day01-网络爬虫　　包名一定要是crawler

第一章-谈谈网络爬虫

1.什么是网络爬虫

在大数据时代，信息的采集是一项重要的工作，而互联网中的数据是海量的，如果单纯靠人力进行信息采集，不仅低效繁琐，搜集的成本也会提高。如何自动高效地获取互联网中我们感兴趣的信息并为我们所用是一个重要的问题，而爬虫技术就是为了解决这些问题而生的。 网络爬虫（Web crawler）也叫做网络机器人，==可以代替人们自动地在互联网中进行数据信息的采集与整理==。它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==，可以自动采集所有其能够访问到的页面内容，以获取或更新这些网站的内容和检索方式。 从功能上来讲，爬虫一般分为数据采集，处理，储存三个部分。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

2.网络爬虫可以做什么

- 可以实现搜索引擎
- 大数据时代，可以让我们获取更多的数据源。
- 快速填充测试和运营数据
- 为人工智能提供训练数据集

3.网络爬虫常用的技术（Java）

3.1底层实现 HttpClient + Jsoup

HttpClient 是 Apache Jakarta Common 下的子项目，用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。HttpClient 已经应用在很多的项目中，比如 Apache Jakarta 上很著名的另外两个开源项目Cactus 和 HTMLUnit 都使用了 HttpClient。更多信息请关注http://hc.apache.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。

3.2开源框架 Webmagic

webmagic是一个开源的Java爬虫框架，目标是简化爬虫的开发流程，让开发者专注于逻辑功能的开发。webmagic的核心非常简单，但是覆盖爬虫的整个流程，也是很好的学习爬虫开发的材料.

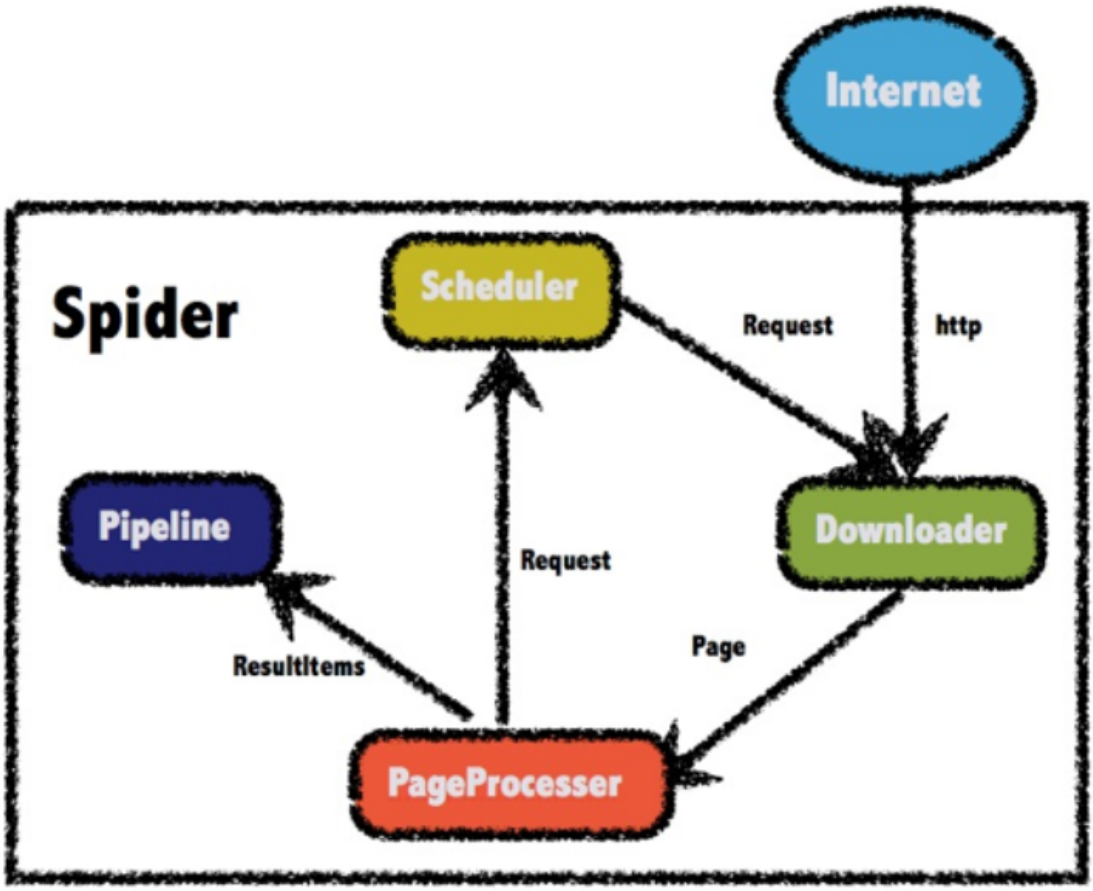
webmagic的主要特色

- 完全模块化的设计，强大的可扩展性。
- 核心简单但是涵盖爬虫的全部流程，灵活而强大，也是学习爬虫入门的好材料。
- 提供丰富的抽取页面API。
- 无配置，但是可通过POJO+注解形式实现一个爬虫。
- 支持多线程。
- 支持分布式。
- 支持爬取js动态渲染的页面。
- 无框架依赖，可以灵活的嵌入到项目中去

第二章-爬虫框架Webmagic

1.架构解析

WebMagic项目代码分为核心和扩展两部分。核心部分(webmagic-core)是一个精简的、模块化的爬虫实现，而扩展部分则包括一些便利的、实用性的功能。扩展部分(webmagic-extension)提供一些便捷的功能，例如注解模式编写爬虫等。同时内置了一些常用的组件，便于爬虫开发。 WebMagic的设计目标是尽量的模块化，并体现爬虫的功能特点。这部分提供非常简单、灵活的API，在基本不改变开发模式的情况下，编写一个爬虫。 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。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。



四大组件:

- 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ApacheHttpClient作为下载工具。

- 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。

- 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

- 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

2.PageProcessor

2.1爬取页面全部内容

2.1.1相关的API

- Spider是爬虫启动的入口。在启动爬虫之前，我们需要使用一个PageProcessor创建一个Spider对象，然后使用run()进行启动。 同时Spider的其他组件（Downloader、Scheduler、Pipeline）都可以通过set方法来进行设置。

方法	说明	示例
create(PageProcessor)	创建Spider	Spider.create(newGithubRepoProcessor())
addUrl(String...)	添加初始的URL	spider.addUrl("http://webmagic.io/docs/")
thread(n)	开启n个线程	spider.thread(5)
run()	启动，会阻塞当前线程执行	spider.run()
start()/runAsync()	异步启动，当前线程继续执行	spider.start()
stop()	停止爬虫	spider.stop()
addPipeline(Pipeline)	添加一个Pipeline，一个Spider可以有多个Pipeline	spider .addPipeline(newConsolePipeline())
setScheduler(Scheduler)	设置Scheduler，一个Spider只能有个一个Scheduler	spider.setScheduler(newRedisScheduler())
setDownloader(Downloader)	设置Downloader，一个Spider只能有个一个Downloader	spider .setDownloader(newSeleniumDownloader())
get(String)	同步调用，并直接取得结果	ResultItems result = spider.get("http://webmagic.io/docs/")
getAll(String...)	同步调用，并直接取得一堆结果	List results = spider.getAll("http://webmagic.io/docs/","http://webmagic.io/xxx")

- 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

方法	说明	示例
Selectable getUrl()	获取当前页url	page.getUrl()
Html getHtml()	获取当前页内容	page.getHtml()
putField(String, Object)	保存抓取的结果	page.putField("title",page.getHtml().toString)
addTargetRequests()	添加url 去抓取	

- Site用于定义站点本身的一些配置信息，例如编码、HTTP头、超时时间、重试策略等、代理等，都可以通过设置Site对象来进行配置

方法	说明	示例
setCharset(String)	设置编码	site.setCharset("utf-8")
setUserAgent(String)	设置UserAgent	site.setUserAgent("Spider")
setTimeOut(int)	设置超时时间， 单位是毫秒	site.setTimeOut(3000)
setRetryTimes(int)	设置重试次数	site.setRetryTimes(3)
setCycleRetryTimes(int)	设置循环重试次数	site.setCycleRetryTimes(3)
addCookie(String,String)	添加一条cookie	site.addCookie("dotcomt_user","code4craft")
setDomain(String)	设置域名，需设置域名后，addCookie才可生效	site.setDomain("github.com")
addHeader(String,String)	添加一条addHeader	site.addHeader("Referer","https://github.com")
setHttpProxy(HttpHost)	设置Http代理	site.setHttpProxy(newHttpHost("127.0.0.1",8080))

2.1.2代码实现

页面路径: https://blog.csdn.net/nav/ai

- 创建工程，引入依赖

```
    <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>
```

```
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
            <version>2.0</version>
        </dependency>
    </dependencies>
```

- 编写类实现网页内容的爬取

```
public class MyProcessor implements PageProcessor {

    @Override
    //分析页面
    public void process(Page page) {
        System.out.println(page.getHtml().toString());
    }

    @Override
    //配置
    public Site getSite() {
        Site site = new Site();
        //设置失败后重试时长为3s，设置两个页面之间爬去间隔为0.1s
        site.setRetryTimes(3000).setSleepTime(100);
        return site;
    }

}
```

- 创建爬虫任务类CrawlerTask

```
public class CrawlerTask {

    public static void main(String[] args) {
        Spider.create(new MyProcessor()).addUrl("https://blog.csdn.net/nav/ai").run();
    }

}
```

2.2爬取指定内容（XPath）

如果我们想爬取网页中部分的内容，需要指定xpath。

XPath，即为XML路径语言（XMLPathLanguage），它是一种用来确定XML文档中某部分位置的语言。XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。 语法详见附录A. 我们通过指定xpath来抓取网页的部分内容

语法: `xpath(xpath字符串)`

```
System.out.println(page.getHtml().xpath("//*[@id=\"nav\"]/div/div/ul/li[5]/a").toString());
```

以上代码的含义：id为nav的节点下的div节点下的div节点下的ul下的第5个li节点下的a节点

2.3添加目标地址

我们可以通过添加目标地址，从种子页面爬取到更多的页面. 运行后发现好多地址都出现在控制台

```
    @Override
    //分析页面
    public void process(Page page) {
       //把当前页面的所有链接都添加到目标页面进行爬取
       page.addTargetRequests(page.getHtml().links().all());

System.out.println(page.getHtml().xpath("//[@id=\"nav\"]/div/div/ul/li[5]/a/text()").toString())
;
    }
```

2.4目标地址正则匹配

需求：只提取播客的文章详细页内容，并提取标题

页面路径: https://blog.csdn.net/fjl_CSDN/article/details/79019437

语法: `links().regex(正则)`

```
    @Override
    //分析页面
    public void process(Page page) {
        //https://blog.csdn.net/fjl_CSDN/article/details/79019437
        page.addTargetRequests(page.getHtml().links().regex("https://blog.csdn.net/[a-zA-Z0-
9_]+/article/details/[0-9]{8}").all());
        System.out.println(page.getHtml().xpath("//*
[@id=\"mainBox\"]/main/div[1]/div/div/div[1]/h1/text()").toString());
    }
```

3.Pipeline

3.1ConsolePipeline 控制台输出

我们可以通过ConsolePipeline对象把爬取的内容打印到控制台

```
public class MyProcessor implements PageProcessor {

    @Override
    //分页页面，提取路径,数据和存储
    public void process(Page page) {
        //https://blog.csdn.net/fjl_CSDN/article/details/79019437
        page.addTargetRequests(page.getHtml().links().regex("https://blog.csdn.net/[a-zA-Z0-
9_]+/article/details/[0-9]{8}").all());
        //保存抓取的结果


        page.putField("title",page.getHtml().xpath("//*
```

```
[@id=\"mainBox\"]/main/div[1]/div/div/div[1]/h1/text()").toString());
    }

    @Override
    //配置
    public Site getSite() {
        Site site = new Site();
        //设置失败后重试时长为3s，设置两个页面之间爬去间隔为0.1s
        site.setRetryTimes(3000).setSleepTime(100);
        return site;
    }


}
```

- CrawlerTask任务类

```
public class CrawlerTask {
   public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .addPipeline(new ConsolePipeline()) //向控制台输出
                .run();
    }
}
```

3.2FilePipeline 文件保存

我们可以通过FilePipeline对象以文件方式保存爬取的内容(文件保存在本地,不适合分布式)

```
package com.itheima.crawler;

import us.codecraft.webmagic.Page;
import us.codecraft.webmagic.Site;
import us.codecraft.webmagic.Spider;
import us.codecraft.webmagic.pipeline.ConsolePipeline;
import us.codecraft.webmagic.pipeline.FilePipeline;
import us.codecraft.webmagic.processor.PageProcessor;

public class MyProcessor implements PageProcessor {

    @Override
    //分页页面，提取路径,数据和存储
    public void process(Page page) {
        //https://blog.csdn.net/fjl_CSDN/article/details/79019437
        page.addTargetRequests(page.getHtml().links().regex("https://blog.csdn.net/[a-zA-Z0-
9_]+/article/details/[0-9]{8}").all());
        //保存抓取的结果
        page.putField("title",page.getHtml().xpath("//*
[@id=\"mainBox\"]/main/div[1]/div/div/div[1]/h1/text()").toString());

    }
```

```
    @Override
    //配置
    public Site getSite() {
        Site site = new Site();
        //设置失败后重试时长为3s，设置两个页面之间爬去间隔为0.1s
        site.setRetryTimes(3000).setSleepTime(100);
        return site;
    }


}
```

- CrawlerTask任务类

```
    public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .addPipeline(new ConsolePipeline()) //向控制台输出
                .addPipeline(new FilePipeline("E:\\source\\tensquare\\crawler"))//以文件方式保存
                .run();
    }
```

3.3JsonFilePipeline

我们可以通过以JsonFilePipeline对象以json方式保存爬取的内容

```
package com.itheima.crawler;

import us.codecraft.webmagic.Page;
import us.codecraft.webmagic.Site;
import us.codecraft.webmagic.Spider;
import us.codecraft.webmagic.pipeline.ConsolePipeline;
import us.codecraft.webmagic.pipeline.FilePipeline;
import us.codecraft.webmagic.pipeline.JsonFilePipeline;
import us.codecraft.webmagic.processor.PageProcessor;

public class MyProcessor implements PageProcessor {

    @Override
    //分页页面，提取路径,数据和存储
    public void process(Page page) {
        //https://blog.csdn.net/fjl_CSDN/article/details/79019437
        page.addTargetRequests(page.getHtml().links().regex("https://blog.csdn.net/[a-zA-Z0-
9_]+/article/details/[0-9]{8}").all());
        //保存抓取的结果
        page.putField("title",page.getHtml().xpath("//*
[@id=\"mainBox\"]/main/div[1]/div/div/div[1]/h1/text()").toString());
    }


    @Override
```

```
    //配置
    public Site getSite() {
        Site site = new Site();
        //设置失败后重试时长为3s，设置两个页面之间爬去间隔为0.1s
        site.setRetryTimes(3000).setSleepTime(100);
        return site;
    }


}
```

- CrawlerTask任务类

```
    public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .addPipeline(new ConsolePipeline()) //向控制台输出
                .addPipeline(new FilePipeline("E:\\source\\tensquare\\crawler"))//以文件方式保存
                .addPipeline(new JsonFilePipeline("E:\\source\\tensquare\\crawler"))//以json文件
方式保存
                .run();
    }
```

3.4定制Pipeline

如果以上Pipeline都不能满足你的需要，你可以定制Pipeline

- 创建类MyPipeline实现接口Pipeline

```
/**
 * 自定义Pipeline
 */
public class MyPipeline implements Pipeline {

    @Override
    public void process(ResultItems resultItems, Task task) {
        //获得到title
        String title = resultItems.get("title");
        System.out.println("我自定义的类型title:" + title);
        //调用Dao存到数据库
    }
}
```

- 使用MyPipeline

```
    public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .addPipeline(new ConsolePipeline()) //向控制台输出
                .addPipeline(new FilePipeline("E:\\source\\tensquare\\crawler"))//以文件方式保存
                .addPipeline(new JsonFilePipeline("E:\\source\\tensquare\\crawler"))//以json文件
方式保存
                .addPipeline(new MyPipeline()) //自定义的Pipeline
                .run();
    }
```

4.Scheduler

我们刚才完成的功能，每次运行可能会爬取重复的页面，这样做是没有任何意义的。Scheduler(URL管理) 最基本的功能是实现对已经爬取的URL进行标示。可以实现URL的增量去重。 目前scheduler主要有三种实现方式：

内存队列 QueueScheduler

文件队列 FileCacheQueueScheduler

Redis队列 RedisScheduler

4.1内存队列QueueScheduler

使用setScheduler来设置Scheduler

```
    public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .setScheduler(new QueueScheduler()) //设置内存队列
                .run();
    }
```

4.2文件队列FileCacheQueueScheduler

使用文件保存抓取URL，可以在关闭程序并下次启动时，从之前抓取到的URL继续抓取

```
    public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .setScheduler(new FileCacheQueueScheduler("E:\\source\\tensquare\\crawler")) //
设置文件队列
                .run();
    }
```

运行后文件夹E:\source\tensquare\crawler会产生两个文件blog.csdn.net.urls.txt和blog.csdn.net.cursor.txt

4.3Redis队列RedisScheduler

使用Redis保存抓取队列，可进行多台机器同时合作抓取

- 运行redis服务端
- 修改代码

```
    public static void main(String[] args) {
        Spider.create(new MyProcessor())
                .addUrl("https://blog.csdn.net/nav/ai")
                .setScheduler(new RedisScheduler("127.0.0.1")) //设置Redis队列
                .run();
    }
```

第三章-十次方文章爬取　包名一定要是crawler

1.需求分析

每日某时间(凌晨0点)段整从CSDN播客中爬取文档，存入文章数据库中。

2.频道设置

频道名称	地址
资讯	https://blog.csdn.net/nav/news
人工智能	https://blog.csdn.net/nav/ai
区块链	https://blog.csdn.net/nav/blockchain
数据库	https://blog.csdn.net/nav/db
前端	https://blog.csdn.net/nav/web
编程语言	https://blog.csdn.net/nav/lang

- 向数据库tensquare_article的tb_channel表中添加记录

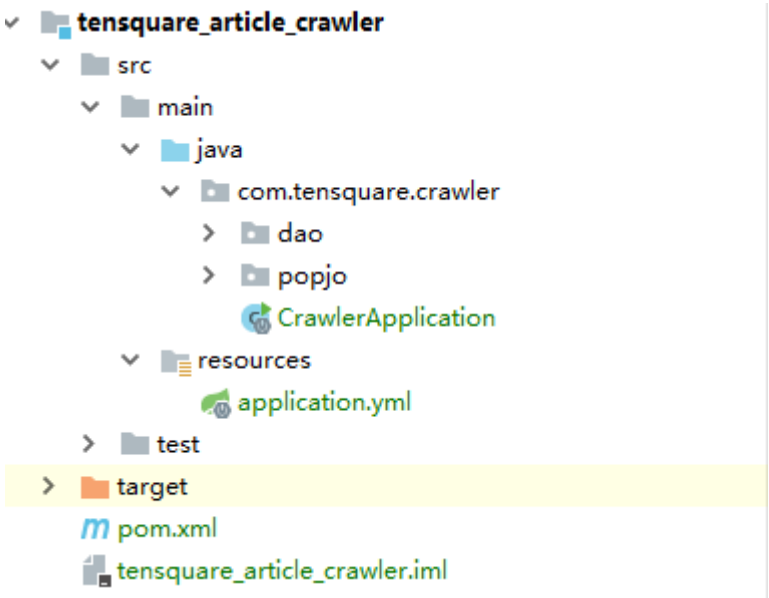
```
DROP TABLE IF EXISTS `tb_channel`;
CREATE TABLE `tb_channel` (
  `id` varchar(20) NOT NULL COMMENT 'ID',
  `name` varchar(100) DEFAULT NULL COMMENT '频道名称',
  `state` varchar(1) DEFAULT NULL COMMENT '状态',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='频道';

-- ----------------------------
-- Records of tb_channel
-- ----------------------------
INSERT INTO `tb_channel` VALUES ('ai', '人工智能', '1');
INSERT INTO `tb_channel` VALUES ('blockchain', '区块链', '1');
INSERT INTO `tb_channel` VALUES ('db', '数据库', '1');
```

```
INSERT INTO `tb_channel` VALUES ('lang', '编程语言', '1');
INSERT INTO `tb_channel` VALUES ('new', '资讯', '1');
INSERT INTO `tb_channel` VALUES ('web', '前端', '1');
```

id	name	state
new	资讯	1
ai	人工智能	1
blockchain	区块链	1
db	数据库	1
web	前端	1
lang	编程语言	1

3.代码编写

3.1模块搭建



- 创建模块tensquare_article_crawler ,引入依赖

```
    <dependencies>
        <dependency>
            <groupId>us.codecraft</groupId>
            <artifactId>webmagic-core</artifactId>
            <version>0.7.3</version>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>us.codecraft</groupId>

            <artifactId>webmagic-extension</artifactId>
```

```
            <version>0.7.3</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.tensquare</groupId>
            <artifactId>tensquare_common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
            <version>2.0</version>
        </dependency>
    </dependencies>
```

- 创建配置文件application.yml

```
server:
  port: 9014
spring:
  application:
    name: tensquare-crawler #指定服务名
  datasource:
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/tensquare_article?characterEncoding=UTF8
    username: root
    password: 123456
  jpa:
    database: MySQL
    show-sql: true
redis:
  host: 127.0.0.1
```

- 创建启动类

```
@EnableScheduling
@SpringBootApplication
public class CrawlerApplication {

    @Value("${redis.host}")
    private String redis_host;

    public static void main(String[] args) {

        SpringApplication.run(CrawlerApplication.class,args);
```

```
    }

    @Bean
    public IdWorker idWorker(){
        return  new IdWorker(1,1);
    }

    @Bean
    public RedisScheduler redisScheduler(){
        return new RedisScheduler(redis_host);
    }
}
```

- 实体类及数据访问接口

  拷贝文章微服务 的pojo和dao

3.2爬取类

- 创建文章爬取类ArticleProcessor

```
package com.tensquare.crawler.processor;

import org.springframework.stereotype.Component;
import us.codecraft.webmagic.Page;
import us.codecraft.webmagic.Site;
import us.codecraft.webmagic.processor.PageProcessor;

/**
 * 文章的爬取类
 */

@Component
public class ArticleProcessor implements PageProcessor {

    @Override
    public void process(Page page) {
        //只提取播客的文章详细页内容
        page.addTargetRequests(page.getHtml().links().regex("https://blog.csdn.net/[a-zA-Z0-
9_]+/article/details/[0-9]{8}").all());
        //把标题和文章内容获得到保存
        String title = page.getHtml().xpath("//*
[@id=\"mainBox\"]/main/div[1]/div/div/div[1]/h1/text()").toString();
        String content = page.getHtml().xpath("//*[@id=\"article_content\"]/div[2]").toString();
        System.out.println("title=" + title);
        System.out.println("content=" + content);

        if(title != null && content != null){
            System.out.println("title=" + title);
            System.out.println("content=" + content);
            page.putField("title",title);
            page.putField("content",content);

        }else{
```

```
            page.setSkip(true);
        }
    }
    @Override
    public Site getSite() {
        Site site = new Site();
        //设置失败后重试时长为3s，设置两个页面之间爬去间隔为0.1s
        site.setRetryTimes(3000).setSleepTime(100);
        return site;
    }
}
```

3.3入库类

创建文章入库类ArticleDbPipeline ，负责将爬取的数据存入数据库

```
package com.tensquare.crawler.pipeline;

import com.tensquare.crawler.dao.ArticleDao;
import com.tensquare.crawler.pojo.Article;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import us.codecraft.webmagic.ResultItems;
import us.codecraft.webmagic.Task;
import us.codecraft.webmagic.pipeline.Pipeline;
import util.IdWorker;

/**
 * 入库类
 */

@Component
public class ArticleDbPipeline implements Pipeline {

    @Autowired
    private ArticleDao articleDao;

    @Autowired
    private IdWorker idWorker;

    private String channelId;

    public void setChannelId(String channelId) {
        this.channelId = channelId;
    }

    @Override
    public void process(ResultItems resultItems, Task task) {
        String title = resultItems.get("title");
        String content = resultItems.get("content");


        Article article = new Article();
```

```
        article.setId(idWorker.nextId()+"");
        article.setTitle(title);
        article.setContent(content);
        article.setChannelid(channelId);

        articleDao.save(article);
    }
}
```

3.4任务类

```
@Component
public class ArticleTask {

    @Autowired
    private ArticleProcessor articleProcessor;

    @Autowired
    private ArticleDbPipeline articleDbPipeline;

    @Autowired
    private RedisScheduler redisScheduler;

    @Scheduled(cron="05 06 13 * * ?")
    public void aiTask(){
        Spider spider = Spider.create(articleProcessor);
        spider.addUrl("https://blog.csdn.net/nav/ai");
        articleDbPipeline.setChannelId("ai");
        spider.setScheduler(redisScheduler);
        spider.addPipeline(articleDbPipeline);
        spider.addPipeline(new ConsolePipeline());
        spider.start();
    }

    @Scheduled(cron="05 06 13 * * ?")
    public void dbTask(){
        Spider spider = Spider.create(articleProcessor);
        spider.addUrl("https://blog.csdn.net/nav/db");
        articleDbPipeline.setChannelId("db");
        spider.setScheduler(redisScheduler);
        spider.addPipeline(articleDbPipeline);
        spider.addPipeline(new ConsolePipeline());
        spider.start();
    }

}
```

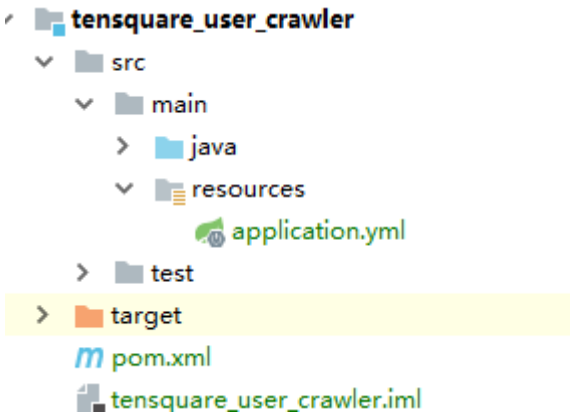
第四章-十次方用户数据爬取 包名一定要是crawler

1.需求分析

从csdn中爬取用户昵称和头像，存到用户表，头像图片存储到本地

2.代码编写 包名一定要是crawler

2.1模块搭建



- 创建工程tensquare_user_crawler。pom.xml引入依赖

```
<dependencies>
        <dependency>
            <groupId>us.codecraft</groupId>
            <artifactId>webmagic-core</artifactId>
            <version>0.7.3</version>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>us.codecraft</groupId>
            <artifactId>webmagic-extension</artifactId>
            <version>0.7.3</version>

        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.tensquare</groupId>
            <artifactId>tensquare_common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
            <version>2.0</version>
        </dependency>
    </dependencies>
```

- 创建配置文件application.yml

```
server:
  port: 9015
spring:
  application:
    name: tensquare-user-crawler #指定服务名
  datasource:
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/tensquare_user?characterEncoding=UTF8
    username: root
    password: 123456
  jpa:
    database: MySQL
    show-sql: true
redis:
  host: 127.0.0.1
```

- 创建启动类

```
@EnableScheduling
@SpringBootApplication
public class CrawlerApplication {
    @Value("${redis.host}")
    private String redisHost;

    public static void main(String[] args) {
        SpringApplication.run(CrawlerApplication.class, args);
    }
```

```
    @Bean
    public IdWorker idWorker() {
        return new IdWorker(1, 1);
    }

    @Bean
    public RedisScheduler redisScheduler() {
        return new RedisScheduler(redisHost);
    }

}
```

- 实体类及数据访问接口

  拷贝用户微服务 的pojo和dao

2.2爬取类

- 创建UserProcessor

```
@Component
public class UserProcessor implements PageProcessor {

    @Override
    //分析页面 存储
    public void process(Page page) {
        //只提取播客的文章详细页内容
        page.addTargetRequests(page.getHtml().links().regex("https://blog.csdn.net/[a-zA-Z0-9_]+/article/details/[0-9]{8}").all());
        //获得昵称
        String nickName =  page.getHtml().xpath("//*[@id=\"uid\"]/text()").toString();
        //获得图像
        String img =  page.getHtml().xpath("//*[@id=\"asideProfile\"]/div[1]/div[1]/a").css("img","src").toString();
        if(nickName != null && img != null){
            page.putField("nickName",nickName);
            page.putField("img",img);
        }else{
            page.setSkip(true);
        }
    }


    @Override
    //配置
    public Site getSite() {
        Site site = new Site();
        site.setRetryTimes(3).setSleepTime(3000);
        return site;
    }
}
```

2.3下载工具类

资源提供了工具类，拷贝至tensquare_common工程的util包下

```
package util;

import java.io.*;
import java.net.URL;
import java.net.URLConnection;

public class DownloadUtil {
    public static void download(String urlStr, String filename, String
            savePath) throws IOException {
        URL url = new URL(urlStr);
        //打开url连接
        URLConnection connection = url.openConnection();
        //请求超时时间
        connection.setConnectTimeout(5000);
        //输入流
        InputStream in = connection.getInputStream();
        //缓冲数据
        byte[] bytes = new byte[1024];
        //数据长度
        int len;
        //文件
        File file = new File(savePath);
        if (!file.exists())
            file.mkdirs();
        OutputStream out = new FileOutputStream(file.getPath() + "\\" + filename);
        //先读到bytes中
        while ((len = in.read(bytes)) !=-1){
            //再从bytes中写入文件
            out.write(bytes, 0, len);
        }
        //关闭IO
        out.close();
        in.close();
    }

}
```

2.4入库类

创建UserPipeline

```
@Component
public class UserPipeline implements Pipeline {

    @Autowired
    private IdWorker idWorker;

    @Autowired
```

```
    private UserDao userDao;


    @Override
    public void process(ResultItems resultItems, Task task) {
        String nickName =  resultItems.get("nickName");
        String img =  resultItems.get("img");
//https://avatar.csdn.net/4/4/E/3_baidu_20183817.jpg


        String fileName= img.substring(img.lastIndexOf("/")+1);

        //保存用户到数据库
        User user = new User();
        user.setId(idWorker.nextId()+"");
        user.setNickname(nickName);
        user.setAvatar(fileName);
        userDao.save(user);

        //下载文件到磁盘
        try {
            DownloadUtil.download(img,fileName,"E:/source/tensquare/img");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2.5任务类

```
@Component
public class UserTask {

    @Autowired
    private UserProcessor userProcessor;

    @Autowired
    private UserPipeline userPipeline;

    @Autowired
    private RedisScheduler redisScheduler;

    @Scheduled(cron = "50 13 15 * * ?")
    public  void userTask(){
        Spider spider = Spider.create(userProcessor);
        spider.addUrl("https://blog.csdn.net");
        spider.addPipeline(userPipeline);
        spider.addPipeline(new ConsolePipeline());
        spider.setScheduler(redisScheduler);
        spider.start();


    }
```

```
}
```

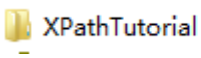
附录

1.正则语法

符号	作用
\d	数字
\D	非数字
\w	单词：a-zA-Z0-9_
\W	非单词
.	通配符，匹配任意字符
{n}	匹配n次
{n,}	大于或等于n次
{n,m}	在n次和m次之间
+	1~n次
*	0~n次
?	0~1次
^	匹配开头
$	匹配结尾
[a-zA-Z]	英文字母
[a-zA-Z0-9]	英文字母和数字
[*xyz*]	字符集合, 匹配所包含的任意一个字符

2.xpath语法

- 获得标签请见文档



- 常用的功能函数

表达式	描述	用法	说明
startswith	选取id值以ma开头的div节点	xpath('//div[startswith(@id,"ma")]')	选取id值以ma开头的div节点
contains	选取id值包含ma的div节点	xpath('//div[contains(@id,"ma")]')	选 ma取的idd值iv节包点含
and	选取id值包含ma和in的div节点	xpath('//div[contains(@id,"ma")and contains(@id,"in")]')	选取id值包含ma和in的div节点
text()	选取节点文本包含ma的div节点	xpath('//div[contains(text(),"ma")]')	选取节点文本包含ma的div节点
css(标签名,属性名)	获得某个标签的属性值	css("img","src")	获得img标签的src的属性值

3.cron表达式语法

- 格式 (cron在线程生成http://cron.qqe2.com/)

```
{秒数} {分钟} {小时} {日期} {月份} {星期} {年份(可为空)}
```

- 基本语法

字段	允许值	允许的特殊字符
秒	0-59	, - * /
分	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? / L W C
月份	1-12	, - * /
星期	0-7或SUN-SAT 0,7是SUN	, - * ? / L C #

- 特殊字符

特殊字符	代表含义
,	枚举
-	区间
*	任意
/	步长
?	日/星期冲突匹配
L	最后
W	工作日
C	和calendar联系后计算过的值
#	星期，4#2，第2个星期四