

浙江大学计算机科学与技术学院

Java 程序设计课程报告

2020—2021 学年 秋冬 学期

题目	图书搜索引擎
学号	3180102095
学生姓名	臧可
所在专业	计算机科学与技术
所在班级	计科 1804

目 录

1 引言.....	1
1.1 设计目的.....	1
1.2 设计说明.....	1
2 总体设计.....	2
2.1 功能模块设计.....	2
2.2 流程图设计.....	2
3 详细设计.....	4
3.1 htmlunit 动态加载网页.....	4
3.2 Jsoup 爬取详细信息.....	5
3.3 Lucene 存储书本信息.....	7
3.4 图书查询.....	10
4 测试与运行.....	12
4.1 程序测试.....	12
4.2 程序运行.....	13
5 总结.....	15
参考文献.....	17

1 引言

本次开发的是一个图书搜索引擎，这是一个综合性的题目，涉及到 Java 爬虫、信息抽取、索引建立、查询等技术，可以对 Java 语言中的各项功能有更好的理解和使用，通过具体的程序来加深对 Java 语言的掌握，提高自己的编程水平，为以后的工作打下一定的基础。

1.1 设计目的

图书搜索引擎可以实现对于当当网页图书信息的爬取、存储和查询。本文使。具体功能如下：

- (1) 实现 Web 爬虫，爬取当当网站的网页
- (2) 解析网页内容，对内容进行结构化，并存储到文件中，包括标题、作者、分类、出版社、出版时间、图书照片、编辑推荐、内容简介、目录、价格。
- (3) 为内容建立索引。为每个图书重新编号。
- (4) 可以通过命令行进行内容检索，并展示内容列表

1.2 设计说明

本程序采用 Java 程序设计语言，在 Eclipse 平台下编辑、编译与调试。最后生成 BookCrawler.jar 和 BookSearcher.jar 两个文件可以用命令行运行，BookCrawler 进行当当网页信息的爬取，BookSearcher 对已爬取信息进行检索。

2 总体设计

2.1 功能模块设计

本程序需实现的主要功能有：

- (1) 用户可以自由输入想要爬取的当当网站图书号范围
- (2) 用户可以根据标题、作者、出版社、存储编号四类信息进行检索
- (3) 用户可以在检索后获得存储编号、标题、作者、出版社、出版时间、价格和图书详细信息。

程序的功能模块如图 1 所示：

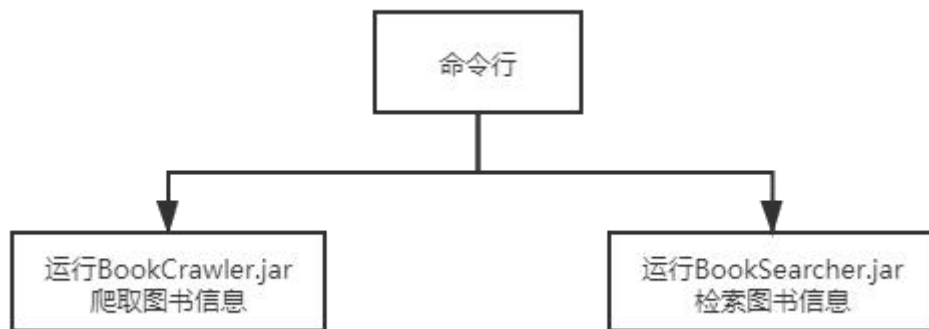


图 1 功能模块

2.2 流程图设计

程序总体流程如图 2 所示：

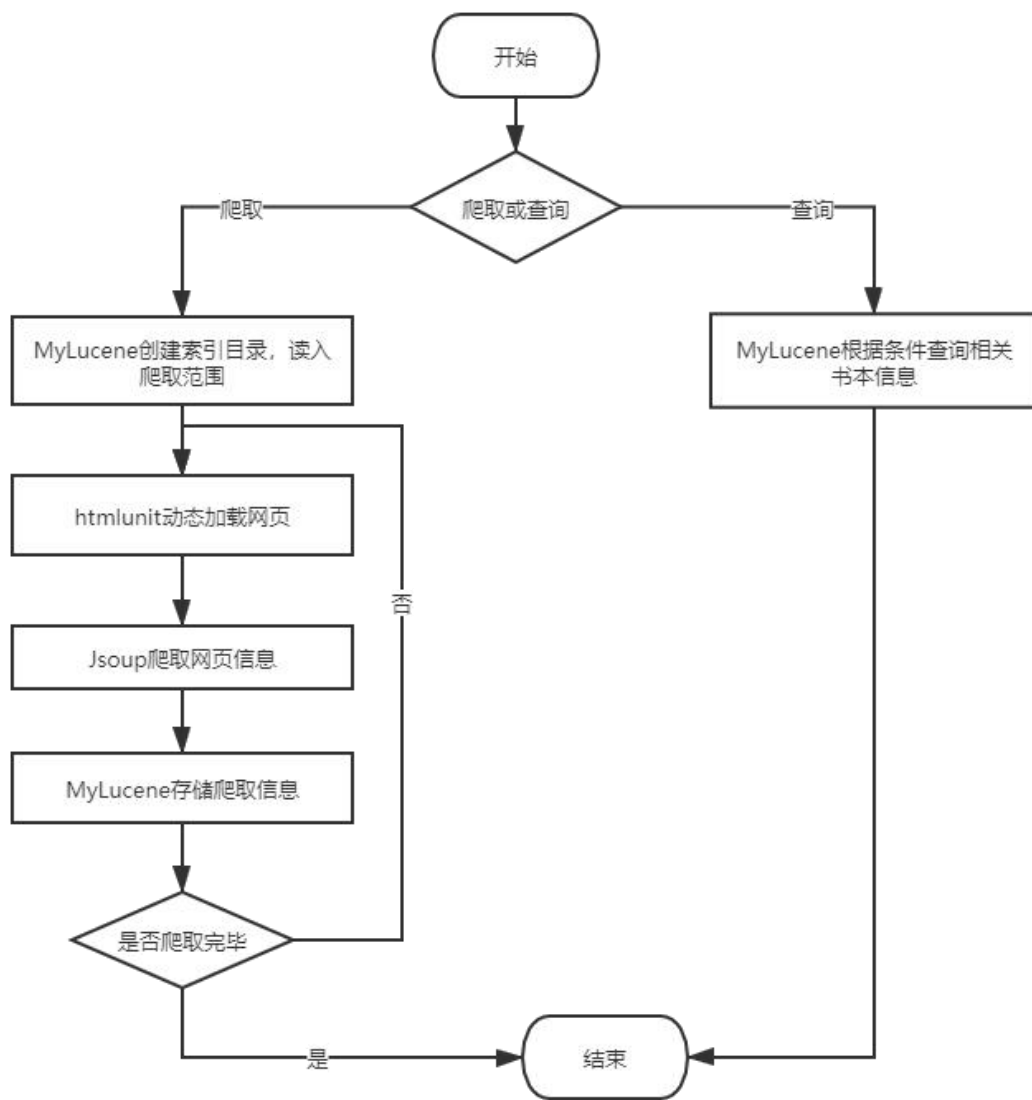


图 2 总体流程图

3 详细设计

3.1 htmlunit 动态加载网页

爬取的当当网页只有基础信息（标题、作者、价格、出版社）是静态加载的，爬取详细信息如编辑推荐、作者介绍等则需要动态加载网页。

本实验导入了 jar 包 jsoup 和 htmlunit 实现对浏览器的模拟完成动态爬取网页和网页信息的抓取。

详细代码如下

```
//创建一个 webclient
WebClient webClient = new
WebClient(BrowserVersion.CHROME);

//参数设置
// 1 启动 JS
webClient.getOptions().setJavaScriptEnabled(true);
// 2 禁用 CSS，可避免自动二次请求 CSS 进行渲染
webClient.getOptions().setCssEnabled(false);
//3 启动客户端重定向
webClient.getOptions().setRedirectEnabled(true);
// 4 运行错误时，是否抛出异常

webClient.getOptions().setThrowExceptionOnScriptError(false);
// 5 设置超时
webClient.getOptions().setTimeout(5000);
//6 设置忽略证书
webClient.getOptions().setUseInsecureSSL(true);
//7 设置 Ajax
webClient.setAjaxController(new
NicelyResynchronizingAjaxController());
//8 设置 cookie
webClient.getCookieManager().setCookiesEnabled(true);

// 等待 JS 驱动 dom 完成获得还原后的网页
webClient.waitForBackgroundJavaScript(8000);
HtmlPage page = null;
try {
```

```

        page = webClient.getPage(url);
    } catch (FailingHttpStatusCodeException | IOException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    //4.将页面转成指定格式
    webClient.waitForBackgroundJavaScript(8000);    //等待
js 脚本执行完成
    if(page == null) return ;
    String html = page.asXml();
    Document document = Jsoup.parse(html);

```

其中，三个等待时间都是经过反复测试后确定下来的较优选择：可以保证在大多数情况下网页可以动态加载成功，并且爬取每页信息不会耗费太多的无用等待时间。

3. 2 Jsoup 爬取详细信息

对网页进行一个动态的加载，确保之后爬取的信息是完整的，然后用 Jsoup 进行网页内容的爬取。将爬取的内容存储到静态全局变量 webStrings 中，方便之后存储信息。

```

static String[] webStrings = new String[11];

```

详细代码如下

```

    Element titleElement = document.select("div.name_info >
h1").first();
    Element authorElement = document.select("span:contains(作
者)").first();
    Elements typeElement =
document.select("#detail-category-path");
    Element publicElement = document.select("span:contains(出
版社)").first();
    Element timeElement = document.select("span:contains(出版
时间)").first();
    Elements priceElement = document.select("#dd-price");
    Elements editorElement =
document.select("div#abstract.section >
div.descrip");//div#abstract.section > div.descrip
    Elements abstractElement =
document.select("div#content.section > div.descrip");

```

```

        Elements authorIntroElement =
document.select("div#authorIntroduction.section > div.descrip");
        Elements catalogElement =
document.select("div#catalog.section > div.descrip >
#catalog-textarea");
        Element pictureElement = document.select("div.pic > a.img >
img").first();

//      System.out.println(editorElement.html()); //测试动态加载用

        webStrings[0] = (titleElement!=null)?("标题:
"+titleElement.text()):"";
        webStrings[1] =
(authorElement!=null)?authorElement.text():"";
        webStrings[2] =
(typeElement!=null)?typeElement.text():"";
        webStrings[3] =
(publicElement!=null)?publicElement.text():"";
        webStrings[4] =
(timeElement!=null)?timeElement.text():"";
        webStrings[5] = (priceElement!=null)?("价格:
"+priceElement.text()):"";
        webStrings[6] = (editorElement!=null)?("编辑推荐:
\n"+editorElement.text()):"";
        webStrings[7] = (abstractElement!=null)?("内容简介:
\n"+abstractElement.text()):"";
        webStrings[8] = (authorIntroElement!=null)?("作者简介:
\n"+authorIntroElement.text()):"";

        String catalogString = catalogElement.text();
        String aa = "";
        Pattern p = Pattern.compile("<|>|p|/");
        Matcher m = p.matcher(catalogString);

        webStrings[9] = m.replaceAll(aa).trim();
        webStrings[10] = (pictureElement!=null)?("图片链接:
\n"+pictureElement.attr("src")):"";
    }

```

详细说明:

(1) 使用选择器语法来查找元素

jsoup elements 对象支持类似于 CSS (或 jquery) 的选择器语法, 来实现非常强大和灵活的查找功能。

这个 select 方法在 Document, Element, 或 Elements 对象中都可以使用。且是上下文相关的, 因此可实现指定元素的过滤, 或者链式选择访问。

Select 方法将返回一个 Elements 集合, 并提供一组方法来抽取和处理结果。

(2) 判断每个 Element 是否为空

每一本书的详细信息不一定全部存在。

当对应 Element 为空的时候, 程序会因为 string 存储了一个空值而报错停止。为了避免上述情况的发生, 用 A? B: C 的语法解决了这一问题。若 Element 为空, 则返回空字符串, 存储信息的时候不会产生错误, 查询的时候改值只会显示属性值内容则显示为空。

(3) 获得详细的 catalogstring

因为部分图书的 catalog 过长, 有时候动态加载直接显示的页面不会显示完整的目录。检查网页源码, 找到 “`div#catalog.section > div.descrip > #catalog-textarea`” 这块区域存放着隐藏的完整目录, 当点击网页上的显示完整目录的按钮的时候才会显现。

这部分代码隐藏状态下会包含 `<p></p>` 符号, 若直接存储到 String 中则显示出来的结果会不美观, 所以用 `Pattern.matcher()` 过滤无效信息。

3.3 Lucene 存储书本信息

通过改编作业提供的一个简单的 Lucene 的基础工程实现。

为了方便操作, 爬取网页和查询两个操作的 main 函数是分开来的, 所以 `public static void search(String filePath)` 函数设置成静态的, 在查询书本的 main 函数里再调用。

具体代码如下:

```
public class MyLucene {  
  
    private static int bookid = 1;  
  
    public static void main(String[] args){  
        MyLucene w = new MyLucene();  
        String filePath = "c:/JAVA/index";// 创建索引的存储目录
```

```

        w.createIndex(filePath);// 创建索引
    }

    public void createIndex(String filePath){
        File f = new File(filePath);
        IndexWriter iwr = null;
        try {
            Directory dir = FSDirectory.open(f);
            Analyzer analyzer = new IKAnalyzer();
            IndexWriterConfig conf = new
IndexWriterConfig(Version.LUCENE_4_10_0, analyzer);
            iwr = new IndexWriter(dir, conf);// 建立 IndexWriter。
固定套路
//
            int i = 29131096;//测试用
            Scanner scanner = new Scanner(System.in);
            int begin, end;
            System.out.println("输入你想要爬取的当当网图书号范围: ");
            begin = scanner.nextInt();
            end = scanner.nextInt();
            for (int i = begin; i <= end; ++i) {
                bookid = i;
                System.out.println("正在爬取编号"+i+"的图书信息");

                CrawBook.getBookInfo("http://product.dangdang.com/"+i+".htm
1");

                Document doc = getDocument();
                iwr.addDocument(doc);// 添加 doc, Lucene 的检索是以
document 为基本单位
                System.out.println("编号"+i+"的图书信息存储完毕! ");
            }
            System.out.println("图书信息存储完毕! ");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try {
            iwr.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

```

    public Document getDocument(){
        // doc 中内容由 field 构成，在检索过程中，Lucene 会按照指定的
        Field 依次搜索每个 document 的该项 field 是否符合要求。
        Document doc = new Document();
        String[] webStrings = CrawBook.webStrings;
        if(webStrings[0]== null || webStrings[0].equals(""))
return doc;
        String str0 = String.format("%8d", bookid).replace(" ",
"0");
        Field f0 = new TextField("bookid", str0, Field.Store.YES);
        Field f1 = new TextField("title", webStrings[0],
Field.Store.YES);
        Field f2 = new TextField("author", webStrings[1],
Field.Store.YES);
        Field f3 = new TextField("type", webStrings[2],
Field.Store.YES);
        Field f4 = new TextField("public", webStrings[3],
Field.Store.YES);
        Field f5 = new TextField("time", webStrings[4],
Field.Store.YES);
        Field f6 = new TextField("price", webStrings[5],
Field.Store.YES);
        Field f7 = new TextField("editor", webStrings[6],
Field.Store.YES);
        Field f8 = new TextField("abstract", webStrings[7],
Field.Store.YES);
        Field f9 = new TextField("authorIntro", webStrings[8],
Field.Store.YES);
        Field f10 = new TextField("catalog", webStrings[9],
Field.Store.YES);
        Field f11 = new TextField("picture",
webStrings[10],Field.Store.YES);

        doc.add(f0);
        doc.add(f1);
        doc.add(f2);
        doc.add(f3);
        doc.add(f4);
        doc.add(f5);
        doc.add(f6);
        doc.add(f7);
        doc.add(f8);
        doc.add(f9);
        doc.add(f10);
    }

```

```

        doc.add(f11);

        return doc;
    }

    public static void searrh(String filePath) {
        ...
    }
}

```

详细说明：

（1）根据当当网书号爬虫

经调查发现当当网每一本图书详细页面的网址组成均为 `"http://product.dangdang.com/"+i+".html"`，其中 `i` 是每个书特有的编码。

所以将程序设计成可以从控制台输入想要爬取的书号范围，一次性可以爬取多本图书这种形式。

每爬一本书调用一次 `CrawBook.getBookInfo()` 函数，对需要爬取的网页进行动态加载和信息爬取。

（2）存储书本信息

将静态全局变量 `CrawBook.webStrings` 中的信息加到相应的 `doc` 中。

在存储书本信息之前进行一个判断：如果 `webStrings[0] == null || webStrings[0].equals("")`（即网页不存在或者图书信息不存在）直接返回空的 `doc`，避免程序因为出错停止。

3.4 图书查询

图书查询写在 `main` 函数里，内容也十分简单，就是调用 `MyLucene` 里面的静态函数进行查询。

`main` 函数代码如下所示：

```

public class main {
    public static void main(String[] args) throws Exception {
        MyLucene.searrh("c:/JAVA/index");
    }
}

```

`public static void searchh(String filePath)`也是套用老师给的框架

实现，相信代码如下：

```
public static void searchh(String filePath) {
    Scanner scanner = new Scanner(System.in);
    File f = new File(filePath);
    try {
        IndexSearcher searcher = new
IndexSearcher(DirectoryReader.open(FSDirectory.open(f)));
        System.out.println("请输入你想检索的类型（a-书号，b-标题，
c-作者，d-分类，e-出版社）：");
        String searchType = scanner.nextLine();
        Analyzer analyzer = new IKAnalyzer();
        String queryType = null;
        switch (searchType) {
            case "a":
                queryType = "bookid";
                break;
            case "b":
                queryType = "title";
                break;
            case "c":
                queryType = "author";
                break;
            case "d":
                queryType = "type";
                break;
            case "e":
                queryType = "public";
                break;
            default:
                break;
        }
        QueryParser parser = new
QueryParser(Version.LUCENE_4_10_0, queryType, analyzer);
        System.out.println("请输入你想检索的内容：");
        String queryStr = scanner.nextLine();
        // 指定 field 为“name”，Lucene 会按照关键词搜索每个 doc 中的
name。

        Query query = parser.parse(queryStr);
        TopDocs hits = searcher.search(query, 1); // 前面几行代
码也是固定套路，使用时直接改 field 和关键词即可
```

```

        for (ScoreDoc doc : hits.scoreDocs) {
            Document d = searcher.doc(doc.doc);
            System.out.println(d.get("bookid"));
            System.out.println(d.get("title"));
            System.out.println(d.get("author"));
            System.out.println(d.get("type"));
            System.out.println(d.get("public"));
            System.out.println(d.get("time"));
            System.out.println(d.get("price"));
            System.out.println(d.get("editor"));
            System.out.println(d.get("abstract"));
            System.out.println(d.get("authorIntro"));
            System.out.println("目录: \n"+d.get("catalog"));
            System.out.println(d.get("picture"));
        }
    } catch (IOException | ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

详细说明：

通过 `switch` 函数判断用户想要根据哪一类限定来进行检索。一开始用的是 `int`，但是不知道为什么会报错，改成 `String` 后就没有问题了。

根据读者之后输入的检索内容，存储到 `queryStr` 中进行查询。

查询到符合条件的信息后在终端打印出所有信息。

4 测试与运行

4.1 程序测试

用 Eclipse 导出可执行的 `.jar` 文件，方便在终端操作显示内容。

在程序代码基本完成后，经过不断的调试与修改，最后测试本次所设计的图书搜索引擎能够正常运行，基本功能可以实现，但是在一些细节方面仍然需要完善，比如命令行进行爬虫的时候动态加载网页的过程中会显示大量警告，这些暂时没有找到原因解决；动态加载网页的时间较慢，导致爬虫时间较长，或许可以有更好的写法可以加快爬虫速度；当当网的价格编码类型是 UTF-8，终端可以显

示的中文编码类型是 GBK，所以终端显示出来的价格会显示？，在 Eclipse 控制台上修改编码类型才可以显示正确结果¥。总的来说本次设计在功能上已经基本达到要求，其他细节方面有待以后完善。

4.2 程序运行

命令行输入：`java -jar C:\JAVA\workspace\JavaHW4-new\BookCrawler.jar` 准备爬虫，中端显示提示“输入你想要爬取的当当网图书号范围”



```
PS C:\JAVA\workspace\JavaHW4-new> java -jar C:\JAVA\workspace\JavaHW4-new\BookCrawler.jar
输入你想要爬取的当当网图书号范围：
```

图3 准备爬虫

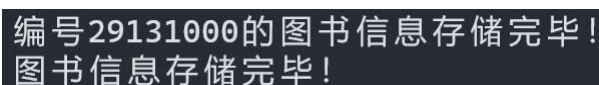
输入图书号范围，开始爬虫：



```
输入你想要爬取的当当网图书号范围：
29130000 29131000
```

图4 开始爬虫


爬虫结束后会显示提示：



```
编号29131000的图书信息存储完毕！
图书信息存储完毕！
```

图5 爬虫结束

命令行输入 `java -jar C:\JAVA\workspace\JavaHW4-new\BookSeacher.jar` 开始检索，终端显示“请输入你想检索的类型...”提示，根据提示和想要检索的类型输入 a, b, c, d, e



```
C:\JAVA\workspace\JavaHW4-new>java -jar C:\JAVA\workspace\JavaHW4-new\BookSeacher.jar
请输入你想检索的类型（a-书号，b-标题，c-作者，d-分类，e-出版社）：
```

图6 检索类型选取

输入检索类型书号，接着根据提示输入想要检索的内容，即对应书号（这次测试中爬取了 1000 本书，书号 01000 存在）：



```
a
请输入你想检索的内容：
29131000
```

图7 检索内容选取

显示检索结果：

29131000
标题：研学旅行在绍兴：亲近自然
作者：谢金土 、 胡仲德
所属分类： 图书 > 中小学教辅 > 中小学阅读 > 课外阅读
出版社：浙江科学技术出版社
出版时间：2020年06月
价格：？ 18.00
编辑推荐：

内容简介：

作者简介：

目录：

图片链接：
http://img3m0.ddimg.cn/52/12/29131000-1_w_3.jpg

图 8 检索结果展示

5. 总结

本次作业是我目前为止花费时间最长的一次作业。写完后我对于 **java** 网络爬虫、**java** 异常处理都有了更加深入的了解。**debug** 的过程非常痛苦，问题解决后也是很喜悦的。

（1）动态爬取网页详细信息

在花了一晚上学习了基础的 **Jsoup** 信息后我已经可以写出初步的 **Java** 爬虫代码，但不知为何爬取不到详细信息。

之后花了很久的时间去寻找合适的可以动态加载网页的 **jar** 包。修改别的博主写的调用 **htmlunit jar** 包爬取网页的框架，成功爬取到了详细信息，但是速度极慢无比，并且一旦断网就报错，需要重爬。然而校网很不稳定，在不断尝试了 2 天后最高爬虫记录维持在了 200+ 本。

（2）改进爬取时间

后来在写实验报告的时候再一次检查 **htmlunit** 部分的代码的时候想到爬取速度慢可能是等待网页加载时间过长原因，于是修改了超时时间、加载时间和等待时间，结果快多了。当把时间改得很小爬取了 1k 本后开始了查询，结果这次的问题是所有书本都没有显示详细信息了。才明白自己把时间修改得太小会导致动态加载未完成。于是又去参考了很多博主写的 **htmlunit** 代码，把原来冗余的代码修改了一番，并且反复测试一本书爬取到详细信息需要的最短时间，把最终的时间设置的比最短时间稍长一些，然后进行爬取。

（3）解决断网停爬问题

现在爬取速度比较令人满意了，但是一旦断网还是得重头再来。而校网并不能支持长时间爬取网页信息，所以断网重连后接着爬是必须的。

反复检查了 **java** 断网后的报错信息和上网查询，在发现我自己写的异常抛出和老师给的 **Lucene** 参考里面的异常抛出不同后，想到了应该是异常抛出的问题，如果令网络断开后程序仅仅是打印异常信息而不是停止运行，就不会停止爬虫。于是把所有页面写在类名后面的 **throw exception** 写到了类里面的函数里，并且使抓到异常的时候只是 `e.printStackTrace()`，即只会打印错误信息不会停止。

这一次断网或者电脑休眠都不会影响爬虫终止了，一旦连上就会继续。

（4） 解决部分书号页面不存在或部分书本信息不全问题

后来发现还是会有爬到一半停止的情况发生。根据写在 **for** 循环内部的输出信息我定位到出错的书本，发现了如题问题。

这个问题增加了 **Element** 值的 **null** 判断和 **string** 值的 **null** 判断便解决了，详细实现方式在第 3 部分已经写过，在此不再赘述。

参考文献

- [1] 《Jsoup (一) Jsoup 详解 (官方)》
<https://www.cnblogs.com/zhangyinhua/p/8037599.html>
- [2] 《使用 HtmlUnit + Jsoup 解析 动态网页》
https://blog.csdn.net/qq_32662595/article/details/88189584
- [3] 《Java 中异常问题 (异常抛出后是否继续执行的问题)》
https://blog.csdn.net/qq_33999844/article/details/88549982