# 基于 NoSQL 技术的文档检索系统设计与实现


学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2007 级

姓　　名　兰　超

指导教师　张坤龙

**2011 年 6 月 15 日**

# 摘　要

NoSQL 技术随着 Web2.0 的兴起成了一个极其热门的新领域，并诞生了许多不同于传统关系型数据库的非关系型存储技术。这些 NoSQL 数据库提供的很多特性使之非常适合作为文档检索和存储的解决方案。论文工作分为两个部分，第一部分是基于 Apache MINA 和 Drools，设计并实现了一个轻量级中间组件 MO，实现了调用逻辑与具体实现分离，使系统具有高的可扩展性和与低的耦合度，上层应用通过统一的方式调用服务，用户可以方便地对现有底层功能修改、扩展、集成不同的应用组件，而不对系统造成大的影响。第二部分是基于 NoSQL 方式的存储技术，设计并实现了针对 PDF 等格式的文档的存储、索引和检索的系统。存储索引和检索的内容包括文档的全文和描述文档的元数据。对于全文采用建立倒排索引的方式支持全文检索，通过对象存储的方式存储索引和全文；对于文档的元数据，采用 XML 描述和存储，利用 XQuery 来对其查询。最后论文对系统性能做了测试，在索引和检索方面系统均达到了课题要求。

**关键词**：NoSQL；全文检索；倒排索引；文档检索

# ABSTRACT

NoSQL technology has become a new, active area with the rise of Web 2.0. A number of non-relational databases were designed, which are different from the traditional relational databases. The NoSQL databases provide a lot of features that makes them suitable as the solutions of a document retrieval and management system. Firstly, the paper, based on Apache MINA and Drools, design and implement a lightweight intermediate component MO. System using MO could achieve the separation of calling logic and function realization. So the system can easily be modified, expanded or integrate a set of different applications without effect other parts of the system. Calling of the services from the upper level are unified in a fixed interface. System can be highly scalable and loosely coupled. Secondly, the paper design and implement a system which could index, store, and retrieval documents in special formats such as PDF. Indexing and retrieving include both the full context and metadata that described in XML format of the document.

**Key words:** NoSQL; full-text search; Inverted index; document retrieval

# 目 录


第一章 绪论.....1

1.1 研究背景.....1

1.1.1 应用背景.....1

1.1.2 基于关系型数据库的文档存储检索.....1

1.1.3 全文检索技术.....2

1.1.4 XML 可扩展标记语言.....2

1.1.5 NoSQL 数据库技术.....3

1.2 研究目的及意义.....4

1.3 研究现状.....4

1.3.1 基于关系数据库的文档存储检索方案.....4

1.3.2 非关系型数据库解决方案.....4

1.3.3 信息检索技术.....5

1.4 本文结构.....5

第二章 相关算法和技术.....7

2.1 NoSQL 技术.....7

2.1.1 CAP 理论.....7

2.1.2 NoSQL 数据库.....9

2.2 基于关键词的全文检索.....10

2.2.1 文本索引查询模型.....11

2.2.2 相似度测量算法.....12

2.3 XML 数据的管理.....14

2.4 XQuery.....15

2.5 JSON 技术.....16

2.6 Apache MINA.....17

2.7 Drools 规则引擎.....17

2.8 IKAnalyzer 分词器.....18

2.9 本章小结.....18

第三章 可扩展系统框架.....19

3.1 整体设计.....20

3.2 数据交换格式方案.....21
3.3 MO 的设计与实现.....23
3.3.1 I/O 框架.....24
3.3.2 事件处理逻辑单元.....25
3.3.3 MO 运行流程.....27
3.4 本章小结.....28
第四章 文档检索服务.....29
4.1 全文检索.....29
4.1.1 文本抽取.....31
4.1.2 分词.....32
4.1.3 倒排索引.....34
4.1.4 存储.....38
4.1.5 关键词检索.....39
4.2 文档元数据检索.....39
4.3 本章小结.....41
第五章 实验结果及分析.....42
5.1 文档索引.....42
5.2 文档检索.....43
5.2.1 关键词检索测试.....43
5.2.2 元数据检索测试.....44
5.3 结果分析.....44
5.4 本章小结.....45
第六章 总结及展望.....46
6.1 本文工作总结.....46
6.2 课题展望.....46
参考文献.....48
外文资料
中文译文
致谢

# 第一章　绪论

## 1.1　研究背景

论文的目的是设计一个针对多种格式文档进行处理的可扩展的应用服务框架，并在框架中实现对文档的全文和元数据的存储，索引和检索等服务。首先需要设计一个系统框架，使得系统可以通过统一的方式对上层应用提供服务，按照统一协议接受请求和返回数据，并且要便于对现有服务进行修改和扩展，使得逻辑和具体实现分离。

### 1.1.1　应用背景

文档是人类智慧的资产。现实中有很多海量的文档管理的问题，如存储管理困难，查找缓慢，效率低下等。因此，需要有一种方法，除了能够把文档的内容合理存储以外，还可以对文档高效便捷的查询。这就需要开发者针对用户的需求，首先设计一种合理的系统架构，选择一种恰当的存储方式，然后合理的设计系统的检索功能。要设计合理的检索功能，最直接的功能应该是文档的全文检索，即用户可以通过提交关键词，系统计算并返回和用户查询相关的文档。同时，文档除了正文之外，包括很多其他信息，如标题，出处，日期等等。用户也许希望细化他们查询，对文档的不同部分做不同的限定，例如用户可能希望查询 2010 年发表于某期刊的某一主题的文章，这是简单的全文检索无法做到的。但是这种查询可以以全文检索为基础，对其功能进行扩展，提供文档的元数据查询，来满足用户的这种需求，同时保证检索的效率。

### 1.1.2　基于关系型数据库的文档存储检索

传统的关系型数据库是由 IBM 的 E. F. Codd 在 1970 年发明的，Codd 在论文[1]中提出了关系型数据库这个概念，并且做出了关系型数据库的定义以及著名的 Codd’ s 12 rules。简单来说，关系型数据库就是数据被以一系列严格形式化描述的表的形式组织起来，这些表能够被通过多种方式访问或重组而不破坏原有的数据库的组织形式。所以关系型数据库就是将数据呈现为关系，而且能够对数据进行关系运算。这些特点使得关系型数据库成为了数据库的主流，而且至今被广泛采用。关系型数据库的检索和存储是基于关系运算，存储数据首先要定义表完整的 Schema，所以能够对具有完整 Schema 结构化的数据提供高效的查询性能。Schema 定义好了之后，加载数据，然后定义合适的索引，这样 SQL 语言就能够提供出色的查询性能。但是，关系型数据库有几项先天不足[1]，比如‘Schema first, data later’的方法缺乏灵活性，用户往往无法使用这种方法

来存储和管理规模较大的非结构化数据或者是结构比较奇异的数据。即使数据是结构化的，设计数据的 Schema 仍然是一个艰巨的而且是容易出现纰漏的过程。所以，对于大规模的文档类型的数据，通常的做法都是把这些数据存放在 LOB 列中。为了让关系型数据库能够对这些数据进行检索，一般还要借鉴信息检索的方法，在这部分数据之上建立一个文档的倒排索引[2]，以加速信息检索风格的基于关键词的检索[3]。在现在的 SQL 语句中，提供了一个 CONTAINS() 语句[4]，可以提供对目标列的关键词检索。

### 1.1.3 全文检索技术

在过去的十几年中，文本搜索引擎获得了巨大的发展。一系列新的索引表示方法推动了包括 index 存储，index 构建和 query 评估等一个广泛领域的不断创新。文本检索是文本搜索引擎的核心技术，也是当前整个信息检索领域的核心技术。搜索引擎索引整个 Web 互联网，提供了一种快速便捷的访问信息的方式，而这在十年前还是不可想象的。而且文本检索还在其他的方面起到了关键作用，比如在桌面操作系统中为很多应用提供了高效的文本搜索，为普通用户提供桌面搜索，帮助他们在 PC 上快速定位文档。

搜索引擎在结构上很类似与数据库系统，文档被存储在仓库里，系统维护一个文档的索引。用户向系统提交查询，系统处理用户查询，找到匹配的项，然后返回给用户。但是，二者有很多不同的地方。数据库系统必须构建比较复杂的结构化的查询，而大部分搜索引擎接受的查询是一些短语和词组的结合。在数据库系统中，“匹配”指的是一条记录满足特定的逻辑条件；在搜索引擎中，“匹配”指的是文档从统计规律上满足条件，甚至文档都不需要包含所有的查询关键词。数据库系统要返回所有满足条件的结果，而搜索引擎返回一定数量的满足条件的通过相似度排序的结果。数据库系统对每条记录都提供一个主键，可以通过主键来访问该记录，而对于搜索引擎来讲，每一个查询都可能有成千上万条记录和其相关度大于 0。所以，尽管搜索引擎无需做类似数据库的连接等操作，但要对查询达到很高的响应速度是非常困难的，特别是在当关键词要检索涉及大量的文档，每篇文档中又包含有大量的短语的情况下。

搜索引擎面临的挑战促使了一系列新的算法和数据结构的诞生，包括文本索引的表示，文本索引的构建技术，文本索引算法的评估等等。主流的 Web 搜索引擎提供的基于这些技术的索引在快速响应方面起到了关键的作用。通过合理的压缩和组织结构，索引所需的空间和磁盘的负载以及检索时消耗的系统资源已经降低到了和以前相比很小的一部分。

### 1.1.4 XML 可扩展标记语言

在 1998 年 2 月，XML 被引入软件工业界，他给整个行业带来了一场革命。XML 提供了一种用来结构化文档和数据的通用且适应性强的格式，它不仅仅可以用在 Web 上，更可以推广在几乎任何地方。XML 的基本思想是用标记表示数据的意义而不是像 HTML 仅用标签表示数据的显示方式。将内容和形式相分离是 XML 的重要特点，数据以 XML 的形式编码使得 Web 服务和应用程序以一种简单有效的格式提供信息，这简化了程序服务之间的交互过程。并且 XML 不采用固定的标记集合，用户可以根据需要定义任何一种标签来描述文档的数据元素。

XML 具有四个主要特点。

- 简单性
- 扩展性
- 操作性
- 开放性

现在的数据大多数还都存储在关系型数据库中，但是正如前文提到的，关系型数据库对于非结构化数据处理方式不够灵活，而 XML 处理数据的灵活方式使得其在处理非结构化数据上有着先天的优势，所以关系型数据库逐渐开始使用 XML 来存储和管理非结构化数据。现在，在 SQL/XML 标准中 XML 类型还被作为关系型数据库数据的标准类型，半结构化或者没有 Schema 的数据可以被存储到这种 XML 类型列当中，并通过 XQuery 来查询相关的数据[5]。

### 1.1.5 NoSQL 数据库技术

现代互联网飞速的发展带来了海量的数据，交互式的在线服务的需求不断发展，用户不断的增多，这要求应用必须高度的可扩展，如果采用关系型数据库，如 MySQL，应用可以快速的开发出来，但是当用户规模扩大到数以百万计的时候，就需要重新设计基础存储结构。而且应用必须面向用户，开发周期必须短，投入市场必须快。最重要的是，服务必须响应迅速。所以，存储系统必须低延迟。同时，数据必须对用户来说是持久可见的，用户对数据的修改必须立刻体现并且持久存在。最后，这些服务必须保证高可用，用户可以 24/7 获得服务。

为了满足上述的需求，关系型数据库提供了很多方便的功能来满足应用的需求。但是关系数据库很难扩展到数以亿计的用户规模。NoSQL 数据库例如 Google 的 BigTable[6]，Apache Hadoop HBase[7]，Facebook Cassandra[8]等等都提供了高可扩展性，适合远距离传递数据，而且保证低延迟。而且现在的 NoSQL 数据库如 Google 的 MetaStore[9]除了提供高可扩展性以外，还对 NoSQL 数据库

做了很多关系型数据库的特点，采用同步数据复制来提供高可用性，采用统一的数据视图，而且对远距离低延迟传输支持完整的 ACID 模式，来方便交互式应用的开发。

## 1.2 研究目的及意义

论文的目的是设计一个针对多种格式文档进行处理的可扩展的应用服务框架，并在框架中实现对文档的全文和元数据的存储，索引和检索等服务。首先需要设计一个系统框架，使得系统可以通过统一的方式对上层应用提供服务，按照统一协议接受请求和返回数据，并且要便于对现有服务进行修改和扩展，使得逻辑和具体实现分离。其次，需要在设计好的框架内实现具体的服务。目前需要实现的服务包括对多种格式（如 PDF，WORD）等文档进行内容抽取，全文检索。同时，系统要能够对用 XML 描述的文档的原数据进行存储，管理和查询等，实现同时支持文档的全文检索和元数据查询。

## 1.3 研究现状

### 1.3.1 基于关系数据库的文档存储检索方案

文档属于非结构化数据，对于存储在关系型数据库中的非结构化数据，目前的主流数据库提供商（IBM[10], Microsoft[11],Oracle[12]）都实现在 SQL 的关键词查询 SQL CONTAINS()扩展，支持指定内容的全文检索。并且也支持 XQuery 的 contains()。在 XML 出现之后，其结构和内容分离的形式使其非常适合作为描述类似文档的非结构化数据的基础。文章[13]提出了可以将关键词检索集成到 XML 检索语言当中去。在 XQuery 规范中[14]，基于文本内容的关键词检索是支持的，用户可以显式的在文档内部特定的部分来检索特定的关键词。这样，用户就可以提交类似 ‘ *$document//footnote contains “xml”* ‘的查询。这样就可以根据用户的需求只在文档的页脚中检索”*xml*”一词。这种形式的检索不仅能够向经典的全文检索一样返回文档和关键词的位置，而且能够返回进一步进行 XQuery 所需要的结点信息。这种检索从能力上讲远远超过了传统的信息检索，它使得关键词检索，结果过滤和传递融为一体，完成了文章[15]中描述的愿景。

### 1.3.2 非关系型数据库解决方案

此外，近年来热门的 NoSQL 技术是另一个研究的热点。NoSQL 数据库的出发点是：非结构化，分布式，开源，横向可扩展。NoSQL 的最初目的是做现代网络规模的数据库。从 2009 年至今，NoSQL 在迅速的发展着，同时越来越多的特点被考虑进来：无需 schema，便于数据迁移，简单的 API，海量的数据

存储。迄今为止，已经有很多成功的 NoSQL 数据库出现，并且已经被很多大型网络公司所采用，如列存储的 Hadoop/HBase[7]，Amazon SimpleDB[16]，文档存储的 MongoDB[17]，对象数据库 Versant[18]，XML 数据库 Sedna[19]等等。例如 Google Megastore[20]便是其中之一，其突出特点有：

- Megastore 建立在 Google 的 BigTable 之上，支持单一数据中心的容错存储；
- 异步的数据复制，对远距离数据传输做优化；
- 将数据分布在大量小的数据库之中，每一个小的数据库都有独立的 log；
- 支持一致性；
- 层次化的表结构；
- 出色的备份和恢复能力等。

所以未来的海量数据存储可以脱离开关系型数据库的种种约束，转而采用 NoSQL 的思想去解决问题。

### 1.3.3 信息检索技术

信息检索是研究文档搜索的研究领域，主要包括文档内部信息，以及和文档相关的原信息(metadata)的检索。第一个自动信息检索系统出现在 1950 年到 1060 年，1970 年是的信息检索系统已经很够在数量巨大的文档中获得很好的检索效率了。1992 年，美国 Department of Defense 和 NIST 联合开始了 Text Retrieved Conference(TREC)会议，旨在促进信息检索系统的发展。互联网时代，网络当中的文档数量庞大，如何有效的抽取和检索成为了网络的重中之重。互联网时代的系统大都采用分布式管理，资源的为数字表达，多媒体化，载体多种多样。资源涵盖了社会各个领域，分布式无序，缺少统一的规范和结构，特征抽取复杂，这导致了很多信息检索方法的转变，促成了新型的 Web 搜索的出现，如 Yahoo！和 Google。

目前，自然语言处理技术不断发展，以及语义 Web 的不断研究，标识这信息检索将要进入一个新的阶段。目前信息检索技术正往两个方向发展，一是从传统的文本检索向多媒体检索发展，提高传统文本检索的智能化，自动化，对新的多媒体内容进行检索。二是信息资源集中向网络化和分布化发展。面对 Internet 上的海量数据，如何有效的存储和检索，提高准确率，也是当前研究的热点。

## 1.4 本文结构

本文的绪论介绍了研究背景，研究目标和意义以及研究现状。第二章介绍了 NoSQL 数据库等论文使用的技术和算法。第三章介绍了系统的框架设计实

现方案。第四章介绍系统具体业务逻辑的设计和实现方案。第五章介绍了实现系统的测试实验结果。第六章对全文总结并提出了下一步工作。

# 第二章　相关算法和技术

目前存在许多已经成熟的技术，对于实现基于 NoSQL 技术的文档检索系统十分关键，本章就这些技术做简要介绍。

## 2.1　NoSQL 技术

随着互联网的不断发展，信息量正在以几何级数增长，其中非结构化信息所占的比重也越来越高。现代计算机处理的信息已经由原来的结构化数据逐渐转变为结构化，半结构化和非结构化并存的海量数据。因此，如何对非结构化数据进行更好的管理是一个越来越重要的问题。传统的擅长处理结构化数据的关系型数据库在面对海量的非结构化数据是已经显现出诸多的局限，开始制约系统对于数据规模的需求，而全文检索数据库系统在处理海量的信息检索上具有优势，但是在处理结构化数据，做关系运算时存在很多不足，难以满足复杂环境下的应用对数据存取的要求。因此，如何通过 NoSQL 数据库结合二者的优势去改进现有的非结构化数据存储检索方案显得尤为迫切。

### 2.1.1　CAP 理论

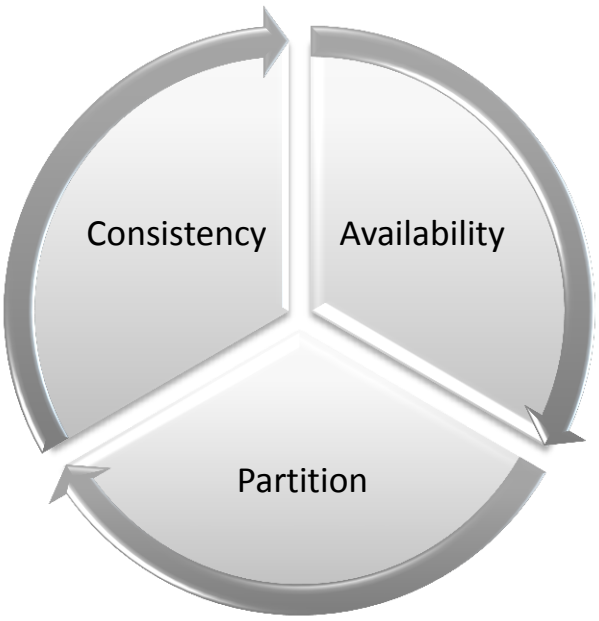



图 2-1　CAP 示意图

- C: Consistency 一致性
- A: Availability 可用性（快速获取数据）
- P: Tolerance of network Partition 分区容忍性（分布式）

Eric Brewer 教授在十年前提出了著名的 CAP 理论，后来 Seth Gilbert 和

Nancy Lynch 两人证明了 CAP 理论的正确性。CAP 理论指出一个分布式系统不可能同时满足一致性，可用性和分区容忍性三个特点。在设计系统的时候，就要做出取舍，最多只能同时满足两个。如果关注高一致性，那就需要处理系统因为低可用性而造成的写操作失败的情况。而如果关注高可用性的话，就需要关注可能无法读取系统最新的数据。因此，根据系统不同的关注点，需要采取的策略也不同，真正理解了系统的需求才是关键的问题所在，剩下的就是对 CAP 理论进行应用，做好取舍工作，处理好系统可能会因此出现的问题。为了详细的说明这个问题，构造表 2-1 场景。

表 2-1　场景说明

| 符号 | 说明 |
|---|---|
| N | 同一份数据被复制的节点数 |
| R | 读取一份数据需要访问的节点数 |
| W | 写入一份新的数据操作结束前需要访问的结点数 |

为了保证强一致性，R + W > N 必须被满足，这样才能保证读操作总是从至少一个已经存储了当前值的结点读取数据。如图 2-2 所示，N = 3，R = 2，且 W = 2。那么新的值 x2 被写到两个结点中，并且被从两个结点中读，那么结果最少也要覆盖一个结点，所以读操作一定能够访问到最新的数据，得到一致的结果。

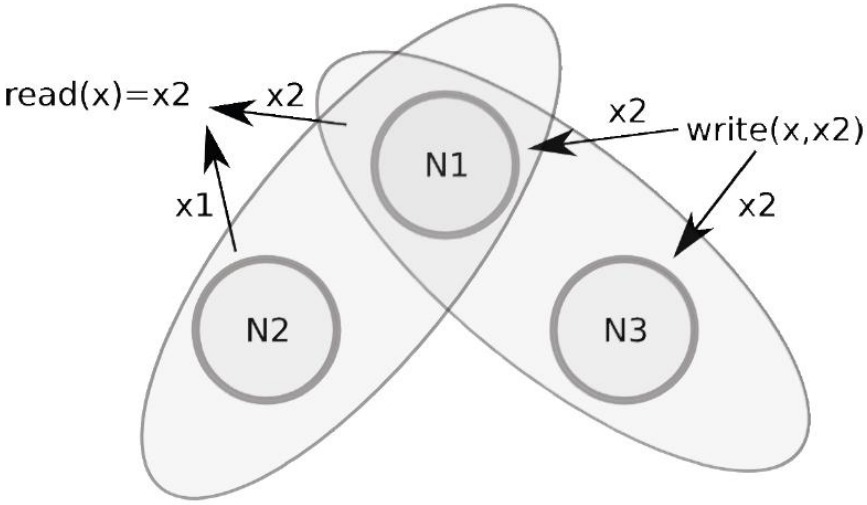



图 2-2　强一致性示例

### 2.1.2 NoSQL 数据库

在过去，关系型数据库几乎被用在了所有领域，由于其丰富完善的功能，出色的 Query 能力和完善的事务管理，几乎所有的任务都能够用关系型数据库完成。但是这些优点同样构成了关系型数据库的局限，因为它使得关系型数据库特别难以做成分布式存储和管理。特别是在事务管理和 Join 运算时，分布式会效率很低。现在出现了很多 NoSQL 数据库，它们有有限的功能集，不完整的 ACID 支持，但是特别适合用在分布式的环境下。这些数据库当前被称为 NoSQL 数据库。从 Google 趋势图 2-4 来看，NoSQL 从 2009 年 4 月出现以来已经成为了一个热点。

本课题需要设计和使用 NoSQL 数据库，具体来讲，NoSQL 数据库和关系型数据库的关系可以如图 2-3 所示。

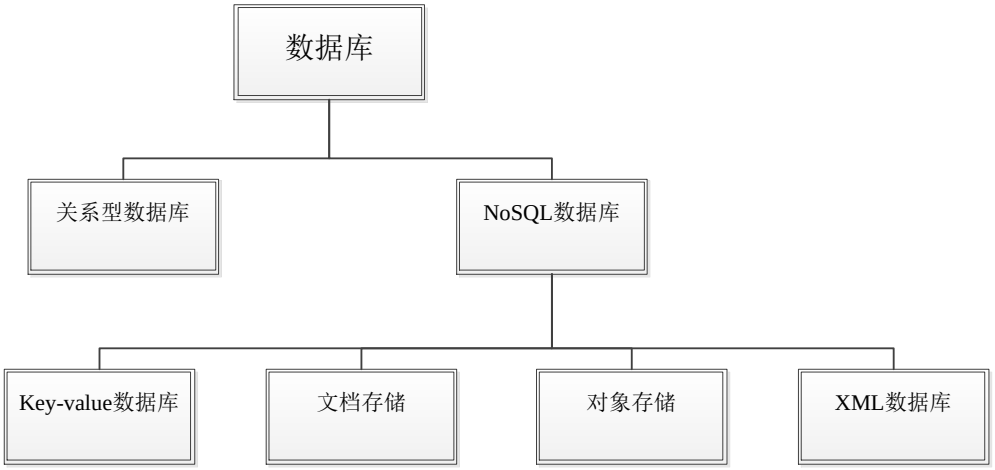



图 2-3　数据库分类

NoSQL 打破了传统关系型数据库模式，这种数据库无需在存储数据之前实现定义好表的结构，也无需出现表的连接和分割。NoSQL 出现的根本原因在于应用对数据的可扩展性的要求。可扩展性在 Web 应用里指的是，如果应用可以运行在足够多的服务器上，那么在特定时间内，任意多的数据或工作负载都能被处理。在实际应用环境下，很多情况都要面对高可扩展性的要求。比如 Amazon 提出了 SOA 的概念，即 Service Oriented Architecture 来保证应用层能够扩展。Amazon 将服务分类，不同的服务采用不同的存储模式。比如对于一致性要求高的数据，如交易数据，SOA 将它们放在 RDBMS(关系型数据库)里，而其他的数据则可以放在分布式存储的 NoSQL 数据库里，如 Amazon 的 Dynamo，S3 或者 SimpleDB。

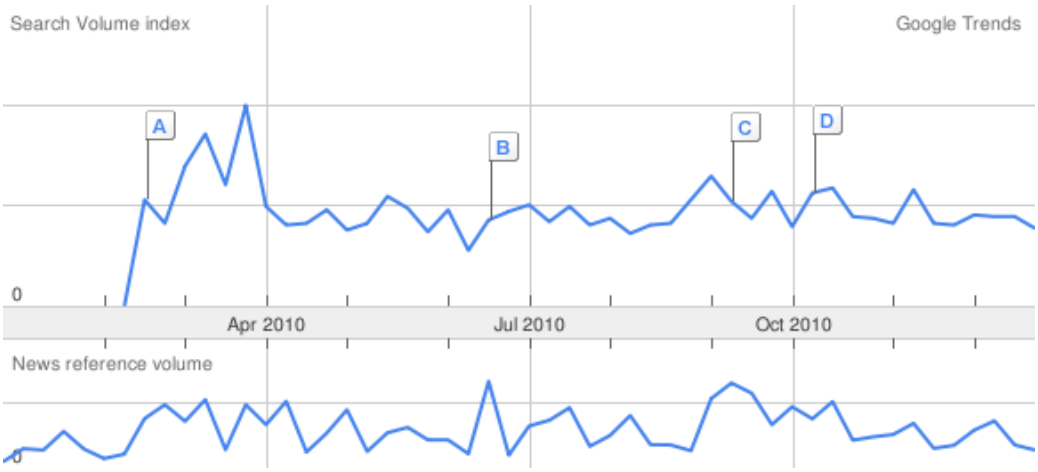



图 2-4　NoSQL 数据库热度

论文使用的 NoSQL 数据库有两个，Joafip 和 Sedna。Joafip 是一种基于 NoSQL 思想的对象存储工具。能够将 Java 数据对象持久化到文件系统而不使用数据库。Joafip 的优点有如下。

- 容易使用
- 保证数据库 ACID 特性
- 面向对象，直接存储对象
- 支持多线程访问数据
- 将所有的对象存储在文件系统中，通过堆文件来记录对象实例

Joafip 的以简单的 key-object 形式存储对象，在获取对象时同样通过 key 直接查找，非常便捷，适合作为本系统的全文索引部分的存储系统。

Sedna 是一个开源的免费原生 XML 数据库，提供了全功能的核心数据库功能，包括持久化存储，ACID 事务，索引，安全，热备等功能。Sedna 实现了 W3C 的 XQuery 规范，支持 XQuery 查询。

## 2.2　基于关键词的全文检索

在过去的十几年中，文本搜索引擎获得了巨大的发展。一系列新的索引表示方法推动了包括 index 存储，index 构建和 query 评估等一个广泛领域的不断创新。文本检索是文本搜索引擎的核心技术，也是当前整个信息检索领域的核心技术。搜索引擎索引整个 Web 互联网，提供了一种快速便捷的访问信息的方式，而这在十年前还是不可想象的。而且文本检索还在其他的方面起到了关键作用，比如在操作系统中为很多应用提供了高效的文本搜索，为桌面用户提供桌面搜索，以帮助用户在 PC 上快速定位他们的文件。

搜索引擎在结构上很类似与数据库系统，文档被存储在仓库里，系统维护

一个文档的索引。用户向系统提交查询，系统处理用户查询，找到匹配的项，然后返回给用户。但是，二者有很多不同的地方。数据库系统必须构建比较复杂的结构化的查询，而大部分的搜索引擎接受的查询是一些短语和词组的结合。在数据库系统中，“匹配”指的是一条记录满足特定的逻辑条件；在搜索引擎中，“匹配”指的是文档从统计规律上满足条件，甚至文档都不需要包含所有的查询关键词。数据库系统要返回所有满足条件的结果，而搜索引擎返回一定数量的满足条件的通过相似度排序的结果。数据库系统对每条记录都提供一个主键，可以通过主键来访问该记录，而对于搜索引擎来讲，每一个查询都可能有成千上万条记录和其相关度大于 0。所以，尽管搜索引擎无需做类似数据库的连接等操作，要达到很高的响应速度是非常困难的，特别是关键词要检索的是大量的文档，每篇文档中又包含有大量的短语。

因此，搜索引擎面对临的挑战促使了一系列新的算法和数据结构的诞生，包括文本索引的表示，文本索引的构建技术，文本索引算法的评估等等。主流的 Web 搜索引擎提供的基于这些技术的索引在快速响应方面起到了关键的作用。通过合理的压缩和组织结构，索引所需的空间和磁盘的负载以及检索时消耗的系统资源已经降低到了和以前相比很小的一部分。

### 2.2.1 文本索引查询模型

典型的文本内容包括网页，报纸，学术出版物，公司财报等。这些文本材料从内容和大小上看差异很大，例如一个科技工作者十年左右写的论文如果用纯文本来计算可能有 10 兆字节左右，十年之内的邮件可能有 100 兆字节左右。一个小型的大学图书馆的藏书用纯文本来结算可能有 100G 字节左右，而截至到 2005 年互联网上的文本内容已经超过了 100T 字节。面对这么大规模的数据，就必须采取合适的查询模型才能满足快速响应且结果准确的要求。

传统数据库的主要检索方式是通过 key 或者记录标示。这种搜索很少用在文本数据库里。文本数据具有结构化属性，例如作者标签和其他元数据，比如学科，分类信息。但是这些信息只在基于内容的检索中有用，并不能用在像关系数据库中的 key 那样使用。

主流的文本检索模型是检索满足信息需求的内容。用户通常可以用很多不同的方式检索所需的信息，一般首先提交一个初始的查询，然后不断的细化查询，直至系统返回正确的结果。如果查询的某一步不能返回正确的结果或者是范围，用户就需要修改查询的限定，来重新执行查询过程。这个查询过程中，信息需求是被体现在用户提交的查询词上，用户可能多次提交信息需求，直到查询到用户认为满意的结果。

信息检索系统另一个有别于数据库系统的是其对匹配的定义，在文本检索中，匹配是指用户认为其相关。但是相关性是一个不准确的概念，因为可能一篇文档根本不包含用户提交的关键词，但是内容可能是高度相关的。用户可能只能从系统返回的很多结果中找到一部分自己认为是相关的结果，而且不同的系统可能返回的结果很不相同。所以，检索系统定义了一个概念叫有效性：系统的有效性是指系统返回的前 r 个结果有效的结果占的比重。不同的系统对准确率和效率的权衡取舍都是不同的。

系统的有效性有很多不同的衡量方法，其中最常用的两种是精确率和召回率。分别指的是返回结果中有效的结果占得比重和返回的有效结果占所有有效结果的比重。系统对于有效结果的鉴别主要是通过内部相似度来衡量的。

### 2.2.2 相似度测量算法

目前的搜索引擎都是通过对结果的排序来衡量潜在的答案的。在一个排序的查询中，相似度是通过一个统计的相似度得出的，统计相似度用来衡量查询词句和每个文本之间的近似程度。所以，统计得出的相似值越大，系统就认为这篇文档更加相关。下面通过一个相似度模型来具体说明。

规定表 2-2 的统计值。

表 2-2　符号说明

| 符号 | 含义 |
|---|---|
| $f_{d,t}$ | 短语 t 在文档 d 中出现的频率 |
| $f_{q,t}$ | 短语 t 在 query 中出现的频率 |
| $f_t$ | 包含短语 t 的文档数量 |
| $F_t$ | 短语 t 在集合中出现的总数 |
| $N$ | 集合中文档的总数 |
| $n$ | 集合中索引的短语总数 |

以上的统计值要满足如下的规则：

1、在很多文档都出现的短语要给予更少的权重。

2、在一篇文档中出现很多次的短语要给予更多的权重。

3、包含很多短语的文档要给予更少的权重。

典型的计算相似度的方法是计算在 n 维向量空间中 query 向量$< W_{q,t} >$和文档向量$< W_{d,t} >$的 cosine 值。

$$w_{q,t} = \ln\left(1 + \frac{N}{f_t}\right) \quad (2\text{-}1)$$

$$w_{d,t} = 1 + \ln f_{d,t} \quad (2\text{-}2)$$

$$W_d = \sqrt{\sum_t w_{d,t}^2} \quad (2\text{-}3)$$

$$W_q = \sqrt{\sum_t w_{q,t}^2} \quad (2\text{-}4)$$

$$S_{q,d} = \frac{\sum_t w_{d,t} \cdot w_{q,t}}{W_d \cdot W_q} \quad (2\text{-}5)$$

$W_q$对于一个给定的查询来说是固定的，因此其对文档排序的影响可以忽略。这组公式的变体可以是不用对数表示，或者是在计算$w_{q,t}$的时候将 N 替换为$\max_t\{f_t\}$，或者是在查询词比较长的时候用$1 + \ln f_{q,t}$去乘以$w_{q,t}$，或者是用不同的组合方式去组合$w_{q,t}$和$w_{d,t}$，或者干脆将$w_d$定义成为文档的长度或字数，等等。

通常，这些变体都用$w_{q,t}$衡量短语的倒排文档频率（IDF），把$w_{d,t}$作为衡量短语频率（TF），相似度通常都是 TF×IDF 的公式。还有其他很多的变体公式基于统计学原理，而且在 TREC 中被证明是有效的，例如最著名的 Okapi 计算。

$$w_{q,t} = \ln\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \cdot \frac{(k_3+1) \cdot f_{q,t}}{k_3 + f_{q,t}} \quad (2\text{-}6)$$

$$w_{d,t} = \frac{(k_1+1) f_{d,t}}{K_d + f_{d,t}} \quad (2\text{-}7)$$

$$K_d = k_1\left((1-b) + b\frac{W_d}{W_A}\right) \quad (2\text{-}8)$$

$$S_{q,d} = \sum_{t \in q} w_{q,t} \cdot w_{d,t} \quad (2\text{-}9)$$

其中，$k_1$和$b$是参数，分别设为 1.2 和 0.75。$k_3$也是一个参数，一般设为∞，所以表达式$\frac{(k_3+1) \cdot f_{q,t}}{k_3 + f_{q,t}}$假定为等于$f_{q,t}$；$W_d$和$W_A$是文档长度和平均文档长度。当文档集规模不是非常大时，表 2-3 所示的算法可以很好的完成查询检索过程。

表 2-3　全文检索算法

| 对于查询 q，按照相关度高低顺序返回前 $r$ 个和 q 相关的文档 |
|---|
| 1) Calculate $w_{q,t}$ for each query term t in q. |
| 2) For each document d in the collection, |
| a) Set $S_d \leftarrow 0$. |
| b) For each query term t, |
| Calculate or read $w_{d,t}$, and |
| Set $S_d \leftarrow S_d + w_{q,t} \times w_{d,t}$. |
| c) Calculate or read $W_d$. |
| d) Set $S_d \leftarrow S_d / W_d$. |
| 3) Identify the $r$ greatest $S_d$ values and return the corresponding documents. |

## 2.3　XML 数据的管理

针对 XML 数据，目前存在以下几种类型：

- XML 数据库

其特点是以自然的方式处理 XML，基本的逻辑存储单位就是 XML 文档，存储和查询 XML 数据都较为方便，是专门针对 XML 数据设计的数据库。

- 原有数据库基础上对 XML 实行扩展

这种方法是通过对 XML 文件和现有数据库存储的形式二者之间建立映射关系，将映射之后的数据存储到现有数据库当中，利用现有的数据库出色的管理能力来管理 XML 数据。这种方法的优点是无需改变现有的数据库管理系统，缺点是可能会丢失原 XML 数据的信息，失去数据的原貌。

- 混合 XML 数据库

混合数据库是结合了上述二者的特点，对现有的数据库管理系统进行扩展，使其支持 XML 数据，同时保持对传统方式的支持。

如何存储 XML 是一个重要的问题，因为底层的存储对上层查询等操作有着重大的性能影响[21]。对于 XML 的存储，已有的存储策略主要有以下几种：基于文件的存储，利用 RDBMS(Relational DataBase Management System，关系型数据库管理系统)，利用面向对象的数据库 ODBMS(Object-Oriented Database Management System, 面向对象的数据库管理系统)，XML 数据库管理系统。

## 2.4 XQuery

XQuery 是一种由 World Wide Web Consortium(W3C)为了满足对特定的 XML 数据查询的需求而设计的语言。与关系型数据不同的是，XML 数据是高度灵活的，不像关系型数据有特定的结构，XML 数据通常是难以预测的，松散的，自描述的。

正因为 XML 数据的结构是难以预测的，所以对 XML 数据的检索过程通常和典型的关系型查询不同。XQuery 语言提供了这种需求的灵活性，比如，可以构建如下的 XML 查询：

- 在未知深度和结构的 XML 数据中查询数据。
- 改变数据的结构。
- 返回混合类型的数据。

在 XQuery 中，表达式是 Query 的主体部分。表达式可以被嵌套来构成 Query。每个 Query 都有一个 prolog，prolog 的作用是包含一系列的声明来定义 Query 执行的环境。Query 的主体包含一个表达式，来定义 Query 的结果。表达式可以由多个 XQuery 表达式通过运算符和关键词组合而成。

XQuery 的 FLWOR 表达式指的是 XQuery 的一种查询表达式构建方法。XQuery 查询支持 XPath 表达式，但是可以用描述能力更强的 FLWOR 表达式来构建查询。下文用一个 XQuery 实例说明 XQuery 的特点。

*for $v in $doc//video return $v*

表达式的含义是返回所有 doc 中的 video 元素。

可以在此基础上增加 where 子句来精确查询，如：

*for $v in $doc//video*

*where $v/year = 1999*

*return $v/title*

这便能够返回所有 1999 年的 video 的 title。这条等价于关系数据库 SQL 语句：

*SELECT v.title FROM video v WHERE v.year = 1999*

或者写成 XPath 表达式：

*$doc//video[year=1999]/title*

这两条查询是完全等价的。其实 XQuery 中的 FLWOR 语句和 XPath 表达式是完全等价的，只是在描述形式上有所区别，熟悉 XPath 的使用者可以继续在 XQuery 中使用 XPath 表达式，但是 XQuery 的 FLWOR 语句更接近于 SQL

语言，如果把 XML 数据想象成表的话，那么 XQuery 的新特性更符合这种思考习惯。

FLWOR 语句的名称来源于五个关键词的首字母。F 代表 *for*，表示对元素的迭代。L 表示 *let*，W 表示 *where*。每一个 FLWOR 语句至少要出现一个 *where* 或者 *let*。*let* 语句可以定义一个变量，给他赋予一个值。*where* 语句和 SQL 中的 *where* 很像，它规定一个条件，来过滤满足条件的元素。O 表示 *order by*，用于对结果排序。R 表示 *return*，用于构造返回的结果。

FLWOR 表达式是 XQuery 语言的核心部分，就像路径表达式是 XPath 的核心部分一样。XQuery 的另一项重要功能是可以输出一个构造好的 XML 文档，然后可以继续用作 XQuery 语句的输入。这极大的增强了 XQuery 的功能和灵活性。XQuery 全文规范集成了基于 XQuery 文本搜索功能的关键词检索，使得用户可以显式的在文档内容中使用关键词搜索。例如 XQuery 的全文检索语句‘*doc//footnote ftcontains "xquery"*’可以让用户实在脚注里检索关键词。此外，XQuery 的全文搜索结果不仅包含文件的标识符，文字的位置等经典检索提供的内容，而且还包括结点标识符，使得文档结构进一步从文档结点信息去寻找文档的结构的 XQuery 变得容易。这使得关键词检索，结果过滤和转换可以用一种语言实现。

## 2.5 JSON 技术

JSON(JavaScrip Object Notation)是一种轻量级的数据交换格式，易于人的编写阅读，同时也易于机器的解析合成。JSON 是基于 JavaScript Programming Language 的一个子集，其格式完全独立于编程语言，但是类似于 C 语言的风格。这些特性使得 JSON 很适合作为本系统的理想数据交换语言。

JSON 可以构建两种结构：

- “名称/值”对的集合。
- 有序的列表。

JSON 具有如图 2-5 的形式。

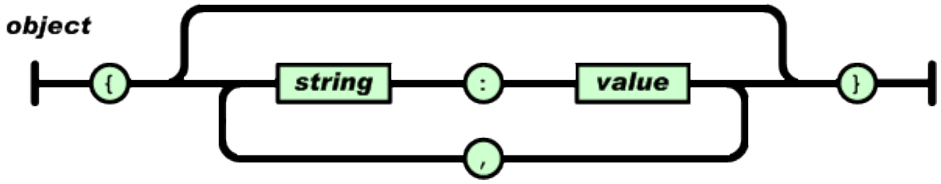



图 2-5 JSON 格式

对象是一个无序的“名称/值”的对的集合。一个对象以“{”开始，以“}”结束。

开始后首先是“名称”，“名称”结束后用“：”分割，之后是“值”。可以有多个“名称/值”对，之间用“，”分隔。

## 2.6 Apache MINA

MINA 是一个网络应用框架，能帮助开发者便捷的开发具有高可用性和高可扩展性的网络应用。MINA 通过 Java NIO，在多种传输协议如 TCP/IP 和 UDP/IP 之上提供了一种抽象的事件驱动的异步 API。它主要提供以下的功能：

- 针对多种协议的统一接口：
  - 通过 Java NIO 的 TCP/IP 和 UDP/IP 传输。
  - 通过 RXTX 的穿行通讯。
  - 实现自己的传输。
- Filter 接口，类似于 servlet 的 filters
- 提供底层和高层的 API：
  - 底层的使用 ByteBuffers。
  - 高层的使用用户定义的消息对象和编码。
- 通过 StreamIoHandler 支持基于流的 IO。

## 2.7 Drools 规则引擎

Drools 是用 Java 实现的基于 Chrales Forgy 的 RETE 算法实现的规则引擎。该实现称为 RETE OO，意为具有 OO 接口的 RETE。规则引擎由推理引擎发展而来，是一种嵌入在应用程序中的组件，实现了将业务决策从应用程序代码中分离出来，并使用预定义的语义模块编写业务决策。接受数据输入，解释业务规则，并根据业务规则做出业务决策。

规则引擎中的规则是知识的编码，图 2-6 所举得例子就是一条完整的规则。规则包含有 attributes，一个 Left hand side（LHS）和一个 Right hand side（RHS）。典型的规则是如图 2-6 描述的。

```
rule " < name > "
  < attribute >  < value >
  when
    < LHS >
  then
    < RHS >
end
```

图 2-6　Drools 规则示例

使用规则引擎的优点主要是实现了逻辑与数据的分离，数据保存在系统对象中，逻辑保存在规则中，可以使用更接近自然语言的方式来编写规则。并且规则相对于编码更容易对复杂问题进行描述。目前 Drools 使用的 Rete 算法已

经经受了大量的实际考验，能够提供对系统数据对象的非常有效率的匹配。

## 2.8 IKAnalyzer 分词器

IK Analyzer 是一个开源的，基于 Java 语言的轻量级中文分词工具包。最初，它是以 Luence 为应用主题的，结合词典分词和文法分析算法的中文分词组件。现在 IK Analyzer 已经发展成为面向 Java 的公共分词组件，独立于 Lucene 项目。IK Analyzer 的特性有[22]：

- 采用的特有的“正向迭代最细粒度的切分算法”，具有 60 万字/秒的高速处理能力。
- 采用了多子处理器的分析模式，支持英文字母，数字，常用中文数量词，罗马数字，科学统计法，中文词汇等的分词处理。
- 优化的词典存储，更小的内存占用。支持用户词典扩展定义。IK Analyzer 的结构设计如图 2-7。

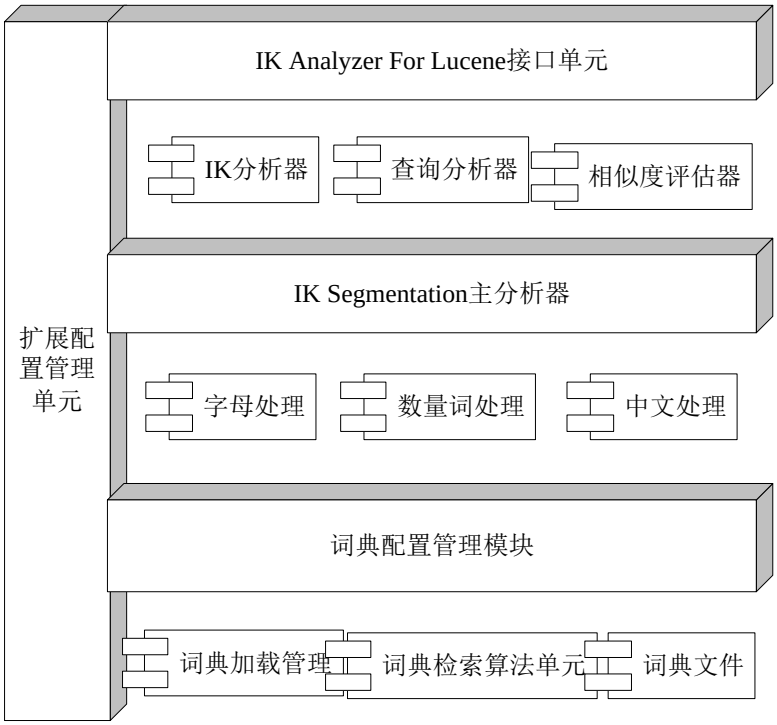



图 2-7　IK Analyzer 结构图

## 2.9 本章小结

本章首先简要介绍了课题实现所需的技术和算法。包括 NoSQL 数据库，全文检索的数据模型，相似度测量算法，XQuery 的数据模型和简要介绍，以及实现服务器架构所需的 JSON，Apache MINA 框架和 Drools 规则引擎的相关介绍。

# 第三章　可扩展系统框架

从需求来看，系统需要提供多种格式的文档的全文和元数据的存储检索查询服务。客户端可以接受用户的文档，对文档进行存储和索引，并且接受用户对已索引文档的查询。服务端响应用户的查询，调用底层的具体实现来处理用户的请求，并返回相应的结果。但是仅仅实现这些服务是不够的，一个好的应用应该有好的可扩展性。课题需要系统有高可扩展性，要满足如下要求：

- 服务端提供一个统一的接口，所有的底层服务均通过该接口调用。
- 服务的调用方式与底层具体实现无关，均通过统一的协议调用，参数传递和返回数据都采用统一数据格式。
- 能够方便修改和扩展底层服务而不影响系统其它部分。

综上所述，系统需首先开发一个组件作为底层服务和调用者之间的中间件，底层服务通过该中间件来提供服务。调用者通过协议规定的方式提交请求给中间件，中间件按照一定的规则选择具体的服务去调用，服务将返回结果递交给中间件，中间件再将数据通过统一的格式返回给调用者。这样就实现了调用逻辑和具体实现的分离，从而满足了课题的要求。当需要添加和修改底层服务时，只需在中间件中修改相关的规则即可，而不会影响系统其它部分，实现了各个部分之间的轻松耦合。

在本系统中，设计并开发的中间件是称为 MO 的组件。MO 介于客户端和底层服务中间。客户端接收用户的请求，然后按照规定的协议的方式去调用 MO 提供的接口。MO 解析客户端的请求，按照规定好的规则去选择底层的服务来响应请求。底层服务处理完毕之后将结果返回给 MO，MO 再返回给客户端。因此，除了设计 MO 以外，还需要设计调用 MO 的协议。在本系统中，需要规定的是如何去调用一项服务，如何给服务传递参数以及如何返回结果。具体来讲，就是要规定一种统一的数据交换格式，以及 MO 对上述三种情况的格式规定。同时，交换的数据应该使用合理的封装方式，封装方式要便于程序的生成和解析，而且还要轻量。

接受请求和返回数据可以看作是 MO 的 I/O 实现。除此之外，MO 还应该有内部的选择判断逻辑，即根据输入的内容判断具体调用的底层服务。因此，首先应该制定一种规则，使得 MO 可以依据规则来判断具体的操作。为了保证可扩展性，这种规则不应该是有程序来实现的，而是要满足调用逻辑和具体实现无关。这种设计不仅仅简化了编程逻辑的难度，而且在对现有服务进行修改或者是添加一项新的服务时，就无需对其程序代码修改，只需要在底层服务中添加上服务的具体实现，然后再修改相应的规则或者添加一条新的规则即可，

无需改变现有的其他服务的逻辑或者代码。

## 3.1 整体设计

整体设计如图 3-1 所示。

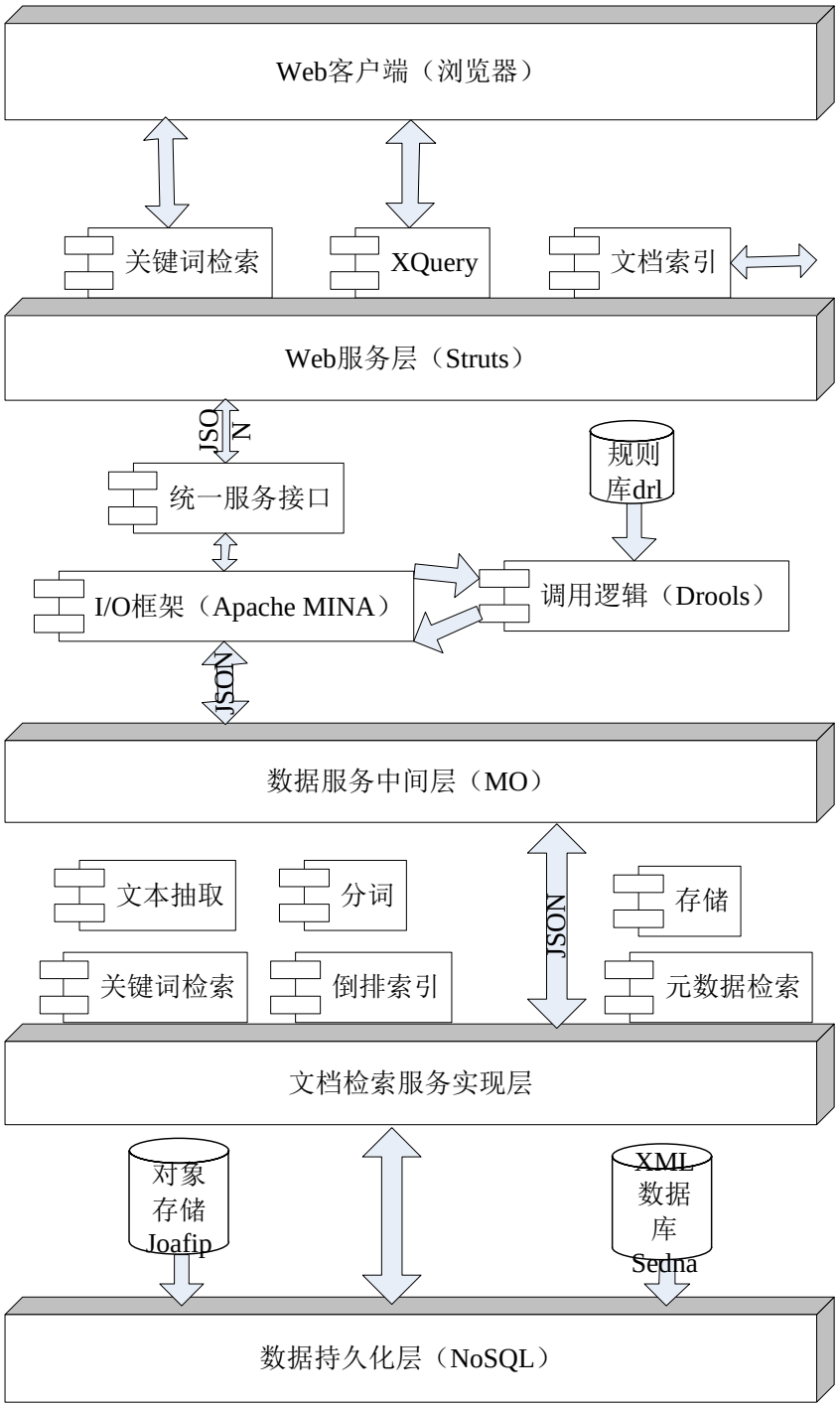



图 3-1 整体架构

系统大致可划分为上层应用，数据服务中间层，底层服务。上层应用包括客户端和 Web 服务层，接受用户请求，呈现结果，具体包括对给定的文档进行

处理，索引，存储，对已索引文档数据进行检索，呈现检索结果等等。数据服务中间层即 MO 作为中间件，负责为底层不同的服务提供统一的调用接口和数据交换服务。底层服务包括文档检索服务实现层和数据持久化层，实现了具体的业务逻辑，构成系统的基础。从本系统的设计目标来看，底层服务并非互相独立，而是要通过中间层 MO 结合起来。底层的服务包括文本抽取，分词，倒排索引的建立，数据存储，关键词检索，元数据检索。

## 3.2 数据交换格式方案

系统涉及不同的层面和不同的服务。层与层之间交换数据。例如上层应用需要向 MO 请求服务，按照本章引论中的分析，请求应该是满足协议规定格式的数据。而且对 MO 发出的请求既要包含希望调用的服务信息，还应该包含需要传递给该服务的参数。对于 MO 返回给上层的数据，应当包含由哪项底层服务返回的结果，服务执行成功与否，以及具体返回的结果数据。所以设计的交换格式既要包含所有信息，又要尽可能短，还要尽量减少数据交换的次数。这就需要选择合适的数据格式和便捷的数据封装方式。数据交换格式应该便于程序生成和解析，而且最好便于人的阅读和编写。2-4 节中介绍的 JSON 是一种理想的封装数据的方式。

JSON 的 key-value 对的存储方式很适合作为与 MO 的交互的数据封装格式，对于 MO 中服务的请求，可以规定如图 3-2 的数据格式。

```
"{'op':'search','input':'$keyword'}"
```

图 3-2 MO 传入参数格式

其中，第一个“名称/值”对规定需要调用的服务；第二个“名称/值”对说明需要传递给该服务的参数。

对于 MO 的返回值，可以规定如图 3-3 格式：

```
"{'retCode':'xxxx','result':'yyy'}"
```

图 3-3 MO 传出参数格式

第一个“名称/值”对说明返回码，用以标识返回数据的方法；第二个“名称/值”对说明了返回值。

可见，JSON 格式的数据具有足够的表达能力，而且十分的轻量。JSON 还支持以一个有序的列表的形式存储数据。JSON 数据的解析也十分的快捷方便，提供的多语言的编程接口，便于程序生成 JSON 格式的数据和对已有的数据进

行解析。而且 JSON 的可读性也非常好，便于人的阅读。所以本系统广泛采用 JSON 作为系统的数据交换方式，系统的数据流图如图 3-4。

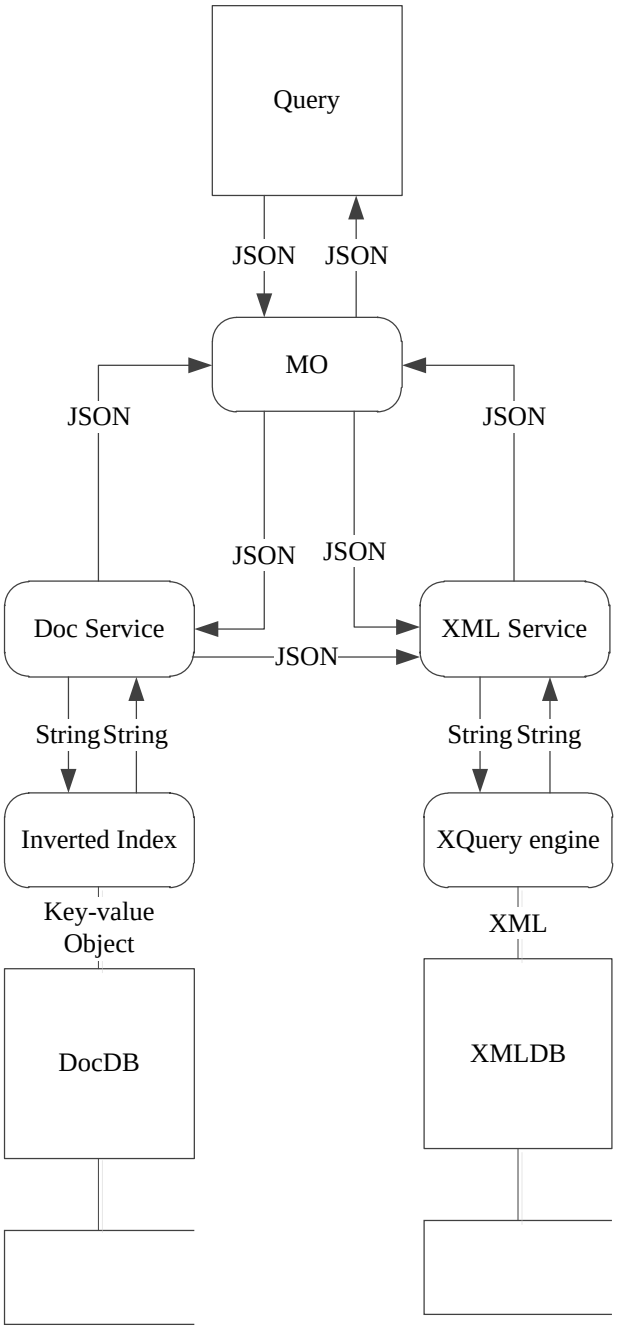



图 3-4　系统数据流图

MO 的输入输出均采用 JSON，JSON 的格式遵照系统协议规定。底层服务之间的数据交换依照具体的实现方案和使用的技术而定。这样便实现了对底层不同服务的提供统一的接口和数据交换格式的需求。

## 3.3 MO 的设计与实现

MO 需要响应上层用户请求，并且需要调用下层服务，接受下层服务返回的数据，然后返回给上层用户结果。因此实现 MO 需要首先建立一个合理的 I/O 框架。本系统的 MO 内部的数据流如图 3-5 所示。MO 提供一个对外的接口，负责接收请求和返回数据。当有外部请求时，MO 的请求监听组件将请求加入到内部维护的一个请求列表中。在列表中的请求会依次被送给请求处理部分处理。请求处理单元将请求交给事务处理逻辑单元处理。

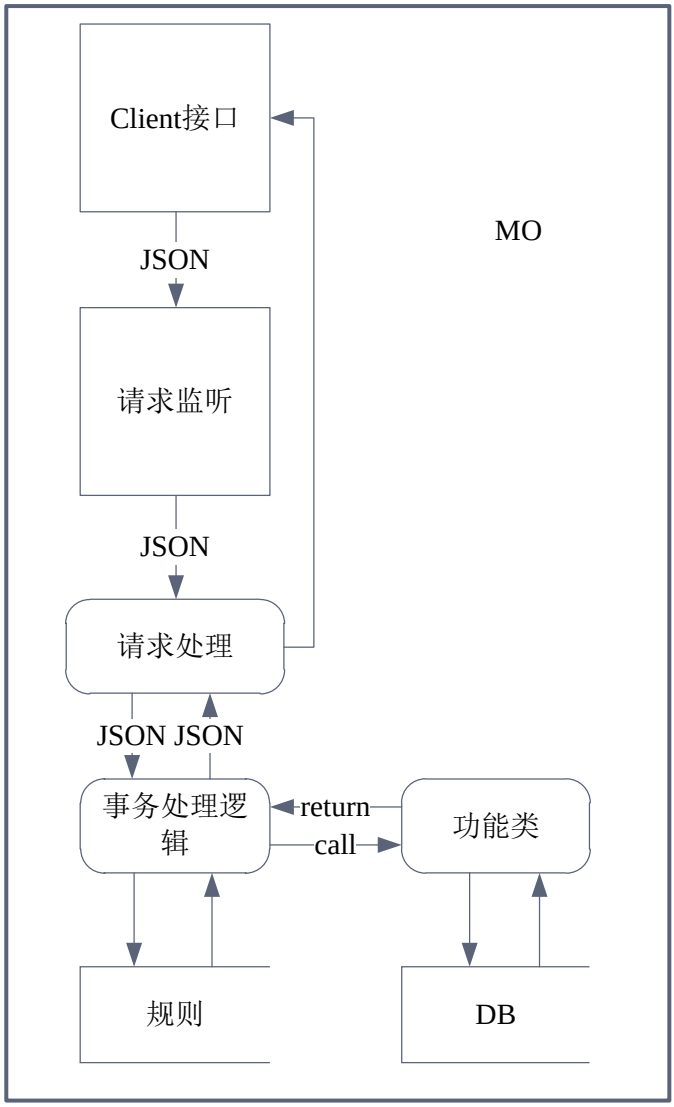



图 3-5 MO 内部数据流图

事务处理单元按照制定好的规则选择相应的功能类去响应用户请求，等待功能类返回结果，将结果返回给请求处理单元，请求处理单元再返回给用户。可见，MO 整个内部的数据传输应该是以异步的非阻塞的事件驱动的形式进行。

### 3.3.1 I/O 框架

为实现 MO 的 I/O 框架，论文选择了 Apache MINA 作为 MO 的 I/O 解决方案。MINA 的特性如论文 2.5 中所介绍。MINA 通过 Java NIO 提供了一种抽象的异步通讯 API，使用 MINA 能够便捷的进行非阻塞 I/O 而不用考虑过多的传输细节，并且是事件驱动的方式。基于 MINA 的 MO I/O 的具体实现如下：

首先需要实现系统的请求监听单元。TextLineServer 类完成了这一功能，其类图如图 3-4 所示。TextLineServer 可以实现图 3-6 中请求监听单元的功能，通过启动一个服务程序来等待接收用户的请求，然后将请求交给图 3-5 中的请求处理部分。

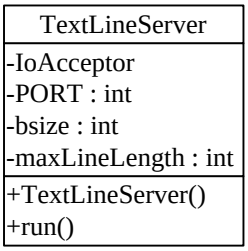



图 3-6 TextLineServer 类图

TextLineServer 内部维护了一个 org.apache.mina.core.service.IoAcceptor 类的对象。IoAcceptor 对象可以监听一个系统端口，维护一个消息队列，设定一个事件处理逻辑。当其监听的端口有输入的消息以后，IoAcceptor 便将其加入消息队列，并且不断地从消息队列中拿出消息交给以对象形式传递给它的请求处理单元处理。TextLineServer 的属性 PORT 定义了它监听的端口，bsize 规定了消息队列的最大值，maxLineLength 规定了最大的消息长度，用以对 IoAcceptor 初始化。对 IoAcceptor 的初始化在对象建立的时候进行，执行 TextLineServer 的 run()方法后，IoAcceptor 便开始监听 PORT 规定的端口的消息。

实现了请求监听单元后需要实现相应的请求处理单元。请求处理单元是由类 TextLineServerHandler 实现的。TextLineServerHandler 的类图如图 3-7 所示。

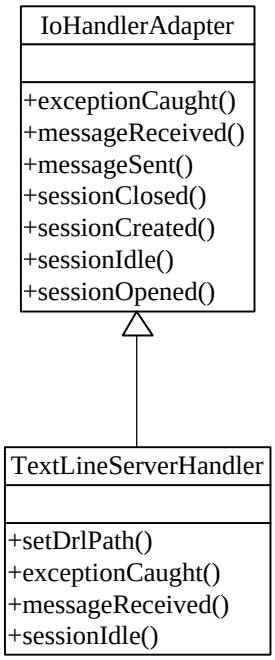



图 3-7 TextLineServerHandler 类图

TextLineServerHandler 继承自 org.apache.mina.core.service.IoHandlerAdapter，实现了 org.apache.mina.core.service.IoHandler 接口。TextLineServerHandler 类内部定义了 TextLineServer 各个状态下采取的行为。其实例对象被作为参数传递给 TextLineServer，TextLineServer 根据当前连接的状态选择执行不同的方法。例如在 TextLineServer 接收到输入的消息后，便执行 messageReceive()方法来响应。

### 3.3.2 事件处理逻辑单元

事件处理逻辑单元是 MO 的核心部分。这部分负责响应输入 MO 的消息。依照消息选择具体的功能类去处理数据，然后在将结果返回。事件处理逻辑单元应能具备以下功能：

- 可以接受消息，并对消息做出正确响应。
- 具有描述调用逻辑的规则库，且规则库可以从外部修改。
- 规则与程序语言无关，便于修改和阅读，便于机器解析。
- 规则描述能力强，且与具体实现分离。

基于上述的考虑，本系统选择了规则引擎 Drools 作为事件处理逻辑单元的解决方案。Drools 的基本特点如论文 2.6 中所述，使用规则引擎的优点主要是

实现了逻辑与数据的分离，数据保存在系统对象中，逻辑保存在规则中，可以使用更接近自然语言的方式来编写规则。并且规则相对于编码更容易对复杂问题进行描述。因此，使用规则引擎满足上文描述的事件处理逻辑的需求，可以用规则引擎来实现 MO 的核心部分。具体实现如图 3-8。

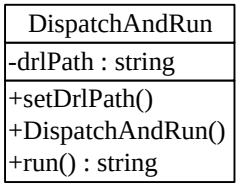



图 3-8 事务处理逻辑单元设计

规则引擎的规则是写在*.drl 文件中的。drlPath 变量用于保存 drl 文件的路径。setDrlPath()方法用来指定 drl 文件路径。run()方法接受一条消息，和规则库里的规则进行匹配，并执行相关的程序。论文 3.2 中介绍了如图 3-2 的 MO 接受的消息的格式。与之对应，MO 中规则引擎的规则文件使用如图 3-9 所示的格式。每条规则包含其判断条件，以及执行方法。例如规则 1，当 RulesInfo r 的 methodname 为*”search”*，那么便将 globaleInfo 中的 JsonOut 的”retCode”设为 0001，”result”设为执行将参数传递给 YydocStoreXML.search()后执行返回的结果。因此，规则引擎通过规则文件来判断调用逻辑。当底层服务有变化时，只需修改规则文件即可，而无需变动上层代码，这便实现了本章开篇对 MO 的需求。

```
#input format is "{'op':'search','input':'$keyword'}"
rule "0001 - full text search"
    when
        r:RulesInfo(methodName == "search")
    then
        globalInfo.setJsonOut("{" +
          "'retCode':'0001'," +
          "'result':'" +
          new YydocStoreXML().search(r.getJsonStr()) +
          "'}");
end

#input format is "{'op':'indexDoc','input':'$doc'}"
rule "0002 - index document to the database"
    when
        r:RulesInfo(methodName == "indexDoc")
    then
        globalInfo.setJsonOut("{" +
          "'retCode':'0002'," +
          "'result':'" +
          new YydocStoreXML().indexDoc(r.getJsonStr()) +
          "'}");
end

#input format is "{'op':'xquery','input':'$xquerystring'}"
rule "0003 - Return the xml list of articles"
    when
        r:RulesInfo(methodName == "xquery")
    then
        globalInfo.setJsonOut("{" +
          "'retCode':'0003'," +
          "'result':'" +
          new YydocStoreXML().xquery(r.getJsonStr()) +
          "'}");
end
```

图 3-9 规则文件实例

### 3.3.3 MO 运行流程

MO 执行的流程如图 3-10 所示。

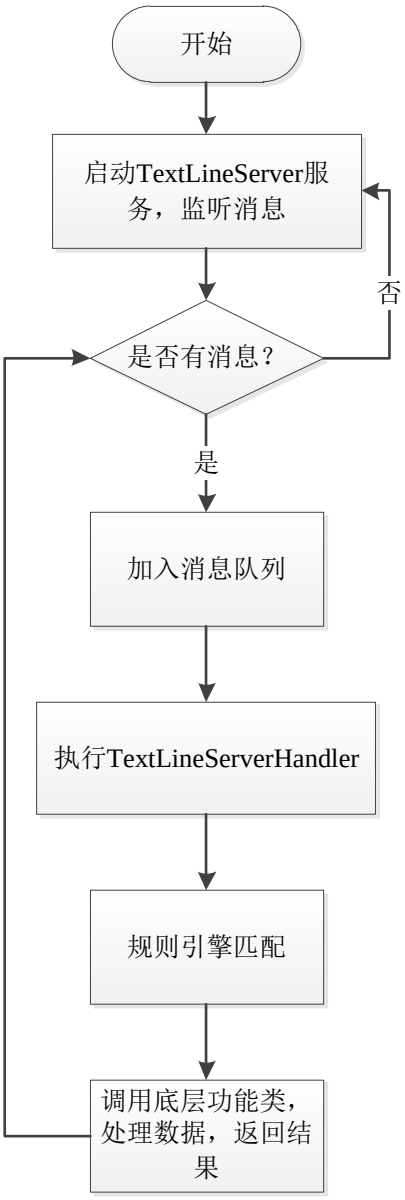



图 3-10 MO 运行时流程

## 3.4 本章小结

本章介绍了课题系统的可扩展框架的实现。设计并实现了 MO 这一中间件，从而使得底层的服务对上层应用提供统一的调用接口，调用逻辑和具体实现分离，实现了系统的松耦合和高度可扩展性，为系统的开发搭建起了框架，具体的服务实现后只需在 MO 的规则文件中添加规则即可方便的通过 MO 对上层应用提供服务而不影响现有的程序代码。

# 第四章　文档检索服务

文档检索服务是本系统的底层服务部分。即第三章中介绍的 MO 所调用的功能类。文档检索服务主要包括两部分数据存储检索服务：全文检索和元数据检索。全文检索部分具体来讲应实现以下功能：

- 文档内容的抽取和过滤；
- 文本内容的存储和查询；
- 原文档的存储和查询；
- 文本内容的处理，建立倒排索引；
- 对文本内容的基于关键词的检索，提供包含关键词的文档和文档中关键词的位置；

元数据检索部分具体应实现以下功能：

- 以 XML 格式存储文档的元数据信息；
- 支持对元数据的检索。

上述各项服务均可以作为 MO 的底层服务，通过 MO 的统一接口对上层应用提供服务。每项具体的服务均对应 MO 规则库中的一条规则，上层应用通过正确格式的请求便可以获得上述各项服务。

## 4.1　全文检索

全文检索是本系统的重要组成部分。全文检索包含两个部分：文档的抽取，索引和关键词检索。系统从用户提供的不同格式的文档中抽取出其中的文本内容，然后对文本内容建立索引，具体采用的索引方式是倒排索引。倒排索引的作用是提供一个关键词和包含该关键词的文档的列表，就像书的索引，每个关键词对应了其在书中出现的位置。建立了倒排索引以后，便可以提供关键词查询的功能，利用倒排索引去检索与关键词相关的文档。全文检索部分设计的数据流图如图 4-1 所示。

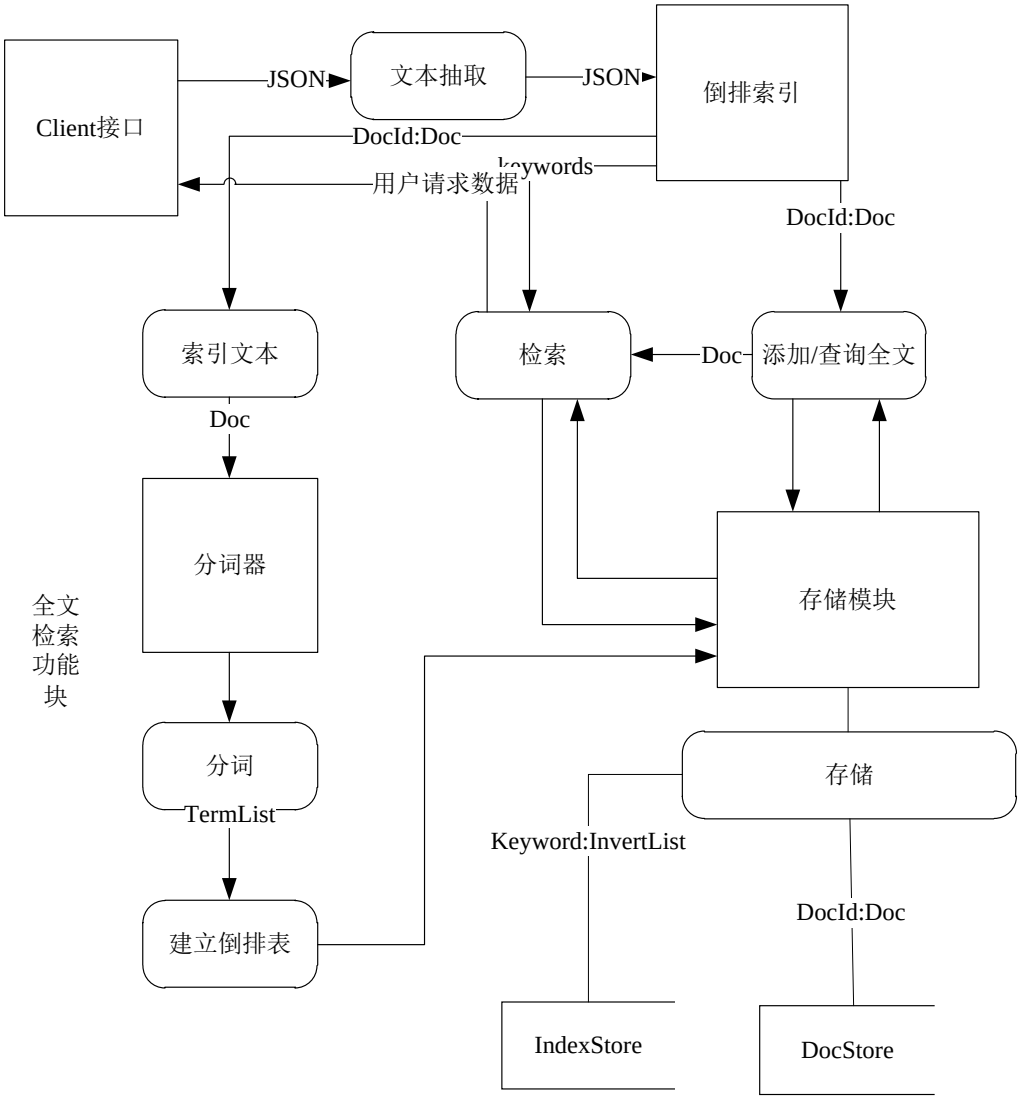



图 4-1 全文检索功能块数据流图

全文检索功能主要有五个部分：文本抽取，分词，倒排索引建立，存储，关键词检索。全文索引建立过程的流程如图 4-2。首先从其他格式的文档中抽取文本内容，然后对文本内容进行分析，分词。然后利用分词结果建立倒排索引，最后存储倒排索引。

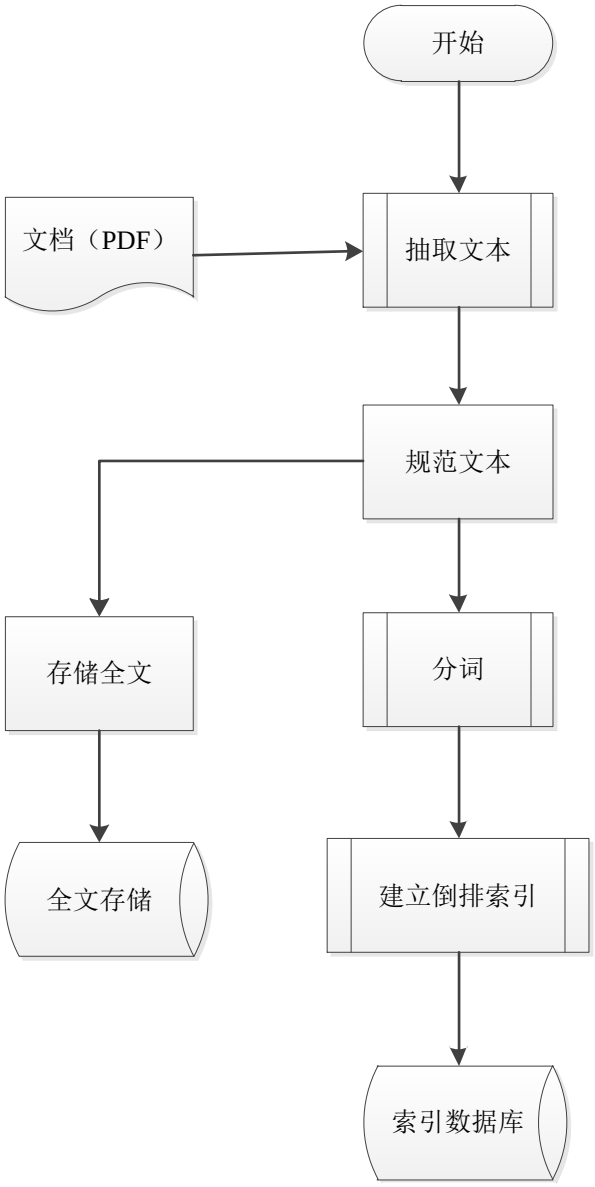



图 4-2　全文检索建立的流程图

### 4.1.1　文本抽取

系统在检索之前，首先要获得文本内容，这需要实现一个能够从不同格式的文档中抽取文本内容的抽取器。文本抽取器接受一个非文本格式的文档，然后输出正确的文本格式内容。本文内容应该与源文档保持一致，内容必须包括文档的所有有效部分而不是格式信息。

文本文档抽取器的主要作用是将其他格式文档中的文本内容抽取出来，并且规范化。例如，PDF 文本文档抽取器的类图如图 4-3。

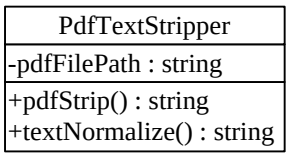



图 4-3　PDF 文本抽取器类图

该类首先获取 PDF 文件，利用 FileInputStream 读入，调用 pdfStrip()方法把 PDF 文档转化为文本内容，返回一个包含全部文本的字符串。

剥离文本内容具体使用 Apache PDFbox。PDFBox 是一个针对 PDF 文档提供各种工具的开源项目，其中一项功能便是从 PDF 文档中抽取文本信息。但是如果抽取的是英文的话，会出现问题，比如同一个英文单词由于排版的原因可能出现在两行中，在行末用连字符连接。所以，从 PDF 中抽取出来的文本还不能直接被使用，需要对文本进行规范化。textNormalize()方法就是去实现这个功能。该方法接受一个原始的文本数据，按照预先制定的规则来将文本规范化，最后返回规范化之后的文本数据。

文档抽取器是通过 MO 服务器来对应用提供服务的，所以方法的输入输出应该满足本系统规定的统一 JSON 字符串的格式。文本抽取器属于 MO 功能类的一部分，在 MO 的规则库中对应的调用规则如图 4-4 所示。

```
#input format is "{'op':'pdfStripper','input':'$pdfdata'}"
rule "0004 - return the text content of a pdf file in byte[] format"
    when
        r:RulesInfo(methodName == "pdfStripper")
    then
        globalInfo.setJsonOut("{" +
          "'retCode':'0004'," +
          "'result':'" +
          new YydocStoreXML().pdfStripper(r.getJsonStr()) +
          "'}");
end
```

图 4-4　PDF 抽取器在 MO 规则库规则

### 4.1.2　分词

抽取出来的文本内容在建立索引之前首先要经过分词。分词指的是将一个汉字序列切分成一个一个单独的词。分词是文本挖掘的基础，对于输入的一段文字，成功的进行分词，可以达到电脑自动识别语句含义的效果。分词需要专门的分词器，分词器的一般工作流程如图 4-5 所示。

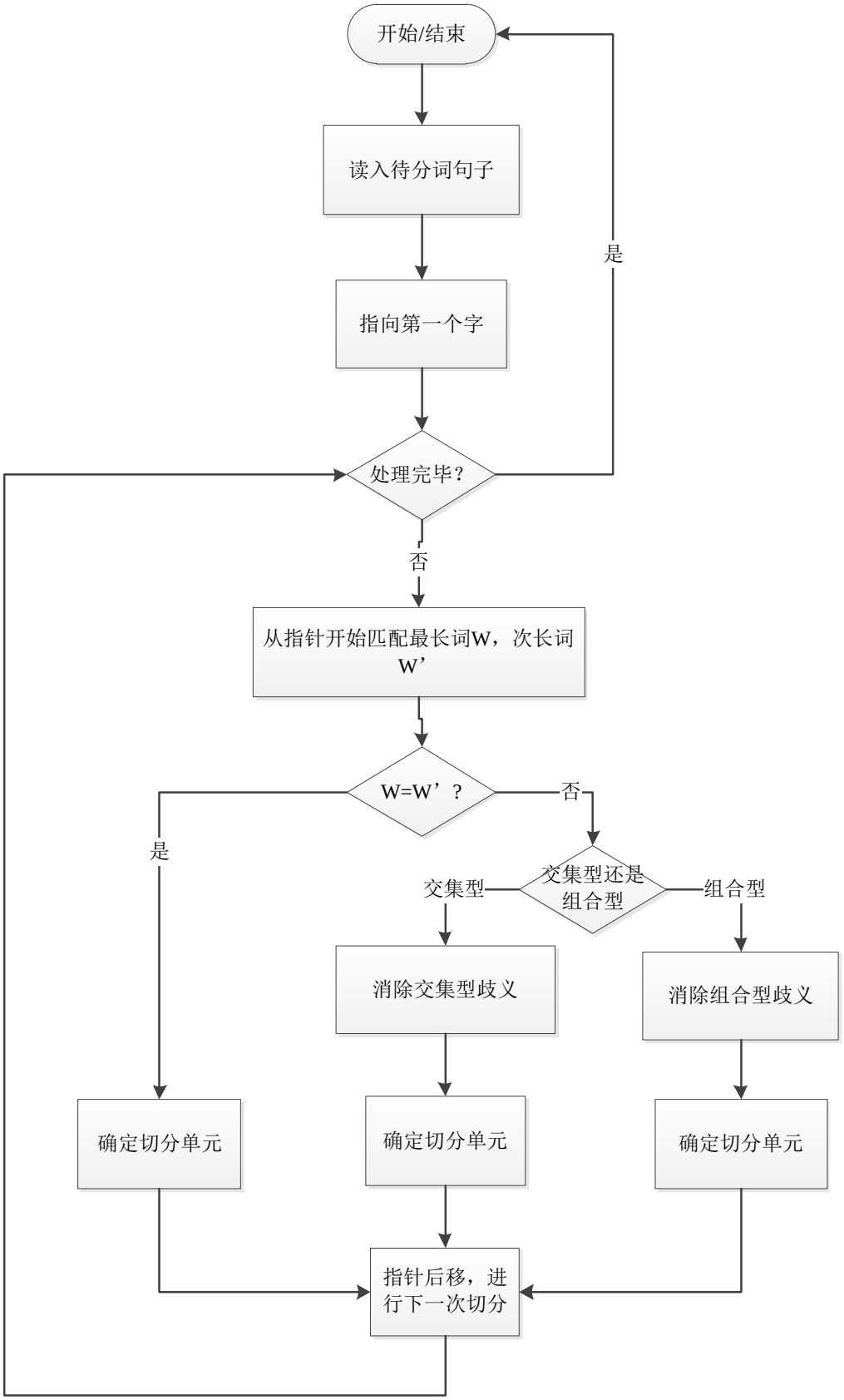

开始/结束
读入待分词句子
指向第一个字
处理完毕？
是
否
从指针开始匹配最长词W，次长词W’
W=W’？
是
否
交集型还是组合型
交集型
组合型
消除交集型歧义
消除组合型歧义
确定切分单元
确定切分单元
确定切分单元
指针后移，进行下一次切分

图 4-5　分词器工作流程

分词的过程中，不仅要记录划分的词，还应同时记录词在文中的位置，这样便于下一步的索引建立的过程。分词结束后，分词器返回的应该是一个词汇的列表，列表包括关键词，词在文中的位置信息。下一步利用这些信息建立倒排索引。

Segment 类如图 4-6 主要完成文档的分词工作。只有一个静态方法 run()，接受文档的全文，返回一个包含分词信息的 JSON 格式的字符串。具体的格式如图 4-7。

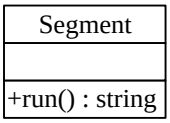



图 4-6　Segment 类图

```
{key:{no:"xx", pos:"x y z"}}
```

图 4-7 分词返回数据格式

key 表示一个词，key 对应的 value 也是一个 JSON 格式的 key-value 对，no 表示该词出现的次数，pos 表示该词出现的位置。返回的 JSON 字符串由所有的词汇的 key-value 对构成。Segment 类在内部使用了 2.7 中介绍的 IK Analyzer 分词器。

### 4.1.3　倒排索引

倒排索引包含应至少包含两个部分：

- 所有关键词的表。
- 相应关键词对应的倒排表，倒排表中是包含该关键词的文档编号和该词在文档中出现的次数和相应的位置。

倒排表的格式应满足图 4-8 形式。


图 4-8　倒排索引结构

其中 keyword 指关键词表中的每个关键词，DocId 指包含有该关键词的文档编号，No 指的是在该文档中关键词出现的次数，POS 是指文档中关键词出现的位置，位置可能有多个。DocId 和 TF.，POS 构成一个元组，每个关键词可

能包含多个元组。

一般来说，倒排索引的大小能够占到原数据的 40%。在倒排索引数据库中，除了保存索引信息之外，还要保存文档的文本内容或原始的文档，如 PDF 文件。因为通过倒排索引是无法还原全文信息的，在实际应用当中用户需要在需要与关键词相关的原文信息，所以系统应该用统一的文档编号来管理全文和索引，以便在查询时方便的返回用户请求的数据。所以，系统需要存储两种数据：索引，全文。两种数据都可以采用 Key-value 对的形式存储。索引的 key 是关键词，value 是倒排表；全文的 key 是文档编号，value 是全文内容。

分词结束后，建立倒排索引所需要的关键词表可以用分词表来构造。索引所需的位置信息也可以直接用分词返回的位置信息构造。索引所需的关键词在文档中出现的频率(TF)需要索引器进行统计来得出结果。建立倒排索引的流程如图 4-9。

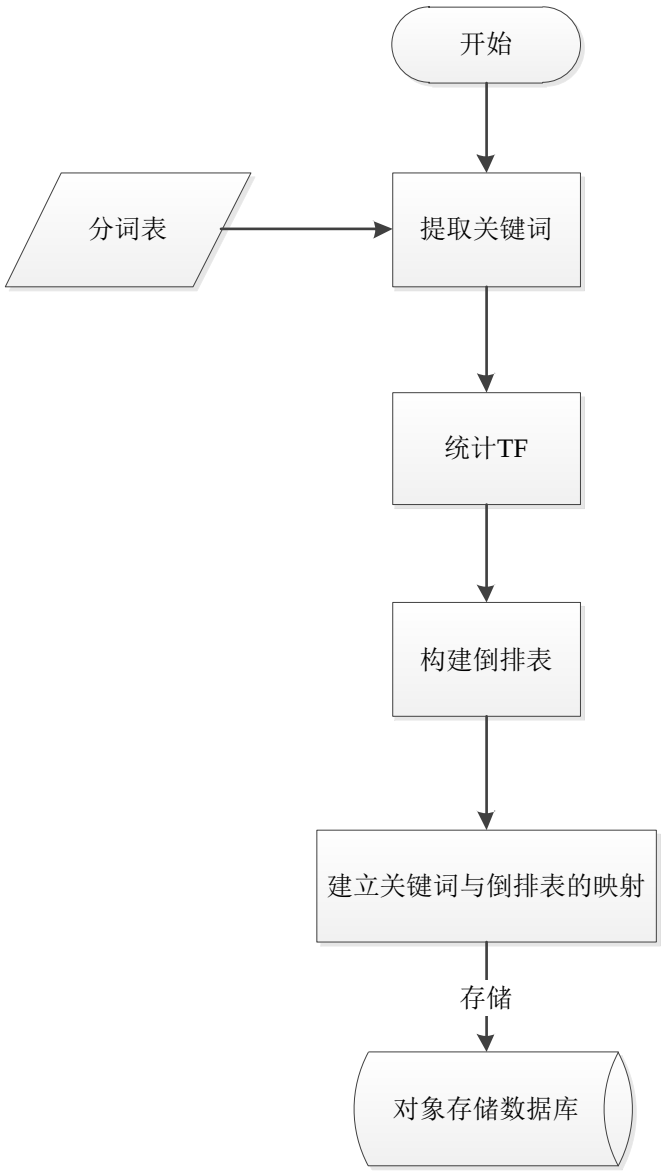



图 4-9　倒排索引建立流程

倒排索引的建立具体是由 InvertIndex 类实现的，如图 4-10 所示。


图 4-10　倒排索引类图

InvertIndex 类内部维护了三个 SimpleFileStore 对象，提供了八个方法。其功能和接受返回的数据格式表 4-1。

表 4-1　InvertIndex API

| 方法名称 | 方法功能 |
|---|---|
| IndexStore | 存储倒排索引表，采用 key-value 的形式 |
| docStore | 存储文档内容，采用 key-value 的形式 |
| metaStore | 存储文档的 MetaData，采用 key-value 的形式 |
| boolean createdb (String dbDir) | 在路径为 dbDir 的空目录下实例化三个 SimpleFileStore。返回 true 或者 false |
| boolean loadDB (String dbDir) | 载入在路径为 dbDir 的已经存在的数据库来实例化 SimpleFileStore。返回 true 或者 false。 |
| boolean index(String docId, String doc) | 参数 docId 表示要索引的文档编号，doc 表示文档的内容。将 doc 添加至倒排索引。返回 true 或者 false。 |
| boolean addDoc(String docId, String doc) | 参数表示同上，将 doc 内容添加至文本数据库中。返回 true 或者 false。 |

表 4-1 续

| | |
|---|---|
| boolean addMetadata (String docId, String metadata) | 参数 docId 表示文档编号，metadata 表示文档的 MetaData。将编号为 docId 的文档的 MetaData 添加至数据库。返回 true 或者 false。 |
| Object getDoc (String docId) | 参数 docId 表示文档编号，返回存储的文档对象。 |
| Object getMetadata (String docId) | 参数 docId 表示文档编号，返回存储的编号为 docId 的 MetaData。 |
| String search (String keywords) | 参数 keywords 表示关键词列表，词与词之间用空格分隔。返回包含关键词的文档 id，格式为{docId1:, docId2:,…} |

### 4.1.4 存储

系统实现了 SimpleFileStore 类来对倒排索引以及文档全文提供存储功能。该类在内部使用了开源工具 Joafip。Joafip 是一种基于 NoSQL 思想的对象存储工具。能够将 Java 数据对象持久化到文件系统而不使用数据库。Joafip 的优点如论文 2.1.3 中介绍。

Joafip 的以简单的 key-object 形式存储对象，在获取对象时同样通过 key 直接查找，非常便捷，适合作为本系统的全文索引部分的存储系统，所以，SimpleFileStore 利用 Joafip 为下文要介绍的倒排索引类实现了可以存储关键词的到排表，文档的全文以及文档的简单信息的容器，组成了全文索引的 DB。

SimpleFileStore 的实现如图 4-11 所示提供了六个方法。setStorage()方法用于配置数据在文件系统实际存放的位置；open(), close()方法分别用来打开和关闭当前的存储；writeObject(), removeObject()和 readObject()分别实现了对 SimpleFileStore 的写入，删除和查找对象的功能。

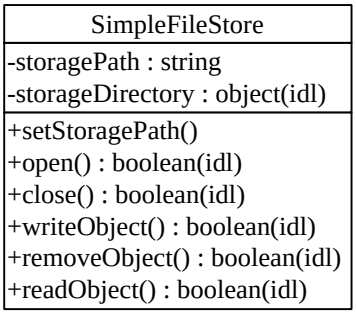



图 4-11　SimpleFileStore 类图

### 4.1.5　关键词检索

建立好倒排索引之后，需要一个检索的算法，来利用倒排索引返回用户需要的文档列表，该算法的描述如下：

表 4-2　文本检索算法

| 使用倒排索引检索返回满足查询 q 的前 r 篇文档 |
|---|
| 1) 对于每一篇文章，分配一个累加器$A_d$，并且初始化为 0。。 |
| 2) 对每一个查询 t， |
| a) 计算 t 中词 w 的权重$w_{q,t}$，取回 t 对应的倒排表。 |
| b) 对于每一个文档 d 和词在文档中的频率$f_{d,t}$，在倒排表中计算$w_{d,t}$，同时令 $A_d \leftarrow A_d + w_{q,\mathrm{t}} \times w_{d,t}$。 |
| 3) 读取$w_d$的值 |
| 4) 对每个$A_d > 0$，令$S_d \leftarrow A_d / W_d$。 |
| 5) 返回前 r 个最大的$S_d$对应的文档。 |

## 4.2　文档元数据检索

元数据在本系统中是以 XML 文件的形式存在，所以文档的元数据检索系统应该是一个可以处理 XML 数据的系统。元数据检索服务的数据流图设计如图 4-12。

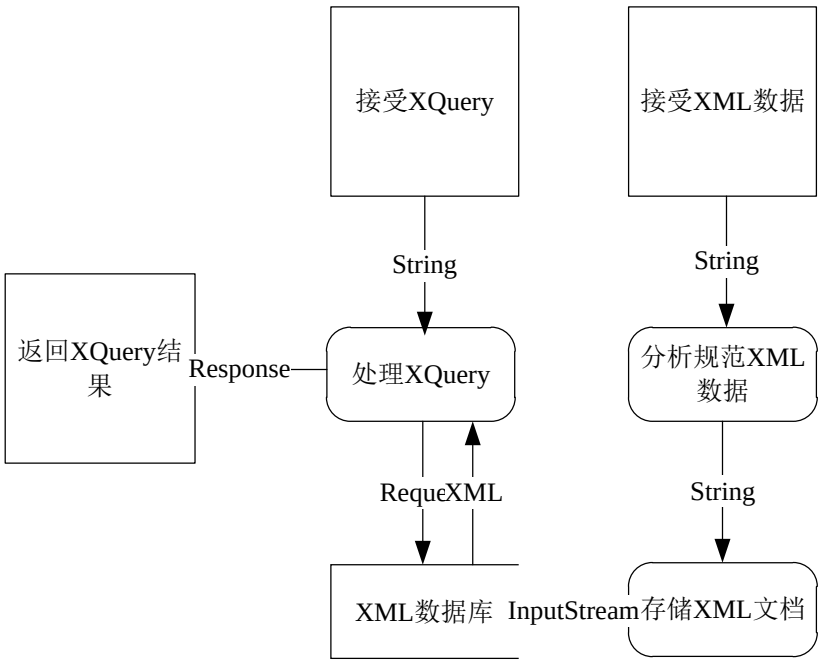



图 4-12 原数据服务数据流图

XML 数据处理功能块要求能够存储 XML 数据和对 XML 数据做 XQuery 查询。本系统是通过 NoSQL 数据库 Sedna 来实现这一功能。Sedna 的特点如论文 2.1.3 中所述。所以，本系统利用 Sedna 来存储和处理文档 Metadata 部分的数据。

XML 数据处理主要由一个功能类 XMLServer 来完成，功能类的类如图 4-13。

| **XMLServer** |
| --- |
| -url : string(idl)<br>-dbname : string(idl)<br>-user : string<br>-password : string<br>-sednaConnection : object(idl) |
| +addXML() : boolean(idl)<br>+xquery() : string<br>+connnetToDB() : boolean(idl) |

图 4-13 XMLServer 类图

其中 url 指的是 Sedna 数据库的服务地址，dbname 指的是所用的数据库名称。User，password 为数据库的用户名密码。sednaConnection 是 XMLServer 类内部维护的 Sedna 连接。可以通过 sednaConnection 对 Sedna 数据库进行添加，删除 XML 文档，对数据库执行 XQuery 查询。XMLServer 类的工作原理如图

4-14。

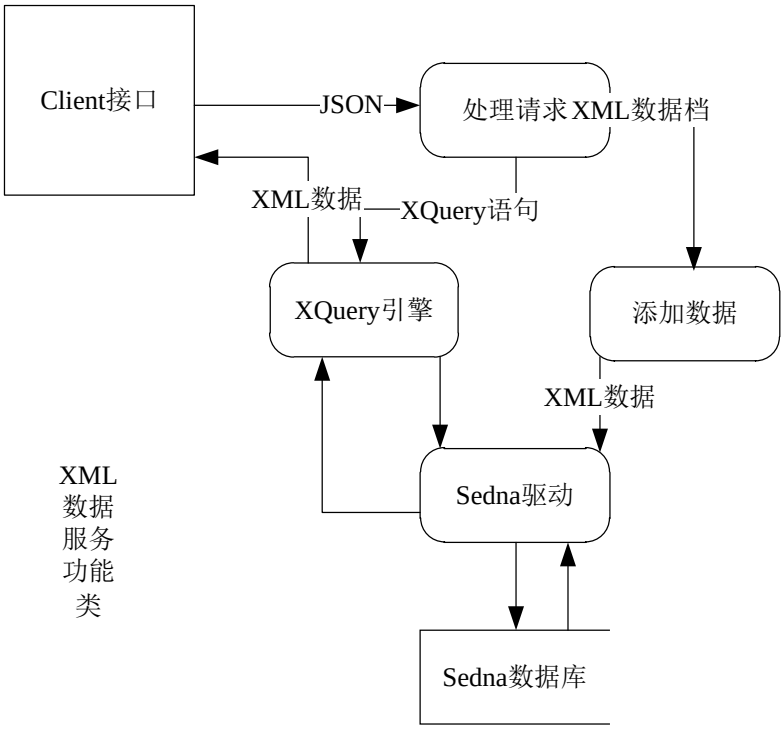



图 4-14　XML 数据服务功能类

## 4.3　本章小结

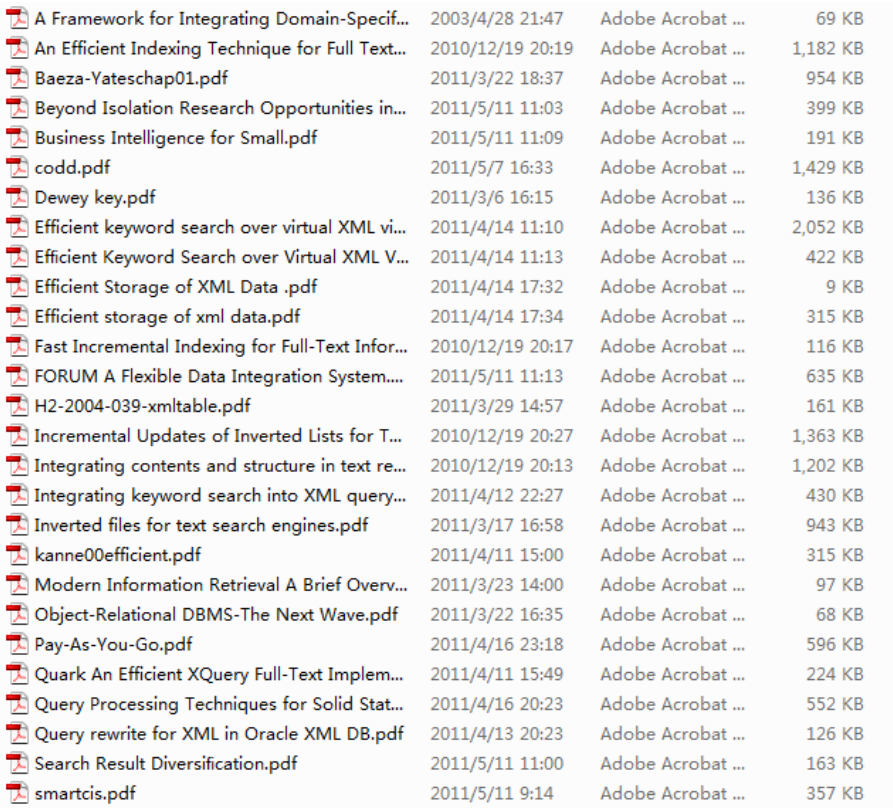
本节主要介绍了系统的主要底层应用的设计与实现。主要包括两个部分，一是全文检索部分，二是元数据检索部分。其中全文检索部分又包含文本抽取，分词，倒排索引建立，内容的存储和检索等子模块。这些模块都通过第三章介绍的 MO 来对上层应用提供服务。每项服务均对应 MO 规则中的一条规则说明，上层应用只需按照 MO 的规则要求提交正确格式的请求就可以获得底层应用提供的服务。

# 第五章　实验结果及分析

对于本系统主要测试其两方面的性能：一是其索引性能，二是其查询性能。实验环境为 PC，Intel Core i5 CPU，2.67GHz 主频双核，4G 内存。操作系统 Windows 7 64 位，Java SE Runtime Environment (Build 1.6.0_23-b05)，Tomcat 5.5。

## 5.1　文档索引

实验选取的数据源为 DBLP 中 2010 年的学术论文原文的 PDF 文件。论文均为长文，平均约 10000 英文单词/篇。本实验通过给系统提交 PDF 文件，测试其平均索引时间来检验系统的性能。测试输入文件的实例如图 5-1。



| | | | |
|---|---|---|---|
| A Framework for Integrating Domain-Specif... | 2003/4/28 21:47 | Adobe Acrobat ... | 69 KB |
| An Efficient Indexing Technique for Full Text... | 2010/12/19 20:19 | Adobe Acrobat ... | 1,182 KB |
| Baeza-Yateschap01.pdf | 2011/3/22 18:37 | Adobe Acrobat ... | 954 KB |
| Beyond Isolation Research Opportunities in... | 2011/5/11 11:03 | Adobe Acrobat ... | 399 KB |
| Business Intelligence for Small.pdf | 2011/5/11 11:09 | Adobe Acrobat ... | 191 KB |
| codd.pdf | 2011/5/7 16:33 | Adobe Acrobat ... | 1,429 KB |
| Dewey key.pdf | 2011/3/6 16:15 | Adobe Acrobat ... | 136 KB |
| Efficient keyword search over virtual XML vi... | 2011/4/14 11:10 | Adobe Acrobat ... | 2,052 KB |
| Efficient Keyword Search over Virtual XML V... | 2011/4/14 11:13 | Adobe Acrobat ... | 422 KB |
| Efficient Storage of XML Data .pdf | 2011/4/14 17:32 | Adobe Acrobat ... | 9 KB |
| Efficient storage of xml data.pdf | 2011/4/14 17:34 | Adobe Acrobat ... | 315 KB |
| Fast Incremental Indexing for Full-Text Infor... | 2010/12/19 20:17 | Adobe Acrobat ... | 116 KB |
| FORUM A Flexible Data Integration System.... | 2011/5/11 11:13 | Adobe Acrobat ... | 635 KB |
| H2-2004-039-xmltable.pdf | 2011/3/29 14:57 | Adobe Acrobat ... | 161 KB |
| Incremental Updates of Inverted Lists for T... | 2010/12/19 20:27 | Adobe Acrobat ... | 1,363 KB |
| Integrating contents and structure in text re... | 2010/12/19 20:13 | Adobe Acrobat ... | 1,202 KB |
| Integrating keyword search into XML query... | 2011/4/12 22:27 | Adobe Acrobat ... | 430 KB |
| Inverted files for text search engines.pdf | 2011/3/17 16:58 | Adobe Acrobat ... | 943 KB |
| kanne00efficient.pdf | 2011/4/11 15:00 | Adobe Acrobat ... | 315 KB |
| Modern Information Retrieval A Brief Overv... | 2011/3/23 14:00 | Adobe Acrobat ... | 97 KB |
| Object-Relational DBMS-The Next Wave.pdf | 2011/3/22 16:35 | Adobe Acrobat ... | 68 KB |
| Pay-As-You-Go.pdf | 2011/4/16 23:18 | Adobe Acrobat ... | 596 KB |
| Quark An Efficient XQuery Full-Text Implem... | 2011/4/11 15:49 | Adobe Acrobat ... | 224 KB |
| Query Processing Techniques for Solid Stat... | 2011/4/16 20:23 | Adobe Acrobat ... | 552 KB |
| Query rewrite for XML in Oracle XML DB.pdf | 2011/4/13 20:23 | Adobe Acrobat ... | 126 KB |
| Search Result Diversification.pdf | 2011/5/11 11:00 | Adobe Acrobat ... | 163 KB |
| smartcis.pdf | 2011/5/11 9:14 | Adobe Acrobat ... | 357 KB |

图 5-1　部分检索文档示例

测试结果如表 5-1。

可见，系统的索引效率很高，且比较健壮，对索引中可能遇到的特殊字符或者其他异常有较好的容错性。

表 5-1　索引测试结果

| | |
|---|---|
| 索引文件数 | 62 |
| 总索引时间 | 76570ms |
| 平均文件索引时间 | 1235ms |
| 成功数 | 62 |
| 失败数 | 0 |
| 成功率 | 100% |

## 5.2　文档检索

### 5.2.1　关键词检索测试

本实验对系统提交关键词，测试其准确率和响应时间。主要测试系统的全文检索性能。选取的关键词及测试结果如表 5-2。

表 5-2　检索测试结果

| 关键词 | 返回文档数 | 所需时间 | 精确率 |
|---|---|---|---|
| *database* | 48 | 11ms | 100% |
| *xquery* | 14 | 29ms | 100% |
| *xml* | 24 | 8ms | 100% |
| *inverted list* | 32 | 36ms | 100% |
| *object-oriented DBMS* | 36 | 17ms | 100% |

可见，本系统的关键词检索具有很高的响应速度和高的精确率。对于关键词检索来说，还需了解检索数据集规模对检索时间的影响。下面的实验便针对不同规模的数据集，测试了本系统对相同关键词的响应时间。测试结果如下，时间单位 ms。

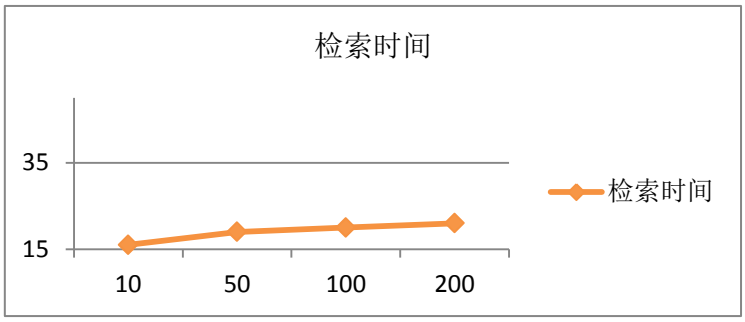



图 5-2 关键词检索与数据集规模

### 5.2.2 元数据检索测试

本系统除了支持传统的关键词检索以外，最重要的是其支持 XML 检索。首先对系统的 XQuery 性能做测试。对系统提交如下的 XQuery 语句。

```
for $x in doc('dblp')/dblp/article
where contains($x/journal, 'SIGMOD') and $x/year=2010
return $x
```

图 5-3 测试 XQuery 语句 1

```
for $x in doc('dblp')/dblp/article
where contains($x/title, 'Search result diversification.')
return $x
```

图 5-4 测试 XQuery 语句 2

系统返回所有满足要求的信息。执行测试语句 1，第一次检索由于需要做初始化，加载数据库等工作，所需时间较长，耗时约 7630ms，第二次执行相同的检索仅需 749ms。返回了所有系统中满足 *where* 语句的结果，以 XML 形式展现。执行测试语句 2，返回了标题为“*Search result diversification.*”的文章的所有信息，耗时 982ms。

## 5.3 结果分析

本系统查询性能较高，对于基于关键词的检索，实现的倒排索引结构可以提供很高的查询性能，且查询的性能受数据库规模影响较小。单纯的 XQuery 性能较关键词查询较差，且查询性能受数据规模影响，随着数据库规模增大，查询消耗时间也随之增长。全文检索和 XQuery 集成的检索方式下，系统响应时间和普通全文检索类似。这是因为全文检索的效率较高，且经过关键词已经过滤了大部分不相关信息，所以在已过滤的信息基础上再做 XQuery 查询，系

统效率就会很高。

系统的索引效率受磁盘 I/O 影响较大，所以本系统采用 cache 的方式，首先在内存中利用 Map 数据结构来维护倒排索引表，需要写入磁盘时才将内存中的数据持久化。在内存中插入，修改倒排表中的数据时间较短，因此能够避免许多不必要的磁盘 I/O，大大加快索引的速度。

## 5.4 本章小结

本章主要介绍了对文档的扩展全文检索系统的测试和结果分析。首先对系统的索引过程进行测试分析，然后在索引所得的数据之上做了全文检索及 XQuery 检索的测试和结果分析。

# 第六章　总结及展望

## 6.1　本文工作总结

首先，论文基于 NoSQL 方式的存储技术，设计并实现了针对如 PDF 等格式的文档的存储、索引和检索的系统。存储索引和检索的内容包括文档的全文和描述文档的元数据。对于全文采用建立倒排索引的方式支持全文检索，通过对象存储的方式存储索引和全文；对于文档的元数据，采用 XML 描述和存储，利用 XQuery 来对其查询。其次，论文基于 Apache MINA 和 Drools，设计并实现了一个轻量级中间组件 MO，实现了调用逻辑与具体实现分离，使得系统可以方便地对现有底层功能修改，扩展，集成不同的应用组件，而不对系统造成大的影响。上层应用通过统一的方式调用服务，从而是系统具有高的可扩展性和与低的耦合度。具体论文研究了如下问题：

1. 深入了解了 NoSQL 数据库的特点，以及其和传统关系型数据库相比优势和劣势所在。
2. 对 XML 技术和 XQuery 查询有了比较透彻的了解。
3. 对于全文检索技术以及其和 XQuery 查询相结合的应用做了研究。
4. 实现了一个全文检索的倒排索引的建立和查询引擎。
5. 实现了一个对 XML 文档执行 XQuery 查询的系统。
6. 将倒排索引和 XQuery 想结合，实现了全文检索加元数据查询系统。
7. 设计了合理的应用架构，实现了一个统一的数据交换的框架 MO，便于应用的开发和扩展。
8. 实现了大规模文档管理检索的系统，达到了课题的预期目标。

## 6.2　课题展望

本课题由于研究时间短，对于系统的实现也只是初步的，对于下一步的研究可以对现有的系统做如下的扩展：

1. 继续完善系统，继续优化，完善功能，增加实用性。
2. 对倒排索引的实现方法做优化，提高索引的效率，以应对更大规模的数据。
3. 将现有的思想和技术移植到关系数据库之上，结合 XML 索引技术，对现有存储在关系型数据库的大规模文档提供一种过渡性质的检索解决方案，使得关系型数据库中的文档也具有全文检索加 Metadata 检索的能力，而无需将大量的数据从关系型表中迁移到非关系型数据库中去。可以在现有的关系型数据库之上做扩展，提供一种 XML 视图，然后建

立一个 XML 索引，通过对 XML 索引做 XQuery 查询。在对关系型数据库中存储的文档数据建立倒排索引表来支持关键词查询。在倒排索引中加入标示其在内容在 XML 索引中结点的位置来将倒排索引和 XML 索引结合起来，为关系型数据库提供高效的非结构化文档的查询能力。

# 参考文献


[1] E. F. Codd. 1970. A relational model of data for large shared data banks. Commun[J]. ACM 13, 6 (June 1970), 377-387.

[2] R. A. Baeza-Yates, G. Navarro: Integrating Contents and Structure in Text Retrieval.[J] SIGMOD Record 25(1): 67-79 (1996).

[3] G. Salton and M. McGill. Introduction to Modern Information Retrieval.[M]McGraw-Hill, New York, 1983.

[4] ISO/IEC 13249-2:2000, Information technology— Database languages — SQL Multimedia and Application Packages —Part 2: Full-Text[S], International Organization For Standardization, 2000.

[5] I.O. for Standardization (ISO). Information Technology-Database Language SQL-Part 14: XML-Related Specifications (SQL/XML)[S]

[6] Chang, F., J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes and R.E. Gruber: Bigtable: A distributed storage system for structured data[R]. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI06), 2006.

[7] Hadoop[EB/OL]. http://hadoop.apache.org/

[8] Apache Cassandra[EB/OL]. http://cassandra.apache.org/

[9] JJ Furman, Jonas S Karlsson, Jean Michel Leon, Steve Newman, Alex Lloyd, and Philip Zeyliger. Megastore: A Scalable Data System for User Facing Applications[R]. Proceedings of the ACM SIGMOD international conference on Management of Data, 2008.

[10] DB2[EB/OL]. http://www.ibm.com/developerworks/data/library/techarticle/dm-0606seubert/

[11] SQLServer[EB/OL]. http://msdn.microsoft.com/en-us/library/bb522491.aspx

[12] Oracle[EB/OL]. http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10492/xdb09sea.htm

[13] D. Florescu, D. Kossmann, I. Manolescu: Integrating Keyword Search into XML Query Processing.[R] BDA 2000.

[14] XPath[S]. http://www.w3.org/TR/xpath-full-text-10/

[15] R. A. Baeza-Yates, G. Navarro: Integrating Contents and Structure in Text Retrieval.[J] SIGMOD Record 25(1): 67-79 (1996).

[16] SimpleDB[EB/OL]. http://aws.amazon.com/simpledb/

[17] MongoDB[EB/OL]. http://www.mongodb.org/

[18] Versant[EB/OL]. http://www.versant.com/index.aspx

[19] Sedna[EB/OL]. http://modis.ispras.ru/sedna/

[20] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel L ´eon, Yawei Li, Alexander Lloyd, Vadim Yushprakh Megastore: Providing Scalable, Highly Available Storage for Interactive Services[EB/OL]. http://telecomblogs.in/wp-content/uploads/2011/02/Google-Infra-Research-Paper.pdf

[21] Tian Feng，DeWittDJ，et a1．The Design and Performance Evaluation of Alternative XML Storage Strategies[J]．SIGMOD Record,March 2002，31(1)：68-79．

[22] IKAnalyzer[EB/OL]. http://code.google.com/p/ik-analyzer/

# 致谢

由衷的感谢我的导师张坤龙，毕业设计的研究工作是在他的悉心指导和耐心鼓励下得以完成的。从毕业设计选题到毕业论文完成的半年多时间里他给予了我很多教诲和帮助，在论文定稿阶段，他多次为本人审稿，倾注了大量的精力和心血。导师严谨的治学态度、诲人不倦的作风和一丝不苟的敬业精神给了我无穷的动力。这些无疑将深刻地影响我未来的一生。在论文完成之际，从心底里向我的导师致以诚挚的谢意。

感谢在毕业设计整个过程中提供工作环境并给予悉心指点的清华大学邢春晓、张勇老师以及整个清华大学信息技术研究院 WEB 中心的老师和同学。

感谢父母、兄弟。

感谢评阅本文的专家、教授以及四年来给予指导和帮助的天津大学计算机学院全体教师和同学。