如何获取网页数据

爬虫系统分类

关键技术分析

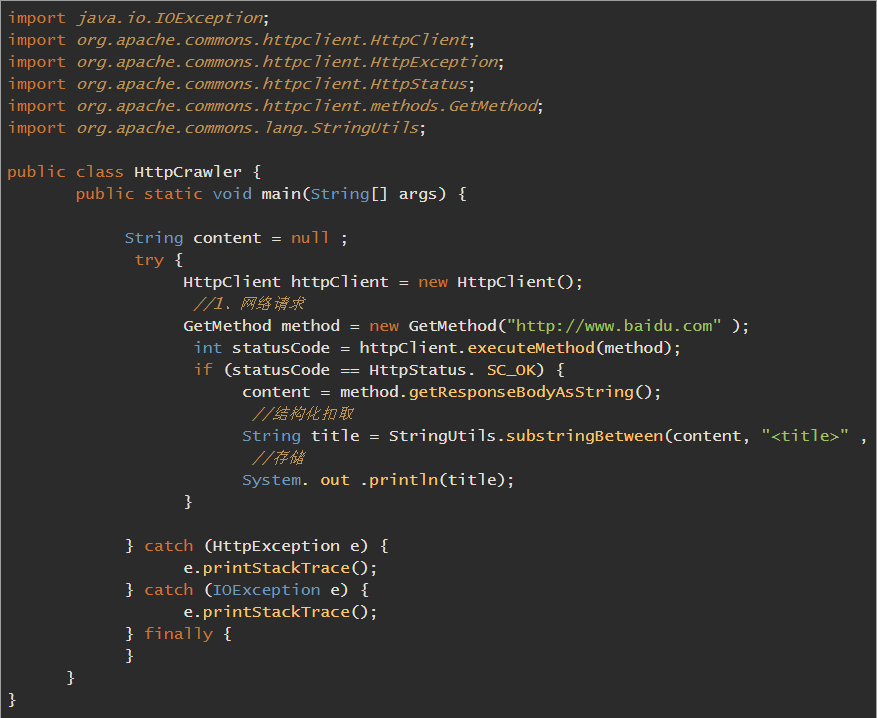
常用的开源爬虫系统

nutch的介绍

我们来看一下作为人是怎么获取网页数据的呢？

1、打开浏览器，输入网址url访问页面内容。  
2、复制页面内容的标题、作者、内容。  
3、存储到文本文件或者excel。

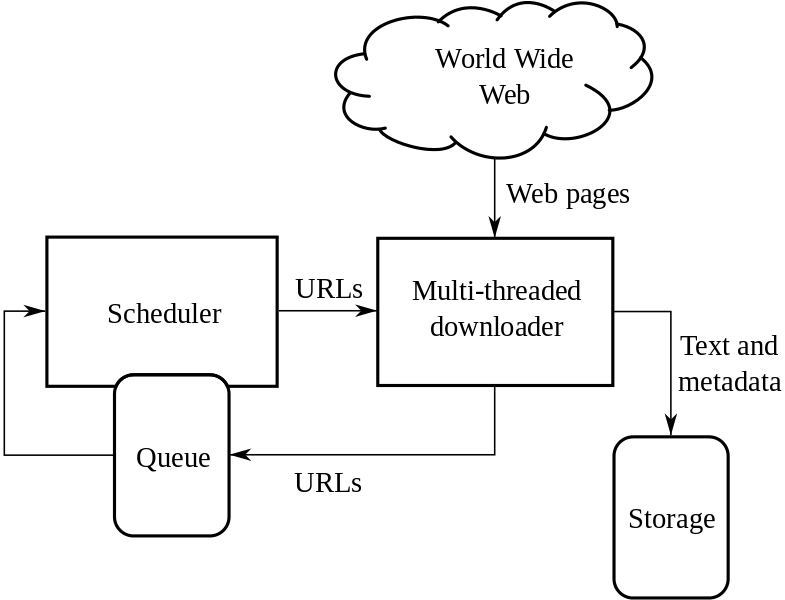
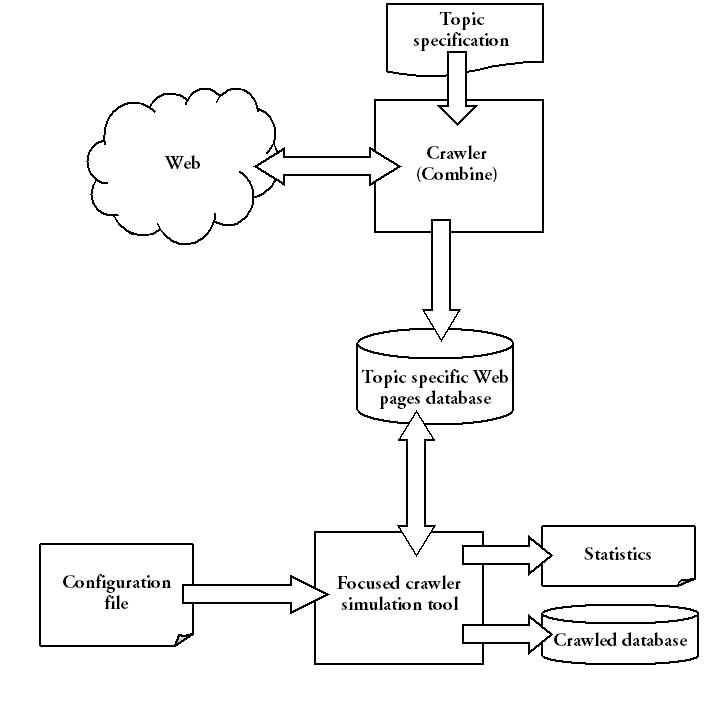
从技术角度来说整个过程主要为 网络访问、扣取结构化数据、存储。我们看一下用java程序如何来实现这一过程。



对于通用型的爬虫，例如搜索引擎蜘蛛，需要指对互联网大部分网页无差别进行抓取。这时候难点就在于页面下载和链接管理上–如果要高效的抓取更多页面，就必须进行更快的下载；同时随着链接数量的增多，需要考虑如果对大规模的链接进行去重和调度，就成了一个很大的问题。一般这些问题都会在大公司有专门的团队去解决，比如这里有一篇来自淘宝的[快速构建实时抓取集群](http://www.searchtb.com/2011/07/%E5%BF%AB%E9%80%9F%E6%9E%84%E5%BB%BA%E5%AE%9E%E6%97%B6%E6%8A%93%E5%8F%96%E9%9B%86%E7%BE%A4.html?spm=0.0.0.0.hHzGxv)。对Java来说，如果你要研究通用爬虫，那么可以看一下[**heritrix**](http://www.oschina.net/p/heritrix)或者[**nutch**](http://www.oschina.net/p/nutch)。

聚焦爬虫，又称主题爬虫

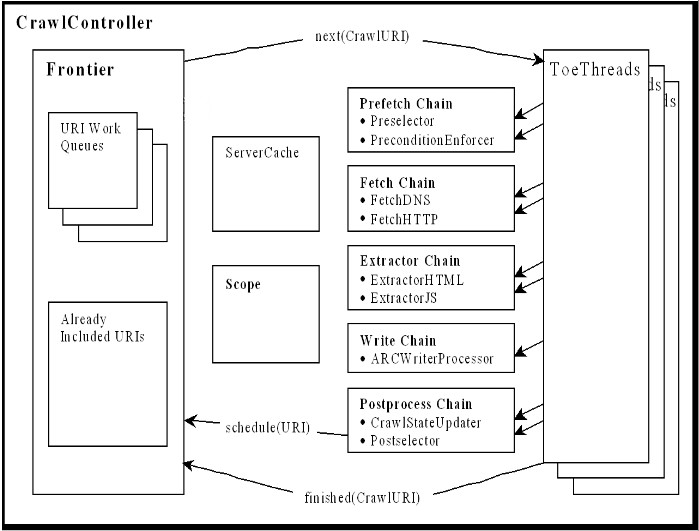
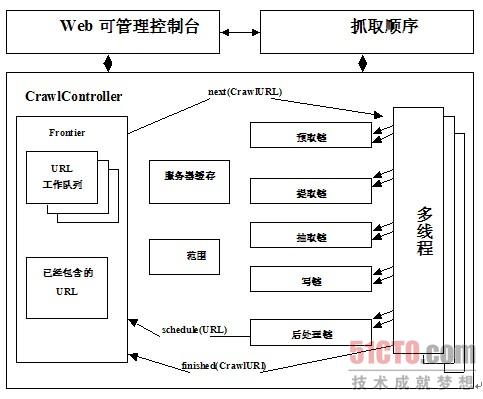
而垂直类型的爬虫要解决的问题则不一样，比如想要爬取一些网站的新闻、博客信息，一般抓取数量要求不是很大，难点则在于如何高效的定制一个爬虫，可以精确的抽取出网页的内容，并保存成结构化的数据。这方面需求很多，

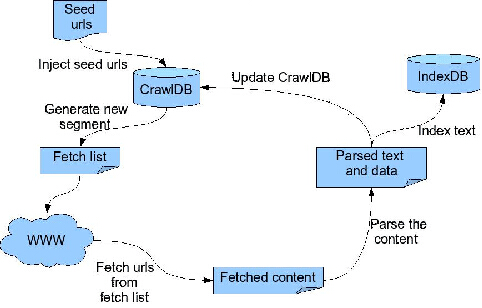
 

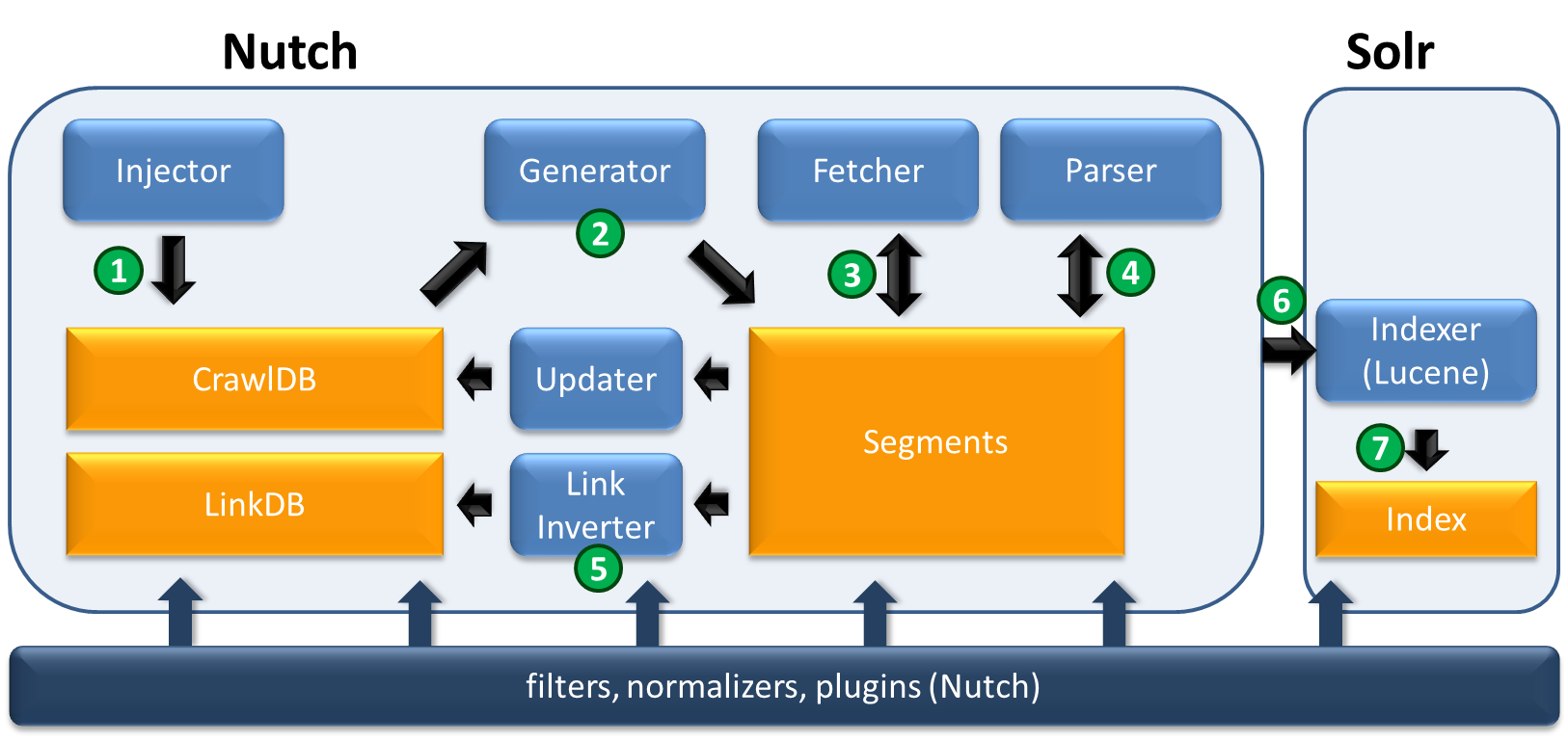
主要目的不同。 Heritrix 是个 “archival crawler” — 用来获取完整的、精确的、站点内容的深度复制。包括获取图像以及其他非文本内容。**抓取并存储**相关的内容。对内容来者不拒，不对页面进行内容上的修改。

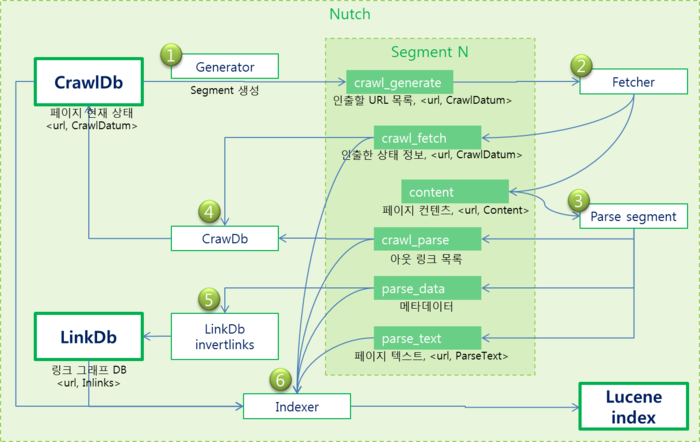
Nutch支持分布式抓取，并有Hadoop支持，可以进行多机分布抓取，存储和索引。另外很吸引人的一点在于，它提供了一种插件框架，使得其对各种网页内容的解析、各种数据的采集、查询、集群、过滤等功能能够方便的进行扩展，正是由于有此框架，使得 Nutch 的插件开发非常容易，第三方的插件也层出不穷，极大的增强了 Nutch 的功能和声誉。

这两个我都用过，都是通用搜索引擎爬虫，Heritrix的可定制性强点，有web管理界面使用起来也蛮方便的，可配置的参数很多，但代码复杂度也蛮高的，二次开发需要读懂他的设计思想。Nutch使用起来就蛮复杂的，一堆的设置，基于命令行的，不利于二次开发。我做的是垂直搜索与数据抓取，这些都不适合，最终还是选择自己开发了一个爬虫，一般来说，我们最终需要的都不是原始的HTML页面。我们需要对爬到的页面进行分析，将感兴趣的数据转化成结构化的数据，并存储下来，这才是爬虫的核心价值。







说到底就是模拟人的正常行为操作

**一定时间内单IP访问次数**，没有哪个人会在一段持续时间内过快访问，除非是随意的点着玩，持续时间也不会太长。可以采用大量不规则代理IP来模拟。

**一定时间内单账号访问次数**，这个同上，正常人不会这么操作。可以采用大量行为正常的账号，行为正常就是普通人怎么在社交网站上操作，如果一个人一天24小时都在访问一个数据接口那就有可能是机器人了。

**1、有些网站利用js生成网页内容**，直接查看源代码是一堆js。 可以使用mozilla、webkit等可以解析浏览器的工具包解析js、ajax，不过速度会有点慢。  
**2、网页里有一些css隐藏的文字**。使用工具包把css隐藏文字去掉。  
**3、图片flash信息**。 如果是图片中文字识别，这个比较好处理，能够使用ocr识别文字就行，如果是flash目前只能存储整个url。  
**4、一个网页有多个网页结构**。如果只有一套抓取规则肯定不行的，需要多个规则配合抓取。  
**5、html不完整**，不完整就不能按照正常模式去扣取。这个时候用xpath肯定解析不了，我们可以先用htmlcleaner清洗网页后再解析。

我们现在大部分爬虫系统，很多都需要自己设定正则表达式，或者使用htmlparser、jsoup等软件来硬编码解决结构化抓取的问题。

javascript引擎大家比较熟悉的应该是当前比较火的Chrome V8引擎。以快速闻名。v8基于c++开发。这里就不多阐述了。另外一个是 基于java的Rhino引擎，

**Downloader-页面下载**

对于一些Javascript动态加载的网页，仅仅使用http模拟下载工具，并不能取到页面的内容。这方面的思路有两种：一种是抽丝剥茧，分析js的逻辑，再用爬虫去重现它(比如在网页中提取关键数据，再用这些数据去构造Ajax请求，最后直接从响应体获取想要的数据)；   
另一种就是：内置一个浏览器，直接获取最后加载完的页面。这方面，js可以使用**PhantomJS**，它内部集成了webkit。而Java可以使用**Selenium**，这是一个非常强大的浏览器模拟工具。

### PageProcessor-页面分析及链接抽取

这里说的页面分析主要指HTML页面的分析。页面分析可以说是垂直爬虫最复杂的一部分，

HTML分析是一个比较复杂的工作，Java世界主要有几款比较方便的分析工具：

#### Jsoup

Jsoup是一个集强大和便利于一体的HTML解析工具。它方便的地方是，可以用于支持用jquery中css selector的方式选取元素，这对于熟悉js的开发者来说基本没有学习成本。

String content = "blabla";

Document doc = JSoup.parse(content);

Elements links = doc.select("a[href]");

Jsoup还支持白名单过滤机制，对于网站防止XSS攻击也是很好的。

#### HtmlParser

HtmlParser的功能比较完备，也挺灵活，但谈不上方便。这个项目很久没有维护了，最新版本是2.1。HtmlParser的核心元素是Node，对应一个HTML标签，支持getChildren()等树状遍历方式。HtmlParser另外一个核心元素是NodeFilter，通过实现NodeFilter接口，可以对页面元素进行筛选。这里有一篇HtmlParser的使用文章：[使用 HttpClient 和 HtmlParser 实现简易爬虫](https://www.ibm.com/developerworks/cn/opensource/os-cn-crawler/)。

#### Apache tika

tika是专为抽取而生的工具，还支持PDF、Zip甚至是Java Class。使用tika分析HTML，需要自己定义一个抽取内容的Handler并继承org.xml.sax.helpers.DefaultHandler，解析方式就是xml标准的方式。crawler4j中就使用了tika作为解析工具。SAX这种流式的解析方式对于分析大文件很有用，我个人倒是认为对于解析html意义不是很大。

InputStream inputStream = null;

HtmlParser htmlParser = new HtmlParser();

htmlParser.parse(new ByteArrayInputStream(page.getContentData()),

contentHandler, metadata, new ParseContext());

#### HtmlCleaner与XPath

HtmlCleaner最大的优点是：支持XPath的方式选取元素。XPath是一门在XML中查找信息的语言，也可以用于抽取HTML元素。XPath与CSS Selector大部分功能都是重合的，但是CSS Selector专门针对HTML，写法更简洁，而XPath则是通用的标准，可以精确到属性值。XPath有一定的学习成本，但是对经常需要编写爬虫的人来说，这点投入绝对是值得的。

学习XPath可以参考w3school的[XPath 教程](http://www.w3school.com.cn/xpath/)。下面是使用HtmlCleaner和xpath进行抽取的一段代码：

HtmlCleaner htmlCleaner = new HtmlCleaner();

TagNode tagNode = htmlCleaner.clean(text);

Object[] objects = tagNode.evaluateXPath("xpathStr");

#### 几个工具的对比

在这里评价这些工具的主要标准是“方便”。就拿抽取页面所有链接这一基本任务来说，几种代码分别如下：

XPath:

tagNode.evaluateXPath("//a/@href")

CSS Selector:

*//使用类似js的实现*

$("a[href]").attr("href")

HtmlParser：

Parser p = new Parser(value);

NodeFilter aFilter = new TagNameFilter("a");

NodeList nodes = p.extractAllNodesThatMatch(aFilter);

for (int i = 0; i < nodes.size(); i++) {

Node eachNode = nodes.elementAt(i);

if (eachNode instanceof LinkTag) {

LinkTag linkTag = (LinkTag) eachNode;

System.out.println(linkTag.extractLink());

}

}

XPath是最简单的，可以精确选取到href属性值；而CSS Selector则次之，可以选取到HTML标签，属性值需要调用函数去获取；而HtmlParser和SAX则需要手动写程序去处理标签了，比较麻烦。

