**网络爬虫**

**第一天**

# 第一章：课程计划

1. 入门程序
2. 网络爬虫介绍
3. HttpClient抓取数据
4. Jsoup解析数据
5. 爬虫案例

学习目标：

1. 能够完成爬虫入门程序（第二章：2.1）
2. 能够说出爬虫解决的问题（第三章）
3. 能够使用 HttpClient的Get请求（第四章：4.1、4.2）
4. 能够使用HttpClient的Post请求（第四章：4.3、4.4）
5. 能够设置HttpClient的连接池（第四章：4.5）
6. 能够使用Jsoup解析字符串、解析文件（第五章：5.2.1、5.2.2、5.2.3）
7. 能够使用dom方式查询解析Html（第五章：5.2.4）
8. 能够使用选择器查询解析Html（第五章：5.2.5、5.2.6、5.2.7）
9. 能够说出SKU和SPU的区别（第六章：6.1.1）
10. 能够实现当天案例（第六章）

# 第二章：网络爬虫

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

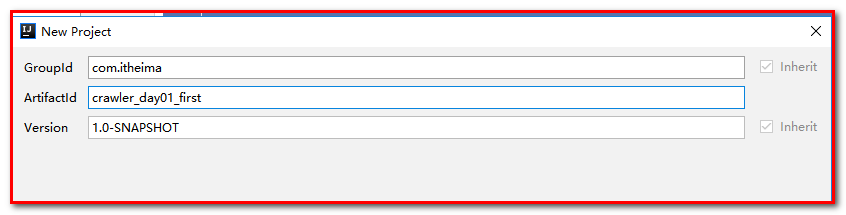
## 爬虫入门程序

### 环境准备

* JDK1.8
* IntelliJ IDEA
* IDEA自带的Maven

### 环境准备

创建Maven工程crawler\_day01\_first并给pom.xml加入依赖



导入maven的依赖

<**packaging**>jar</**packaging**>  
<**dependencies**>  
 *<!-- HttpClient -->* <**dependency**>  
 <**groupId**>org.apache.httpcomponents</**groupId**>  
 <**artifactId**>httpclient</**artifactId**>  
 <**version**>4.5.3</**version**>  
 </**dependency**>  
  
 *<!-- 日志 -->* <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-log4j12</**artifactId**>  
 <**version**>1.7.25</**version**>  
 </**dependency**>  
</**dependencies**>

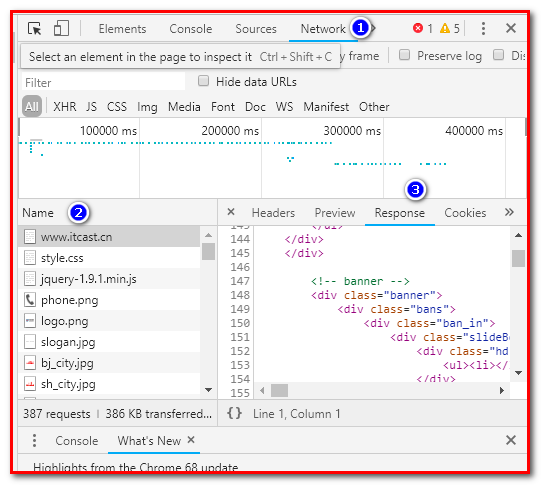
### 加入log4j.properties

**log4j.rootLogger**=**DEBUG,A1  
log4j.logger.cn.itcast** = **DEBUG  
  
log4j.appender.A1**=**org.apache.log4j.ConsoleAppender  
log4j.appender.A1.layout**=**org.apache.log4j.PatternLayout  
log4j.appender.A1.layout.ConversionPattern**=**%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n**

### 编写代码

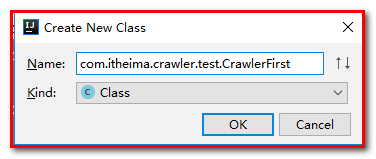
使用浏览器访问www.itcast.cn，可以通过浏览器抓取响应的数据。





创建类

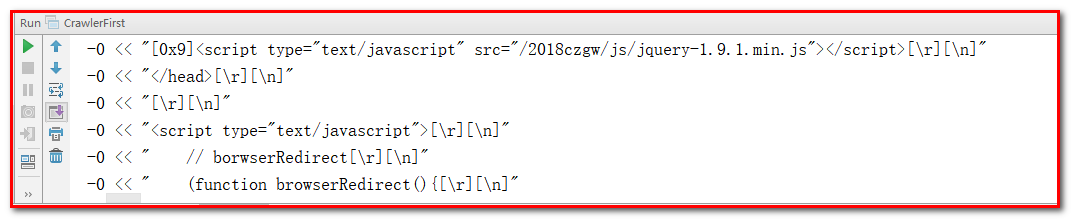
com.itheima.crawler.test.CrawlerFirst.java



编写最简单的爬虫，抓取传智播客首页：<http://www.itcast.cn/>

**public class** CrawlerFirst {  
 **public static void** main(String[] args) **throws** Exception {  
 *//1. 打开浏览器,创建HttpClient对象* CloseableHttpClient httpClient = HttpClients.*createDefault*();  
  
 *//2. 输入网址,发起get请求创建HttpGet对象* HttpGet httpGet = **new** HttpGet(**"http://www.itcast.cn"**);  
  
 *//3.按回车，发起请求，返回响应，使用HttpClient对象发起请求* CloseableHttpResponse response = httpClient.execute(httpGet);  
  
 *//4. 解析响应，获取数据  
 //判断状态码是否是200* **if** (response.getStatusLine().getStatusCode() == 200) {  
 HttpEntity httpEntity = response.getEntity();  
 String content = EntityUtils.*toString*(httpEntity, **"utf8"**);  
  
 System.***out***.println(content);  
 }  
 }  
  
}

测试结果：可以获取到页面数据



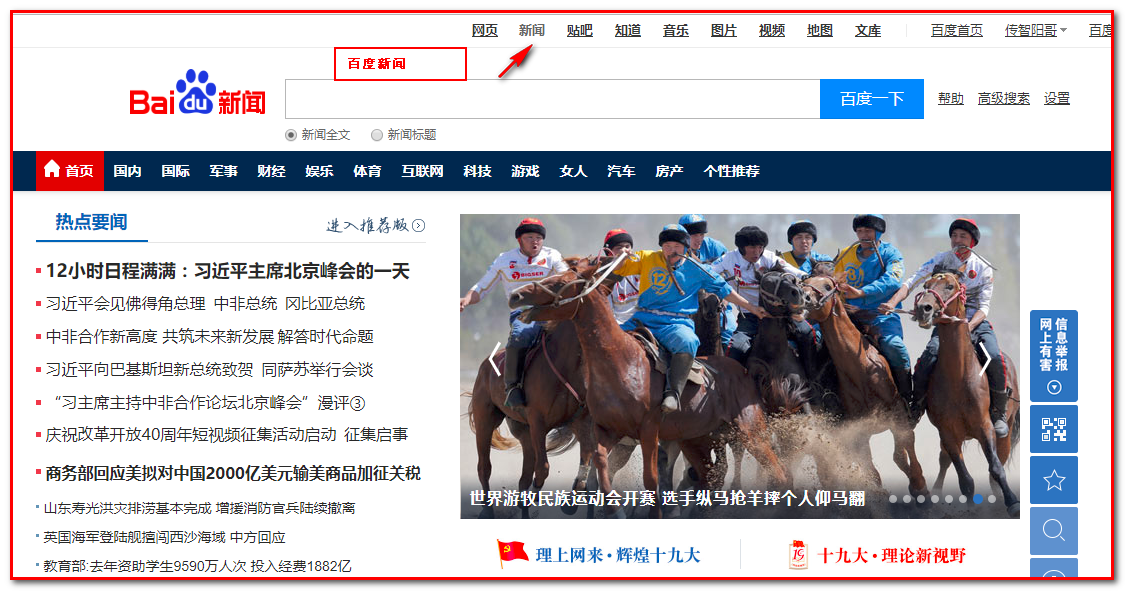
# 第三章：网络爬虫

## 网络爬虫介绍

在大数据时代，信息的采集是一项重要的工作，而互联网中的数据是海量的，如果单纯靠人力进行信息采集，不仅低效繁琐，搜集的成本也会提高。如何自动高效地获取互联网中我们感兴趣的信息并为我们所用是一个重要的问题，而爬虫技术就是为了解决这些问题而生的。

网络爬虫（Web crawler）也叫做网络机器人，可以代替人们自动地在互联网中进行数据信息的采集与整理。它是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本，可以自动采集所有其能够访问到的页面内容，以获取相关数据。

从功能上来讲，爬虫一般分为数据采集，处理，储存三个部分。爬虫从一个或若干初始网页的URL开始，获得初始网页上的URL，在抓取网页的过程中，不断从当前页面上抽取新的URL放入队列,直到满足系统的一定停止条件。



## 为什么学网络爬虫

我们初步认识了网络爬虫，但是为什么要学习网络爬虫呢？只有清晰地知道我们的学习目的，才能够更好地学习这一项知识。在此，总结了4种常见的学习爬虫的原因：

* + - 1. 可以实现搜索引擎

我们学会了爬虫编写之后，就可以利用爬虫自动地采集互联网中的信息，采集回来后进行相应的存储或处理，在需要检索某些信息的时候，只需在采集回来的信息中进行检索，即实现了私人的搜索引擎。

* + - 1. 大数据时代，可以让我们获取更多的数据源。

在进行大数据分析或者进行数据挖掘的时候，需要有数据源进行分析。我们可以从某些提供数据统计的网站获得，也可以从某些文献或内部资料中获得，但是这些获得数据的方式，有时很难满足我们对数据的需求，而手动从互联网中去寻找这些数据，则耗费的精力过大。此时就可以利用爬虫技术，自动地从互联网中获取我们感兴趣的数据内容，并将这些数据内容爬取回来，作为我们的数据源，再进行更深层次的数据分析，并获得更多有价值的信息。

* + - 1. 可以更好地进行搜索引擎优化（SEO）。

对于很多SEO从业者来说，为了更好的完成工作，那么就必须要对搜索引擎的工作原理非常清楚，同时也需要掌握搜索引擎爬虫的工作原理。

而学习爬虫，可以更深层次地理解搜索引擎爬虫的工作原理，这样在进行搜索引擎优化时，才能知己知彼，百战不殆。

* + - 1. 有利于就业。

从就业来说，爬虫工程师方向是不错的选择之一，因为目前爬虫工程师的需求越来越大，而能够胜任这方面岗位的人员较少，所以属于一个比较紧缺的职业方向，并且随着大数据时代和人工智能的来临，爬虫技术的应用将越来越广泛，在未来会拥有很好的发展空间。

# 第四章：HttpClient

网络爬虫就是用程序帮助我们访问网络上的资源，我们一直以来都是使用HTTP协议访问互联网的网页，网络爬虫需要编写程序，在这里使用同样的HTTP协议访问网页。

这里我们使用Java的HTTP协议客户端 HttpClient这个技术，来实现抓取网页数据。

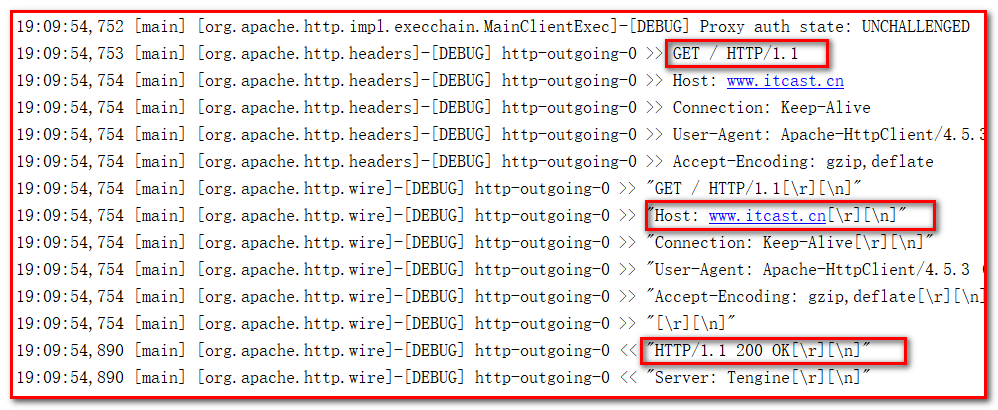
## GET请求

访问传智官网，请求url地址：

<http://www.itcast.cn/>

**public class** HttpGetTest {  
  
 **public static void** main(String[] args) {  
 *//创建HttpClient对象* CloseableHttpClient httpClient = HttpClients.*createDefault*();  
  
 *//创建HttpGet对象，设置url访问地址* HttpGet httpGet = **new** HttpGet(**"http://www.itcast.cn"**);  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *//使用HttpClient发起请求，获取response* response = httpClient.execute(httpGet);  
  
 *//解析响应* **if** (response.getStatusLine().getStatusCode() == 200) {  
 String content = EntityUtils.*toString*(response.getEntity(), **"utf8"**);  
 System.***out***.println(content.length());  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 *//关闭response* **try** {  
 response.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 httpClient.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

请求结果



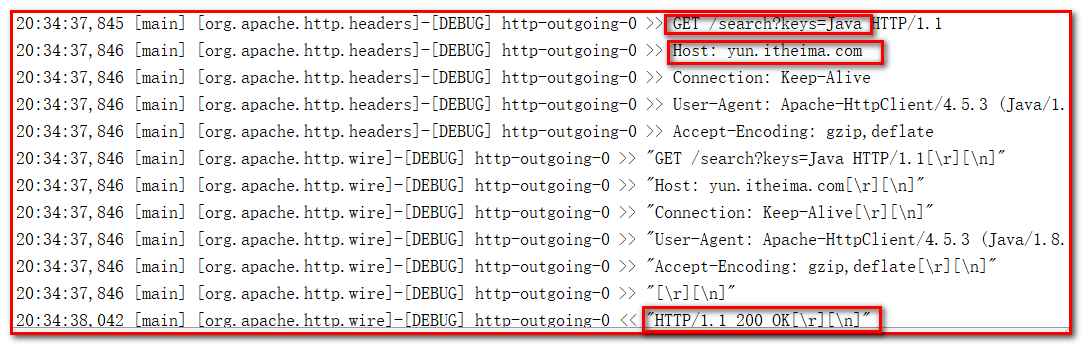
## 带参数的GET请求

在传智中搜索学习视频，地址为：

<http://yun.itheima.com/search?keys=Java>

**public class** HttpGetParamTest {  
  
 **public static void** main(String[] args) **throws** Exception {  
 *//创建HttpClient对象* CloseableHttpClient httpClient = HttpClients.*createDefault*();  
  
 *//设置请求地址是：http://yun.itheima.com/search?keys=Java  
 //创建URIBuilder* URIBuilder uriBuilder = **new** URIBuilder(**"http://yun.itheima.com/search"**);  
 *//设置参数（可以是1个，也可以是多个）* uriBuilder.setParameter(**"keys"**,**"Java"**);  
  
 *//创建HttpGet对象，设置url访问地址* HttpGet httpGet = **new** HttpGet(uriBuilder.build());  
  
 System.***out***.println(**"发起请求的信息："**+httpGet);  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *//使用HttpClient发起请求，获取response* response = httpClient.execute(httpGet);  
  
 *//解析响应* **if** (response.getStatusLine().getStatusCode() == 200) {  
 String content = EntityUtils.*toString*(response.getEntity(), **"utf8"**);  
 System.***out***.println(content.length());  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 *//关闭response* **try** {  
 response.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 httpClient.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

请求结果



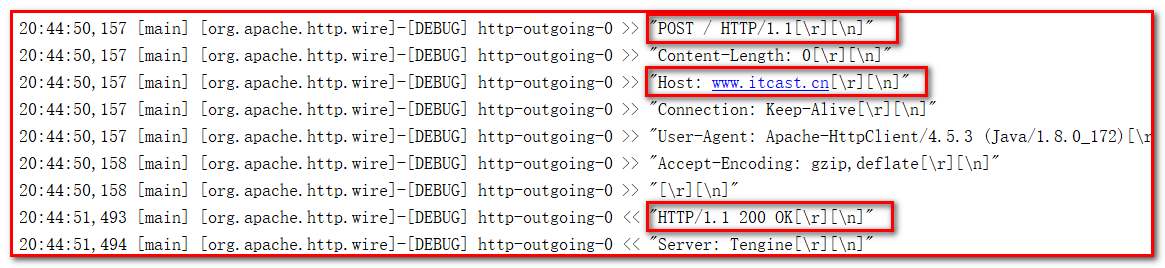
## POST请求

使用POST访问传智官网，请求url地址：

<http://www.itcast.cn/>

**public class** HttpPostTest {  
  
 **public static void** main(String[] args) {  
 *//创建HttpClient对象* CloseableHttpClient httpClient = HttpClients.*createDefault*();  
  
 *//创建HttpPost对象，设置url访问地址* HttpPost httpPost = **new** HttpPost(**"http://www.itcast.cn"**);  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *//使用HttpClient发起请求，获取response* response = httpClient.execute(httpPost);  
  
 *//解析响应* **if** (response.getStatusLine().getStatusCode() == 200) {  
 String content = EntityUtils.*toString*(response.getEntity(), **"utf8"**);  
 System.***out***.println(content.length());  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 *//关闭response* **try** {  
 response.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 httpClient.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

请求结果：



## 带参数的POST请求

在传智中搜索学习视频，使用POST请求，url地址为：

<http://yun.itheima.com/search>

url地址没有参数，参数keys=java放到表单中进行提交

**public class** HttpPostParamTest {  
  
 **public static void** main(String[] args) **throws** Exception {  
 *//创建HttpClient对象* CloseableHttpClient httpClient = HttpClients.*createDefault*();  
  
 *//创建HttpPost对象，设置url访问地址* HttpPost httpPost = **new** HttpPost(**"http://yun.itheima.com/search"**);  
  
 *//声明List集合，封装表单中的参数* List<NameValuePair> params = **new** ArrayList<NameValuePair>();  
 *//设置请求地址是：http://yun.itheima.com/search?keys=Java* params.add(**new** BasicNameValuePair(**"keys"**,**"Java"**));  
  
 *//创建表单的Entity对象,第一个参数就是封装好的表单数据，第二个参数就是编码* UrlEncodedFormEntity formEntity = **new** UrlEncodedFormEntity(params,**"utf8"**);  
  
 *//设置表单的Entity对象到Post请求中* httpPost.setEntity(formEntity);  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *//使用HttpClient发起请求，获取response* response = httpClient.execute(httpPost);  
  
 *//解析响应* **if** (response.getStatusLine().getStatusCode() == 200) {  
 String content = EntityUtils.*toString*(response.getEntity(), **"utf8"**);  
 System.***out***.println(content.length());  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 *//关闭response* **try** {  
 response.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 httpClient.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

请求结果



## 连接池

如果每次请求都要创建HttpClient，会有频繁创建和销毁的问题，可以使用连接池来解决这个问题。

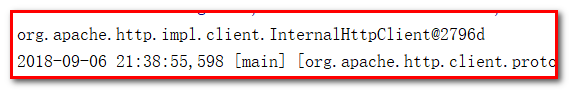
测试以下代码，并断点查看每次获取的HttpClient都是不一样的。

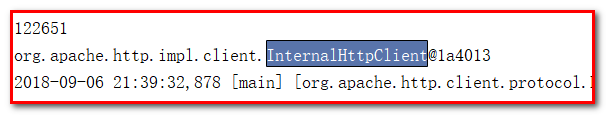
**public class** HttpClientPoolTest {  
  
 **public static void** main(String[] args) {  
 *//创建连接池管理器* PoolingHttpClientConnectionManager cm = **new** PoolingHttpClientConnectionManager();  
  
 *//设置最大连接数* cm.setMaxTotal(100);  
  
 *//设置每个主机的最大连接数（这里指的主机就是http://www.itcast.cn网站）* cm.setDefaultMaxPerRoute(10);  
  
 *//使用连接池管理器发起请求  
 doGet*(cm);  
 *doGet*(cm);  
 }  
  
 **private static void** doGet(PoolingHttpClientConnectionManager cm) {  
 *//不是每次创建新的HttpClient，而是从连接池中获取HttpClient对象* CloseableHttpClient httpClient = HttpClients.*custom*().setConnectionManager(cm).build();

System.***out***.println(httpClient);

HttpGet httpGet = **new** HttpGet(**"http://www.itcast.cn"**);  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 response = httpClient.execute(httpGet);  
  
 **if** (response.getStatusLine().getStatusCode() == 200) {  
 String content = EntityUtils.*toString*(response.getEntity(), **"utf8"**);  
  
 System.***out***.println(content.length());  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 **if** (response != **null**) {  
 **try** {  
 response.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 *//不能关闭HttpClient，由连接池管理HttpClient  
 //httpClient.close();* }  
 }  
  
 }  
}

如果使用System.out.println()查看httpClient对象发现：2个HttpClient是不一样的，说明连接池获取的HttpClient是线程安全的。





## 请求参数

有时候因为网络，或者目标服务器的原因，请求需要更长的时间才能完成，我们需要自定义相关时间设置

**public class** HttpConfigTest {  
  
 **public static void** main(String[] args) {  
 *//创建HttpClient对象* CloseableHttpClient httpClient = HttpClients.*createDefault*();  
  
 *//创建HttpGet对象，设置url访问地址* HttpGet httpGet = **new** HttpGet(**"http://www.itcast.cn"**);  
  
 *//配置请求信息* RequestConfig config = RequestConfig.*custom*().setConnectTimeout(1000) *//创建连接的最长时间，单位是毫秒* .setConnectionRequestTimeout(500) *//设置获取连接的最长时间，单位是毫秒* .setSocketTimeout(10\*1000) *//设置数据传输的最长时间，单位是毫秒* .build();  
  
 *//给请求设置请求信息* httpGet.setConfig(config);  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *//使用HttpClient发起请求，获取response* response = httpClient.execute(httpGet);  
  
 *//解析响应* **if** (response.getStatusLine().getStatusCode() == 200) {  
 String content = EntityUtils.*toString*(response.getEntity(), **"utf8"**);  
 System.***out***.println(content.length());  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 *//关闭response* **try** {  
 response.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 httpClient.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

# 第五章：Jsoup

我们抓取到页面之后，还需要对页面进行解析。可以使用字符串处理工具解析页面，也可以使用正则表达式，但是这些方法都会带来很大的开发成本，所以我们需要使用一款专门解析html页面的技术。

## jsoup介绍

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。

jsoup的主要功能如下：

从一个URL，文件或字符串中解析HTML；

使用DOM或CSS选择器来查找、取出数据（语法类似于JQuery）；

可操作HTML元素、属性、文本；

先加入Jsoup依赖：

*<!--Jsoup-->*<**dependency**>  
 <**groupId**>org.jsoup</**groupId**>  
 <**artifactId**>jsoup</**artifactId**>  
 <**version**>1.10.3</**version**>  
</**dependency**>  
*<!--测试-->*<**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.12</**version**>  
</**dependency**>  
*<!--工具-->*<**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-lang3</**artifactId**>  
 <**version**>3.7</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>commons-io</**groupId**>  
 <**artifactId**>commons-io</**artifactId**>  
 <**version**>2.6</**version**>  
</**dependency**>

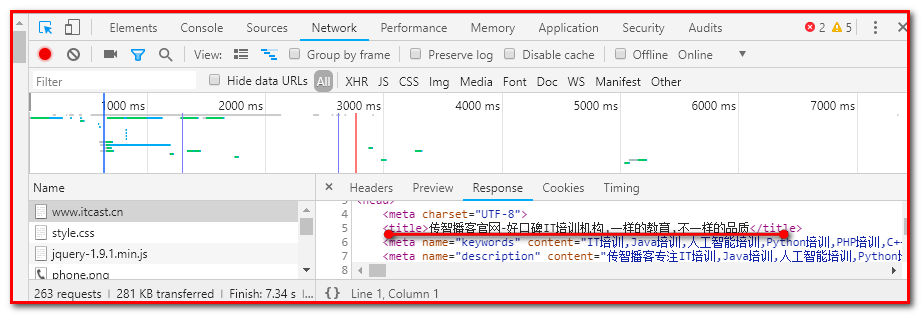
## jsoup解析

### 解析url

创建包com.itheima.test，创建类：

Jsoup可以直接输入url，它会发起请求并获取数据，封装为Document对象

【需求】：解析http://www.itcast.cn网址中，第一个title标签的文本内容

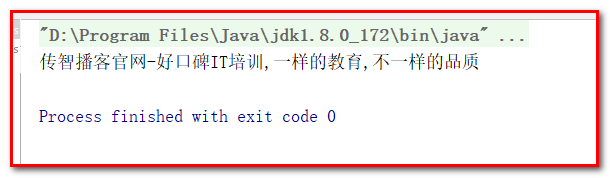


【代码】

**public class** JsoupFirstTest {  
  
 @Test  
 **public void** testUrl() **throws** Exception {  
 *//解析url地址,第一个参数是访问的url，第二个参数是访问时候的超时时间* Document doc = Jsoup.*parse*(**new** URL(**"http://www.itcast.cn"**), 1000);  
  
 *//使用标签选择器，获取title标签中的内容* String title = doc.getElementsByTag(**"title"**).first().text();  
  
 *//打印* System.***out***.println(title);  
 }

}

输出：



PS：虽然使用Jsoup可以替代HttpClient直接发起请求解析数据，但是往往不会这样用，因为实际的开发过程中，需要使用到多线程，连接池，代理等等方式，而jsoup对这些的支持并不是很好，所以我们一般把jsoup仅仅作为Html解析工具使用

### 解析字符串

在桌面上创建jsoup.html，先准备以下html文件内容

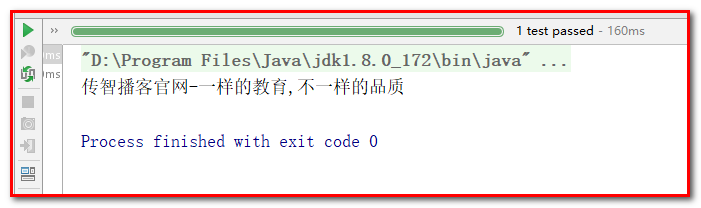
|  |
| --- |
| <html>  <head>  <title>传智播客官网-一样的教育,不一样的品质</title>  </head>  <body>  <div class="city">  <h3 id="city\_bj">北京中心</h3>  <fb:img src="/2018czgw/images/slogan.jpg" class="slogan"/>  <div class="city\_in">  <div class="city\_con" style="display: none;">  <ul>  <li id="test" class="class\_a class\_b">  <a href="http://www.itcast.cn" target="\_blank">  <span class="s\_name">北京</span>  </a>  </li>  <li>  <a href="http://sh.itcast.cn" target="\_blank">  <span class="s\_name">上海</span>  </a>  </li>  <li>  <a href="http://gz.itcast.cn" target="\_blank">  <span abc="123" class="s\_name">广州</span>  </a>  </li>  <ul>  <li>天津</li>  </ul>  </ul>  </div>  </div>  </div>  </body>  </html> |

Jsoup可以直接输入字符串，并封装为Document对象

【需求】：解析jsoup.html网址中，第一个title标签的文本内容

@Test  
**public void** testString() **throws** Exception {  
 *//使用工具类读取文件，获取字符串* String content = FileUtils.*readFileToString*(**new** File(**"C:\\Users\\administratorad\\Desktop\\jsoup.html"**), **"utf8"**);  
  
 *//解析字符串* Document doc = Jsoup.*parse*(content);  
  
 String title = doc.getElementsByTag(**"title"**).first().text();  
  
 System.***out***.println(title);  
  
}

控制台输出：



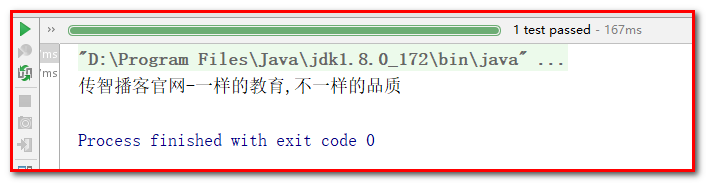
### 解析文件

Jsoup可以直接解析文件，并封装为Document对象

【需求】：解析jsoup.html网址中，第一个title标签的文本内容

@Test  
**public void** testFile() **throws** Exception {  
 *//解析文件* Document doc = Jsoup.*parse*(**new** File(**"C:\\Users\\administratorad\\Desktop\\jsoup.html"**), **"utf8"**);  
  
 String title = doc.getElementsByTag(**"title"**).first().text();  
  
 System.***out***.println(title);  
  
}

控制台输出：



### 使用dom方式遍历文档

#### 元素获取

1. 根据id查询元素getElementById
2. 根据标签获取元素getElementsByTag
3. 根据class获取元素getElementsByClass
4. 根据属性获取元素getElementsByAttribute
5. 根据属性名称和属性值获取元素getElementsByAttributeValue

【需求1】：找到id为city\_bj的元素，并输出文本的内容

【需求2】：找到<span>标签的第一个元素，并输出文本的内容

【需求3】：找到样式为class\_a class\_b的第一个元素，并输出文本的内容

【需求4】：找到属性名称为abc的第一个元素，并输出文本的内容

【需求5】：找到属性名称为href，属性值为http://sh.itcast.cn的第一个元素，并输出文本的内容

@Test  
**public void** testDOM() **throws** Exception {  
 *//解析文件，获取Document对象* Document doc = Jsoup.*parse*(**new** File(**"C:\\Users\\administratorad\\Desktop\\jsoup.html"**), **"utf8"**);  
  
  
 *//获取元素  
 //1. 根据id查询元素getElementById  
 //Element element = doc.getElementById("city\_bj");  
  
 //2. 根据标签获取元素getElementsByTag  
 //Element element = doc.getElementsByTag("span").first();  
  
 //3. 根据class获取元素getElementsByClass  
 //Element element = doc.getElementsByClass("class\_a class\_b").first();  
 //Element element = doc.getElementsByClass("class\_a").first();  
 //Element element = doc.getElementsByClass("class\_b").first();  
  
  
 //4. 根据属性获取元素getElementsByAttribute  
 //Element element = doc.getElementsByAttribute("abc").first();* Element element = doc.getElementsByAttributeValue(**"href"**, **"http://sh.itcast.cn"**).first();  
  
 *//打印元素的内容* System.***out***.println(**"获取到的元素内容是："** + element.text());  
  
}

#### 元素中获取数据

1. 从元素中获取id
2. 从元素中获取className
3. 从元素中获取属性的值attr
4. 从元素中获取所有属性attributes
5. 从元素中获取文本内容text

【需求1】：找到id为test的元素，读取该元素的id字符串

【需求2】：找到id为test的元素，读取该元素的class名称（classNames为复数的集合）

【需求3】：找到id为test的元素，读取该元素的id属性值和class属性值

【需求4】：找到id为test的元素，读取该元素的所有属性的字符串形式（toString）

【需求5】：找到id为test的元素，读取该元素的文本内容

@Test  
**public void** testData() **throws** Exception {  
 *//解析文件，获取Document* Document doc = Jsoup.*parse*(**new** File(**"C:\\Users\\administratorad\\Desktop\\jsoup.html"**), **"utf8"**);  
  
   
 *//根据id获取元素* Element element = doc.getElementById(**"test"**);  
  
 String str = **""**;  
  
 *//元素中获取数据  
 //1. 从元素中获取id* str = element.id();  
  
 *//2. 从元素中获取className* str = element.className();  
 *//Set<String> classSet = element.classNames();  
 //for (String s : classSet ) {  
 // System.out.println(s);  
 //}  
  
 //3. 从元素中获取属性的值attr  
 //str = element.attr("id");* str = element.attr(**"class"**);  
  
 *//4. 从元素中获取所有属性attributes* Attributes attributes = element.attributes();  
 System.***out***.println(attributes.toString());  
  
 *//5. 从元素中获取文本内容text* str = element.text();  
  
 *//打印获取到的内容* System.***out***.println(**"获取到的数据是："** + str);  
  
}

### 使用选择器语法查找元素

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。

### Selector选择器概述

**1、tagname**: 通过标签查找元素，比如：span

**2、#id**: 通过ID查找元素，比如：# city\_bj

**3、.class**: 通过class名称查找元素，比如：.class\_a

**4、[attribute]**: 利用属性查找元素，比如：[abc]

**5、[attr=value]**: 利用属性值来查找元素，比如：[class=s\_name]

【需求1】：找到<span>元素的集合，循环遍历其中的文本内容

【需求2】：找到id为city\_bj的第一个元素，输出其文本内容

【需求3】：找到class为.class\_a的第一个元素，输出其文本内容

【需求4】：找到属性为abc的第一个元素，输出其文本内容

【需求5】：找到class为s\_name的元素集合，循环遍历其中的文本内容

@Test  
**public void** testSelector() **throws** Exception {  
  
 *//解析html文件，获取Document对象* Document doc = Jsoup.*parse*(**new** File(**"C:\\Users\\administratorad\\Desktop\\jsoup.html"**), **"utf8"**);  
  
 *//tagname: 通过标签查找元素，比如：span* Elements elements = doc.select(**"span"**);  
 *//for (Element element : elements) {  
 // System.out.println(element.text());  
 //}  
  
 //#id: 通过ID查找元素，比如：#city\_bj  
 //Element element = doc.select("#city\_bj").first();  
  
 //.class: 通过class名称查找元素，比如：.class\_a  
 //Element element = doc.select(".class\_a").first();  
  
 //[attribute]: 利用属性查找元素，比如：[abc]* Element element = doc.select(**"[abc]"**).first();  
  
 *//[attr=value]: 利用属性值来查找元素，比如：[class=s\_name]* Elements elements1 = doc.select(**"[class=s\_name]"**);  
 **for** (Element element1 : elements1) {  
 System.***out***.println(element1.text());  
 }  
  
  
 *//打印结果* System.***out***.println(**"获取到的结果是："** + element.text());  
}

### Selector选择器组合使用（最多：最精确）

**1、el#id**: 元素+ID，比如： h3#city\_bj

**2、el.class**: 元素+class，比如： li.class\_a

**3、el[attr]**: 元素+属性名，比如： span[abc]

**4、任意组合**: 比如：span[abc].s\_name

**5、ancestor child**: 查找某个元素下子元素（子元素可以是直接子元素，也可以是间接子元素）

比如：.city\_con li 查找"city\_con"下的所有li

**6、parent > child**: 查找某个父元素下的直接子元素，比如：

.city\_con > ul > li 查找city\_con第一级（直接子元素）的ul，再找所有ul下的第一级li

**7、parent > \***: 查找某个父元素下所有直接子元素

【需求1】：找到<h3>标签元素、id为city\_bj的第一个元素，输出文本的值。

【需求2】：找到<li>标签元素、class为class\_a的第一个元素，输出文本的值。

【需求3】：找到<span>标签元素、属性为abc的第一个元素，输出文本的值。

【需求4】：找到<span>标签元素、属性为abc、class为s\_name的第一个元素，输出文本的值。

【需求5】：找到class为city\_con下的间接子元素<li>标签的元素集合，循环遍历输出其中文本的值。

【需求6】：找到class为city\_con下的直接子元素<ul>标签的直接子元素<li>标签的元素集合，循环遍历输出其中文本的值。

【需求7】：找到class为city\_con下的直接子元素<ul>标签的直接子元素下所有的元素集合，循环遍历输出其中文本的值。

@Test  
**public void** testSelector2()**throws** Exception{  
 *//解析html文件，获取Document对象* Document doc = Jsoup.*parse*(**new** File(**"C:\\Users\\administratorad\\Desktop\\jsoup.html"**), **"utf8"**);  
  
 *//el#id: 元素+ID，比如： h3#city\_bj* Element element = doc.select(**"h3#city\_bj"**).first();  
  
 *//el.class: 元素+class，比如： li.class\_a* element = doc.select(**"li.class\_a"**).first();  
  
 *//el[attr]: 元素+属性名，比如： span[abc]* element = doc.select(**"span[abc]"**).first();  
  
 *//任意组合: 比如：span[abc].s\_name* element = doc.select(**"span[abc].s\_name"**).first();  
  
 *//ancestor child: 查找某个元素下子元素，比如：.city\_con li 查找"city\_con"下的所有li* Elements elements = doc.select(**".city\_con li"**);  
  
 *//parent > child: 查找某个父元素下的直接子元素，比如：  
 //.city\_con > ul > li 查找city\_con第一级（直接子元素）的ul，再找所有ul下的第一级li* elements = doc.select(**".city\_con > ul > li"**);  
  
 *//parent > \*: 查找某个父元素下所有直接子元素* elements = doc.select(**".city\_con > ul > \*"**);  
  
  
 System.***out***.println(**"获取到的内容是："**+element.text());  
  
 **for** (Element element1 : elements) {  
 System.***out***.println(**"遍历的结果："**+element1.text());  
 }  
}

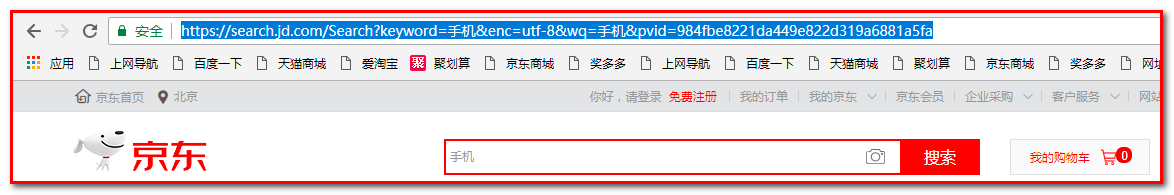
# 第六章：爬虫案例

学习了HttpClient和Jsoup，就掌握了如何抓取数据和如何解析数据，接下来，我们做一个小练习，把京东的手机数据抓取下来，存储到数据库中。

主要目的是HttpClient和Jsoup的学习。

## 需求分析

首先访问京东，搜索手机，分析页面，我们抓取以下商品数据：



商品图片、价格、标题、商品详情页



### SPU和SKU

除了以上四个属性以外，我们发现上图中的苹果手机有四种产品，我们应该每一种都要抓取。那么这里就必须要了解spu和sku的概念

SPU = Standard Product Unit （标准产品单位）

SPU是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。通俗点讲，属性值、特性相同的商品就可以称为一个SPU。

例如上图中的苹果手机就是SPU，包括红色、深灰色、金色、银色，这是什么呢？

SKU=stock keeping unit(库存量单位)

SKU即库存进出计量的单位， 可以是以件、盒、托盘等为单位。SKU是物理上不可分割的最小存货单元。在使用时要根据不同业态，不同管理模式来处理。在服装、鞋类商品中使用最多最普遍。

例如上图中的苹果手机有几个款式，红色苹果手机，就是一个sku

查看页面的源码也可以看出区别



## 开发准备

### 数据库表分析

根据需求分析，我们创建的表如下：

CREATE TABLE `jd\_item` (

`id` bigint(10) NOT NULL AUTO\_INCREMENT COMMENT '主键id',

`spu` bigint(15) DEFAULT NULL COMMENT '商品集合id',

`sku` bigint(15) DEFAULT NULL COMMENT '商品最小品类单元id',

`title` varchar(100) DEFAULT NULL COMMENT '商品标题',

`price` bigint(10) DEFAULT NULL COMMENT '商品价格',

`pic` varchar(200) DEFAULT NULL COMMENT '商品图片',

`url` varchar(200) DEFAULT NULL COMMENT '商品详情地址',

`created` datetime DEFAULT NULL COMMENT '创建时间',

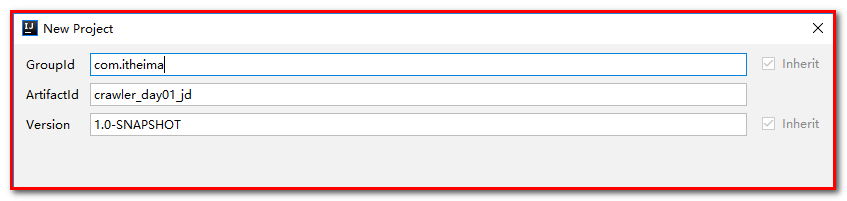
`updated` datetime DEFAULT NULL COMMENT '更新时间',

PRIMARY KEY (`id`),

KEY `sku` (`sku`) USING BTREE

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='京东商品表';

### 创建工程



### 添加依赖

使用Spring Boot+Spring Data JPA和定时任务进行开发，

需要创建Maven工程并添加以下依赖

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.2.RELEASE</**version**>  
 </**parent**>  
  
 <**groupId**>com.itheima</**groupId**>  
 <**artifactId**>crawler\_day01\_jd</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**dependencies**>  
 *<!--SpringMVC-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 *<!--SpringData Jpa-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
 </**dependency**>  
  
 *<!--MySQL连接包-->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 </**dependency**>  
  
 *<!-- HttpClient -->* <**dependency**>  
 <**groupId**>org.apache.httpcomponents</**groupId**>  
 <**artifactId**>httpclient</**artifactId**>  
 </**dependency**>  
  
 *<!--Jsoup-->* <**dependency**>  
 <**groupId**>org.jsoup</**groupId**>  
 <**artifactId**>jsoup</**artifactId**>  
 <**version**>1.10.3</**version**>  
 </**dependency**>  
  
 *<!--工具包-->* <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-lang3</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
  
</**project**>

### 添加配置文件

加入application.properties配置文件

*#DB Configuration:***spring.datasource.driverClassName**=**com.mysql.jdbc.Driver  
spring.datasource.url**=**jdbc:mysql://127.0.0.1:3306/itcastcrawler  
spring.datasource.username**=**root  
spring.datasource.password**=**root***#JPA Configuration:***spring.jpa.database**=**MySQL  
spring.jpa.show-sql**=**true**

## 代码实现

### 编写domain

创建包com.itheima.jd.domain，创建类Item.java

根据数据库表，编写domain

@Entity  
@Table(name = **"jd\_item"**)  
**public class** Item {  
 *//主键* @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private** Long **id**;  
 *//标准产品单位（商品集合）* **private** Long **spu**;  
 *//库存量单位（最小品类单元）* **private** Long **sku**;  
 *//商品标题* **private** String **title**;  
 *//商品价格* **private** Double **price**;  
 *//商品图片* **private** String **pic**;  
 *//商品详情地址* **private** String **url**;  
 *//创建时间* **private** Date **created**;  
 *//更新时间* **private** Date **updated**;  
  
 **public** Long getId() {  
 **return id**;  
 }  
  
 **public void** setId(Long id) {  
 **this**.**id** = id;  
 }  
  
 **public** Long getSpu() {  
 **return spu**;  
 }  
  
 **public void** setSpu(Long spu) {  
 **this**.**spu** = spu;  
 }  
  
 **public** Long getSku() {  
 **return sku**;  
 }  
  
 **public void** setSku(Long sku) {  
 **this**.**sku** = sku;  
 }  
  
 **public** String getTitle() {  
 **return title**;  
 }  
  
 **public void** setTitle(String title) {  
 **this**.**title** = title;  
 }  
  
 **public** Double getPrice() {  
 **return price**;  
 }  
  
 **public void** setPrice(Double price) {  
 **this**.**price** = price;  
 }  
  
 **public** String getPic() {  
 **return pic**;  
 }  
  
 **public void** setPic(String pic) {  
 **this**.**pic** = pic;  
 }  
  
 **public** String getUrl() {  
 **return url**;  
 }  
  
 **public void** setUrl(String url) {  
 **this**.**url** = url;  
 }  
  
 **public** Date getCreated() {  
 **return created**;  
 }  
  
 **public void** setCreated(Date created) {  
 **this**.**created** = created;  
 }  
  
 **public** Date getUpdated() {  
 **return updated**;  
 }  
  
 **public void** setUpdated(Date updated) {  
 **this**.**updated** = updated;  
 }  
}

### 编写dao

创建com.itheima.dao，创建类ItemDao.java

**public interface** ItemDao **extends** JpaRepository<Item,Long> {  
}

### 编写service

创建包com.itheima.service。

ItemService接口

**public interface** ItemService {  
  
 */\*\*  
 \* 保存商品  
 \** ***@param item*** *\*/* **public void** save(Item item);  
  
 */\*\*  
 \* 根据条件查询商品  
 \** ***@param item*** *\** ***@return*** *\*/* **public** List<Item> findAll(Item item);  
}

创建包com.itheima.service.impl。

ItemServiceImpl实现类

@Service  
@Transactional  
**public class** ItemServiceImpl **implements** ItemService {  
  
 @Autowired  
 **private** ItemDao **itemDao**;  
  
 @Override  
 **public void** save(Item item) {  
 **itemDao**.save(item);  
 }  
  
 @Override  
 **public** List<Item> findAll(Item item) {  
 *//声明查询条件* Example<Item> example = Example.*of*(item);  
 *//根据查询条件进行查询数据* List<Item> list = **itemDao**.findAll(example);  
 **return** list;  
 }  
}

### 编写引导类

在包com.itheima.jd下创建，springboot的引导类。

@SpringBootApplication  
*//使用定时任务，需要先开启定时任务，需要添加注解*@EnableScheduling  
**public class** Application {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(Application.**class**, args);  
 }  
}

### 封装HttpClient

创建包com.itheima.jd.utils，创建类HttpUtils.java

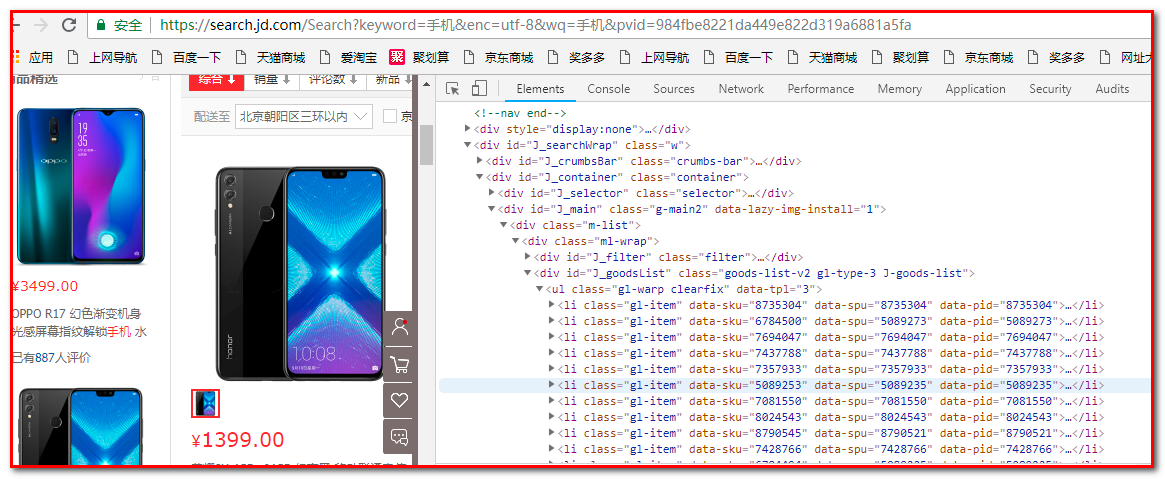
我们需要经常使用HttpClient，所以需要进行封装，方便使用

目的：传递一个url（http的地址），返回对应地址下的HTML的文档

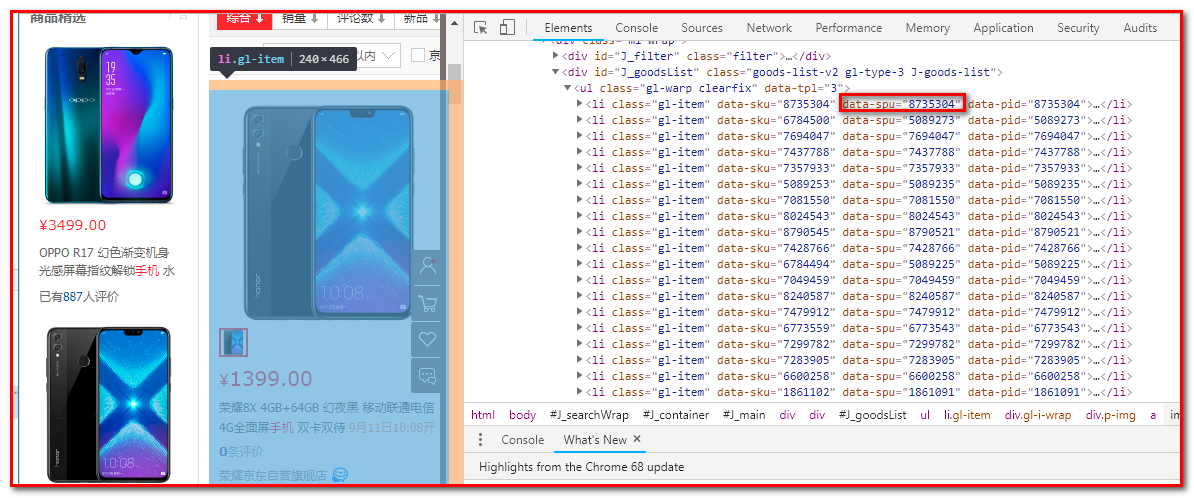
@Component  
**public class** HttpUtils {  
  
 *// 使用连接管理器* **private** PoolingHttpClientConnectionManager **cm**;  
  
 **public** HttpUtils() {  
 **this**.**cm** = **new** PoolingHttpClientConnectionManager();  
  
 *// 设置最大连接数* **cm**.setMaxTotal(200);  
  
 *// 设置每个主机的并发数* **cm**.setDefaultMaxPerRoute(20);  
 }  
  
 *//获取内容* **public** String getHtml(String url) {  
 *// 获取HttpClient对象* CloseableHttpClient httpClient = HttpClients.*custom*().setConnectionManager(**cm**).build();  
  
 *// 声明httpGet请求对象* HttpGet httpGet = **new** HttpGet(url);  
 *// 设置请求参数RequestConfig* httpGet.setConfig(**this**.getConfig());  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *// 使用HttpClient发起请求，返回response* response = httpClient.execute(httpGet);  
 *// 解析response返回数据* **if** (response.getStatusLine().getStatusCode() == 200) {  
 String html = **""**;  
  
 *// 如果response.getEntity获取的结果是空，在执行EntityUtils.toString会报错  
 // 需要对Entity进行非空的判断* **if** (response.getEntity() != **null**) {  
 html = EntityUtils.*toString*(response.getEntity(), **"UTF-8"**);  
 }  
  
 **return** html;  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 **if** (response != **null**) {  
 *// 关闭连接* response.close();  
 }  
 *// 不能关闭，现在使用的是连接管理器  
 // httpClient.close();* } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **return null**;  
 }  
  
 *//获取图片* **public** String getImage(String url) {  
 *// 获取HttpClient对象* CloseableHttpClient httpClient = HttpClients.*custom*().setConnectionManager(**cm**).build();  
  
 *// 声明httpGet请求对象* HttpGet httpGet = **new** HttpGet(url);  
 *// 设置请求参数RequestConfig* httpGet.setConfig(**this**.getConfig());  
  
 CloseableHttpResponse response = **null**;  
 **try** {  
 *// 使用HttpClient发起请求，返回response* response = httpClient.execute(httpGet);  
 *// 解析response下载图片* **if** (response.getStatusLine().getStatusCode() == 200) {  
 *// 获取文件后缀（文件类型）* String extName = url.substring(url.lastIndexOf(**"."**));  
 *// 使用uuid生成图片名* String imageName = UUID.*randomUUID*().toString() + extName;  
  
 *// 声明输出的文件* OutputStream outstream = **new** FileOutputStream(**new** File(**"D:/images/"** + imageName));  
 *// 使用响应体输出文件* response.getEntity().writeTo(outstream);  
  
 *// 返回生成的图片名* **return** imageName;  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 **if** (response != **null**) {  
 *// 关闭连接* response.close();  
 }  
 *// 不能关闭，现在使用的是连接管理器  
 // httpClient.close();* } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **return null**;  
 }  
  
 *//获取请求参数对象* **private** RequestConfig getConfig() {  
 RequestConfig config = RequestConfig.*custom*().setConnectTimeout(1000)*// 设置创建连接的超时时间* .setConnectionRequestTimeout(500) *// 设置获取连接的超时时间* .setSocketTimeout(10000) *// 设置连接的超时时间* .build();  
  
 **return** config;  
 }  
  
}

### 实现数据抓取

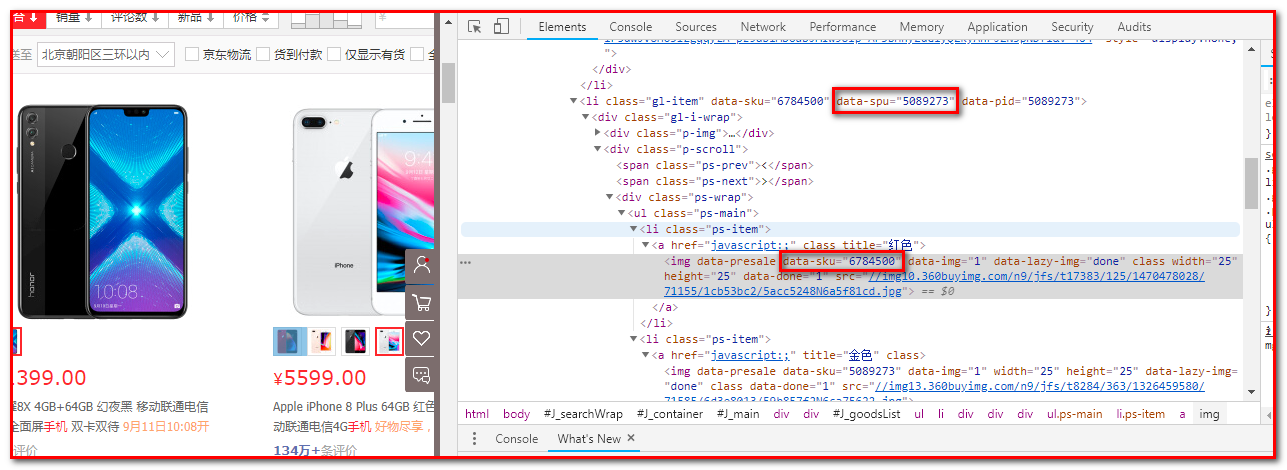
使用谷歌浏览器查看页面的商品标签。



【查找spu】



【查找sku】

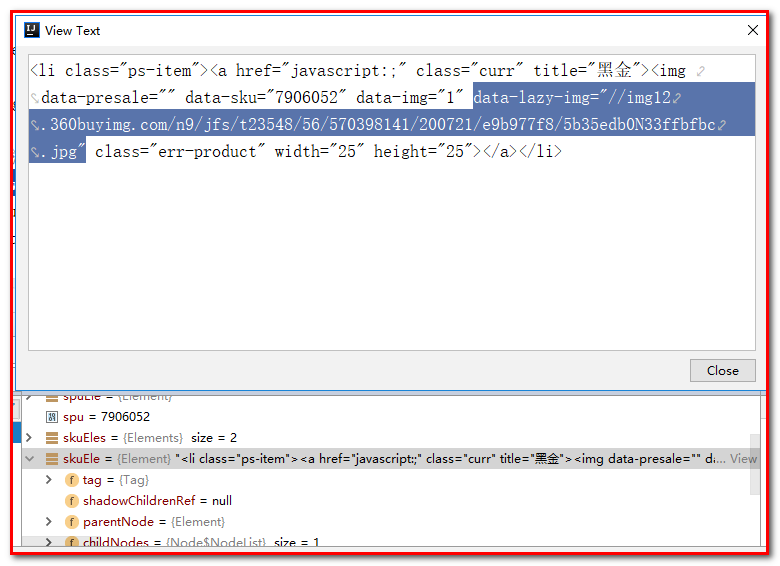


使用定时任务，可以定时抓取最新的数据

@Component  
**public class** ItemTask {  
 @Autowired  
 **private** HttpUtils **httpUtils**;  
 @Autowired  
 **private** ItemService **itemService**;  
 *// json数据解析器* **private static final** ObjectMapper ***mapper*** = **new** ObjectMapper();  
  
  
 *//设置方法每间隔100秒重复执行* @Scheduled(fixedDelay = 1000\*100)  
 **public void** testTask() **throws** IOException {  
 String url = **"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&s=59&click=0&page="**;  
  
 **for** (**int** i = 1; i <= 10; i+=2) {  
 String html = **httpUtils**.getHtml(url + i);  
 *//System.out.println(html);  
 //解析html* parseHtml(html);  
 }  
 }  
  
 *//解析html* **private void** parseHtml(String html) **throws** IOException {  
 *//使用Jsoup解析网页内容* Document doc = Jsoup.*parse*(html);  
  
 *//得到所有SPU标签对象* Elements spuEles = doc.select(**"div#J\_goodsList > ul > li"**);

*//循环得到spu对应的所有SKU标签对象* **for** (Element spuEle : spuEles) {  
 *//得到spu的值* String spuId = spuEle.attr(**"data-spu"**);  
 *//得到spu对应的所有sku标签信息* Elements skuEles = spuEle.select(**"li.ps-item img"**);  
 **for** (Element skuEle : skuEles) {  
 *//得到sku的值* String skuId = skuEle.attr(**"data-sku"**);  
  
 *//创建Item对象，并封装数据* Item item = **new** Item();  
 item.setSku(Long.*parseLong*(skuId));  
  
 List<Item> list = **this**.**itemService**.findAll(item);  
 **if**(list.size()>0){  
 **continue**;  
 }   
  
 item.setSpu(Long.*parseLong*(spuId));  
 item.setCreated(**new** Date());  
 item.setUpdated(**new** Date());  
 *// 商品详情的url地址* item.setUrl(**"https://item.jd.com/"**+skuId+**".html"**);  
 *// 商品图片，完成上传* String imgUrl = **"https:"**+skuEle.attr(**"data-lazy-img"**).replace(**"/n9/"**,**"/n7/"**);  
 String imgName = **this**.**httpUtils**.getImage(imgUrl);  
 item.setPic(imgName);  
 *// 商品标题* String infoHtml = **this**.**httpUtils**.getHtml(item.getUrl());  
 Document infoDoc = Jsoup.*parse*(infoHtml);  
 String title = infoDoc.select(**"div.sku-name"**).text();  
 item.setTitle(title);  
 *// 商品价格* String jsonHtml = **this**.**httpUtils**.getHtml(**"https://p.3.cn/prices/mgets?&skuIds=J\_"** + skuId);  
 **double** price = ***mapper***.readTree(jsonHtml).get(0).get(**"p"**).asDouble();  
 item.setPrice(price);  
  
 *//保存数据* **itemService**.save(item);  
  
 }  
 }  
  
 }  
}

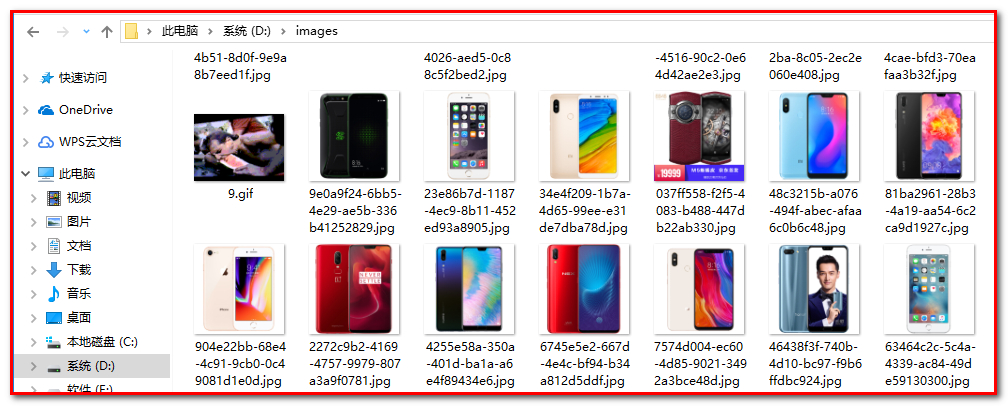
这里注意有些时候我们在谷歌浏览器看到的属性和真实的属性不一致，我们可以使用idea的debug调试。这是存放图片地址的data-lazy-img属性，而不是src属性



执行main方法：

使用定时任务，抓取数据

查看下载图片：



查看数据库：

