用 Nutch 构建垂直搜索引擎的方案

余栋柱　黄　讴

(广州市烟草专卖局　510310　广东省电信规划设计院　510630)

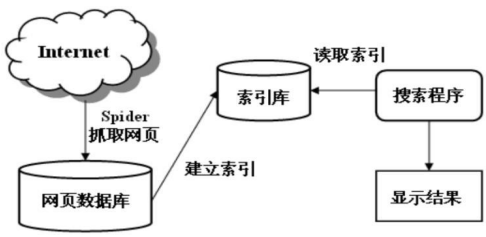
搜索引擎是互联网上检索信息的工具,能有效地提高信息定位的效率。通用搜索引擎的查询结果难以满足精确搜索的要求,于是垂直搜索成为搜索引擎的一个重要补充。垂直搜索引擎与通用搜索引擎最大的区别就是前者对网页进行了页面净化、信息抽取、页面分类、数据挖掘等深度的加工,从而为用户提供覆盖率和准确率都比较高的搜索结果。

本文提出如何在 Nutch 开源框架基础上构建一个完整的垂直搜索引擎的方案,并对于网页内容自动分类、页面内容解析和提取、中文分析器以及检索以及结果排序这些功能进行设计和解决,从而实现自定义的垂直搜索引擎。

1　垂直搜索和 Nutch 的介绍

1.1　垂直搜索的介绍

搜索引擎的提供服务的过程包括三个主要步骤:网页搜集,建立索引,提供搜索服务。从搜索引擎的工作步骤可以将搜索引擎系统分成三个组成部分:网页抓取程序,网页处理程序和网页搜索程序。下图展示了搜索引擎的体系结构:



网页抓取程序又叫 spider 或 crawler,它以一个 URL 集合作为起点,顺着这些 URL 网页的超链接,以深度优先、广度优先或启发式的方式循环地在互联网上发现信息,把网页下载到本地。网页处理程序对抓取程序获得的网页进行分析处理,这些处理包括去除 HTML 标签、存储网页的元数据、分类、信息抽取、网页排重、数据挖掘等。分析处理完后,处理程序为所有的分析数据建立倒排索引文件。在建立索引的时候,索引程序要对网页的文本进行分词。

索引文件是网页采集和网页搜索之间联系的纽带。网页搜索程序接收到用户输入的关键字后,将关键词转化成系统能够理解的精确查询,然后读取索引,得到相关网页的列表,按一定的规则排序,呈现给用户。

垂直搜索引擎在每个部分都采用了与通用搜索引擎不同的策略,因此提供跟完全不同的服务质量。垂直搜索以某方面的专业网站作为抓取目标,在抓取过程中一般采用深度优先的算法。这种抓取策略为垂直搜索提供了专业和全面的信息源。

1.2　Nutch 的介绍

Nutch 是 apache 软件基金会 Lucene 项目下的一个子项目,是一个开源的用 Java 实现的搜索引擎。它基于 Lucene 的查询引擎和索引引擎,包括了网页爬虫,网页解析器、索引器以及检索器,提供了构建搜索引擎所需的全部工具。

Nutch 从总体上可以分为爬网程序和搜索程序两个部分。爬网程序搜集页面,并把抓取回来的数据解析并做成反向索引;搜索程序则对反向索引搜索,回答用户的请求。爬网程序和搜索程序可以分别位于不同的机器上,它们可以各自独立运行,索引是它们都要用到的公共部分。

1.3　垂直搜索引擎系统架构

利用 Nutch,可以方便快速的构造出一个性能良好的搜索引擎。但是,用于构建垂直搜索引擎,仍有以下问题需要解决:

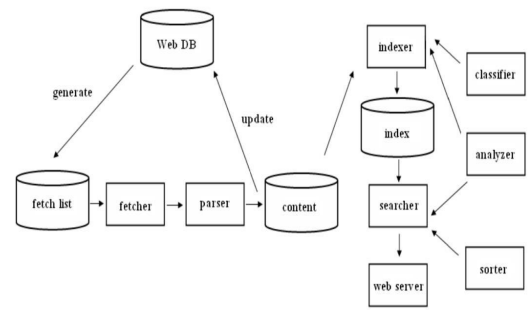
1) 网页内容自动分类。需要实现一个网页分类器,根据网页内容,按照系统制定的分类规则对网页进行分类。

2) 页面内容解析和提取。Nutch 以 Lucene 作为底层支撑,提供了良好的全文检索功能,但它对只对网页进行简单解析和处理,垂直搜索引擎注重页面的主题和内容提取,因此需要实现一个智能化较高、通用性较强的页面解析程序。

3) 中文分词。我们的垂直搜索引擎主要针对的是中文网页,中文分词是系统的一个重要组成部分,需要实现一个中文分词器。本文推荐用开源的 Paoding 中文分词开发包。它是一个使用 Java 开发的开源分词系统,可以作为互联网、企业内部网使用的中文搜索引擎分词组件。它包含了一个词库和一个分词器,提供了良好的分词功能。

4) 检索结果进行排序。实现自定义排序功能。

针对以上问题,我们提出如下方案,方案以 Nutch 为基础,通过实现相应的插件接口,从中文分词、页面信息解析和提取、索引创建以及页面检索等方面对 Nutch 进行定制和扩展,构建一个垂直搜索引擎。下图展现了基于 Nutch 的垂直搜索引擎的系统架构。



与 Nutch 原来的架构相比,该引擎添加了页面解析(parser)、页面分类(classifier)、中文分词(analyzer)、检索排序(sorter)等关键功能模块,用于完成页面解析、分类、中文分词以及检索结果排序等功能。

下面将会根据系统框架,为每一个功能模块的实现进行探讨。

2　用 Nutch 构建垂直搜索引擎的研究

2.1　Nutch 爬网程序的运行机制

Nutch 定义了一组命令,用户只要按一定的逻辑执行命令,即可以完成网页抓取、页面解析等过程。Crawl 命令能完整地展示 Nutch 爬网程序的运行过程。Crawl 命令的工作过程如下:

1) 创建一个新的网页数据库 crawldb。

2) 以一个或多个存有 URL 列表的文本文件为种子,将其中包含的 URL 导入 crawldb 中。

3) 根据 crawldb 的信息,生成待抓取的网页列表,存放到一个数据段(segment)中。

4) 根据生成的 segment 的信息, 开始进行页面抓取和页面解析, 将页面的内容和页面上的链接及其他信息结果保存到 segment 中。

5) 将解析得到的新的 url 更新 crawldb 中。

6) 根据 segments 中的信息, 对页面链接进行倒排(invertlinks), 生成 linkdb。

7) 以 crawldb、segments 和 linkdb 中的数据为基础, 为每个 segment 创建索引。

8) 进行网页排重, 对有重复的网页只保留一个副本。

9) 对各个 segment 中的索引数据进行合并, 建立最终索引库。

Nutch 在爬网的时候可以在过程 2 和 5 之间循环调用, 每一轮抓取过程都将上一轮分析页面所得到的链接加入到 crawldb 中, 以实现增量爬网。Nutch 在爬网的过程中采用的是广度优先的爬行算法。

crawldb 保存了整个爬网过程中的所有链接及其状态。segments 中保存了 Nutch 在解析网页时产生的所有信息。

网络爬虫下载完一个网页后, 会调用解析器(Parsers) 对页面进行分析。Nutch 提供的 HTML 解析器首先提取网页中所有合法的 URL 链接, 然后解析出网页的元数据(parse- data) 和网页的文本内容(parse- text), 以供索引器进行索引。

2.2 网页分类的设计实现

垂直搜索的意义在于为用户提供聚类性较强的搜索服务, 而网页的类别一般是根据其内容来划分。因此, 根据网页内容的自动分类是垂直搜索引擎的另一核心功能。classifier 模块负责将 fetch 下来的网页进行分类, 在 contents 存储了的页面的 meta data 和文本内容。在 Classifier 中维持一个分类规则, 其中的规则可以采用 XML 的配置文件进行定义。我们可以结合分词器 Analyzer, 对每一个网页的内容进行分词, 将分词之后的页面内容与 classifier 中的分类规则进行匹配检验, 并为网页打上分类标签信息, 供索引器调用, 最终实现分类的功能。

2.3 信息抽取的设计实现

信息抽取子系统的处理过程可以分成两部分, 前一过程从原始的网页到结构化信息的抽取, 后一个过程对结构化信息存储。之所以把这两个部分分开设计, 是因为在 Nutch 中, 网页解析和网页索引是两个独立的处理过程, 它们可以用不同的 Nutch 命令执行。

我们可以把系统对网页的处理分成四个步骤: 网页规范化(toXML), 网页净化(refine), 结构化信息抽取(extract), 信息存储(index)。其中前三个处理过程是结构化信息抽取的过程, 最后一个过程是对结构化信息的存储。为了让我们的子系统可以无缝地集成到 Nutch 系统中, 我们将以上前三个处理步骤设计一个 Nutch 解析插件(parsefilter), 而将信息存模块设计成一个 Nutch 索引插件(indexfilter)。

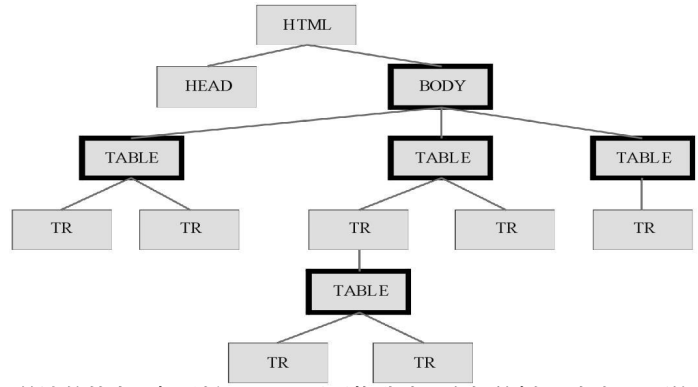
(1) 网页规范化

网页规范化是指将形式比较自由的 HTML 页面转化成结构严络的 XML 文档。将 HTML 转化为 XML 后, 即可以采用 XML 技术对页面进行处理。这里我们采用第三方的软件 NekoHTML 将 HTML 页面转化为 DOM 树型结构, 再由将 DOM 树型结构转化成标准的 XML 文件。

(2) 网页净化

一个网页一般包括两部分内容, 一部分体现了网页主题信息的正文内容, 另一部分是与主题内容无关的导航条、广告信息、版权信息以及调查表单等内容, 我们称之为“网页噪音”。网页净化模块的功能就是将网页中的噪音内容尽量去除, 以提高信息抽取的效率和准确度。

吸收多个国内中文网页净化的算法的优点并综合考虑垂直搜索搜集的目标网页的特点后, 本文提出一种新的网页净化的方法。这种



算法的基本思想是将 HTML 页面构造成一个标签树, 再根据网页的标签对网页进行分块, 然后对每一块添加语义属性, 根据语义属性的特点对标签树进行剪枝, 最后得到一个精简后的标签树, 还原成 HTML。

每一个块的内容都有其特定的特征, 我们在这里定义语义量、链接密度和内容相关度三个概念(语义参数)。

语义量是指一个块中非超链接文本包含的相关词的个数。链接密度是指一个块中链接文本包含的相关词个数与该块中所有文本包含的相关词个数的比例。链接密度越大, 该块越有可能与主题无关。内容相关度是一个大于等于 0 小于等于 1 的数, 相关度越接近 1, 说明两个块的相关程度越大。

我们对每一个块进行语义量和链接密度的计算, 对于非根块结点, 计算出该块结点与其父块结点的内容相关度, 将这些参数作为结点的属性加入到 DOM 树中, 便完成了扩展语义 DOM 树的构建。

(3) 剪枝算法

网页净化的目的是将与主题无关的内容从页面中剔除掉, 这里我们将根据块结点的语义属性对结点进行取舍操作。系统定义三个阈值: 语义量阈值 SC、链接密度阈值 LD 和内容相关度 CR。剪枝算法根据块结点的语义属性与阈值的关系对扩展的语义 DOM 树进行剪枝操作。剪技的条件如下:

1. 语义量小于阈值 SC。这说明该块的信息量很小, 与网页的主题信息无关。

2. 链接密度大于阈值 LD。这说明该块很有可能是导航或广告等链接。

3. 语义相关度小于 CR。CR 直接反应一个块和其父块的内容是否一致。父结点包含的信息量一般比子结点要大, 因此它更有可能被保留下来, 而保留下来的父结点如果链接密度小于 CR, 代表它包含了主题内容, 它的子结点如果与它的内容相关度太小, 则很有可能与主题无关。

在剪枝的操作过程中, 被剪掉的结点会对父结点的语义属性产生影响, 因此算法要从 DOM 树的最底层的块开始进行剪枝, 并将其语义影响更新到父结点。

(4) 信息抽取

净化过后的网页结构相对简单, 我们主要采用基于模板的信息抽取方法来提取结构化信息。我们制定不同的匹配规则, 综合使用正则表达式和 XPath 技术完成信息抽取。

基于模板的信息抽取方法先定位结构化数据所在的位置, 然后提取该位置的内容。模板中要定义以下几个参数

1) URL 集合。同一模板生成的网页的 URL 一般都是类似的, 我们用正则表达式来指定的一个 URL 集合。URL 是系统决

定使用哪个模板进行数据抽取的入口。

2) 一个基本 XPath 表达式。它指明了从网页文档根结点到有用信息块的绝对路径。这个 XPath 的计算结果为包含了结构化数据记录的一个或多个结点。

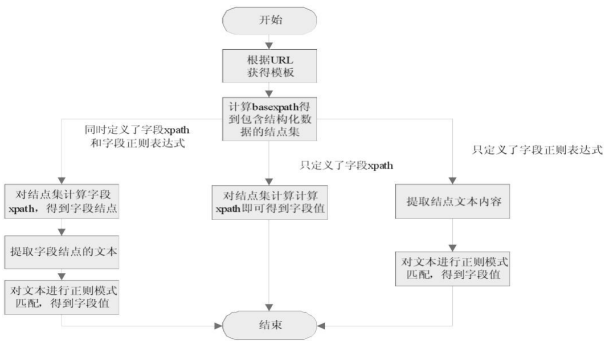
3) 多个字段 XPath 表达式。它们指明了从包含结构化数据的结点到各个字段值的相对路径。

4) 多个字段正则表达式。它们指明了字段值与包含结构化数据记录的结点的文本的关系,系统可以根据正则表达式进行字段值的提取。

由于在结点处理方面 XPath 更具优势,如果字段结点的全部内容都为结构化数据的字段值,可以直接用 XPath 提取该值。正则表达式的方法在文本处理方面有其自身的优点,我们也用它来提取结构数据的字段值。

这些参数我们将采用 XML 配置文件的形式实现到系统中。

结构化数据抽取的算法如下图所示:



XPath 技术是基于 XML 的,它把一个文档当成一个结点树,我们可以在 DOM 的基础上直接使用 XPath;正则表达式基于字符串的匹配,用它进行数据抽取的时候,它把文档当成一个字符串。我们的算法充分利用了正则表达式和 XPath 的优点。

2.4 信息存储设计实现

结构化信息可以直接存储到关系数据库中,以便进行进一步的数据挖掘和其它的智能处理。我们在 Nutch 的索引程序里添加一个插件,即可以完成存储模块的设计。我们的插件将增加对结构化数据的索引。我们把结构化数据的每一个字段作为一个字段(Field),添加到页面的索引中。

Nutch 的 Field 有三种属性,即 analyzed、indexed、stored。对每一类结构化数据的每一个字段,都要制定一个索引的规则,定义该字段的存储类别。我们用将 XML 配置文件来维护索引的存储类别。

2.5 查询器的设计实现

Nutch 的一个强大功能就是多种查询方式,如基于词条的查询、布尔查询、模糊查询。这些查询方式为构造搜索引擎提供了便利。开发者可以用不同的方式来检索查询,进而更加容易描述自己的查找要求。而这些查询类全部继承自 Query。

Query 类本身是一个抽象类,Query 类规定了它的子类的行为的同时,又给予多种查询方式和足够的自由度来实现自己。Query 其实也就是指对于需要查询的字段采用什么样的方式进行查询,如模糊查询、语义查询、短语查询、范围查询、组合查询等,还有就是 QueryParser,QueryParser 可用于创建不同的 Query,还有一个 MultiFieldQueryParser 支持对于多个字段进行同一关键字的查询,IndexSearcher 指的为需要对何种目录下的索引文件进行何种方式的分析的查询,有点像对数据库的哪种索引表进行查询并按一定方式进行记录中字段的分解查询的概念,通过 IndexSearcher 以及 Query 即可查询出需要的结果,Nutch 返回的为 Hits。通过遍历 Hits 可获取返回的结果的 Document,通过 Document 则可获取 Field 中的相关信息。

2.6 查询结果排序和分页

(1) 自定义的结果排序

Nutch 的搜索结果默认按相关度排序,它是基于内部的 Score 和 DocID,Score 又基于关键词的内部评分和做索引时的 boost。默认 Score 高的排前面,如果 Score 一样,再按索引顺序,先索引的排前面。

Score 在每次执行查找时都是不一样的,根据查找的关键字来进行确定。文档的排序是按照相关度自高向低排。我们也可以通过改变 boost 值来 Score 值,就是人为地增加某个文档的相关度,使得在搜索结果中排得靠前的位置上。可以使用 Document 类所提供的 setBoost 方法来改变 boost 值。

也可以使用 Nutch 自带的一个 Sort 类对检索结果进行排序。使用它时,只需要实例化一个 Sort 对象,并使用 Search 提供的接口来实现。而 Sort 所提供的排序功能是以 Field 为基础的,最终的排序准则,总是以某个 Field(或者多个)的值为基础,经过这样的处理,最终的排序就转变成对所有文档中同一个 Field(或者多个 Field)的值的排序了。

(2) 分页的解决

分页是 Web 搜索引擎中的一个常见问题。在得到搜索结果后,通常结果的数量众多,无法在一个页面内显示。常用的方法有:依赖于 session;多次查询;缓存 + 多次查询;缓存 + 多次查询 + 数据库。当一些查询用户比较多的时候,本文推荐采用缓存 + 多次查询 + 数据库的方式。

这种方式就是在缓存和多次查询的基础上,用数据库来缓解一部分访问压力。其实在检索时,有很多需要展示给用户的数据是可以存放于数据库中的,而不必完整地放在索引中。用户检索时,只须对有限的关键字进行检索,然后从索引中获取数据库的一个 ID 号,进而转用数据库来读取大量数据,传回页面。

这种方式将原先不得不存在缓存中的数据转移到了数据库中,在缓存中只需要存储少量数据,真正的数据可以根据 ID 值从数据库中读取。这样进一步减小了每个检索结果在缓存中的数据量,可以缓存更多的检索结果,加快了检索的速度。

3 总结

本论文在介绍了搜索引擎的基本原理和 Nutch 框架,提出了在 Nutch 上构建垂直搜索引擎的方案。分别对网页内容自动分类、页面内容解析和提取、中文分析器、多功能查询器以及检索与结果排序这些功能进行讨论,并提出解决方案,最终实现自定义的垂直搜索引擎。

【参考文献】

[1] 王琦,唐世渭,杨冬青,王腾蛟. 基于 DOM 的网页主体自动提取. 计算机研究与发展, 2004, 42(10): 1786- 1792.

[2] 张志刚,陈 静,李晓明. 一种 HTML 网页净化方法. 情报学报, 2004 年 8 月: 387- 393.

[3] 陈淑珍,卢昌荆. WEB 文本挖掘的中文分词系统的设计与实现. 三明学院学报. 2005, 2: 197- 200.

[4] 李晓明,闫宏飞,王继民. 搜索引擎——原理、技术与系统. 科学出版社, 2005 年 4 月 1 日.