《大数据分析方法》

（2021年秋季学期）

翟祥
北京林业大学

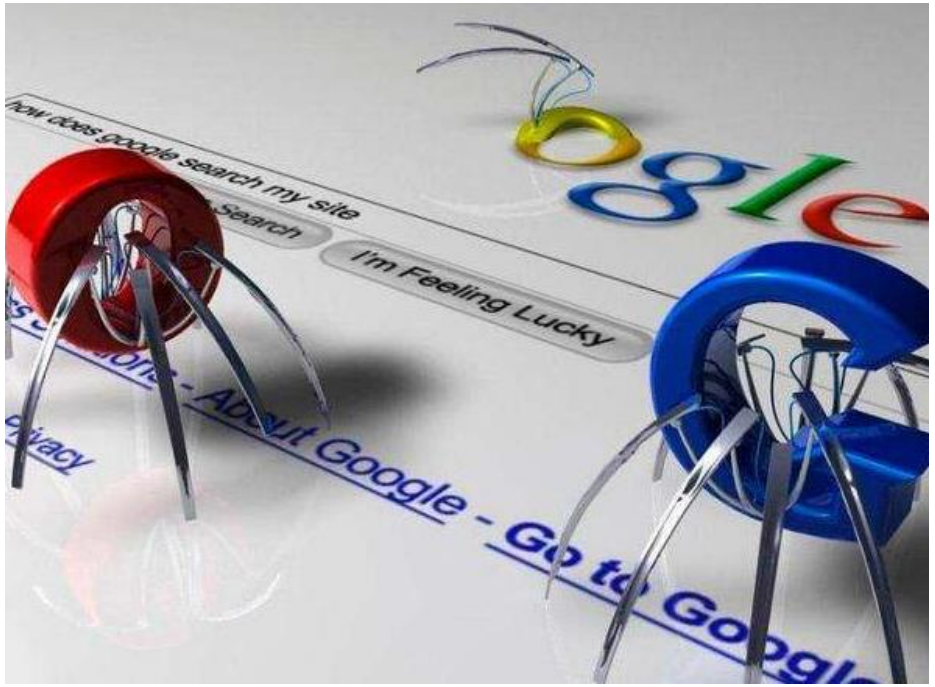
E-mail: zhaixbh@126.com


第4章
互联网大数据处理

第4章　互联网大数据处理

4.1.1概述

互联网信息自动抓取，最常见且有效的方式是使用网络爬虫。

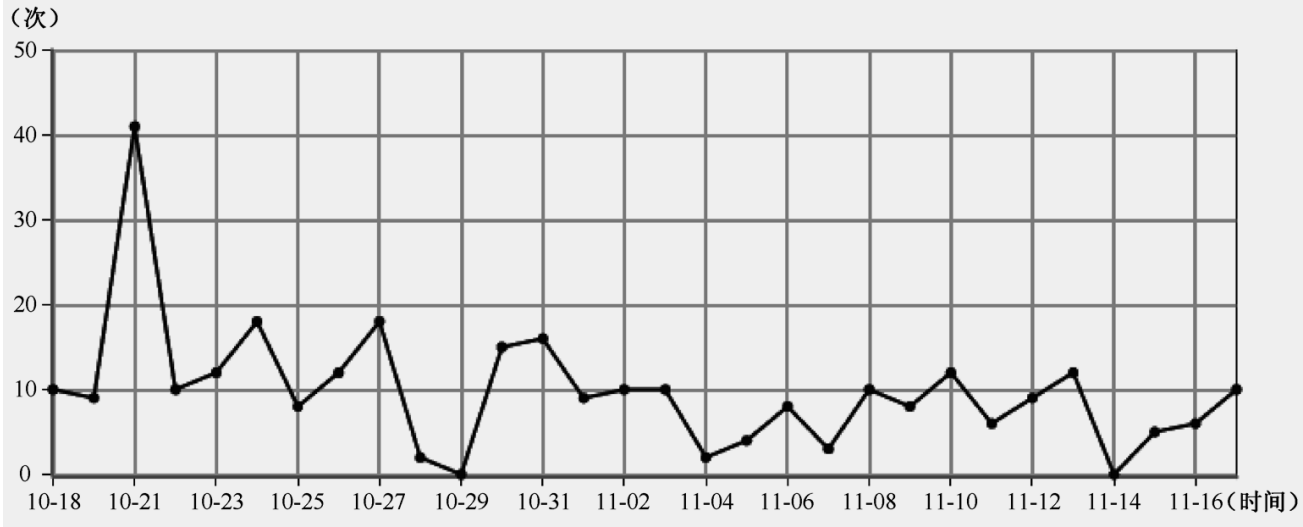


爬虫可以被分为两类：
一类叫作“通用爬虫”；
另一类叫作“聚焦爬虫”。

目前成熟的网络爬虫有很多，其中不乏Googlebot、百度蜘蛛这样的广分布式多服务器多线程的商业爬虫和GNU Wget、Apache Nutch这样的灵活方便的开源爬虫搜索引擎。

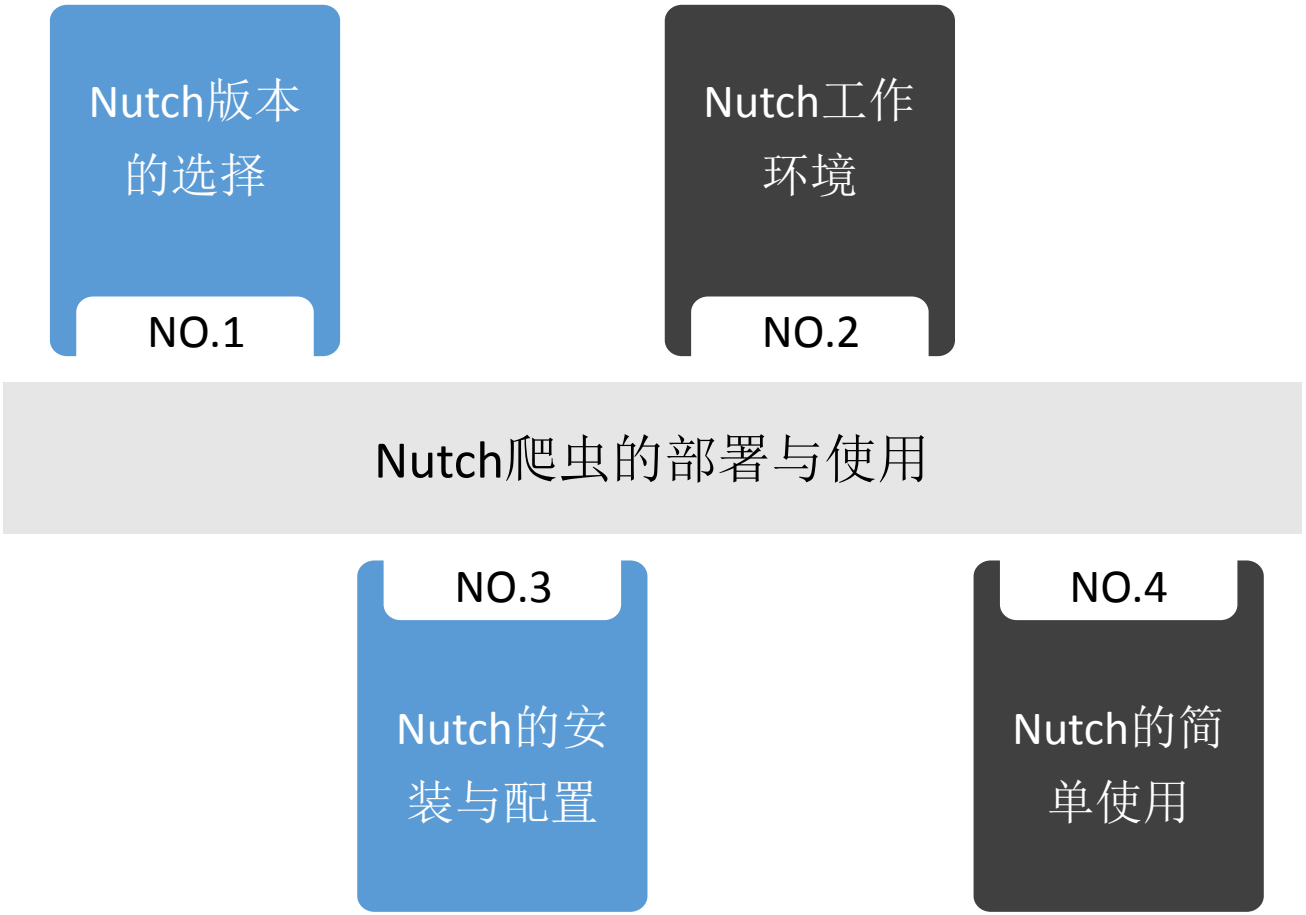
4.1.1概述

目前成熟的网络爬虫有很多，其中不乏Googlebot、百度蜘蛛这样的广分布式多服务器多线程的商业爬虫和GNU Wget、Apache Nutch这样的灵活方便的开源爬虫（爬虫搜索引擎）。



百度蜘蛛对某网站的抓取频率

4.1.2Nutch爬虫


Nutch版本的选择
NO.1
Nutch工作环境
NO.2
Nutch爬虫的部署与使用
NO.3
Nutch的安装与配置
NO.4
Nutch的简单使用

4.1.2Nutch爬虫

Nutch版本的选择	Nutch1.x是基于Hadoop集成环境的，Nucth的数据是存储在HDFS上的。Nutch2.x是基于Apach Gora的，Nutch可以访问HBase、Cassandra、MySQL等，所以，在编译Nutch之前，需要先安装HBase，另外Nutch的编译需要ant命令，所以，在编译Nutch之前还要安装Ant。

4.1.2Nutch爬虫

Nutch工作环境：

（1）Nutch仅支持在Linux系统下使用，本书使用的是Ubuntu 14.04.3 LTS，若要在Windows下使用Nutch，需要安装模拟Linux操作系统的软件Cygwin。

（2）JDK：本书使用的是jdk-8u51-linux-x64.tar.gz。

（3）HBase：可从网上下载最新版。

（4）Ant：本书使用的是apache-ant-1.9.6-bin.tar.gz。

（5）Nutch-2.2.1：可在Nutch官方网站下载最新版本的Nutch。

（6）Tomcat：本书使用的是apache-tomcat-8.0.24.tar.gz。

4.1.2Nutch爬虫

Nutch的安装与配置应该包括下面5个部分：

1. JDK的安装与配置
2. 下载并解压HBase
3. Ant的安装与配置
4. Nutch的安装与配置
5. 将Nutch和Solr集成在一起

- 4.1.2Nutch爬虫

Nutch的简单使用

一站式抓取

进入apache-nutch-2.2.1/runtime/local目录查看一站式抓取命令。

分布式抓取

可以分为2步：Nutch数据文件夹组成和生成抓取列表。

4.1.3案例：招聘网站信息抓取

Nutch查询界面

考虑如下场景：现在需要通过调查全国所有公司的规模和分布情况，来评估每个省份的经济实力。我们要做的第一步就是数据的收集工作。可以通过编写爬虫程序，自动进行数据收集工作，特别是从招聘网站上的公司介绍页面获取数据。

4.1.3案例：招聘网站信息抓取

1.采用聚焦爬虫

2.生成“种子”

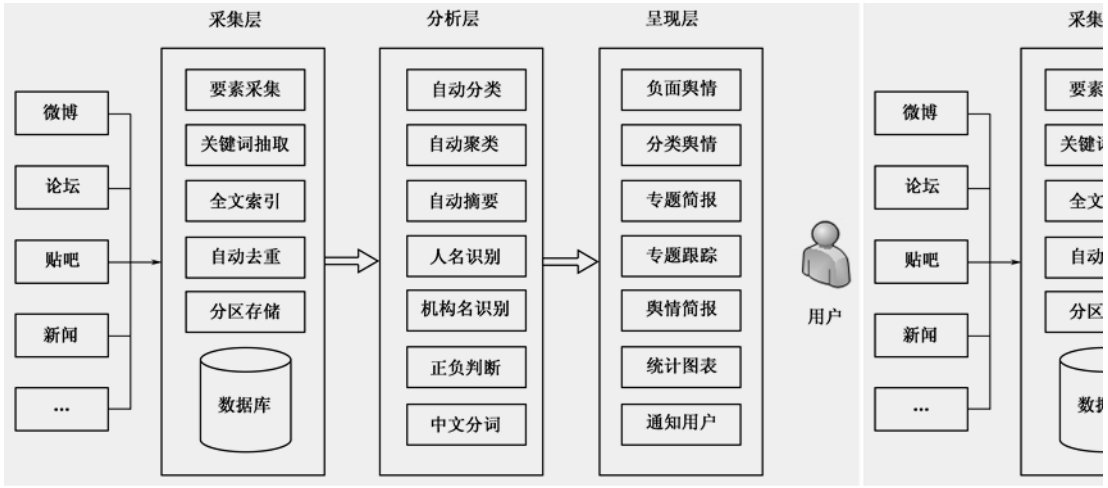
3.依次打开每一个URL，得到页面HTML

4.对HTML进行解析，提取需要的信息

5.使用多线程

使用Python实现简单的聚焦爬虫来完成这项任务

4.1.4案例：舆情信息汇聚



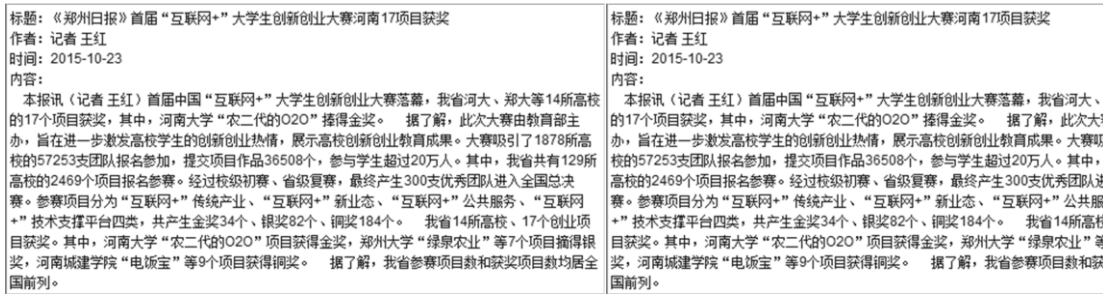
舆情监控系统架构

通常情况下，网络舆情监控系统由采集层（舆情采集模块）、分析层和呈现层（分析浏览模块）实现。

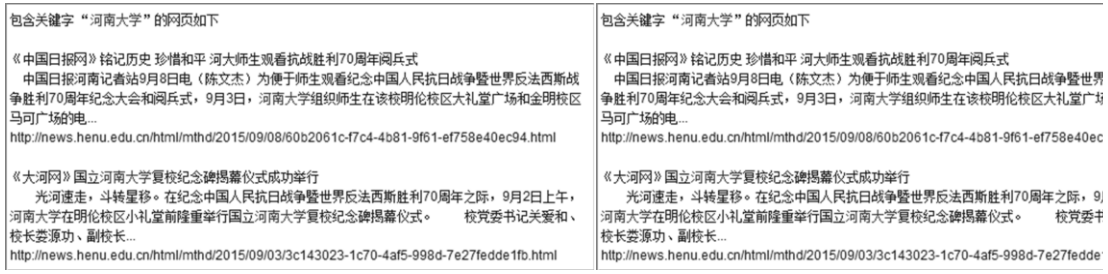
可通过网络信息自动抓取等技术手段，便捷、高效地获取与自己相关的网络舆情，不仅信息保真，而且覆盖全面。通过网络舆情监控系统最终形成专题简报、专题追踪、舆情简报等，为全面掌握网络舆情动态，正确引导舆情动向，提供了可靠、有力的数据分析依据。

4.1.4案例：舆情信息汇聚

抓取河南大学新闻网新闻主题



河南大学新闻网页关键信息提取



河南大学新闻网页关键字检索

第4章　互联网大数据处理

4.2.1概述

定义

文本分词是将字符串文本划分为有意义的单位的过程，如词语、句子或主题。

中文分词也叫作切分，是将中文文本分割成若干个独立、有意义的基本单位的过程。

分词算法基本的工作原理是根据输入的字符串文本进行分词处理、过滤处理，输出分词后的结果，包括英文单词、中文单词及数字串等一系列切分好的字符串。

分词原理图

4.2.1概述

现有的中文分词算法可以分为以下3类：

1 基于字符串匹配的分词方法

它是将待处理的中文字符串与一个“尽可能全面”的词典中的词条按照一定的规则进行匹配，若某字符串存在于词典中，则认为该字符串匹配成功。

2 基于统计的分词方法

由于词是特定的字组合方式，那么在上下文中，相邻的单字共同出现的频率越高，则在该种字组合方式下就越有可能是构成了一个词。

3 基于理解的分词方法

该方法通过语义信息和语句信息来解决歧义分词问题，并且在分词的同时进行语义和句法分析。

4.2.1概述

各种分词方法的优劣对比表

分词方法	基于字符串	基于理解	基于统计
歧义识别	差	强	强
新词识别	差	强	强
词库	需要	不需要	不需要
语料库	不需要	不需要	需要
规则库	不需要	需要	不需要
算法复杂性	容易	很难	一般
技术成熟度	成熟	不成熟	成熟
实施难度	容易	很难	一般
分词准确度	一般	准确	较准
分词速度	快	慢	一般

4.2.2MMSEG分词工具

MMSEG分词算法中包含了4种符合汉语语言中基本的成词习惯的歧义消解规则。

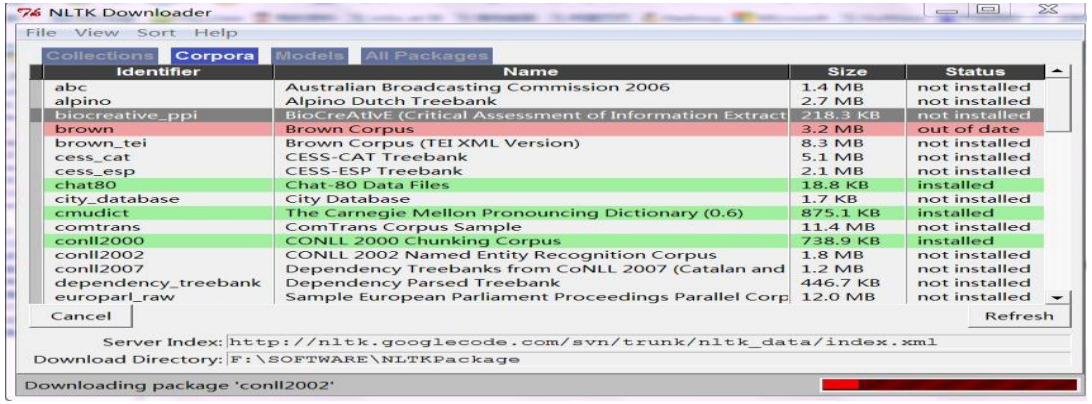
MMSEG分词算法中有两个重要的概念：Chunk和规则（Rule）。其中，一个Chunk就是一段字符串文本的一种分割方式，包括根据上下文分出的一组词及各个词对应的4个属性。规则的目的是过滤掉不符合特定要求的Chunk。为便于理解，我们可以将规则看做过滤器。

4.2.2MMSEG分词工具

Chunk中各属性及其含义

属性	含义
长度（Length）	Chunk中各个词的长度之和
平均长度（Average Length）	长度/词数
标准差的平方（Variance）	标准差的平方
自由语素度（Degree of Morphemic Freedom）	各单字词词频的对数之和

4.2.3斯坦福NLTK分词工具



有些文本的形成和变化过程与时间是紧密相关的，因此，如何将动态变化的文本中时间相关的模式与规律进行可视化展示，是文本可视化的重要内容。引入时间轴是一类主要方法，常见的技术以河流图居多。河流图按照其展示的内容可以划分为主题河流图、文本河流图及事件河流图等。

第4章　互联网大数据处理

4.3.1倒排索引原理

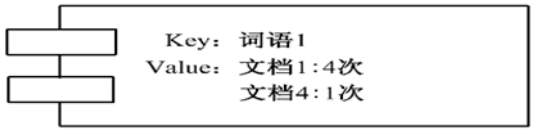
词语和文档的关系

如果使用一个矩阵来描述词语和文档之间的关系，不难得出如下“矩阵”。其中，每一列代表一个文档，每一行代表一个词语，每一个单元格代表“此文档中出现此词语的次数”。

出现次数	文档1	文档2	文档3	文档4
词语1	4			1
词语2	3		4	
词语3	3	1		
词语4		3	9	

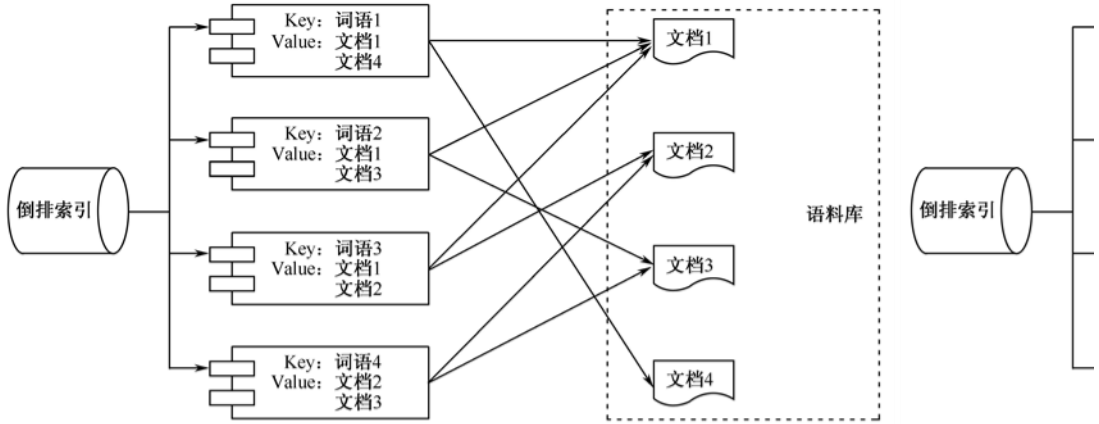
矩阵中的第一列说明“在文档1中，词语1出现了4次、词语2和词语3均出现了3次，并且文档1中不再有其他词语出现”。同理，矩阵中的第一行则说明“词语1在文档1中出现在4次，在文档4中出现1次，在其他文档中不出现”。其他行列同理。

4.3.1倒排索引原理



可以在倒排索引的Value项里增加信息

倒排索引的数据结构



倒排索引使用的数据结构

倒排索引可以使用这样一个Map来实现：每一个词语都是Map中的一个键（Key），这个键对应的Value是一个集合，里面保存着包含这个词语的文档的编号。存储形式为：Map< String key, Set< Struct< DocID > value > >。
同理，如果要在倒排索引中加入更多信息，可以在Value中增加记录项目。

4.3.1倒排索引原理

倒排索引的建立实例

假设现在有两篇文档，每篇文档的内容如下：

文档	内容
文档1	The quick brown fox jumped over the lazy dog.
文档2	Quick brown foxes leap over lazy dogs in summer.

其建立实例的步骤如下：

1.文章本分词

2.去除无关词语

3.词语归一化

4.建立词语-文档矩阵

5.建立到排索引

4.3.1倒排索引原理

倒排索引的更新策略

01

完全重建策略

先进行“文档暂存”，待文档暂存区达到一定数量后，对所有文档重新建立索引。

02

再合并策略

新文档会立即被解析，解析结果会进行“索引暂存”，待索引暂存区达到一定数量后，再将新旧索引合并。

03

原地更新策略

新文档立刻被解析，解析结果立刻被加入旧索引中。

04

混合策略

其思想是混合地使用上述几种策略，取长补短，以达到最好的性能。

4.3.2倒排索引实现

1 任务概述

要求对文件建立倒排索引，使之能够被方便地查询。

2 遍历读取文件

所有的文件都存放在文件夹中，首先要把这些文件读取出来，才能进行后续处理。

3 对单个文件进行处理

包括文本分词、去除无关词语、词语归一化和建立单个文件的信息统计表。

4 将单个文件信息和总体的倒排表进行合并

转变“词语-出现次数”统计表为“词语-文件-出现次数”倒排表。

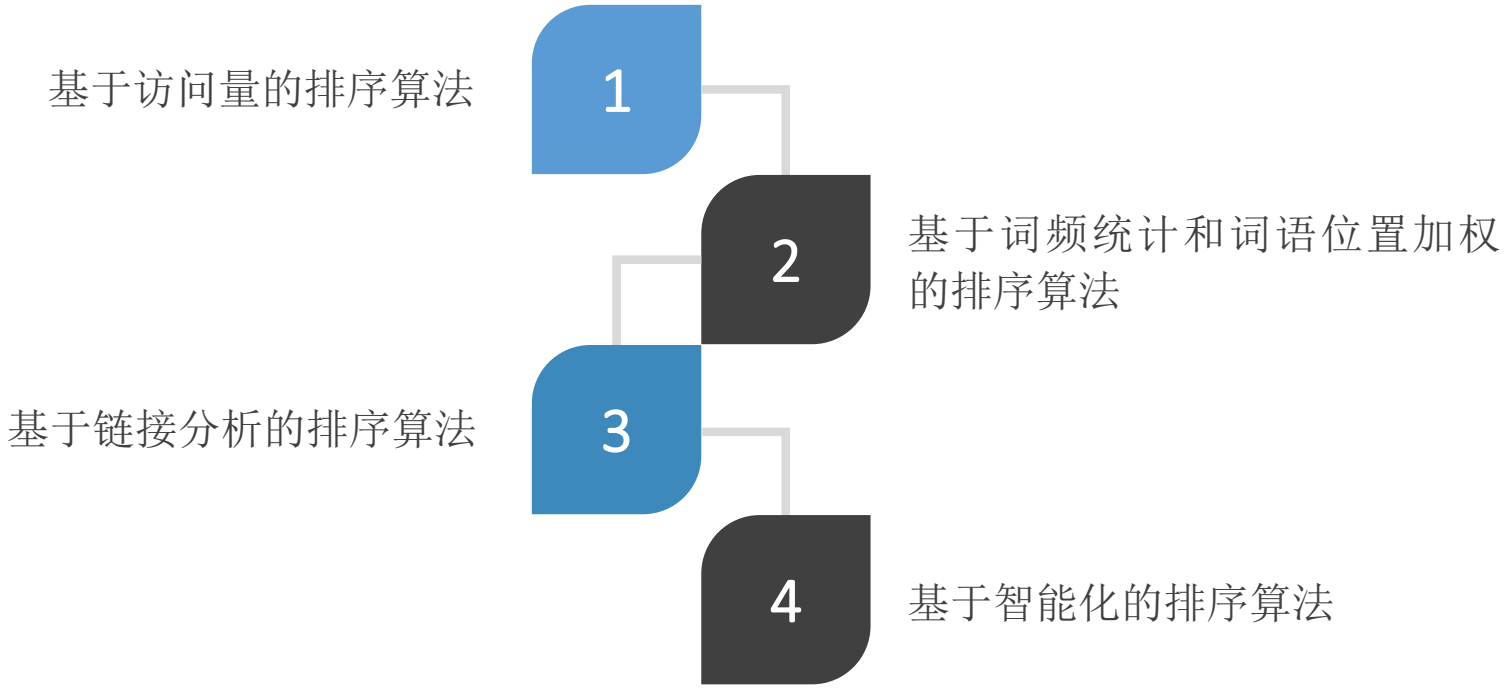
5 查询处理

通过Key查找到对应的Value即可。

第4章　互联网大数据处理

4.4.1 概述

网页排序可分为4种算法大致可分为4种：



4.4.2 TD-IDF算法

TF-IDF是一种统计方法，不仅可以用于评估一个词语对于语料库中某一份文档的重要程度，还可以对搜索结果进行排序，使“重要的”和“贴合搜索关键词的”网页排在前面。基于TF-IDF的网页评分系统在搜索引擎中被广泛使用。

TF的计算公式很多，最简单的形式为：

$$TF = \frac{\text{某个词语A在文档B出现的次数}}{\text{文档B的长度}}$$

逆文档频率的计算公式也有许多，最简单的形式如下：

$$IDF = \log\left(\frac{\text{语料库中文档总数}}{\text{在语料中有多少个文档包含词语A}}\right)$$

除了最简单的形式外，下面这种形式的计算公式也经常被使用：

$$IDF=\log\left(\frac{N-n+0.5}{n+0.5}\right)$$

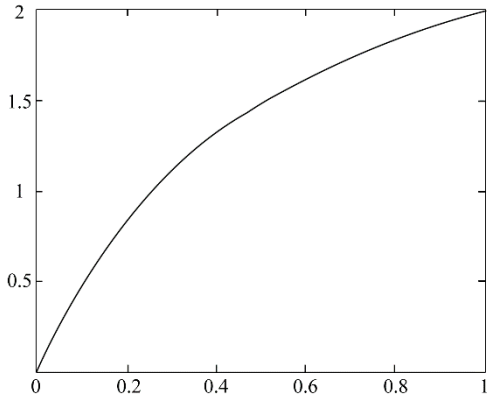
4.4.3 BM25算法

BM25算法是一种基于统计方法的排序算法，是二元独立模型的扩展，或者看作是TF-IDF算法的变形。此算法也是一种有效的相关性评分手段，被搜索引擎广泛使用。

给出查询关键词A，则语料库中某篇文档B的BM25分数定义如下：

$$\text{Score}(A,B)=\text{IDF}(\text{A})\times\frac{f\times(k_1+1)}{f+k_1\times(1-b+b\times k_2)}$$

在这里，IDF是逆文档频率，f是“词语A在文章B中出现的频率”。



当取IDF=1、k1=2、b=0.75、k2=200时，BM25公式的曲线

4.4.3 BM25算法

使用BM25算法来对查询到的网页进行评分，其关键代码如下：

```
class BM25:
    def __init__(self, referance):
        self.referance = referance
        self.k1 = 2
        self.k2 = referance.wordCount / referance.fileCount
        self.b = 0.75
      def getRank(self, word, result):
        for filename in result.keys():
            f = self.referance.invertedTable[word][filename]
            idf = math.log(self.referance.fileCount /
len(self.referance.invertedTable[word]))
            result[filename] = (idf * f * (self.k1 + 1)) / (f + self.k1 * (1 - self.b + self.b
* self.k2))
        return result
```

4.4.4PageRank算法

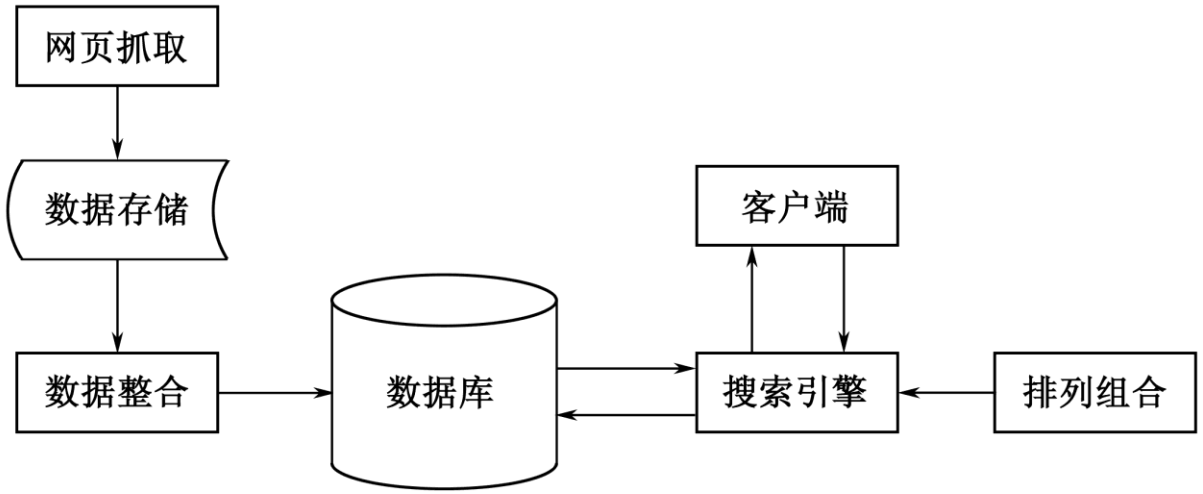
PageRank算法
核心思想

PageRank算法的核心思想是让页面之间通过超链接来进行“投票”：页面A上有一个指向页面H的超链接，就相当于页面A给页面H“投了一票”；一个网页被越多网页链接到，那么这个网页就越受大家信赖，此网页越重要，PageRank值越高；一个很重要、PageRank值很高的网页（如网页B）链接到了其他网页，那么这些网页的PageRank值也会因此提高。

第4章　互联网大数据处理

4.5.1系统架构

面向历史领域的智能信息检索引擎，从互联网上抓取重大历史事件的网站内容，经过数据汇聚和整合从而在数据库中建立专门的数据库。通过在数据库中检索与用户查询条件匹配的相关记录，然后将查询结果进行优化，并按照一定的排序方式将最终结果返回给用户。全文检索系统架构图如下所示。



面向历史领域的智能信息检索引擎的系统架构

4.5.2数据抓取与整合

3种数据采集方式

手动录入
提供内容输入的界面，由历史学家或爱好者手动录入历史事件。

半自动采集
通过自然语言处理、机器学习和人工标注相结合的方法自动抽取历史事件的关键要素。

面向历史领域的非结构化互联网数据抓取
收录用户推荐的重要历史网站和系统自动抓取的历史相关的网页。

4.5.3查询引擎

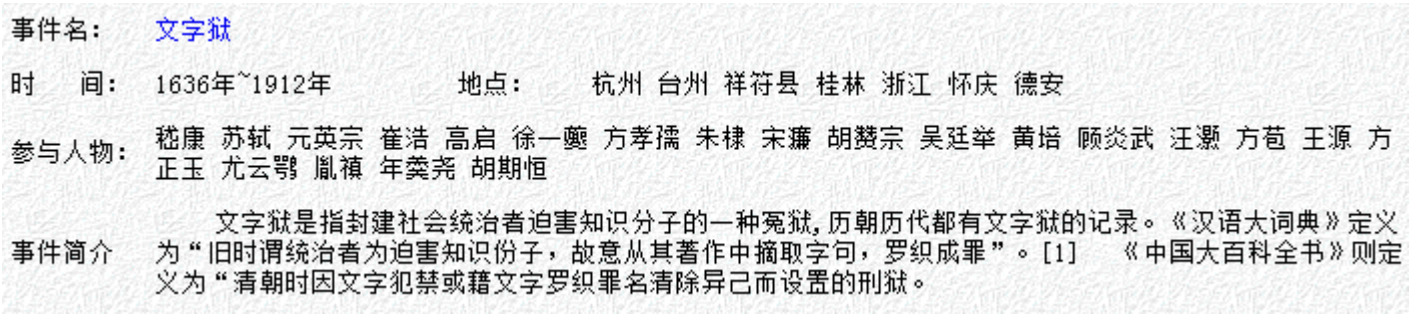
历史信息检索系统使用Java语言开发，为使代码保持较强的可读性和逻辑性，该系统使用Hibernate开源框架进行数据持久化操作。

在历史信息检索中，为了让用户体验尽量达到最好，每个搜索字段之间要保逻辑持“与”的关系。

相同字段之间搜索不同内容的时候也要保持逻辑“与”的关系。

4.5.4运行效果

静态单字段查询界面

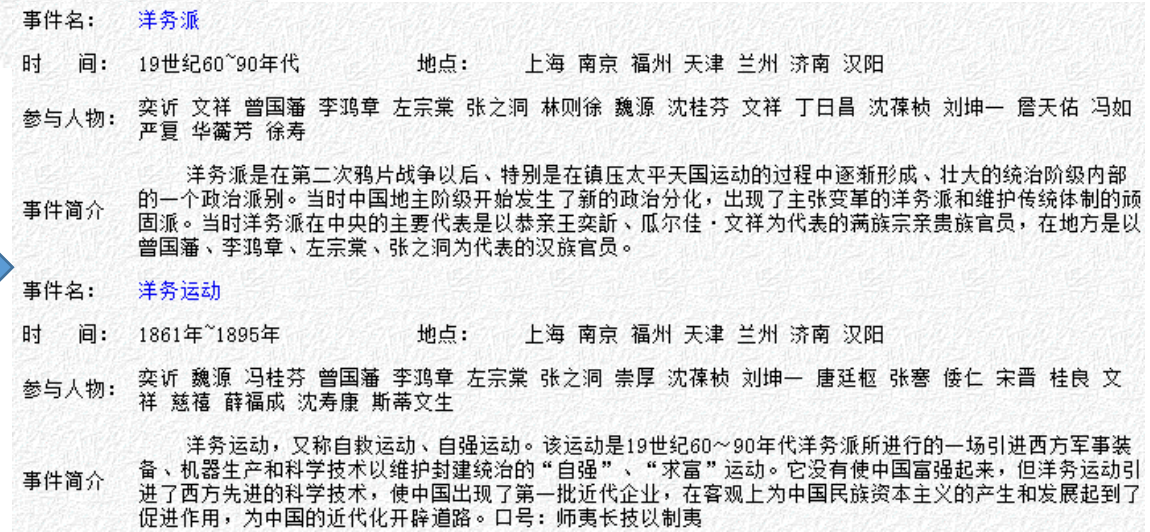


静态单字段查询结果

4.5.4运行效果

动态单字段查询界面

动态单字段查询结果



感谢聆听