# 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。

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

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';

### 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>cn.itcast.crawler</groupId>  
 <artifactId>itcast-crawler-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/crawler  
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pojo

根据数据库表，编写pojo

@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;

set/get  
}

### 编写dao

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

### 编写Service

ItemService接口

public interface ItemService {  
  
 //根据条件查询数据  
 public List<Item> findAll(Item item);  
  
 //保存数据  
 public void save(Item item);  
}

ItemServiceImpl实现类

@Service  
public class ItemServiceImpl implements ItemService {  
  
 @Autowired  
 private ItemDao itemDao;  
  
 @Override  
 public List<Item> findAll(Item item) {  
 Example example = Example.*of*(item);  
 List list = this.itemDao.findAll(example);  
 return list;  
 }  
  
 @Override  
 @Transactional  
 public void save(Item item) {  
 this.itemDao.save(item);  
 }  
}

### 编写引导类

@SpringBootApplication  
//设置开启定时任务  
@EnableScheduling  
public class Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Application.class, args);  
 }  
}

### 封装HttpClient

我们需要经常使用HttpClient，所以需要进行封装，方便使用

@Component  
public class HttpUtils {  
  
 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;  
  
 public static final ObjectMapper *MAPPER* = new ObjectMapper();  
  
  
 //设置定时任务执行完成后，再间隔100秒执行一次  
 @Scheduled(fixedDelay = 1000 \* 100)  
 public void process() throws Exception {  
 //分析页面发现访问的地址,页码page从1开始，下一页oage加2  
 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&cid2=653&cid3=655&s=5760&click=0&page=";  
  
 //遍历执行，获取所有的数据  
 for (int i = 1; i < 10; i = i + 2) {  
 //发起请求进行访问，获取页面数据,先访问第一页  
 String html = this.httpUtils.getHtml(url + i);  
  
 //解析页面数据，保存数据到数据库中  
 this.parseHtml(html);  
  
 }  
 System.*out*.println("执行完成");  
 }  
  
  
 //解析页面，并把数据保存到数据库中  
 private void parseHtml(String html) throws Exception {  
 //使用jsoup解析页面  
 Document document = Jsoup.*parse*(html);  
  
 //获取商品数据  
 Elements spus = document.select("div#J\_goodsList > ul > li");  
  
 //遍历商品spu数据  
 for (Element spuEle : spus) {  
 //获取商品spu  
 Long spuId = Long.*parseLong*(spuEle.attr("data-spu"));  
  
 //获取商品sku数据  
 Elements skus = spuEle.select("li.ps-item img");  
 for (Element skuEle : skus) {  
 //获取商品sku  
 Long skuId = Long.*parseLong*(skuEle.attr("data-sku"));  
  
 //判断商品是否被抓取过，可以根据sku判断  
 Item param = new Item();  
 param.setSku(skuId);  
 List<Item> list = this.itemService.findAll(param);  
 //判断是否查询到结果  
 if (list.size() > 0) {  
 //如果有结果，表示商品已下载，进行下一次遍历  
 continue;  
 }  
  
 //保存商品数据，声明商品对象  
 Item item = new Item();  
  
 //商品spu  
 item.setSpu(spuId);  
 //商品sku  
 item.setSku(skuId);  
 //商品url地址  
 item.setUrl("https://item.jd.com/" + skuId + ".html");  
 //创建时间  
 item.setCreated(new Date());  
 //修改时间  
 item.setUpdated(item.getCreated());  
  
  
 //获取商品标题  
 String itemHtml = this.httpUtils.getHtml(item.getUrl());  
 String title = Jsoup.*parse*(itemHtml).select("div.sku-name").text();  
 item.setTitle(title);  
  
 //获取商品价格  
 String priceUrl = "https://p.3.cn/prices/mgets?skuIds=J\_"+skuId;  
 String priceJson = this.httpUtils.getHtml(priceUrl);  
 //解析json数据获取商品价格  
 double price = *MAPPER*.readTree(priceJson).get(0).get("p").asDouble();  
 item.setPrice(price);  
  
 //获取图片地址  
 String pic = "https:" + skuEle.attr("data-lazy-img").replace("/n9/","/n1/");  
 System.*out*.println(pic);  
 //下载图片  
 String picName = this.httpUtils.getImage(pic);  
 item.setPic(picName);  
  
 //保存商品数据  
 this.itemService.save(item);  
 }  
 }  
 }  
}