网络爬虫快速实战

黑马程序员

目录

目录

第 1 章　爬虫基础概念

1.1 爬虫是什么

网络爬虫 编辑

本词条由"科普中国"百科科学词条编写与应用工作项目 审核 。

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

中文名	网络爬虫	别　称	网络蜘蛛
外文名	web crawler	目　的	按要求获取万维网信息

以上数据来源于 百度百科

爬虫又叫网络爬虫，网络蜘蛛，一种运行在互联网上用来获取数据的的自动程序。

- 互联网的数据，有很多，一般都是根据业务需求来的。
  - 网页（文字、图片、视频）
  - 商品数据
- 怎么获取数据？
  - HTTP 协议
  - 人的操是通过浏览器的，程序是利用网络请求的相关协议获取数据。
- 自动化，尽可能减少人工的干预。
  - 爬虫开发的技术，没有限制的。
    python 做网络爬虫是非常流行的。
    Java 编写爬虫框架。

思考题：某公司要获取京东的数据，也要获取淘宝的数据，又要获取豆瓣上的数据，请问开发

一个爬虫程序好？还是多个程序好？

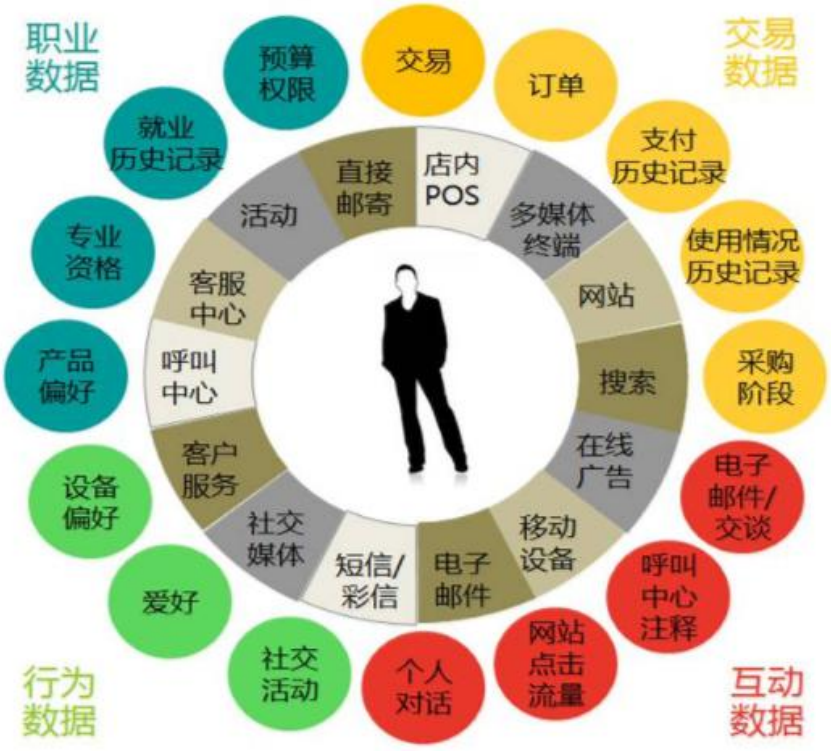
1.2 爬虫解决了什么问题

1.2.1 爬虫是为获取数据而存在的

一般会用来做数据分析，先通过对数据的清洗，抽取，转换，将数据做成标准化的数据，然后进行数据分析和挖掘，得到数据的商业价值。

1.2.2 数据分为内部数据和外部数据

在互联网公司，不管内部数据还是外部数据，其实都是为了获取用户相关的数据。
拿到用户的行为数据之后，会分析用户。
比如说电商类网站就是为推荐商品，搜索类的网站为了精准营销(家具类) 广告联盟。



1.2.2.1 公司内部数据

业务数据，公司使用 BI（Business Intelligence）、CRM 系统、ERP 系统、邮件系统等产生的数据；
财务数据，其中包括公司的支出、采购、收入等多项与公司日常运作有关的数据；
用户数据，无论是网站、APP 还是游戏，用户注册都会填写邮箱、电话、身份证号码等数据，这些数据其实非常有价值，此外还要加上用户使用公司产品留下的行为数据。
历史数据，公司沉淀下来的其他各种数据。

1.2.2.2 外部数据

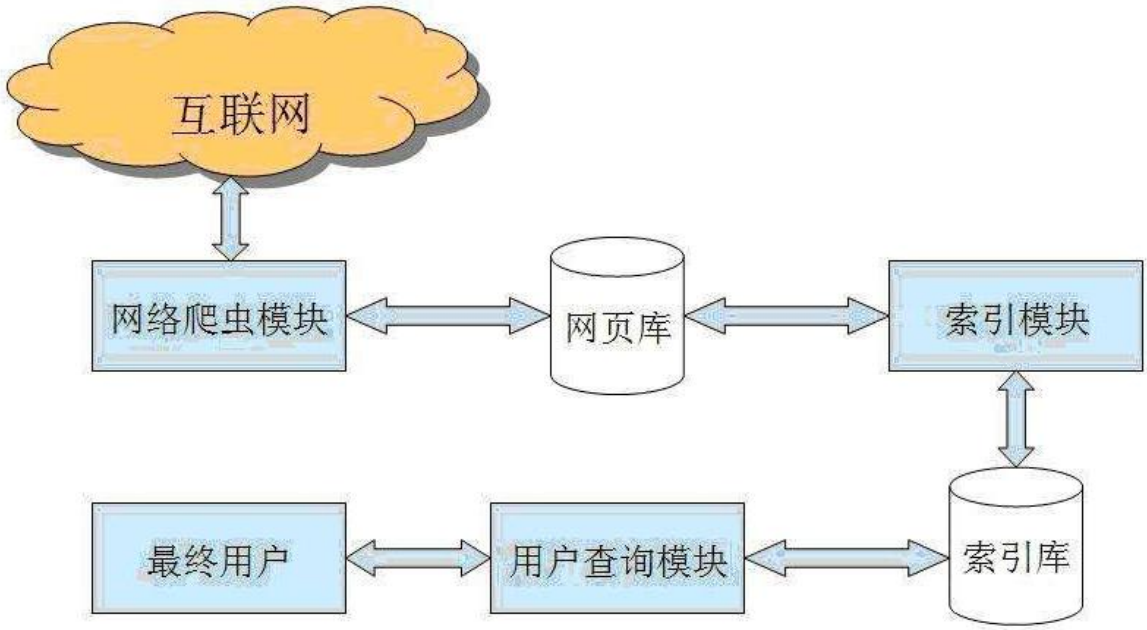
社交网站数据，包括微信、微博、人人网、Twitter、Facebook、LinkedIn 等社交媒体上的数据。
说明：社交数据部分是可以爬取的，另外一部分是需要运营方授权的。
线下采集数据，包括 Wifi 热点数据、地图数据等。
说明：这一块目前做的公司比较少，但同时也比较有价值。
政府开放数据，包括企业征信数据、企业注册数据、法院公示数据、公共交通数据等。
说明：如果你想要找的话，可到对应政府网站下载。
智能设备数据，包括智能设备、传感器数据。
说明：你知道吗？一部智能手机，至少拥有 8 个传感设备。
网络爬虫数据，包括互联网上所有可以爬回的数据，文字、视频、图片其实也是数据，而且是非结构化数据。
企业交易数据，包括商家流水数据、支付宝交易数据、信用卡消费数据等等。
说明：目前这一部分数据是最难获取的，因为数据就是宝贵的资产。

企业开放数据，比如微博开放了商业数据 API，腾讯开放了腾讯云分析 SDK 上报的应用数据，高德地图开放了 LBS 数据等等。
说明：如果想找更多的数据 API，我推荐你去数据堂、聚合数据这两家网站上看一下，上面有大量的 API 接口。
其它数据，比如天气数据、交通数据、人口流动数据、位置数据等等。
说明：只有想不到没有弄不到。

1.2.2.3 额外扩展

大数据就是整合完公司内部外部数据，进行大数据存储，然后通过清洗，标注、去重、去噪、关联等过程可以将数据进行结构化，也可以进行大数据挖掘和数据分析，再以数据可视化呈现结果，打通数据孤岛形成数据闭环，将数据转换成“石油”和“生产资料”，最后应用到我们日常的生活、学习和工作中去。

1.3 爬虫与搜索系统的关系



搜索系统的数据是爬虫爬取过来？不一定。

搜索系统可以简单的分为两类，**通用搜索，站内搜索**。

通用搜索：像百度，谷歌会爬取互联网上所有的数据
站内搜索：只需要业务系统的数据。
垂直搜索：行业数据和自己的数据。

总结：搜索一定会包含爬虫（除站内搜索外），爬虫爬取的数据不一定是为搜索服务。除了搜索功

能以外，爬虫爬取的数据主要用来做数据分析。

1.4 爬虫的简单分类

网络爬虫按照系统结构和实现技术，大致可以分为以下几种类型：

- 通用网络爬虫（General Purpose Web Crawler）
- 聚焦网络爬虫（Focused Web Crawler）
- 增量式网络爬虫（Incremental Web Crawler）
- 深层网络爬虫（Deep Web Crawler）

实际的网络爬虫系统通常是几种爬虫技术相结合实现的。

1.4.1 通用网络爬虫

通用网络爬虫又称全网爬虫（Scalable Web Crawler），爬行对象从一些种子 URL 扩充到整个 Web，主要为门户站点搜索引擎和大型 Web 服务提供商采集数据。

由于商业原因，它们的技术细节很少公布出来。

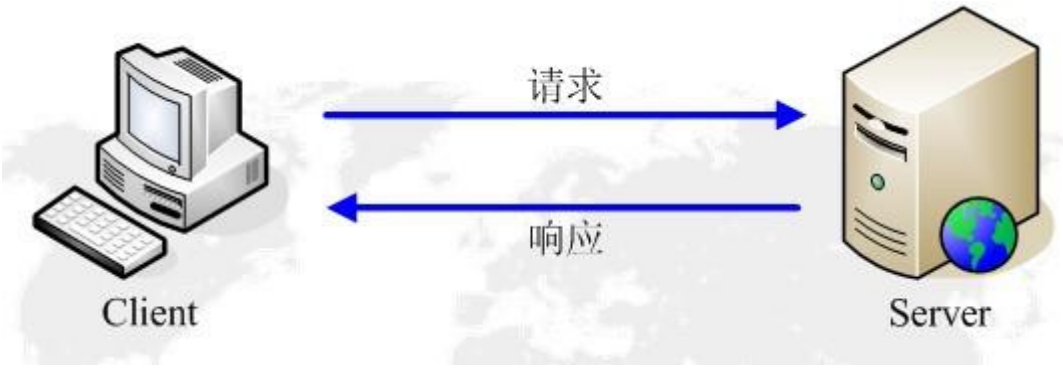
这类网络爬虫的爬行范围和数量巨大，对于爬行速度和存储空间要求较高，对于爬行页面的顺序要求相对较低，同时由于待刷新的页面太多，通常采用并行工作方式，但需要较长时间才能刷新一次页面。 虽然存在一定缺陷，通用网络爬虫适用于为搜索引擎搜索广泛的主题，有较强的应用价值。

1.4.2 聚焦网络爬虫

聚焦网络爬虫（Focused Crawler），又称主题网络爬虫（Topical Crawler），是指选择性地爬行那些与预先定义好的主题相关页面的网络爬虫。

和通用网络爬虫相比，聚焦爬虫只需要爬行与主题相关的页面，极大地节省了硬件和网络资源，保存的页面也由于数量少而更新快，还可以很好地满足一些特定人群对特定领域信息的需求。

1.5 爬虫的运行原理



1.5.1 爬虫开发本质是 HTTP 请求

使用 HTTP GET 协议获取数据，使用 HTTP POST 协议提交数据。

客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本、以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。

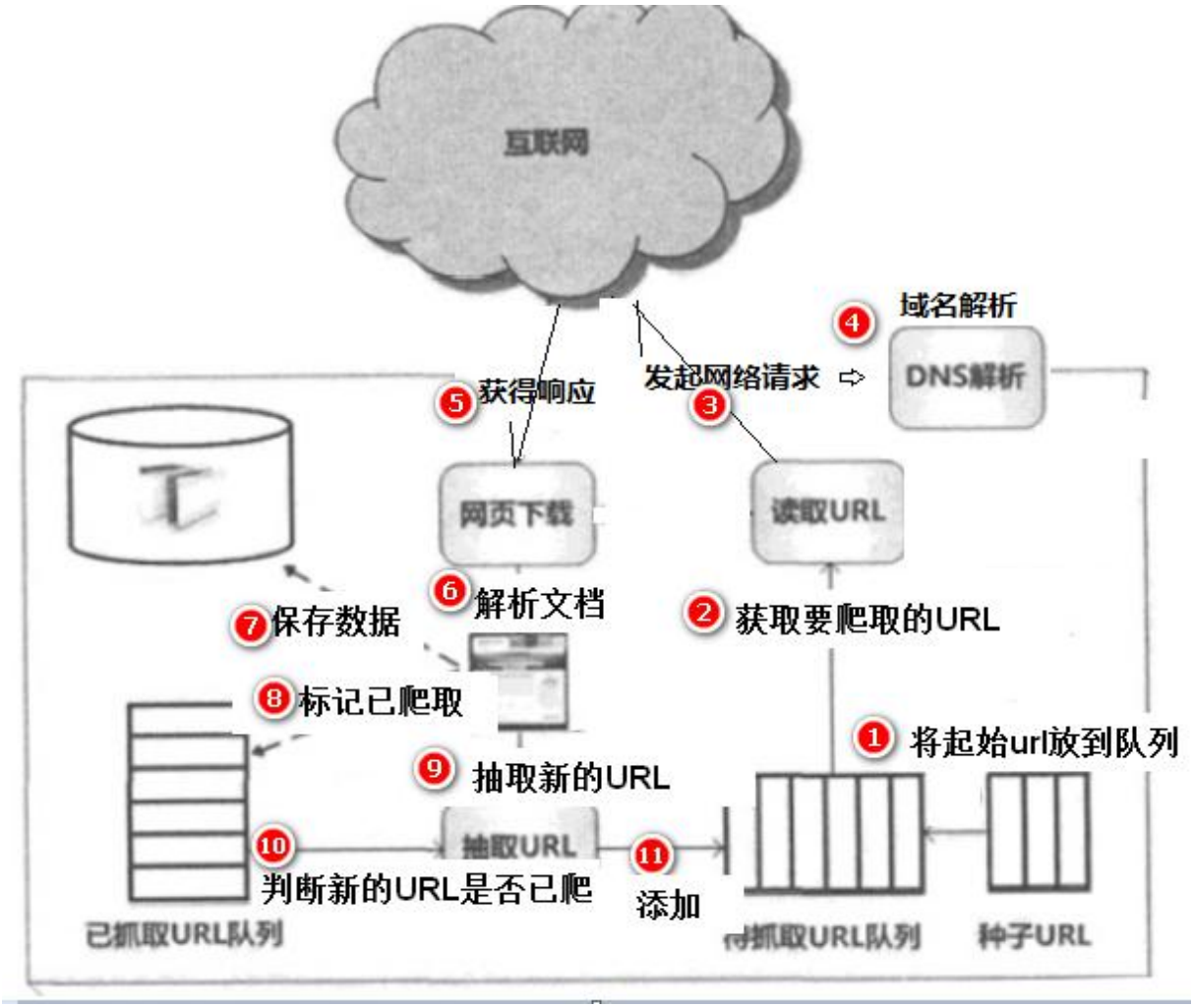
服务器以一个状态行作为响应，响应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。

通常 HTTP 消息包括客户机向服务器的请求消息和服务器向客户机的响应消息。这两种类型的消息由一个起始行，一个或者多个头域，一个指示头域结束的空行和可选的消息体组成。

1.5.2 HTTP 常见状态码

所有的状态码:百度百科

1.5.3 爬虫运行的原理

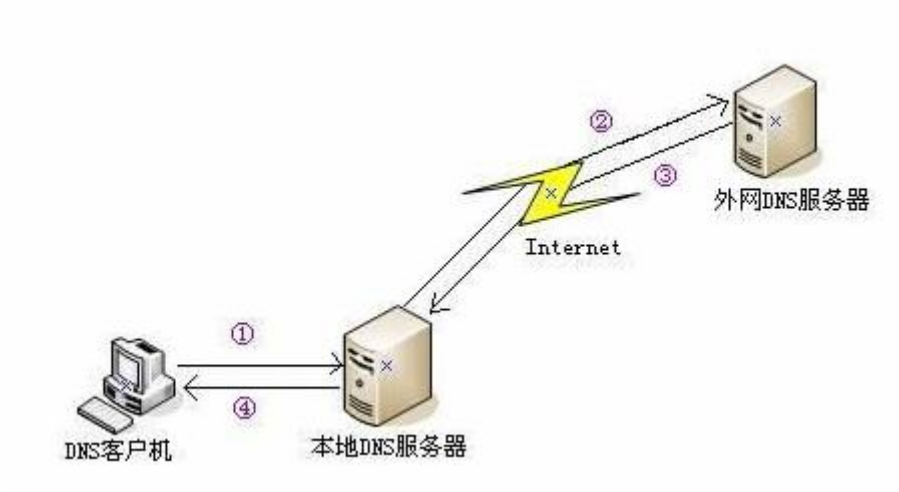


① 指定一个种子 url 放入到队列中
② 从队列中获取某个 URL
③ 使用 HTTP 协议发起网络请求
④ 在发起网络请求的过程中，需要将域名转化成 IP 地址，也就是域名解析
⑤ 得到服务器的响应，此时是二进制的输入流

⑥ 将二进制的输入流转换成 HTML 文档，并解析内容（我们要抓取的内容，比如标题）。
⑦ 将解除出来的内容保持到数据库
⑧ 记录当前 URL，并标记为已爬取，避免下次重复爬取。
⑨ 从当前的 HTML 文档中，解析出页面中包含的其它 URL，以供下次爬取
⑩ 判断解析出来的 URL 是否已经爬取过了，如果已经爬取就丢弃掉
⑪ 将还没爬取过的 URL，存放到等待爬取的 URL 队列中。
⑫ 重复以上的步骤，指导等待爬取的 URL 队列中没有数据

1.5.4 DNS 域名解析了解

DNS（Domain Name System，域名系统），因特网上作为域名和 IP 地址相互映射的一个分布式数据库，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的 IP 数串。
通过主机名，最终得到该主机名对应的 IP 地址的过程叫做域名解析（或主机名解析）。
DNS 协议运行在 UDP 协议之上，使用端口号 53。



更多信息请访问百度百科，DNS

第 2 章　爬虫的开发基础基础

2.1 如何模拟浏览器进行网络请求

本质上就是发送 HTTP 请求

2.1.1 Java 网络请求原生 API-Get 请求

使用原生 API 发送 Get 请求

```
package cn.itcast.spider;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

/**
 使用 JDK 的 api 进行 get 请求

 1.在使用 httpurlconnection 时，默认就是 get 请求。如何改成 post 请求？
 2.http 协议中，可以指定 header，想添加 user-agent

 */
public class BasicHttpGet {

   public static void main(String[] args) throws Exception {
      //1.指定一个 url
      String domain = "http://www.itcast.cn";
      //2.发起一个请求
      URL url = new URL(domain);
      HttpURLConnection conn = (HttpURLConnection)url.openConnection();

      //添加请求方式
      conn.setRequestMethod("GET");

      //添加请求头------如果编写爬虫，真实浏览器发送的 header 都拷贝
      conn.setRequestProperty("Accept", "text/html");
      /**
       Accept:text/html
       **/

      //3.获取返回值
      InputStream inputStream = conn.getInputStream();
      //3.1 将输入流转换字符串
      BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
      //3.2 一次读取 bufferReader 的数据
      String line =null;
      while((line=bufferedReader.readLine())!=null){
         System.out.println(line);
      }
      //4.关闭流
```

```
        inputStream.close();
    }

}
```

2.1.2 Java 网络请求原生 API-Post 请求

```
package cn.itcast.spider;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

/**
 使用 JDK 的 api 进行 POST 请求

 1.在使用 httpurlconnection 时，默认就是 get 请求。如何改成 post 请求？

 第一步：设置请求方法 setRequestMethod("POST")
 第二步：设置 doOutPut(true)

 2.http 协议中，可以指定 header，想添加 user-agent
 */
public class BasicHttpGet {

    public static void main(String[] args) throws Exception {
        //1.指定一个 url
        String domain = "http://www.itcast.cn";
        //2.发起一个请求
        URL url = new URL(domain);
        HttpURLConnection conn = (HttpURLConnection)url.openConnection();

        //2.1 添加请求方式
        conn.setRequestMethod("POST");
        //2.2 添加请求头------如果编写爬虫，真实浏览器发送的 header 都拷贝
        conn.setRequestProperty("Accept", "text/html");
        /**
         Accept:text/html
         **/
```

```
        //2.3 发送一些数据
        conn.setDoOutput(true);
        OutputStream outputStream = conn.getOutputStream();
        // 编写什么样格式的数据？  username=zhangsan&passwd=123
        outputStream.write("username=zhangsan&passwd=123".getBytes());
        outputStream.flush();
        outputStream.close();

        //3.获取返回值
        InputStream inputStream = conn.getInputStream();
        //3.1 将输入流转换字符串
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
        //3.2 一次读取 bufferReader 的数据
        String line =null;
        while((line=bufferedReader.readLine())!=null){
            System.out.println(line);
        }
        //4.关闭流
        inputStream.close();
    }
}
```

2.1.3 使用 HttpClient

HttpClient 是 Apache Jakarta Common 下的子项目，可以用来提供高效的、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。

HtppClient 官网

为什么有 HttpClient?

- 超文本传输协议（HTTP）可能是当今互联网上使用的最重要的协议
- 虽然 java.net 包提供了通过 HTTP 访问资源的基本功能，但它并没有提供许多应用程序所需的完全灵活性或功能

使用 HttpClient 的 maven 依赖

```
<dependencies>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.3</version>
    </dependency>
    <dependency>
```

```
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>fluent-hc</artifactId>
        <version>4.5.3</version>
    </dependency>
</dependencies>
```

2.1.4 使用 HttpClient 进行 Get 请求

```
package cn.itcast.spider.httpClient;

import java.nio.charset.Charset;

import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

/**
 使用 httpclient 来进行 get 请求
 */
public class HCGet {
   public static void main(String[] args) throws Exception {
      // 1.拿到一个 httpclient 的对象
      CloseableHttpClient httpClient = HttpClients.createDefault();
      // 2.设置请求方式和请求信息
      HttpGet httpGet = new HttpGet("http://www.itcast.cn");
      // 3.执行请求
      CloseableHttpResponse response = httpClient.execute(httpGet);
      // 4.获取返回值
      String html =
EntityUtils.toString(response.getEntity(),Charset.forName("utf-8"));
      // 5.打印
      System.out.println(html);

   }
}
```

2.1.5 使用 HttpClient 进行 Post 请求

```
package cn.itcast.spider.httpClient;

import org.apache.http.client.entity.UrlEncodedFormEntity;
```

```
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;

import java.nio.charset.Charset;
import java.util.ArrayList;

/**
 使用 httpclient 来进行 post 请求
 */
public class HCPost {
   public static void main(String[] args) throws Exception {
       // 1.拿到一个 httpclient 的对象
       CloseableHttpClient httpClient = HttpClients.createDefault();
       // 2.设置请求方式和请求信息
//     HttpGet httpGet = new HttpGet("http://www.itcast.cn");
       HttpPost httpPost = new HttpPost("http://www.itcast.cn");

       //2.1 提交 header 头信息
       httpPost.addHeader("user-agent", "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36");
       //2.1 提交请求体
       //提交方式 1:一般用在原生 ajax 请求
//     httpPost.setEntity(new StringEntity("username=zhangsan&passwd=123"));
       //提交方式 2：大多数的情况下用这种方式
       ArrayList<BasicNameValuePair> parameters = new
ArrayList<BasicNameValuePair>();
       parameters.add(new BasicNameValuePair("username", "zhangsan"));
       parameters.add(new BasicNameValuePair("passwd", "123"));
       httpPost.setEntity(new UrlEncodedFormEntity(parameters));

       // 3.执行请求
       CloseableHttpResponse response = httpClient.execute(httpPost);
       // 4.获取返回值
       String html =
EntityUtils.toString(response.getEntity(),Charset.forName("utf-8"));
       // 5.打印
       System.out.println(html);

   }
}
```

2.1.6 使用 HttpClient 的流畅 API（了解）

```
package cn.itcast.spider.httpClient;

import java.nio.charset.Charset;
import org.apache.http.client.fluent.Request;
/**
 流畅的 api
 *
 */
public class FluentHttpClient {
   public static void main(String[] args) throws Exception {
      String html =
Request.Get("http://www.itcast.cn").execute().returnContent().asString(Char
set.forName("utf-8"));
      System.out.println(html);

//    Request.Post("http://targethost/login")
//          .bodyForm(Form.form().add("username",  "vip").add("password",
"secret").build()).execute()
//          .returnContent();
   }
}
```

2.2 如何解析爬虫爬取回来的数据

2.2.1 使用 Document 获取文本数据

2.2.1.1 Document 是什么

每个载入浏览器的 HTML 文档都会成为 Document 对象。
Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

2.2.1.2 Document 对象集合

- all[] 提供对文档中所有 HTML 元素的访问。
- anchors[] 返回对文档中所有 Anchor 对象的引用。

- applets 返回对文档中所有 Applet 对象的引用。
- forms[] 返回对文档中所有 Form 对象引用。
- images[] 返回对文档中所有 Image 对象引用。
- links[] 返回对文档中所有 Area 和 Link 对象引用。

2.2.1.3 Document 对象属性

- body 提供对 元素的直接访问。 对于定义了框架集的文档，该属性引用最外层的 。
- cookie 设置或返回与当前文档有关的所有 cookie。
- domain 返回当前文档的域名。
- lastModified 返回文档被最后修改的日期和时间。
- referrer 返回载入当前文档的文档的 URL。
- title 返回当前文档的标题。
- URL 返回当前文档的 URL。

2.2.1.4 Document 对象方法

- close() 关闭用 document.open() 方法打开的输出流，并显示选定的数据。
- getElementById() 返回对拥有指定 id 的第一个对象的引用。
- getElementsByName() 返回带有指定名称的对象集合。
- getElementsByTagName() 返回带有指定标签名的对象集合。
- open() 打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出。
- write() 向文档写 HTML 表达式 或 JavaScript 代码。
- writeln() 等同于 write() 方法，不同的是在每个表达式之后写一个换行符。

2.2.1.5 在 Chrome Browse 中操作 Document

2.2.2 使用 Jsoup 操作 HTML 文档

2.2.2.1 Jsoup 是什么？

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。

官网地址

```
<dependency>
    <!-- jsoup HTML parser library @ https://jsoup.org/ -->
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>1.10.3</version>
</dependency>
```

2.2.2.2 JSOUP 的核心功能

- JSOUP 的输入信息
  字符串
  文件
  URL
- JSOUP 抽取数据
  使用 Document 语法进行解析数据
  使用选择器的方式进行解析（学习对象）

2.2.2.3 如何使用 Jsoup 选择器获取数据

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。

Selector 选择器概述

- tagname: 通过标签查找元素，比如：a

- ns|tag: 通过标签在命名空间查找元素，比如：可以用 fb|name 语法来查找 元素
- id: 通过 ID 查找元素，比如：#logo
- class: 通过 class 名称查找元素，比如：.masthead
- [attribute]: 利用属性查找元素，比如：[href]
- [^attr]: 利用属性名前缀来查找元素，比如：可以用[^data-] 来查找带有 HTML5 Dataset 属性的元素
- [attr=value]: 利用属性值来查找元素，比如：[width=500]
- [attr^=value], [attr$=value], [attr=value]: 利用匹配属性值开头、结尾或包含属性值来查找元素，比如：[href=/path/]
- [attr~=regex]: 利用属性值匹配正则表达式来查找元素，比如： img[src~=(?i).(png|jpe?g)]
- *: 这个符号将匹配所有元素
- Seector 选择器组合使用
- el#id: 元素+ID，比如： div#logo
- el.class: 元素+class，比如： div.masthead
- el[attr]: 元素+class，比如： a[href]
- 任意组合，比如：a[href].highlight
- ancestor child: 查找某个元素下子元素，比如：可以用.body p 查找在"body"元素下的所有 p 元素
- parent > child: 查找某个父元素下的直接子元素，比如：可以用 div.content > p 查找 p 元素，也可以用 body > 查找 body 标签下所有直接子元素
- siblingA + siblingB: 查找在 A 元素之前第一个同级元素 B，比如：div.head + div
- siblingA ~ siblingX: 查找 A 元素之前的同级 X 元素，比如：h1 ~ p
- el, el, el:多个选择器组合，查找匹配任一选择器的唯一元素，例如：div.masthead, div.logo

伪选器 selectors

- :lt(n): 查找哪些元素的同级索引值（它的位置在 DOM 树中是相对于它的父节点）小于 n，比如：td:lt(3) 表示小于三列的元素
- :gt(n):查找哪些元素的同级索引值大于 n，比如： div p:gt(2)表示哪些 div 中有包含 2 个以上的 p 元素
- :eq(n): 查找哪些元素的同级索引值与 n 相等，比如：form input:eq(1)表示包含一个 input 标签的 Form 元素
- :has(seletor): 查找匹配选择器包含元素的元素，比如：div:has(p)表示哪些 div 包含了 p 元素
- :not(selector): 查找与选择器不匹配的元素，比如： div:not(.logo) 表示不包含 class=logo 元素的所有 div 列表
- :contains(text): 查找包含给定文本的元素，搜索不区分大不写，比如： p:contains(jsoup)
- :containsOwn(text): 查找直接包含给定文本的元素
- :matches(regex): 查找哪些元素的文本匹配指定的正则表达式，比如：div:matches((?i)login)
- :matchesOwn(regex): 查找自身包含文本匹配指定正则表达式的元素
- 注意：上述伪选择器索引是从 0 开始的，也就是说第一个元素索引值为 0，第二个元素 index 为 1 等

2.2.2.4 HTML 页面解析实战

```
package cn.itcast.spider;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

/**
 使用 Jsoup 解析文档
 *
 */
public class JsoupTest {
   public static void main(String[] args) throws Exception {
      //1.输入方式 字符串、文件、html 片段、url
      Document doc = Jsoup.connect("http://www.163.com/").get();
      //2、通过标签名称访问元素
//    Elements elements = document.select("a[href=http://www.52ij.com/]");

//    通过 id 访问元素

//    Elements els = doc.select("#js_N_navHighlight");

//    Elements els = doc.select(".JS_NTES_LOG_FE");

//    Elements els = doc.select("[^ne-c]");

      //<ul class="ntes-quicknav-column ntes-quicknav-column-11">

      Elements els = doc.select("ul[class=ntes-quicknav-column
ntes-quicknav-column-11] > h3");

      //3、打印
      for (Element element : els) {
         System.out.println(element);
      }
   }

}
```

第 3 章　后台登录与数据收集

3.1 运行目标网站

参见资料【tomcat4shop_p80.zip】

① 解压压缩包 tomcat4shop_p80.zip
② 进入 tomcat4shop_p80\webapps\shop\WEB-INF\classes 找到数据库文件 ssm.sql
③ 打开虚拟机，运行数据库软件
④ 导入建表语句
⑤ 修改数据库连接文件\tomcat4shop_p80\webapps**ROOT**\WEB-INF\classes\properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://mysql:3306/ssm?characterEncoding=utf-8
jdbc.username=root
jdbc.password=root
```

⑥ 修改数据库密码

```
use mysql;
UPDATE user SET Password = PASSWORD('root') WHERE user = 'root';
flush privileges;
```

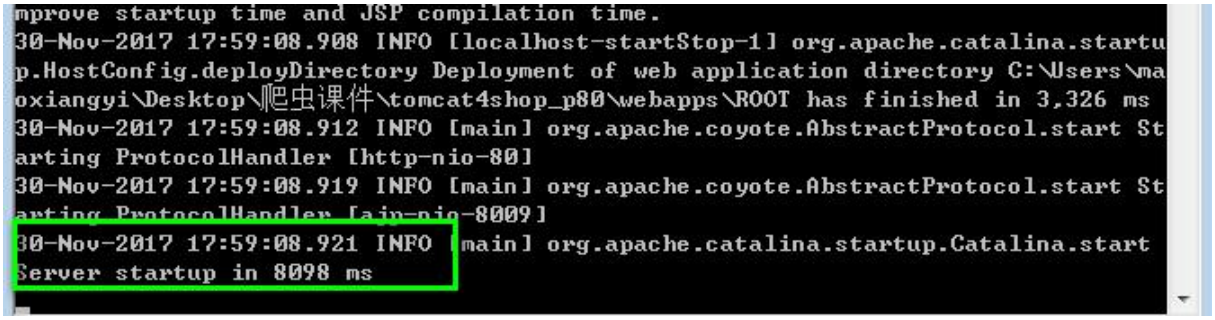
⑦ 开启数据库远程连接权限

```
use mysql;
grant all privileges on *.to root@"%" identified by "root" with grant option;
flush privileges;
```

⑧ 修改 hosts 信息

```
192.168.140.130 mysql
127.0.0.1 shop.itcast.cn
```

⑨ 启动 tomcat 服务
运行 tomcat4shop_p80\bin\startup.bat



⑩ 然后访问浏览器:

3.2 目标网站分析

3.2.1 登录界面分析

```
<form action="/login/login.html" method="post">
    <input type="hidden" name="reURL" value="http://shop.itcast.cn/item/itemList.html">
    <label>账户</label>
    <input type="text" name="username">
    <br>
    <label>密码</label>
    <input type="password" name="password">
    <br>
    <input type="submit" value="login">
</form>
```

请求的 URL：http://shop.itcast.cn/login/login.html

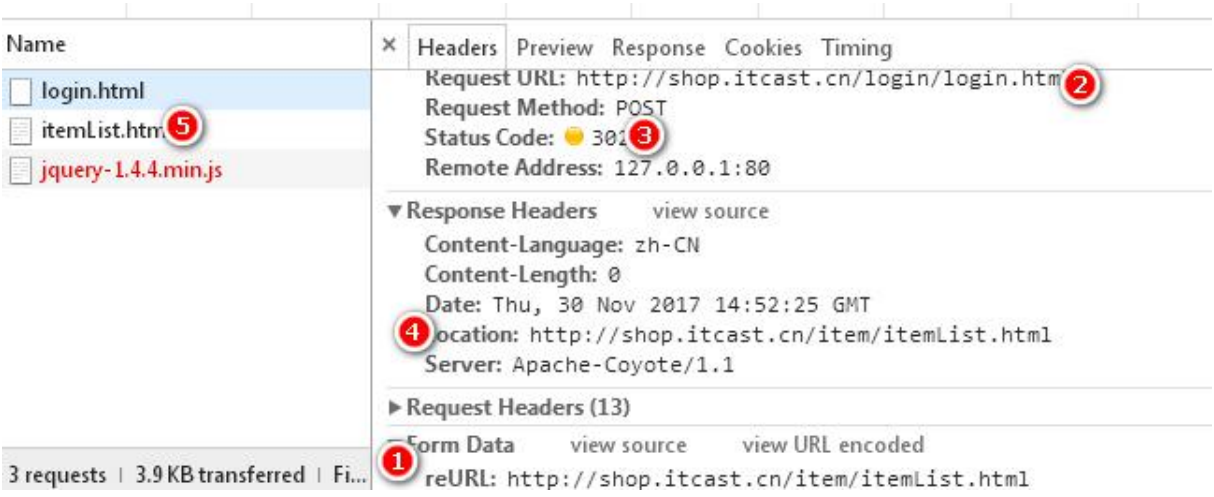
请求的参数：

第一个参数：reURL= http://shop.itcast.cn/item/itemList.html

第二个参数：username = itcast

第三个参数：password = itcast

3.2.2 登录成功重定向



① 客户端发起请求，携带三个参数

② 服务端接收到请求之后，进行登录处理。

③ 登录成功逻辑执行完毕之后，服务端返回 302

④ 302 重定向的地址是: http://shop.itcast.cn/item/itemList.html

⑤ 客户端根据 302 的地址，再次发起请求，得到商品列表页。

3.3 使用爬虫登录并获取数据

3.3.1 导入 pom 依赖

```
<dependencies>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>fluent-hc</artifactId>
        <version>4.5.3</version>
    </dependency>
    <dependency>
        <!-- jsoup HTML parser library @ https://jsoup.org/ -->
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.10.3</version>
    </dependency>
</dependencies>
```

3.3.2 核心代码编写

```
package cn.itcast.spider.login;

import java.util.ArrayList;

import org.apache.http.Header;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
```

```
import org.jsoup.select.Elements;

/**
 登录之后获取数据
 *
 */
public class Login {
   public static void main(String[] args) throws Exception {
      // 1.分析请求的 url 的信息
      // 找到 POST 请求的 url:http://shop.itcast.cn/login/login.html
      // 分析出提交的三个参数:username,password,reURL
      String url = "http://shop.itcast.cn/login/login.html";
      HttpPost httpPost = new HttpPost(url);

      // 2.设置登录的参数
      // 传递参数的方式 1.Stringentity
      // 传递参数的方式 2.basicnameandvalupair
      ArrayList<BasicNameValuePair> parameters = new
ArrayList<BasicNameValuePair>();
      parameters.add(new BasicNameValuePair("username", "itcast"));
      parameters.add(new BasicNameValuePair("password", "itcast"));
      parameters.add(new BasicNameValuePair("reURL",
"http://shop.itcast.cn/item/itemList.html"));
      httpPost.setEntity(new UrlEncodedFormEntity(parameters));

      // 3.获取一个 httpclient 对象，用来执行 http 请求
      CloseableHttpClient httpclient = HttpClients.createDefault();
      // 4.执行请求，并得到返回对象
      CloseableHttpResponse response = httpclient.execute(httpPost);

      // 获取状态码 302 重定向
      // 5.获取重定向的跳转的 location 地址
      String location = null;
      if (response.getStatusLine().getStatusCode() == 302) {
         Header[] allHeaders = response.getAllHeaders();
         for (Header header : allHeaders) {
            if (header.getName().equals("Location")) {
               location = header.getValue();
            }
         }
      }
      /**
       登录之后，要记录登录的状态，有几种方式。 session 中 jsession
       */
```

```
        // 6.根据登录之后返回的 location 地址，获取数据
        CloseableHttpClient hc1 = HttpClients.createDefault();
        CloseableHttpResponse res1 = hc1.execute(new HttpGet(location));
        if (res1.getStatusLine().getStatusCode() == 200) {
            String html = EntityUtils.toString(res1.getEntity());
            // Jsoup 解析
            Document doc = Jsoup.parse(html);
            // 6.1 获取到商品列表所在的 form 表单
            Elements elements =
doc.select("form[action=/item/batchUpdate.html]");
            // 6.2 获取到所有的 tr 标签，发现有个 tr 标签的表头
            elements = elements.select("tr");
            ArrayList<Product> arrayList = new ArrayList<Product>();
            // 6.3 迭代所有的的 tr
            for (int i = 1; i < elements.size(); i++) {
                Product product = new Product();
                Element element = elements.get(i);
                // 6.3.1 获取 id
                // <input type="hidden" name="itemList[0].id"
                Elements ids = element.select("input[type=hidden]");
                for (Element id : ids) {
                    String value = id.attr("value");
                    int idValue = Integer.parseInt(value) - 1;
                    product.setId(idValue+"");
                }
                // 6.3.2 获取标题
                Elements names = element.select("input[name$=name]");
                product.setName(names.get(0).attr("value"));
                // 6.3.3 获取价格
                Elements prices = element.select("input[name$=price]");
                product.setPrice(prices.get(0).attr("value"));
                // 6.3.4 获取发布时间
                Elements createtimes =
element.select("input[name$=createtime]");
                product.setCreateTime(createtimes.get(0).attr("value"));
                // 6.3.5 获取描述信息
                Elements details = element.select("input[name$=detail]");
                product.setDes(details.get(0).attr("value"));
                arrayList.add(product);
            }
            //7.保存数据
            save2db(arrayList);
        }
```

```
    }

    /**
     伪实现
     @param arrayList
     */
    private static void save2db(ArrayList<Product> arrayList) {
        for (Product product : arrayList) {
            System.out.println(product);
        }

    }
}
```

3.3.3 使用到的 Product 类

```
package cn.itcast.spider.login;

public class Product {

    private String id;
    private String name;
    private String price;
    private String createTime;
    private String des;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPrice() {
        return price;
    }
    public void setPrice(String price) {
```

```
        this.price = price;
    }
    public String getCreateTime() {
        return createTime;
    }
    public void setCreateTime(String createTime) {
        this.createTime = createTime;
    }
    public String getDes() {
        return des;
    }
    public void setDes(String des) {
        this.des = des;
    }
    @Override
    public String toString() {
        return "Product [id=" + id + ", name=" + name + ", price=" + price + ",
createTime=" + createTime + ", des="
                + des + "]";
    }

}
```

第 4 章　新闻网站爬取

案例实战：爬取虎嗅网站的数据，并保存到数据库。

4.1 网站爬取分析

4.1.1 需求说明

网址：https://www.huxiu.com/
类型：自媒体网站
数据爬取的范围：
首页的新闻列表--->对应的每篇文章的信息
分页的新闻列表--->对应的每篇文章的信息
文章信息：标题、作者、时间、收藏、评论、正文、URL

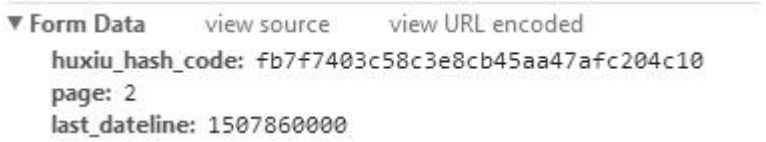
4.1.2 功能分析

首页解析

1）网址：https://www.huxiu.com/
2）发现文章列表的 class 是 mod-info-flow
3）发现单篇文章的 URL 的组成是 huxiu.com/article/{aid}.html
4）发现每个列都是由一个 div 组成，div 有上有个属性叫做 dataid,比如`data-aid="217969"`

解析分页

1）网址：https://www.huxiu.com/v2_action/article_list
2）ajax 异步请求提交了三个参数



参数说明

huxiu_hash_code:fb7f7403c58c3e8cb45aa47afc204c10

浏览器的唯一标识，在虎嗅服务器端生成。

page:2

分页的页码号

last_dateline:1507860000

时间标记

3) ajax 异步请求的返回值分析

参数说明

last_dateline:"1507762380"

msg:"获取成功"
result:1
total_page:1615
data:所有数据

注意，分页请求成功之后，会有一个 data 字段，这个字段的内容就是分页的 html 文档，直接将 html 文档追加到首页的内容上即可实现分页功能。

4.1.3 构建 Maven 工程

```
<dependencies>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.3</version>
    </dependency>
    <!-- jsoup HTML parser library @ https://jsoup.org/ -->
    <dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.10.3</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc
-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>4.2.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.41</version>
    </dependency>
    <dependency>
        <groupId>c3p0</groupId>
        <artifactId>c3p0</artifactId>
        <version>0.9.1.2</version>
    </dependency>

    <dependency>
```

```
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.31</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.1</version>
    </dependency>
</dependencies>
```

4.2 代码开发

4.2.1 主体代码

```
package cn.itcast.spider.huxiu;

import java.io.IOException;
import java.nio.charset.Charset;
import java.util.ArrayList;

import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import com.google.gson.Gson;

/**
 * 1、解析首页 2、实现分页的代码
```

```
 */
public class HuxiuSpider {

   private static ArticleDao articleDao = new ArticleDao();
   private static String dateLine = null;

   public static void main(String[] args) throws IOException {
       // 抽取方法的快捷键：alt+shift+M
       ArrayList<String> indexArticleIdList = getIndexArticleUrlList();
       // 获取首页文章列表对应的新闻列表
       ArrayList<Article> indexArticleListContent =
getIndexArticleListContent(indexArticleIdList);
       // 打印
       for (Article article : indexArticleListContent) {
       articleDao.save(article);
       }
      for (int page = 2; page <= 1615; page++) {
         try {
            System.out.println(page);
            // 编写分页
            String pagingUrl =
"https://www.huxiu.com/v2_action/article_list";
            HttpPost httpPost = new HttpPost(pagingUrl);
            // 设置参数
            ArrayList<NameValuePair> arrayList = new
ArrayList<NameValuePair>();
            arrayList.add(new BasicNameValuePair("huxiu_hash_code",
"fb7f7403c58c3e8cb45aa47afc204c10"));
            arrayList.add(new BasicNameValuePair("page", page + ""));
            arrayList.add(new BasicNameValuePair("last_dateline",
dateLine));
            httpPost.setEntity(new UrlEncodedFormEntity(arrayList));
            // 执行网络参数
            String jsonText = getHtmlByRequest(httpPost);
            // 想将 json 串转成对象
            Gson gson = new Gson();
            HuxiuPagingResponse huxiuPagingResponse = gson.fromJson(jsonText,
HuxiuPagingResponse.class);
            // 每一次请求，都需要解析出新的 dataLine
            dateLine = huxiuPagingResponse.getLast_dateline();
            // 获取数据
            String htmlData = huxiuPagingResponse.getData();

            Document doc = Jsoup.parse(htmlData);
```

```
            // 解析出 div 的某个属性 data-id
            Elements aidElements = doc.select("div[data-aid]");
            // 依次得到每个新闻的 aid
            ArrayList<String> idList = new ArrayList<String>();
            for (Element element : aidElements) {
               String aid = element.attr("data-aid");
               idList.add(aid);
            }

            // 一次爬取
            ArrayList<Article> pagingArticleList =
getIndexArticleListContent(idList);
            for (Article article : pagingArticleList) {
               articleDao.save(article);
            }
         } catch (Exception e) {
            //log.errer()
            System.out.println(page);
            System.out.println(e);
         }
      }

   }

   /**
    * 根据文件列表 一次获取文章信息
    *
    * @param indexArticleUrlList
    *            文章的 url
    * @return
    * @throws IOException
    * @throws ClientProtocolException
    */
   private static ArrayList<Article>
getIndexArticleListContent(ArrayList<String> indexArticleIdList)
         throws IOException, ClientProtocolException {
      ArrayList<Article> articleList = new ArrayList<Article>();
      // 依次访问每个页面
      for (String aid : indexArticleIdList) {
         String aidUrl = "http://www.huxiu.com/article/" + aid + ".html";
         try {
            Article article = new Article();
            article.setId(aid);
            // 获取到单个新闻页面的 html
```

```
            String aidHtml = getHtml(aidUrl);
            Document detailDocument = Jsoup.parse(aidHtml);
            // 解析文章 title
            Elements titles = detailDocument.select(".t-h1");
            String title = titles.get(0).text();
            article.setTitle(title);
            // 解析文章 author author-name
            Elements names = detailDocument.select(".author-name");
            String name = names.get(0).text();
            article.setAuthor(name);
            // 解析文章发布时间
            Elements dates = detailDocument.select("[class^=article-time]");
            String date = dates.get(0).text();
            article.setCreateTime(date);
            // 解析文章 收藏数
            Elements shares =
detailDocument.select("[class^=article-share]");
            String share = shares.get(0).text();
            article.setSc(share);
            // 解析文章 评论数
            Elements pls = detailDocument.select("[class^=article-pl]");
            String pl = pls.get(0).text();
            article.setPl(pl);
            // 解析文章 点赞数 num
            Elements nums = detailDocument.select(".num");
            String num = nums.get(0).text();
            article.setZan(num);
            // 解析文章正文内容 article-content-wrap
            Elements content = detailDocument.select(".article-content-wrap
p");
            String contentText = content.text();
            article.setContent(contentText);
            article.setUrl(aidUrl);

            articleList.add(article);
        } catch (Exception e) {
            System.out.println(aidUrl);
            System.out.println(e);
        }

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            }
        }
        return articleList;
    }

    /**
     * 获取 html 文档
     *
     * @throws IOException
     * @throws ClientProtocolException
     */
    private static String getHtml(String aidUrl) throws IOException,
ClientProtocolException {
        // 2.发起一个 httpget 请求
        HttpGet indexHttpGet = new HttpGet(aidUrl);
        return getHtmlByRequest(indexHttpGet);
    }

    private static String getHtmlByRequest(HttpRequestBase request) throws
IOException, ClientProtocolException {
        // User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
        // (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
        request.setHeader("User-Agent",
                "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/50.0.2661.102 Safari/537.36");

        // 3.使用 httpclient 执行，得到一个 entity。
        CloseableHttpClient indexHttpClient = HttpClients.createDefault();
        CloseableHttpResponse indexResponse = indexHttpClient.execute(request);
        String html = null;
        if (200 == indexResponse.getStatusLine().getStatusCode()) {
            HttpEntity indexEntity = indexResponse.getEntity();
            // 4.将 entity 转成字符串 (html)
            html = EntityUtils.toString(indexEntity, Charset.forName("utf-8"));
        }
        return html;
    }

    /**
     * 获取首页的文章列表信息
     *
     * @throws IOException
     * @throws ClientProtocolException
```

```
    */
   private static ArrayList<String> getIndexArticleUrlList() throws
IOException, ClientProtocolException {
      ArrayList<String> aidList = new ArrayList<String>();
      // 1.指定首页 url http://www.huxiu.com
      String indexUrl = "http://www.huxiu.com";
      // 2.发起一个 httpget 请求
      HttpGet indexHttpGet = new HttpGet(indexUrl);
      // User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
      // (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
      indexHttpGet.setHeader("User-Agent",
            "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/50.0.2661.102 Safari/537.36");

      // 3.使用 httpclient 执行，得到一个 entity。
      CloseableHttpClient indexHttpClient = HttpClients.createDefault();
      CloseableHttpResponse indexResponse =
indexHttpClient.execute(indexHttpGet);
      // 发现 1:没有数据，打印状态码
      int statusCode = indexResponse.getStatusLine().getStatusCode();
      // 发现 2:打印 statusCode 之后发现 500 异常，属于服务器异常
      // 尝试在 header 中添加一个 user-agent
      System.out.println(statusCode);

      HttpEntity indexEntity = indexResponse.getEntity();
      // 4.将 entity 转成字符串 (html)
      String html = EntityUtils.toString(indexEntity,
Charset.forName("utf-8"));

      // 5.使用 Jsoup 进行解析，得到 文章的列表 ， 获得文章 aid。
      Document indexDocument = Jsoup.parse(html);

      // 获取 date_line
      Elements dateLines = indexDocument.select("[data-last_dateline]");
      dateLine = dateLines.get(0).attr("data-last_dateline");

      // 5.1 解析出 div 的某个属性 data-id
      Elements aidElements = indexDocument.select("div[data-aid]");
      // 5.2 依次得到每个新闻的 aid
      for (Element element : aidElements) {
         String aid = element.attr("data-aid");
         aidList.add(aid);
      }
      return aidList;
```

```
    }
}
```

4.2.2 保存数据库代码

```
package cn.itcast.spider.huxiu;

import com.mchange.v2.c3p0.ComboPooledDataSource;
import org.springframework.jdbc.core.JdbcTemplate;

/**
 * JdbcTemplate 指定数据源
 *         drivermanagersource 数据源 bug
 *         c3p0,druid
 *         ComboPooledDataSource
 *
 */
public class ArticleDao extends JdbcTemplate {

    public ArticleDao() {
        // 创建 C3P0 的 datasource 1.配置 2.代码
        ComboPooledDataSource dataSource = new ComboPooledDataSource();
        // 1.url
        // 2.driver
        // 3.username&password
        dataSource.setUser("root");
        dataSource.setPassword("root");

dataSource.setJdbcUrl("jdbc:mysql://mysql:3306/spider?characterEncoding=utf
-8");
        setDataSource(dataSource);
    }

    public void save(Article article) {
        String sql = "INSERT INTO `spider`.`huxiu_article` (`id`, `title`,
`author`, `createTime`, `zan`, `pl`, `sc`, `content`, `url` )
VALUES( ?,?,?,?,?,?,?,?,?)";
        update(sql,
article.getId(),article.getTitle(),article.getAuthor(),article.getCreateTim
e(),article.getZan(),article.getPl(),article.getSc(),article.getContent(),a
rticle.getUrl());
    }
```

```
}
```

4.2.3 数据库脚本

参见资料【虎嗅爬虫/spider.sql】

第 5 章 爬虫优化与部署

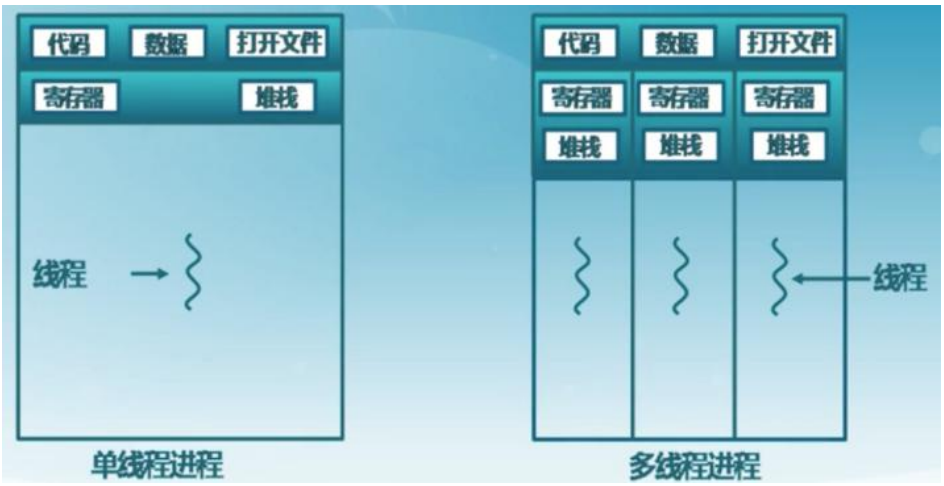
5.1 使用多线程技术

5.1.1 线程是什么？

线程是运行在进程中的一个独立实体，是 CPU 调度和分派的基本单位。
线程基本不拥有系统资源，可以与同属一个进程的其它线程共享进程所用多的全部资源。

5.1.2 多线程是什么？

一个进程中多个线程的情况，我们叫做多线程。
多个线程会共享进程所拥有的全部资源。



5.1.3 线程的开销

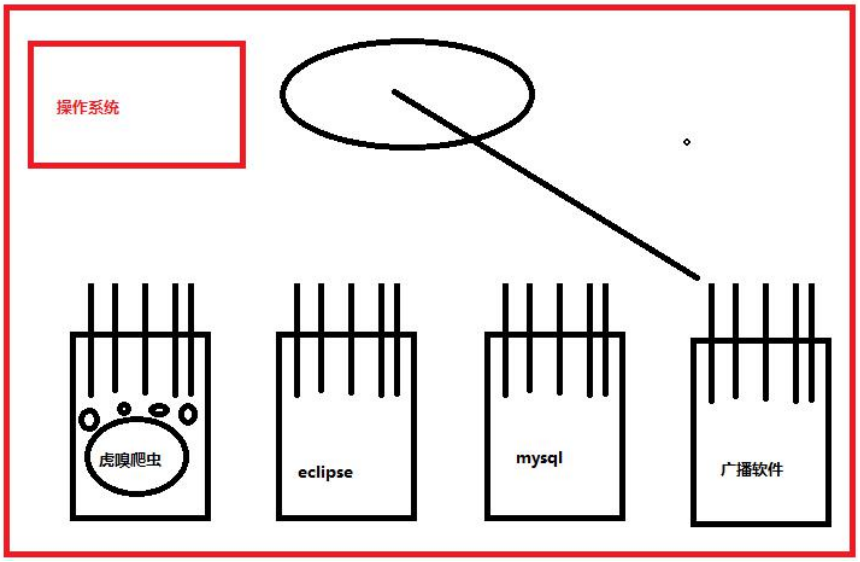
- 关于时间，创建线程使用是直接向系统申请资源的，对操作系统来说,创建一个线程的代价

是十分昂贵的，需要给它分配内存、列入调度,同时在线程切换的时候还要执行内存换页,CPU 的缓存被 清空,切换回来的时候还要重新从内存中读取信息,破坏了数据的局部性。

- 关于资源,Java 线程的线程栈所占用的内存是在 Java 堆外的，所以是不受 java 程序控制的，只受系统资源限制，默认一个线程的线程栈大小是 1M（当让这个可以通过设置-Xss 属性设置，但是要注意栈溢出问题），但是，如果每个用户请求都新建线程的话，1024 个用户光线程就占用了 1 个 G 的内存，如果系统比较大的话，一下子系统资源就不够用了，最后程序就崩溃了。

5.1.4 多线程的作用

通过多个线程并发执行，从而提高任务处理的速度。



5.1.5 创建线程几种方式

- 继承 Thread 类，并复写 run 方法，创建该类对象，调用 start 方法开启线程。
- 实现 Runnable 接口，复写 run 方法，创建 Thread 类对象，将 Runnable 子类对象传递给 Thread 类对象。调用 start 方法开启线程。
- 通过 Callable 和 Future 创建线程（了解）

```
public static void main(String[] args) throws ExecutionException {
    //Callable 的返回值就要使用 Future 对象，Callable 负责计算结果，Future 负责拿到结果
    //1、实现 Callable 接口
    Callable<Integer> callable = new Callable<Integer>() {
        public Integer call() throws Exception {
            int i=999;
            //do something
            // eg request http server and process
```

```
                return i;
            }
        };
        //2、使用 FutureTask 启动线程
        FutureTask<Integer> future = new FutureTask<Integer>(callable);
        new Thread(future).start();
        //3、获取线程的结果
        try {
            Thread.sleep(5000);// 可能做一些事情
            System.out.println(future.get());
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
}
```

5.1.6 线程池 Executors 类

- 创建固定数目线程的线程池
- 创建一个可缓存的线程池
- 定时及周期性的执行任务的线程池
  - scheduleAtFixedRate 这个方法是不管你有没有执行完，反正我每隔几秒来执行一次，以相同的频率来执行
  - scheduleWithFixedDelay 这个是等你方法执行完后，我再隔几秒来执行，也就是相对延迟后，以固定的频率去执行

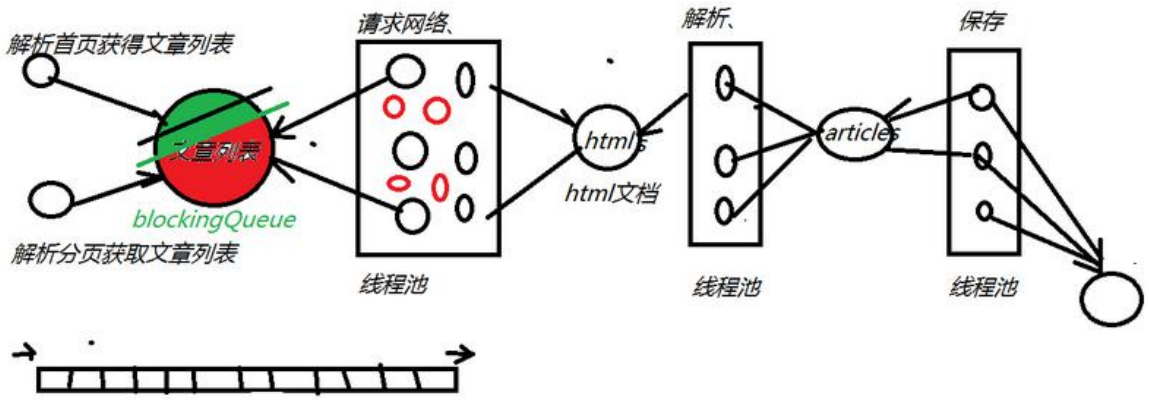
```
public static    void testFixedThreadPool() {
        //创建固定的线程池，使用 3 个线程来并发执行提交的任务。底层是个无界队列
        ExecutorService executorService = Executors.newFixedThreadPool(6);
        executorService.execute(new MyThread());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyThread());
        executorService.execute(new MyThread());
}

public static void testSingleThreadPool() {
        //创建单线程,在任务执行时，会依次执行任务。底层是个无界队列。
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        executorService.execute(new MyThread());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyRunnable());
```

```
        executorService.execute(new MyRunnable());
        executorService.execute(new MyThread());
    }

    public static void testCacheThreadPool() {
        //创建非固定数量，可缓存的线程池。当提交的任务数量起起伏伏时，会自动创建或者减少执行线程的数量。
        //当然，重用线程是线程池的基本特征。
        ExecutorService executorService = Executors.newCachedThreadPool();
        executorService.execute(new MyThread());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyRunnable());
        executorService.execute(new MyThread());
    }

    public static void testScheduledThreadPool(){
        //创建一个定时执行线程池
        ScheduledExecutorService executorService = Executors.newScheduledThreadPool(30);
        //1、配置任务的执行周期
        //scheduleAtFixedRate 固定周期执行完毕
        executorService.scheduleAtFixedRate(new MyRunnable(),0,1000,TimeUnit.MILLISECONDS);
        //scheduleWithFixedDelay 上一次执行完毕之后下一次开始执行
        executorService.scheduleWithFixedDelay(new MyRunnable(),0,1000,TimeUnit.MILLISECONDS);
    }

    public static void testSingleCacheThreadPool(){
        //创建一个单个线程执行的定时器
        ScheduledExecutorService executorService = Executors.newSingleThreadScheduledExecutor();
        //scheduleAtFixedRate 固定周期执行完毕
        executorService.scheduleAtFixedRate(new MyRunnable(),0,1000,TimeUnit.MILLISECONDS);
        //scheduleWithFixedDelay 上一次执行完毕之后下一次开始执行
        executorService.scheduleWithFixedDelay(new MyRunnable(),0,1000,TimeUnit.MILLISECONDS);
    }

    public static void testMyThreadPool(){
        //自定义连接池稍微麻烦些，不过通过创建的 ThreadPoolExecutor 线程池对象，可以获取到当前线程池的尺寸、正在执行任务的线程数、工作队列等等。
        ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(10,100,10,TimeUnit.SECONDS,new ArrayBlockingQueue<Runnable>(100));
        threadPoolExecutor.execute(new MyThread());
        threadPoolExecutor.execute(new MyRunnable());
        threadPoolExecutor.execute(new MyRunnable());
    }
```

5.1.7 使用多线程优化爬虫程序

见代码

5.2 使用队列技术



引入队列，让程序变得清爽一些。
原生队列在多线程下会有重复消费的问题，建议使用阻塞队列（详见课件）。

5.2.1 基本概念

Queue 用于模拟队列这种数据结构。

- 队列通常是指“先进先出（FIFO）”的容器。
- 队列的头部保存在队列中存放时间最长的元素，尾部保存存放时间最短的元素。
- 新元素插入到队列的尾部，取出元素会返回队列头部的元素。
- 通常，队列不允许随机访问队列中的元素。

5.2.2 使用队列

在 java5 中新增加了 java.util.Queue 接口，用以支持队列的常见操作。
该接口扩展了 java.util.Collection 接口。
Queue 使用时要尽量避免 Collection 的 add()和 remove()方法，而是要使用 offer()来加入元素，使用 poll()来获取并移出元素。
它们的优点是通过返回值可以判断成功与否，add()和 remove()方法在失败的时候会抛出异常。
如果要使用而不移出该元素，使用 element()或者 peek()方法。

	抛出异常	返回特殊值
插入	add(e)	offer(e)
移除	remove()	poll()
检查	element()	peek()

- 插入数据
  - void add(Object e):　　将指定元素插入到队列的尾部。
  - boolean offer(Object e):　　将指定的元素插入此队列的尾部。当使用容量有限的队列时，此方法通常比 add(Object e)有效。
- 移除
  - Object remove():　　获取队列头部的元素，并删除该元素。
  - Object poll():　　返回队列头部的元素，并删除该元素。如果队列为空，则返回 null。
- 检查
  - object element():　　获取队列头部的元素，但是不删除该元素。
  - Object peek():　　返回队列头部的元素，但是不删除该元素。如果队列为空，则返回 null。

注意，Queue 通常不允许插入 Null，尽管某些实现（比如 LinkedList）是允许的，但是也不建议。

5.2.3　双端队列

Queue 还有一个 Deque 接口，Deque 代表一个“双端队列”，双端队列可以同时从两端删除或添加元素，因此 Deque 可以当作栈来使用。

java 为 Deque 提供了 ArrayDeque 实现类和 LinkedList 实现类。

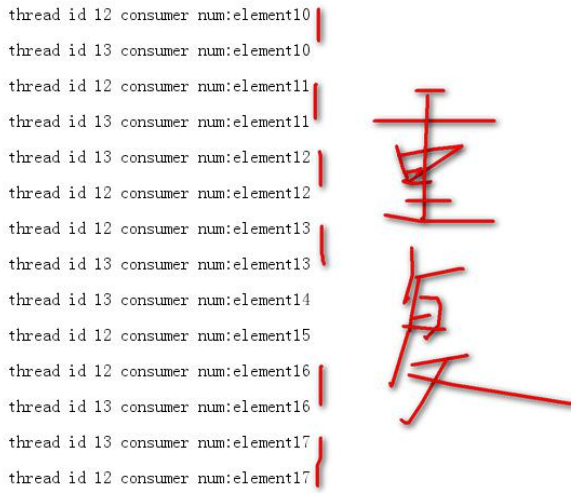
```
public static void main(String[] args) {
    ArrayDeque queue = new ArrayDeque();
    queue.offer("春");
    queue.offer("夏");
    queue.offer("秋");
    System.out.println(queue);
    System.out.println(queue.peek());
    System.out.println(queue);
    System.out.println(queue.poll());
    System.out.println(queue);
}
```

5.2.4　多线程下的原生 Queue

最主要的问题，重复消费，造成数据混乱

```
package cn.itcast.spider.huxiu.queue;
```

```
import java.util.ArrayDeque;

public class DqueueTest {
   public static void main(String[] args) {
      final ArrayDeque<String> arrayDeque = new ArrayDeque<String>();
      for (int i = 0; i < 99; i++) {
         // 添加元素
         arrayDeque.offer("element" + i);
      }
      new Thread(new Runnable() {
         public void run() {
            while(true){
               String ele = arrayDeque.poll();
               System.out.println("thread id
"+Thread.currentThread().getId()+" consumer num:"+ele);
               try {
                  Thread.sleep(100);
               } catch (InterruptedException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
               }
            }
         }
      }).start();;
      new Thread(new Runnable() {
         public void run() {
            while(true){
               String ele = arrayDeque.poll();
               System.out.println("thread id
"+Thread.currentThread().getId()+" consumer num:"+ele);
               try {
                  Thread.sleep(100);
               } catch (InterruptedException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
               }
            }
         }
      }).start();;
   }
}
```



多运行几次之后，你会看到有重复的元素被消费。

5.2.5 阻塞队列

BlockingQueue,如果 BlockQueue 是空的,从 BlockingQueue 取东西的操作将会被阻断进入等待状态,直到 BlockingQueue 进了东西才会被唤醒.

同样,如果 BlockingQueue 是满的,任何试图往里存东西的操作也会被阻断进入等待状态,直到 BlockingQueue 里有空间才会被唤醒继续操作

BlockingQueue 阻塞队列

- ArrayBlockingQueue
- LinkedBlockingQueue

常见操作

- 加入数据
  - add():把 anObject 加到 BlockingQueue 里,即如果 BlockingQueue 可以容纳,则返回 true,否则报异
  - offer():表示如果可能的话,将 anObject 加到 BlockingQueue 里,即如果 BlockingQueue 可以容纳,则返回 true,否则返回 false.
  - put():把 anObject 加到 BlockingQueue 里,如果 BlockQueue 没有空间,则调用此方法的线程被阻断直到 BlockingQueue 里面有空间再继续.
- 取出数据
  - poll():取走 BlockingQueue 里排在首位的对象,若不能立即取出,则可以等 time 参数规定的时间,取不到时返回 null
  - take():取走 BlockingQueue 里排在首位的对象,若 BlockingQueue 为空,阻断进入等待状态直到 Blocking 有新的对象被加入为止

5.2.6 阻塞队列-无重复消费

```
public static void main(String[] args) {

    final BlockingQueue queue = new ArrayBlockingQueue(3);
    //创建三个线程 生产数据 put
    for (int i = 0; i < 2; i++) {
        new Thread() {
            public void run() {
                while (true) {
                    try {
                        Thread.sleep((long) (Math.random() * 1000));
                        System.out.println(Thread.currentThread().getName() + "----------准备放
数据!");
                        queue.put(1);
                        System.out.println(Thread.currentThread().getName() + "----------已经放
了数据，" +
                                "队列目前有" + queue.size() + "个数据");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                }
            }

        }.start();
    }
    //创建一个线程消费数据 take 数据
    new Thread() {
        public void run() {
            while (true) {
                try {
                    //将此处的睡眠时间分别改为 100 和 1000，观察运行结果
                    Thread.sleep(1000);
                    System.out.println(Thread.currentThread().getName() + "准备取数据!");
                    queue.take();
                    System.out.println(Thread.currentThread().getName() + "已经取走数据，" +
                            "队列目前有" + queue.size() + "个数据");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
```

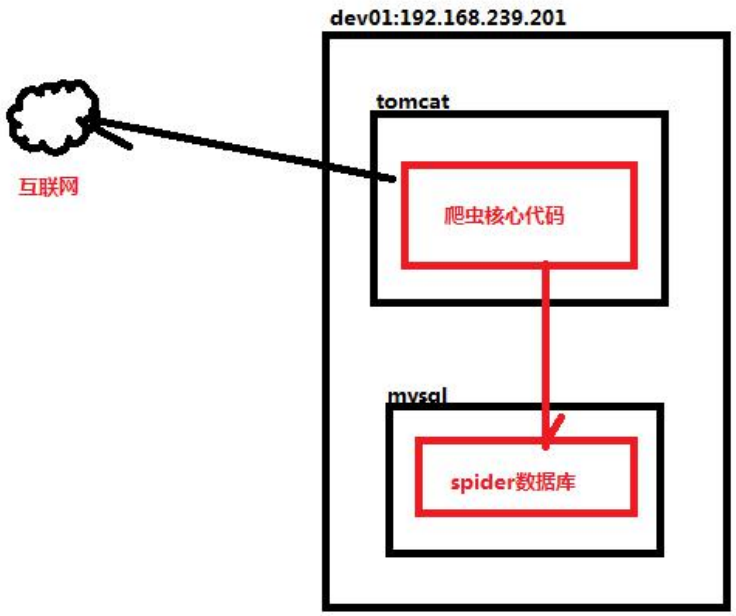
```
            }
        }.start();
    }
```

5.2.7 使用队列技术优化爬虫

见代码

5.3 网络爬虫部署

5.3.1 程序结构



5.3.2 基础环境

- 需要一台服务器
  - 在企业中是真实的硬件
  - 在学习中是虚拟机
- 在服务器上安装 linux 操作系统
  - 互联网企业基本上都是使用 linux 系统

  一般都会安装 CentOS 的版本，6.5 以上。
- 网络防火墙
  - 企业里面都需要配置防火墙

- 学习环境中直接关闭防火墙

5.3.3 配置项目运行环境

5.3.3.1 环境基本情况

Java 项目都需要安装 JDK
爬虫项目需要连接外网，需要 linux 服务器能够联网
爬虫爬取的数据需要保存到数据，需要安装数据库

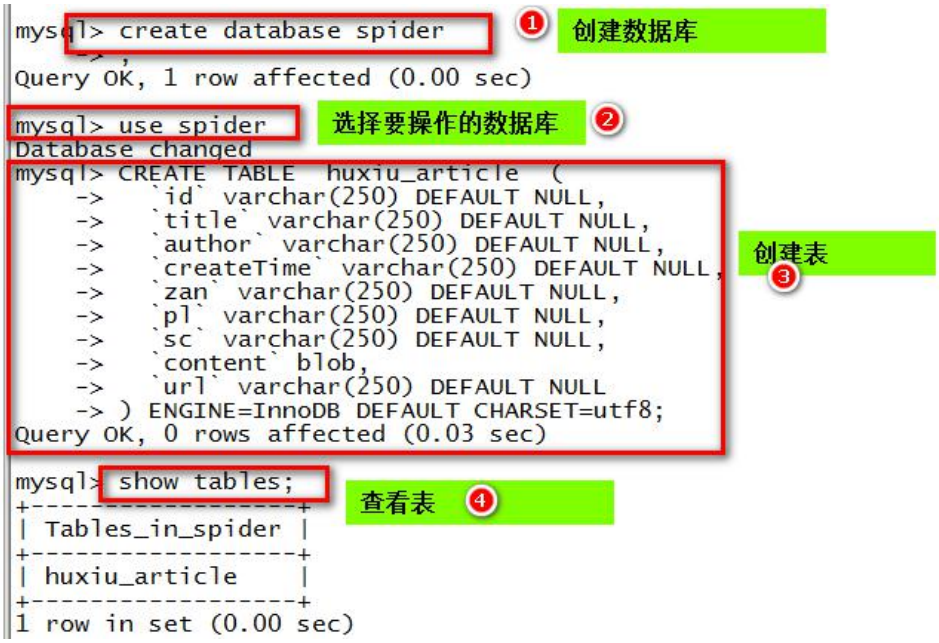
5.3.3.2 安装环境

1）安装 JDK，省略
2）配置 Linux 的网络
三种网络方式
3）安装数据库
安装命令：
yum install -y mysql-server
启动命令：
service mysqld start

5.3.4 部署项目

5.3.4.1 在 mysql 上创建数据库

```
DROP TABLE IF EXISTS `huxiu_article`;
CREATE TABLE `huxiu_article` (
`id` varchar(250) DEFAULT NULL,
`title` varchar(250) DEFAULT NULL,
`author` varchar(250) DEFAULT NULL,
`createTime` varchar(250) DEFAULT NULL,
`zan` varchar(250) DEFAULT NULL,
`pl` varchar(250) DEFAULT NULL,
`sc` varchar(250) DEFAULT NULL,
`content` blob,
`url` varchar(250) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



5.3.4.2 将项目打包

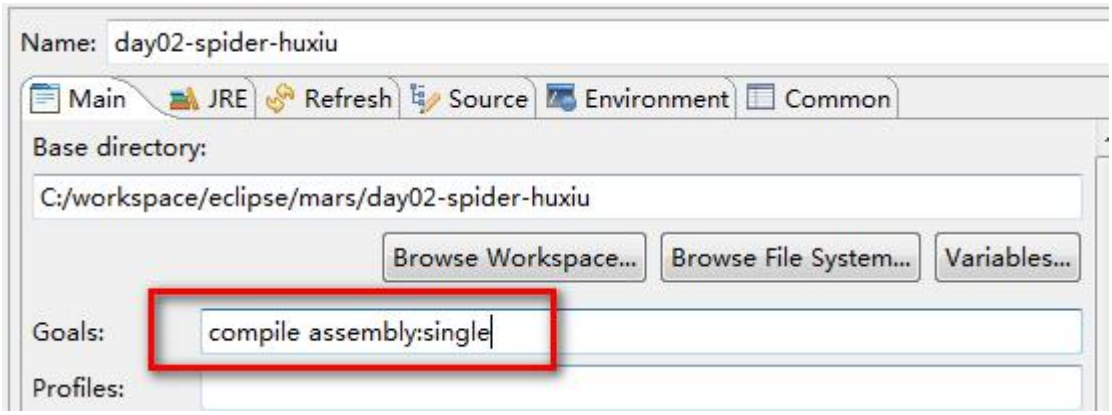
① 修改数据库链接信息

② eclipse 默认打包方式没有依赖的 jar

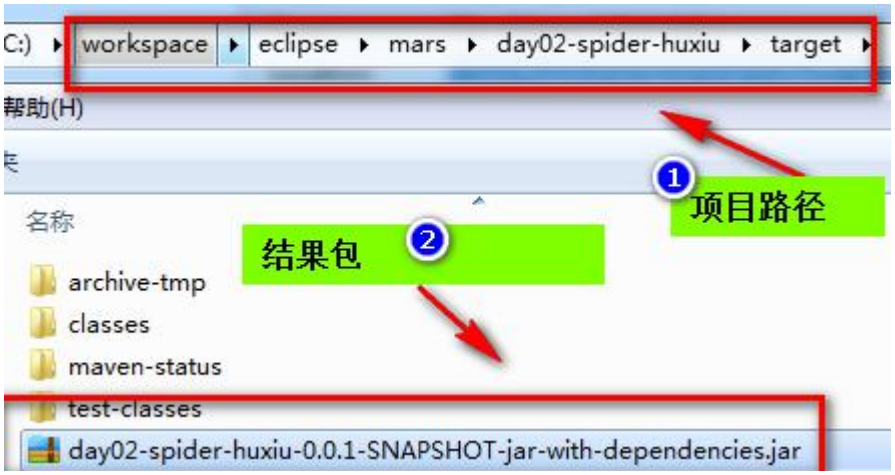
③ 使用 maven 的 assembly plugin 插件

```
<build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>huxiuSpider.HuxiuSpider</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

④ compile assembly:single



⑤ 找到打好的包



⑥ 上传到服务器

rz

5.3.4.3 运行项目

① java -jar xxx.jar 运行项目

② 解决错误 Access denied for user 'root'@'localhost' (using password: YES)

```
use mysql;
UPDATE user SET Password = PASSWORD('root') WHERE user = 'root';
flush privileges;
-
use mysql;
select Host,user,password from user;
-
grant all privileges on *.* to root@"localhost" identified by "root" with grant option;
flush privileges;
```

③ 解决错误 mysql 客户端乱码

```
show variables like '%char%';
```

```
vi /etc/my.cnf
[mysql]
default-character-set=utf8
```

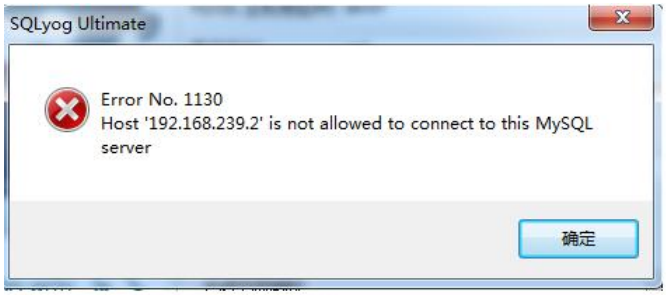
5.3.4.4 访问数据

① 不能连接

解决办法，关闭防火墙

```
service iptables stop
```

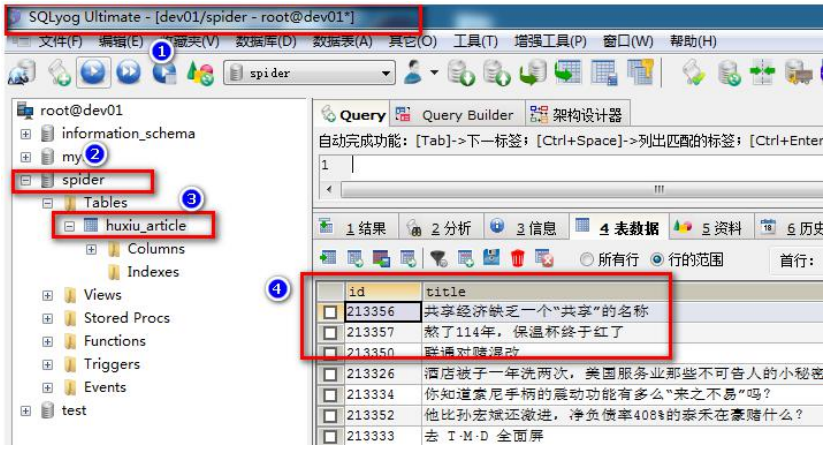
② 不允许连接



解决方法，在mysql.user表中开放权限

```
grant all privileges on *.* to root@"192.168.239.2" identified by "root" with grant option;
flush privileges;
```

③ 成功访问数据



5.3.4.5 几个常见问题

- 数据库问题
  - 创建数据库
  - 修改密码，使项目中的密码与之相同
  - 开放数据库连接
  - mysql 客户端的显示乱码问题
- 系统层面
  - 防火墙的问题
  - 直接关闭防火墙
  - 或者请运维单独配置

第 6 章 分布式爬虫开发

6.1 分布式爬虫概念

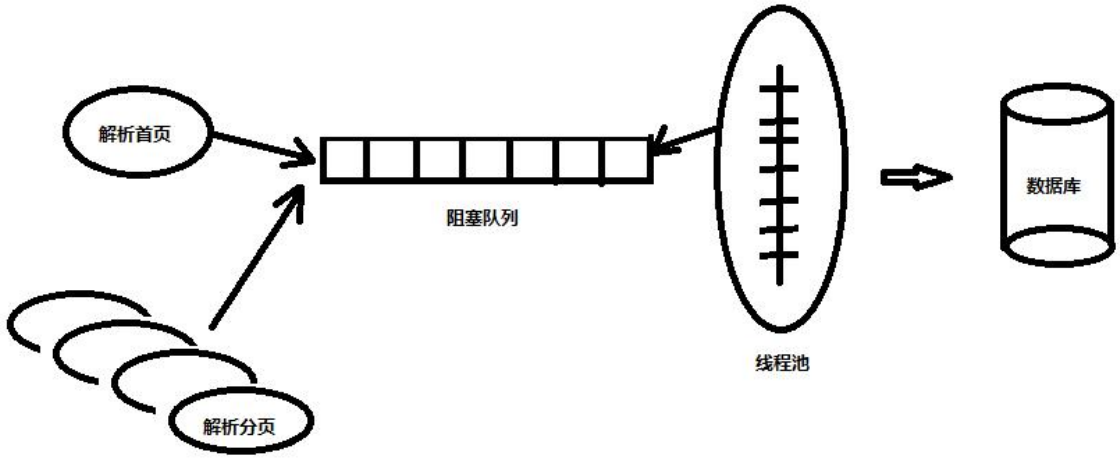
6.1.1 单机爬虫的问题

- 单机爬虫运行在一台服务器的一个 JVM 上。
- 服务器有停电的风险
- 服务器有磁盘坏掉的风险
- 服务器的网卡一般都是千兆网卡，数据量太大会有风险

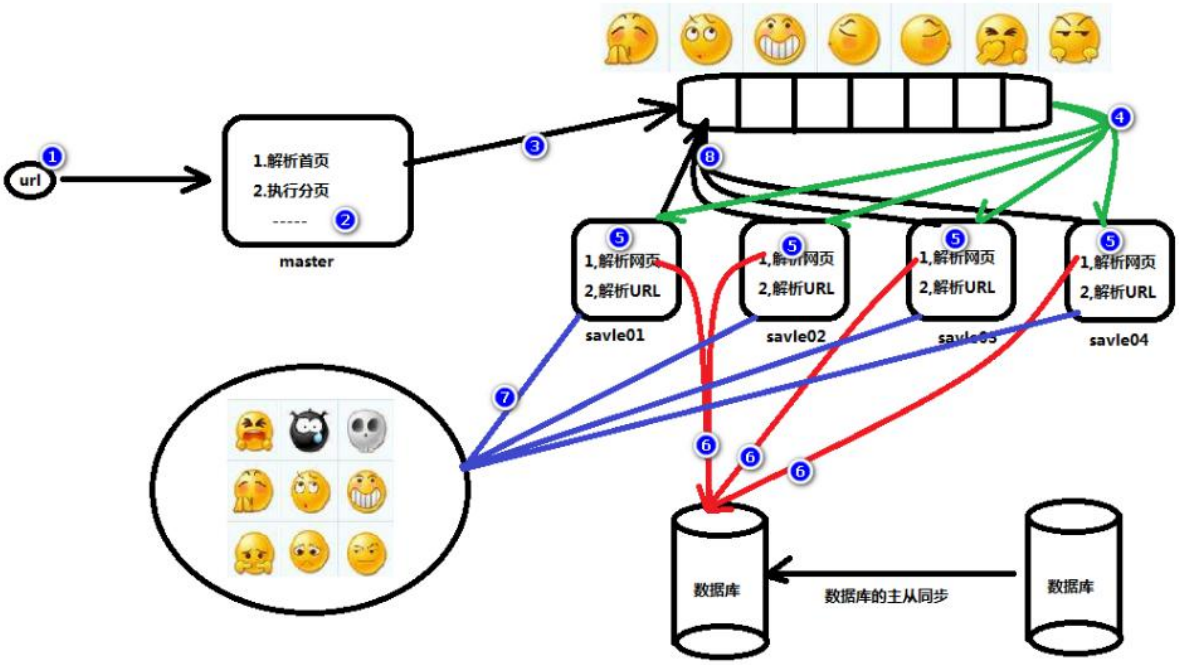
  千兆网络是全双工的,也就是上下行是并行工作的,也就是说上行下行同时能达到理论值 125MB/s
- JVM 本质上是个进程，虽说很稳定，但也有挂掉的风险
- 面对有防御策略的网站，单机爬虫战斗力不足

  参见下章节《爬虫攻防技术》

6.1.2 分布式爬虫的一般架构



上图单机版本的爬虫架构
下图分布式爬虫的一般架构



6.1.3 分布式爬虫开发技术分析

- 需要开发一个 master，分解任务
- 需要开发一个 slave 程序，并启动多个
- 需要一个公共的阻塞式队列（Redis List）
- 需要一个公共的去重的容器（Redis Set）

6.2 京东商城网站爬取需求分析

目标网站：京东
数据范围：京东上的手机信息爬取

6.3 京东商城网站爬取功能分析

6.3.1 功能分析

搜索的 API 分析

首页 API
分页 API

page 的页码是有一个算法（2n-1）
都是 Get 请求
如果爬取的是手机，需要将关键词修改为手机即可。

6.3.2 伪代码编写

- 解析首页
  - 指定首页 url，创建一个 httpclient，执行得到响应值。
  - 使用 Jsoup 解析首页的返回内容,找到商品列表的 DIV
    - 然后解析出 li 标签的 data-sku 属性
    - 拼装成商品的 url：https://item.jd.com/{sku}.html

- **分页请求**
  - 根据页码自动生产分页的 url,page 参数是不定变动的。
  - 创建一个 httpclient，执行得到响应值。
  - 解析想要的内容。

6.4 京东商城网站爬取开发

6.4.1 Master

```
package cn.itcast.spider.jd;

import java.io.IOException;
import java.nio.charset.Charset;
```

```
import org.apache.http.HttpEntity;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import redis.clients.jedis.Jedis;

/**
 * 请求首页，将首页解析出来的 product 组成 url，存放到 redis 的 list 数据结构中。优化下：存放到 redis 中，只存放 pid，节约内存资源。
 *
 * @author maoxiangyi
 *
 */
public class JDProductMaster {
   private static final Jedis jedis = new Jedis("127.0.0.1", 6379);

   public static void main(String[] args) {
      // 1.准备 url
      String indexUrl =
"https://search.jd.com/Search?keyword=%E6%89%8B%E6%9C%BA&enc=utf-8&wq=%E6%89%8B%E6%9C%BA&pvid=aa9a99bb0895488197c2a663d777a51b";
      try {
         HttpGet httpGet = new HttpGet(indexUrl);
         // 2.获取首页的信息
         String html = getHtml(httpGet);
         // 3.解析首页 此处不需要返回值，直接在方法中调用 redis 的 jedis 的客户端
         parseHtml(html);
      } catch (Exception e) {
         System.out.println("首页访问失败！" + indexUrl);
      }
      // 4、做分页请求
      int page = 1;
      for (int num = 2; num <= 100; num++) {
         page = (2 * num) - 1;
```

```
            String
pagingUrl="https://search.jd.com/Search?keyword=%E6%89%8B%E6%9C%BA&enc=utf-
8&qrst=1&rt=1&stop=1&vt=2&wq=%E6%89%8B%E6%9C%BA&cid2=653&cid3=655&click=0&
page="+page;
            HttpGet httpGet = new HttpGet(pagingUrl);
            try {
                String pagingHtml = getHtml(httpGet);
                parseHtml(pagingHtml);
            } catch (Exception e) {
                System.out.println("请求分页失败，分页编号是:"+page);
            }
            try {
                Thread.sleep(3*1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }

    /**
     * 解析搜索页的首页时，需要获取到pid
     *
     * @param html
     */
    public static void parseHtml(String html) {
        if (html != null) {
            Document doc = Jsoup.parse(html);
            Elements eles = doc.select("[data-pid]");
            for (Element element : eles) {
                jedis.lpush("itcast:spider:jd:pids", element.attr("data-pid"));
            }
        }

    }

    /**
     * 专门用来访问 httpget 请求的方法
     *
     * @param httpGet
     * @return
     * @throws IOException
     * @throws ClientProtocolException
     */
```

```
   public static String getHtml(HttpGet httpGet) throws IOException,
ClientProtocolException {
      String html = null;
      CloseableHttpClient hc = HttpClients.createDefault();
      CloseableHttpResponse res = hc.execute(httpGet);
      if (res.getStatusLine().getStatusCode() == 200) {
         HttpEntity entity = res.getEntity();
         html = EntityUtils.toString(entity, Charset.forName("utf-8"));
      }
      return html;
   }
}
```

6.4.2 Slave

```
package cn.itcast.spider.jd;

import java.io.IOException;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.apache.http.HttpEntity;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import com.google.gson.Gson;

import redis.clients.jedis.Jedis;

/**
```

```
 * 从 redis 中获取单个商品的 pid，组成一个 url，进行网络请求。 得到一个 html 文档，将 html 文档解析成一个 product 对象。 然后保存到数据库。
 *
 * @author maoxiangyi
 *
 */
public class JDProductSlave {

   public static void main(String[] args) {
      //1.创建一个线程池
      ExecutorService threadPool = Executors.newFixedThreadPool(10);
      //2.提交多个线程任务到线程池中
      for (int i = 0; i < 1; i++) {
         threadPool.execute(new Runnable() {
            public void run() {
               Jedis jedis = new Jedis("127.0.0.1",6379);
               //直获取数据，解析
               while(true){
                  //4.从 redis 中获取一个 pid
                  //扩展:要设置去重的话，需要和第 10 步配合使用，在这里加个 if 语句。
                  String pid = jedis.rpop("itcast:spider:jd:pids");
                  //5.组装成个 url
                  String url = "https://item.jd.com/"+pid+".html";
                  //6.发起一个 http 请求
                  HttpGet httpGet = new HttpGet(url);
                  try {
                     //7.得到一个返回值
                     String html = getHtml(httpGet);
                     //8.解析
                     Product product = parseDetail(html);
                     product.setPid(pid);
                     product.setUrl(url);
                     //8.1----解析价格
                     String priceUrl ="https://p.3.cn/prices/mgets?callback=jQuery2695111&type=1&area=1&pdtk=&pduid=15048650832308459287 7&pdpin=&pin=null&pdbp=0&ext=11000000&source=item-pc&skuIds=J_";
                     HttpGet priceHttpGet = new HttpGet(priceUrl+pid);
                     String priceData = getHtml(priceHttpGet);
                     int startIndex = priceData.indexOf("[{");
                     int endIndex = priceData.indexOf("]);");
                     priceData= priceData.substring(startIndex+1, endIndex);
```

```
                    //使用 Gson 解析 json 传
                    Gson gson = new Gson();
                    HashMap map = gson.fromJson(priceData, HashMap.class);
                    product.setPrice((String) map.get("p"));

                    //9.保存到数据库
                    //---------------保存数据-----------
                    // save2db
                    //--------将商品对象，存放到 redis 中。
                    //--------------

                    //10.将爬取过的 url 存到到 redis 的 set 容器中



                } catch (Exception e) {
                    System.out.println("slave 访问商品详情页失败"+pid);
                    System.out.println(e);
                }

                try {
                    Thread.sleep(10*1000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }


    });
  }
}

/**
 * 专门用来访问 httpget 请求的方法
 *
 * @param httpGet
 * @return
 * @throws IOException
 * @throws ClientProtocolException
 */
public static String getHtml(HttpGet httpGet) throws IOException,
ClientProtocolException {
```

```
        String html = null;
        CloseableHttpClient hc = HttpClients.createDefault();
        CloseableHttpResponse res = hc.execute(httpGet);
        if (res.getStatusLine().getStatusCode() == 200) {
            HttpEntity entity = res.getEntity();
            html = EntityUtils.toString(entity, Charset.forName("utf-8"));
        }
        return html;
    }

    /**
     * 解析商品详情页，得到一个商品对象
     * @param html
     * @return
     */
    public static Product parseDetail(String html) {
        Product product = new Product();
        if(html!=null){
            Document doc = Jsoup.parse(html);
//          //标题
            String title =doc.select("div.sku-name").get(0).ownText();
            product.setTitle(title);

            //获取价格 价格是一个动态的，需要单独的接口进行访问
            ArrayList<String> imgList = new  ArrayList<String>();
            //获取轮播图片
            Elements eles = doc.select("script[charset=gbk]");
            for (Element element : eles) {
                String text = element.toString();
                int imgIndex = text.indexOf("imageList");
                int endIndex = text.indexOf("cat");
                endIndex = endIndex-6;
                String imgs = text.substring(imgIndex+13,endIndex);
                //imageList:
["jfs/t5815/57/1307544515/161558/a4577c0f/592536afN51878437.jpg"
//              ,"jfs/t5698/177/1314573205/100804/5310986/592536b4N68b02458.jpg
"
//              ,"jfs/t5917/88/146435037/125441/ef9a8d3c/592536b8Nee463a6a.jpg"
,"jfs/t5911/105/135621496/82361/3d8ff7c1/592536bdNeea3b55a.jpg","jfs/t5677/
305/1297228629/116225/22f696f8/592536c1Nf81817ce.jpg","jfs/t5878/98/1320837
698/78028/2b5c4c19/592536c5Ne25cee5c.jpg"
//              ,"jfs/t5845/75/1299831685/118141/60f2b863/592536c8N5e724868.jpg
"],
```

```
            String[] imgArr = imgs.split("\",\"");
            for (String img : imgArr) {
               imgList.add(img.replace("\"],", ""));
            }
         }
         product.setImgUrls(imgList);
      }
      return product;
   }

}
```

6.4.3 Product

```
package cn.itcast.spider.jd;

import java.util.ArrayList;

/**
 * 想要什么数据，
 * @author maoxiangyi
 *
 */
public class Product {

   private String pid;//商品编号
   private String title;//标题
   private String price;//价格
   private String url;//商品
   private ArrayList<String> imgUrls;//图片
   public String getPid() {
      return pid;
   }
   public void setPid(String pid) {
      this.pid = pid;
   }
   public String getTitle() {
      return title;
   }
   public void setTitle(String title) {
      this.title = title;
   }
```

```
    public String getPrice() {
        return price;
    }
    public void setPrice(String price) {
        this.price = price;
    }
    public String getUrl() {
        return url;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public ArrayList<String> getImgUrls() {
        return imgUrls;
    }
    public void setImgUrls(ArrayList<String> imgUrls) {
        this.imgUrls = imgUrls;
    }
    @Override
    public String toString() {
        return "Product [pid=" + pid + ", title=" + title + ", price=" + price
+ ", url=" + url + ", imgUrls=" + imgUrls
                + "]";
    }

}
```

6.5 分布式爬虫部署实战

学生动作实践

1） 部署 redis
2） 部署 mysql
3） 一台服务器部署 master
4） 部署两个 slave

第 7 章　爬虫攻防技术

7.1 如何识别爬虫

7.1.1　如何发现爬虫

- 单一 IP 非常规的访问频次
- 单一 IP 非常规的数据流量
- 大量重复简单点的网站浏览行为
- 只下载网页，没有后续的 JS\CSS 请求
- 设置陷阱，使用 hidden 属性对用户隐藏标签但爬虫可见。
- 判断请求头
- 判断 cookie

7.1.2　如何避免被发现

- 多主机策略，分布式爬取。
- 调整爬取速度
- 通过变换 IP 地址或者使用代理服务器来演示
- 频繁修改自己的 User-Agent
- Header 中的 Cache-Control 修改为 no-cache
- 当返回状态码是 403（服务器资源禁止访问），改变 Header 和 IP。不断改变。

7.1.3　使用代理 IP 绕过验证码

参数名称	必填项	默认值	示 例	说明
ddbh	是	无	ddbh=12345	此参数是唯一的ID，标识用户，不可缺失，请妥善保管，以免他人盗用。
old	否	空	old=1	只要old不为空，是过滤重复提取的，但一般只过滤24小范围内。如要严格过滤请联系我
noinfo	否	空	noinfo=true	当noinfo不为空时。不显示其它无关信息，只保留显示IP和端口信息，方便解析。
dk	否	空	dk=80或dk=80,3128	筛选特定的端口，如dk=8080 表示只提取端口8080的IP
dq	否	空	dq=北京	筛选特定地区的IP，如地区是中文的请用gb2312编码一下。
killdk	否	空	killdk=8089或kildk=80,3128	去掉不需要的端口
https	否	空	https=1	只提https代理Ip
sl	否	1	sl=10	每次提取的IP数量。

完整示例：http://www.httpdaili.com/api.asp?ddbh=123&noinfo=true&old=1&sl=10

- 从网站上爬取代理 IP 地址（一般不可用）
- 从淘宝上购买便宜的 IP 地址（2000 条 2 块）
- 购买正规的代理服务（昂贵）
- 使用阿里云试试

7.2 如何使用代理 IP 绕过网站监测

① 购买代理 IP
② 每次请求使用一个代理 IP
③ 配置代理

```
HttpHost proxy = new HttpHost(ip, port);
ConnectionConfig connectionConfig =
ConnectionConfig.custom().setBufferSize(4128).build();
DefaultProxyRoutePlanner routePlanner = new DefaultProxyRoutePlanner(proxy);
CloseableHttpClient hc =
HttpClients.custom().setDefaultConnectionConfig(connectionConfig)
      .setRoutePlanner(routePlanner).build();
```

④ 发起请求

```
package doVote4FenDou;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.apache.http.Header;
import org.apache.http.HttpHost;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.config.ConnectionConfig;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.conn.DefaultProxyRoutePlanner;
```

```
import org.apache.http.util.EntityUtils;

public class DoVote {
   public static void main(String[] args) throws Exception {
       multiThread();
   }

   /**
    * 使用多线程进行爬取
    **/
   private static void multiThread() throws FileNotFoundException,
IOException, InterruptedException {
      //1.创建阻塞队列用来存放代理 IP 地址
      final ArrayBlockingQueue<String> arrayBlockingQueue = new
ArrayBlockingQueue<String>(10000);
      //2.读取文件，该文件中保存着代理 ip 和端口号。如 103.29.186.189:3128
      String ips =
FileUtil.readFileToString("C:/workspace/eclipse/mars/doVote4FenDou/src/main
/resources/ips.txt");
      String[] ipkv = ips.split(" ");
      System.out.println("total ips:"+ipkv.length);
      for (String ip : ipkv) {
         System.out.println(ip);
         arrayBlockingQueue.put(ip);
      }
      int size = 10;
      //3.开启多线程进行投票
      ExecutorService pool = Executors.newFixedThreadPool(size);
      for (int i = 0; i < size; i++) {
         pool.submit(new Thread(new Runnable() {
            public void run() {
               while (true) {
                  try {
                     String ipkv = arrayBlockingQueue.take();
                     System.out.println(ipkv);
                     String[] kv = ipkv.split(":");
                     dovote(kv[0], Integer.parseInt(kv[1]));
                  } catch (Exception e) {
                     System.out.println("请求失败!" + e);
                  }
               }
            }
         }));
      }
```

```
    }

    private static void dovote(String ip, int port) throws IOException,
ClientProtocolException {
        //1.投票的请求接口
        String url = "http://fendou.itcast.cn/article/updatevote";
        //2.准备请求头的信息
        Map<String, String> headers = getHeader();
        //3.创建 POST 请求对象
        HttpPost httpPost = new HttpPost(url);
        //4.准备请求头的信息
        for (Map.Entry<String, String> header : headers.entrySet()) {
            httpPost.addHeader(header.getKey(), header.getValue());
        }
        //5.创建代理 HTTP 请求
        HttpHost proxy = new HttpHost(ip, port);
        ConnectionConfig connectionConfig =
ConnectionConfig.custom().setBufferSize(4128).build();
        DefaultProxyRoutePlanner routePlanner = new
DefaultProxyRoutePlanner(proxy);
        CloseableHttpClient hc =
HttpClients.custom().setDefaultConnectionConfig(connectionConfig)
                .setRoutePlanner(routePlanner).build();
        //6.使用代理 HttpClient 发起投票请求
        CloseableHttpResponse res = hc.execute(httpPost);
        //7.打印 http 请求状态码
        System.out.println("statusCode:" +
res.getStatusLine().getStatusCode());
        for (Header header : res.getAllHeaders()) {
            //8.打印所有的 response header 信息,发现有 set-cookie 的信息就成功了。
            System.out.println(header);
        }
        //9.打印 html 信息 如果返回为空字符串 就是投票成功。有返回值的基本就是
失败了。
        String html = EntityUtils.toString(res.getEntity(),
Charset.forName("utf-8"));
        System.out.println("返回值:" + html);
    }

    private static String getPS(CloseableHttpClient hc) throws IOException,
ClientProtocolException {
        HttpGet httpGet = new

HttpGet("http://fendou.itcast.cn/article/look/aid/193?qq-pf-to=pcqq.c2c");
```

```
        CloseableHttpResponse res1 = hc.execute(httpGet);
        String ps = res1.getHeaders("set-cookie")[0].getValue();
        return ps;
    }

    private static Map<String, String> getHeader() {
        HashMap<String, String> headers = new HashMap<String, String>();
        headers.put("Referer",
"http://fendou.itcast.cn/article/look/aid/193?qq-pf-to=pcqq.c2c");
        headers.put("host", "fendou.itcast.cn");
        headers.put("Origin", "http://fendou.itcast.cn");
        headers.put("User-Agent",
                "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/50.0.2661.102 Safari/537.36");
        headers.put("X-Requested-With", "XMLHttpRequest");
        headers.put("cookie", "PHPSESSID=8bj2oolgfq23idj1n3qofmm893;
UM_distinctid=15e0d154d8a617-095bb15b8b50fd-414a0229-100200-15e0d154d8b4b3
");

        headers.put("Accept", "*/*");
        headers.put("Accept-Encoding", "gzip, deflate");
        headers.put("Accept-Language", "zh-CN,zh;q=0.8");
        headers.put("Connection", "keep-alive");

        return headers;
    }

}
```

7.3 如何使用 Google chrome 进行代码调试

课堂演示

7.4 如何绕过网站验证码

7.4.1 一些经验

- 1、绕过验证码。跳过验证码直接访问需要的页面内容。
- 2、请求头中自带验证码。有些网站的验证码会在前台 js 校验。服务器生成的验证码会在请求头中。可以获取请求头，并把验证码解析出来。

- 3、session 不刷新。国内很多网站存在一个通病：验证码验证成功后，直接获取请求资源。（忘记了刷新 cookie 对应的验证码）可以预先设定一个 cookie 和验证码。利用这个漏洞访问网站。对于多线程无法控制以及有些网站验证码定期不访问失效问题。可以添加一个定时访问程序来解决
- 4、利用第三方插件。对于有些网站验证码比较简单。只含阿拉伯数字和英文字母。可以用第三方的插件来识别。例如：tess4j、tesseract
- 5、有些网站的验证码是从库中随机取出一个来的。对于这类静态的验证码。可以自己建立一个验证码静态库。自己建立好图片和验证码答案的链接。采用 map 的映射方法就可以进行识别参见。
- 6、自己定制写验证码的程序

7.4.2 关于图片识别

验证码本质上是图片，识别验证码需要图片识别技术。
图片识别技术的专业名称叫做光学字符识别(OCR,Optical Character Recognition)
OCR 是指对文本资料进行扫描，然后对图像文件进行分析处理，获取文字及版面信息的过程。OCR 技术非常专业，一般多是印刷、打印行业的从业人员使用，可以快速的将纸质资料转换为电子资料。关于中文 OCR，目前国内水平较高的有清华文通、汉王、尚书，其产品各有千秋，价格不菲。国外 OCR 发展较早，像一些大公司，如 IBM、微软、HP 等，即使没有推出单独的 OCR 产品，但是他们的研发团队早已掌握核心技术，将 OCR 功能植入了自身的软件系统。

7.4.3 关于开源类库 Tesseract

Tesseract-OCR 支持中文识别，并且开源和提供全套的训练工具，是快速低成本开发的首选。
Tess4J 则是 Tesseract 在 Java PC 上的应用

Tesseract 的 OCR 引擎最先由 HP 实验室于 1985 年开始研发，至 1995 年时已经成为 OCR 业内最准确的三款识别引擎之一。然而，HP 不久便决定放弃 OCR 业务，Tesseract 也从此尘封。
数年以后，HP 意识到，与其将 Tesseract 束之高阁，不如贡献给开源软件业，让其重焕新生－－2005 年，Tesseract 由美国内华达州信息技术研究所获得，并求诸于 Google 对 Tesseract 进行改进、消除 Bug、优化工作。
Tesseract 目前已作为开源项目发布在 Google Project，其项目主页在这里查看。

7.4.4 使用机器视觉技术识别验证码

7.4.4.1 导入 maven 依赖

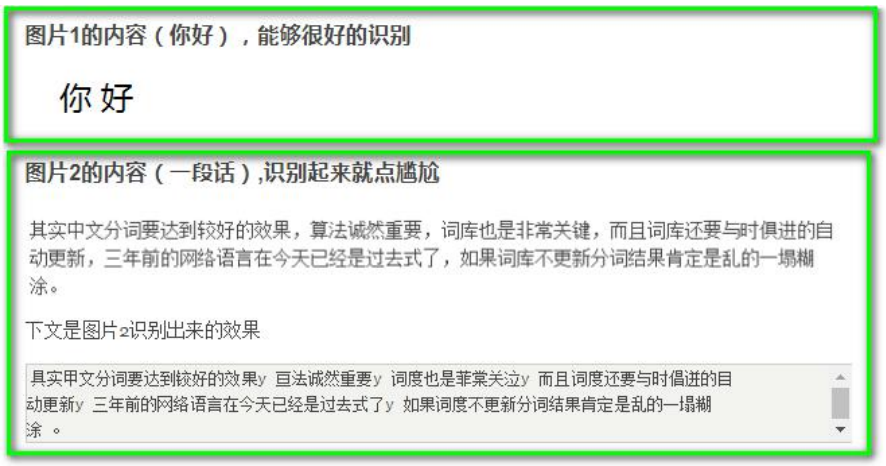
```
<!-- https://mvnrepository.com/artifact/net.sourceforge.tess4j/tess4j -->
        <dependency>
```

```
            <groupId>net.sourceforge.tess4j</groupId>
            <artifactId>tess4j</artifactId>
            <version>3.4.0</version>
        </dependency>
```

7.4.4.2 代码开发

```
File imageFile = new File("input dir/shuzi.png");
        Tesseract tessreact = new Tesseract();
        //需要指定训练集  训练集到  https://github.com/tesseract-ocr/tessdata  下载。
        tessreact.setDatapath("E:\\itcast\\env\\tess4j\\tessdata");
        //注意   默认是英文识别，如果做中文识别，需要单独设置。
        tessreact.setLanguage("chi_sim");
        try {
            String result = tessreact.doOCR(imageFile);
            System.out.println(result);
        } catch (TesseractException e) {
            System.err.println(e.getMessage());
        }
```

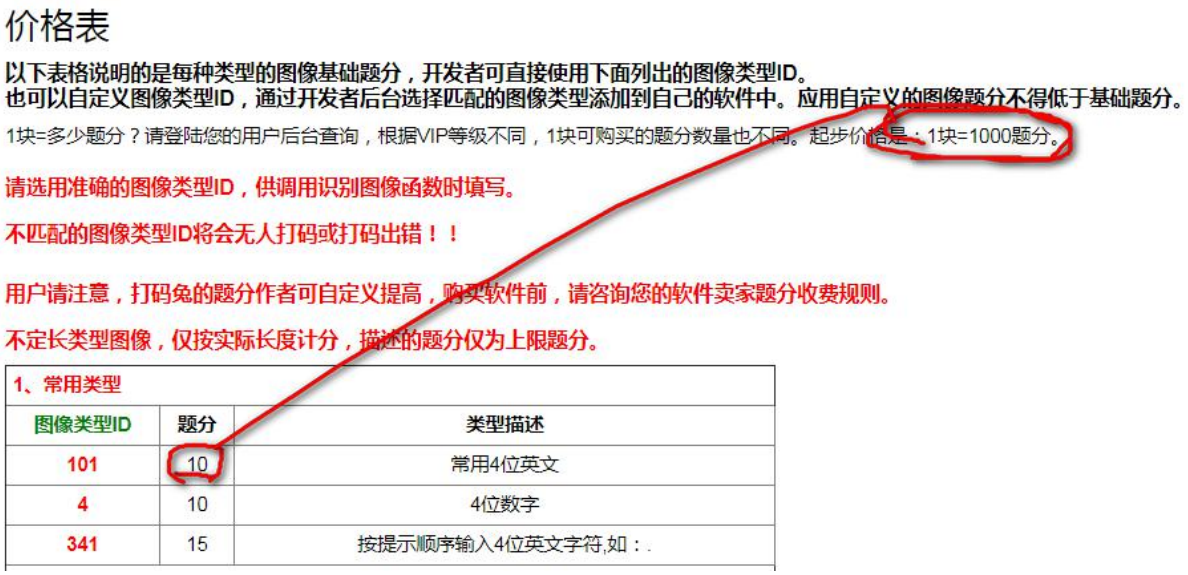
7.4.4.3 一些效果



7.4.5 使用第三方服务识别验证码

最好的方式是买买买。

http://www.dama2.com/



价格表

以下表格说明的是每种类型的图像基础题分，开发者可直接使用下面列出的图像类型ID。
也可以自定义图像类型ID，通过开发者后台选择匹配的图像类型添加到自己的软件中。应用自定义的图像题分不得低于基础题分。

1块=多少题分？请登陆您的用户后台查询，根据VIP等级不同，1块可购买的题分数量也不同。起步价格是：1块=1000题分。

请选用准确的图像类型ID，供调用识别图像函数时填写。

不匹配的图像类型ID将会无人打码或打码出错！！

用户请注意，打码兔的题分作者可自定义提高，购买软件前，请咨询您的软件卖家题分收费规则。

不定长类型图像，仅按实际长度计分，描述的题分仅为上限题分。

1、常用类型

图像类型ID	题分	类型描述
101	10	常用4位英文
4	10	4位数字
341	15	按提示顺序输入4位英文字符,如：.

http://wiki.dama2.com/index.php?n=ApiDoc.Pricedesc

第 8 章　扩展及参考

8.1 自动投票案例分析

了解级别、参见代码

8.2 微信 Web 爬虫案例分析

了解级别、参见代码

8.3 扩展文档

- 利用爬虫技术能做到哪些很酷很有趣很有用的事情？
  https://www.zhihu.com/question/27621722
- 各种语言写网络爬虫有什么优点缺点？
  https://www.zhihu.com/question/26790346
- 通俗的讲，网络爬虫到底是什么？
  https://www.zhihu.com/question/24098641
- 有免费的网络爬虫软件使用吗？
  https://www.zhihu.com/question/21877755
- GitHub 上有哪些优秀的 Java 爬虫项目？

https://www.zhihu.com/question/31427895