基于电信DPI数据的电商用户行为分析

# 第三章 电商网站基础数据获取与处理

## 3.1电商网站的基础数据采集

### 3.1.1 网络爬虫技术原理概述

网络爬虫（Web crawler），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本，它们被广泛用于互联网搜索引擎或其他类似网站，可以自动采集所有其能够访问到的页面内容，以获取或更新这些网站的内容和检索方式。从功能上来讲，爬虫一般分为数据采集，处理，储存三个部分。传统爬虫从一个或若干初始网页的URL开始，获得初始网页上的URL，在抓取网页的过程中，不断从当前页面上抽取新的URL放入队列,直到满足系统的一定停止条件。

在网络爬虫的系统框架中，主过程由控制器，解析器，资源库三部分组成。控制器的主要工作是负责给多线程中的各个爬虫线程分配工作任务。解析器的主要工作是下载网页，进行页面的处理，主要是将一些JS脚本标签、CSS代码内容、空格字符、HTML标签等内容处理掉，爬虫的基本工作是由解析器完成。资源库是用来存放下载到的网页资源，一般都采用大型的数据库存储，如Nosql/Mysql数据库，并对其建立索引。

控制器

控制器是网络爬虫的中央控制器，它主要是负责根据系统传过来的URL链接，分配一线程，然后启动线程调用爬虫爬取网页的过程。

解析器

解析器是负责网络爬虫的主要部分，其负责的工作主要有：下载网页的功能，对网页的文本进行处理，如过滤功能，抽取特殊HTML标签的功能，分析数据功能。

资源库

主要是用来存储网页中下载下来的数据记录的容器，并提供生成索引的目标源。中大型的数据库产品有：Oracle、Sql Server等。

Web网络爬虫系统一般会选择一些比较重要的、出度(网页中链出超链接数)较大的网站的URL作为种子URL集合。网络爬虫系统以这些种子集合作为初始URL，开始数据的抓取。因为网页中含有链接信息，通过已有网页的 URL会得到一些新的 URL，可以把网页之间的指向结构视为一个森林，每个种子URL对应的网页是森林中的一棵树的根节点。这样，Web网络爬虫系统就可以根据广度优先算法或者深度优先算法遍历所有的网页。由于深度优先搜索算法可能会使爬虫系统陷入一个网站内部，不利于搜索比较靠近网站首页的网页信息，因此搜索引擎一般采用广度优先搜索算法采集网页。Web网络爬虫系统首先将种子URL放入下载队列，然后简单地从队首取出一个URL下载其对应的网页。得到网页的内容将其存储后，再经过解析网页中的链接信息可以得到一些新的URL，将这些URL加入下载队列。然后再取出一个URL，对其对应的网页进行下载，然后再解析，如此反复进行，直到遍历了整个网络或者满足某种条件后才会停止下来。

### 3.1.2 基于深度优先网络爬虫的电商网站商品信息获取

网络爬虫的实现主要关注于一下三个方面：

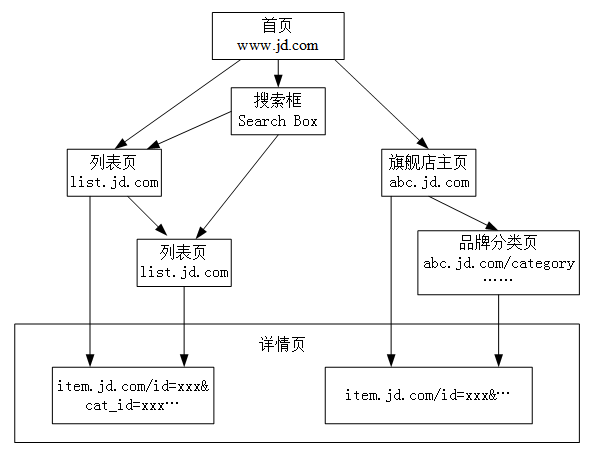
(1) 对抓取目标的描述或定义

(2) 对网页或数据的分析与过滤；

(3) 对URL的搜索策略

由于本文不是致力于实现类似搜索引擎的web通用爬虫，而是聚焦于垂直细分领域里电商网站的商品信息获取，因此该应用场景适用于根据网站特点自定义的专用爬虫。

电商网站的页面结构通常由主页(Home Page)，列表页(List Page)，类别页(Category Page)以及详情页(Detail Page)构成，如下图所示：



针对电商网站其页面结构的特性，本文采用了深度优先遍历策略的网络爬虫获取电商网站的商品信息。

深度优先遍历策略是指网络爬虫会从起始页开始，一个链接一个链接跟踪下去，处理完这条线路之后再转入下一个起始页，继续跟踪链接。我们以下面的图为例：



该结构中根据深度遍历优先策略的访问路径为：A-F-G-E-H-I-B-C-D

以京东商城网站为例，商品页面信息的获取流程为

1.首先选取种子URL，并将这些URL放入待抓取URL队列；

2.从待抓取URL队列中取出待抓取在URL，并将URL对应的网页下载下来，存储进已下载网页库中。此外，将这些URL放进已抓取URL队列；

3. 分析已抓取URL队列中页面中包含的其他URL，解析并过滤，并且将URL放入待抓取URL队列，从而进入下一个循环。

京东商城商品页面的种子URL为其主页，通过Chrome浏览器的页面解析器分析得出其一级商品分类的URL。



根据其页面的Html结构，进一步得到列表页的Html结构，如下所示：



在列表页面中，得到了商品详情页的URL，即最终目标的URL，如图中item.jd.com/2351657.html。

### 3.1.3 Html页面解析

在上一小节中提到的Html页面结构解析方法，文档对象模型 (DOM)作为Html的解析模型和基础，在另一个层面上即是HTML和XML文档的编程接口。它提供了对文档的结构化的表述，并定义了一种方式可以使从程序中对该结构进行访问，从而改变文档的结构，样式和内容。DOM 将文档解析为一个由节点和对象（包含属性和方法的对象）组成的结构集合。简言之，它会将web页面和脚本或程序语言连接起来。

一个web页面是一个文档。这个文档可以在浏览器窗口或作为HTML源码显示出来。但上述两个情况中都是同一份文档。文档对象模型（DOM）提供了对同一份文档的另一种表现，存储和操作的方式。 DOM是web页面的完全的面向对象表述，它能够使用如 JavaScript等脚本语言进行修改。

目前Python语言主要有正则表达式的方法和基于Xpath的Lxml库以及基于Dom解析的BeautifulSoup库。

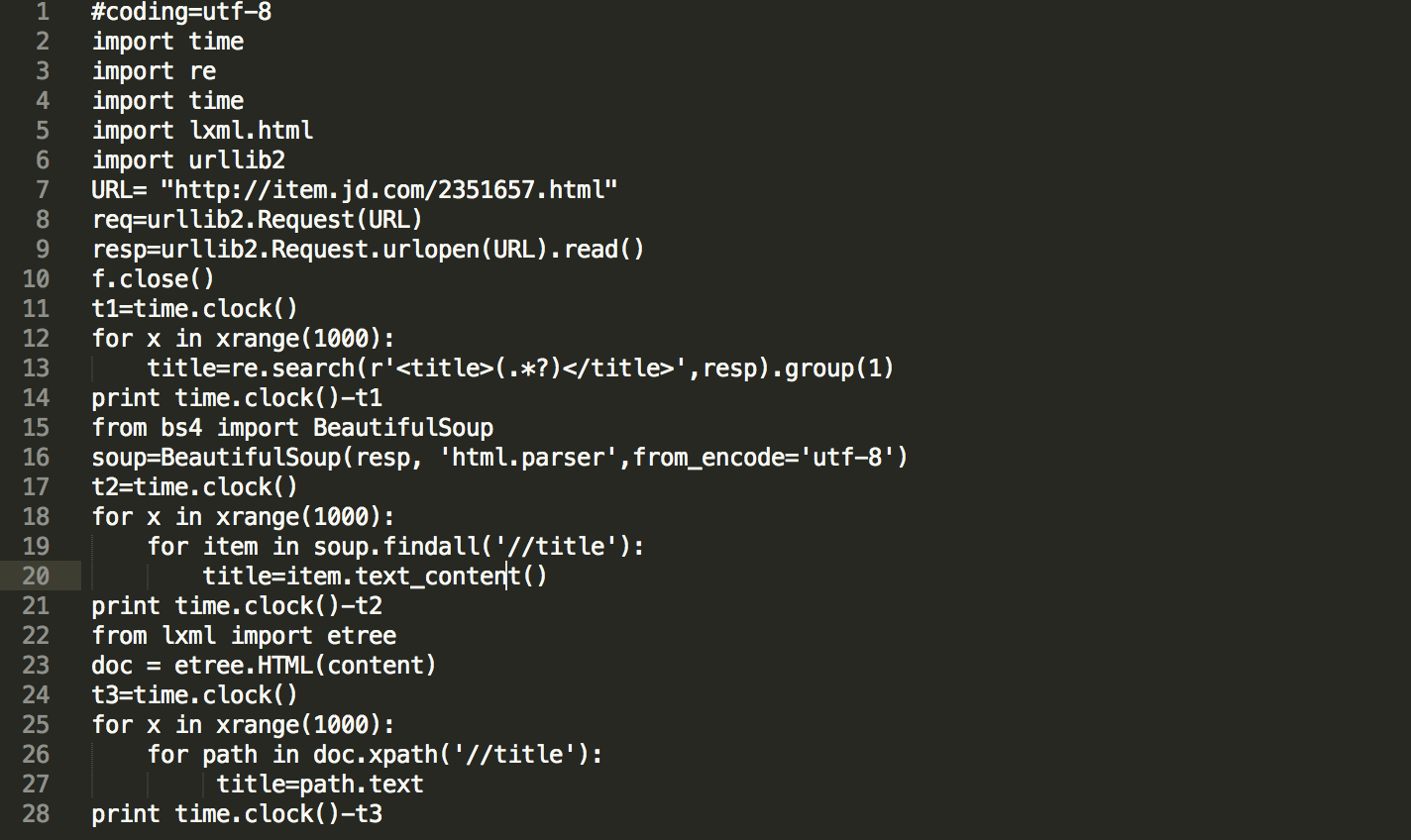
正则表达式解析页面原理：即将Html页面当做一个文本文件，去匹配该文本文件中出现的目标字符，举例说明如下：

|  |  |  |
| --- | --- | --- |
| 应用范围 | 说明 | 正则表达式 |
| 获取<tr></tr>标签之间内容 | 获取两个标签之间的内容，通常这种标签都是成对出现的 | regx\_ example= r'<tr>(.\*?)</tr>'  result=re.findall(regx\_example,language,re.S|re.M) |
| 获取超链接<a href=..></a>之间内容 | 分析网页链接，获取URL或网页内容 | regx\_ example = r'<a .\*?>(.\*?)</a>'  result=re.findall(regx\_ example, content, re.S|re.M) |
| 获取网页标题title | 京东商品页面标题中含有丰富的信息 | regx\_example= re.findall(r'<title>(.\*?)</title>', content) |

Xpath解析页面原理：XPath即为XML路径语言（XML Path Language），它是一种用来确定XML文档中某部分位置的语言。XPath基于XML的树状结构，提供在数据结构树中找寻节点的能力。Xpath 要求一定清楚文档层次结构，它通过元素和属性进行导航，可以使用绝对路径或相对路径查找，其解析只会局部遍历

BeautifulSoup解析页面原理：Beautiful Soup是一个可以从HTML或XML文件中提取数据的Python库. BeautifulSoup和XPath的原理不一样，BeautifulSoup是基于DOM的，会载入整个文档，解析整个DOM树，因此时间和内存开销都会大很多。Beautifulsoup 不必清楚文档结构，可以直接找某些标签，简单粗暴。

对于京东商品页面的解析三种方法的性能对比测试，关键代码如下：



性能对比结果如下：

0.0057619817698

0.133525497136

0.0778033702951

由上可以看出，正则表达式解析最快，BeautifulSoup最为耗时，对于电商网站页面清晰的结构，综合执行性能和开发效率而言，Xpath更适用于本文中电商网站商品页面信息的解析获取。

### 3.1.4 多线程和分布式爬虫在电商网站商品信息采集中的应用

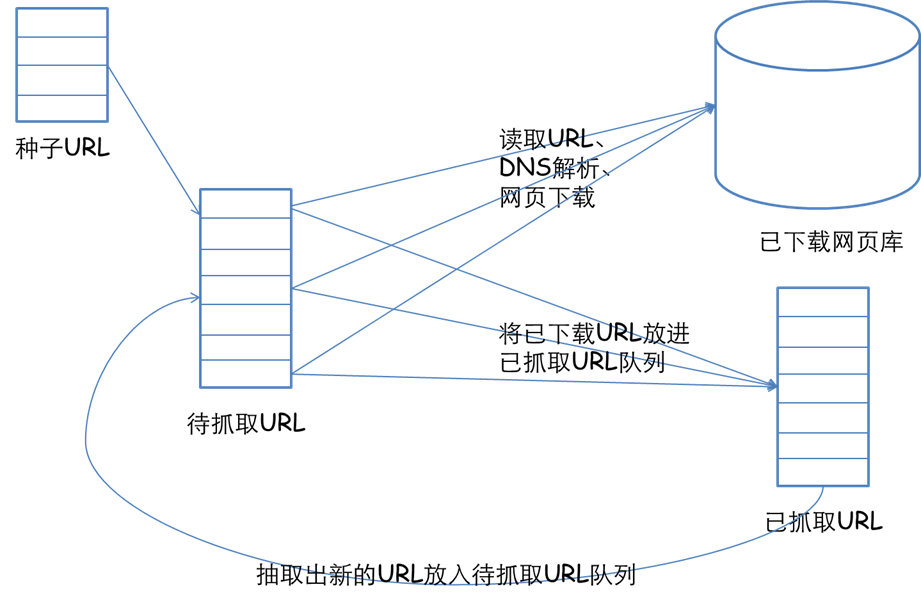
现代计算机技术的迅猛发展，多核和多CPU已经批片集成到计算机硬件中。而多线程是让这些硬件资源得到充分利用的一种编程技术。多线程，是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。具有这种能力的系统包括对称多处理机、多核心处理器以及芯片级多处理（Chip-level multithreading）或同时多线程（Simultaneous multithreading）处理器。在一个程序中，这些独立运行的程序片段叫作“线程”（Thread），利用它编程的概念就叫作多线程处理技术。

使用多线程的优缺点：  
优点：  
1）适当的提高程序的执行效率（多个线程同时执行）。  
2）适当的提高了资源利用率（CPU、内存等）。  
缺点：  
1）占用一定的内存空间。  
2）线程越多CPU的调度开销越大。  
3）程序的复杂度会上升。

在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。在这种情况下可以释放一些珍贵的资源如内存占用等等

网络爬虫作为一种大量涉及到文件读写和网络IO的技术，多线程无疑是网络爬虫性能高效提升的重要手段和途径。

多线程爬虫结构如下图所示：



Python中对多线程提供了方便的支持，主要是通过thread和threading这两个模块来实现的。python的thread模块是比较底层的模块(或者说轻量级)，python的threading模块是对thread做了一些包装的，可以更加方便的被使用。

Threading模块经常和Queue结合使用,Queue模块中提供了同步的、线程安全的队列类，包括FIFO（先入先出）队列Queue，LIFO（后入先出）队列LifoQueue，和优先级队列PriorityQueue。这些队列都实现了锁原语，能够在多线程中直接使用。可以使用队列来实现线程间的同步。

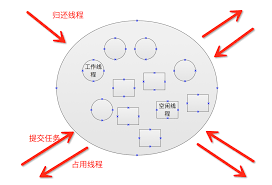
Threading模块中常用的函数和对象如下表所示：

|  |  |  |
| --- | --- | --- |
| 名称 | 对象属性 | 作用 |
| Threading.active\_count() | 函数 | 返回当前线程对象Thread的个数 |
| Threading.enumreate() | 函数 | 返回当前运行的线程对象Thread(包括后台的)的list |
| Threading.Condition() | 函数 | 返回条件变量对象的工厂函数, 主要用户线程的并发 |
| Threading.Lock() | 函数 | 返回一个新的锁对象, 是在thread模块的基础上实现的 与acquire()和release()结合使用 |
| Threading.Thread | 类 | 一个表示线程控制的类, 这个类常被继承 |
| Threading.Timer | 类 | 定时器,线程在一定时间后执行 |
| Threading.ThreadError | 类 | 引发中各种线程相关异常 |

通常多线程技术应用到在抓取网页的时候，一个简单的思路就是为每个网页启动一个线程。在要抓取的网页比较少的时候——比如百十来个——这样子还是可行的。但是网页比较多的时候，这样做就不太合理了。因为线程的创建启动和运行都会消耗很多的资源，线程启动太多会耗尽资源导致机器卡死。而且，创建线程后只执行一次也是一种浪费。为了减少线程的创建、实现线程的重复利用，我们需要引入线程池。

可以使用python的ThreadPoolExecutor来创建线程池，线程池的基本思想还是一种对象池的思想，开辟一块内存空间，里面存放了众多(未死亡)的线程，池中线程执行调度由池管理器来处理。当有线程任务时，从池中取一个，执行完成后线程对象归池，这样可以避免反复创建线程对象所带来的性能开销，节省了系统的资源。线程池的优势在于：

1）避免线程的创建和销毁带来的性能开销。  
 2）避免大量的线程间因互相抢占系统资源导致的阻塞现象。  
 3）能够对线程进行简单的管理并提供定时执行、间隔执行等功能。



随着大数据时代的来临，web网页数据量相应的暴增，网络爬虫的资源消耗也随其快速增长，多线程在单机上的瓶颈逐渐凸显，CPU，内存，网络带宽，磁盘大小无一对爬虫技术带来巨大的挑战，自然而然，分布式计算及存储技术在爬虫领域也发挥出重要的作用。

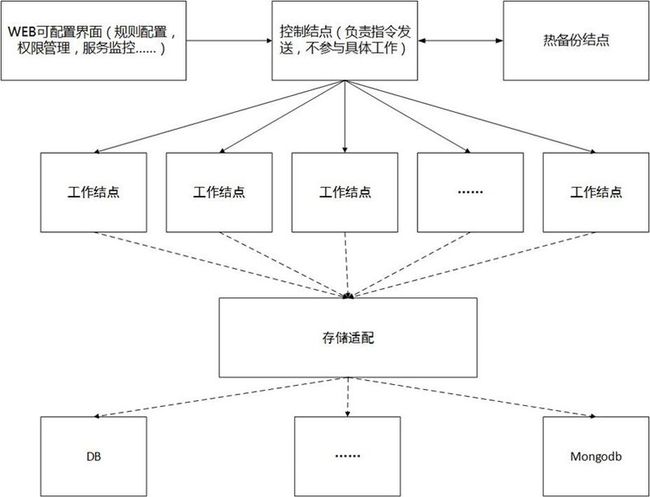
分布式爬虫的优点如下：

高并发，单机服务器资源有限，使用分布式爬虫可以获得更高的并发量。

高可用性，爬虫部署到N台服务器，即使N-1台服务器故障，也不会影响到服务。

可伸缩性，添加爬虫或减少爬虫，不会对服务造成任何影响。

经济性，部分爬虫可以部署到廉价机器上，甚至自己电脑上。



该分布式爬虫架构中，控制节点负责收集需要爬取的URL集合，根据制定的规则将待爬取的URL分配到工作节点中，并收集从工作节点根据自身运行情况返回来的信息，动态的调整工作节点任务权重，即当工作节点负载高时，降低对其的任务分配，反之亦然。工作节点根据自身分配得到的任务建立HTTP请求，下载并解析页面，将结果放入村吃适配的缓存队列中。缓存队列通常是依据生产者/消费者模型实现的。

生产者/消费者模型描述的是有一块缓冲区作为仓库，生产者可将产品放入仓库，消费者可以从仓库中取出产品，生产者/消费者模型关注的是以下几个方面：

1）生产者生产的时候消费者不能消费

2）消费者消费的时候生产者不能生产

3）缓冲区空时消费者不能消费

4）缓冲区满时生产者不能生产

生产者/模型作为一种重要的模型，它的优点在于：

1、解耦。因为多了一个缓冲区，所以生产者和消费者并不直接相互调用，这一点很容易想到，这样生产者和消费者的代码发生变化，都不会对对方产生影响，这样其实就把生产者和消费者之间的强耦合解开，变为了生产者和缓冲区/消费者和缓冲区之间的弱耦合

2、通过平衡生产者和消费者的处理能力来提高整体处理数据的速度，这是生产者/消费者模型最重要的一个优点。如果消费者直接从生产者这里拿数据，如果生产者生产的速度很慢，但消费者消费的速度很快，那消费者就得占用CPU的时间片白白等在那边。有了生产者/消费者模型，生产者和消费者就是两个独立的并发体，生产者把生产出来的数据往缓冲区一丢就好了，不必管消费者；消费者也是，从缓冲区去拿数据就好了，也不必管生产者，缓冲区满了就不生产，缓冲区空了就不消费，使生产者/消费者的处理能力达到一个动态的平衡。

## 3.2基于开源爬虫框架的电商网站数据获取

### 3.3.1 常用开源爬虫框架介绍

#### 1．Nutch

 Nutch是基于hadoop的框架的，所以它天生就是分布式的,可以同时利用多台机器来爬取数据。众所周知的是，IP对于爬虫来说是一个重要的资源，大量的爬取访问可能会被网站封杀所以拥有多个IP是很重要的。这也是“分布式”和“多线程”对于爬虫来说的区别。Nutch的设计就是可以支持增量的爬取，可以判断网页是否在一段时间内被抓取过，如果抓取了就不会重复抓取。这个能力对于很多种数据来说是很重要的，如新闻等。 Nutch有很多可以定制的插件、机制、策略，并且支持多种持久化存储方式，如Hbase，Mysql等

#### 2.Scrapy是基于python开发的轻量级开源爬虫框架，是基于单机的批处理任务，其抓取模型成熟稳定，模块间相互独立。虽然其没有分布式，也不支持增量。但Scrapy提供了可定制的能力，如爬取的机制、过滤url和继续爬取的策略等，并且Scrapy提供了方便的扩展接口，通过和其他第三方库结合，通过消息队列等RPC通信方式能实现分布式爬虫和增量爬虫。

**3 PySpider是binux做的一个爬虫架构的开源化实现。主要的功能需求是：**

**抓取、更新调度多站点的特定的页面，需要对页面进行结构化信息提取，灵活可扩展，稳定可监控。**

**面对结构迥异的各种网站，单一的抓取模式并不一定能满足，灵活的抓取控制是必须的。为了达到这个目的，单纯的配置文件往往不够灵活，于是，通过脚本去控制抓取是最后的选择。PySpider框架提供了去重调度，队列，抓取，异常处理，监控等丰富功能给抓取脚本，并保证灵活性。为之称道的事，PySpider提供了web的编辑调试环境，以及web任务监控，使开发者更加方便快捷的开发部署其爬虫程序。**

由于是在远程服务器上搭建开发环境，不具备可视化的web环境，兼顾Python快速调试开发的特性，本文选取Scrapy框架作为爬取电商网站商品信息的工具。

### 3.3.2 Scrapy开源爬虫框架原理介绍

Scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架。 可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。  
其最初是为了web页面抓取所设计的， 也可以应用在获取API所返回的数据(例如 Amazon Associates Web Services ) 或者通用的网络爬虫。Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。Scrapy 使用了 Twisted 异步网络库来处理网络通讯。整体架构大致如下：



Scrapy主要包括了以下组件：

1. 引擎(Scrapy): 用来处理整个系统的数据流处理, 触发事务(框架核心)
2. 调度器(Scheduler): 用来接受引擎发过来的请求, 压入队列中, 并在引擎再次请求的时候返回. 可以想像成一个URL（抓取网页的网址或者说是链接）的优先队列, 由它来决定下一个要抓取的网址是什么, 同时去除重复的网址
3. 下载器(Downloader): 用于下载网页内容, 并将网页内容返回给蜘蛛(Scrapy下载器是建立在twisted这个高效的异步模型上的)
4. 爬虫(Spiders): 爬虫是主要干活的, 用于从特定的网页中提取自己需要的信息, 即所谓的实体(Item)。用户也可以从中提取出链接,让Scrapy继续抓取下一个页面
5. 项目管道(Pipeline): 负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，将被发送到项目管道，并经过几个特定的次序处理数据。
6. 下载器中间件(Downloader Middlewares): 位于Scrapy引擎和下载器之间的框架，主要是处理Scrapy引擎与下载器之间的请求及响应。
7. 爬虫中间件(Spider Middlewares): 介于Scrapy引擎和爬虫之间的框架，主要工作是处理蜘蛛的响应输入和请求输出。
8. 调度中间件(Scheduler Middewares): 介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。

Scrapy运行流程大概如下：

* A) 首先，引擎从调度器中取出一个链接(URL)用于接下来的抓取
* B) 引擎把URL封装成一个请求(Request)传给下载器，下载器把资源下载下来，并封装成应答包(Response).
* C) 引擎把URL封装成一个请求(Request)传给下载器，下载器把资源下载下来，并封装成应答包(Response)
* D) 若是解析出实体（Item）,则交给实体管道进行进一步的处理。
* E) 若是解析出的是链接（URL）,则把URL交给Scheduler等待抓取

### 3.3.3基于Scrapy和Redis的电商网站分布式爬虫设计

Scrapy新建京东商品爬虫工程,安装配置好Scrapy后，命令行输入Scrapy startproject Jdspider，Scrapy将在当前目录下建立起工程，其工程目录组织结构为下图所示：

//插入截图

分别在item.py，pipeline.py及JDSpider目录下实现自定义的JDItem类，JDpipeline类以及JDSpider类，其分别继承自Scrapy框架实现的Item，pipeline以及Spider类。

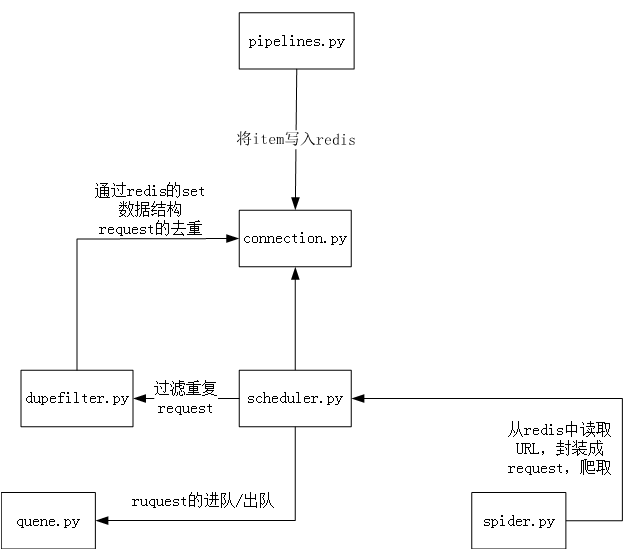
可以看出，Scrapy提供了功能清晰的底层类，开发者可以快捷的定制自己的爬虫。

Redis是一个key-value存储系统，速度很快因为将数据存放在内存，在必要时可以转到硬盘。支持数据类型有string,lists,sets,zsets。这些数据类型都支持push/pop,add/remove以及取交集并集差集等操作，对这些操作都是原子性的，redis还支持各种不同的排序能力。redis本身属于一个数据库类型的系统，不过在分布式中由于redis其队列性非常适用，逐步被开发成分布式的基石组件。

对于已有的scrapy程序，对其扩展成分布式程序还是比较容易的。关键步骤如下：

1. 利用一台高性能服务器，用于redis队列的维护以及数据的存储。
2. 扩展scrapy程序，让其通过服务器的redis来获取start\_urls，并改写pipeline里数据存储部分，把存储地址改为服务器地址。
3. 在服务器上写一些生成url的脚本，并定期执行。

利用scrapy和redis本文实现一个可以在多个主机上同时运行的分布式爬虫。其关键组件如下图：



### 3.3.4 电商网站采集数据处理及持久化存储设计

从上小节中，爬虫将电商网站商品页面获取解析后，提取商品的关键信息，如下表所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 商品ID | 名称 | 价格 | 一级类别 | 二级类别 | 三级类型 | 评论总数 | 好评度 |

通过Xpath和正则表达式将上述信息从Html页面中解析出后，需将其持久化到本地，通常持久化存储的方式包括以下几种：

1. 文件存储，如CSV，Json格式文本
2. 关系型数据库存储，如Mysql，Oracle
3. 非关系型数据库存储，如MongoDB，Hbase

一般来讲，文件存储适用于数据量不大而且不关注与并发访问的应用场景，

关系型数据库存储适用于数据结构固定清晰并且数据之间存在关系的场景，非关系型数据库应用于非结构型、海量数据的分布式存储，易横向扩展以及支持高并发读写。

在本文中，根据电商网站海量商品信息的特点，选用了MongoDB作为持久化存储的工具。而Scrapy爬虫框架也提供了对MongoDB的访问API接口。

#### 3.3.5 爬虫实现方案的性能对比

测试环境如下：

CPU:

## 3.4 反爬虫策略介绍和相应解决方案设计

目前来讲，随着Web的普及，网络爬虫已经成了常用的网络技术，对于一个内容型驱动的知名垂直网站来说，受到网络爬虫的光顾是不可避免的。一些智能的搜索引擎爬虫的爬取频率比较合理，对网站资源消耗比较少，但是很多糟糕的网络爬虫，对网页爬取能力很差，经常并发几十上百个请求循环重复抓取，这种爬虫对中小型网站往往是毁灭性打击，特别是一些缺乏爬虫编写经验的程序员写出来的爬虫破坏力极强，造成的网站访问压力会非常大，会导致网站访问速度缓慢，甚至无法访问。一般网站从三个方面反爬虫：用户请求的Headers，用户行为，网站目录和数据加载方式。前两种比较容易遇到，大多数网站都从这些角度来反爬虫。第三种一些应用ajax的网站会采用，这样增大了爬取的难度。

### 3.4.1 基于Headers的反爬虫策略

从用户请求的Headers反爬虫是最常见的反爬虫策略。很多网站都会对Headers的User-Agent进行检测，还有一部分网站会对Referer进行检测（一些资源网站的防盗链就是检测Referer）。

如果遇到了这类反爬虫机制，可以直接在爬虫中添加Headers，将浏览器的User-Agent复制到爬虫的Headers中；或者将Referer值修改为目标网站域名[评论：往往容易被忽略，通过对请求的抓包分析，确定referer，在程序中模拟访问请求头中添加]。对于检测Headers的反爬虫，在爬虫中修改或者添加Headers就能很好的绕过。

### 3.4.2 基于用户行为的反爬虫策略

有一部分网站是通过检测用户行为，例如同一IP短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。

大多数网站都是前一种情况，对于这种情况，使用IP代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理ip，检测后全部保存起来。这样的代理ip爬虫经常会用到，最好自己准备一个。有了大量代理ip后可以每请求几次更换一个ip，这在requests或者urllib2中很容易做到，这样就能很容易的绕过第一种反爬虫。

对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

### 3.4.3动态页面的反爬虫策略

上述的几种情况大多都是出现在静态页面，还有一部分网站，我们需要爬取的数据是通过ajax请求得到，或者通过Java生成的。首先用Firebug或者Chrome自带的网络分析工具对网络请求进行分析。如果能够找到ajax请求，也能分析出具体的参数和响应的具体含义，我们就能采用上面的方法，直接利用requests或者urllib2模拟ajax请求，对响应的json进行分析得到需要的数据。

能够直接模拟ajax请求获取数据固然是极好的，但是有些网站把ajax请求的所有参数全部加密了。根本没办法构造自己所需要的数据的请求。我这几天爬的那个网站就是这样，除了加密ajax参数，它还把一些基本的功能都封装了，全部都是在调用自己的接口，而接口参数都是加密的。遇到这样的网站，就不能用上面的方法了，我用的是selenium + phantomJS框架，调用浏览器内核，并利用phantomJS执行js来模拟人为操作以及触发页面中的js脚本。从填写表单到点击按钮再到滚动页面，全部都可以模拟，不考虑具体的请求和响应过程，只是完完整整的把人浏览页面获取数据的过程模拟一遍。

## 3.5 本章小结

本章是围绕电商网站基础数据采集的过程进行分析。实现一个Web请求并解析网页是网络爬虫的基础技术，但面对复杂的Web结构，设计一个性能高效的网络爬虫也是巨大的挑战。本章从单线程的网络爬虫技术出发，通过多线程，线程池的技术手段，达到优化网络爬虫性能的目的。

另外，通过介绍几种优秀的网络爬虫框架，分析其实现原理和技术架构，结合电商网站商品信息采集的应用场景，应用Scrapy框架完成对京东网站商品页面的获取和解析，并将所需商品信息的解析结果持久化到本地。

最后，介绍了常见的反爬虫策略，以及对于反爬虫策略如何采用相应的技术手段突破其限制。