day08_爬虫系统搭建

目标

- 了解爬虫是什么
- 了解webmagic及其四大组件
- 了解爬虫系统中的ip代理
- 能够导入爬虫系统
- 知道文档下载和文档解析的思路

1爬虫是什么

网络爬虫（Web crawler)，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，它们被广泛用于互联网搜索引擎或其他类似网站，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。

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。

通俗理解：爬虫是一个模拟人类请求网站行为的程序。可以自动请求网页、把数据抓取下来，然后使用一定的规则提取有价值的数据。

聚焦爬虫：通常我们自己撸的为聚焦爬虫面向主题爬虫、面向需求爬虫：会针对某种特定的能容去爬取信息，而且保证内容需求尽可能相关

2名词解释

2.1 Webmagic：

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。

1、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。

2、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。

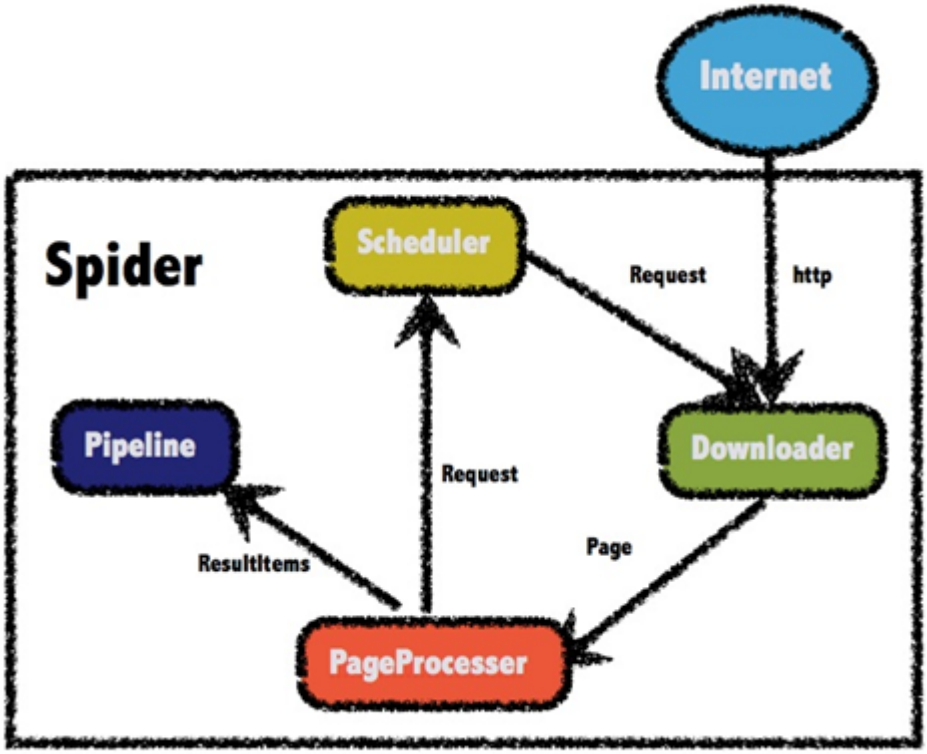
3、WebMagic的核心在webmagic-core包中，其他的包你可以理解为对WebMagic的一个扩展——这和作为用户编写一个扩展是没有什么区别的。

4、虽然核心需要足够简单，但是WebMagic也以扩展的方式，实现了很多可以帮助开发的便捷功能。例如基于注解模式的爬虫开发，以及扩展了XPath语法的Xsoup等。这些功能在WebMagic中是可选的，它们的开发目标，就是让使用者开发爬虫尽可能的简单，尽可能的易维护。

2.2 webmagic的总体架构：

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。



2.3 webmagic的总体架构的四大组件

2.3.1 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。

2.3.2 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。

2.3.3 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。

2.3.4 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。

更多内容可以查看官网文档 http://webmagic.io/docs/zh/

2.4代理IP：

当我们对某些网站进行爬去的时候，我们经常会换IP来避免爬虫程序被封锁。其实也是一个比较简单的操作，目前网络上有很多IP代理商，例如西刺，芝麻，犀牛等等。这些代理商一般都会提供透明代理，匿名代理，高匿代理。

2.4.1代理IP类型：

代理IP一共可以分成4种类型。前面提到过的透明代理IP，匿名代理IP，高匿名代理IP，还有一种就是混淆代理IP。最基础的安全程度来说呢，他们的排列顺序应该是这个样子的高匿 > 混淆 > 匿名 > 透明。

2.5 Selenium方式下载页面：

Selenium 是一个用于 Web 应用程序测试的工具。它的优点在于，浏览器能打开的页面，使用 selenium 就一定能获取到。但 selenium 也有其局限性，相对于脚本方式，selenium 获取内容的效率不高。

我们主要使用它可以调用chrome浏览器来获取必须要的Cookie，因为csdn的cookie通过js来生成的，需要浏览器才能得到Cookie

2.5.1chrome的无头（headless）模式：

在 Chrome 59中开始搭载Headless Chrome。这是一种在无需显示headless的环境下运行 Chrome 浏览器的方式。从本质上来说，就是不用 chrome 浏览器来运行 Chrome 的功能！它将 Chromium 和 Blink 渲染引擎提供的所有现代 Web 平台的功能都带入了命令行。

由于存在大量的网页是动态生成的，在使用浏览器查看源代码之后，发现网页dom只有一个root元根元素和一堆js引用，根本看不到网页的实际内容，因此，爬虫不仅需要把网页下载下来，还需要运行JS解析器，将网站呈现出最终的效果。

在Headless出现之前，主要流行的是PhantomJS这个库，原理是模拟成一个实际的浏览器去加载网站。Headless Chome出现之后，PhantomJS地位开始不保。毕竟Headless Chome本身是一个真正的浏览器，支持所有chrome特性，而PhantomJS只是模拟，因此Headless Chome更具优势

2.5.2 webdriver

WebDriver针对各个浏览器而开发，取代了嵌入到被测Web应用中的JavaScript。与浏览器的紧密集成支持创建更高级的测试，避免了JavaScript安全模型导致的限制。除了来自浏览器厂商的支持，

成支持创建更高级的测试，避免了JavaScript安全模型导致的限制。除了来自浏览器厂商的支持，WebDriver还利用操作系统级的调用模拟用户输入。WebDriver支持Firefox(FirefoxDriver)、IE(InternetExplorerDriver)、Opera (OperaDriver)和Chrome (ChromeDriver)。 它还支持Android (AndroidDriver)和iPhone (IPhoneDriver)的移动应用测试。它还包括一个基于HtmlUnit的无界面实现，称为HtmlUnitDriver。WebDriver API可以通过Python、Ruby、Java和C#访问，支持开发人员使用他们偏爱的编程语言来创建测试。

2.5.3 ChromeDriver下载

ChromeDriver 是 google 为网站开发人员提供的自动化测试接口，它是 **selenium2** 和 **chrome浏览器** 进行通信的桥梁。selenium 通过一套协议（JsonWireProtocol ：https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol）和 ChromeDriver 进行通信，selenium 实质上是对这套协议的底层封装，同时提供外部 WebDriver 的上层调用类库。

和chrome版本的对应关系

chromedriver 版本	chrome 版本
ChromeDriver 2.36	Chrome v63-65
ChromeDriver 2.35	Chrome v62-64
ChromeDriver 2.34	Chrome v61-63
ChromeDriver 2.33	Chrome v60-62

下载地址如下

http://npm.taobao.org/mirrors/chromedriver/

详细内容可以查看 https://www.jianshu.com/p/31c8c9de8fcd

2.6 Xpath

2.6.1 Xpath是什么

xpath是一种在xml中查找信息的语言，普遍应用于xml中，在类xml的html中也可以使用，在selenium自动化中起核心作用，是写selenium自动化脚本的基础。

2.6.2 Xpath的定位

xpath的定位主要由路径定位、标签定位、轴定位组合构成，外加筛选功能进行辅助，几乎可以定位到任意元素

(1)标签定位

通过标签名即可找到文档中所有满足的标签元素，如：

xpath	说明
div	找到所有的div标签元素
input	找到所有的input标签元素
*	替代任意元素或属性
@属性名	找到指定名称的属性

(2)路径定位

通过路径描述来找到需要的元素，“/”开头表示从根路径开始，其他位置表示子元素或分隔符；“//”表示后代元素；“..”表示父元素（上一级）；“.”表示当前元素；“|”表示多条路径

xpath	说明
/html	找到根元素html
//div	找到所有的div元素
//div[@id='id1']/span	找到id=“id1”的div元素的子元素span
//div[@id='id1']//span	找到id=“id1”的div元素下的所有后代元素span
//div[@id='id1']/@class	找到id=“id1”的div元素的class属性
//div[@id='id1']/span|//div[@id='id2']/span	找到id=“id1”和id=“id2”的div元素的子元素span

(3)轴定位

通过轴运算符加上“::”和“标签”，找到需要的元素，类似路径定位，如：

xpath	说明
//div[@id='id1']/child::span	找到id=“id1”的div元素的子元素span， 同//div[@id='id1']/span
//div[@id='id1']/attribute::class	找到id=“id1”的div元素的class属性， 同//div[@id='id1']/@class
//div[@id='id1']/preceding-sibling::*	找到与id=“id1”的div元素同级别的，且在它之前的所有元素

下表是轴运算符的列表

轴名称	结果
ancestor	选取当前节点的所有先辈（父、祖父等）
ancestor-or-self	选取当前节点的所有先辈（父、祖父等）以及当前节点本身
attribute	选取当前节点的所有属性
child	选取当前节点的所有子元素
descendant	选取当前节点的所有后代元素（子、孙等）。
descendant-or-self	选取当前节点的所有后代元素（子、孙等）以及当前节点本身。
following	选取文档中当前节点的结束标签之后的所有节点。
namespace	选取当前节点的所有命名空间节点
parent	选取当前节点的父节点。
preceding	选取文档中当前节点的开始标签之前的所有节点。
preceding-sibling	选取当前节点之前的所有同级节点。
following-sibling	选取当前节点之后的所有同级节点。
Self	选取当前节点

一般情况下，我们使用简写后的语法。虽然完整的轴描述是一种更加贴近人类语言，利用自然语言的单词和语法来书写的描述方式，但是相比之下也更加啰嗦。

(4)筛选

通过以上方法找出来的元素会找到很多你本意不需要的元素，因此还需要通过一些筛选运算来找到对应的元素，筛选方式多种多样，下面的各种例子助你定位又快又准。

通用的筛选条件是以[xxxx]形式出现的（上面的例子中已有体现），常见筛选如下：

- 属性筛选：

属性名前+@来表示属性，如下

xpath	说明
//div[@class='class1']	筛选class属性值等于class1的div
//div[@hight>10]	筛选hight属性值大于10的div(仅限数字)
//div[text()='divtext']	筛选文本是divtext的div
//div[contains(@class,'class1')]	筛选class属性中包含class1的div
//div[contains(text(),'text1')]	筛选文本包含text1的div
//div[text()='text1' and @class='class1']	同时满足两个条件的筛选，类似的，“或者” 的话用“or”，运算优先级高的用"()"括起来
//div[text()='text1' and not(@class)]	筛选文本包含 text1，且无class属性的 div

- 序号筛选：

通过序号（从1开始），或排序运算查找元素

xpath	说明
//div[@id='id1']/span[1]	找到id=“id1”的div元素后代的第一个span元素，如[4]则是第4个
//div[@id='id1']/span[last()]	找到id=“id1”的div元素子元素的最后一个span元素，如[last()-2]则是倒数第3个
//div[@id='id1']/span[position()>2 and position() < 7]	找到id=“id1”的div元素后代的第3、4、5、6个span元素
//div[@id='id1']/text()[2]	找到id=“id1”的div元素的第二段文本（注：此处用于文本被子元素分割，需要选择后面文本的情况：如 this is text one**haha**this is text two ）

特别注意：序号筛选时，指定是当前元素的同级的第n个，如果当前元素的祖先中有元素不是唯一的，那么序号筛选是无效的。

通过括号将祖先括起来，再指定序号，可以使当前元素前的祖先是指定的、且唯一的，如：

(//div[@class='class1']//span[@class='class2'])[1]/div[3]

这样就可以十分准确的定位到需要span下的第3个div，没有此括号，
当//div[@class='class1']//span[@class='class2']找到多个元素时，就算用[3]也则只能定位到第1个

2.7 Cron表达式

命令常见于Unix和类Unix的操作系统之中，用于设置周期性被执行的指令。

2.7.1 Cron表达式的形式

```
@Component
public class ScheduleTask {
    @Scheduled(cron = "0/10 * * * * ?")
}
```

Cron表达式是一个字符串，字符串以5或6个空格隔开，分开 6或*7个域，每一个域代表一个含义,Cron有如下两种语法格式：

2.7.2 cron表达式格式

Seconds Minutes Hours DayofMonth Month DayofWeek Year

或

Seconds Minutes Hours DayofMonth Month DayofWeek

(1)每一个域可出现的字符如下：

Seconds可出现 ： - * / , 四个字符，有效范围为0-59的整数 **Minutes**可出现： - * / , 四个字符，有效范围为0-59的整数 **Hours****可出现：- * / , 四个字符，有效范围为0-23的整数** DayofMonth**可出现： - * / , ? L W C八个字符，有效范围为1-31的整数** Month**可出现：- * / , 四个字符，有效范围为1-12的整数或JAN-DEc** DayofWeek**可出现：- * / , ? L C #四个字符，有效范围为1-7的整数或SUN-SAT两个范围。1表示星期天，2表示星期一， 依次类推** Year**可出现： - * / , 四个字符，有效范围为1970-2099年

(2)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 或者 JAN-DEC , - * / **星期** 1-7 或者 SUN-SAT , - * / ? L C # **年**（可选） 留空, 1970-2099 , - * /

(3)一些示例帮助理解

0 0 10,14,16 * * ? 每天上午10点，下午2点，4点 0 0/30 9-17 * * ? 朝九晚五工作时间内每半小时 0 0 12 ? * WED 表示每个星期三中午12点 "0 0 12 * * ?" 每天中午12点触发 "0 15 10 ? * *" 每天上午10:15触发 "0 15 10 * * ?" 每天上午10:15触发 "0 15 10 * * ? *" 每天上午10:15触发 "0 15 10 * * ? 2005" 2005年的每天上午10:15触发 "0 * 14 * * ?" 在每天下午2点到下午2:59期间的每1分钟触发 "0 0/5 14 * * ?" 在每天下午2点到下午2:55期间的每5分钟触发 "0 0/5 14,18 * * ?" 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发 "0 0-5 14 * * ?" 在每天下午2点到下午2:05期间的每1分钟触发 "0 10,44 14 ? 3 WED" 每年三月的星期三的下午2:10和2:44触发 "0 15 10 ? * MON-FRI" 周一至周五的上午10:15触发 "0 15 10 15 * ?" 每月15日上午10:15触发 "0 15 10 L * ?" 每月最后一日的上午10:15触发 "0 15 10 ? * 6L" 每月的最后一个星期五上午10:15触发 "0 15 10 ? * 6L 2002-2005" 2002年至2005年的每月的最后一个星期五上午10:15触发 "0 15 10 ? * 6#3" 每月的第三个星期五上午10:15触发

(4)cron在线生成

理解以后觉得翻阅资料很麻烦?cron在线生成器帮助你 cron在线生成工具地址：
http://cron.qqe2.com/

3设计思路

1、配置初始化的URL，首先访问初始化的URL，先解析初始URL,并获取需要筛选的用户空间的链接

2、将用户空间的URL链接交给WebMagic进行数据抓取，并进行分页处理，获取有效的文章链接。

3、将文章交给WebMagic 进行数据抓取，如果抓取过程中出现失败，则采用selenium+Chrome 的方式抓取页面，并进行cookie重置

4、解析完成后得到Html页面交给下一级解析器进行数据解析，得到需要的数据，并将数据封装成固定的格式进行存储

5、定时任务定期对点击量比较高的数据进行重新抓取并更新数据。

4 需求分析

4.1 功能需求

为黑马头条提供大量的数据积累，使用爬虫对CSDN的大量博客内容进行抓取，提升黑马头条的数据量以及点击量，为以后的大数据采集提供前置数据。

4.1.1 CSDN爬虫需求

- 获取CSDN文章的 标题、作者内容、发布日期、文章来源，阅读量，评论数据
- 将文章内容按照图片以及文本的方式进行存储，存储格式如下

```
[
    {
        type: 'text',
        value: 'text'
    },
    {
        type: 'image',
        value: 'https://p3.pstatp.com/large/pgc-image/RVFRw8xCiUeTbd',
        style:{
            height:'810px'
        }
    }
]
```

- 文章可能存在多条评论，将评论数据进行存储
- 要进行代理IP的自管理，即自动进行代理IP的抓取以及定时检查无效代理IP，并进行删除，实时保证代理IP库是可用的。

4.1.2 爬虫常见问题

- CSDN使用混淆加密js设置cookie，浏览器才能解析，无法进行人工还原算法，没有办法手动获取cookie并进行注入，所以导致访问被拦截

  解决方案：使用selenium+chromedriver 先通过chrome的headless(无头) 方式进行进行访问浏览器，获取cookie以及内容，更新cookie后就可以进行正常访问了。

- CSDN获取首页数据比较麻烦。

  解决方案，分三步，第一步获取初始化的URL,解析用户空间，然后处理分页数据，最后获取最终的文章URL。

5 导入heima-leadnews-crawler项目

资料文件夹中导入项目：heima-leadnews-crawler

6 爬虫服务的初始化工作

以上介绍了爬虫所需要的一些技术以及常用的组件以及工具类，对常用的结构也做了一些了解，下来开始讲解下详细的工作流程

6.1初始化URL的获取

因为CSDN的初始化URL是有规律的，例如https://www.csdn.net/nav/java，https://www.csdn.net/nav/arch，等，我们将需要爬取类型的专栏配置即可。

有了初始化的URL我们需要配置以下，在crawler.properties配置文件配置即可

```
crawler.init.url.prefix=https://www.csdn.net/nav/
crawler.init.url.suffix=java,web,arch,db,mobile,ops,sec,cloud,engineering,iot,fund,career
```

因为前缀都一样，我们采用两个字段存储，这样可以更简单的配置。

系统启动的时候首先会获取配置的字段进行拼接

在com.heima.crawler.config.CrawlerConfig中读取拼接url列表

```
@Configuration
@Log4j2
@Getter
@Setter
@PropertySource("classpath:crawler.properties")
@ConfigurationProperties(prefix = "crawler.init.url")
public class CrawlerConfig {

    private String prefix;
    private String suffix;

    /**
     * 拼接初始化的URL
     * @return
     */
    public List<String> getInitCrawlerUrlList() {
        List<String> initCrawlerUrlList = null;
        if (StringUtils.isNotEmpty(suffix)) {
            String[] initCrawlerUrlArray = suffix.split(",");
            if (null != initCrawlerUrlArray && initCrawlerUrlArray.length > 0) {
                for (int i = 0; i < initCrawlerUrlArray.length; i++) {
                    String initUrl = initCrawlerUrlArray[i];
```

```
                    if (StringUtils.isNotEmpty(initUrl)) {
                        if (!initUrl.toLowerCase().startsWith("http")) {
                            initUrl = prefix + initUrl;
                            initCrawlerUrlArray[i] = initUrl;
                        }
                    }
                }
            }
            initCrawlerUrlList =
Arrays.asList(initCrawlerUrlArray).stream().filter(x ->
StringUtils.isNotEmpty(x)).collect(Collectors.toList());
        }
        return initCrawlerUrlList;
    }

}
```

组件中初始化url

com.heima.crawler.process.original.AbstractOriginalDataProcess抽象类

```
public abstract class AbstractOriginalDataProcess extends AbstractProcessFlow {
    @Override
    public void handel(ProcessFlowData processFlowData) {

    }

    @Override
    public CrawlerEnum.ComponentType getComponentType() {
        return null;
    }

    /**
     * 解析初始的数据
     *
     * @return
     */
    public abstract List<ParseItem> parseOriginalRequestData(ProcessFlowData
processFlowData);
}
```

编写实现类，把url列表转换为对象

com.heima.crawler.process.original.impl.CsdnOriginalDataProcess

```
@Component
@Log4j2
public class CsdnOriginalDataProcess extends AbstractOriginalDataProcess {

    @Autowired
    private CrawlerConfig crawlerConfig;

    @Override
    public List<ParseItem> parseOriginalRequestData(ProcessFlowData
processFlowData) {
        List<ParseItem> parseItemList = null;
        //从crawlerConfigProperty 中获取初始化URL列表
```

```
        List<String> initCrawlerUrlList = crawlerConfig.getInitCrawlerUrlList();
        if (null != initCrawlerUrlList && !initCrawlerUrlList.isEmpty()) {
            parseItemList = initCrawlerUrlList.stream().map(url -> {
                CrawlerParseItem parseItem = new CrawlerParseItem();
                url = url + "?rnd=" + System.currentTimeMillis();
                parseItem.setUrl(url);
                parseItem.setDocumentType(CrawlerEnum.DocumentType.INIT.name());
                parseItem.setHandelType(processFlowData.getHandelType().name());
                log.info("初始化URL:{}", url);
                return parseItem;
            }).collect(Collectors.toList());
        }
        return parseItemList;
    }

    //优先级
    @Override
    public int getPriority() {
        return 10;
    }
}
```

测试类

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class CsdnOriginalDataProcessTest {
    @Autowired
    private CsdnOriginalDataProcess csdnOriginalDataProcess;

    @Test
    public void test(){
        List<ParseItem> parseItems =
                csdnOriginalDataProcess.parseOriginalRequestData(new
ProcessFlowData());
        System.out.println(parseItems);
    }
}
```

6.2 下载

> 解析完数据后就需要进行下载操作

6.2.1前置工作

crawler.properties

判断是否下载成功的cookiename

页面加载完成以后，js代码写入到cookie，用来验证区分人或机器访问，csdn网站反爬虫的一种方式

```
crux.cookie.name=acw_sc__v2
```

爬虫代理IP的实体类

com.heima.model.crawler.core.proxy.CrawlerProxy

```java
/**
 * 代理IP实体类
 */
@Setter
@Getter
public class CrawlerProxy implements Serializable {


    public CrawlerProxy(String host, Integer port) {
        this.host = host;
        this.port = port;
    }

    private String host;

    private Integer port;


    /**
     * 获取代理信息
     *
     * @return
     */
    public String getProxyInfo() {

        return this.host + ":" + port;
    }


    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        CrawlerProxy that = (CrawlerProxy) o;
        return host.equals(that.host) &&
                port.equals(that.port);
    }

    @Override
    public int hashCode() {
        return Objects.hash(host, port);
    }

    @Override
    public String toString() {
        return "CrawlerProxy{" +
                "host='" + host + '\'' +
                ", port=" + port +
                '}';
    }
}
```

爬虫代理IP工厂类

为了将爬虫的代理IP装换位需要类型的代理类型

com.heima.crawler.factory.CrawlerProxyFactory

```
/**
 * 代理工厂
 */
public class CrawlerProxyFactory {

    /**
     * 不使用代理
     */
    private static final String NOT_USE_PROXY = "NOT_USE_PROXY";


    /**
     * 代理对象httpclient的代理
     * @param crawlerProxy
     * @return
     */
    public static HttpHost getHttpHostProxy(CrawlerProxy crawlerProxy) {
        if (null != crawlerProxy &&
StringUtils.isNotEmpty(crawlerProxy.getHost()) && null !=
crawlerProxy.getPort()) {
            return new HttpHost(crawlerProxy.getHost(), crawlerProxy.getPort());
        }
        return null;
    }

    /**
     * 获取webmagic 代理对象
     *
     * @return
     */
    public static us.codecraft.webmagic.proxy.Proxy
getWebmagicProxy(CrawlerProxy crawlerProxy) {
        if (null != crawlerProxy &&
StringUtils.isNotEmpty(crawlerProxy.getHost()) && null !=
crawlerProxy.getPort()) {
            return new us.codecraft.webmagic.proxy.Proxy(crawlerProxy.getHost(),
crawlerProxy.getPort());
        }
        return null;
    }

    /**
     * 获取selenium Cookie
     *
     * @return
     */
    public static org.openqa.selenium.Proxy getSeleniumProxy(CrawlerProxy
crawlerProxy) {
        if (null != crawlerProxy &&
StringUtils.isNotEmpty(crawlerProxy.getHost()) && null !=
crawlerProxy.getPort()) {
            org.openqa.selenium.Proxy proxy = new org.openqa.selenium.Proxy();
            proxy.setHttpProxy(crawlerProxy.getProxyInfo());
```

```
            return proxy;
        }
        return null;
    }

    /**
     * 获取ProxyInfo 信息
     *
     * @param proxy
     * @return
     */
    public static String getCrawlerProxyInfo(CrawlerProxy proxy) {
        String proxyInfo = NOT_USE_PROXY;
        if (null != proxy) {
            proxyInfo = proxy.getProxyInfo();
        }
        return proxyInfo;
    }
}
```

爬虫Cookie实体类

com.heima.model.crawler.core.cookie.CrawlerCookie

下载页面的时候需要往cookie设值，不需要自己管理

```
public class CrawlerCookie {

    public CrawlerCookie() {
    }

    public CrawlerCookie(String name, boolean isRequired) {
        this.name = name;
        this.isRequired = isRequired;
    }

    /**
     * cookie名称
     */
    private String name;
    /**
     * cookie 值
     */
    private String value;
    /**
     * 域名
     */
    private String domain;
    /**
     * 路径
     */
    private String path;


    /**
     * 过期时间
```

```
     */
    private Date expire;

    /**
     * 是否是必须的
     */
    private boolean isRequired;

    /**
     * 校验是否过期
     *
     * @return
     */
    public boolean isExpire() {
        boolean flag = false;
        if (null != expire) {
            flag = expire.getTime() <= (new Date()).getTime();
        }
        return flag;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public String getDomain() {
        return domain;
    }

    public void setDomain(String domain) {
        this.domain = domain;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public Date getExpire() {
        return expire;
    }

    public void setExpire(Date expire) {
```

```
        this.expire = expire;
    }

    public boolean isRequired() {
        return isRequired;
    }

    public void setRequired(boolean required) {
        isRequired = required;
    }

    @Override
    public String toString() {
        return "CrawlerCookie{" +
                "name='" + name + '\'' +
                ", value='" + value + '\'' +
                ", domain='" + domain + '\'' +
                '}';
    }
}
```

延时回调接口

为了判断下载页面是否成功，因为csdn的cookie登录验证是通过js实现的，需要通过Selenium下载页面后等待一会检测cookie是否注入成功

com.heima.model.crawler.core.callback.DelayedCallBack

```
/**
 * 延时回调接口
 */
public interface DelayedCallBack {
    /**
     * 延时调用方法
     *
     * @param time
     * @return
     */
    public Object callBack(long time);

    /**
     * 判断是否存在
     *
     * @return
     */
    public boolean isExist();


    /**
     * 获取每次睡眠时间
     *
     * @return
     */
    public long sleepTime();
```

```
    /**
     * 超时时间
     *
     * @return
     */
    public long timeOut();
}
```

延时调用工具类

com.heima.model.crawler.core.delayed.DelayedUtils

```
public class DelayedUtils {

    public static void delayed(long delayedTime) {
        try {
            Thread.sleep(delayedTime);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    /**
     * 延时方法
     *
     * @param callBack
     */
    public static Object delayed(DelayedCallBack callBack) {
        boolean flag = false;
        long sleepTime = callBack.sleepTime();
        long timeOut = callBack.timeOut();
        long currentTime = System.currentTimeMillis();
        Object obj = null;
        while (true) {
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            long duration = System.currentTimeMillis() - currentTime;
            boolean isExist = callBack.isExist();
            obj = callBack.callBack(duration);
            if (isExist) {
                flag = true;
            } else if (duration > timeOut) {
                flag = true;
            }
            if (flag) {
                break;
            }
        }
        return obj;
    }


    /**
```

```
     * 并发过滤
        如果有多个并发，只取第1个
     */
    public synchronized static ConcurrentCallBack getConcurrentFilter(final long
time) {
        final ConcurrentEntity concurrentEntity = new ConcurrentEntity();
        concurrentEntity.setTimeInterval(time);
        return new ConcurrentCallBack() {
            public boolean filter() {
                boolean flag = false;
                //数据初始化
                long duration = System.currentTimeMillis() -
concurrentEntity.getCurrentTime();
                if (duration > time) {
                    concurrentEntity.setAvailable(true);
                    concurrentEntity.setCallCount(0);
                    concurrentEntity.setCurrentTime(System.currentTimeMillis());
                }
                long callCount = concurrentEntity.getCallCount();
                concurrentEntity.setCallCount(++callCount);
                if (callCount <= 1 && concurrentEntity.isAvailable()) {
                    flag = true;
                    concurrentEntity.setAvailable(false);
                }
                return flag;
            }
        };
    }


    static class ConcurrentEntity {

        /**
         * 当前时间
         */
        private long currentTime = System.currentTimeMillis();
        /**
         * 时间区间
         */
        private long timeInterval = 10000;

        /**
         * 是否可用
         */
        private boolean available = true;

        /**
         * 调用次数
         */
        private long callCount = 0;

        public long getCurrentTime() {
            return currentTime;
        }

        public void setCurrentTime(long currentTime) {
            this.currentTime = currentTime;
        }
```

```
        public long getTimeInterval() {
            return timeInterval;
        }

        public void setTimeInterval(long timeInterval) {
            this.timeInterval = timeInterval;
        }

        public boolean isAvailable() {
            return available;
        }

        public void setAvailable(boolean available) {
            this.available = available;
        }

        public long getCallCount() {
            return callCount;
        }

        public void setCallCount(long callCount) {
            this.callCount = callCount;
        }
    }
}
```

6.2.2 SeleniumClient 下载页面

crawler.properties

配置chrome驱动位置

选择某一种 window说着linux

```
# windows
webdriver.chrome.driver=D:/Program Files/chromedriver/chromedriver.exe
# linux
#webdriver.chrome.driver=/usr/local/sbin/chromeDriver/chromedriver
```

SeleniumClient下载工具类 通过Selenium下载页面

com.heima.crawler.utils.SeleniumClient

```
/**
 * Selenium 工具类封装
 * 使用 selenium+webDriver+headless Chrome 方式下载数据
 * <p>
 * 使用了 无头浏览器模式 回调模式以及延时模式
 */

@Log4j2
public class SeleniumClient {
```

```
    /**
     * 默认超时时间
     */
    private static final long timeOut = 10000;
    /**
     * 睡眠时间
     */
    private static final long sleepTime = 1000;
    /**
     * 读取crawler.properties 配置文件
     */
    private static final ResourceBundle resourceBundle =
ResourceBundle.getBundle("crawler");

    /**
     * json数据的xpath 表达式
     */
    public static final String SELENIUM_JSON_DATA_XPATH = "//pre/text()";


    /**
     * 创建ChromeDriver 驱动
     *
     * @param proxy 代理服务
     * @return
     */
    private SeleniumDriver initChromeDriver(Proxy proxy) {
        log.info("开始创建Chrome驱动");
        SeleniumDriver seleniumDriver = new SeleniumDriver();
        long currentTime = System.currentTimeMillis();
        //创建service服务
        ChromeDriverService chromeDriverService = new
ChromeDriverService.Builder().usingDriverExecutable(new
File(resourceBundle.getString("webdriver.chrome.driver"))).usingAnyFreePort().bu
ild();
        ChromeOptions chromeOptions = getChromeOptions(proxy);
        //启动一个 chrome 实例
        WebDriver webDriver = new ChromeDriver(chromeDriverService,
chromeOptions);
        seleniumDriver.setChromeDriverService(chromeDriverService);
        seleniumDriver.setWebDriver(webDriver);
        log.info("创建Chrome驱动完成，耗时：" + (System.currentTimeMillis() -
currentTime));

        return seleniumDriver;
    }

    /**
     * 获取Chrome 配置项
     *
     * @param proxy
     * @return
     */
    private ChromeOptions getChromeOptions(Proxy proxy) {
        //获取chrome驱动的位置
        ChromeOptions chromeOptions = new ChromeOptions();
        //设置无头模式
        chromeOptions.setHeadless(Boolean.TRUE);
```

```
        //不使用沙箱运行
        chromeOptions.addArguments("--no-sandbox");
        chromeOptions.addArguments("--disable-dev-shm-usage");
        //启动代理
        if (null != proxy) {
            chromeOptions.setProxy(proxy);
        }
        return chromeOptions;
    }

    /**
     * 处理请求
     *
     * @param url             需要访问的URL
     * @param proxy           处理的代理请求
     * @param chromeCallback 执行完成的回调
     */
    private void handel(String url, Proxy proxy, ChromeCallback chromeCallback)
{
        SeleniumDriver seleniumDriver = null;
        if (StringUtils.isNotEmpty(url) && null != chromeCallback) {
            try {
                seleniumDriver = initChromeDriver(proxy);
                if (null != seleniumDriver && null !=
seleniumDriver.getWebDriver()) {
                    chromeCallback.callBack(seleniumDriver.getWebDriver());
                }
            } catch (Exception e) {
                log.info("chrome调用失败：" + e.getMessage());
            } finally {
                log.info("关闭chrome驱动");
                closeChrome(seleniumDriver);
            }
        }
    }


    /**
     * 获取Html
     *
     * @param url
     * @return
     */
    public CrawlerHtml getCrawlerHtml(String url, CrawlerProxy crawlerProxy,
String cookieName) {
        log.info("Selenium 开始抓取Html数据，url:{},cookieName:{},proxy:{}", url,
cookieName, crawlerProxy);
        CrawlerHtml crawlerHtml = new CrawlerHtml(url);
        crawlerHtml.setProxy(crawlerProxy);
        Proxy proxy = null;
        if (null != crawlerProxy) {
            proxy = CrawlerProxyFactory.getSeleniumProxy(crawlerProxy);
        }

        handel(url, proxy, driver -> {
            driver.get(url);
            List<CrawlerCookie> crawlerCookieList = delayed(driver, cookieName);
            crawlerHtml.setCrawlerCookieList(crawlerCookieList);
```

```
            crawlerHtml.setHtml(driver.getPageSource());
        });
        log.info("Selenium 抓取Html数据结束，url:{},cookieName:{},cookieValue:
{},proxy:{}", url, cookieName, crawlerHtml.getCrawlerCookieList(),
crawlerProxy);
        return crawlerHtml;
    }


    /**
     * 获取Cookie
     *
     * @param url
     * @return
     */
    public List<CrawlerCookie> getCookie(String url, CrawlerProxy proxy, String
cookieName) {
        CrawlerHtml crawlerHtml = getCrawlerHtml(url, proxy, cookieName);
        return crawlerHtml.getCrawlerCookieList();
    }


    /**
     * 获取Cookie 的延时方法
     * 因为浏览器打开页面有可能js还没有执行完成，获取的数据是不准确定，通过有没有写入一些特殊的
cookie来判断页面是否已经加载完成
     * 如果没有这个cookie则页面一直重复循环，一直等到达到超时时间
     *
     * @param driver
     * @param cookieName
     */
    private List<CrawlerCookie> delayed(final WebDriver driver, final String
cookieName) {
        Object value = DelayedUtils.delayed(new DelayedCallBack() {
            public List<CrawlerCookie> callBack(long time) {
                Set<Cookie> cookieSet = driver.manage().getCookies();
                return getCrawlerCookie(cookieSet);
            }

            /**
             * 判断是否存在该cookie 如果返回false
             * 则一直循环不会退出直到达到超时时间
             * @return
             */
            @Override
            public boolean isExist() {
                Set<Cookie> cookieSet = driver.manage().getCookies();
                return isExistCookieName(cookieSet, cookieName);
            }

            /**
             *每次循环获取需要睡眠的时间
             * 防止一直获取cpu资源耗费的太多，默认1秒重复获取一次
             * @return
             */
            public long sleepTime() {
                return sleepTime;
            }
```

```
            /**
             * 配置的超时时间，防止页面没有找到cookie就一直在循环，没有退出
             * 当达到超时时间就自动退出，将最新的页面数据返回
             * @return
             */
            public long timeOut() {
                return timeOut;
            }
        });
        return (List<CrawlerCookie>) value;
    }

    /**
     * 判断是否包含需要的Cookie
     *
     * @param cookieSet
     * @param cookieName
     * @return
     */
    private boolean isExistCookieName(Set<Cookie> cookieSet, String cookieName)
{
        boolean flag = false;
        if (null != cookieSet && !cookieSet.isEmpty()) {
            for (Cookie cookie : cookieSet) {
                if (cookie.getName().equals(cookieName)) {
                    flag = true;
                    break;
                }
            }
        }
        return flag;
    }

    /**
     * 获取 Cookie
     *
     * @param cookieSet
     * @return
     */
    private List<CrawlerCookie> getCrawlerCookie(Set<Cookie> cookieSet) {
        List<CrawlerCookie> crawlerCookieList = new ArrayList<CrawlerCookie>();
        for (Cookie cookie : cookieSet) {
            CrawlerCookie crawlerCookie = fillCrawlerCookie(cookie);
            if (null != crawlerCookie) {
                crawlerCookieList.add(crawlerCookie);
            }
        }
        return crawlerCookieList;
    }


    private CrawlerCookie fillCrawlerCookie(Cookie cookie) {
        CrawlerCookie crawlerCookie = new CrawlerCookie();
        crawlerCookie.setDomain(cookie.getDomain());
        crawlerCookie.setPath(cookie.getPath());
        crawlerCookie.setName(cookie.getName());
        crawlerCookie.setValue(cookie.getValue());
```

```
        crawlerCookie.setExpire(cookie.getExpiry());
        return crawlerCookie;
    }

    /**
     * 获取json数据
     *
     * @param crawlerHtml
     * @return
     */
    public String getJsonData(CrawlerHtml crawlerHtml) {
        String jsonData = null;
        if (null != crawlerHtml) {
            String htmlStr = crawlerHtml.getHtml();
            if (StringUtils.isNotEmpty(htmlStr)) {
                jsonData = new
Html(htmlStr).xpath(SELENIUM_JSON_DATA_XPATH).toString();
            }
        }
        return jsonData;
    }

    interface ChromeCallback {
        void callBack(WebDriver webDriver);
    }

    /**
     * 关闭浏览器
     */
    private void closeChrome(SeleniumDriver seleniumDriver) {
        if (null != seleniumDriver) {
            WebDriver webDriver = seleniumDriver.getWebDriver();
            ChromeDriverService chromeDriverService =
seleniumDriver.getChromeDriverService();
            try {
                webDriver.quit();
            } finally {
                if (null != chromeDriverService) {
                    chromeDriverService.stop();
                }
            }
            if (null != chromeDriverService) {
                if (chromeDriverService.isRunning()) {
                    chromeDriverService.stop();
                }
            }
        }

    }

    class SeleniumDriver {
        private WebDriver webDriver;

        private ChromeDriverService chromeDriverService;

        public WebDriver getWebDriver() {
            return webDriver;
        }
```

```
        public void setWebDriver(WebDriver webDriver) {
            this.webDriver = webDriver;
        }

        public ChromeDriverService getChromeDriverService() {
            return chromeDriverService;
        }

        public void setChromeDriverService(ChromeDriverService
chromeDriverService) {
            this.chromeDriverService = chromeDriverService;
        }
    }

}
```

配置 CrawlerConfig

com.heima.crawler.config.CrawlerConfig

```
@Bean
public SeleniumClient getSeleniumClient() {
    return new SeleniumClient();
}
```

6.2.3 测试

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class SeleniumClientTest {

    @Autowired
    private SeleniumClient seleniumClient;

    @Test
    public void test(){
        CrawlerHtml crawlerHtml =
 seleniumClient.getCrawlerHtml("http://www.baidu.com",null,null);
        System.out.println(crawlerHtml.getHtml());
    }
}
```

6.2.4 爬虫相关类

爬虫Cookie帮助类

cookie操作的帮助类

com.heima.crawler.helper.CookieHelper

```
/**
 * CookieHelper 用于管理爬取过程中使用对应代理Cookie的管理，
 * CSDN网站的关键cookie与IP地址做了绑定，
 * 如果用非本机的IP访问就会被拦截所以要管理各种代理的cookie，
 * 并且在cookie失效后进行自动更新。
 */
public class CookieHelper {


    /**
     * 代理IP存放的地方
     * <ip+端口号,cookie列表>
     */
    private Map<String, List<CrawlerCookie>> proxyCookieCacheMap = new
ConcurrentHashMap<String, List<CrawlerCookie>>();
    /**
     * 数据锁
     */
    private final String SYNCHRONIZED_TAG = "SYNCHRONIZED_TAG";

    private SeleniumClient seleniumClient = new SeleniumClient();

    public CookieHelper() {
    }


    /**
     * 关键获取cookie的名称
     */
    private String cookieName;


    public CookieHelper(String cookieName) {
        this.cookieName = cookieName;
    }


    /**
     * 并发过滤器 用于多个并发进行访问的时候只有一个并发进行操作，其他并发被拦截
     */
    private final ConcurrentCallBack concurrentFilter =
DelayedUtils.getConcurrentFilter(50000);

    /**
     * 强制更新Cookie
     *
     * @param url
     * @return
     */
    public List<CrawlerCookie> updateCookie(String url, CrawlerProxy proxy) {
        List<CrawlerCookie> cookieList = getProxyCookieList(proxy);
        if (null != cookieList) {
            cookieList.clear();
            List<CrawlerCookie> tmpList = getCookieEntity(url, proxy);
            updateCookie(tmpList, proxy);
        }
        return cookieList;
```

```
    }

    /**
     * 更新Cookie
     *
     * @param crawlerCookieList
     */
    public void updateCookie(List<CrawlerCookie> crawlerCookieList, CrawlerProxy
proxy) {
        if (null != crawlerCookieList && !crawlerCookieList.isEmpty()) {
            putProxyCookieList(proxy, crawlerCookieList);
        }
    }


    /**
     * 获取Cookie
     *
     * @param url
     * @return
     */
    public List<CrawlerCookie> getCookieEntity(String url, CrawlerProxy proxy) {
        CrawlerCookie crawlerCookie = getCookieEntity(url, getCookieName(),
proxy);
        return new ArrayList<CrawlerCookie>() {{
            add(crawlerCookie);
        }};
    }

    /**
     * 获取缓存的Cookie列表
     *
     * @param url
     * @param proxy
     * @return
     */
    public List<CrawlerCookie> getCacheCookieList(String url, CrawlerProxy
proxy) {
        List<CrawlerCookie> cookieList = getProxyCookieList(proxy);
        if (null != cookieList && !cookieList.isEmpty()) {
            return cookieList;
        } else {
            List<CrawlerCookie> tmpList = getAloneCookieEntity(url, cookieName,
proxy);
            updateCookie(tmpList, proxy);
            return cookieList;
        }
    }


    /**
     * 获取Cookie
     *
     * @param url
     * @return
     */
    public CrawlerCookie getCookieEntity(String url, String cookieName,
CrawlerProxy proxy) {
```

```
        CrawlerCookie resultCookie = null;
        List<CrawlerCookie> crawlerCookieList = getCacheCookieList(url, proxy);
        if (null != crawlerCookieList && !crawlerCookieList.isEmpty()) {
            for (CrawlerCookie crawlerCookie : crawlerCookieList) {
                if (crawlerCookie.getName().equals(cookieName)) {
                    resultCookie = crawlerCookie;
                }
            }
        }
        return resultCookie;
    }

    /**
     * 单独获取Cookie
     *
     * @param url
     * @return
     */
    private List<CrawlerCookie> getAloneCookieEntity(String url, String
cookieName, CrawlerProxy proxy) {
        synchronized (SYNCHRONIZED_TAG) {
            List<CrawlerCookie> crawlerCookieList = null;
            boolean filter = concurrentFilter.filter();
            if (filter) {
                CrawlerHtml crawlerHtml = seleniumClient.getCrawlerHtml(url,
proxy, cookieName);
                if (null != crawlerHtml) {
                    crawlerCookieList = crawlerHtml.getCrawlerCookieList();
                }

            }

            return crawlerCookieList;
        }
    }


    /**
     * 获取代理Cookie
     *
     * @param crawlerProxy
     * @return
     */
    private List<CrawlerCookie> getProxyCookieList(CrawlerProxy crawlerProxy) {
        String proxyInfo =
CrawlerProxyFactory.getCrawlerProxyInfo(crawlerProxy);
        List<CrawlerCookie> cookieList = proxyCookieCacheMap.get(proxyInfo);
        if (null == cookieList) {
            cookieList = new ArrayList<CrawlerCookie>();
            putProxyCookieList(crawlerProxy, cookieList);
        }
        return cookieList;
    }


    /**
     * 添加代理Cookie
     *
```

```
     * @param crawlerProxy
     * @param cookieList
     */
    private void putProxyCookieList(CrawlerProxy crawlerProxy,
List<CrawlerCookie> cookieList) {
        String proxyInfo =
CrawlerProxyFactory.getCrawlerProxyInfo(crawlerProxy);
        proxyCookieCacheMap.remove(proxyInfo);
        proxyCookieCacheMap.put(proxyInfo, cookieList);
    }


    public void setCookieName(String cookieName) {
        this.cookieName = cookieName;
    }

    public String getCookieName() {
        return cookieName;
    }

}
```

配置 CrawlerConfig

com.heima.crawler.config.CrawlerConfig

```
private static final ResourceBundle resourceBundle =
ResourceBundle.getBundle("crawler");
private static final String CRUX_COOKIE_NAME =
resourceBundle.getString("crux.cookie.name");
/**
     * 设置Cookie辅助类
     *
     * @return
     */
@Bean
public CookieHelper getCookieHelper() {
    return new CookieHelper(CRUX_COOKIE_NAME);
}
```

爬虫抓取的帮助类

定义了爬虫下载过程中涉及到的各种下载内容校验以及对象的设置

com.heima.crawler.helper.CrawlerHelper

```
/**
 * 抓取辅助类
 */
```

```java
@Log4j2
public class CrawlerHelper {

    /**
     * 抓取保存请求数据的主键
     */
    private final String CRAWLER_PROCESS_FLOW_DATA =
"CRAWLER_PROCESS_FLOW_DATA";

    /**
     * 数据转换主键
     */
    private final String CRAWLER_PROCESS_PARSE_ITEM_DATA =
"CRAWLER_PROCESS_PARSE_ITEM_DATA";

    /**
     * 数据校验辅助类
     */
    private DataValidateCallBack dataValidateCallBack;

    public DataValidateCallBack getDataValidateCallBack() {
        return dataValidateCallBack;
    }

    public void setDataValidateCallBack(DataValidateCallBack
dataValidateCallBack) {
        this.dataValidateCallBack = dataValidateCallBack;
    }


    /**
     * 获取 ParseItem
     *
     * @param request
     * @return
     */
    public ParseItem getParseItem(Request request) {
        ParseItem parseItem = null;
        if (null != request) {
            Object parseItemObject =
request.getExtra(CRAWLER_PROCESS_PARSE_ITEM_DATA);
            if (parseItemObject instanceof JSONObject) {
                parseItem = ((JSONObject)
parseItemObject).toJavaObject(CrawlerParseItem.class);
            } else if (parseItemObject instanceof ParseItem) {
                parseItem = (ParseItem) parseItemObject;
            }
        }
        return parseItem;
    }


    /**
     * 设置
     *
     * @return
     */
    public void setParseItem(Request request, ParseItem parseItem) {
```

```
        if (null != request && null != parseItem) {
            Map<String, Object> extraMap = request.getExtras();
            if (null == extraMap) {
                extraMap = new HashMap<String, Object>();
                request.setExtras(extraMap);
            }
            if (!extraMap.containsKey(CRAWLER_PROCESS_PARSE_ITEM_DATA)) {
                extraMap.put(CRAWLER_PROCESS_PARSE_ITEM_DATA, parseItem);
            }
        }
    }


    /**
     * 获取操作的处理类型
     *
     * @param request
     * @return
     */
    public String getHandelType(Request request) {
        String handelType = CrawlerEnum.HandelType.FORWARD.name();
        ParseItem parseItem = getParseItem(request);
        if (null != parseItem) {
            handelType = parseItem.getHandelType();
        }
        return handelType;
    }


    /**
     * 获取操作的文档类型
     *
     * @param request
     * @return
     */
    public String getDocumentType(Request request) {
        String documentType = CrawlerEnum.DocumentType.OTHER.name();
        ParseItem parseItem = getParseItem(request);
        if (null != parseItem) {
            documentType = parseItem.getDocumentType();
        }
        return documentType;
    }


    /**
     * 请求校验
     *
     * @param page
     * @return
     */
    public boolean requestValidation(Page page) {
        long currentTime = System.currentTimeMillis();
        log.info("开始校验下载数据，url:{}", page.getUrl());
        boolean flag = false;
        DataValidateCallBack dataValidateCallBack = getDataValidateCallBack();
        if (null != dataValidateCallBack) {
            flag = dataValidateCallBack.validate(page.getHtml().toString());
```

```
            log.info("校验数据状态：flag:{}", flag);
        }
        log.info("校验下载数据完成，url:{}，状态：{},耗时：{}", page.getUrl(), flag,
System.currentTimeMillis() - currentTime);
        return flag;
    }


    /**
     * 请求校验
     *
     * @param crawlerHtml
     * @return
     */
    public boolean requestValidation(CrawlerHtml crawlerHtml) {
        boolean flag = false;
        long currentTime = System.currentTimeMillis();
        log.info("开始校验下载数据，url:{}", crawlerHtml.getUrl());
        if (null != crawlerHtml &&
StringUtils.isNotEmpty(crawlerHtml.getHtml())) {
            DataValidateCallBack dataValidateCallBack =
getDataValidateCallBack();
            if (null != dataValidateCallBack) {
                flag =
dataValidateCallBack.validate(crawlerHtml.getHtml().toString());
                log.info("校验数据状态：flag:{}", flag);
            }
        }
        log.info("校验下载数据完成，url:{}，状态：{},耗时：{}", crawlerHtml.getUrl(),
flag, System.currentTimeMillis() - currentTime);
        return flag;
    }

}
```

配置 CrawlerConfig

com.heima.crawler.config.CrawlerConfig

```
  /**
     * 数据校验匿名内部类
     * @param cookieHelper
     * @return
     */
    private DataValidateCallBack getDataValidateCallBack(CookieHelper
cookieHelper) {
        return new DataValidateCallBack() {
            @Override
            public boolean validate(String content) {
                boolean flag = true;
                if (StringUtils.isEmpty(content)) {
                    flag = false;
                } else {
                    boolean isContains_acw_sc_v2 =
content.contains("acw_sc__v2");
```

```
                    boolean isContains_location_reload =
content.contains("document.location.reload()");
                    if (isContains_acw_sc_v2 && isContains_location_reload) {
                        flag = false;
                    }
                }
                return flag;
            }
        };
    }



/**
     * CrawerHelper 辅助类
     *
     * @return
     */
@Bean
public CrawlerHelper getCrawerHelper() {
    CookieHelper cookieHelper = getCookieHelper();
    CrawlerHelper crawerHelper = new CrawlerHelper();
    DataValidateCallBack dataValidateCallBack =
getDataValidateCallBack(cookieHelper);
    crawerHelper.setDataValidateCallBack(dataValidateCallBack);
    return crawerHelper;
}
```

6.2.6 代理下载相关(暂不使用代理)

配置crawler.properties

配置是否启用代理IP

```
# 是否使用代理IP
proxy.isUsedProxyIp=false
```

代理IP的提供者

这个类是为了随机获取一个可用的代理IP进行

com.heima.model.crawler.core.proxy.CrawlerProxyProvider

```
/**
 * 代理IP的提供者
 */
public class CrawlerProxyProvider {
    /**
     * 读写锁特点
     * 读读共享
     * 写写互斥
     * 读写互斥
```

```
     */
    private ReadWriteLock lock = new ReentrantReadWriteLock();
    //获取读锁
    private Lock readLock = lock.readLock();
    //获取写锁
    private Lock writeLock = lock.writeLock();
    /**
     * 随机数生成器，用以随机获取代理IP
     */
    private Random random = new Random();
    /**
     * 是否启动代理IP
     */
    private boolean isUsedProxyIp = true;
    /**
     * 动态代理IP自动更新阈值
     */
    private int proxyIpUpdateThreshold = 10;

    public CrawlerProxyProvider() {
    }

    public CrawlerProxyProvider(List<CrawlerProxy> crawlerProxyList) {
        this.crawlerProxyList = crawlerProxyList;
    }

    /**
     * 代理Ip池
     */
    private List<CrawlerProxy> crawlerProxyList = null;
    /**
     * ip池回调
     */
    private ProxyProviderCallBack proxyProviderCallBack;


    /**
     * 随机获取一个代理IP保证每次请求使用的IP都不一样
     *
     * @return
     */
    public CrawlerProxy getRandomProxy() {
        CrawlerProxy crawlerProxy = null;
        readLock.lock();
        try {
            if (isUsedProxyIp && null != crawlerProxyList &&
!crawlerProxyList.isEmpty()) {
                int randomIndex = random.nextInt(crawlerProxyList.size());
                crawlerProxy = crawlerProxyList.get(randomIndex);
            }
        } finally {
            readLock.unlock();
        }
        return crawlerProxy;
    }

    public void updateProxy() {
        //不使用代理IP 则不进行更新
```

```
        if (isUsedProxyIp && null != proxyProviderCallBack) {
            writeLock.lock();
            try {
                crawlerProxyList = proxyProviderCallBack.getProxyList();
            } finally {
                writeLock.unlock();
            }
        }
    }

    /**
     * 设置代理IP不可用
     *
     * @param proxy
     */
    public void unavailable(CrawlerProxy proxy) {
        if (isUsedProxyIp) {
            writeLock.lock();
            crawlerProxyList.remove(proxy);
            writeLock.unlock();
//            proxyProviderCallBack.unvailable(proxy);
            if (crawlerProxyList.size() <= proxyIpUpdateThreshold) {
                updateProxy();
            }
        }
    }


    public List<CrawlerProxy> getCrawlerProxyList() {
        return crawlerProxyList;
    }

    public void setCrawlerProxyList(List<CrawlerProxy> crawlerProxyList) {
        this.crawlerProxyList = crawlerProxyList;
    }

    public boolean isUsedProxyIp() {
        return isUsedProxyIp;
    }

    public void setUsedProxyIp(boolean usedProxyIp) {
        isUsedProxyIp = usedProxyIp;
    }

    public ProxyProviderCallBack getProxyProviderCallBack() {
        return proxyProviderCallBack;
    }

    public void setProxyProviderCallBack(ProxyProviderCallBack
proxyProviderCallBack) {
        this.proxyProviderCallBack = proxyProviderCallBack;
    }
}
```

配置 CrawlerConfig

com.heima.crawler.config.CrawlerConfig

```
/**
     * 是否使用代理Ip
     */
private boolean isUsedProxyIp =
Boolean.parseBoolean(resourceBundle.getString("proxy.isUsedProxyIp"));

    /**
     * CrawlerProxyProvider bean
     *
     * @return
     */
    @Bean
    public CrawlerProxyProvider getCrawlerProxyProvider() {
        CrawlerProxyProvider crawlerProxyProvider = new CrawlerProxyProvider();
        crawlerProxyProvider.setUsedProxyIp(isUsedProxyIp);
        return crawlerProxyProvider;
    }
```

6.2.7 ProxyHttpClientDownloader类

该类继承了AbstractDownloader 是对ProxyHttpClientDownloader类的重写，对里面的一些校验做了重写，实现了自定义的校验，如果下载数据失败尝试换取其他的代理IP下载，如果三次都下载失败则尝试重用selenium方式进行下载

```
/**
 * 代理模式下 进行Http方式的下载
 * ProxyHttpClientDownloader类是使用代理方式进行数据页面数据下载，
 * 但是不满足需求做了一些重试的定制，是对HttpClientDownloader类的一些扩展，
 * 是从https://gitee.com/flashsword20/webmagic/blob/master/webmagic-
core/src/main/java/us/codecraft/webmagic/downloader/HttpClientDownloader.java
 * 获取源码后编辑的来的
 * <p>
 * 增加了如果下载失败后自动调用 selenium 的方式进行文档下载
 *
 * @author code4crafter@gmail.com <br>
 * @since 0.1.0
 */
@Log4j2
@Component
public class ProxyHttpClientDownloader extends AbstractDownloader implements
ProcessFlow {

    @Autowired
    private CookieHelper cookieHelper;

    @Autowired
    private CrawlerHelper crawlerHelper;
```

```
    @Autowired
    private CrawlerProxyProvider crawlerProxyProvider;

    @Autowired
    private SeleniumClient seleniumClient;


    private Logger logger = LoggerFactory.getLogger(getClass());

    private final Map<String, CloseableHttpClient> httpClients = new
HashMap<String, CloseableHttpClient>();

    private HttpClientGenerator httpClientGenerator = new HttpClientGenerator();

    private HttpUriRequestConverter httpUriRequestConverter = new
HttpUriRequestConverter();

    private ProxyProvider proxyProvider;

    private boolean responseHeader = true;

    public void setHttpUriRequestConverter(HttpUriRequestConverter
httpUriRequestConverter) {
        this.httpUriRequestConverter = httpUriRequestConverter;
    }

    public void setProxyProvider(ProxyProvider proxyProvider) {
        this.proxyProvider = proxyProvider;
    }

    private CloseableHttpClient getHttpClient(Site site) {
        if (site == null) {
            return httpClientGenerator.getClient(null);
        }

        String domain = site.getDomain();
        CloseableHttpClient httpClient = httpClients.get(domain);
        if (httpClient == null) {
            synchronized (this) {
                httpClient = httpClients.get(domain);
                if (httpClient == null) {
                    httpClient = httpClientGenerator.getClient(site);
                    httpClients.put(domain, httpClient);
                }
            }
        }
        return httpClient;
    }


    /**
     * webmagic 下载页面调用的方法入口
     *
     * @param request 请求的request
     * @param task    任务
     * @return
     */
    @Override
```

```
    public Page download(Request request, Task task) {
        String handelType = crawlerHelper.getHandelType(request);
        long currentTime = System.currentTimeMillis();
        log.info("开始下载页面数据，url:{},handelType:{}",
request.getUrl(),handelType);
        if (task == null || task.getSite() == null) {
            throw new NullPointerException("task or site can not be null");
        }
        CloseableHttpResponse httpResponse = null;
        Site site = task.getSite();

        //设置代理对象
        Proxy proxy = proxyProvider != null ? proxyProvider.getProxy(task) :
null;
        //将 Proxy 转换为我们自己的 CrawlerProxy
        CrawlerProxy crawlerProxy = proxy == null ? null : new
CrawlerProxy(proxy.getHost(), proxy.getPort());
        //添加Cookie
        addCookie(site, request.getUrl(), crawlerProxy);

        CloseableHttpClient httpClient = getHttpClient(site);
        HttpClientRequestContext requestContext =
httpUriRequestConverter.convert(request, task.getSite(), proxy);
        Page page = Page.fail();
        try {
            httpResponse =
httpClient.execute(requestContext.getHttpUriRequest(),
requestContext.getHttpClientContext());

            page = handleResponse(request, request.getCharset() != null ?
request.getCharset() : task.getSite().getCharset(), httpResponse, task);
            //验证httpClient返回的数据是否是正常格式

            boolean downloadStatus = checkDownloadStatus(page, crawlerProxy);
            //下载失败
            if (!downloadStatus) {
                page = seleniumDownload(page);
                downloadStatus = crawlerHelper.requestValidation(page);
            }

            if (downloadStatus) {
                page.setStatusCode(200);
                onSuccess(request);
                log.info("下载数据成功，url:{}，handelType:{},耗时：{}",
request.getUrl(),handelType, System.currentTimeMillis() - currentTime);
            } else {
                onError(request);
                log.error("下载文档失败，url:{},handelType:{},proxy:{},状态码：{}",
page.getUrl().toString(),handelType, proxy, page.getStatusCode());
            }


            return page;
        } catch (IOException e) {
            logger.warn("download page {} error", request.getUrl(), e);
            onError(request);
            return page;
        } finally {
```

```
            if (httpResponse != null) {
                //ensure the connection is released back to pool
                EntityUtils.consumeQuietly(httpResponse.getEntity());
            }
            if (proxyProvider != null && proxy != null) {
                proxyProvider.returnProxy(proxy, page, task);
            }
        }
    }

    /**
     * 校验下载状态
     *
     * @param page
     * @return
     */
    private boolean checkDownloadStatus(Page page, CrawlerProxy proxy) {
        boolean downloadStatus = false;
        if (page.getStatusCode() == 200) {
            downloadStatus = crawlerHelper.requestValidation(page);
        } else {
            crawlerProxyProvider.unavailable(proxy);
        }
        return downloadStatus;
    }

    @Override
    public void setThread(int thread) {
        httpClientGenerator.setPoolSize(thread);
    }

    protected Page handleResponse(Request request, String charset, HttpResponse
httpResponse, Task task) throws IOException {
        byte[] bytes =
IOUtils.toByteArray(httpResponse.getEntity().getContent());
        String contentType = httpResponse.getEntity().getContentType() == null ?
"" : httpResponse.getEntity().getContentType().getValue();
        Page page = new Page();
        page.setBytes(bytes);
        if (!request.isBinaryContent()) {
            if (charset == null) {
                charset = getHtmlCharset(contentType, bytes);
            }
            page.setCharset(charset);
            page.setRawText(new String(bytes, charset));
        }
        page.setUrl(new PlainText(request.getUrl()));
        page.setRequest(request);
        page.setStatusCode(httpResponse.getStatusLine().getStatusCode());
        page.setDownloadSuccess(true);
        if (responseHeader) {

 page.setHeaders(HttpClientUtils.convertHeaders(httpResponse.getAllHeaders()));
        }
        return page;
    }
```

```
    private String getHtmlCharset(String contentType, byte[] contentBytes)
throws IOException {
        String charset = CharsetUtils.detectCharset(contentType, contentBytes);
        if (charset == null) {
            charset = Charset.defaultCharset().name();
            logger.warn("Charset autodetect failed, use {} as charset. Please
specify charset in Site.setCharset()", Charset.defaultCharset());
        }
        return charset;
    }


    //************************************************以下代码是自定义的代码
************************

    /**
     * 初始化webmagic的代理IP
     *
     * @param processFlowData
     */
    @Override
    public void handel(ProcessFlowData processFlowData) {
        Proxy[] proxies =
getProxyArray(crawlerProxyProvider.getCrawlerProxyList());
        if (null != proxies && proxies.length > 0) {
            setProxyProvider(SimpleProxyProvider.from(proxies));
        }
    }


    /**
     * selenium+chrome headless 方式下载
     *
     * @param page
     */
    public Page seleniumDownload(Page page) {
        CrawlerHtml crawlerHtml = proxySeleniumDownloadRetry(page);
        boolean requestValidation =
crawlerHelper.requestValidation(crawlerHtml);
        //校验失败
        if (!requestValidation) {
            //不使用代理尝试本地下载
            crawlerHtml =
seleniumClient.getCrawlerHtml(page.getUrl().toString(), null,
cookieHelper.getCookieName());
            requestValidation = crawlerHelper.requestValidation(crawlerHtml);
        }
        //如果校验成功成功
        if (requestValidation) {
            cookieHelper.updateCookie(crawlerHtml.getCrawlerCookieList(),
crawlerHtml.getProxy());
            Html html = new Html(crawlerHtml.getHtml());
            page.setHtml(html);
        }
        return page;
    }

    /**
```

```
     * 使用代理方式进行下载重试
     *
     * @param page
     * @return
     */
    public CrawlerHtml proxySeleniumDownloadRetry(Page page) {
        CrawlerHtml crawlerHtml = null;
        for (int i = 0; i < 3; i++) {
            long currentTime = System.currentTimeMillis();
            CrawlerProxy proxy = crawlerProxyProvider.getRandomProxy();
            log.info("尝试使用selenium下载数据第{}次，url:{}，代理：{}", i + 1,
page.getUrl(), proxy);
            crawlerHtml =
seleniumClient.getCrawlerHtml(page.getUrl().toString(), proxy,
cookieHelper.getCookieName());
            log.info("尝试使用selenium下载数据第{}次完成，代理：{}，url:{}，耗时：{}",
i, proxy, page.getUrl(), System.currentTimeMillis() - currentTime);
            if (StringUtils.isNotEmpty(crawlerHtml.getHtml())) {
                break;
            }
            //该代理不可用禁用
            crawlerProxyProvider.unavailable(proxy);
        }
        return crawlerHtml;
    }


    /**
     * 根据代理Ip 添加Cookie
     *
     * @param site
     * @param url
     * @param proxy
     */
    private void addCookie(Site site, String url, CrawlerProxy proxy) {
        List<CrawlerCookie> crawlerCookieList =
cookieHelper.getCacheCookieList(url, proxy);
        if (null != site && null != crawlerCookieList &&
!crawlerCookieList.isEmpty()) {
            for (CrawlerCookie crawlerCookie : crawlerCookieList) {
                if (null != crawlerCookie) {
                    site.addCookie(crawlerCookie.getName(),
crawlerCookie.getValue());
                }
            }
        }
    }


    /**
     * 获取代理数组
     *
     * @param crawlerProxyList
     * @return
     */
    private Proxy[] getProxyArray(List<CrawlerProxy> crawlerProxyList) {
        Proxy[] proxyArray = null;
        if (null != crawlerProxyList && !crawlerProxyList.isEmpty()) {
```

```
            proxyArray = new Proxy[crawlerProxyList.size()];
            for (int i = 0; i < crawlerProxyList.size(); i++) {
                proxyArray[i] =
CrawlerProxyFactory.getWebmagicProxy(crawlerProxyList.get(i));
            }
        }
        return proxyArray;
    }


    public CrawlerEnum.ComponentType getComponentType() {
        return CrawlerEnum.ComponentType.DOWNLOAD;
    }

    @Override
    public int getPriority() {
        return 100;
    }
}
```

6.3 文档处理

> 下载完成数据后就需要进行文档处理，这里的处理是分三个步骤
>
> 1. 解析初始化的URL获取列表页，将列表页的数据提交下载处理器
> 2. 解析完列表页后获取最终的需要处理的URL交给下载处理器
> 3. 解析最终URL数据，将解析的数据交给下一级处理器处理

6.3.1 前置工作

实体类

解析实体类ParseItem

定义了转换后对象的骨架

com.heima.crawler.core.parse.ParseItem

```
/**
 * 解析封装对象
 */
public abstract class ParseItem implements Serializable {
    /**
     * 处理类型 有正向 反向两种
     * FORWARD, 正向 REVERSE 反向
     */
    private String handelType = null;

    /**
     * 文档抓取类型
     */
    private String documentType = null;

    /**
     * 渠道名称
     */
    private String channelName;
```

```
    /**
     * 获取初始的URL
     *
     * @return
     */
    public abstract String getInitialUrl();


    /**
     * 获取需要处理的内容
     *
     * @return
     */
    public abstract String getParserContent();

    public String getHandelType() {
        return handelType;
    }

    public void setHandelType(String handelType) {
        this.handelType = handelType;
    }

    public String getDocumentType() {
        return documentType;
    }

    public void setDocumentType(String documentType) {
        this.documentType = documentType;
    }

    public String getChannelName() {
        return channelName;
    }

    public void setChannelName(String channelName) {
        this.channelName = channelName;
    }
}
```

爬虫解析类CrawlerParseItem

> 继承了CrawlerParseItem 对

```
@Setter
@Getter
public class CrawlerParseItem extends ParseItem {

    /**
     * 数据ID
     */
    private String id;
    /**
     * 说明
     */
    private String summary;
```

```
    /**
     * 博客url
     */
    private String url;
    /**
     * 个人空间URL
     */

    private String spatialUrl;

    /**
     * 标签
     */
    private String labels;
    /**
     * 策略
     */
    private String strategy;

    /**
     * 标题
     */
    private String title;

    /**
     * 类型
     */
    private String type;
    /**
     * 文档类型
     */
    private int docType;

    /**
     * 副标题
     */
    private String subTitle;

    /**
     * 作者
     */
    private String author;

    /**
     * 发布日期
     */
    private String releaseDate;


    /**
     * 阅读量
     */
    private Integer readCount;

    /**
     * 评论数量
     */
    private Integer commentCount;
```

```
    /**
     * 点赞量
     */
    private Integer likes;

    /**
     * 图文内容
     */
    private String content;

    /**
     * 压缩后的内容
     */
    private String compressContent;


    public String getInitialUrl() {
        return getUrl();
    }

    @Override
    public String getParserContent() {
        return getContent();
    }

}
```

抓取内容和规则的封装ParseRule

com.heima.crawler.core.parse.ParseRule

```
/**
 * 抓取内容封装
 */
public class ParseRule {
    /**
     * 映射字段
     */
    private String field;
    /**
     * URL 校验规则
     */
    private String urlValidateRegular;

    /**
     * 解析规则类型
     */
    private CrawlerEnum.ParseRuleType parseRuleType;
    /**
     * 规则
     */
    private String rule;

    /**
     * 抓取内容列表
```

```
     */
    private List<String> parseContentList;

    public ParseRule() {
    }

    /**
     * 构造方法
     *
     * @param field
     * @param parseRuleType
     * @param rule
     */
    public ParseRule(String field, CrawlerEnum.ParseRuleType parseRuleType,
String rule) {
        this.field = field;
        this.parseRuleType = parseRuleType;
        this.rule = rule;
    }

    /**
     * 检查是否有效，如果内容为空则判断该类为空
     *
     * @return
     */
    public boolean isAvailability() {
        boolean isAvailability = false;
        if (null != parseContentList && !parseContentList.isEmpty()) {
            isAvailability = true;
        }
        return isAvailability;
    }

    /**
     * 获取合并后的内容
     *
     * @return
     */
    public String getMergeContent() {
        StringBuilder stringBuilder = new StringBuilder();
        if (null != parseContentList && !parseContentList.isEmpty()) {
            for (String str : parseContentList) {
                str = StringUtils.trim(str);
                if (StringUtils.isNotEmpty(str)) {
                    stringBuilder.append(str).append(",");
                }
            }
        }
        return StringUtils.removeEnd(stringBuilder.toString(), ",");
    }


    public String getField() {
        return field;
    }

    public void setField(String field) {
        this.field = field;
```

```
    }

    public String getUrlValidateRegular() {
        return urlValidateRegular;
    }

    public void setUrlValidateRegular(String urlValidateRegular) {
        this.urlValidateRegular = urlValidateRegular;
    }

    public CrawlerEnum.ParseRuleType getParseRuleType() {
        return parseRuleType;
    }

    public void setParseRuleType(CrawlerEnum.ParseRuleType parseRuleType) {
        this.parseRuleType = parseRuleType;
    }

    public String getRule() {
        return rule;
    }

    public void setRule(String rule) {
        this.rule = rule;
    }

    public List<String> getParseContentList() {
        return parseContentList;
    }

    public void setParseContentList(List<String> parseContentList) {
        this.parseContentList = parseContentList;
    }
}
```

爬虫配置属性CrawlerConfigProperty

com.heima.crawler.process.entity.CrawlerConfigProperty

```
/**
 * 爬虫配置相关属性
 */
@Setter
@Getter
public class CrawlerConfigProperty implements Serializable {
    /**
     * 初始化请求
     */
    private List<String> initCrawlerUrlList;
    /**
     * 初始化抓取的xpath表达式
     */
    private String initCrawlerXpath;

    /**
```

```
     * 帮助页面抓取规则
     */
    private String helpCrawlerXpath;

    /**
     * 开启帮助页面分页抓取
     */
    private Integer crawlerHelpNextPagingSize;

    /**
     * 目标页抓取规则
     */
    private List<ParseRule> targetParseRuleList;

}
```

配置CrawlerConfig

com.heima.crawler.config.CrawlerConfig

```
/**
     * 是否开启帮助页面分页抓取
     */
private Integer crawlerHelpNextPagingSize =
Integer.parseInt(resourceBundle.getString("crawler.help.nextPagingSize"));

/**
     * 帮助页面抓取Xpath
     */
private String helpCrawlerXpath = "//div[@class='article-
list']/div[@class='article-item-box']/h4/a";
/**
     * 初始化抓取的Xpath
     */
private String initCrawlerXpath =
"//ul[@class='feedlist_mod']/li[@class='clearfix']/div[@class='list_con']/dl[@cl
ass='list_userbar']/dd[@class='name']/a";

@Bean
public CrawlerConfigProperty getCrawlerConfigProperty() {
    CrawlerConfigProperty crawlerConfigProperty = new CrawlerConfigProperty();
    crawlerConfigProperty.setInitCrawlerUrlList(getInitCrawlerUrlList());
    crawlerConfigProperty.setHelpCrawlerXpath(helpCrawlerXpath);
    crawlerConfigProperty.setTargetParseRuleList(getTargetParseRuleList());

 crawlerConfigProperty.setCrawlerHelpNextPagingSize(crawlerHelpNextPagingSize);
    crawlerConfigProperty.setInitCrawlerXpath(initCrawlerXpath);
    return crawlerConfigProperty;
}



/**
     * 目标页面抓取规则
     *
     * @return
     */
```

```
public List<ParseRule> getTargetParseRuleList() {
    List<ParseRule> parseRuleList = new ArrayList<ParseRule>() {{
        //标题
        add(new ParseRule("title", CrawlerEnum.ParseRuleType.XPATH,
"//h1[@class='title-article']/text()"));
        //作者
        add(new ParseRule("author", CrawlerEnum.ParseRuleType.XPATH,
"//a[@class='follow-nickName']/text()"));
        //发布日期
        add(new ParseRule("releaseDate", CrawlerEnum.ParseRuleType.XPATH,
"//span[@class='time']/text()"));
        //标签
        add(new ParseRule("labels", CrawlerEnum.ParseRuleType.XPATH,
"//span[@class='tags-box']/a/text()"));
        //个人空间
        add(new ParseRule("personalSpace", CrawlerEnum.ParseRuleType.XPATH,
"//a[@class='follow-nickName']/@href"));
        //阅读量
        add(new ParseRule("readCount", CrawlerEnum.ParseRuleType.XPATH,
"//span[@class='read-count']/text()"));
        //点赞量
        add(new ParseRule("likes", CrawlerEnum.ParseRuleType.XPATH,
"//div[@class='tool-box']/ul[@class='meau-list']/li[@class='btn-like-
box']/button/p/text()"));
        //回复次数
        add(new ParseRule("commentCount", CrawlerEnum.ParseRuleType.XPATH,
"//div[@class='tool-box']/ul[@class='meau-list']/li[@class='to-
commentBox']/button/p/text()"));
        //html内容
        add(new ParseRule("content", CrawlerEnum.ParseRuleType.XPATH,
"//div[@id='content_views']/html()"));

    }};
    return parseRuleList;
}
```