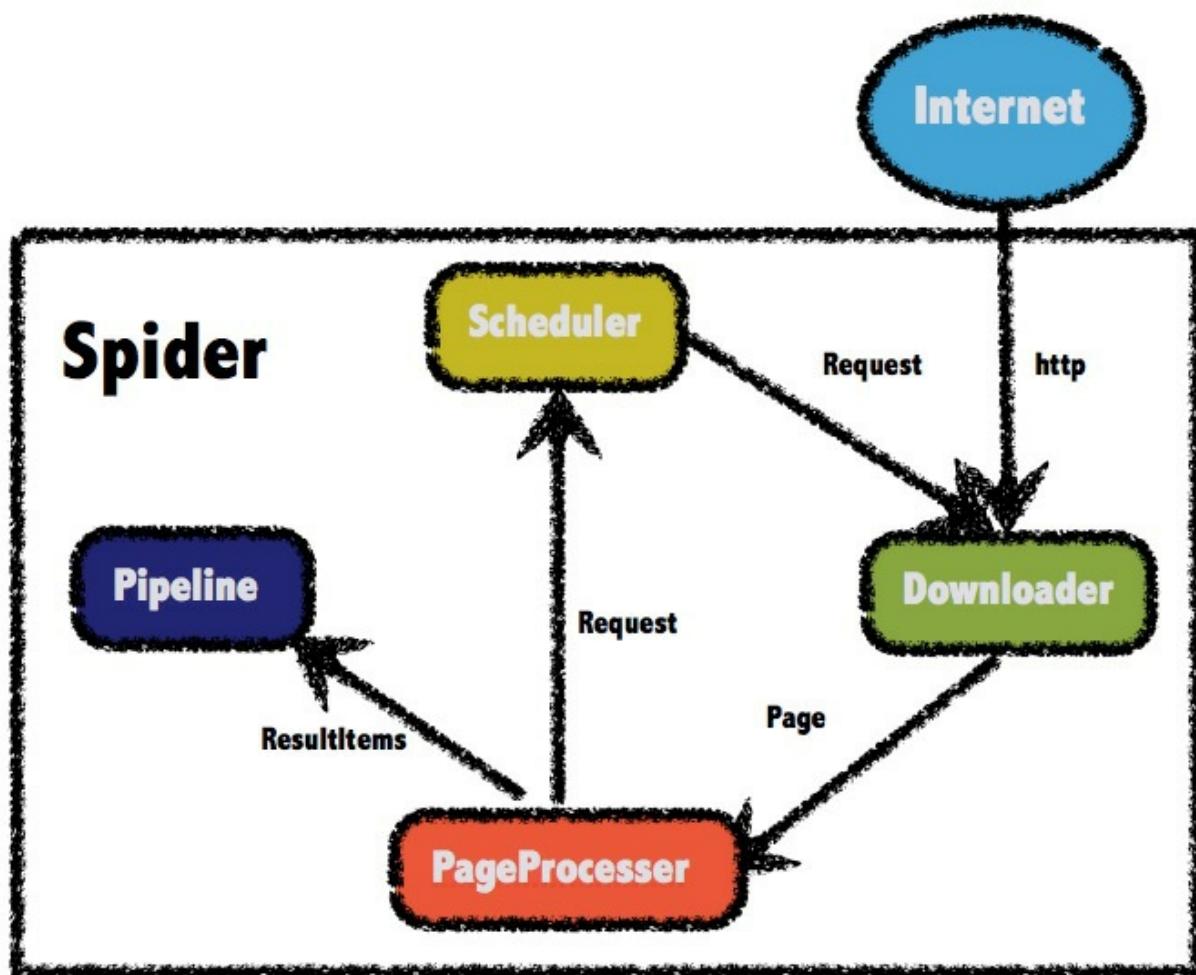
- 1. Рішення для Domain**
- 2. Mirco core з високою гнучкістю**
- 3. Зручний у використанні**

## 1.2 Загальна архітектура

Чотири компонента пошукача WebMagic структуровані в `Downloader`, `PageProcessor`, `Scheduler`, `Pipeline` контактиують поміж собою. Ці компоненти відповідають чотирьом здатністю життєвого циклу пошукача: завантаження, обробки, управління і здатність зберігання. Архітектура WebMagic подібна до `Scrapy`, але реалізована на `Java`.

Пошукач має кілька компонентів, але організованих таким чином, щоб вони могли взаємодіяти один з одним, процес реалізації можна розглядати пошукач, як великий контейнер з логіка у ядрі WebMagic.

Загальна архітектура WebMagic виглядає наступним чином:



### 1.2.1 чотири компонента WebMagic

#### 1. Downloader завантажувач

Downloader відповідає за завантаження з Інтернет-сторінки, для подальшої

обробки. У WebMagic за замовчуванням інструмент завантаження - [Apache HttpClient](#).

## 2. PageProcessor аналіз сторінки

PageProcessor відповідає за аналіз сторінки, витягувати корисну інформацію, а також відкриття нових посилань. WebMagic використовує [Jsoup](#) в якості синтаксичного аналізу інструментів HTML, на основі його розробки аналітичного інструменту XPath [Xsoup](#).

З цих чотирьох компонентів, тільки `PageProcessor` не той самий для кожної сторінки для кожного сайту, цю частину користувач повинен налаштовувати.

## 3. Scheduler планувальник

Менеджер планувальника Scheduler буде давати завдання сканувати URL по черзі. WebMagic за замовчуванням від JDK використовує менеджер черги URL-ів у пам'яті `memory queue management`, та може встановлювати пріоритети. Також підтримує використання розподіленого управління через Redis.

Якщо проект має деякі особливі потреби, що не бόли реалізовіні, тоді зможете налаштовувати свій власний планувальник Scheduler.

## 4. Pipeline конвеєрне збереження даних

Конвеєрна обробка даних Pipeline відповідає за прийняття результатів, включаючи перерахунки, зберігання в файли, у бази даних і так далі. WebMagic надається за замовчуванням "на консоль" і другий спосіб у програмі "Зберегти в файл" результат.

`Pipeline` визначає спосіб збереження результатів, якщо ви хочете зберегти у визначеній базі даних, тоді потрібно написати відповідну обробку даних. Як правило потрібно тільки написати `Pipeline`.

### 1.2.2 для об'єктів передачі даних

#### 1. Запит `Request`

`Request` - це пакетний рівень з URL-адресою, `Request` запит відповідний до URL-адреси.

Він є носієм для взаємодії `PageProcessor` та `Downloader`. Для `Downloader` це єдиний спосіб впливати на `PageProcessor`.

На додаток до самого URL, він містить поля зі структурою ключ-значення `extra`. Ви можете зберегти деякі додаткові спеціальні `extra` атрибути, а потім прочитати в інших місцях для виконання різних функцій. Наприклад, деяка додаткова інформація на одній сторінці, що використана при роботі в іншій, і так далі.

## 2. Сторінка Page

`Page` представлення сторінки `Downloader` для її завантаження - зміст може бути HTML, чи він може бути JSON або в інших текстових форматах.

Процес екстракції сторінки `Page` прописаний у ядрі WebMagic, який забезпечує способи видобутку та зберегання результатів, і так далі. Наразі у [розділі 4](#) ми будемо детально її використовувати.

## 3. Результатуюча одиниця ResultItems

`ResultItems` еквівалентно до колекції - мапи `Map` у `Java`, яка проводить обробку результатів `PageProcessor` для подальшого використання у конвеєрному збереженню результатів `Pipeline`. API у нього та у мапи `Map` дуже схожі, окрім поля `skip` яке при встановлені `true` означає, що не повинен бути оброблений `Pipeline` конвеєром збереження результатів.

### 1.2.3 Керування запуском движка пошукача --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](#).

Павук Spider є внутрішнім процесом ядра WebMagic. Павук `Spider` має властивості `Downloader`, `PageProcessor`, `Pipeline`, що можуть бути встановлені `setter`-ами у різні функції. Павук `Spider` WebMagic - це товка вхіду, яка інкапсулює створення пошукових роботів, запуск, зупинку, можливості роботи у багатопоточному режимі. Ось набір кожного компонента, і наведений приклад запуску у багатопоточності.

Детальніше про налаштування павука у частині 4 - [Конфігурація пошукача, запуск і зупинка](#).

```
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 Швидкий старт

Склалося враження, що приведена велика кількість компонентів, але насправді не турбуйтесь - бо старту їх потрібно не так багато конфігурувати, тому що велика частина модулів WebMagic вже забезпечує реалізацію за замовчуванням.

Загалом, для отримання пошукача необхідності написати класс з імплементацією `PageProcessor`, де `create`-том створюється `Spider` із початковими даними для сканеру. У розділі 4 ми розповімо як писати налаштовання шукача `PageProcessor` і `Spider` щоб розпочати - [Імплементація PageProcessor](#) -а.

## 1.3 Компоненти проекту

WebMagic код проекту складається з декількох частин, в кореневому каталозі розподілені за різним ім'ям каталогу. Вони не залежать від Maven проекту.

### 1.3.1 Основна частина

WebMagic включає в себе два пакети, обидва пакети із великим накопиченим практичним досвідом та досить зрілі:

#### Webmagic-core ядро

`Webmagic-core` є основною частиною WebMagic, містить тільки основні модулі та основний пошуковий екстрактор. WebMagic-core мета полягає в тому, щоб стати базою основою для реалізації пошукача.

#### Webmagic-extension розширення

`Webmagic-extension` є у WebMagic основним модулем розширення, який забезпечує деякі з більш зручний інструмент написання пошукача. У тому числі визначення формату для сканеру прямо в анотаціях, JSON, розподілена робота та інші заходи підтримки.

### 1.3.2 Периферійні функції

Крім того, до проекту WebMagic додано декілька пакетах із експериментальними можливостями. Мета полягає в тому, щоб надати змогу ознайомитись із попередніми зразками та для інтеграції. Через обмежені можливості, ці пакети не були широко використані і випробувані, рекомендується завантажити вихідний код, а потім вносити змінити при стиканні з проблемами та багами.

#### Webmagic-зразки

Ось деякі приклади пошукача написані автором раніше. Через брак часу, деякі з цих прикладів використовують до сих пір зі старою версією API, чи тому, що можуть бути деякі зміни в структурі цільової сторінки і приклад втратив актуальність. На сьогоднішній день, приведені приклади - `us.codecraft.webmagic.processor.example` та для пакету `webmagic-core` `webmagic-core package of us.codecraft.webmagic.example`.

## Webmagic-scripts скрипти

WebMagic для написання сценарію правил для пошукача є кілька варіантів.

Основна мета, щоб розробникам на мові Java дозволити просто та швидкого розробляти. Наведено загальний сценарій.

В даний час проект заморожено, так як користувачів не так сильно це цікавить, ви можете подивитися на сценарному, якщо маєте інтерес, ось тут: [webmagic-scripts simple document](#).

## Webmagic-selenium

WebMagic і `Selenium` в поєднанні модулів. `Selenium` є аналогом інструмента для рендеринга сторінки браузера, у WebMagic передаються на сканування динамічні сторінки з `Selenium`.

## Webmagic-saxon

WebMagic і Saxon в поєднанні модулів. Saxon це XPath, XSLT аналітичні інструменти, webmagic покладаються на Saxon підтримку розбору за правилами XPath2.0.

### 1.3.3 webmagic-avalon Авалон

`Webmagic-avalon` це спеціальний проект, яким хочемо досягти до продукту, що заснований на інструменті WebMagic, який охоплює створення сканерів, пошукачів і інші інструменти управління бекенд. [Avalon \(uk\)](#) чи [Avalon \(ch\)](#) Артура легенда "ідеальний острів", `webmagic-Авалон` мета полягає в тому, щоб забезпечити загальну пошукача засобів досягнення цієї мети не так просто, так що у назві також мається про трохи «ідеальні», але автор прагне до досягнення цієї мети.

Ви можете подивитися, якщо зацікавлені в цьому проекті, тут [проект WebMagic-Avalon](#).

## 2. Використовуйте WebMagic

Основною особливістю WebMagic вимагає дві банки: `webmagic-версії {core-} .jar` і `webmagic-термопарнє {версія} .jar`. Просто додайте їх в дорозі до класів, і ви добре йти.

WebMagic використовує Maven для управління своїми залежностями, але причини ви можете використовувати його без Maven.

## 2.1 Використання Maven для управління залежностями

WebMagic побудована на Maven, тому настійно рекомендовано використання Maven для управління проектом. Ви можете додати WebMagic шляхом додавання наступних рядків у вашому файлі 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>
```

Якщо ви не знайомі з Maven, Ось введення: [Що таке Maven?](#)

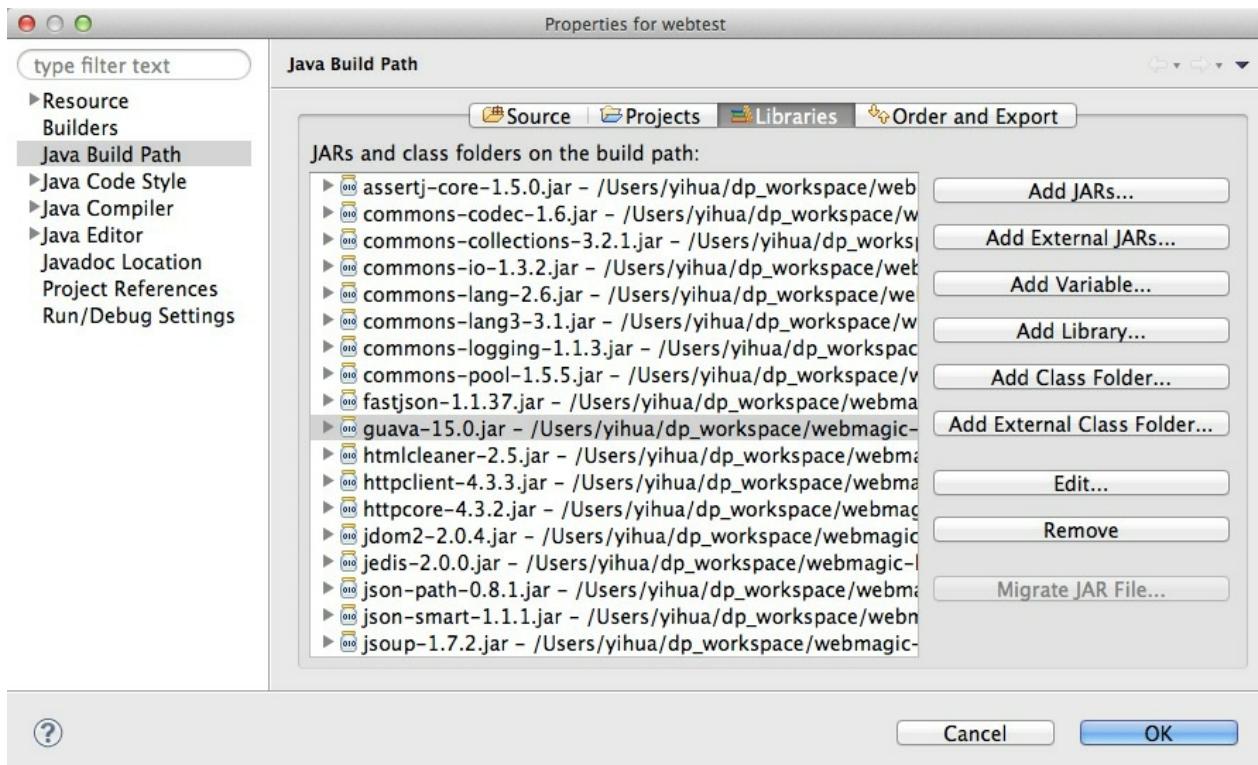
WebMagic використовує SLF4J-log4j12 як реалізація SLF4J. Якщо у вас є свій власний вибір SLF4J РЕАЛІЗАЦІЇ, виключити форсування у ваші залежності.

```
<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>
```

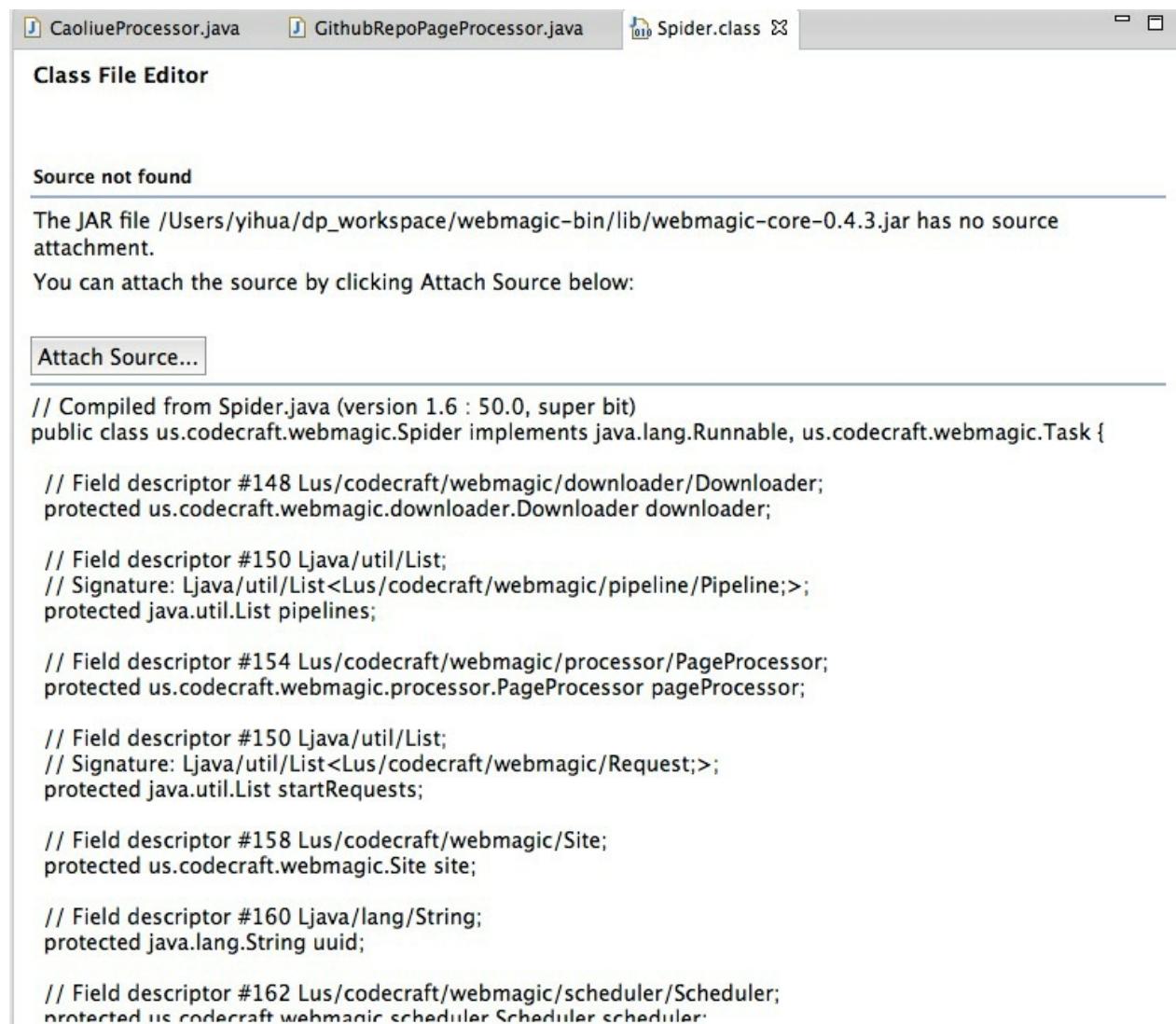
## 2.2 Без Maven

Якщо ви не знаєте, Maven або не подобається він, ви можете завантажити найновіші сирці з <http://webmagic.io>. Я також переглянути всі сирці та залежності webmagic, натисніть [тут](#) для завантаження.

Після завантаження, просто розпакувати архів і додати у свій classpath шляхи до jar файла.



Оскільки у WebMagic можна змінити настройки, що дійсно корисно, щоб побачити вихідний код. Ви можете отримати оновлення `webmagic-core-{version}-sources.jar` та `webmagic-extension-{version}-sources.jar` на <http://webmagic.io>, просто натисніть на "Attach Source".



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;
    protected us.codecraft.webmagic.scheduler.Scheduler scheduler;
}
```

## 2.3 Перший проект

Після додавання залежностей Maven для вашого проекту, ми готові написати наш найперший шукач. Тут ми беремо Github, наприклад:

```
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 {

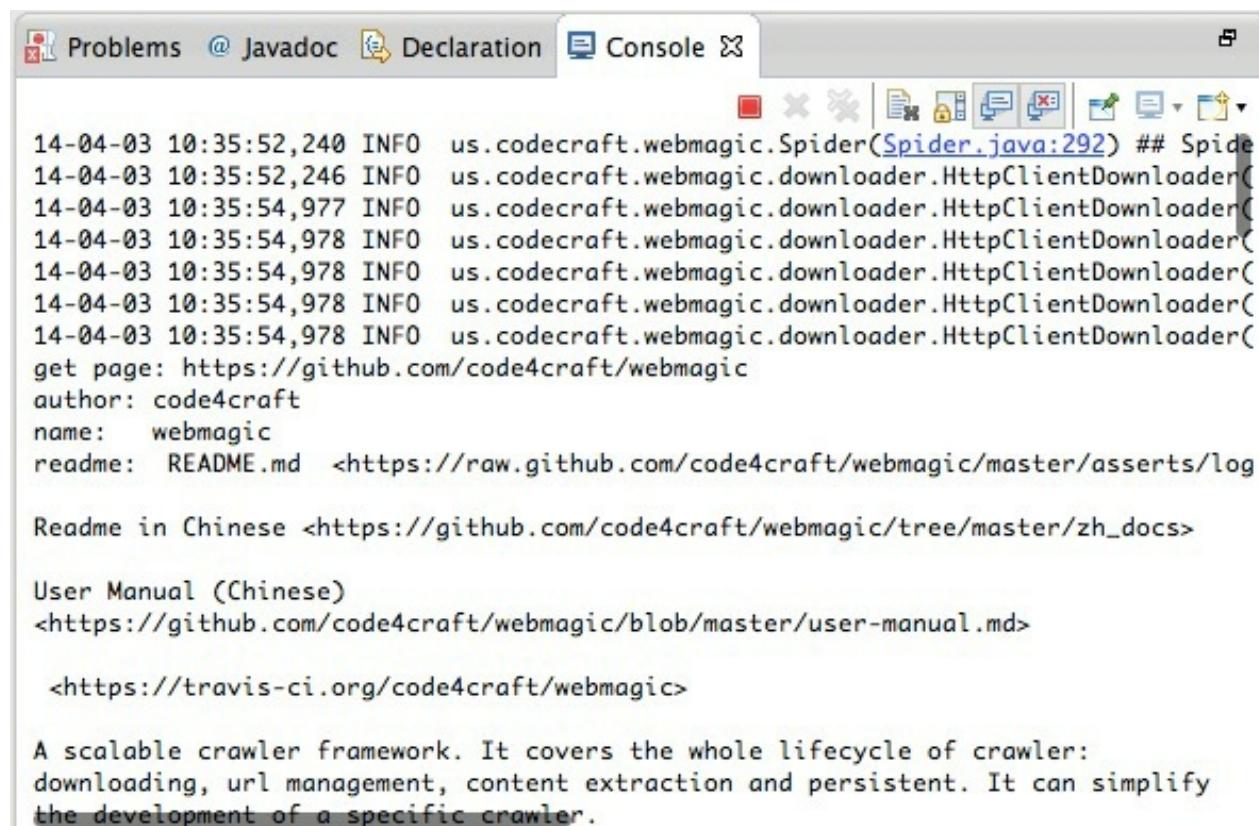
    private Site site = Site.me().setRetryTimes(3).setSleepTime(100);

    @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+)/.*").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()"));
    }

    @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();
    }
}
```

Тепер стартуємо run() у main()... і ось - voilà.



The screenshot shows a Java IDE interface with the 'Console' tab selected. The console window displays the following log output:

```
14-04-03 10:35:52,240 INFO us.codecraft.webmagic.Spider(Spider.java:292) ## Spider
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/assets/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. Завантажте та компілювати вихідний код**

Якщо ви зацікавлені в джерелі WebMagic, то ви можете завантажити вихідний код і компілювати спосіб використовувати WebMagic. "Дуже просто вторинне розвиток" також є однією з цілей WebMagic.

WebMagic це чистий проект Java, якщо ви знайомі з Maven, а потім завантажити та компілювати вихідний код дуже простий. Якщо ви не знайомі з Maven не має значення, в цьому розділі наведено відомості як імпортувати цей проект в Eclipse.

### 3.1 Отримати сирцевий код

WebMagic в даний час зберігається у двох сховищах:

- <https://github.com/code4craft/webmagic> Github репозиторій має останню версію коду та всі завдання issue, та запит на pull request - всі тут. Якщо вважаєте що проект хороший, то не забудьте додати зірочку!
- <http://git.oschina.net/flashsword20/webmagic>

Це сховище містить всі скомпільовані залежності, зберегаються тільки стабільні версії проекту, остання версія для оновленні знаходитьться на GitHub. Oschina відносно стабільним доступом в країні, і служить в основному в якості дзеркала.

Завантажити з репозіторія можна командою git термінала

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

або

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

Ви завантажите останню версію коду.

Якщо ви не знайомі з використанням самого git, ми рекомендуємо подивитися [@Хуан Юн Завантажити Smart Source з Git OSC](#) чи з [wikipedia](#)

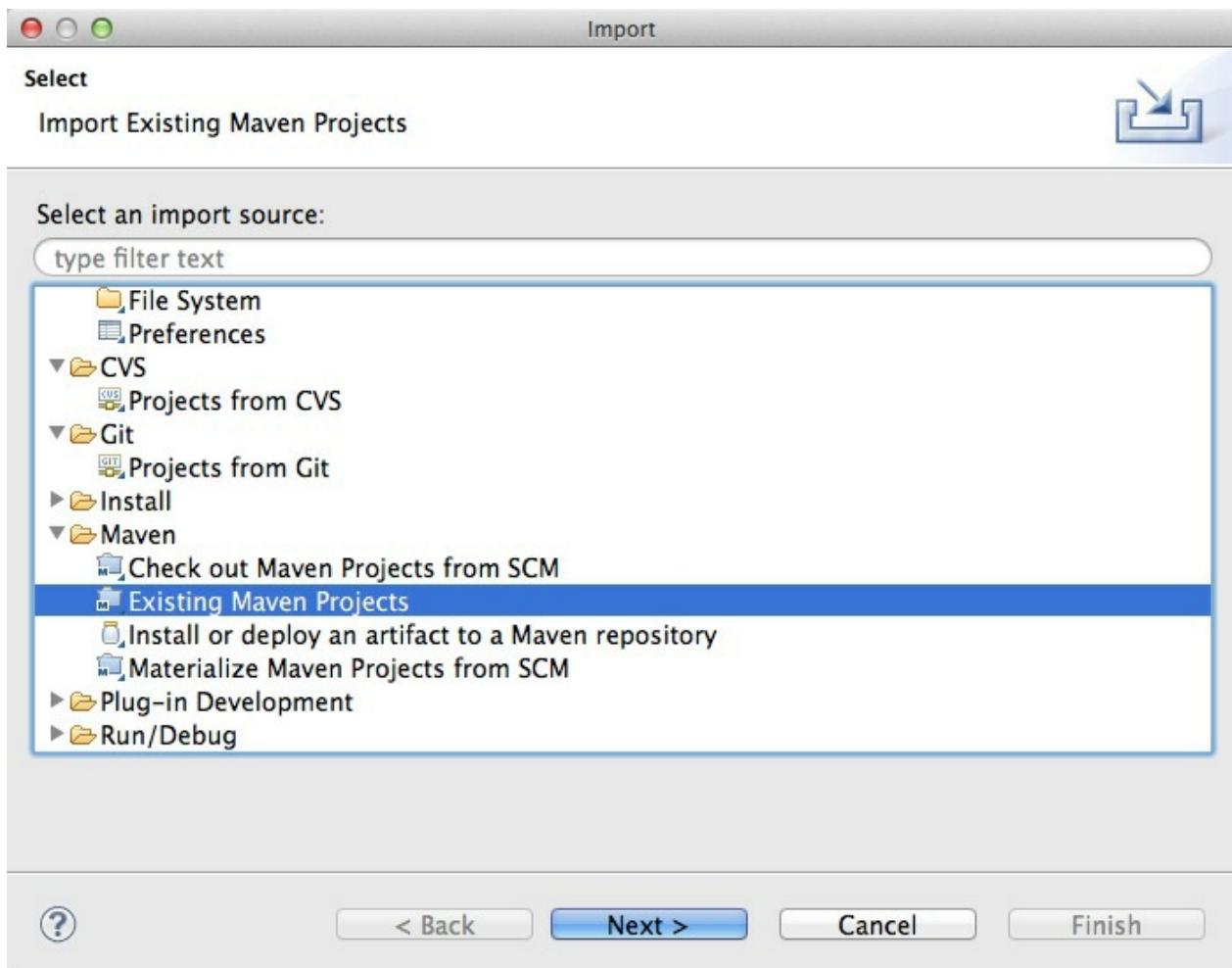
## 3.2 Імпорт проекту

IntelliJ IDEA за замовчуванням поставляється з підтримкою Maven, вибрать елементи, які ви можете імпортувати у Maven проекти.

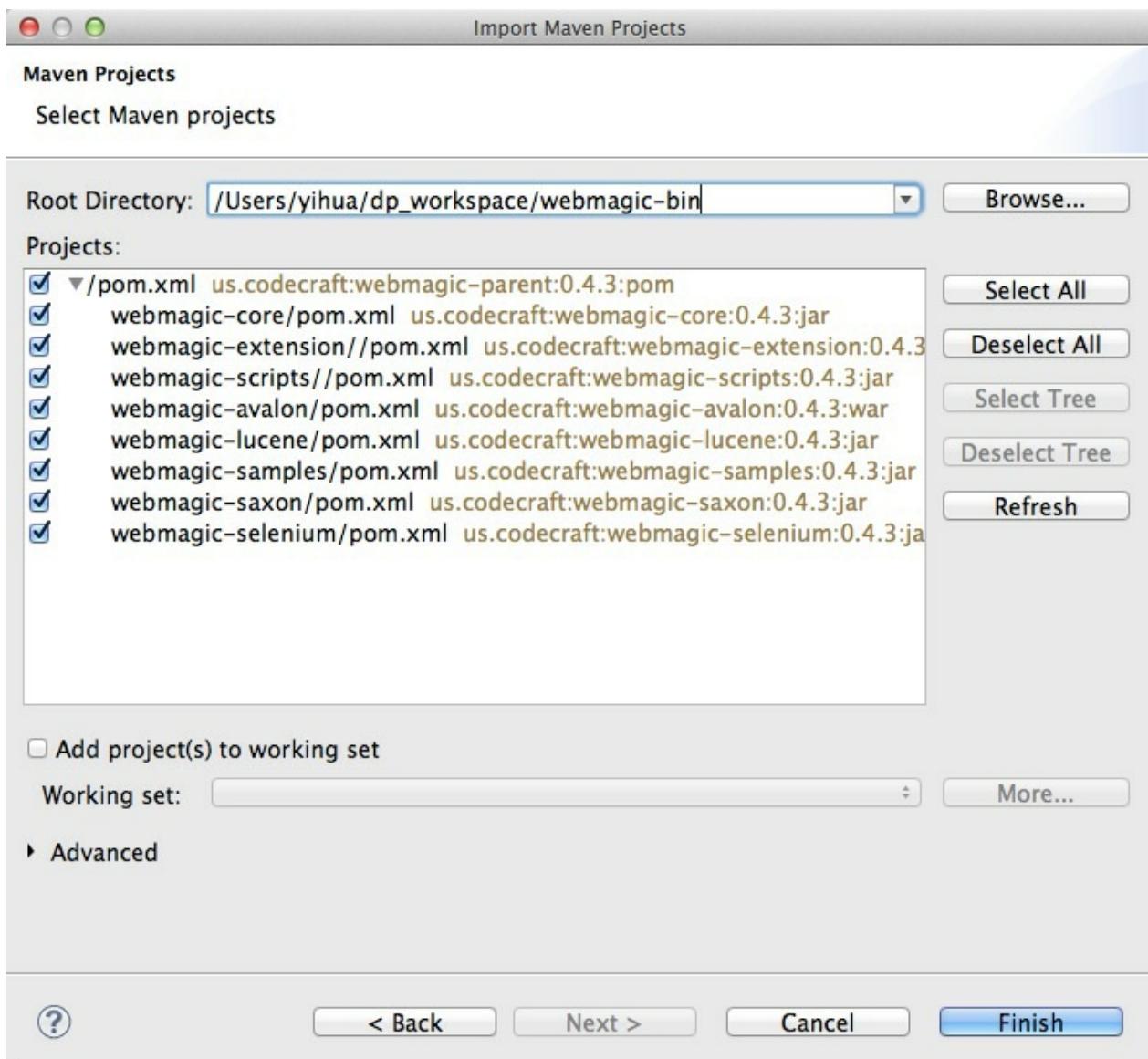
### 3.2.1 m2e плагін

Користувачам Eclipse рекомендується встановити m2e plug-in плагін за адресою установки: <https://www.eclipse.org/m2e/download/>

Після установки, виберіть File-> Import ... і виберіть Існуючі проекти Maven Maven->Existing Maven Projects і натисніть кнопку Далі Next, можна папку для імпорту в проект.



Після імпорту проекту, щоб побачити екран вибору, ви можете натиснути кнопку Готово finish.



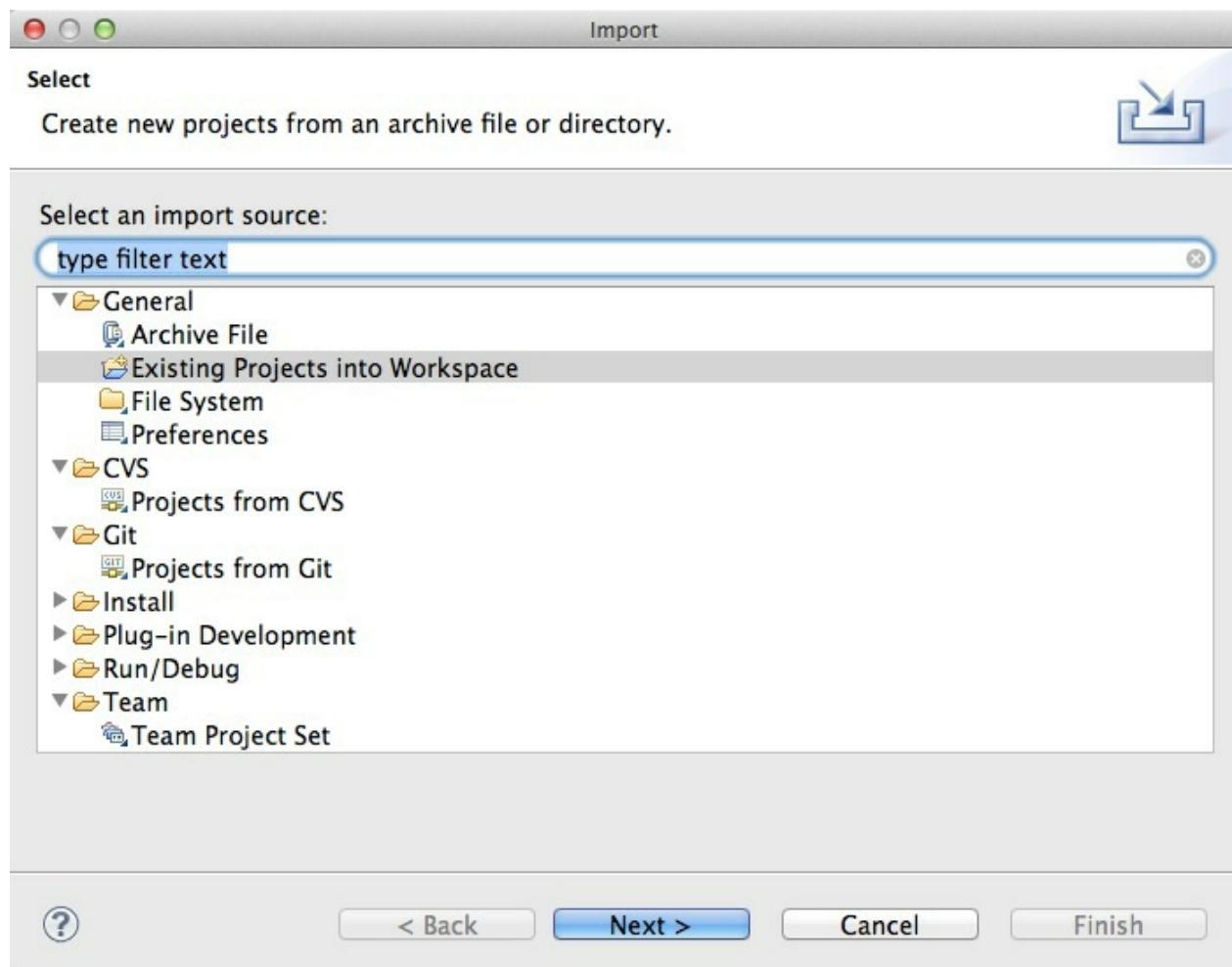
### 3.2.2 Використання Maven плагін для Eclipse

Якщо m2e плагін `m2e plug-in` не встановлено, але ви можете встановити Maven, він відносно простий у використанні. Команда в кореневій директорії проекту:

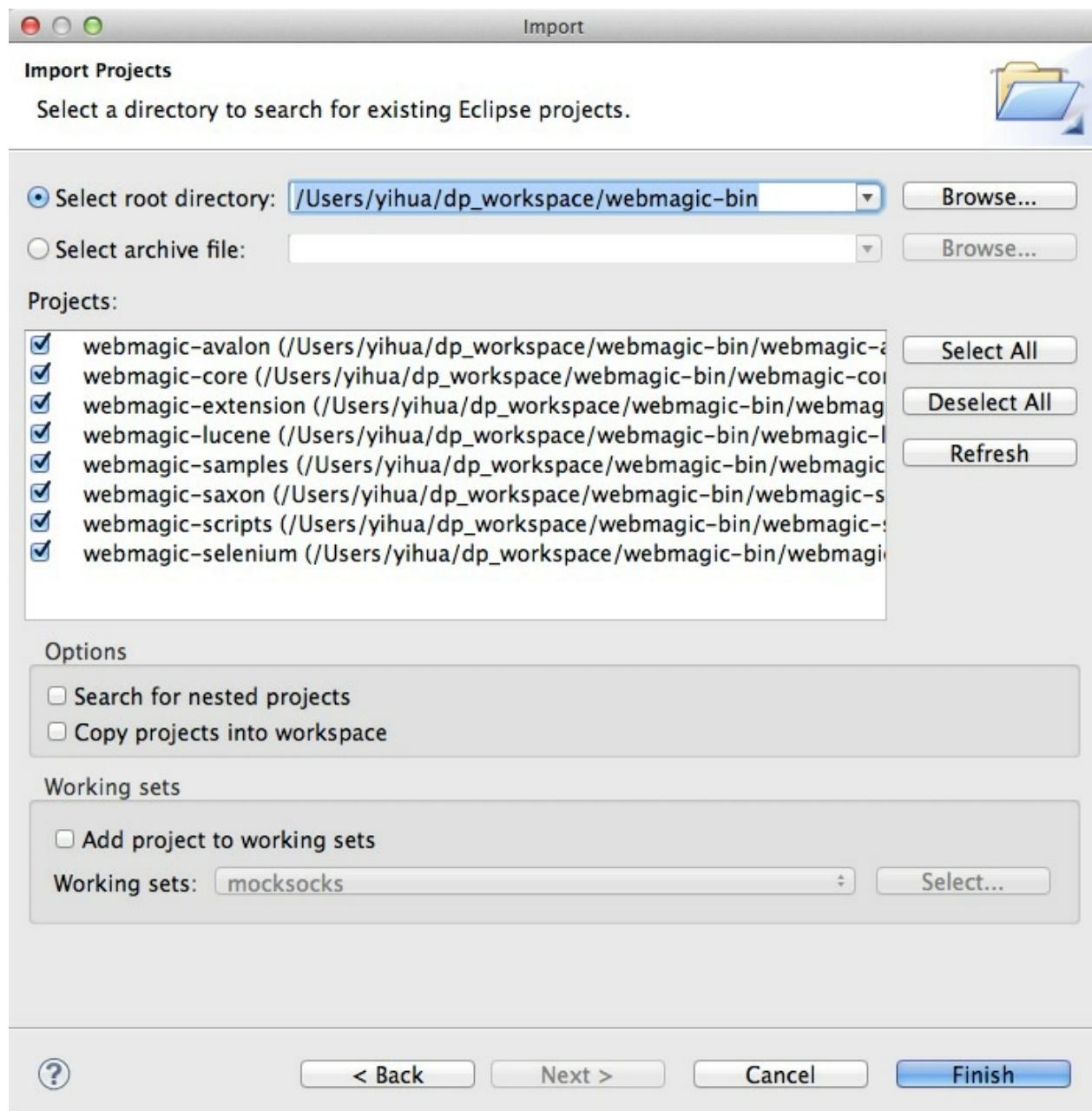
```
mvn eclipse:eclipse
```

Сгенеробється файл конфігурації структури Maven для проекту Eclipse, а потім виберіть `File->Import and open General->Existing Projects into Workspace` для імпорту підготовленого проекту.

## Імпорт проекту



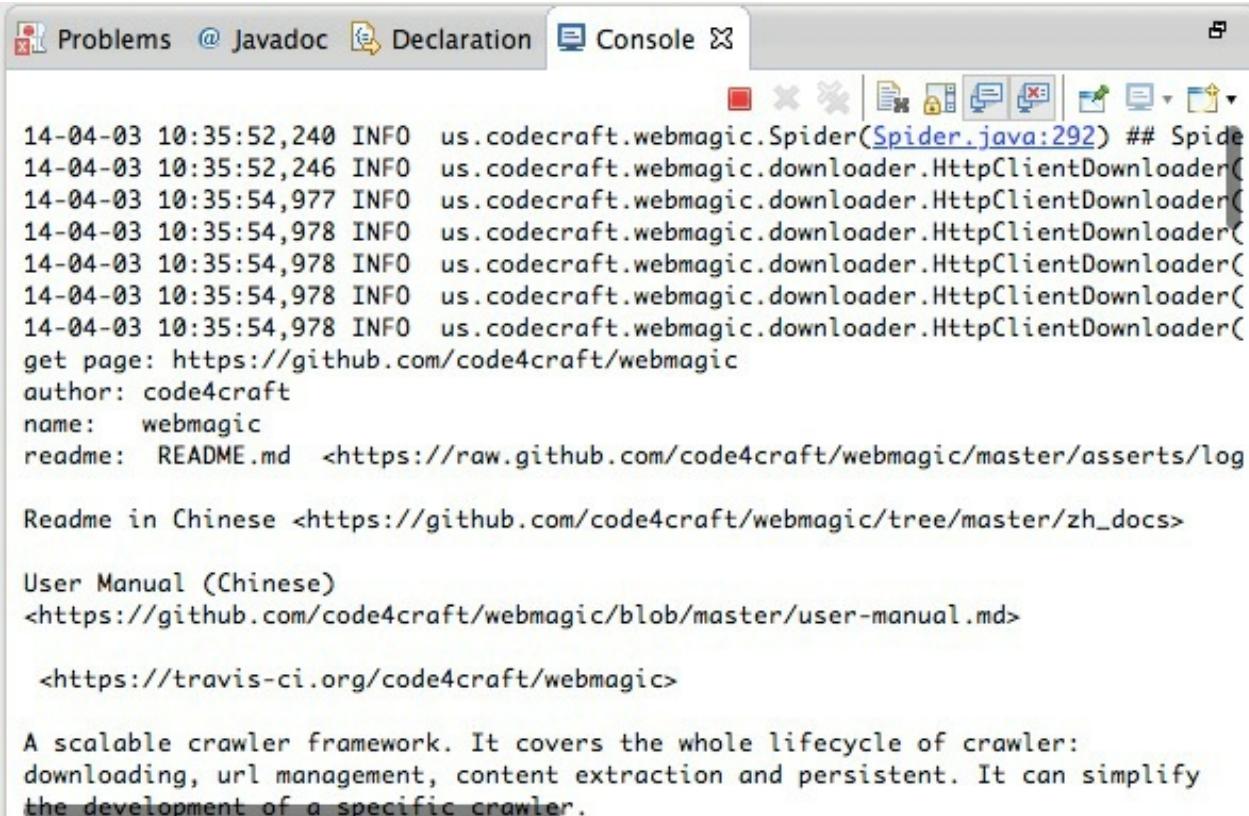
Після імпорту проекту, побачете екран вибору, та можете натиснути кнопку Готово finish.



### 3.3 Компіляція коду з сирців і запуск

Як імпорт пройшов успішно то не повинно бути ніякої помилки компіляції! І вже на цьому етапі ви можете запустити webmagic-core проект, включений в пакет прикладів: "us.codecraft.webmagic.processor.example.GithubRepoPageProcessor".

Крім того, подивіться вивід в консоль - при компіляції коду, що пройшла успіху виглядає наступним чином:



The screenshot shows an IDE interface with several tabs at the top: Problems, @ Javadoc, Declaration, and Console. The Console tab is active, displaying the following log output:

```
14-04-03 10:35:52,240 INFO us.codecraft.webmagic.Spider(Spider.java:292) ## Spider
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.
```

## 4. Основні функції пошукача

Для основної реалізація пошукача у WebMagic - необхідно лише імплементувати (написати клас, який реалізує інтерфейс) `PageProcessor` . В основному цей клас містить увесь код, що вам потрібно написати для обходу сайту.

Також в цьому розділі описується використання API екстракції WebMagic, що є найбільш поширеною проблемою пошукача - зберігання результатів.

## 4.1 Імплементація PageProcessor

У наведеному прикладі - клас `GithubRepoPageProcessor`, імплементує `PageProcessor`. Будемо настроювати `PageProcessor` за допомогою `PageProcessor`, що ділиться на три частини: конфігурація пошукача, екстракція елементів сторінки і виявлення посилань (лінків).

```
public class GithubRepoPageProcessor implements PageProcessor {

    // Part I: crawl the site configuration, including coding, crawler space, retries, etc.
    /// Частина 1: пошукачу вказують сайт, включаючи кодування, простір для пошуку, повторні спроби і т.д.
    private Site site = Site.me().setRetryTimes(3).setSleepTime(1000);

    @Override
    // Process custom crawler logic core interfaces, where the preparation of extraction logic
    /// Процес призначення пошукачу логіки основних інтерфейсів, де визначаються правила екстракції даних
    public void process(Page page) {
        // Part II: the definition of how to extract information about the page, and preserved
        /// Частина 2: визначення правил екстракції інформацію зі сторінки і в якому полі классу зберігається
        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
            /// пропустити цю сторінку
            page.setSkip(true);
        }
        page.putField("readme", page.getHtml().xpath("//div[@id='readme']/tidyText()"));

        // Part III: From the subsequent discovery page url address to crawler
        /// Частина 3: Із результатів попередніх обробок вишукуємо URL адреси інших сторінок для передачі пошукачеві на обробку
        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
    /// URL адреса "https://github.com/code4craft" з якої починається пошук
    .addUrl("https://github.com/code4craft")
    //Open 5 threads of Crawler
    /// Відкрвати 5 паралельних потоків пошукача
    .thread(5)
    //Start Crawler
    /// Запуск в роботу пошукача
    .run();
}
}

```

## 4.1.1 Конфігурація пошукача

Перша частина конфігурації сканеру пошукача, включають в себе: кодування, інтервал сканування, час очікування, число повторних спроб, а також включає в себе деякі параметри браузера: User Agent, cookie і налаштування проксі-сервера. Докладніше у [4.5 Знайомство з інструментами вилучення - екстракцією: Jsoup та Xsoup](#). Якщо коротенько, то тут у прикладі встановлено: число повторних спроб - до 3 разів, та сканувати з інтервалом в 1 секунду.

## 4.1.2 Екстракція елементів сторінок

Друга частина - ядро пошукового сканеру: витягти інформацію (екстракція) з HTML-сторінки, що щойно завантажена. WebMagic використовує в основному три технології екстракції: XPath, регулярні вирази (regular expressions) і CSS селектори. Окрім того, для ресурсів де інформація у форматі JSON, ви можете використовувати JsonPath.

### 1. XPath

Започатковано [XPath](#), як мова запитів XML елементів, але згодом також поширився та став зручним і для HTML, наприклад:

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

Цей код використовує XPath, та означає: "знати всі атрибути класу @class 'entry-title public' для тегів (елемент) `<h1>` і знати дочірній тег-узол (ноду node) `<strong>`, у якого наявний тег `<a>` та з нього витягнути текстову інформацію." Відповідний до нього HTML-код наприклад наступний:

```
▼-----  
▼<h1 itemscope itemtype="http://data-vocabulary.org/Breadcrumb" class="entry-title public">  
► <span class="repo-label">...</span>  
► <span class="mega-octicon octicon-repo">...</span>  
► <span class="author">...</span>  
  <span class="repohead-name-divider">/</span>  
▼ <strong>  
  <a href="/code4craft/webmagic" class="js-current-repository js-repo-home-link">webmagic</a>  
  </strong>  
► <span class="page-context-loader">...</span>  
</h1>
```

## 2. CSS селектор

CSS і XPath селектори схожі за синтаксисом мови. Для front-end розробки знайоме формулювання `$(‘h1.entry-title’)`, що відповідне до попереднього прикладу. Об'єктивно кажучи, його простіше писати, ніж XPath, але якщо ви пишете більш складні правила екстракції, то це відносно невеликі проблеми.

## 3. Регулярні вирази regular Expressions

Регулярні вирази ([див. wikipedia](#)) є універсальною мовою для екстракції тексту.

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

Цей код використовує регулярний вираз для екстракції усіх лінків з [“https://github.com/code4craft/webmagic”](https://github.com/code4craft/webmagic).

## 4. JsonPath

JsonPath це мова дуже схожа у використанні на XPath, але для швидкого пошуку у JSON контенті. Докладніше про використаний у WebMagic формат JsonPath можна знайти тут: <https://code.google.com/p/json-path/>

### 4.1.3 Пошук лінків

Після обробки сторінки нашим сканером по всім виразам з логікою, краулер закриється!

Але тепер виникає проблема: сторінок сайту багато, і з самого початку ми можемо не знати та перерахувати всі. Але в отсканованих сторінках можна пошукати ще і додаткові линки. Таким чином можна додати їх до списку черги лінків для сканування. Цей функціонал також є невід'ємною частиною пошукача.

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

Цей код складається з двох частин:

```
* регулярний вираз посилань на сайт `page.getHtml().links().regex("(https://github\\.com\n/\\w+/\\w+)").all()` - тобто отримати всі лінки з "(https://github\\.com/\\w+/\\w+)" - це за шаблоном https://github.com/ОднаЧиБільшеБукв/ОднаЧиБільшеБукв,  
* `page.addTargetRequests()` - виклик метода - усі ці знайдені за шаблоном посилання будуть додані в чергу для подальшого пошукового сканування.
```

## 4.2 Послідовність селекторності з API Selectable

Вибір релевантної послідовності API Selectable для селекторів - це основна функція WebMagic. Для використання інтерфейсу селекторності (селективного вибору для екстракції) можна використати пряму послідовність доступу до елементів сторінки та потрібно подбати про уточнюючи деталі.

З попереднього прикладу можна побачити, що `page.getHtml()` повертає об'єкт `html`, який імплементує інтерфейс `Selectable`. Цей інтерфейс містить декілька важливих частин. Я розділив його на дві категорії: розтин на частини для екстракції і повернення отриманих результатів.

### Селектори для екстракції (починаючи з API v.4.2.1):

метод	опис	приклади
<code>xpath(String xpath)</code>	використання XPath селектора	<code>html.xpath("//div[@class='title']")</code>
<code>\$(String selector)</code>	використання CSS селектора для вибору	<code>html.\$("div.title")</code>
<code>\$(String selector, String attr)</code>	використання CSS селектора для вибору	<code>html.\$("div.title", "text")</code>
<code>css(String selector)</code>	аналогічно функції з <code>\$()</code> , використання CSS селектора для вибору	<code>html.css("div.title")</code>
<code>links()</code>	вибрати усі лінки	<code>html.links()</code>
<code>regex(String regex)</code>	використати регулярний вираз для екстракції	<code>html.regex("\.(.*?)")</code>
<code>regex(String regex, int group)</code>	використати регулярний вираз для екстракції та вказати порядковий номер групи для обробки	<code>html.regex("\.(.*?)", 1)</code>
<code>replace(String regex, String replacement)</code>	замінити вміст за <code>regex</code> на <code>replacement</code>	<code>html.replace("\", "")</code>

Це є частиною процесу екстракції, відпрацьовує та повертає результат через

інтерфейс `API Selectable`, що означає - послідовність безперервної черги викликів для екстракції. Приклад використання послідовності API.

Припустимо зараз я хочу охопити та зібрати (екстрактувати) всі проекти `Java` на GitHub, ці елементи можна побачити <https://github.com/search?l=Java&p=1&q=stars%3A%3E1&s=stars&type=Repositories>

`l=Java&p=1&q=stars%3A%3E1&s=stars&type=Repositories` - це наведені результати пошуку.

Щоб уникнути занадто широкого сканування, я вказав обмеження - сканувати лише посилання з секції закладок сторінок (пейджинатора). Напевно писати правила обходу усього контенту складніше булоб?

Last updated a month ago

**loopj/android-async-http**  
An Asynchronous HTTP Library for Android  
Last updated 6 days ago

**spring-projects/spring-framework**  
The Spring Framework  
Last updated 5 hours ago

**JakeWharton/Android-ViewPagerIndicator**  
Paging indicator widgets compatible with the ViewPager from the Android Support Library and ActionBarSherlock.  
Originally based on Patrik Åkerfeldt's ViewFlow.  
Last updated a year ago

**clojure/clojure**  
The Clojure programming language  
Last updated 3 days ago

How are these search results? Tell us!

Спочатку розглянемо типову структуру HTML сторінки:

```
<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>
```

Видно, що можна використовувати CSS селектори, щоб екстрактувати (витягти) цей

тег `<div>`, а потім там взяти всі посилання. Щоб перестрахуватися застосуємо відбір регулярним виразом. Таким чином формат витягнутого - це URL за нашим шаблоном. У підсумку остаточне формулювання виглядає так:

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

Тоді ми можемо додати ці URL-и до списку для сканування:

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

Це ж справді не складно? На додаток для пошуку посилань-лінків, використання API Selectable послідовності селекторної екстракції - також може зробити багато роботи. У розділі 9 **TODO** ми знову будемо розбирати наведені вище приклади.

## 4.2.2 Get Results API для отримання результатів:

Коли викликаємо послідовності, ми звичайно очікуємо отримати у результаті тип `String`. За для цього використовуємо **Get Results API**. Ми вже знайомі з правилами екстракції `xPath` та `css` селектори або регулярний вираз `regex`, за якими завжди можна витягти декілька елементів. У WebMagic вони були уніфіковані, тому ви можете отримати один або кілька елементів за допомогою різних API.

метод	опис	приклади
<code>get()</code>	повертає результат типу <code>String</code>	<code>String link= html.links().get()</code>
<code>toString()</code>	функція <code>get()</code> , повертає результат типу <code>String</code>	<code>String link= html.links().toString()</code>
<code>all()</code>	повертає всі результати вилучення	<code>List links= html.links().all()</code>
<code>match()</code>	якщо результат співпадає	<code>if (html.links().match()){ xxx; }</code>

Наприклад, у випадках коли ми точно знаємо, що сторінка буде мати результат, тоді можна використовувати `selectable.get()` або `selectable.toString()` для отримання цього результату.

У цьому випадку `selectable.toString()` використовує `toString()` цього інтерфейсу,

це добре для системи виведення та фреймворків при інших комбінаціях, більш зручними. Тому що при нормальних обставинах, нам потрібно тільки вибрати один елемент!

`selectable.all()` - у результаті отримаєте всі елементів.

Ну а тепер, якщо озирнутися назад на `GithubRepoPageProcessor` у Частину [4.1 Імплементація PageProcessor](#), можливо ви вже відчуватимете себе більш ясно, чи не так? Переписати специфікацію `main()` метода, і тоді вже можна побачити оновлені результати роботи пошукача у виводу в консолі.

## 4.3 Збереження результатів

Ну добре, сканери пошукача написано, і тепер стає питання: Як зібрани результи зберігти у будь-яке сховище?

Компонент WebMagic для зберігання результатів називається інфопровод

Pipeline .

Наприклад, ми вирішили це виводити в консолі через вбудований інфопровод

Pipeline - питання вирішує ConsolePipeline - компонент.

А якщо тепер я хочу, щоб зберегти результати пошукача у форматі JSON, як це зробити? Мені просто потрібно замінити Pipeline - на JsonFilePipeline .

```
public static void main(String[] args) {
    Spider.create(new GithubRepoPageProcessor())
        // From "https://github.com/code4craft" began to grasp
        /// Починати роботу пошукачем зі сторінки "https://github.com/code4craft"
        .addUrl("https://github.com/code4craft")
        /// Зберегати результати пошукача у форматі JSON
        .addPipeline(new JsonFilePipeline("D:\\webmagic\\"))
        // Open 5 threads of Crawl
        /// Дозволити 5 потоків пошукача
        .thread(5)
        // Start Crawl
        /// Запускаємо пошукач
        .run();
}
```

У прикладі - ця загрузка буде збережена на диску D: у каталозі webmagic.

Змінюючи інфопровод, можна домогтися збереження результатів в файл, базу даних та інше. Детальніше будемо ознайомитись у Частині [6.1 Налаштування інфопроводу Pipeline](#).

Нарешті ми завершили базову підготовку пошукача, хоча ще є декілька функцій настройки.

## 4.4 Конфігурація пошукача, запуск і зупинка

### 4.4.1 Пошукач Spider

`Spider` початок запуску пошукача.

Перед початком скануванням пошукачем, нам потрібно за допомогою `PageProcessor` створити об'єкт павука `Spider`, а тоді запустити його методом `run()`.

На відміну інші компоненти веб-павука `Spider` (завантажувач `Downloader`, планувальник `Scheduler`, інфопровід `Pipeline`) можуть бути установлені методами `set`.

метод	опис	приклади
<code>create(PageProcessor)</code>	створити <code>Spider</code>	<code>spider.create(new GithubRe...</code>
<code>addUrl(String...)</code>	додати URL ініціалізації	<code>spider.addUrl("http://webmag...</code>
<code>addRequest(Request...)</code>	додати Request ініціалізації	<code>spider.addRequest("http://we...</code>
<code>thread(n)</code>	к-ть потоків n	<code>spider.thread(5)</code>
<code>run()</code>	запуск, розблокування поточного потоку його визовом	<code>spider.run()</code>
<code>start()/runAsync()</code>	асинхронний старт, продовження з поточного потоку thread	<code>spider.start()</code>
<code>stop()</code>	стоп пошукача	<code>spider.stop()</code>
<code>test(String)</code>	тестова сторінка для пошукача	<code>spider.test("http://webmagic...</code>
<code>addPipeline(Pipeline)</code>	додати інфопровід <code>Pipeline</code> , у <code>Spider</code> може бути кілька	<code>spider.addPipeline(new Con...</code>

	Pipeline	
setScheduler(Scheduler)	установки Планувальника Scheduler , Spider ПОВИНЕН МАТИ Scheduler	spider.setScheduler(new Rec
setDownloader(Downloader)	установки Завантажувача Downloader , Spider ПОВИНЕН бути МАТИ Downloader	spider.setDownloader(new SeleniumDownloader())
get(String)	синхронні вивози та прямий доступ до результатів	ResultItems result = spider.get("http://webmagic.io/xxx")
getAll(String...)	синхронні вивози та прямий доступ до купи результатів	List<ResultItems> results = spider.getAll("http://webmagic.io/xxx" "http://webmagic.io/xxx")

#### 4.4.2 Site

Сам сайт, деяка інформація про конфігурацію, така як кодування, HTTP заголовок, тайм-аут, стратегія повторних спроб, агентів і т.д., можуть бути налаштовані шляхом установок set об'єкта `Site`.

метод	опис	приклади
setCharset(String)	установка кодування сторінки encoding code page	site.setCharset("utf-8")
setUserAgent(String)	установка UserAgent	site.setUserAgent("Spider")
setTimeOut(int)	установка тайм-аут в мілісекундах	site.setTimeOut(3000)
setRetryTimes(int)	установка к-ті повторних	site.setRetryTimes(3)

	спроб	
setCycleRetryTimes(int)	установка циклу повторних спроб	site.setCycleRetryTimes(3)
addCookie(String, String)	додати cookie	site.addCookie("dotcomt_user", "c
setDomain(String)	установити доменне ім'я, можливо встановити пізніше, вступають в силу тільки з addCookie	site.setDomain("github.com")
addHeader(String, String)	додати заголовок визову header	site.addHeader("Referer", "https://g
setHttpProxy(HttpHost)	налаштування http proxy	site.setHttpProxy(new HttpHost("127.0.0.1", 8080))

Цикл повторних спроб - механізм `CycleRetryTimes` додано починаючи з версії 0.3.0

Цей механізм для тих `URL`, що не зміг завантажити з `setCycleRetryTimes(int)` разів. Пройшовши увесь цикл він не буде повернати назад ці сторінки в кінець черги (повторних спроб). Має сенс для сторінок в мережі, що можуть бути тимчасово чи взагалі недоступні.

## 4.5 Знайомство з інструментами вилучення - екстракцією

Екстрактор пошукача WebMagic extraction базується на використанні [Jsoup](#) і моїй власній розробці [Xsoup](#).

### 4.5.1 Jsoup

[Jsoup](#) це простий HTML-парсер, що підтримує використання `css` селекторів, як спосіб знайти елементи. Для цілей розробки WebMagic, проведено детальний аналіз з сирцевим кодом [Jsoup](#) конкретних статей див. [Jsoup - нотатки дослідження](#).

### 4.5.2 Xsoup

[Xsoup](#) заснований на [Jsoup](#), розроблявся як `xpath` аналізатор. Раніше для парсера WebMagic застосовувався [HtmlCleaner](#), у якого є деякі проблеми під час використання. Основна його проблема полягає в точностях та помилках `XPath` - локалізації у сторінках якого не валідна структура коду, і це складно налаштовувати.

Реалізований [Xsoup](#) більш відповідає до потреб пошукових сканерів. Також присмно відзначити - тестування показує продуктивність [Xsoup](#) перевищує [HtmlCleaner](#) більше, аніж у два рази. Розробка [Xsoup](#) продовжується і досі, зараз підтримується загальний синтаксис для парсингу, що приведено у таблиці:

Назва	Вираз	Підтримка
nodename	nodename	так
immediate parent	/	так
parent	//	так
attribute	[@key=value]	так
nth child	tag[n]	так
attribute	/@key	так
wildcard in tagname	/*	так
wildcard in attribute	/[@*]	так
function	function()	частково

or	a   b	так, починаючи з 0.2.0
parent in path	. or ..	ні
predicates	price>35	ні
predicates logic	@class=a or @class=b	так, починаючи з 0.2.0

Зазначте, що це умовні визначення - частні - лише для пошукача і за для зручності використання функції `XPath`, проте ці функції не є стандартами `XPath` - зверніть на це увагу.

Вираз	Опис	XPath1.0
<code>text(n)</code>	текст безпосередньо n-го дочірнього вузла, якщо 0 - тоді для всіх	тільки <code>text()</code>
<code>allText()</code>	всі прямі і непрямі тексти дочірніх вузлів	не підтримує
<code>tidyText()</code>	всі прямі і непрямі тексти дочірніх вузлів, а також замінити деякі мітки обгорок, так, щоб відображався очищений звичайний текст	не підтримує
<code>html()</code>	внутрішній HTML, HTML теги не вміщати в себе	не підтримує
<code>outerHtml()</code>	внутрішній HTML, вміщувати в тому числі теги HTML у себе	не підтримує
<code>regex(@attr,expr,group)</code>	тут <code>@attr</code> і може бути вибраний з групи, за замовчуванням <code>group0</code>	не підтримує

### 4.5.3 Saxon

[Saxon](#) є потужним аналізатором `XPath` з підтримкою [XPath 2.0](#) синтаксису. [Webmagic-saxon](#) інтеграція з `Saxon` є попередніми. Але тепер, схоже, синтаксис `XPath 2.0` є передовим, і здається, що його використовують у розробці не так багато пошукових сканерів.

## 4.6 Монітор з JMX

Монітор шукача нова функція починаючи з версії 0.5.0. За допомогою цієї функції ви можете перевірити запущених пошукових роботів - скільки сторінок було завантажено, яка кількість сторінок, скільки запущених потоків thread і іншу інформацію. Побачити функціональність імплементованої JMX (Java Management Extensions), ви можете використовувати інструмент JConsole чи т.п. JMX працює, як з локальною так і віддаленої інформацією про пошукач.

Якщо ви раніше не користувалися JMX, то це не має значення, оскільки його використання є відносно простим, про це докладніше пояснюється в цій главі. Якщо ви хочете дізнатися про принцип, чи отримати ще більше інформації про JMX рекомендуюмо прочитати: [JMX \(en\)](#) чи [JMX \(chinese\)](#). У цій частині наведено багато інформації з посилання на джерело.

**Примітка:** Якщо ви визначаєте планувальник Scheduler, вам потрібно імплементувати інтерфейс MonitorableScheduler, та перегляньте ці два метода LeftPageCount та TotalPageCount.

### 4.6.1 Додати у проект моніторинг

Додати моніторинг дуже просто - отримати синглетон SpiderMonitor.instance() від SpiderMonitor, якщо ви бажаєте моніторити Spider, потрібно ще його зареєструвати. Також Ви можете зареєструвати декілька пошукових сканерів Spider для моніторингу у SpiderMonitor.

```
public class MonitorExample {

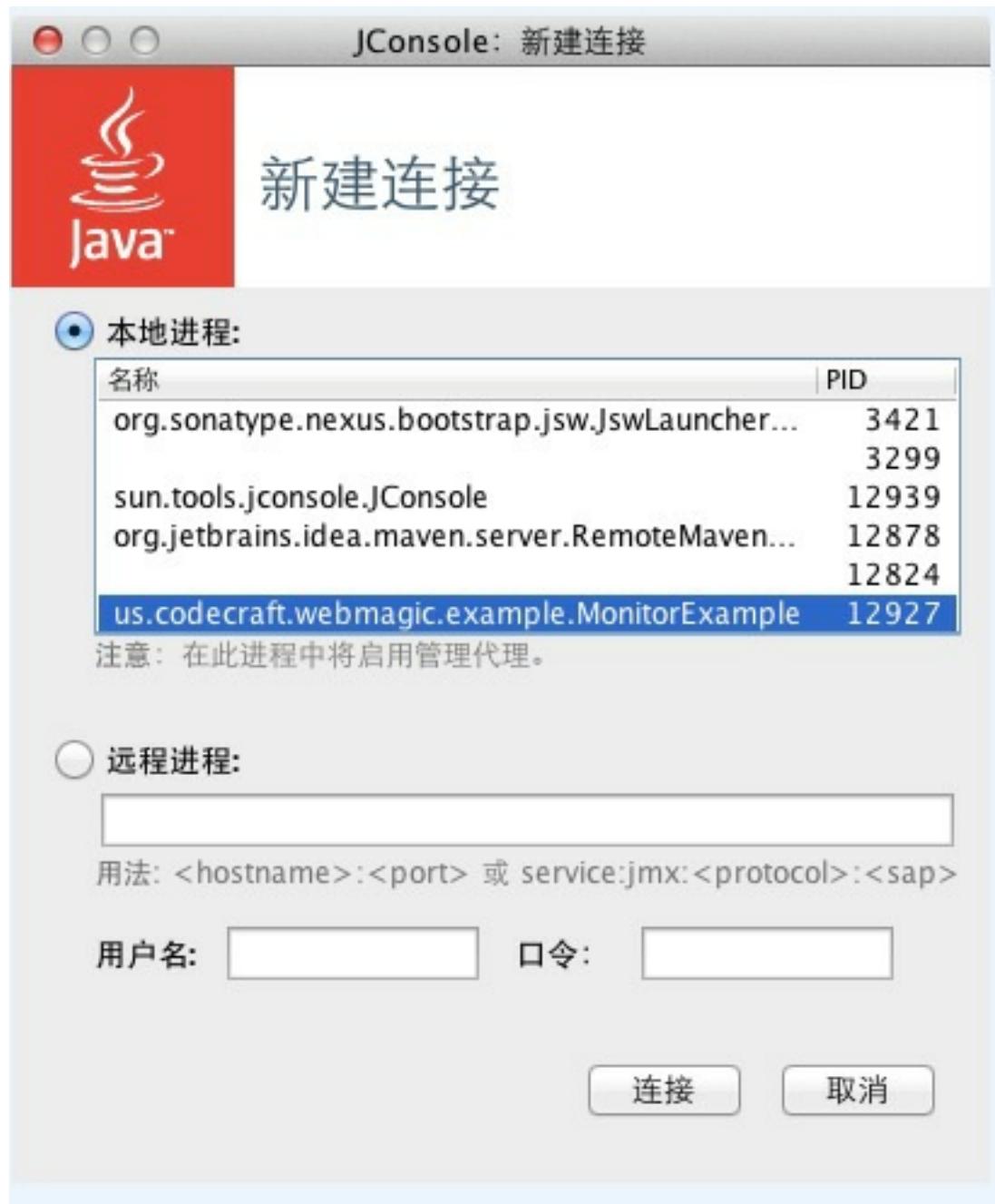
    public static void main(String[] args) throws Exception {
        Spider oschinaSpider = Spider.create(new OschinaBlogPageProcessor())
            .addUrl("http://my.oschina.net/flashsword/blog");
        Spider githubSpider = Spider.create(new GithubRepoPageProcessor())
            .addUrl("https://github.com/code4craft");

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

### 4.6.2 Знайомство інформаційним моніторингом

Моніторинг WebMagic проходить за допомогою `JMX provides control` (забезпечує управління), ви можете використовувати будь-який клієнт JMX-enabled, що підтримує підключення. Будемо використовувати `JDK` для наведеного прикладу `JConsole`. Спочатку стартуємо `Spider` WebMagic і додати код монітора. Потім переходимо в `JConsole`, щоб переглянути його.

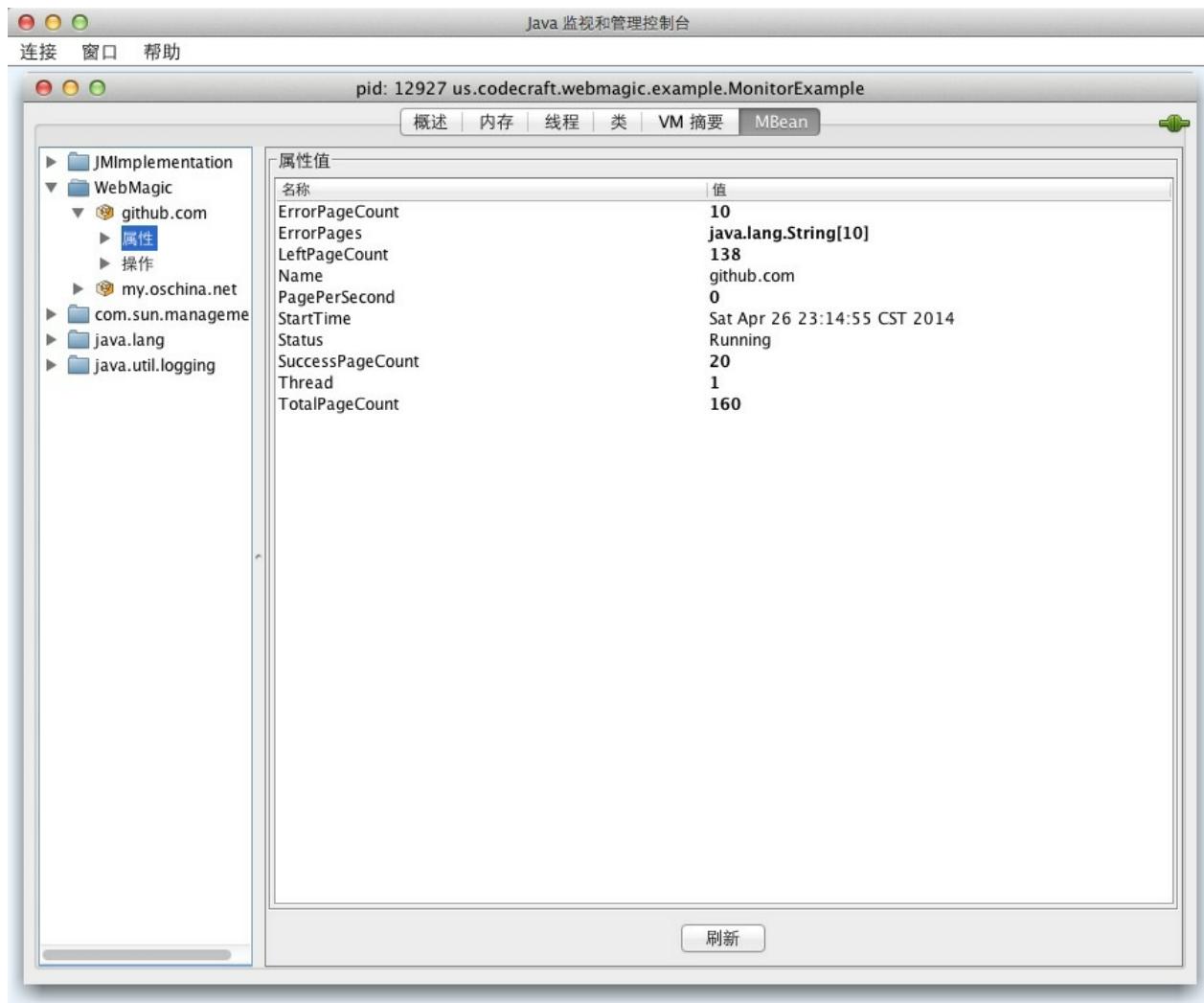
Запускаємо програму відповідно до прикладу 4.6.1, а потім введіть команду `JConsole` (під Windows, у вікні терміналу в DOS (інтерфейс командного рядка (CLI)) вводиться `jconsole.exe` ), щоб запустити `JConsole`.



У відкритому вікні обираємо - стартувати локальний процес WebMagic, вибрати `MBean`. Після підключення відкривається "WebMagic" де побачимо всю інформацію,

що доступна для моніторингу пошукача Spider !

Звідси можна керувати “дією” пошукових сканерів - для запуску `-start()` , а для зупинки `-stop()` .



### 4.6.3 Розширений інтерфейс моніторингу

На додаток до існуючих опцій інформаційного моніторингу, при потребі отримувати більше інформації, що необхідно контролювати, є рішення - це розширення. Ви можете успадковувати `SpiderStatusMXBean` щоби задіяти розширення, конкретні приклади можна побачити тут: [Кастомне розширення - demo](#).

## 5. Використання анотацій при написанні пошукача

WebMagic підтримують використання унікального стилю для написання пошукача анотаціями, вводиться пакет розширенням webmagic-extension для використання цієї функції.

У режимі анотацій, використовується простий базовий об'єкт та додають анотацію, щоб зменшити кількість коду та швидко завершити підготовку пошукового сканеру. Для простих сканерів, писати просто і легко зрозуміти, а також дуже легко управляти. Ось це одна з головних особливостей WebMagic, я назвав її - "Об'єкт/Екстракторний Мапінг" OEM (Object /Extraction Mapping).

Модель розробки заснована на анотаціях моделі полягає в наступному:

1. Перш за все, необхідно визначити екстракцій дані і написати клас.
2. Над початком класа поставити анотацію `@targetUrl`, що визначає стартовій URL для завантаження і вилучення.
3. Додайте анотацію `@ExtractBy` на поле класу, це поле визначає що саме буде екстрактовано (по селекторам чи регулярним виразам).
4. Визначити куди зберігається результат.

Тут ми ще раз навели приклади GitHub із глави 4, для запису подібну функцію сканерів, щоб пояснити використання анотацій. Все, підготовку закінчено для такого хорошого пошукового сканеру, як цей, чи так не простіше?

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

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

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

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

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

## 5.1 Написати клас моделі

Наведемо приклад з глави IV, що буде екстрактувати ім'я GitHub проекту, автора та readme інформацію з профілю, таким чином ми визначаємо клас моделі.

```
public class GithubRepo {  
  
    private String name;  
  
    private String author;  
  
    private String readme;  
  
}
```

Тут опущені getter та setter методи.

Результатом роботи пошукового роботу буде один чи декілька інстансів (екземплярів) цього класу.

## 5.2 TargetUrl та HelpUrl

На другому етапі розглянемо як відкрити URL. Але на початку необхідно означити два поняття: `@TargetUrl` та `@HelpUrl`.

### 5.2.1 TargetUrl та HelpUrl

`HelpUrl/TargetUrl` є дуже ефективним для розробки моделі краулера. `TargetUrl` вказує на URL з цільовими даними для кінцевої екстракції; `HelpUrl` - це проміжний, не фінальний URL, але завдяки якому шукаємо та надаємо доступ до кінцевих необхідних сторінок, що містять цільові данні для екстракції. Майже всю ієрархію, що потрібна для роботи пошукача можна описати цими двома типами обробки URL:

- Для блогу - `HelpUrl` це перелік сторінок пейджинатора, `TargetUrl` сторінка безпосередньо статті чи запису.
- Для форуму - `HelpUrl` це перелік гілок та постів у них, `TargetUrl` сторінка безпосередньо посту.
- Для ecommerce сайтів інтернет магазинів - `HelpUrl` це список категорій та пейджинатори, `TargetUrl` картка товару чи безпосередньо сторінка детального опису товару.

У цьому прикладі для `TargetUrl` - сторінка одного з проектів, що знайдений та входить до сторінки пошуку `HelpUrl`, що відображає посилання на всі проекти.

Враховуючи, що ми знаємо формат URL приведемо приклад:

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

### 5.2.2 Використання регулярних виразів у TargetUrl

Тут ми використовуємо регулярні вирази, щоб вказати область URL. Любий читач можливо знає, що у регулярних виразах не маскувати символ `.` - це неправильно? Й насправді, тут для зручності, WebMagic використовує власний формат регулярних виразів для URL-адреси, в основному двома відмінностями:

- Символи у URL-адреси зазвичай використовується за замовчуванням `.` - що рівнозначні замаскованій escape-послідовності `\.`

- Позначка `\*` замінена просто зірочкою `*` та може бути використаний безпосередньо.

Наприклад `https://github.com/*` валідний регулярний вираз для усіх URL на `https://github.com/`.

У WebMagic сторінка, що отримана з URL `TargetUrl`, за умови, що вони відповідають формату TargetUrl, також буде завантажена. Отже, якщо ви не вкажете `HelpUrl`, а це також можливо - наприклад у якому-небудь посиланні "Next" на сторінці блогу, то в цьому випадку необхідно вказати `HelpUrl`. (TODO???)

### 5.2.3 sourceRegion - область джерела

TargetUrl також підтримує визначення області, що буде джерелом `sourceRegion`, цей параметр є вираженням XPath, яке вказує URL, звідки вийшло - усі інші URL-адреси не з `sourceRegion` не будуть витягуватися.

## 5.3 @ExtractBy

Анотація `@ExtractBy` описує правила екстракції для вилучення елементів.

### 5.3.1 Знайомство з анотацією ExtractBy

Анотація `@ExtractBy` є ключовою для поля і означає - використовуй вказане правило екстракції, а результат екстракції завантажуй в поле під анотацією. Наприклад:

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

Де `//div[@id='readme']/text()` - це правило екстракції представлене у форматі XPath, а в результаті його виконання екстрактовані дані зберігаються у поле `readme` типу строка `String`.

### 5.3.2 Інші правила екстракції

На додаток крім XPath модна використовувати інші формати - CSS селектори, регулярні вирази та JsonPath, що доповнюється параметром анотації включенням типу `type`.

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

### 5.3.3 notNull

`@ExtractBy` може позначатися також властивістю `notNull`, якщо ви знайомі з MySql то ви розумієте, що мається на увазі: це поле має бути не пустим. Якщо пусте - то результат екстракції викидається. Для критичних / принципових атрибутів (таких як заголовок статті, та інше) деякої сторінки, при встановленні `notNull` як `true`, ефективно відсіюються не важливі сторінки.

Зверніть увагу! `notNull` дефолтно є `true`.

### 5.3.4 Multi (deprecated) (застаріле)

`Multi` атрибут типу булінь, що відокремлював правила екстракції для поля з кількома записами від одиничного запису. Для цього використовувався тип

`java.util.List`. Починаючи з версії 0.4.3 при умові - цільове полі має тип `List` - вмикається автоматично на `true`, тому вподальшому не актуально і більше не потрібно налаштовувати.

- до версії 0.4.3

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

- 0.4.3 та пізніше

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

### 5.3.5 ComboExtract (deprecated) (застаріле)

`@ComboExtract` для складних записів, що можуть комбінуватися різними правилами збору, включаючи логічні "AND/OR" ("i/або").

WebMagic з версії 0.4.3 використовує XSoup версії 0.2.0. У цій версії, значно покращено підтримку синтаксису XPath, та підтримує не тільки XPath і регулярні вирази, які використовуються в поєднанні, а також підтримує "|" ("або"). Тому автор вважає, `ComboExtract` це складне поєднання, більше не потрібні.

- XPath комбінується з регулярними виразами

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

- XPath чи take

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

### 5.3.6 ExtractByUrl

`@ExtractByUrl` окрема анотація, що означає - буде екстрактовано з URL.

Підтримується лише у правилах з регулярними виразами.

## 5.4 Використання ExtractBy над класом

У попередньому режимі анотацій над полем, ми мімо сторінку, що відповідає тільки одному результату. А якщо сторінка має кілька екстракцій, як їх записати?

Наприклад, в "QQ food" список сторінок <http://meishi.qq.com/beijing/c/all>, я хочу, щоб витягти всі підприємства й отримати їх імена та інформацію, тоді як це зробити?

Використовуючи анотацію `@ExtractBy` над класом ви можете вирішити цю проблему.

Пояснення в цьому разі для анотації на класі має дуже простий сенс: результат екстракції використовувати повторно для вилучення, так що ця область відповідає результату.

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

Коли використовуєте `@ExtractBy` на полі в класі, тоді із цих областей екстрагуються сторінки. Якщо в даний момент потрібно витягнути лише внутрішні (серцевий код) сторінки без її екстракції, то можна встановити `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 Результати перетворення типу

Перетворення типу (механізм `Formatter`) є розширені функціональні можливості у WebMagic 0.3.2. Оскільки зміст завжди повертається у рядку `String`, але ми хочемо мати у іншому типі динних. `Formatter` може екстрактовані дані автоматично перетворити в кілька основних типів без необхідності вручну прописувати код для перетворення.

Наприклад:

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

### 5.5.1 Підтримка автоматичної конвертації типів даних

Автоматична конвертація підтримує всі базові типи даних та їх типи класів-обгорток.

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

Крім того, він підтримує перетворення типу у `java.util.Date`. Проте, при перетворенні потрібно вказати формат дати. Формат відповідно до стандарту JDK визначені конкретними нормами, докладніше тут:

<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 Явно зазначити конвертацію типів

При нормальніх обставинах, Formatter будуть перетворені у відповідності з типом поля, але при особливих обставинах, нам потрібно буде зазначити вказати тип. Це інколи відбувається переважно в полі типу `list` - список.

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

## 5.5.3 Кастомний Formatter (TODO)

Насправді, на додаток до автоматичних перетворенням типу, Formatter також може робити деякі речі, щоб обробити результати. Наприклад, у нас є необхідний сценарій, результати повинні бути екстрактованими, в результаті екстракт подрібнюється на частини мозаїки, з яких складається необхідний рядок. Тут ми визначаємо шаблон формату рядку `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];
    }
}
```

Ну, ми можемо, після екстракції зробити деякі прості операції!

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

Ця фіча у версії 0.4.3 має баг BUG, але буде виправлено у 0.5.0

## 5.6 Повний процес

На теперішній час ми знаємо URL і відповідний API для екстракції, і основна підготовка пошукача була завершена.

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

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

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

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

### 5.6.1 Створення і запуск пошукача

Вхідна анотацій модель `oospider`, він успадковує клас `Spider`, який впроваджує спеціальний метод створення, та інші схожі методи. Створення анотацію режиму краулер вимагає один або більше класів `Model`, і один чи декілька

`PageModelPipeline` для визначення способу результату.

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

### 5.6.2 PageModelPipeline

У режимі анотацій, результати класу під назвою `PageModelPipeline`, Шляхом реалізації його, ви можете налаштовувати результатуючий вихід.

```
public interface PageModelPipeline<T> {

    public void process(T t, Task task);

}
```

`PageModelPipeline` відповідна з класом `Model`, та може відповідати безлічі моделей `PageModelPipeline`. За винятком випадків, коли ви створюєте також

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

Метод що додає Model до PageModelPipeline.

### 5.6.3 Висновок

Що ж, тепер ми маємо завершити цей приклад:

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

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

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

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

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

## 5.7 AfterExtractor

Іноді режим анотацій не може задоволити всі потреби, можливо тоді, буде потрібно написати код, щоб зробити деякі речі, у цьому випадку ми повинні використовувати інтерфейс `AfterExtractor`.

```
public interface AfterExtractor {
    public void afterProcess(Page page);
}
```

`afterProcess` в кінці метода екстракції, вже після ініціалізації та виклику полів, ви можете прописати деякий код зі своєю спеціальною логікою. По типу, що є у наступному прикладі [Постійне використання Jfinal ActiveRecord у пошукачі webmagic - blog](#):

```
// TargetUrl mean only the following URL format will be extracted to generate the object
model
/// TargetUrl означає - тільки наступного формату URL будуть використані для екстракції та запису у об'єктну модель
// 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 uncomfortable ...
/// Ось зроблена невелике позитивне зміна, ''- Значення за замовчуванням не потрібно збра-
сувати, і '*' буде автоматично замінений на '*', як описано в URL і вигляде трохи незручн
о ...
// Inherited jfinal the Model
/// Наслідується моделі jfinal
// Implement AfterExtractor interfaces can perform other operations after filling properties
/// В реалізації інтерфейса AfterExtractor можуть виконуватися інші операції після заповнення
властивостей
@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
    /// Буде автоматично екстрагувати з анотацією ExtractBy полів і заповнення
    // Default xpath grammar
    /// Синтаксис за замовчуванням Xpath
    @ExtractBy("//title")
    private String title;

    // Extract can be defined syntax Css, Regex, etc.
    /// Екстракт може бути визначений за синтаксисами CSS, Regexp і т.д.
    @ExtractBy(value = "div.BlogContent", type = ExtractBy.Type.Css)
    private String content;

    //Multi labeling drawing result can be a List
```

```
//Результат, що є множиною екстрактованих значень можуть бути збережені списком 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 want to go in filling
    // Jfinal властивість фактично є мапою Map замість поля, та це не має значення, я хочу їх заповнити
    this.set("title", title);
    this.set("content", content);
    this.set("tags", StringUtils.join(tags, ","));
    //save      /// запис
    save();
}

public static void main(String[] args) {
    C3p0Plugin c3p0Plugin = new C3p0Plugin("jdbc:mysql://127.0.0.1/blog?characterEncoding=utf-8", "blog", "password");
    c3p0Plugin.start();
    ActiveRecordPlugin activeRecordPlugin = new ActiveRecordPlugin(c3p0Plugin);
    activeRecordPlugin.addMapping("blog", OschinaBlog.class);
    activeRecordPlugin.start();
    //Start webmagic      // Запуск Webmagic
    OOSpider.create(Site.me()).addStartUrl("http://my.oschina.net/flashsword/blog/145796"), OschinaBlog.class).run();
}
}
```

## Висновок

Режим анотацій в даний час розглядається як кінець презентації WebMagic, модель анотацій фактично повністю заснована на `webmagic-core` у `PageProcessor` і інфопроводі `Pipeline` та реалізація розширення, хто зацікавлений - може повернутися та подивитися на код.

Частково це досягається, але це все ще більш складним, є проблема, якщо ви знайдете деякі деталі коду, прошу [відгук до мене](#).

## 6. Налаштування компонентів

У першому розділі ми посилаємося на компоненти WebMagic. У WebMagic великою перевагою є те, що ви можете налаштовувати гнучкі функції компонентів для досягнення мети під свої потреби.

У класі Павука Spider є поля - `PageProcessor` , `Downloader` , `Scheduler` і `Pipeline` .

Обов'язковий `PageProcessor` повинен бути призначеним при створені павука, інші три компонента можуть бути змінені за допомогою метода setter.

Метод	Опис	Приклад
<code>setScheduler()</code>	Зміна Scheduler	<code>spider.setScheduler(new FileCacheQueueScheduler("D:\data\w</code>
<code>setDownloader()</code>	Зміна Downloader	<code>spider.setDownloader(new SeleniumDownloader()))</code>
<code>addDownloader()</code>	Зміна Pipeline, один Spider може мати кілька pipeline	<code>spider.addPipeline(new FilePipeline(</code>

## 6.1 Налаштування інфопроводу Pipeline

Коли екстракцію закінчено, ми задіємо `Pipeline` щоб зберігти результат екстракції. Також можна налаштувати трубоінфопровід, щоб зробити деякі загальні функції. У цьому розділі ми представимо `Pipeline`, і використаємо два приклади для пояснення, як налаштувати інфопровід pipeline.

### 6.1.1 Введення трубопроводу

Інтерфейс `Pipeline` описується так:

```
public interface Pipeline {

    // ResultItems persists 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
    /// Результатуючі Елементи ResultItems сберігають результат екстракції, ця структура мап
    /// Дані в page.outField(key,value) (ключ, значення) можна отримати використавши Resu
    ltItems.get(key)
    public void process(ResultItems resultItems, Task task);
}
```

Видно, що `Pipeline` зберігає дані, які були екстрактовані за допомогою `PageProcessor`. Це робота, яку можна зробити в `PageProcessor`. Але чому ми використовуємо `Pipeline`? Існує кілька причин для цього:

1. Для того, щоб відокремити модулі. Екстракція сторінки і процес зберігання даних - етапи роботи павука. З одного боку, окремі модулі можуть зробити структуру коду більш ясним. З іншого боку, ми можемо відокремити процеси, пустити процес в іншому потоці або навіть на іншому сервері.
2. Функція `Pipeline` стабільніша, її дуже легко зробити в якості загального компонента. Існує велика різниця між процесами на різних сторінках. Але зберігаються дані майже також, наприклад, зберегати в файл або зберігати в базу даних. Це дуже узагальнено для майже більшості сторінок. Існує багато спільногого `Pipeline` в WebMagic, такі як записи в консолі, зберегти в файлі, зберегти в файлі у форматі JSON.

У WebMagic `Spider` може мати кілька `Pipeline`, досить виконати метод `Spider.addPipeline()`, щоб додати `Pipeline`. Ці всі `Pipeline` будуть як процес. Наприклад, ви можете використовувати:

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

Ви можете записати дані на консолі і зберегти у файлі.

## 6.1.2 Виведення результат в консоль

Щоб ознайомитися з `PageProcessor`, ми використовуємо `GithubRepoPageProcessor` як приклад. Маємо частина коду:

```
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
    /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()"));
}
```

Тепер ми хочемо записати результат в консолі. `ConsolePipeline` може це зробити.

```
public class ConsolePipeline implements Pipeline {

    @Override
    public void process(ResultItems<Object> 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());
        }
    }
}
```

Основуючись на цей приклад, ви можете налаштувати свій власний `Pipeline`.

Отримати дані з `ResultItems` і процес, як ваш власний метод.

## 6.1.3 Зберігаємо результат в MySQL

По-перше, ми вводимо приклад `jobhunter`. Це WebMagic інтегрує Spring Framework

в роботу пошукача інформацію про завдання. У цьому прикладі також показано, як використовувати Mybatis для збереження даних в базі даних MySQL.

У Java, у нас є багато методів, щоб зберегти дані в базі даних, таких як JDBC, dbutils, spring-JDBC, MyBatis. Ці інструменти можуть робити те ж саме, але їх складність не однакова. Якщо ми використовуємо JDBC, ми повинні отримати дані в ResultItem та зберегти їх.

Якщо ми використовуємо framework структуру ORM для збереження даних, ми зіткнемося з великою проблемою. Це framework основа всього потрібно чітко визначену модель, але не ключ-значення формату ResultItem. Ми використовуємо Mybatis як приклад для визначення DAO [MyBatis-Spring](#).

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

Все, що нам потрібно зробити, це імплементувати Pipeline, щоб об'єднати `ResultItem` та `LieTouJobInfo`.

## Режим анотацій

У режимі анотацій, існує [PageModelPipeline](#) в WebMagic:

```
public interface PageModelPipeline<T> {  
  
    //give the well processed object  
    public void process(T t, Task task);  
}
```

В цей час, ми можемо визначити [JobInfoDaoPipeline](#) щоб досягти функції:

```
@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);
    }
}
```

## Основний режим інфопроводу Pipeline

Ми закінчили роботу збереженням даних! Але як використовувати оригінальний інтерфейс `Pipeline`? Це дуже легко! Якщо ви хочете зберегти об'єкт, то ви повинні зберегти дані в якості об'єкта при екстракції його з сторінки.

```
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+)").all());
    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/text()").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);
    }
}
```

В `Pipeline` будемо використовувати

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

тоді отримаємо цей об'єкт

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 WebMagic зоготовки для Pipeline

WebMagic може записати результат на консоль, зберегти дані в файл або зберегти у форматі JSON.

Class	Опис	Remark
ConsolePipeline	write to the console	the result must implements the <code>toString()</code> method
FilePipeline	save the result in the file	the result must implements the <code>toString()</code> 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 Налаштування Планувальника Scheduler

Планувальник Scheduler це компонент WebMagic для управління списком черги URL-ів. Загалом планувальник виконує 2 функцій:

1. Управління списком черги URL-ів, що чекають на обробки пошукачем
2. Відфільтровувати повтори URL-ів

WebMagic мають деякі загальні Scheduler. Якщо ви хочете запустити простий пошукач, то вам не потрібно налаштовувати Планувальник Scheduler. Але має сенс для вас знати деякі з функцій.

Class	Опис	Зауваження
DuplicateRemovedScheduler	абстрактний клас - надає деякі шаблонні методи	роширюючи його можна впроваджувати свої власні функції
QueueScheduler	використовує чергу пам'яті для збереження URL	
PriorityScheduler	використовує пріоритети для черги пам'яті при збереженні URL	використання пам'яті більше, ніж QueueScheduler, але тільки коли ви встановлюєте пріоритети запrosів. Використання PriorityScheduler необхідне тільки якщо потрібно вказувати пріоритетність запитів
FileCacheQueueScheduler	використовується для зберігання URL у файл, коли виходять з програми і при наступному запуску можна продовжити сканування пошукачем URL, що були попередньо збережені в файлі.	Потрібно прописати шлях до файлу. Буде створено два файли .urls.txt та .cursor.txt

RedisScheduler	використовувати Redis для збереження черги, він може сканувати пошукачем у Інтернеті використовуючи розподілену систему	необхідно встановити Redis і запустити його
----------------	---	---

З версії 0.5.1 перероблено планувальник Scheduler - видалення дублів було винесено до окремого інтерфейсу: `DuplicateRemover`. Завдяки цьому ви можете встановити різні `DuplicateRemover` для одного планувальника Scheduler. Є два способи видалити дублікати.

<b>Class</b>	<b>Опис</b>
HashSetDuplicateRemover	використовує HashSet, щоб видаляти, що потребує великий обсяг пам'яті
BloomFilterDuplicateRemover	використовує Фільтр Блума BloomFilter для видалення, використовує менше пам'яті. Але він може не врахувати декілька URL

Всі планувальники за замовчуванням для видалення використовують `HashSetDuplicateRemover` (окрім `RedisScheduler`). Якщо у вас дуже багато URL, рекомендуємо використовувати `BloomFilterDuplicateRemover`. Наприклад:

```
spider.setScheduler(new QueueScheduler()
    .setDuplicateRemover(new BloomFilterDuplicateRemover(10000000)) //10000000 is the est
    imate value of urls ///10000000 приблизне значення кількості URL-ів
)
```

## 6.3 Використання завантажувача Downloader

Завантажувач Downloader за замовчуванням ґрунтується на `httpClient`. Загалом, вам не потрібно імплементувати Downloader, точка входу якого через `HttpClientDownloader`. Вона зроблена, щоб мати змогу використати інші клієнти.

З іншого боку, ви можете завантажити веб-сторінку в будь-який інший спосіб, наприклад, за допомогою `SeleniumDownloader`, щоб зробити рендер веб-сторінки.

## Практичні приклади написання пошукача

Навіть дуже досвідчений користувачам WebMagic можуть інколи зіткнутися з проблемами підготовлення пошукових роботів. Наприклад отримувати і періодично оновлювати чи сканувати сторінки з динамічним рендерингом, тощо.

У цьому розділі наведені приклади та розбір деяких загальні справи та кейси, сподіваємося цим допомогти читачам.

## Основні комбінації сторінки та список деталей

Почнемо з простого прикладу. Наприклад у нас є список сторінок. Вони вміщують лінки, що мають різні формати. Яким чином ми можемо відібрати з них посилання, щоб знайти всі цільові сторінки?

### 1 Початкові лінки прикладу

Ось, наприклад, блог автора Сіна <http://blog.sina.com.cn/flashsword20>. У ньому ми хочемо, щоб останню сторінку статті блога, екстрактувати називу title, контент, дату і іншу інформацію. Але ще сканувати посилання блогу та іншу інформацію зі списком сторінок, щоб отримати всі останні статті цього блогу.

- Сторінка Page

Формат сторінки "[http://blog.sina.com.cn/s/articlelist\\_1487828712\\_0\\_1.html](http://blog.sina.com.cn/s/articlelist_1487828712_0_1.html)", де у "0\_1" - це номеру сторінки "1".

· 动态规划算法的变种--备忘录算法	2010-01-12 15:55 [编辑] 更多▼
· zz职场人情：施受需谨慎	2009-12-26 14:23 [编辑] 更多▼
· 分治算法的一点思考--为什么大多使...	2009-12-25 10:47 [编辑] 更多▼
· 初学PS 贴2张做的海报	2009-12-14 18:06 [编辑] 更多▼

1 2 [下一页 >](#) 共2页

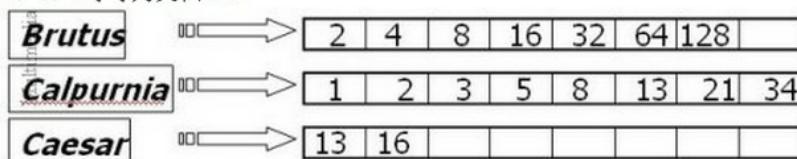
- Стаття сторінки Article Page

Формат статті сторінки "[http://blog.sina.com.cn/s/blog\\_58ae76e80100g8au.html](http://blog.sina.com.cn/s/blog_58ae76e80100g8au.html)", де "58ae76e80100g8au" змінна характеристика.

一道来自百度的面试题--倒排索引的AND操作		(2009-12-14 14:29:04)	[编辑][删除]	<a href="#">+ 转载 ▾</a>
------------------------	--	-----------------------	----------	------------------------

标签: it

倒排索引是以关键词作为索引项来索引文档的一种机制，如图中Brutus、Calpurnia、Caesar为关键词，2、4、8等等为文档ID。



现在有一个查询: Brutus AND Calpurnia AND Caesar。这个查询实际上就是要找出Brutus(以下简称B)、Calpurnia(以下简称C1)和Caesar(C2)的索引文档中的相同项。假设B、C1、C2的长度分别为m、n、p。

比较容易想到的是用归并排序的思想来解决这个问题。即：对3个线性表进行归并排序并对相同项计数，计数为3的项即为结果，这个方法时间复杂度为 $O(m+n+p)$ ，就这个例子我们需要比较 $7+8+2=17$ 次。

## 2 Пошук URL-ів на статті

По вимозі пошукача URL статті є нашою кінцевою метою, так як знайти всі статті в цьому блозі адреса є першим кроком сканерів.

Ми можемо використовувати регулярні вирази

```
http://blog\\.sina\\.com\\cn/s/blog_\\w+\\.html
```

 як фільтр грубої очистки для URL-ів.

Більш складним є те, що цей URL є занадто широким і може повзти до іншої інформації в блозі, тому ми повинні вказати область на сторінці зі списком URL, які необхідно отримати. Тут ми використовуємо `Xpath //div[@class='articleList']` для вибірки усіх областей. Потім використаємо `links()` або `Xpath //a/@href` щоб отримати всі посилання. І, нарешті, застосовуємо регулярні вирази

```
http://blog\\.sina\\.com\\cn/s/blog_\\w+\\.html
```

, щоб відфільтрувати URL та видалити деякі зайні категорії посилань "edit" (редагувати) або "more" (додаткове). Таким чином, ми можемо написати:

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

У той же час, нам потрібно знайти список всіх сторінок, щоб додати їх URL до списку чергі завантажень:

```
page.addTargetRequests(page.getHtml().links().regex("http://blog\\.sina\\.com\\cn/s/articlelist_1487828712_0_\\d+\\.html").all());
```

## 3 Екстракція контенту

Витяг інформації зі сторінки статті досить простий, застосовуємо `Xpath` вираз відповідний до цього.

```
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("\\((.*))"));

```

## 4 Розрізнати лінки на списки та цільові сторінки

Ми визначили список URL цільових сторінок і шляхи їх обробки. Тепер маємо справу їх розрізнати. У цьому випадку відрізнати дуже просто - формат списку URL

на інші сторінки та формат URL сторінки призначення відрізняється. Тому зробимо це!

```
// 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("\\((.*))"));
}
```

Переглянути повний код наведеного прикладу [SinaBlogProcessor.java](#).

## 5 Підсумки

У цьому прикладі використали декілька головних метода:

- Пошук посилань зі сторінки за допомогою регулярних виразів та їх фільтрація по області розташування.
- `PageProcessor` з двома типами налаштувань сторінки, в залежності від його URL, для відокремлення цілей між ними.

Друзі, що мали з деякі незручності та відмінності залишили [#issue83](#). Є плани з версії 0.5.0 додати до фіч у WebMagic `SubPageProcessor` для вирішення цих проблем.

## Обробка пошукачем сторінок на JS рендерингу

З незмінною популярністю технології AJAX та появи таких односторінкових програмних фреймворків як AngularJS, відтепер популярність JS рендеринга сторінок зростає все більше і більше. Для пошукових роботів, ця сторінка більше проблемна: якщо просто витягти вміст HTML, тоді часто не можуть отримати достовірну інформацію. Так як же мати справу з цієї сторінкою?

В цілому існує два підходи:

1. На етапі повзання, пошукач запускає ядро браузера та рендерить (прорисовує) сторінки з викорисанням JS, а потім починає по ній повзати. Цей аспект відповідних інструментів `Selenium`, `HtmlUnit` або `PhantomJs`. Хоча ці інструменти ефективні, але в той же час ще не такі стабільні. Перевага полягає в написанні правил, як і статичній сторінці.
2. При JS рендерингі сторінки данні приходять вкінці і в основному отримуються через AJAX. Тому аналіз AJAX запиту дасть відповідні дані, що є більш доцільним підходом. А по відношенню до стилю сторінки, то малоймовірно що інтерфейс зміниться. Недоліком є відносно складний процес, щоб знайти запит і змоделювати, котре вимагає великої кількості аналітичного досвіду.

На мій погляд, при порівнянні двох способів для разових або дрібносерійних налаштуваннях пошукача у першому підході економить час і зусилля. Але для довгострокового, великомасштабного проекту другий варіант буде ідеальним. Для деяких сайтів чи навіть деяких JS технологій пошукач може бути заплутаним, і в цьому випадку - перший спосіб в основному є панацеєю, в той час як другий буде дуже складним.

Для первого способу - `webmagic-selenium` що визначається як `Downloader` для завантаження сторінки, тобто рендерить ядром браузера. Конфігурація Selenium-а є відносно складною, але з версією платформи і не надто стабільний рішення. Детальніше дивіться цей блог: [використання Selenium для завантаження динамічних сторінок та їх обробки пошукачем](#)

Другий спосіб, і я сподіваюся, що врешті-решт ви подалаєте: розбір оригінальної сторінки JS рендеринга, що це не так складно. Сайт китайською про AngularJS <http://angularjs.cn/> в якості прикладу.

### 1 Як визначити JS рендеринга

Визначити сторінку з JS рендерингом відносно простим спосібом: в браузері

переглянути безпосередньо сирцевий код(під Windows Ctrl + U, під Mac командою + Alt + U). Якщо ефективної інформації немає - майже напевно Js рендеринг.

The screenshot shows the AngularJS Chinese Community website. At the top, there's a navigation bar with tabs for '最新' (Latest), '热门' (Hot), and '更新' (Updated). Below the navigation are several categories: 'AngularJS', 'JavaScript', 'AngularJS 开发指南' (AngularJS Development Guide), '问题与建议' (Issues & Suggestions), 'Node.js', 'jsGen', 'AngularJS 入门教程' (AngularJS Beginner Tutorial), 'AngularJS 技术杂谈' (AngularJS Technical Talk), '招聘' (Recruitment), '前端资讯' (Frontend News), 'WebApp', 'AngularJS 文档资讯' (AngularJS Documentation), '开发经验' (Development Experience), 'Backbone.js', 'MongoDB', 'HTML5', 'scope', 'JavaScript 异步编程' (JavaScript Asynchronous Programming), 'websocket', '区块' (Block), and a search bar labeled '更多' (More). Below this, two search results are displayed:

- 【上海】有孚计算机网络-前端攻城师** (Starred 6) - Published by 'm\_c是女超人' 21 hours ago, under 'JavaScript' and '招聘'.
- 123132123123123** (Starred 5) - Published by 'm\_c是女超人' 21 hours ago.

At the bottom, a browser window shows the source code of the page. The developer tools are open, specifically the 'Network' tab, which displays a message: '找不到结果' (Result not found). The URL in the address bar is 'view-source:angularjs.cn/?p=1'.

У цьому прикладі заголовок сторінки “有孚计算机网络-前端攻城师” (змінна комп’ютерна мережа - front-end відділ) не можна знайдений в вихідному коді, тому робимо висновок - використовується JS рендеринг, і ці дані отримані по AJAX.

## 2 Аналіз request запросяв

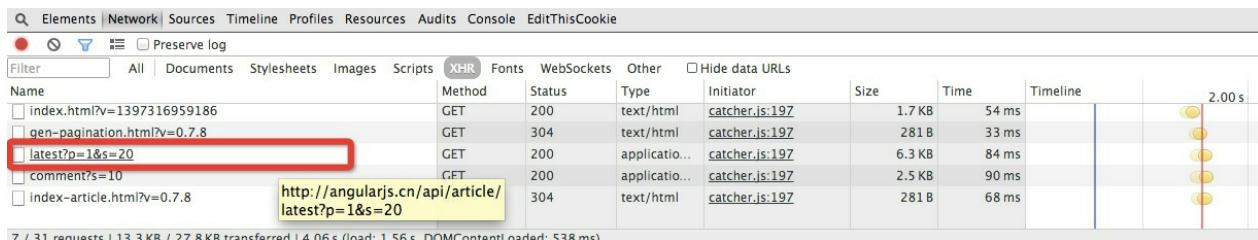
Тут ми входимо в найважчу частину: пошук запиту даних. В цьому нам допоможуть інструменти розробка для перегляду запитів веб-браузера.

На прикладі Chrome ми відкриваємо "Developer Tools" (під Windows, це F12, під Mac це command + Alt + I), потім оновіть сторінку (там можуть бути випадаюче меню або сторінки, та т.п., може ініціювати нові операції запрошу даних), і не забудьте зберегти цей запрос, він використовується для запиту - тому аналізуйте його!

Цей крок вимагає трохи терпіння, але це не з нізвідки. По-перше, що допоможе нам - зверху меню Сортування Sort (All, Document ()Bci, документувати) і інші

варіанти). Якщо це звичайний AJAX, з'являються під етикетою `XHR`, запит JSONP під `Scripts` етикетки, це є два найбільш поширеніх типів даних.

Після цього ви можете подивитися на дані для визначення розміру, результати, як правило, - інтерфейс через який проходить повернення даних. Зрештою в основному покладаються на досвід, ось наприклад "latest? P = 1 & s = 20" виглядає дуже підозріло ...



Для отримання підозрілого адреси, на цей раз можна подивитися на те, що є тілом відповіді. Доки не ясно, інструментами розробника дивимося URL

`http://angularjs.cn/api/article/latest?p=1&s=20` та копіюється в адресний рядок, запит ще раз (якщо для Chrome - то рекомендуємо встановити додаток jsonviewer, з ним легко побачити AJAX результати). Дивіться результати, здається, що ми хочемо знайти.

```
{
  ack: true,
  error: null,
  timestamp: 1397317821146,
  - data: [
    - {
      _id: "A0y2",
      + author: {...},
      date: 1397231093647,
      display: 0,
      status: 0,
      + refer: {...},
      title: "【上海】有孚计算机网络-前端攻城师",
      ...
    }
  ]
}
```

Таким же чином, ми входимо в сторінку детальних записів, та легко знайти конкретний зміст запиту `request http://angularjs.cn/api/article/A0y2`.

### 3 Програмування

Нагадаємо, що попередньо мæмо список цільових сторінок + приклади. Знаходимо цей запит аналогічно попередньому, але замінюється списком способів -AJAX у режимі AJAX. Данні режим AJAX повертаються в форматі JSON. Ну, ми все ще можемо використовувати спосіб як і минулого разу - обробляти як дві частини сторінки:

## 1. Список даних

В цьому списку на сторінці, ми повинні знайти потрібну інформацію, щоб допомогти нам побудувати таргетований AJAX URL. Бачимо, що повинні надати `_id` для повідомлення ID постів, та послати деталі request запиту на кілька фіксованих URL з прікіпленим ID компонентів. Таким чином, на цьому етапі, ми повинні вручну побудувати URL, і додати в чергу для сканування. Тут ми використовуємо мову `JsonPath` для вибору параметрів збору даних (розширеній пакет `webmagic-extension`, який забезпечує підтримку `JsonPathSelector`).

```
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.getRawText());
    if (CollectionUtils.isNotEmpty(ids)) {
        for (String id : ids) {
            page.addTargetRequest("http://angularjs.cn/api/article/"+id);
        }
    }
}
```

## 2. Об'єкти даний

При наявності URL-ів насправді зібрати цільові дані дуже просто, так як дані в форматі `JSON` повністю структурований. Так що відпадає необхідність написати процес аналізу `XPath` в нашій сторінки. Тут ми всюди використовуємо `JsonPath`, щоб отримати заголовок `title` і зміст `content`.

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

Переглянути повний код приклади [AngularJSProcessor.java](#)

## 4 Висновки

У цьому прикладі ми проаналізували процес сканування класичних динамічних сторінок. Насправді, при обробці пошукачем динамічних сторінок найбільша різниця: це більш складний пошук посилання. Ми порівняли дві моделі пошуку:

1. Відображається кінцевий сторінці

Завантажити допоміжну сторінку => знайти посилання => скачати і аналізувати цільові HTML

2. front-end рендеринг сторінки

Знайти допоміжні дані => побудови посилання => скачати і проаналізувати цільовий AJAX

Для різних сайтів, допоміжні дані можуть бути раніше основної HTML сторінки, це можливо коли request запрося йдуть через AJAX. Можливо, навіть обробляти множинний multiple request запит даних, але основна картина незмінна.

Проте, аналіз цих requests запитів даних, це все ще залишається набагато складніше, ніж аналіз сторінок. Так що це насправді важко витягати дані з динамічних сторінок.

Приклад в цьому розділі при аналізі запиту, із наданого зразку для переходу написана послідовність для пошукача  `знайти вторинні дані => побудови посилання => завантаження та аналіз цільовим AJAX .`

PS:

Для WebMagic з 0.5.0 буде додана підтримка послідовності `Json API`, пізніше ви можете використовувати:

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

Таким чином вибираються request `AJAX` запити.

Також підтримується:

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

Таким чином будуються request `JSONP` запити

