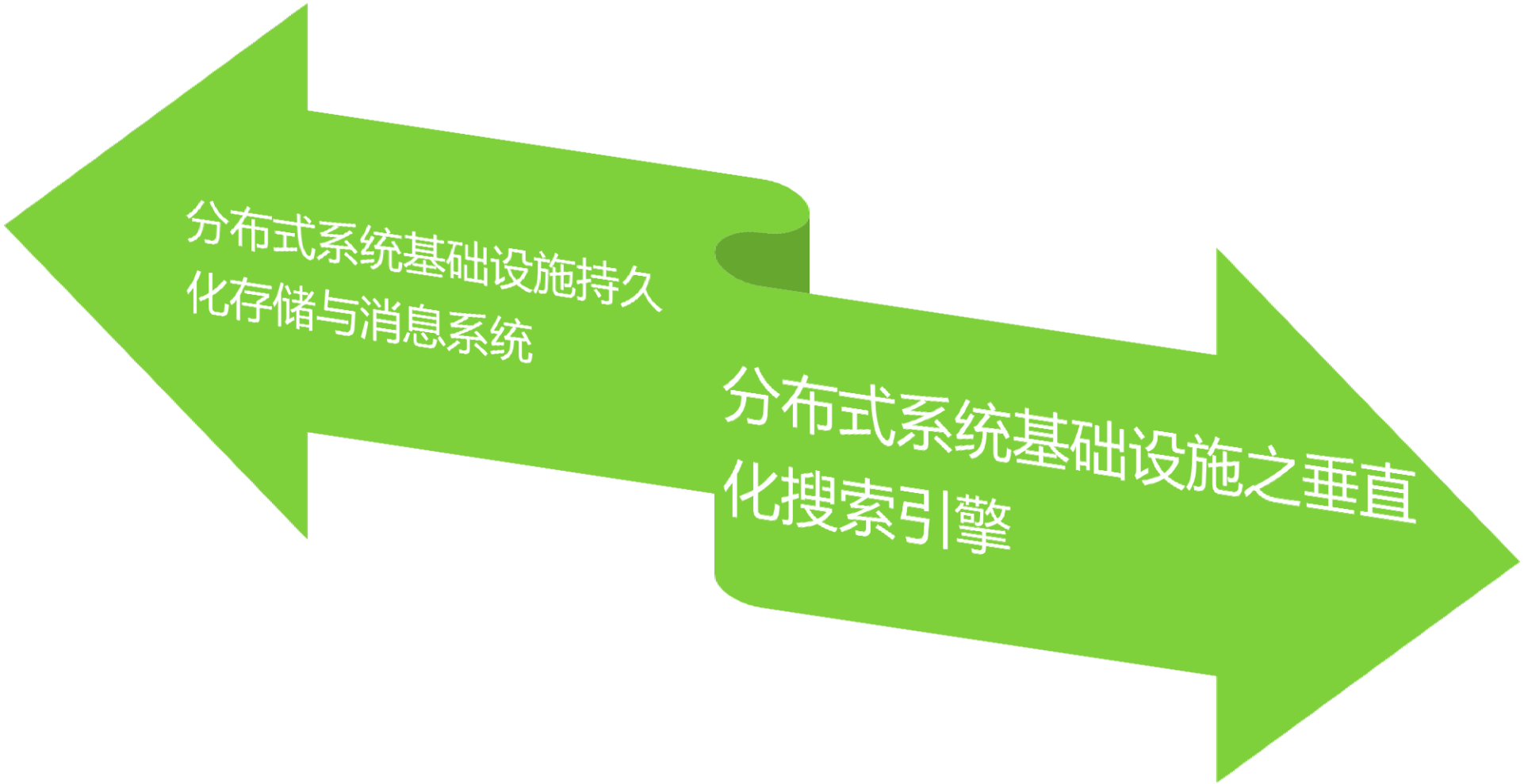
大型电商分布式系统实践 第7周

DATAGURU专业数据分析社区

上节内容回顾



垂直化搜索引擎

垂直化的搜索引擎，与大家所熟知的google和baidu等互联网搜索引擎巨头存在着一些差别，垂直化的搜索引擎主要针对企业内部的自有数据的检索，而不像google和baidu等搜索引擎平台，采用网络爬虫对全网数据进行抓取，从而建立索引并提供给用户进行检索。在分布式系统中，垂直化的搜索引擎是一个非常重要的角色，它即能够满足用户对于全文检索、模糊匹配的需求，解决数据库like查询效率低下的问题，又能够解决分布式环境下，由于采用分库分表、或者是使用NOSQL数据库，导致无法进行多表关联或者进行复杂查询的问题。

Lucene简介

要深入的理解垂直化搜索引擎的架构，不得不提到当前全球范围内使用十分广泛的一个开源检索工具—lucene。lucene是apache旗下的一款高性能、可伸缩的开源的信息检索库，最初是由Doug Cutting所开发，并在SourceForge的网站上提供下载，从2001年9月开始，lucene做为高质量的开源java产品加入到Apache软件基金会，经过多年的不断发展和成熟，lucene被翻译成C++、C#、perl、python等多种语言，在全球范围内众多知名互联网企业中得到了极为广泛的使用。通过lucene，可以十分容易的为你的应用程序添加文本搜索功能，而不必深入地了解搜索引擎实现的技术细节以及高深的算法，极大的降低了搜索技术推广及使用的门槛。

搜索引擎的几个概念--倒排索引

倒排索引(inverted index)，也称为反向索引，是搜索引擎中最常见的数据结构，几乎所有的搜索引擎都会使用到倒排索引，它将文档中的词作为关键字，建立词与文档的映射关系，通过对倒排索引的检索，可以根据词快速获取包含这个词的文档列表，这对于搜索引擎来说至关重要。

搜索引擎的几个概念--分词

分词，又称为切词，就是将句子或者段落进行切割，从中提取出包含固定语义的词。对于英语来说，语言的基本单位就是单词，因此，分词特别容易，只需要根据空格/符号/段落进行分割，并且排除停止词(stop word)，提取词干，即可完成，但是对于中文来说，要将一段文字准确的切分成一个个词，就不那么容易了，中文是以字为最小单位，多个字连在一起才能构成一个表达具体含义的词，中文的句子和段落都有一个明显的标点符号分割，唯独词没有一个形式上的分割符，因此，对于支持中文搜索的搜索引擎来说，需要一个合适的中文分词工具，以便建立倒排索引。 ：提取词干是西方语言特有的处理步骤，比如英文中的单词有单复数的变形，-ing和-ed的变形，但是在搜索引擎中，应该当做同一个词。

搜索引擎的几个概念--停止词

停止词(stop word)，在英语中包含了a、the、and这样使用频率很高的词，如果这些词都被建到索引中进行索引的话，搜索引擎就没有任何意义了，因为几乎所有的文档都会包含这些词，对于中文来说也是如此，中文里面也有一些出现频率很高的词，如“在”、“这”、“了”、“于”等等，这些词没有具体含义，区分度低，搜索引擎对这些词进行索引没有任何意义，因此，停止词需要被忽略掉。

搜索引擎的几个概念--排序

排序，当输入一个关键字进行搜索时，可能会命中许多文档，搜索引擎给用户的价值就是快速的找到需要的文档，因此，需要将相关度更大的内容排在前面，以便用户能够更快的筛选出有价值的内容，这时，就需要有适当的排序算法。一般来说，命中标题的文档将比命中内容的文档有更高的相关性，命中多次的文档比命中一次的文档有更高的相关性，商业化的搜索引擎的排序规则十分复杂，搜索结果的排序融入了广告、竞价排名等因素，由于牵涉的利益广泛，一般属于核心的商业机密。

lucene的几个概念--文档

文档(document)，在lucene的定义中，文档是一系列域(field)的组合，而文档的域则代表一系列与文档相关的内容，与数据库表的记录的概念有点类似，一行记录所包含的字段对应的就是文档的域，举例来说，一个文档比如老师的个人信息，可能包括年龄、身高、性别、个人简介等等内容。

lucene的几个概念--域

域(field)，索引的每个文档中都包含一个或者多个不同名称的域，每个域都包含了域的名称和域对应的值，并且，域还可以是不同的类型，如字符串、整型、浮点型等。

lucene的几个概念—词

词(term)，term是搜索的基本单元，与field对应，它包括了搜索的域的名称以及搜索的关键词，可以用它来查询指定域中包含特定内容的文档。

lucene的几个概念—查询

查询(query)，最基本的查询可能是一系列term的条件组合，称为TermQuery，但也有可能是短语查询(PhraseQuery)、前缀查询(PrefixQuery)、范围查询(包括TermRangeQuery、NumericRangeQuery等)等等。

lucene的几个概念—分词器

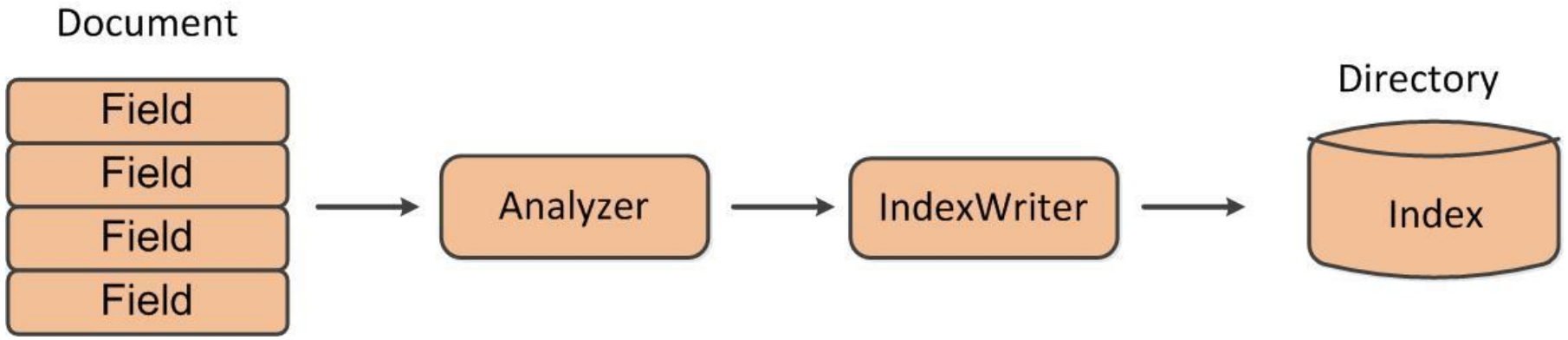
分词器(analyzer)，文档在被索引之前，需要经过分词器处理，以提取关键的语义单元，建立索引，并剔除无用的信息，如停止词等，以提高查询的准确性。中文分词与西文分词区别在于，中文对于词的提取更为复杂。常用的中文分词器包括一元分词、二元分词、词库分词等等。

一元分词，即将给定的字符串以一个字为单位进行切割分词，这种分词方式较为明显的缺陷就是语义不准，如“上海”两个字被切割成“上”、“海”，但是包含“上海”、“海上”的文档都会命中。

二元分词比一元分词更符合中文的习惯，因为中文的大部分词汇都是两个字，但是问题依然存在。

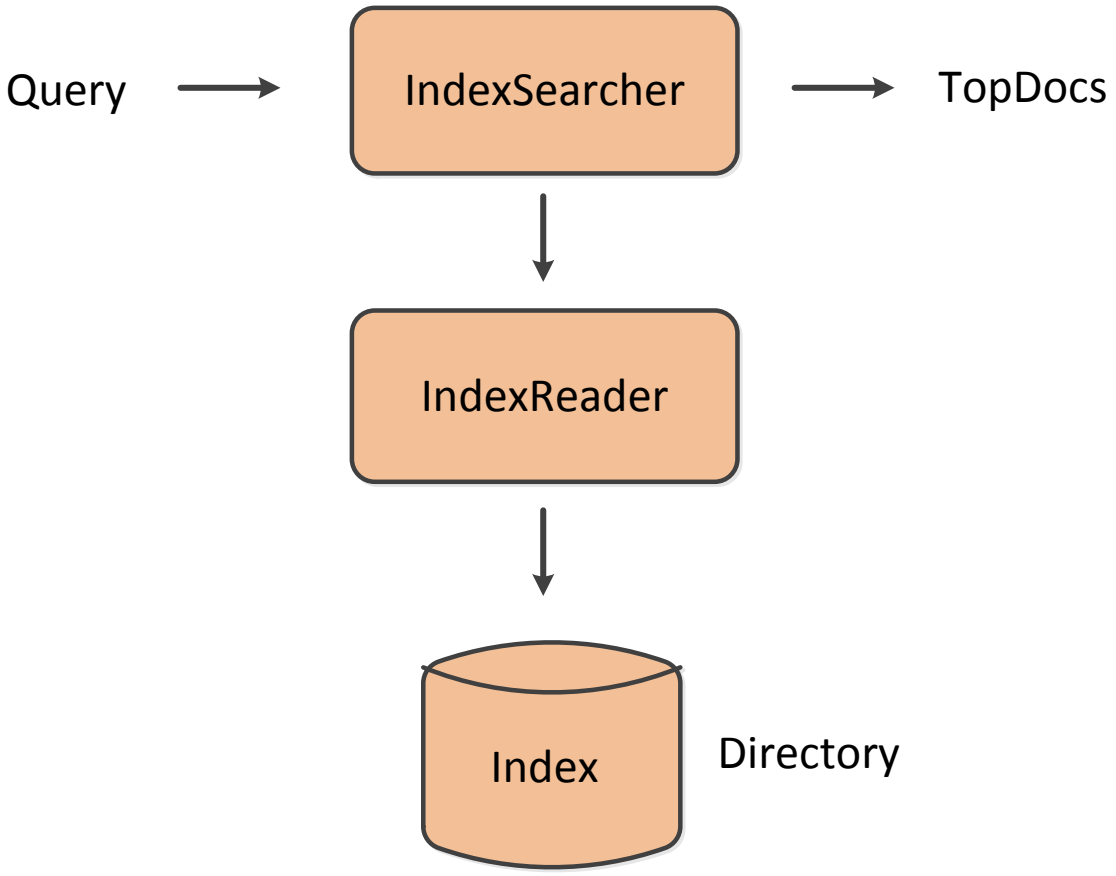
词库分词就是使用词库中定义的词来对字符串进行切分，这样的好处是分词更为准确，但是效率较N元分词更低，且难以识别互联网世界中层出不穷的新兴词汇。

Lucene索引构建过程



lucene索引的构建过程大致分为这样几个步骤，首先，通过指定的数据格式，将lucene的Document传递给分词器Analyzer进行分词，经过分词器分词之后，通过索引写入工具IndexWriter将索引写入到指定的目录。

Lucene索引搜索过程



索引的查询，大概可以分为如下几个步骤，首先，构建查询的Query，通过IndexSearcher进行查询，得到命中的TopDocs，然后通过TopDocs的scoreDocs()方法，拿到ScoreDoc，通过ScoreDoc，得到对应的文档编号，IndexSearcher通过文档编号，使用IndexReader对指定目录下的索引内容进行读取，得到命中的文档返回。

lucene的使用

索引优化

lucene的索引是由段(segment)组成的，每个段可能又包含多个索引文件，每个段包含了一个或者多个Document，段结构使得lucene可以很好的支持增量索引，新增的Document将被添加到新的索引段当中。但是，当越来越多的段被添加到索引当中，索引文件也就越来越多，一般来说，操作系统对于进程打开的文件句柄数是有限的，当一个进程打开太多的文件时，会抛出too many open files异常，并且，执行搜索任务时，lucene必须分别搜索每个段，然后将各个段的搜索结果合并，这样，查询的性能就会降低。

全量索引重建

因为前面说到我们产生索引碎片，而这些索引碎片即使进行了碎片合并而减少碎片数，但是一旦当碎片达到一定大小后就不适合继续进行合并，否则合并代价很大，所以我们无法避免的会因为碎片问题而导致更新实时性和查询QPS性能损耗问题。我们的解决的办法就是通过一段时间对具体业务全部源数据进行一次构建全量索引DUMP工作，用构建好的新的全量主索引去替换原来老的主索引和磁盘索引，从而让实时更新、搜索服务性能恢复到最佳。

Lucene的IndexReader和IndexWriter

Lucene的IndexReader和IndexWriter具有隔离性：

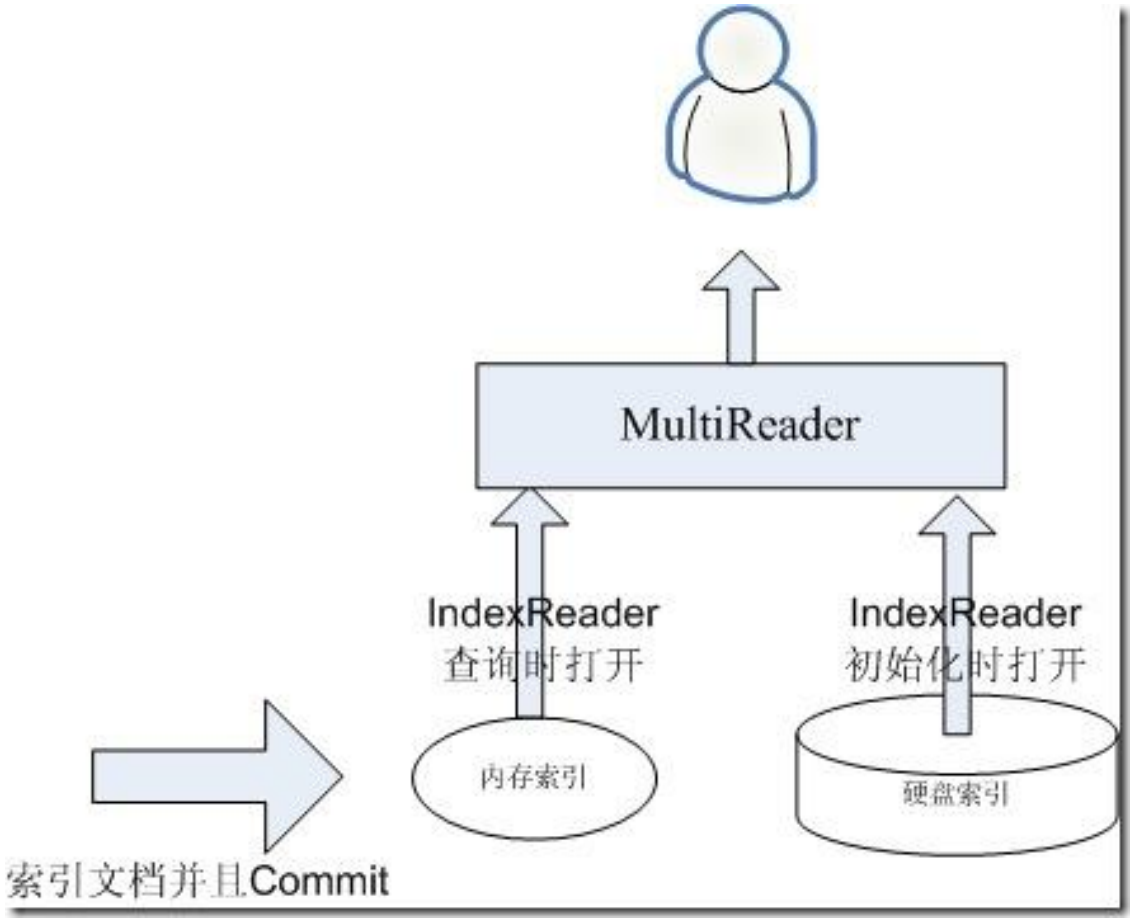
- 当IndexReader.open打开一个索引的时候，相对于给当前索引进行了一次snapshot，此后的任何修改都不会被看到
- 仅当IndexReader.open打开一个索引后，才有可能看到从上次打开后对索引的修改
- 当IndexWriter没有调用Commit的时候，其修改的内容是不能够被看到的，哪怕IndexReader被重新打开
- 欲使最新的修改被看到，一方面IndexWriter需要commit，一方面IndexReader重新打开

实时更新

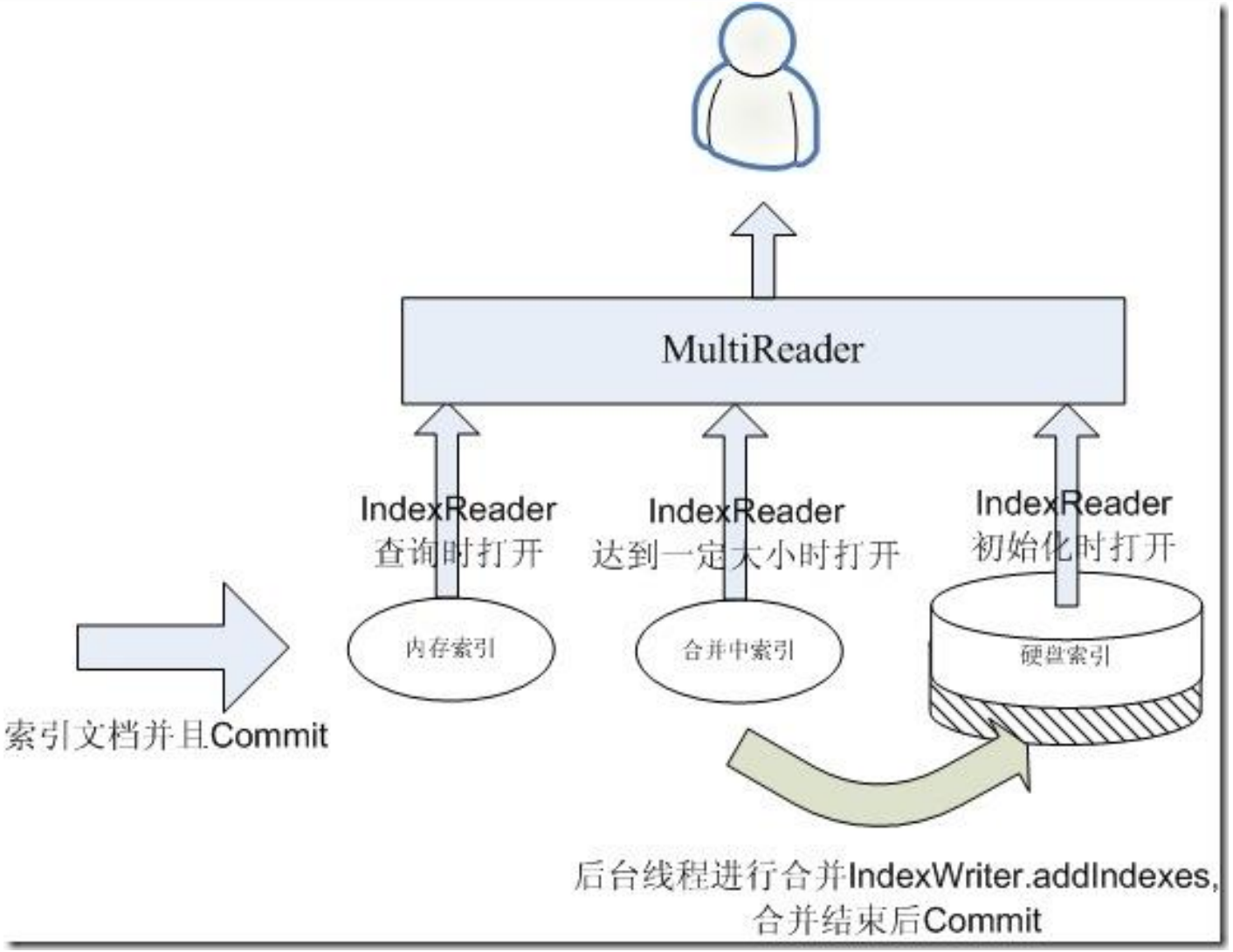
倒排索引是有一定的格式的，而这个格式一旦写入是非常难以改变，那么如何能够增量建索引呢？Lucene使用**段**这个概念解决了这个问题，对于每个已经生成的段，其倒排索引结构不会再改变，而增量添加的文档添加到新的段中，段之间在一定的时刻进行合并，从而形成新的倒排索引结构。

然而也正因为Lucene的索引读写的隔离性，使得Lucene的索引不够实时，如果想Lucene实时，则必须在新添文档后对IndexWriter进行commit，在搜索的时候IndexReader需要重新的打开，然而当索引在硬盘上的时候，尤其是索引非常大的时候，IndexWriter的commit操作和IndexReader的open操作都是非常慢的，根本达不到实时性的需要。

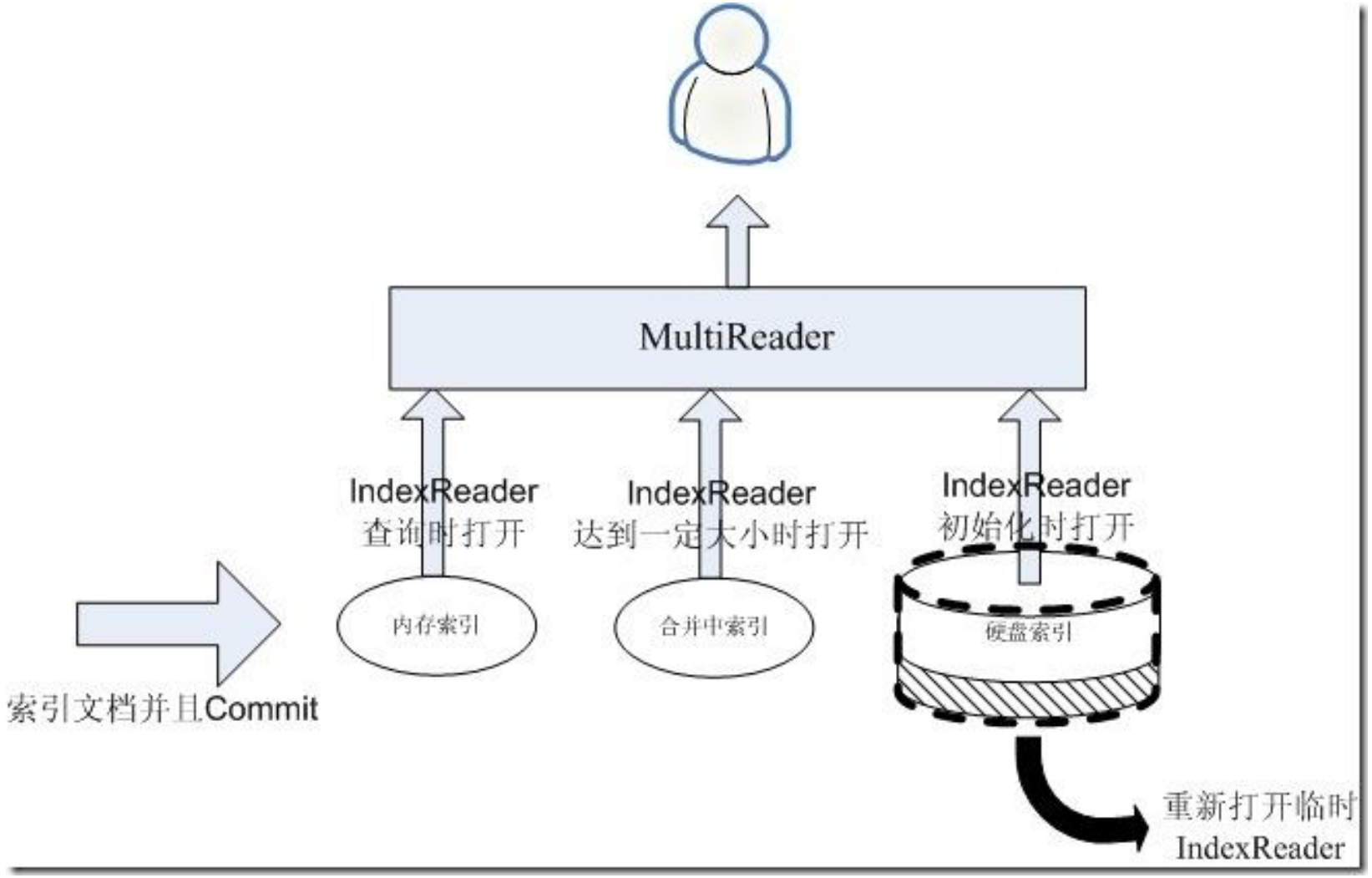
实时更新过程


MultiReader
IndexReader
查询时打开
IndexReader
初始化时打开
内存索引
硬盘索引
索引文档并且Commit

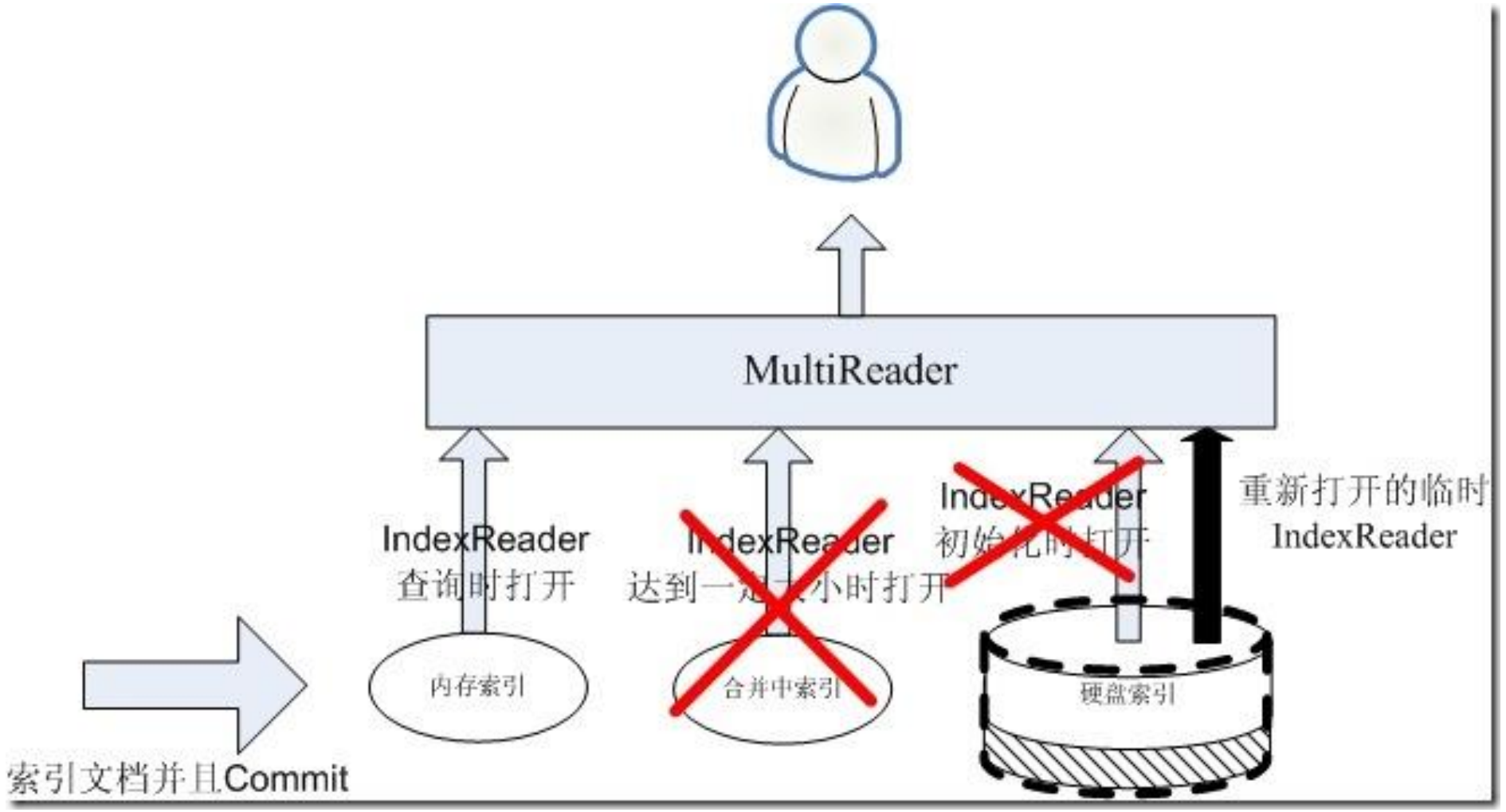
实时更新过程


MultiReader
IndexReader
查询时打开
IndexReader
达到一定大小时打开
IndexReader
初始化时打开
内存索引
合并中索引
硬盘索引
索引文档并且Commit
后台线程进行合并IndexWriter.addIndexes,
合并结束后Commit

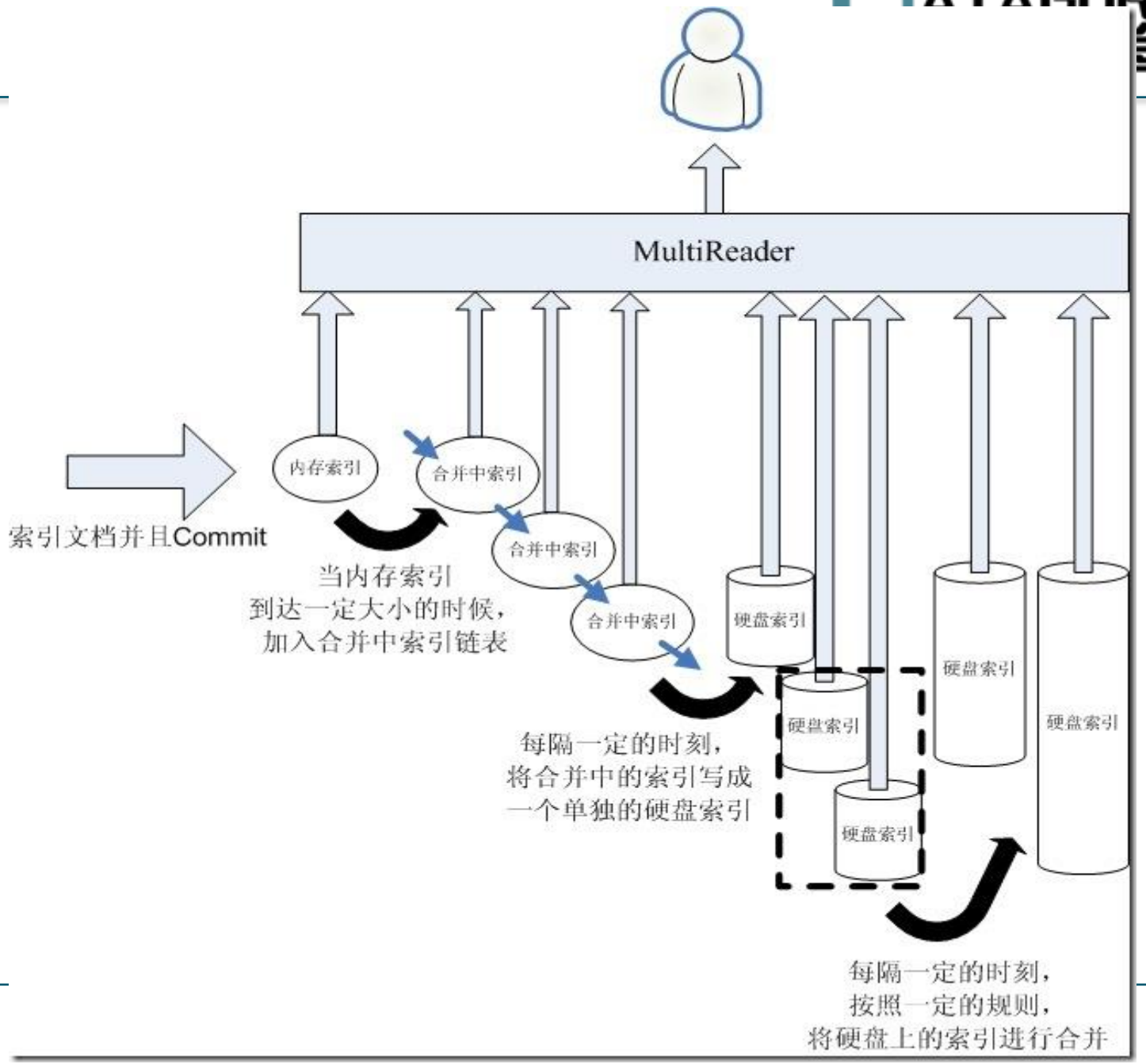
实时更新过程


MultiReader
IndexReader
查询时打开
IndexReader
达到一定大小时打开
IndexReader
初始化时打开
内存索引
合并中索引
硬盘索引
索引文档并且Commit
重新打开临时
IndexReader

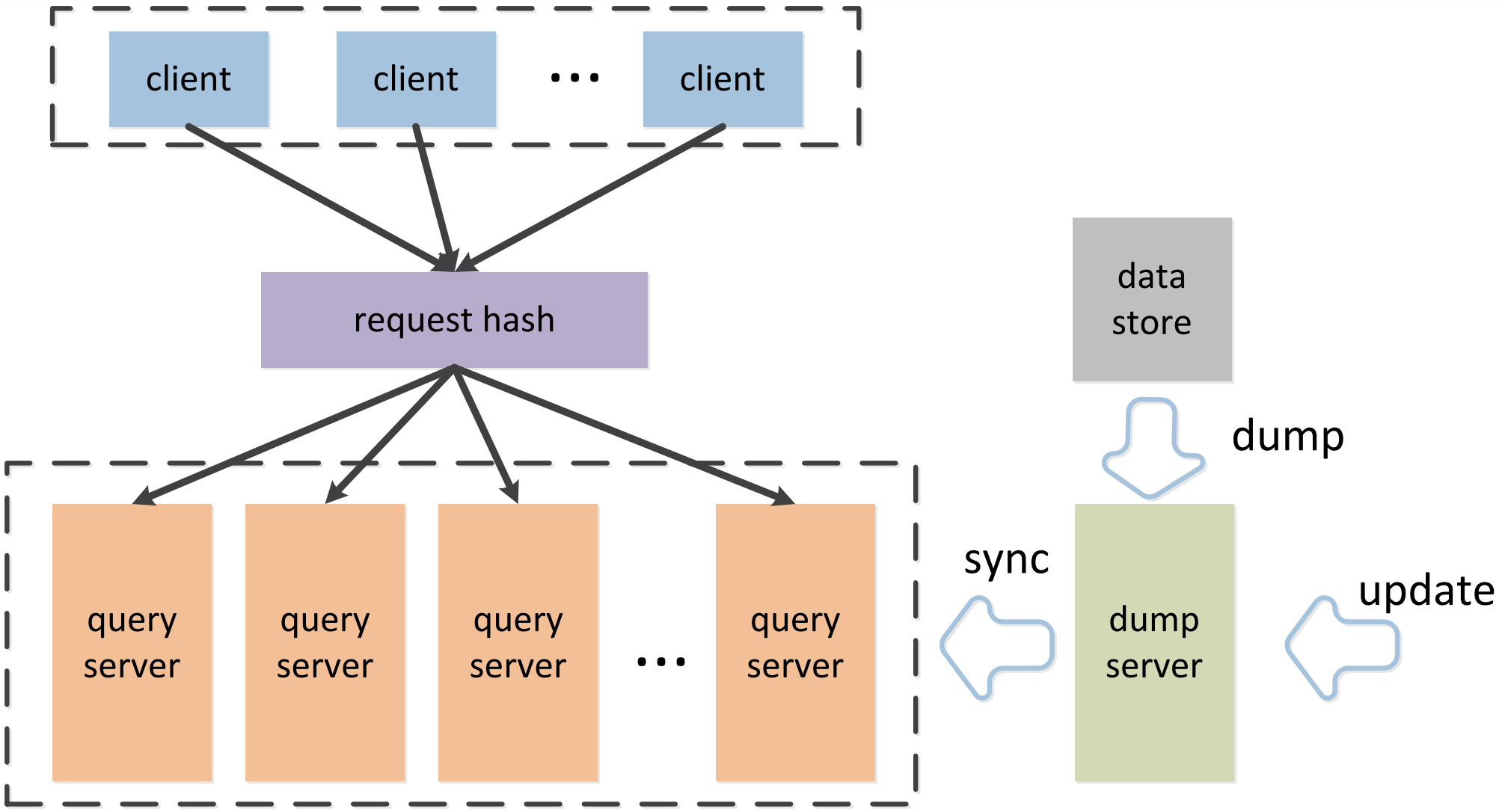
实时更新过程


MultiReader
IndexReader
查询时打开
IndexReader
达到一定大小时打开
IndexReader
初始化时打开
重新打开的临时
IndexReader
内存索引
合并中索引
硬盘索引
索引文档并且Commit

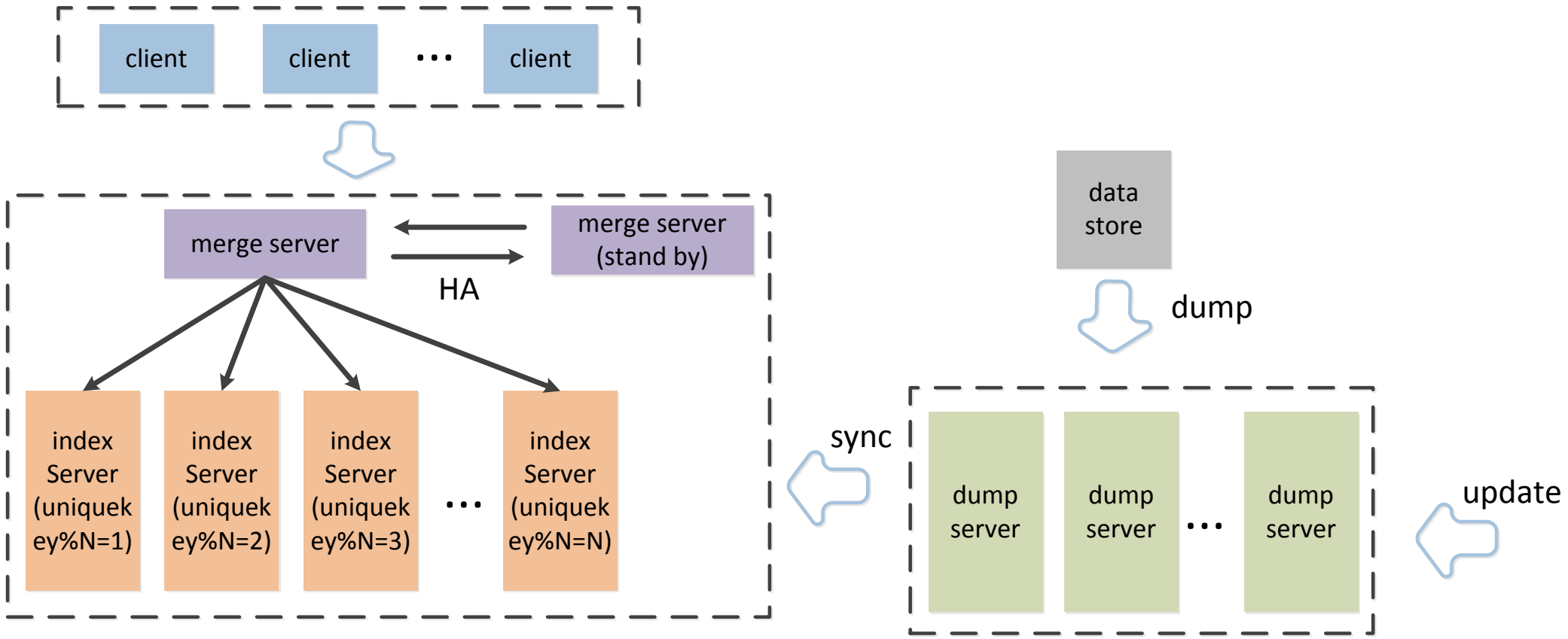
实时更新过程


MultiReader
索引文档并且Commit
内存索引
合并中索引
合并中索引
合并中索引
当内存索引
到达一定大小的时候，
加入合并中索引链表
硬盘索引
硬盘索引
硬盘索引
硬盘索引
硬盘索引
每隔一定的时刻，
将合并中的索引写成
一个单独的硬盘索引
每隔一定的时刻，
按照一定的规则，
将硬盘上的索引进行合并

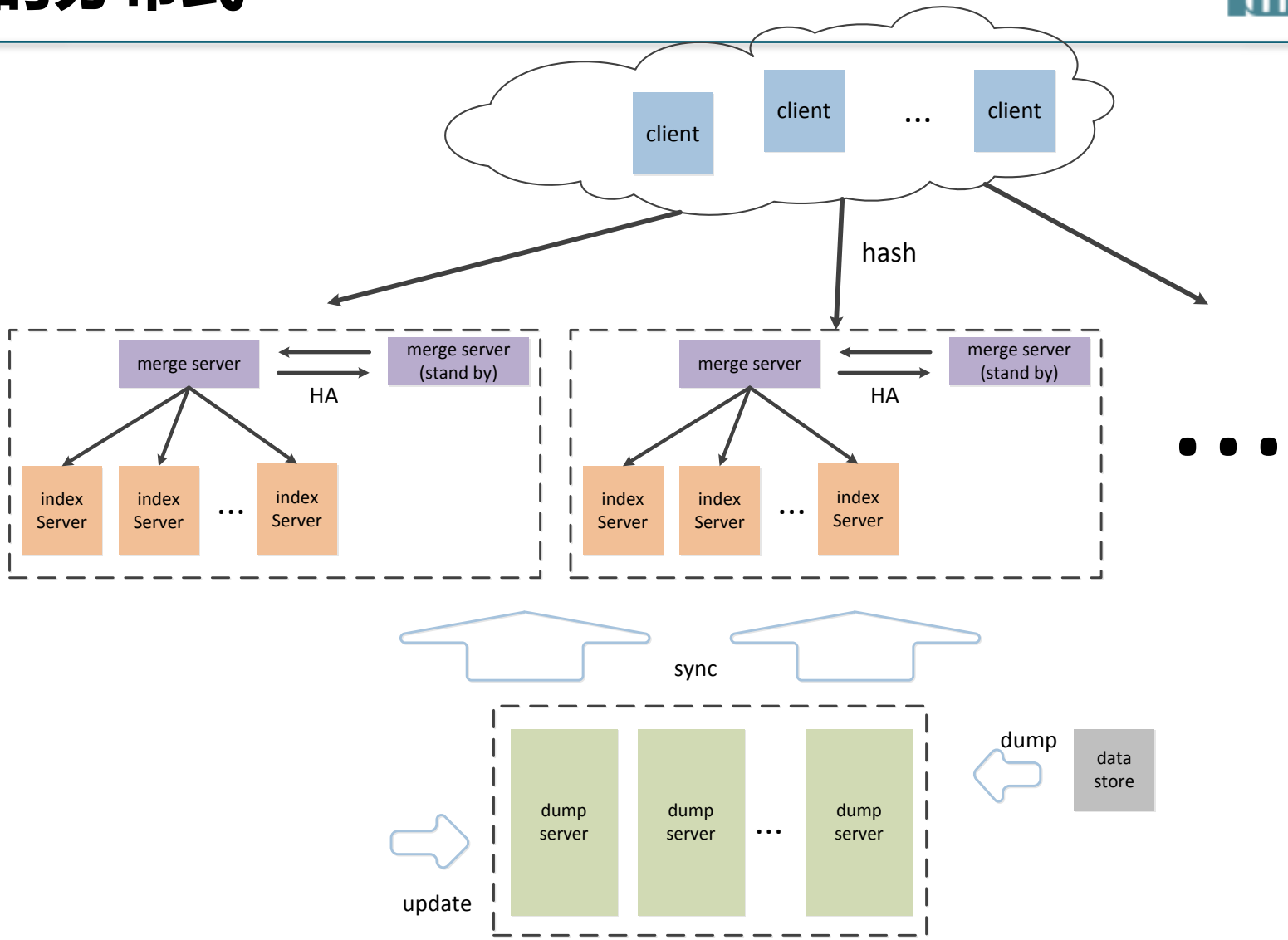
搜索引擎的分布式


client
client
client
request hash
data store
dump
query server
query server
query server
query server
sync
dump server
update

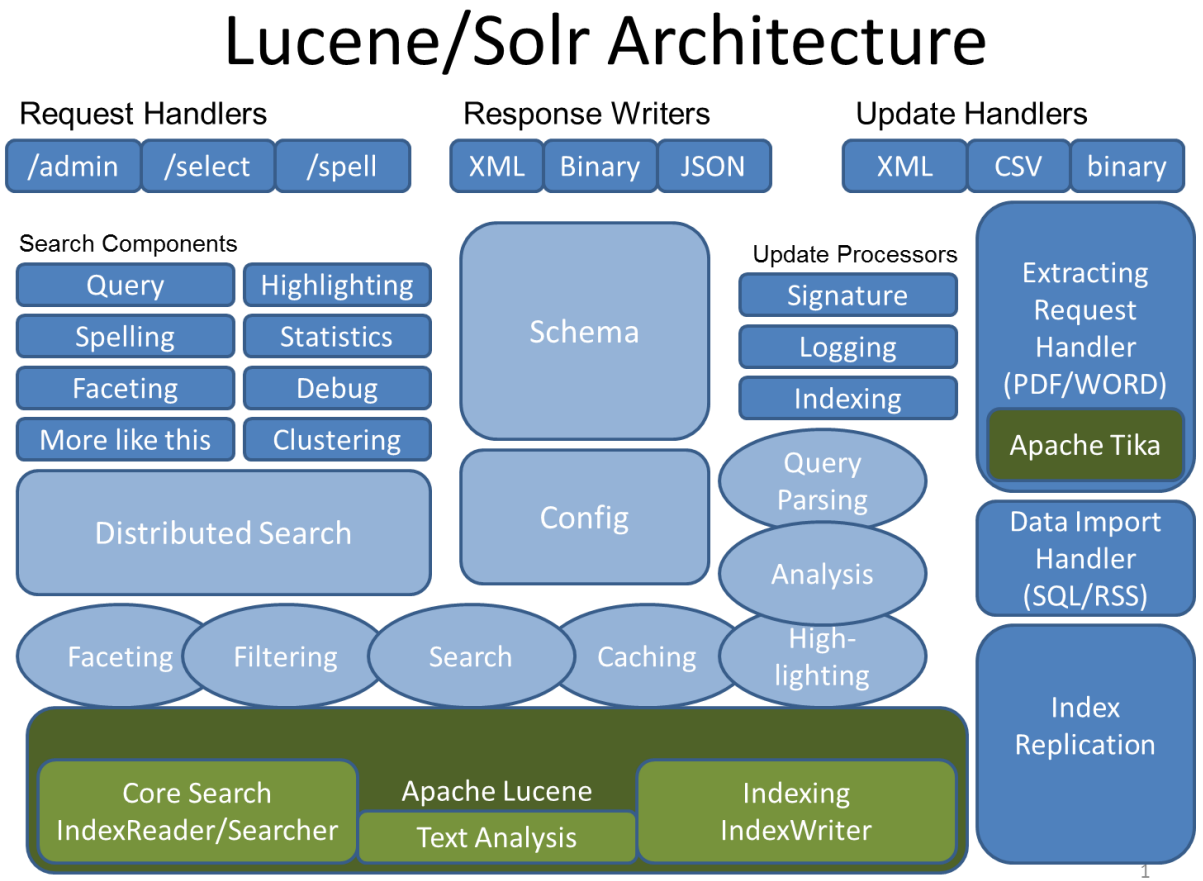
搜索引擎的分布式


client
client
client
merge server
merge server (stand by)
HA
index Server (uniquekey%N=1)
index Server (uniquekey%N=2)
index Server (uniquekey%N=3)
index Server (uniquekey%N=N)
data store
dump
sync
dump server
dump server
dump server
update

搜索引擎的分布式


client
client
...
client
hash
merge server
merge server (stand by)
HA
index Server
index Server
...
index Server
merge server
merge server (stand by)
HA
index Server
index Server
...
index Server
sync
dump server
dump server
...
dump server
update
dump
data store

使用solr的两大优势



1. 支持schema构建索引

2. 对复杂的动态的类SQL的查询条件的支持

3. Solr并不完美，还有很多地方需要改进

Thanks

FAQ时间