JAVA 爬虫框架 WebMagic

第一章 简介

1.1 爬虫概念

网络爬虫（又被称为网页蜘蛛，网络机器人，在 FOAF 社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁、自动索引、模拟程序或者蠕虫。

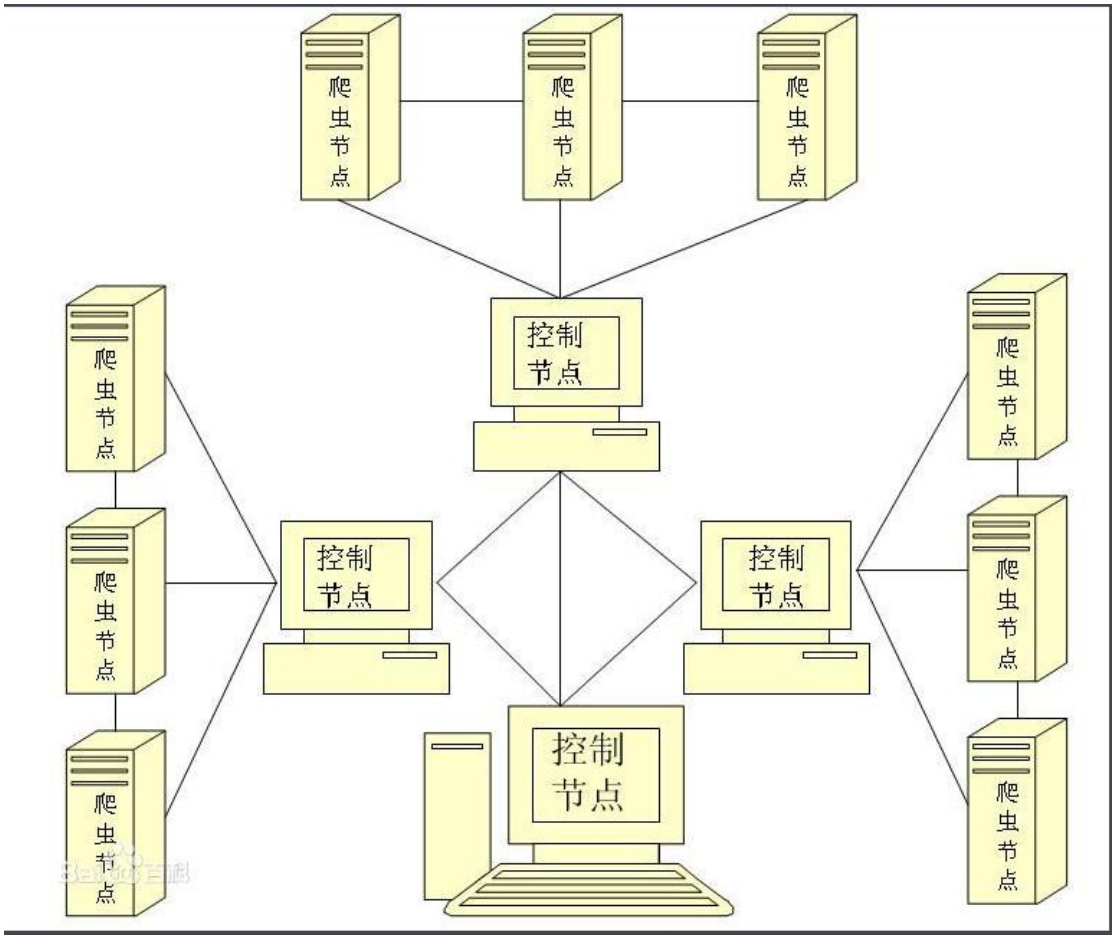
随着网络的迅速发展，万维网成为大量信息的载体，如何有效地提取并利用这些信息成为一个巨大的挑战。搜索引擎(Search Engine)，例如传统的通用搜索引擎百度，AltaVista，Yahoo! 和 Google 等，作为一个辅助人们检索信息的工具成为用户访问万维网的入口和指南。但是，这些通用性搜索引擎也存在着一定的局限性，如：

(1)不同领域、不同背景的用户往往具有不同的检索目的和需求，通用搜索引擎所返回的结果包含大量用户不关心的网页。

(2)通用搜索引擎的目标是尽可能大的网络覆盖率，有限的搜索引擎服务器资源与无限的网络数据资源之间的矛盾将进一步加深。

(3)万维网数据形式的丰富和网络技术的不断发展，图片、数据库、音频、视频多媒体等不同数据大量出现，通用搜索引擎往往对这些信息含量密集且具有一定结构的数据无能为力，不能很好地发现和获取。

(4)通用搜索引擎大多提供基于关键字的检索，难以支持根据语义信息提出的查询。



为了解决上述问题，定向抓取相关网页资源的聚焦爬虫应运而生。聚焦爬虫是一个自动下载网页的程序，它根据既定的抓取目标，有选择的访问万维网上的网页与相关的链接，获取所需要的信息。与通用爬虫(general purpose web crawler)不同，聚焦爬虫并不追求大的覆盖，而将目标定为抓取与某一特定主题内容相关的网页，为面向主题的用户查询准备数据资源。

1 聚焦爬虫工作原理以及关键技术概述

网络爬虫是一个自动提取网页的程序，它为搜索引擎从万维网上下载网页，是搜索引擎的重要组成。传统爬虫从一个或若干初始网页的 URL 开始，获得初始网页上的 URL，在抓取网页的过程中，不断从当前页面上抽取新的 URL 放入队列,直到满足系统的一定停止条件。聚焦爬虫的工作流程较为复杂，需要根据一定的网页分析算法过滤与主题无关的链接，保留有用的链接并将其放入等待抓取的 URL 队列。然后，它将根据一定的搜索策略从队列中选择下一步要抓取的网页 URL，并重复上述过程，直到达到系统的某一条件时停止。另外，所有被爬虫抓取的网页将会被系统存贮，进行一定的分析、过滤，并建立索引，以便之后的查询和检索；对于聚焦爬虫来说，这一过程所得到的分析结果还可能对以后的抓取过程给出反馈和指导。

相对于通用网络爬虫，聚焦爬虫还需要解决三个主要问题：

(1) 对抓取目标的描述或定义；

(2) 对网页或数据的分析与过滤；

(3) 对 URL 的搜索策略。

1.2 爬虫分类

网络爬虫按照系统结构和实现技术，大致可以分为以下几种类型：通用网络爬虫（General Purpose Web Crawler）、聚焦网络爬虫（Focused Web Crawler）、增量式网络爬虫（Incremental Web Crawler）、深层网络爬虫（Deep Web Crawler）。实际的网络爬虫系统通常是几种爬虫技术相结合实现的。

1.2.1 通用网络爬虫

又称全网爬虫（Scalable Web Crawler）爬行对象从一些种子 URL 扩充到整个 Web，主要为门户站点搜索引擎和大型 Web 服务提供商采集数据。 由于商业原因，它们的技术细节很少公布出来。 这类网络爬虫的爬行范围和数量巨大，对于爬行速度和存储空间要求较高，对于爬行页面的顺序要求相对较低，同时由于待刷新的页面太多，通常采用并行工作方式，但需要较长时间才能刷新一次页面。 虽然存在一定缺陷，通用网络爬虫适用于为搜索引擎搜索广泛的主题，有较强的应用价值。

通用网络爬虫的结构大致可以分为页面爬行模块 、页面分析模块、链接过滤模块、页面数据库、URL 队列、初始 URL 集合几个部分。为提高工作效率，通用网络爬虫会采取一定的爬行策略。

常用的爬行策略有：深度优先策略、广度优先策略。

1) 深度优先策略：其基本方法是按照深度由低到高的顺序，依次访问下一级网页链接，直到不能再深入为止。爬虫在完成一个爬行分支后返回到上一链接节点进一步搜索其它链接。当所有链接遍历完后，爬行任务结束。 这种策略比较适合垂直搜索或站内搜索， 但爬行页面内容层次较深的站点时会造成资源的巨大浪费。

2) 广度优先策略：此策略按照网页内容目录层次深浅来爬行页面，处于较浅目录层次的页面首先被爬行。 当同一层次中的页面爬行完毕后，爬虫再深入下一层继续爬行。 这种策略能够有效控制页面的爬行深度，避免遇到一个无穷深层分支时无法结束爬行的问题，实现方便，无需存储大量中间节点，不足之处在于需较长时间才能爬行到目录层次较深的页面。

1.2.2 聚焦网络爬虫

聚焦网络爬虫（Focused Crawler），又称主题网络爬虫（Topical Crawler），是指选择性地爬行那些与预先定义好的主题相关页面的网络爬虫。和通用网络爬虫相比，聚焦爬虫只需要爬行与主题相关的页面，极大地节省了硬件和网络资源，保存的页面也由于数量少而更新快，还可以很好地满足一些特定人群对特定领域信息的需求。

聚焦网络爬虫和通用网络爬虫相比，增加了链接评价模块以及内容评价模块。聚焦爬虫爬行策略实现的关键是评价页面内容和链接的重要性，不同的方法计算出的重要性不同，由此导致链接的访问顺序也不同。

1) 基于内容评价的爬行策略：DeBra 将文本相似度的计算方法引入到网络爬虫中，提出了 Fish Search 算法，它将用户输入的查询词作为主题，包含查询词的页面被视为与主题相关，其局限性在于无法评价页面与主题相关 度 的 高 低 。 Herseovic 对 Fish Search 算 法 进 行 了 改 进 ，提 出 了 Sharksearch 算法，利用空间向量模型计算页面与主题的相关度大小。

2) 基于链接结构评价的爬行策略 ：Web 页面作为一种半结构化文档，包含很多结构信息，可用来评价链接重要性。 PageRank 算法最初用于搜索引擎信息检索中对查询结果进行排序，也可用于评价链接重要性，具体做法就是每次选择 PageRank 值较大页面中的链接来访问。 另一个利用 Web 结构评价链接价值的方法是 HITS 方法，它通过计算每个已访问页面的 Authority 权重和 Hub 权重，并以此决定链接的访问顺序。

3) 基于增强学习的爬行策略：Rennie 和 McCallum 将增强学习引入聚焦爬虫，利用贝叶斯分类器，根据整个网页文本和链接文本对超链接进行分类，为每个链接计算出重要性，从而决定链接的访问顺序。

4) 基于语境图的爬行策略：Diligenti 等人提出了一种通过建立语境图（Context Graphs）学习网页之间的相关度，训练一个机器学习系统，通过该系统可计算当前页面到相关 Web 页面的距离，距离越近的页面中的链接优先访问。印度理工大学（IIT）和 IBM 研究中心的研究人员开发了一个典型的聚焦网络爬虫。 该爬虫对主题的定义既不是采用关键词也不是加权矢量，而是一组具有相同主题的网页。 它包含两个重要模块：一个是分类器，用来计算所爬行的页面与主题的相关度，确定是否与主题相关；另一个是净化器，用来识别通过较少链接连接到大量相关页面的中心页面。

1.2.3 增量式网络爬虫

增量式网络爬虫（Incremental Web Crawler）是指对已下载网页采取增量式更新和只爬行新产生的或者已经发生变化网页的爬虫，它能够在一定程度上保证所爬行的页面是尽可能新的页面。 和周期性爬行和刷新页面的网络爬虫相比，增量式爬虫只会在需要的时候爬行新产生或发生更新的页面 ，并不重新下载没有发生变化的页面，可有效减少数据下载量，及时更新已爬行的网页，减小时间和空间上的耗费，但是增加了爬行算法的复杂度和实现难度。增量式网络爬虫的体系结构[包含爬行模块、排序模块、更新模块、本地页面集、待爬行 URL 集以及本地页面 URL 集。

增量式爬虫有两个目标：保持本地页面集中存储的页面为最新页面和提高本地页面集中页面的质量。 为实现第一个目标，增量式爬虫需要通过重新访问网页来更新本地页面集中页面内容，常用的方法有：

1) 统一更新法：爬虫以相同的频率访问所有网页，不考虑网页的改变频率；

2) 2) 个体更新法：爬虫根据个体网页的改变频率来重新访问各页面；

3) 3) 基于分类的更新法：爬虫根据网页改变频率将其分为更新较快网页子集和更新较慢网页子集两类，然后以不同的频率访问这两类网页。

为实现第二个目标，增量式爬虫需要对网页的重要性排序，常用的策略有：广度优先策略、PageRank 优先策略等。IBM 开发的 WebFountain 是一个功能强大的增量式网络爬虫，它采用一个优化模型控制爬行过程，并没有对页面变化过程做任何统计假设，而是采用一种自适应的方法根据先前爬行周期里爬行结果和网页实际变化速度对页面更新频率进行调整。北京大学的天网增量爬行系统旨在爬行国内 Web，将网页分为变化网页和新网页两类，分别采用不同爬行策略。 为缓解对大量网页变化历史维护导致的性能瓶颈，它根据网页变化时间局部性规律，在短时期内直接爬行多次变化的网页 ，为尽快获取新网页，它利用索引型网页跟踪新出现网页。

1.2.4 Deep Web 爬虫

Web 页面按存在方式可以分为表层网页（Surface Web）和深层网页（Deep Web，也称 Invisible Web Pages 或 Hidden Web）。 表层网页是指传统搜索引擎可以索引的页面，以超链接可以到达的静态网页为主构成的 Web 页面。Deep Web 是那些大部分内容不能通过静态链接获取的、隐藏在搜索表单后的，只有用户提交一些关键词才能获得的 Web 页面。例如那些用户注册后内容才可见的网页就属于 Deep Web。 2000 年 Bright Planet 指出：Deep Web 中可访问信息容量是 Surface Web 的几百倍，是互联网上最大、发展最快的新型信息资源。

Deep Web 爬虫体系结构包含六个基本功能模块 （爬行控制器、解析器、表单分析器、表单处理器、响应分析器、LVS 控制器）和两个爬虫内部数据结构（URL 列表、LVS 表）。其中 LVS（Label Value Set）表示标签/数值集合，用来表示填充表单的数据源。

Deep Web 爬虫爬行过程中最重要部分就是表单填写，包含两种类型：

1) 基于领域知识的表单填写：此方法一般会维持一个本体库，通过语义分析来选取合适的关键词填写表单。 Yiyao Lu[25]等人提出一种获取 Form 表单信息的多注解方法，将数据表单按语义分配到各个组中 ，对每组从多方面注解，结合各种注解结果来预测一个最终的注解标签；郑冬冬等人利用一个预定义的领域本体知识库来识别 Deep Web 页面内容， 同时利用一些来自 Web 站点导航模式来识别自动填写表单时所需进行的路径导航。

2) 基于网页结构分析的表单填写： 此方法一般无领域知识或仅有有限的领域知识，将网页表单表示成 DOM 树，从中提取表单各字段值。 Desouky 等人提出一种 LEHW 方法，该方法将 HTML 网页表示为 DOM 树形式，将表单区分为单属性表单和多属性表单，分别进行处理；孙彬等人提出一种基于 XQuery 的搜索系统，它能够模拟表单和特殊页面标记切换，把网页关键字切换信息描述为三元组单元，按照一定规则排除无效表单，将 Web 文档构造成 DOM 树，利用 XQuery 将文字属性映射到表单字段。

Raghavan 等人提出的 HIWE 系统中，爬行管理器负责管理整个爬行过程，分析下载的页面，将包含表单的页面提交表单处理器处理，表单处理器先从页面中提取表单，从预先准备好的数据集中选择数据自动填充并提交表单，由爬行控制器下载相应的结果页面。

1.3 JAVA 常用爬虫对比

1.3.1 Apache Nutch2

地址:http://nutch.apache.org/

Nutch 是一个开源 Java 实现的搜索引擎。它提供了我们运行自己的搜索引擎所需的全部工具。包括全文搜索和 Web 爬虫。

Nutch 致力于让每个人能很容易，同时花费很少就可以配置世界一流的 Web 搜索引擎. 为了完成这一宏伟的目标, Nutch 必须能够做到:

* 每个月取几十亿网页
* 为这些网页维护一个索引
* 对索引文件进行每秒上千次的搜索

对索引文件进行每秒上千次的搜索
提供高质量的搜索结果
简单来说 Nutch 支持分布式，可以通过配置网站地址、规则、以及采集的深度(通用爬虫或全网爬虫)对网站进行采集，并提供了全文检索功能，可以对采集下来的海量数据进行全文检索；假如您想完成对站点所有内容进行采集，且不在乎采集和解析精度(不对特定页面特定字段内容采集)的需求，建议使用 Apache Nutch。

1.3.2 webmgaic(推荐)

地址:http://webmagic.io/
WebMagic 是一个简单灵活的 Java 爬虫框架。基于 WebMagic，你可以快速开发出一个高效、易维护的爬虫。
特性：
简单的 API，可快速上手
模块化的结构，可轻松扩展
提供多线程和分布式支持

1.3.3 Heritrix

地址:http://crawler.archive.org/
Heritrix 是一个由 java 开发的、开源的网络爬虫，用户可以使用它来从网上抓取想要的资源。其最出色之处在于它良好的可扩展性，方便用户实现自己的抓取逻辑。

1.3.4 WebCollector

地址:https://github.com/CrawlScript/WebCollector
WebCollector 是一个无须配置、便于二次开发的 JAVA 爬虫框架（内核），它提供精简的的 API，只需少量代码即可实现一个功能强大的爬虫。WebCollector-Hadoop 是 WebCollector 的 Hadoop 版本，支持分布式爬取。

1.3.5 crawler4j

地址::https://github.com/yasserg/crawler4j
crawler4j 是一款基于 Java 的轻量级单机开源爬虫框架，最大的一个特点就是简单。另外也支持多线程、支持代理、可以过滤重复 URL
基本上从加载 jar 到工程里面 通过修改示例的代码就可以简单的实现一个爬虫的全部功能，而这一切动作加起来都不需要超过半个小时。

1.3.6 Spiderman

地址:https://m.gitee.com/l-weiwei/spiderman
Spiderman 是一个 Java 开源 Web 数据抽取工具。它能够收集指定的 Web 页面并从这些页面中提取有用的数据。 Spiderman 主要是运用了像 XPath、正则、表达式引擎等这些技术来实现数据抽取。

1.3.7 SeimiCrawler

地址:http://seimi.wanghaomiao.cn/
一个敏捷的，独立部署的，支持分布式的 Java 爬虫框架
SeimiCrawler 是一个强大的，高效敏捷的，支持分布式的爬虫开发框架，希望能在最大程度上降低新手开发一个可用性高且性能不差的爬虫系统的门槛，以及提升开发爬虫系统的开发效率。在 SeimiCrawler 的世界里，绝大多数人只需关心去写抓取的业务逻辑就够了，其余的 Seimi 帮你搞定。设计思想上 SeimiCrawler 受 Python 的爬虫框架 Scrapy 启发很大，同时融合了 Java 语言本身特点与 Spring 的特性，并希望在国内更方便且普遍的使用更有效率的 XPath 解析 HTML，所以 SeimiCrawler 默认的 HTML 解析器是 JsoupXpath,默认解析提取 HTML 数据工作均使用 XPath 来完成（当然，数据处理亦可以自行选择其他解析器）。

1.3.8 jsoup

地址:https://jsoup.org/
jsoup 是一款 Java 的 HTML 解析器，可直接解析某个 URL 地址、HTML 文本内容。它提供了一套非常省力的 API，可通过 DOM，CSS 以及类似于 jQuery 的操作方法来取出和操作数据。

1.4 WebMagic 概述

WebMagic 是一个简单灵活的 Java 爬虫框架。基于 WebMagic，可以快速开发出一个高效、易维护的爬虫。
特性：
简单的 API，可快速上手
模块化的结构，可轻松扩展
提供多线程和分布式支持

WebMagic 的设计参考了业界最优秀的爬虫 Scrapy，而实现则应用了 HttpClient、Jsoup 等 Java 世界最成熟的工具，目标就是做一个 Java 语言 Web 爬虫的教科书般的实现。

如果你是爬虫开发老手，那么 WebMagic 会非常容易上手，它几乎使用 Java 原生的开发方式，只不过提供了一些模块化的约束，封装一些繁琐的操作，并且提供了一些便捷的功能。

如果你是爬虫开发新手，那么使用并了解 WebMagic 会让你了解爬虫开发的常用模式、工具链、以及一些问题的处理方式。熟练使用之后，相信自己从头开发一个爬虫也不是什么难事。

因为这个目标，WebMagic 的核心非常简单——在这里，功能性是要给简单性让步的。

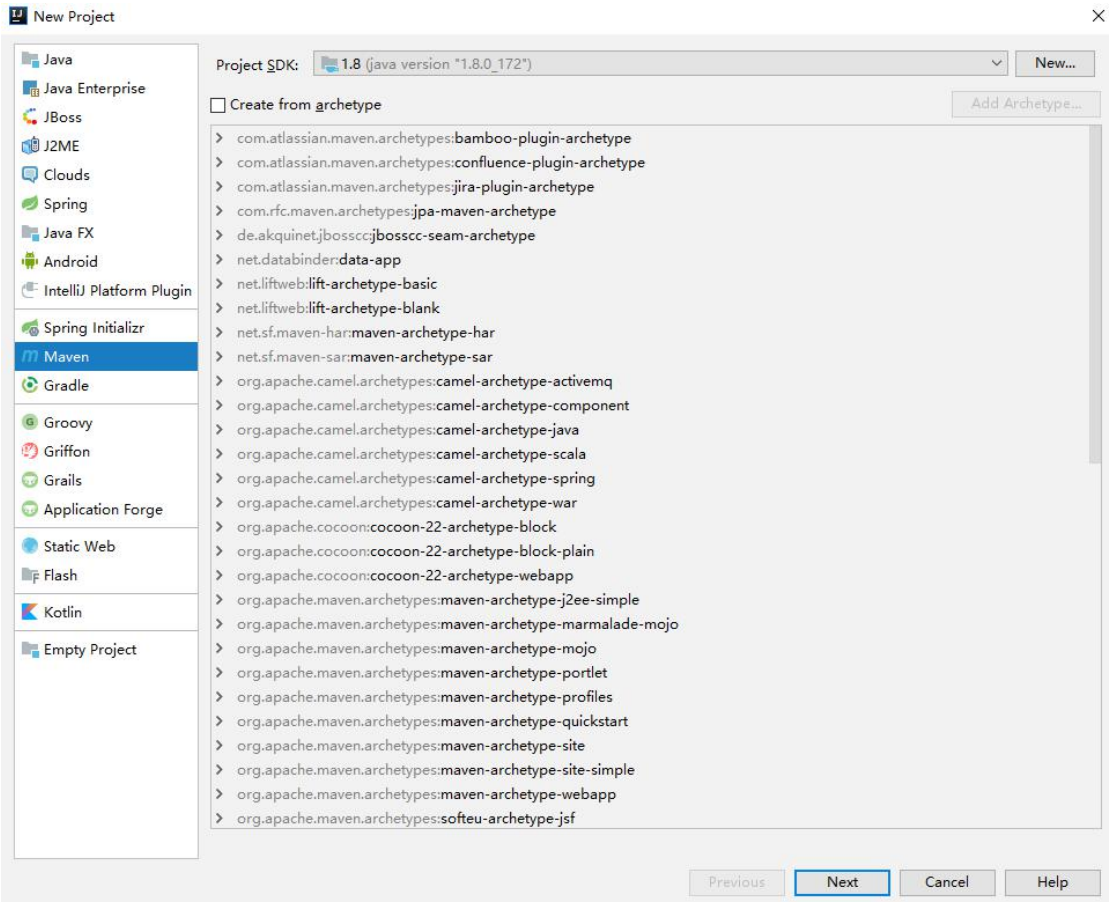
WebMagic 由四个组件(Downloader、PageProcessor、Scheduler、Pipeline)构成，核心代码非常简单，主要是将这些组件结合并完成多线程的任务。这意味着，在 WebMagic 中，你基本上可以对爬虫的功能做任何定制。

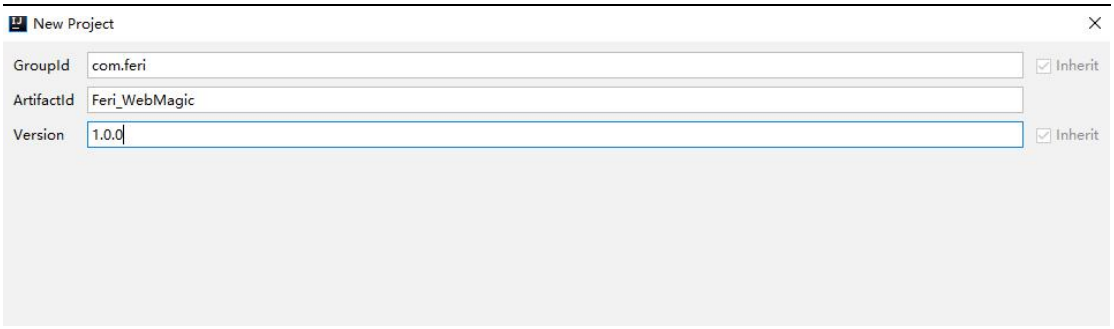
1.5 WebMagic 核心概述

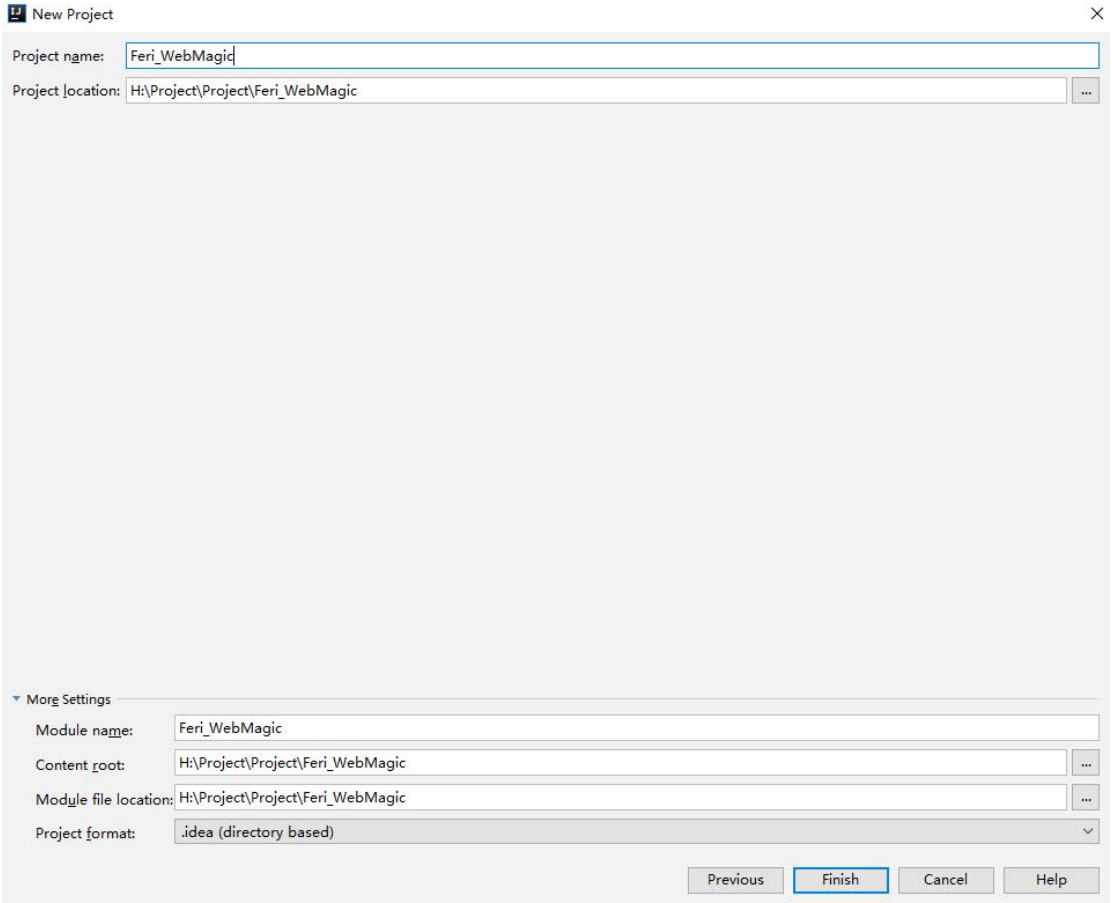
第二章 基本应用

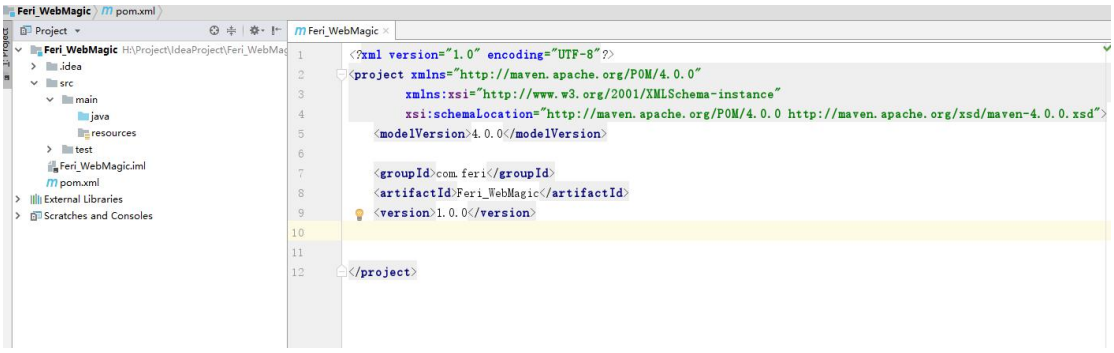
2.1 创建项目

开发工具：Idea+Maven




New Project
GroupId
com.feri
Inherit
ArtifactId
Feri_WebMagic
Version
1.0.0
Inherit


New Project
Project name:
Feri_WebMagic
Project location:
H:\Project\Project\Feri_WebMagic
More Settings
Module name:
Feri_WebMagic
Content root:
H:\Project\Project\Feri_WebMagic
Module file location:
H:\Project\Project\Feri_WebMagic
Project format:
.idea (directory based)
Previous
Finish
Cancel
Help


Feri_WebMagic
pom.xml
Project
Feri_WebMagic
.idea
src
main
java
resources
test
Feri_WebMagic.iml
pom.xml
External Libraries
Scratches and Consoles
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.feri</groupId>
<artifactId>Feri_WebMagic</artifactId>
<version>1.0.0</version>
</project>

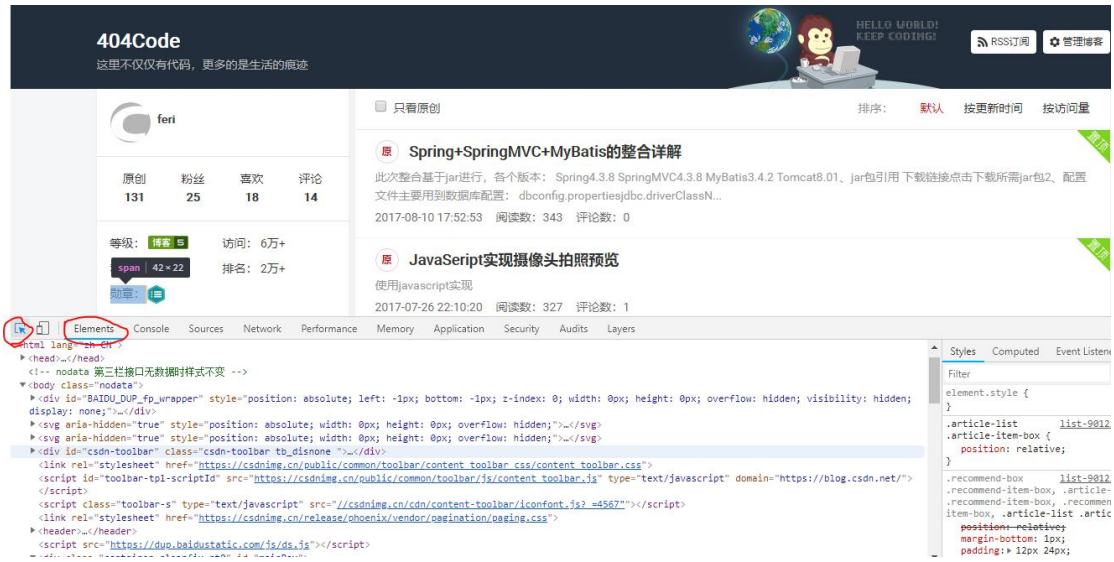
2.2 加载依赖

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.feri</groupId>
    <artifactId>Feri_WebMagic</artifactId>
    <version>1.0.0</version>

    <dependencies>
        <dependency>
            <groupId>us.codecraft</groupId>
            <artifactId>webmagic-core</artifactId>
            <version>0.7.3</version>
        </dependency>
        <dependency>
            <groupId>us.codecraft</groupId>
            <artifactId>webmagic-extension</artifactId>
            <version>0.7.3</version>
        </dependency>
    </dependencies>

</project>
```

2.3 分析网页

抓取 https://blog.csdn.net/xingfei_work 博客的文章信息

分析网页信息，F12 查看网页信息



采用 xpath(xml 或 html 格式提取技术)进行内容提取

找到自己感兴趣的内容

抓取所有的文章标题

```
div[@class='article-item-box']/h4/a/text()
```

以此类推

抓取文章摘要

```
div[@class='article-item-box']/p[@class='content']/a/text()
```

抓取日期

```
div[@class='article-item-box']/div[@class='info-box']/p[1]/span[@class='date']/text()
```

抓取阅读数量

```
div[@class='article-item-box']/div[@class='info-box']/p[2]/span[@class='read-num']/text()
```

抓取评论数量

```
div[@class='article-item-box']/div[@class='info-box']/p[3]/span[@class='read-num']/text()
```

2.4 编写网页内容提取

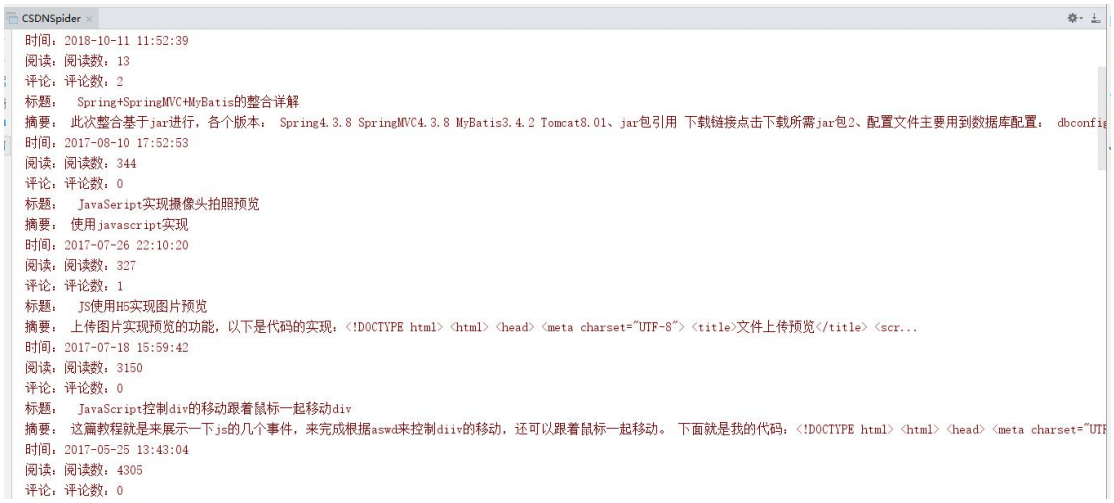
创建类实现 PageProcessor 接口，完成爬取的网页内容的筛选

```
*@Author feri
*@Date Created in 2018/10/9 19:02
*/
public class CSDNProcess implements PageProcessor {
    private Site site=Site.me();
    public void process(Page page) {
        //获取文章的标题
        List<String> titles=page.getHtml().xpath("div[@class='article-item-box']/h4/a/text()").all();
        //获取文章摘要
        List<String> contents=page.getHtml().xpath("div[@class='article-item-box']/p[@class='content']/a/text()").all();
        //获取时间
        List<String> dates=page.getHtml().xpath("div[@class='article-item-box']/div[@class='info-box']/p[1]/span[@class='date']/text()").all();
        //获取阅读量
        List<String> reads=page.getHtml().xpath("div[@class='article-item-box']/div[@class='info-box']/p[2]/span[@class='read-num']/text()").all();
        //获取评论
        List<String> comments=page.getHtml().xpath("div[@class='article-item-box']/div[@class='info-box']/p[3]/span[@class='read-num']/text()").all();
        for(int i=0;i<titles.size();i++){
            System.err.println("标题："+titles.get(i));
            System.err.println("摘要："+contents.get(i));
            System.err.println("时间："+dates.get(i));
            System.err.println("阅读："+reads.get(i));
            System.err.println("评论："+comments.get(i));
        }
    }
    public Site getSite() { return site; }
}
```

2.5 开始爬取数据

```
/**
 *@Author feri
 *@Date Created in 2018/10/9 19:04
 */
public class CSDNSpider {
    public static void main(String[] args) {
        new Spider(new CSDNProcess()).
                addUrl("https://blog.csdn.net/xingfei_work").
                addPipeline(new ConsolePipeline()).run();
    }
}
```

2.6 结果展示



源码下载地址：https://github.com/xingpenghui/Feri_WebMagic

第三章 详解

3.1 核心模块

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

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

3.1.1 Downloader

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

3.1.2 PageProcessor

PageProcessor 负责解析页面，抽取有用信息，以及发现新的链接。WebMagic 使用 Jsoup 作为 HTML 解析工具，并基于其开发了解析 XPath 的工具 Xsoup。

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

3.1.3 Scheduler

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

除非项目有一些特殊的分布式需求，否则无需自己定制 Scheduler。

3.1.4 Pipeline

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

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

常用的 Pipeline 有：ConsolePipeline（结果输出到控制台）、自定义 Pipeline

3.2 数据流转的对象

3.2.1 Request

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

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

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

3.2.2 Page

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

Page 是 WebMagic 抽取过程的核心对象，它提供一些方法可供抽取、结果保存等。在第四章的例子中，我们会详细介绍它的使用。

3.2.3 ResultItems

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