第5节.内容索引子系统

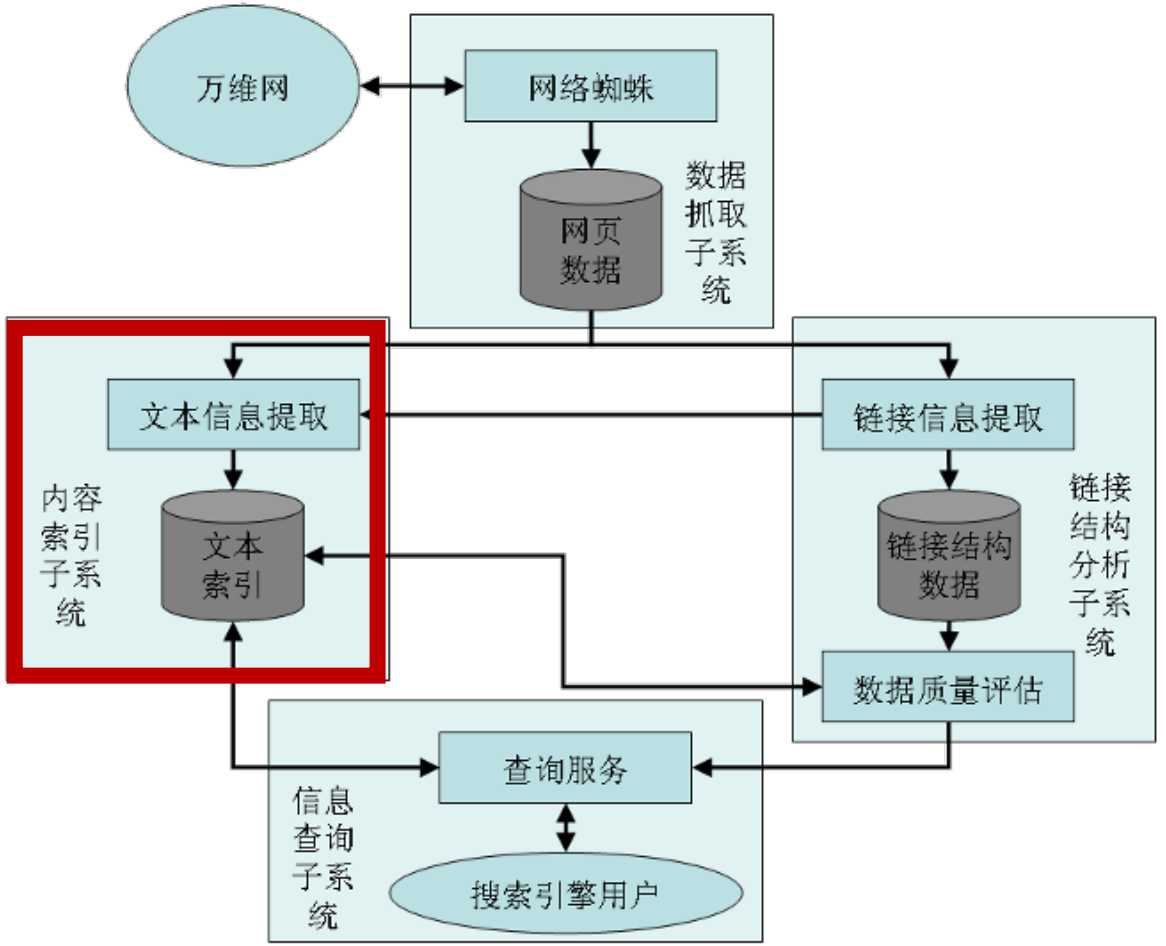
艾清遥

清华大学计算机系

清华大学互联网司法研究院

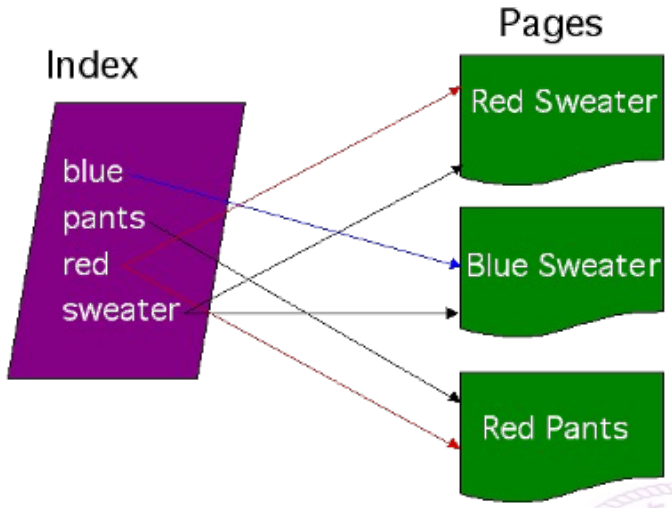
2023年3月21日

搜索引擎体系结构


万维网
网络蜘蛛
网页数据
数据抓取子系统
文本信息提取
内容索引子系统
文本索引
链接信息提取
链接结构数据
链接结构分析子系统
数据质量评估
查询服务
信息查询子系统
搜索引擎用户

引言：内容索引子系统

- 索引子系统的主要功能
  - 建立倒排索引
  - 建立正排索引
  - 提供索引查询服务
- 索引子系统面临的挑战
  - 高效利用系统资源
  - 提供可扩展的索引服务



引言：基本概念回顾

- 词项 (Term)：索引子系统处理的最小语义单位
- 倒排索引结构 (Inverted Index)：
  - 以词项为核心的索引组织形式

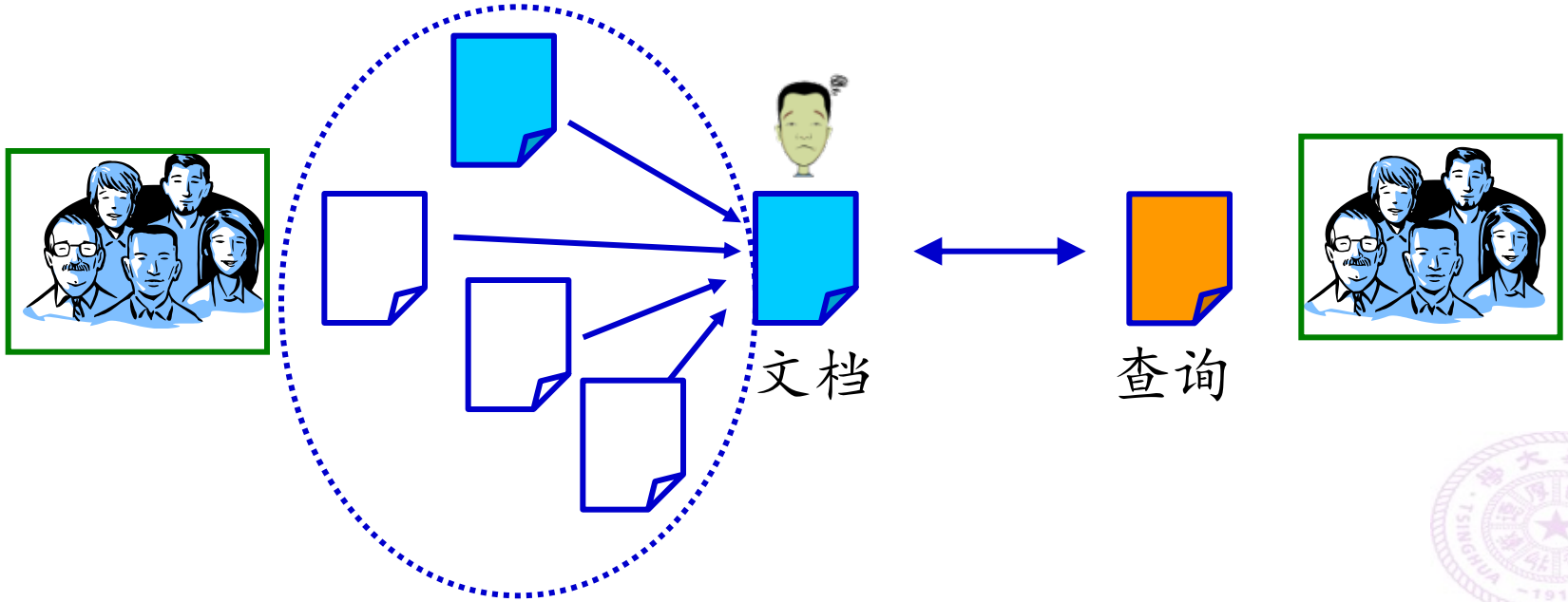
Term 1	Doc 1, pos 1	Doc 1, pos 2	…	Doc p, pos q
Term 2	Doc 1', pos 1'	Doc 1', pos 2'	…	Doc p', pos q'
…				
Term N	Doc $1^{(n)}$, pos $1^{(n)}$	Doc $1^{(n)}$, pos $2^{(n)}$	…	Doc $p^{(n)}$, pos $q^{(n)}$

  - 与人类的长期记忆原理类似
  - 适合于主流商业搜索引擎的交互方式

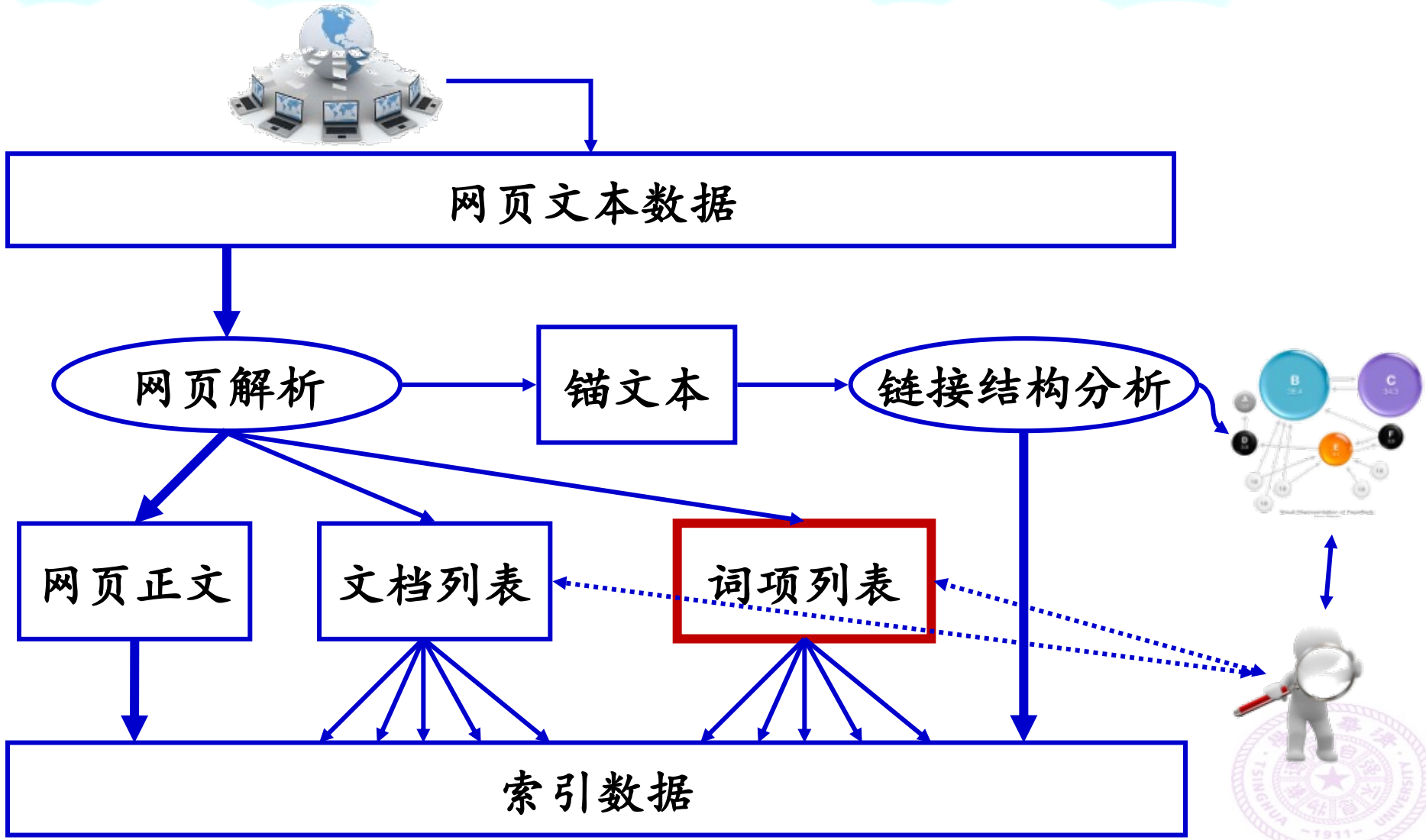
引言：基本概念回顾

- 锚文本：使网页内容与查询内容更好匹配
  - 对链接到网页的内容的描述

<A href="http://www.pku.edu.cn">北大</A>



引言：内容索引子系统基本架构



本节概要

1. 词项列表构建
2. 索引数据结构
3. 索引建立过程
4. 索引查询过程

以Google早期索引结构为例

1. 词项列表构建

- 1.1 词项的作用
  - 作为语义的最小单位出现
    - 需要符合网络数据/用户的表述习惯
    - 根据热门程度调整词项列表：词库更新
  - 作为倒排索引的索引项出现
    - 选取合理的词项数目以高效利用系统资源
    - 数目太大：词项编码长，冗余信息多，浪费空间；
    - 数目太小：词项对应的索引条目多，I/O压力大
- 案例：如何处理数字组成的词项？

1. 词项列表构建

- 案例：如何处理数字组成的词项？

1. 词项列表构建

- 1.2 英文词项处理
  - 大小写统一化、命名实体识别(O‘Neal, Los Angeles)
  - 去除停用词(stop word)
    - 停用词：出现频率高，语义信息量小的词
  - 在尽量保存有用内容的前提下，减少索引空间浪费，保证系统运行效率
  - 极大精简索引结构：去除停用词后，倒排索引可以缩小40%，显著降低I/O负担

1. 词项列表构建

- 1.2 英文词项处理

英文停用词表：http://www.textfixer.com/resources/common-english-words.txt

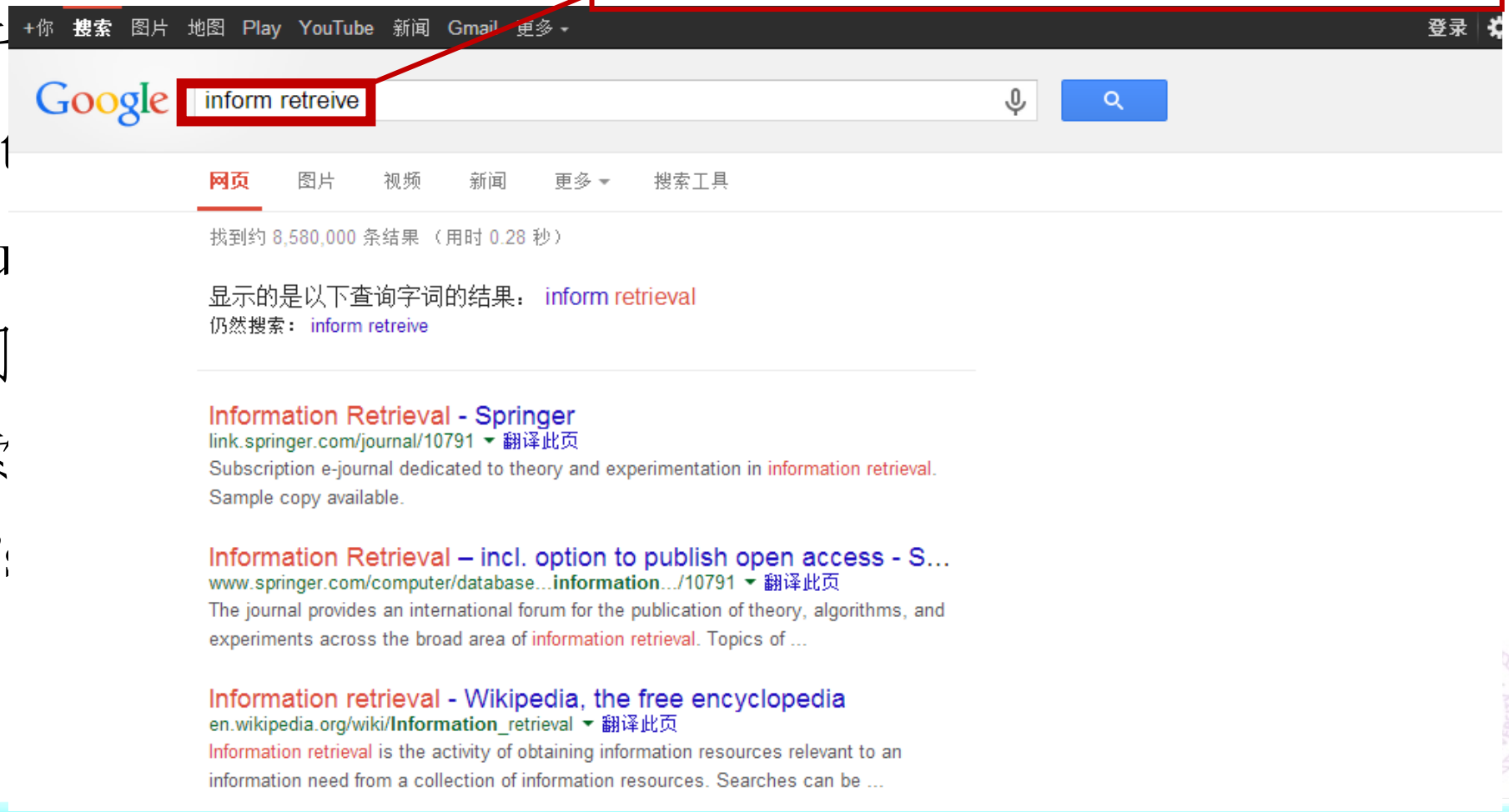
1. 词项列表构建

- 1.2 英文词项处理
  - 取词干(stemming)
  - 词干(stem)：删除词的词缀后剩余的部分
  - Compute, computer, computing => comput
  - 同一词根的不同变形缩减为同一个概念
  - 合并索引项：保证语义不变的情况下缩小索引规模，提高召回率
  - Porter’s stemming algorithm

1. 词项列表构建

- 1.2 英文词项处理
  - 取词干
  - 词干(st
  - Compu
  - 同一词
  - 合并索
  - Porter’s

可能的风险：inform != information



率

1. 词项列表构建

- 1.2 英文词项处理
  - 预处理对词项列表规模的影响

	独立词项			词项-文档			词项-文档-位置		
	number	Δ%	T%	number	Δ%	T%	number	Δ%	T%
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	−2	−2	100,680,242	−8	−8	179,158,204	−9	−9
case folding	391,523	−17	−19	96,969,056	−3	−12	179,158,204	−0	−9
30 stop words	391,493	−0	−19	83,390,443	−14	−24	121,857,825	−31	−38
150 stop words	391,373	−0	−19	67,001,847	−30	−39	94,516,599	−47	−52
stemming	322,383	−17	−33	63,812,300	−4	−42	94,516,599	−0	−52

Manning *et. al.*, Reuters-RCV1

  - “Rule of 30”
    - Top 30 words => 30% appearances

1. 词项列表构建

- 1.3 中文词项处理
  - 编码处理，全/半角处理等
  - 去除停用词：的，是，和，在
  - 风险：停用词的歧义性
    - 的确，和平，存在
  - 案例：不同搜索引擎处理停用词的策略

1. 词项列表构建

- 1.3 中文
  - 编码处
  - 去除停
  - 风险：
    - 的确
  - 案例：

1. 词项列表构建

- 1.3 中文词项处理
  - 中文分词问题：将连续的字序列按照一定的规范重新组合成词序列的过程
  - 基于词典的分词方法
  - 基于理解的分词方法
  - 基于统计的分词方法

1. 词项列表构建

- 1.3 中文词项处理：基于词典的分词方法
  - 由苏联专家在1950年代末提出
  - 将待分析的汉字串与一个“充分大的”机器词典中的词条进行匹配
  - 按照扫描方向的不同：正向匹配、逆向匹配、双向匹配
  - 按照不同长度优先匹配的情况：最大（最长）匹配和最小（最短）匹配

1. 词项列表构建

分词 就是
连续 字序
字序列 词序列
序列
重新 组合
合成 过程

- 中文分词举例（最大匹配）

分词就是将连续的字序列重新组合成词序列的过程

正向匹配：

分词|就是|将|连续|的|字序|列|重新|组合成|词序列|的|过程

反向匹配：

分词|就是|将|连续|的|字序列|重新|组合成|词序列|的|过程

双向匹配：

分词|就是|将|连续|的|字序列|重新|组合|成|词序列|的|过程

词频规则、
上下文规则等

1. 词项列表构建

- 1.3 中文词项处理：分词面临的技术挑战
  - 交集型歧义（overlapping ambiguity）
    - 字符串ABC中，AB，BC同时为词
    - 例：组合成，结合成
    - 例：使用语言的过程就是选词组句的过程
    - 一些解决方案
      - 从语言习惯统计规律上看，哪种分割的可能性大
      - 在语境中，哪种分割方式造成的孤立字少
      - 构建新词，扩充词项列表

1. 词项列表构建

- 1.3 中文词项处理：分词面临的技术挑战
  - 覆盖型歧义（combination ambiguity）
    - 字符串AB中，AB, A, B同时为词
    - 例：中华人民共和国，中国科学院
    - 如何处理：多粒度分词
  - 未登录词识别
    - 词典不可能包括所有的词
    - 人名、地名、机构名、新出现的搭配、旧词新意
    - 如何处理：新词发现（语言现象统计）

1. 词项列表构建

- 1.3 中文词项处理：面向搜索需求的分词
  - 分词算法的时间性能要比较高
    - 不能采用复杂语义理解的方式
  - 多粒度分词，适当引入冗余信息
    - 民进党团 => 民进党团，民进党，党团
  - **用户查询与索引数据需要采用同样的分词方式**
    - 查询：/奥尼尔/；索引：奥/尼/尔 => 查询无结果
    - 查询：/胡戎睿/；索引：胡戎/睿智/力/超群
      => 查询“胡戎睿”无结果

1. 词项列表构建

- 1.3 中文词项处理：面向搜索需求的分词
  - 采用与网络数据环境相匹配的词典
    - 词典质量直接影响分词精度
    - “众包”方式构建词典：细胞词库

1. 词项列表构建

- 案例：搜狗输入法的诞生故事

当时我用的输入法

当时我用的搜索

2005-8-6 12:15 PM

主题： 关于百度拼音输入法创意的补充

我昨天提了一个关于百度拼音输入法的建议，由于很仓促，落了几句。如果把百度拼音输入法里面低调的加入百度搜索的话，人们也会乐于接受，因为人们习惯输入法每天都开着。这时如果再加上只有文件名索引的桌面搜索的话（这样占用空间很少），人们肯定会非常喜欢的。

在天空软件站，仅紫光拼音的下载量就有400万左右，紫光的总下载量应该有至少2000万……

Re: 关于百度拼音输入法创意的补充
2005-8-6 10:06:59

您好:

谢谢您的建议.

感谢您使用百度

百度

马先生:

好想法，好创意！佩服！
如果搜狗输入法的设想实现，您的功劳当属NO.1,我马上反馈给相关负责人，
谢谢您的支持，搜狗有您这样忠诚的网民的关注和投入，肯定能超越竞争对手！:)

1. 词项列表构建

- 1.4 词项列表的规模

Word ID	Number of docs	Pointer
Word ID	Number of docs	Pointer
……		
Word ID	Number of docs	Pointer

N words

  - 词项数目规模庞大
  - 受到内存规模限制
    - 索引建立、内容检索模块客观需要

1. 词项列表构建

- 1.4 词项列表的规模：Heaps' Law
  - 对于规模为L个词的语料库，其独立词项数V为：

$$V = k \cdot L^b$$

  - 其中，参数k：[30,100]；参数b≈0.5。
  - 参数k的取值取决于预处理方法
  - 结论1：词项的规模会随着文档数目的增加持续增长，而不会稳定在某个阈值。
  - 结论2：Web规模文档集合的词汇量必将非常巨大

1. 词项列表构建

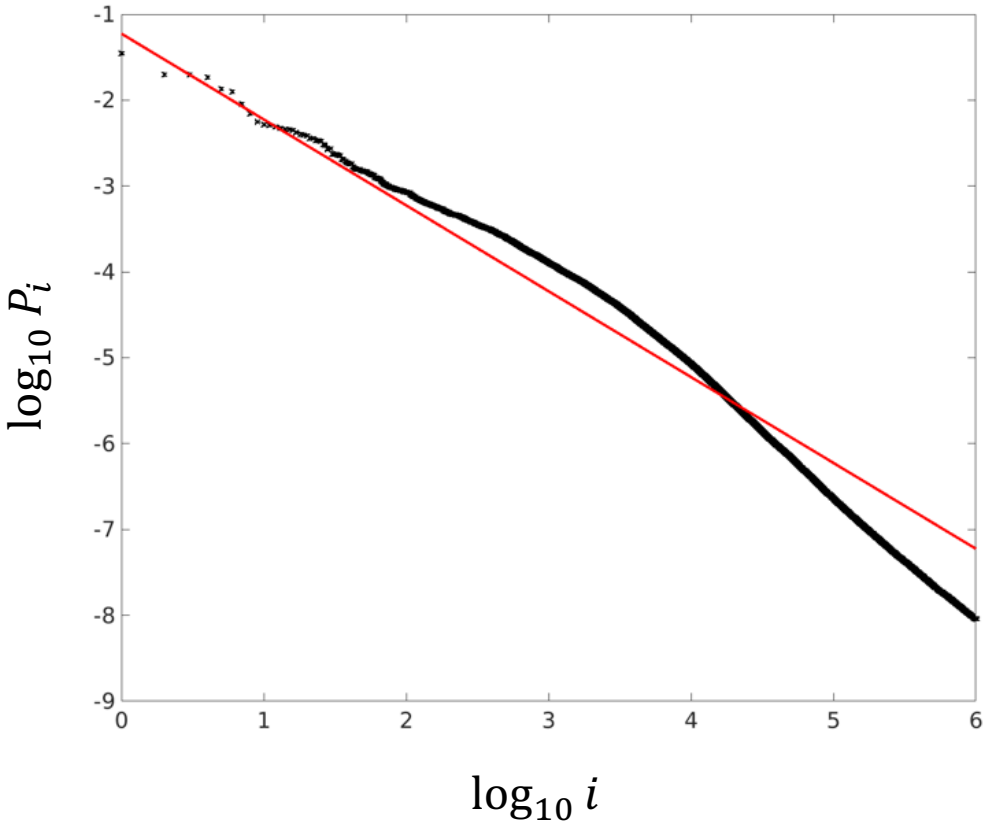
- 1.4 词项列表的规模：Zipf's Law
  - 语料库中的词项按频度进行排序后，记其频度为P_1, P_2, ..., P_n，序号i为1, 2, …, n，则近似有：

$$P_i \propto 1/i$$

  - George K. Zipf: Human behavior and the principle of least-effort. 1949. Harvard University.
  - Zipf's law反映的是人类语言规律的客观特性，在不同语言之间具有普适性。
  - 是幂律 (Power Law) 的一种特殊形式

George Kingsley Zipf
(1902-1950)

Zipf's Law: Google 英文词表



$$\log_{10} \frac{P_i}{N} + \log_{10} i = 0.59$$

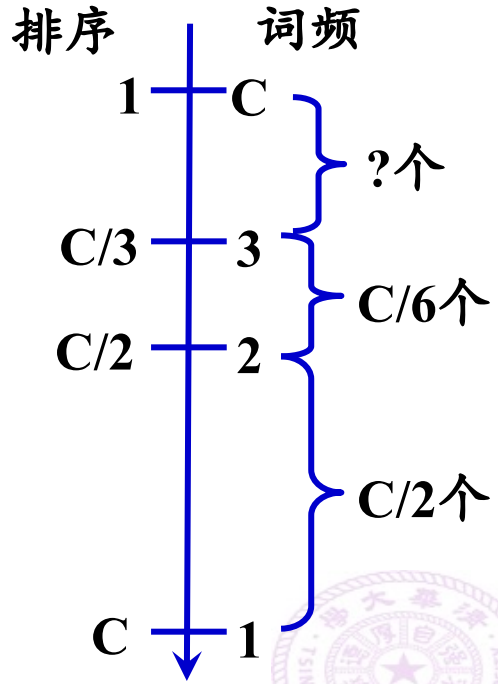
N: 数据集中所有英文单词频数之和

1. 词项列表构建

- 1.4 词项列表的规模：Zipf's Law

$$P_i \propto 1/i \quad \Rightarrow \quad i \propto 1/P_i \quad \Rightarrow \quad i = C/P_i$$

  - $P_i = 1$ 则 $i = C$
  - $P_i = 2$ 则 $i = C/2$，
    词频为2的词项数为$C/2$
  - $P_i = 3$ 则 $i = C/3$，
    词频为3的词项数为$C/6$
  - 低频词是词项列表的主要组成部分

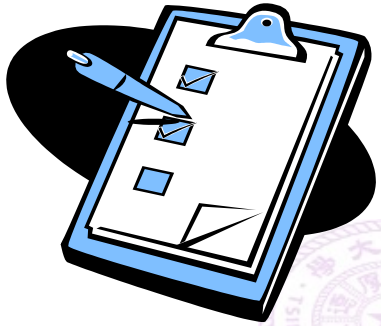


1. 词项列表构建

- 1.4 词项列表的规模
  - Heaps' Law: 词项列表的规模与语料规模共同增长
  - Zipf's Law:大规模语料中，数目较少的高频词占大部分语料内容，但低频词占有词项数目的绝大部分
    - Brown Corpus：包含1,014,312词，其中135个词覆盖语料的 50%，2000个词覆盖语料的80%。
    - 低频词数目众多，存储空间和编码空间的浪费
  - 有损优化：不必将所有词项加入内存中的词项列表，(在内存中)忽略最不常用的词项

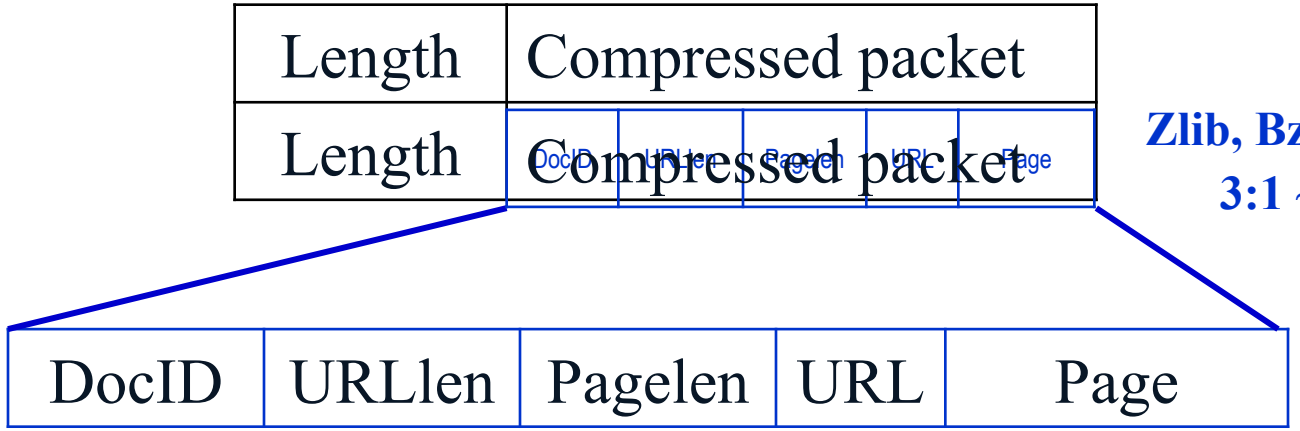
本节概要

1. 词项列表构建
2. 索引数据结构
   2.1 原始数据存放
   2.2 词项出现记录
   2.3 倒排/正排索引
3. 索引建立过程
4. 索引查询过程



2. 索引数据结构

- 2.1 原始数据存放
  - 目的：保证具备重建索引的能力
    - 硬件崩溃时有发生，累计式抓取周期过长
    - Archive the Web: WebInfoMall
  - 存储结构与压缩



2. 索引数据结构

- 2.2 词项出现信息记录(Hit record, token, entry)
  - 索引存储的基本单位 (正排/倒排)
  - 词项在该文档中某次特定出现的信息(2 bytes)

是否大写(1 bit)	字体(3 bits)	位置(12 bits, 4096)
0	000	0000 0000 0010
0	000	0000 0000 0101
0	010	0000 0001 0001

/当/网络/经济/在/此次/经济/危机/中/再度/被/抛向/空中/时/，/众多/的/互联网/企业/都/在/思考/如何/“过冬”/。/对于/企业/来讲/，/网站/推广/和/市场营销/的/作用/日益/凸显/。/经济/危机/正/促使/整个/行业/发生/着/变化/。

2. 索引数据结构

- 2.2 词项出现信息记录

111为保留位

HTML规范中，规定了页面中的六级标题格式：

<html>

<h1>This is the main header</h1>

Some initial text

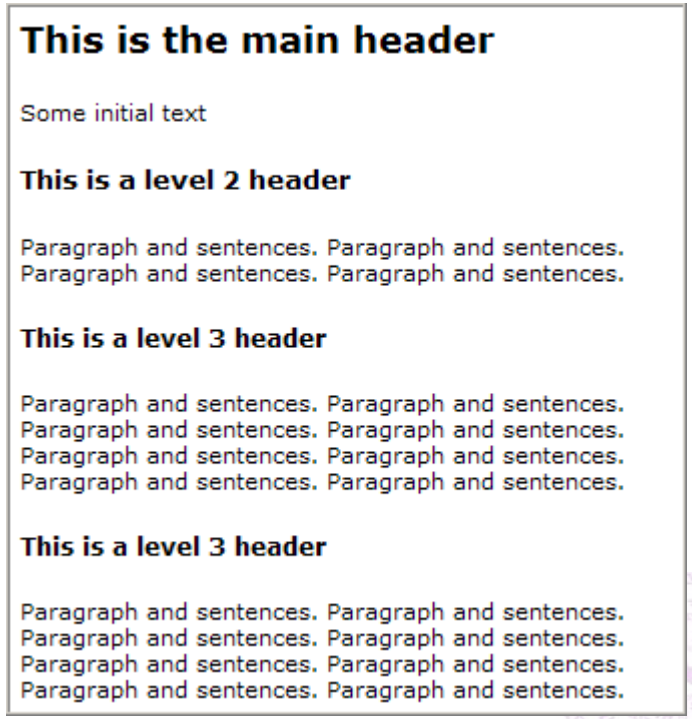
<h2>This is a level 2 header</h2>

Paragraph and sentences.

<h3>This is a level 3 header</h3>

……

</html>



2. 索引数据结构

- 2.2 词项出现信息记录

1111为保留位

是否大写(1 bit)	字体='111'	类别(4 bits)	位置(8 bits, 256)

(HTML特殊域)

```
<HTML>
  <HEAD>
    <TITLE>…</TITLE>
    <Meta>
  </HEAD>
  <BODY>
    …
  </BODY>
</HTML>
```

```
<meta name="keyword" contents="key1, key2 ….."> 
<meta name="description" contents="description…."> 
```

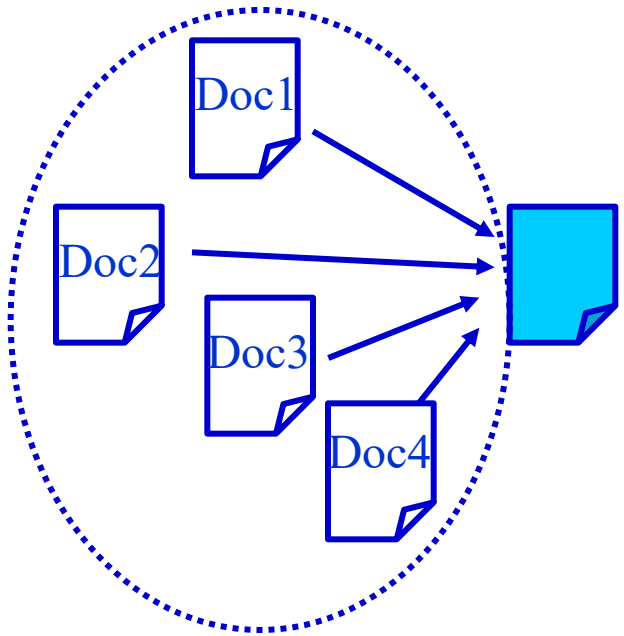
```
<b>粗体</b>
<i>斜体</i>
<u>加下划线</u>
<font color="#ff0000">红色的字</font>
.....
<a href="http://sunspot-design.com">anchor text</a>
```

2. 索引数据结构

- 2.2 词项出现信息记录

文档编号的Hash值

是否大写(1 bit)	字体='111'	类别='1111'	编号(4 bits)	位置(4 bits)
			(锚文本)	(锚文本来源)



Doc 1: 清华/大学/主页

0	111	1111	0000	0000

Doc 2: 清华/主页

0	111	1111	0001	0000

Doc 3: 世纪/清华

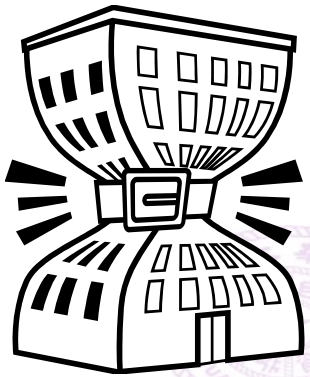
0	111	1111	0010	0001

Doc 4: 清华/我/爱/清华

0	111	1111	0011	0000
0	111	1111	0011	0011

2. 索引数据结构

- 2.2 词项出现信息记录
  - 需要记录的基本信息：位置、字体、类别
  - 方案1：simple encoding (3 integers)
  - 方案2：simple encoding + zip
  - 方案3：hand optimized allocation of bits
  - 优势：节约每一个比特
  - 问题：有损优化，可能的位置信息丢失，可能的锚文本来源丢失



2. 索引数据结构

- 2.3 倒排索引与正排索引
  - 均以词项出现信息记录(HIT record)为最小单位

Term 1	Doc 1, pos 1	Doc 1, pos 2	…	Doc p, pos q
Term 2	Doc 1’, pos 1’	Doc 1’, pos 2’	…	Doc p’, pos q’
…				
Term N	Doc $1^{(n)}$, pos $1^{(n)}$	Doc $1^{(n)}$, pos $2^{(n)}$	…	Doc $p^{(n)}$, pos $q^{(n)}$

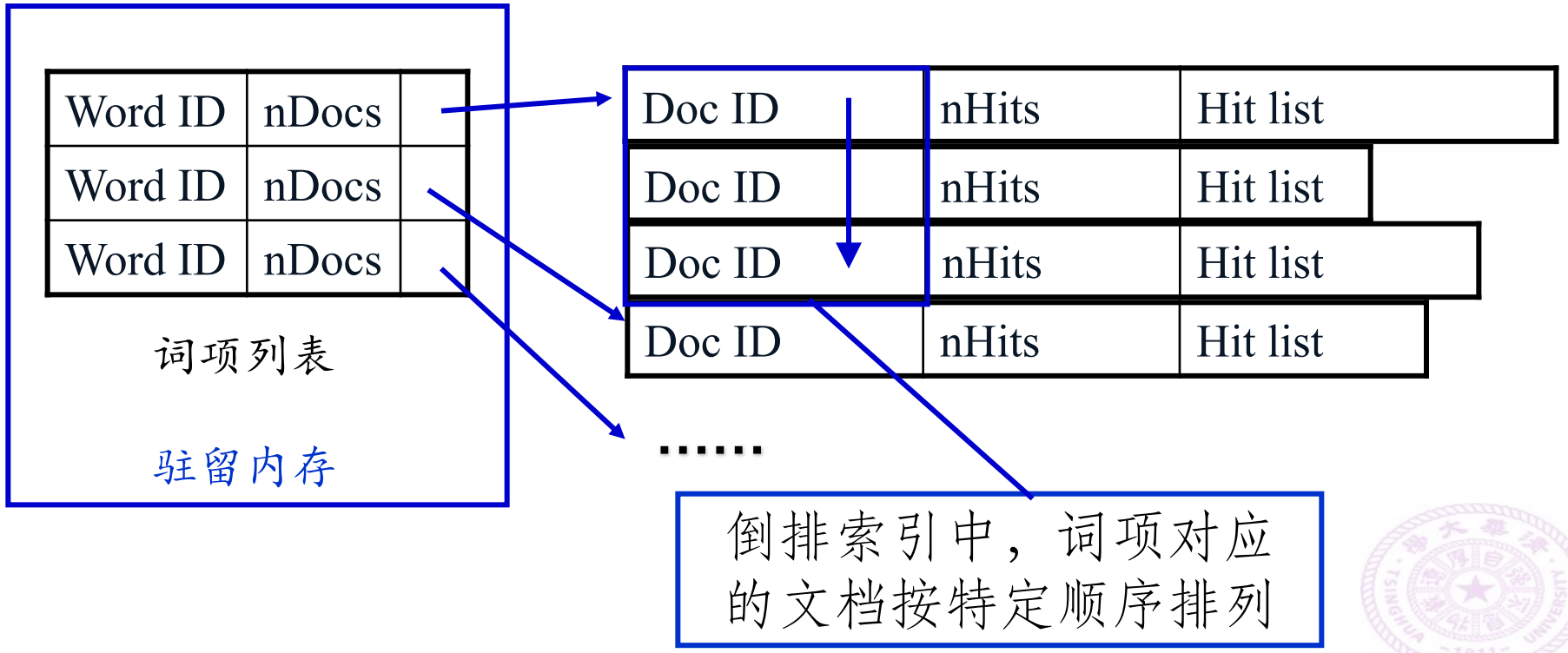
Doc 1	Term 1, pos 1	Term 1, pos 2	…	Term p, pos q
Doc 2	Term 1’, pos 1’	Term 1’, pos 2’	…	Term p’, pos q’
…				
Doc N	Term $1^{(n)}$, pos $1^{(n)}$	Term $1^{(n)}$, pos $2^{(n)}$	…	Term $p^{(n)}$, pos $q^{(n)}$

2. 索引数据结构

- 2.3 倒排索引与正排索引：正排索引
  - 输入Doc ID，输出WordID列表及对应的Hit lists

2. 索引数据结构

- 2.3 倒排索引与正排索引：倒排索引
  - 输入WordID，输出DocID列表及对应的Hit lists

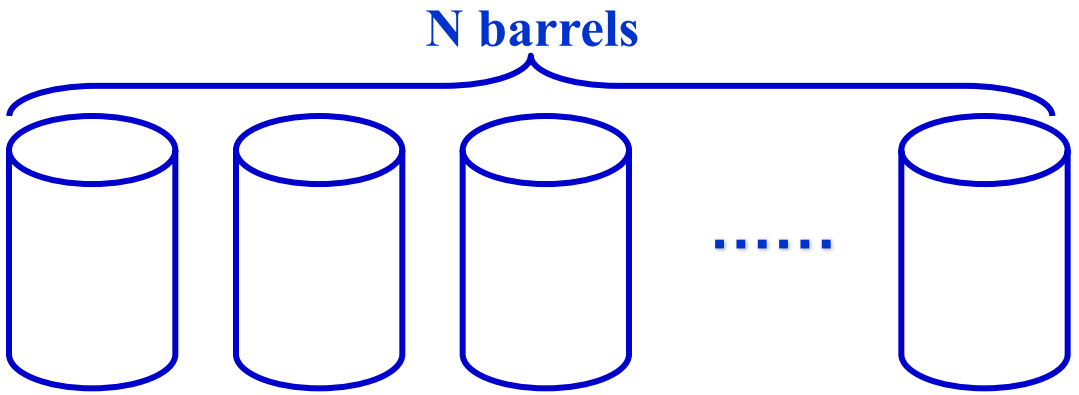


2. 索引数据结构

- 2.4 索引的并行存储结构
  - 在有限的磁盘存储量限制下增加存储总量
  - 在有限的磁盘I/O性能限制下增加索引吞吐效率
  - 并行存储单元
    - 物理或虚拟存储单位:
      “桶”(barrel) / “环”(circle) / “碎片”(shard)
  - 如何划分存储单元
    - 按文档划分
    - 按词项划分

2. 索引数据结构

- 2.4 索引的并行存储结构：按词项划分
  - 每个桶包含1/N的词项，以及这部分词项对应的所有文档
  - 倒排索引：直接访问对应WordID的桶
  - 正排索引：访问所有桶，获得对应DocID的词项列表(不完整)和Hit lists

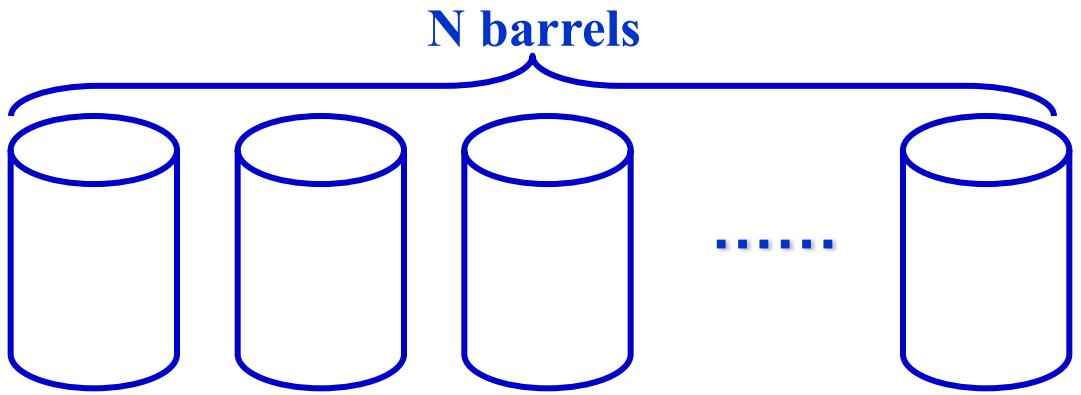


2. 索引数据结构

- 2.4 索引的并行存储结构：按词项划分
  - 优点：对包含K个词项的查询，至多需要K个桶共同进行处理，只需进行O(K)次磁盘查找操作
  - 缺点：
    - 网络传输量大(需要传输整个词项对应的倒排表)
    - 需要在索引结构之外存储文档的其他信息
    - 扩展性较差：更新/增加文档

2. 索引数据结构

- 2.4 索引的并行存储结构：按文档划分
  - 每个桶包含1/N的文档，以及这部分文档包含的所有词项
  - 正排索引：直接访问对应DocID的桶
  - 倒排索引：访问所有桶，获得对应WordID的文档列表(不完整)和Hit lists



2. 索引数据结构

- 2.4 索引的并行存储结构：按文档划分
  - 优点：可扩展性较好
    - 每个桶可以独立处理查询请求
    - 每个桶可以保存文档的其他信息(如PageRank)
    - 网络传输量小 (只需传输查询需求和每个桶的查询结果)
  - 缺点：
    - 需要等待所有桶处理完查询请求，以避免重要文档丢失
    - 处理包含K个词项的查询，需进行O(K*N)次查找操作
    - 可能的解决方式：创建多个重要文档的镜像

本节概要

1. 词项列表构建
2. 索引数据结构
3. 索引建立过程
4. 索引查询过程

3. 索引建立过程

- 3.1 文档预处理

词项列表

锚文本重新定位

MD5(http://www.tsinghua.edu.cn,16) = b8261de8099e73da

<title>清华大学</title>

<style type="text/css">…</style>

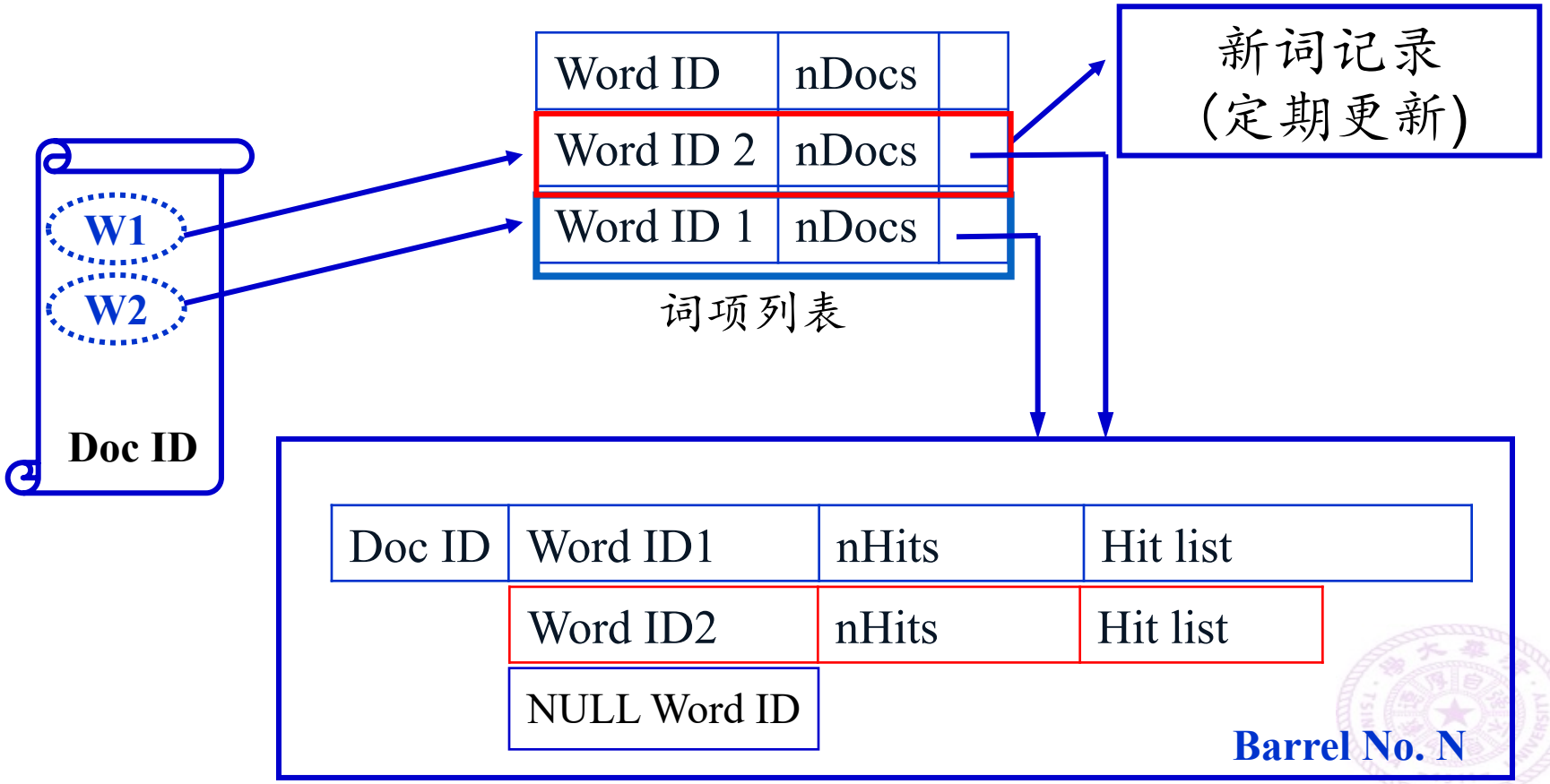
/系

加入：清华、世纪清华、…

移出：清华大学教育基金会、校友总会、…

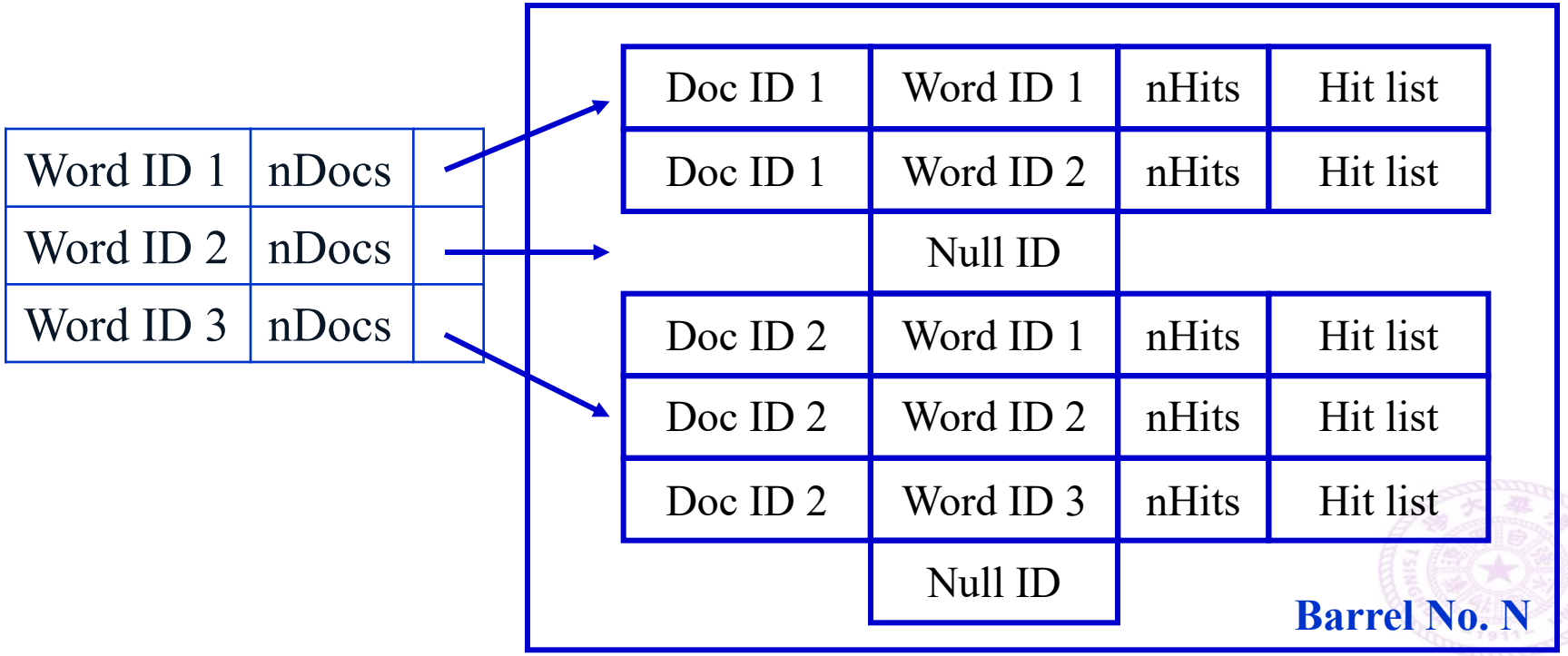
3. 索引建立过程

- 3.2 构建正排索引：以按文档分桶为例



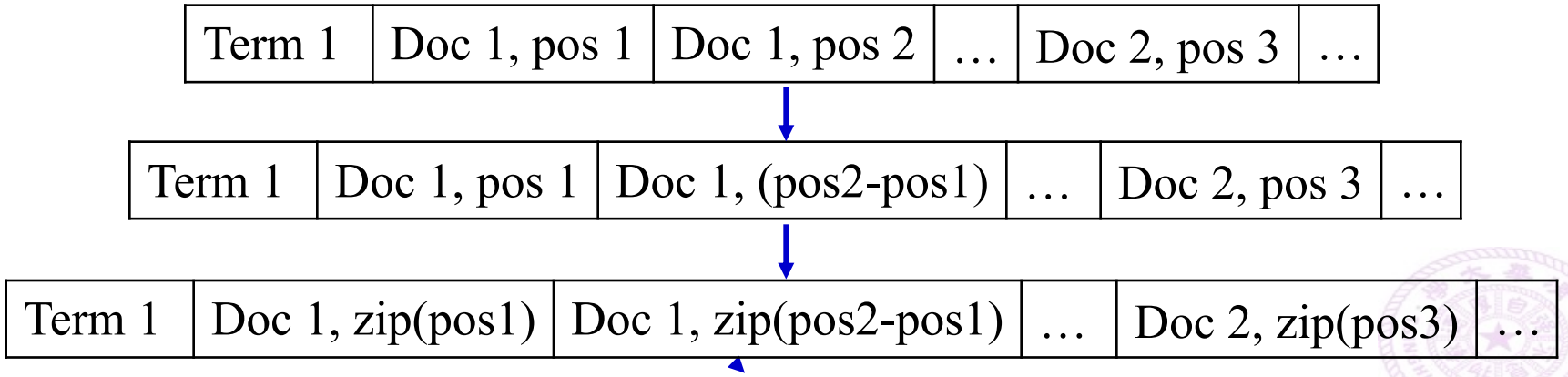
3. 索引建立过程

- 3.3 构建倒排索引
  - 在每个桶内，对正排索引按Word ID进行排序，重构成倒排索引



3. 索引建立过程

- 3.4 性能优化：索引压缩
  - 优点：节约空间，有效利用I/O，增加硬盘寿命
  - 缺点：额外占用计算资源，传统的压缩算法可能导致索引随机读写的困难
  - 方案：主要针对位置信息进行压缩

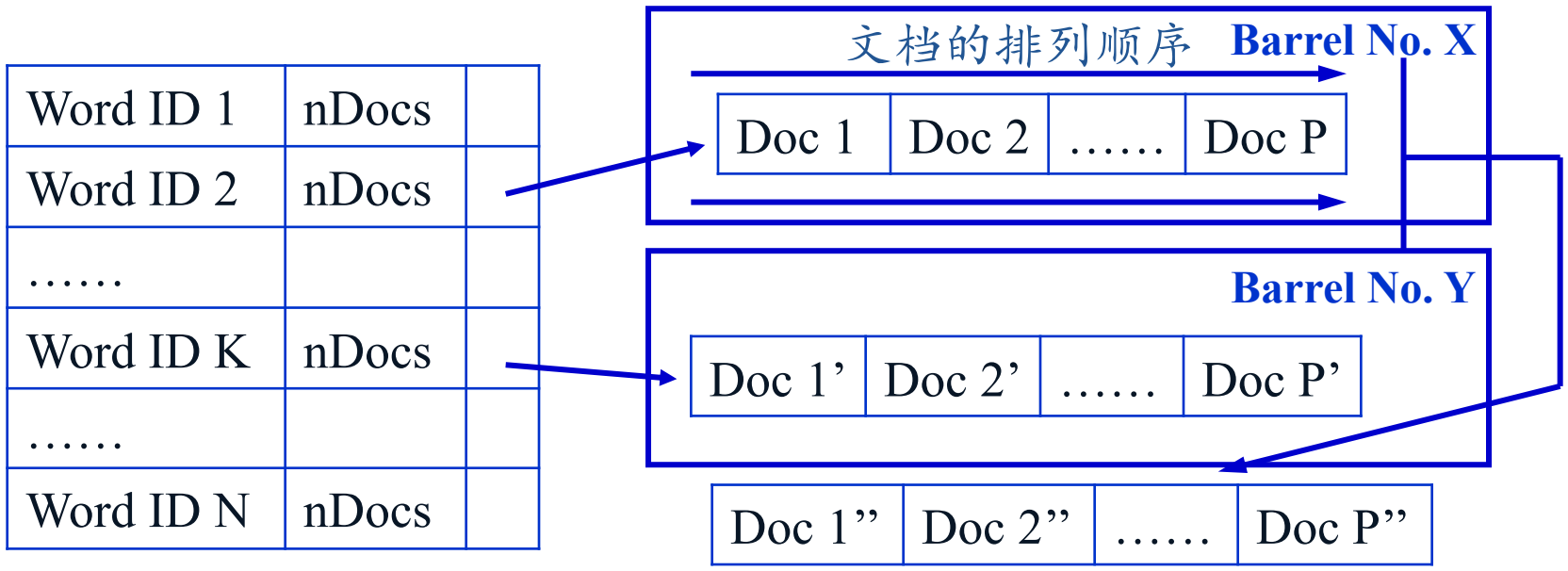


本节概要

1. 词项列表构建
2. 索引数据结构
3. 索引建立过程
4. 索引查询过程

4. 索引查询过程

- 4.1 查询过程：以按词项分桶为例
  - 用户输入：(Word ID 2, Word ID K)



- 系统输出：包含以上查询词的(部分)文档列表

4. 索引查询过程

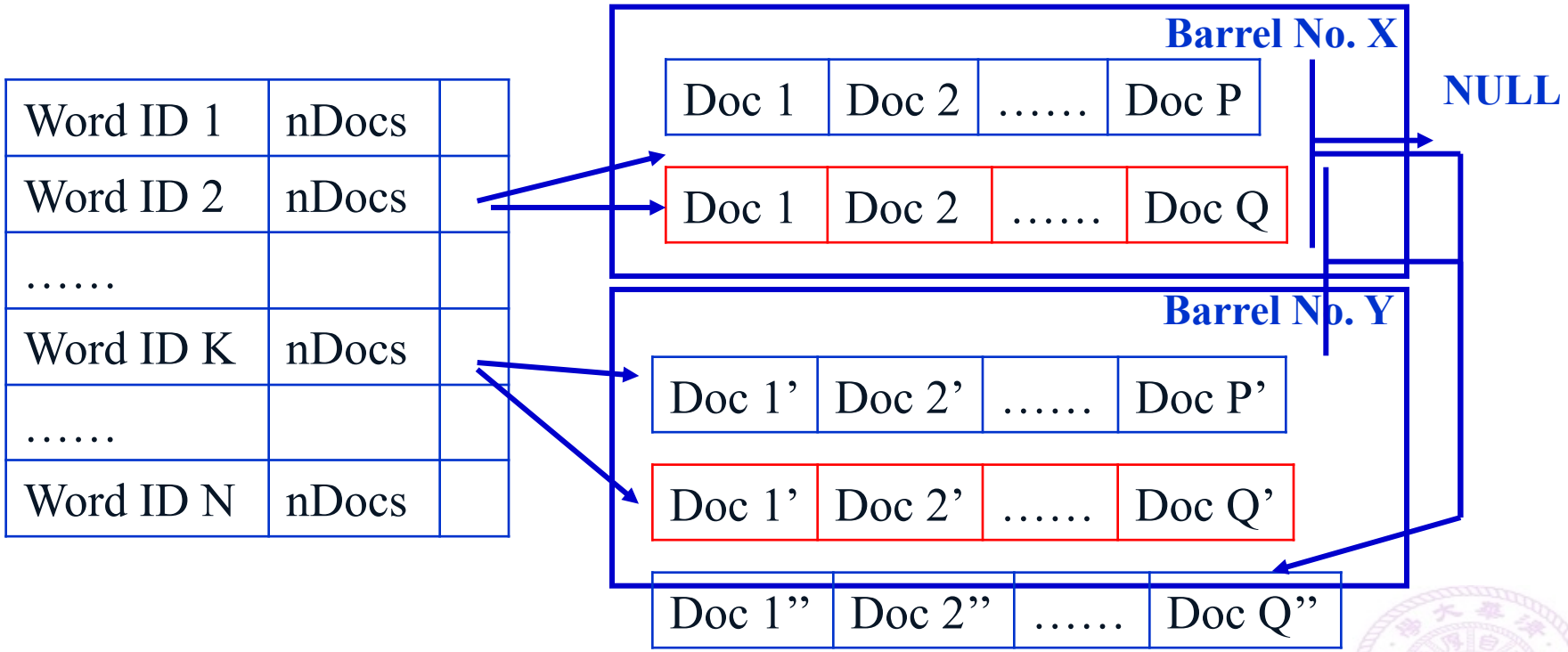
- 4.2 性能优化：缓存服务器
  - 保存常用索引项
  - 保存常用同现词项的文档列表求交结果
  - Google: 缓存命中率30-60%,
  - 本质： 时间局部性原理
    - 容易被缓存命中的查询词，经常是热门的，同时也是资源消耗量巨大的 (网络资源数量多)

4. 索引查询过程

- 4.2 性能优化：关键位置倒排索引
  - 关键
  - 对于
  - 在HT 的文档更为

4. 索引查询过程

- 4.2 性能优化：关键位置倒排索引



总结

- 词项列表构建
  - 分词算法/词项列表构建
- 索引数据结构
  - 原始数据存放/词项出现信息记录/
    正排索引与倒排索引/并行索引结构
- 索引系统运行方式
  - 索引建立/索引使用/性能优化