Multi-Model 数据库 MarkLogic 调研报告

申恒恒　　唐浩　　宋斌鹏
余知栋　　曹召宾　　辛明佳

摘要

多模型数据库（Multi-Model Database）是新一代的数据库，与只支持单一数据模型的传统数据库有所不同，多模型数据库是一种在统一、综合的平台下同时支持多种不同的数据模型的数据库，这些数据模型可包括传统的关系模型和 NoSQL 数据模型，并且多模型数据库具有自己一种或多种的查询语言，并不仅仅依赖于传统的 SQL 查询语言，这使得数据组织、管理和操作变得十分的简单和便捷. 在企业中，拥有结构良好的数据和基于 NoSQL 技术的综合数据平台对用户是有益的 [4] ，这种方法显著地降低了集成，迁移，开发，维护和运营等问题. 因此，在本论文中，我们将介绍多数据模型的数据库代表-MarkLogic, 将会介绍它的底层架构，所支持的数据模型，以及在底层和顶层如何访问、管理和操作数据，最后设计 Benchmark 来对 MarkLogic 进行评测，并与文档数据库 MongoDB 进行比较.

关键字：Multi-Model Database; MarkLogic; MongoDB

1　MarkLogic 简介

NoSQL 是指 non-SQL 或者非关系型数据库，相比传统的关系型数据库，它提供了一种机制来访问和存储结构化和非结构化数据. 由于 NoSQL 的设计简单，易于扩展和高可用性等特点，NoSQL 数据库目前广泛应用于工业界和企业生产环境中.

下面是 NoSQL 按照数据模型进行划分，分类如下：

- 基于键值对存储
- 基于列式存储
- 面向图的存储
- 基于文档的存储
- 多数据模型

支持多数据模型的数据库的设计初衷就是支持多种数据模型而非仅仅支持单个数据模型的应用，而 MarkLogic 就是多数据模型数据库的典型代表.

1.1 什么是 MarkLogic？

MarkLogic 是唯一面向企业级的 NoSQL 数据库，它允许用户存储、管理和搜索数据，并对结构化数据和非结构化数据的操作以及存储进行优化，支持的数据格式有 JSON，XML，RDF，GeoSpatial，文本，和二进制数据. 通过 MarkLogic 管理平台，用户可以在任意架构下快速的开发应用并交付.

MarkLogic 也支持 ACID 的事务处理，高可用、故障恢复和安全等企业数据库必备的特性，MarkLogic 也可以无缝隙地链接到 hadoop 这种大数据系统上，可以更好的储存和分析数据. 它也可以很容易的部署在云平台上，比如 AWS，GoogleCloud 等云服务平台上. 其自身内置了应用服务器和文本搜索等功能，支持 REST 特性，这种特性在现在企业开发实践中是非常重要的.

另外，它支持多种编程语言和多种语言的应用编程接口，并且封装了大量的数据操作，使得应用开发变得简单.

1.2 工作原理

MarkLogic 使用 XML 文档作为他的数据模型，并且将文档存在事务存储库中. 对每个加载进来的文档建立单词索引，值索引以及短语索引，正因为它的通用索引 (Universal Index) 的特性，因此对其建立索引时不需要预先了解文档那个结构或者遵循一定的规则.Marklogic Server 使用无共享架构（share-nothing）组建集群，并通过支持大规模数据存储和卓越的性能在市场中脱颖而出.

除了支持 XML 文档存储之外，MarkLogic 还可以存储 JSON，文本，和二进制文档，JSON 文档为了索引的方便内部通常转化 XML 进行建立索引，文本文档也会被建立索引，每个文本文档都是没有父节点的 XML 文本节点一样. 二进制文档在默认情况下是不建立索引的，但也可以使用文件的元数据或者内容来进行建立索引，这是可选的.

2 背景介绍

MarkLogic 作为一个企业级的 NoSQL 数据库解决方案，其产品是一个集存储、管理、搜索 JSON 和 XML 文档和图形数据的 multi-model NoSQL 数据库. 可以将海量数据通过灵活的组织结构去构建庞大的 web 应用. 它拥有很多重要和优秀的特性，使得成为 NoSQL 数据库众多产品的独树一帜的一个产品，本节主要会介绍 MarkLogic 的历史背景、客户、架构以及杀手锏-技术创新点.

2.1 MarkLogic 的历史背景

2001 年，一群住在旧金山湾区附近富有文档搜索经验的工程师们成立了一家专注于处理海量 XML 文档的公司. 由于 XML 文档包含标记（markup），所以他们将公司命名为 MarkLogic.

MarkLogic 发现他们在美国联邦政府系统中的产品有一个需求，即 TB 级的智库信息以及大型出版物需要存储和搜索它们的 XML 文档. 自 2001 年以来，MarkLogic 已经发展成为成熟的、通用的、高度可扩展的文档存储并对 ACID 事务和细粒度的、基于角色的访问控制提供支持. 最初，MarkLogic 开发者使用的主要语言是 XQuery 与 REST 的组合，新版本支持 Java 和其他语言的编程接口.

MarkLogic 是一个商业产品，对于任何超过 40 GB 的数据集都需要软件许可.NoSQL 与商用产品和开源产品联系紧密，为业务问题提供了创新的解决方案[9].MarkLogic 是市场上九大领先的 NoSQL 数据库供应商之一[11]，2017 年，Gartner 将 MarkLogic 评为数据仓库市场中的" 有远见者"[10].

2.2 主要客户

- ABN AMRO（荷兰银行，荷兰银行是荷兰最大的跨国金融机构之一，在世界各地设有分支机构.）
- BBC（英国广播公司）
- Centers for Medicare & Medicaid Services (CMS) （美國醫療保險和醫療補助服務中心）
- NBC（全国广播公司，美国）
- O'Reilly Media(奥莱利媒体, 著名的书籍出版商)

2.3 架构

MarkLogic 定义了两种类型的集群节点，如表1所示：

节点	解释
查询节点	接收查询请求并协调与执行查询相关的所有活动
文档节点	包含 XML 文档，并负责在本地文件系统上执行查询

表 1: MarkLogic 的集群节点定义.

查询请求被发送到一个查询节点后，即被分发到每个包含 XML 文档索引的远端服务器. 所有符合要求的文档会被返回至查询节点. 当所有文档节点完成响应后，查询结果即被返回.

MarkLogic 的架构是将查询任务移动至文档而不是将文档移动至查询服务器，这样的架构对千兆字节的文档具有线性可扩展性.

如图1所示，MarkLogic Server 共有三层架构，从下向上分别为数据层、搜索层和接口层[8].

首先数据层（Data Layer）是 NOSQL 数据库存储的基础，因为 MarkLogic 是基于文档存储的，所以在这一层主要负责存储和对文档的底层操作，这里有两种文档存储的

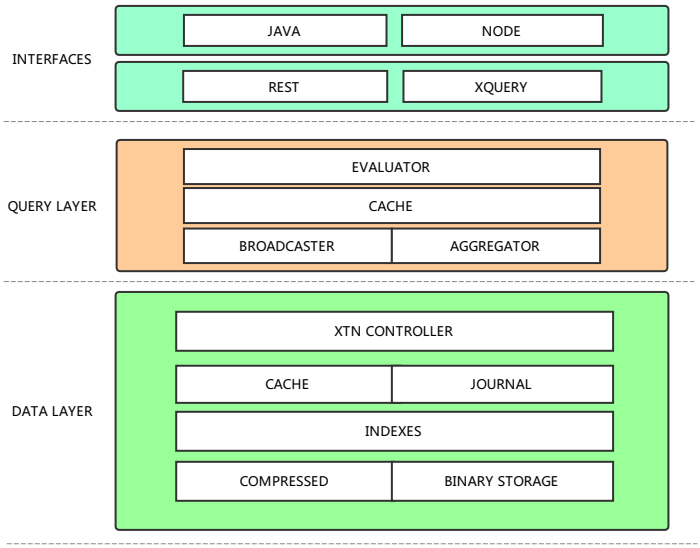

图 1: MarkLogic Server 整体架构图.

方式，分别为压缩后的文本文档，比如 XML 或者 JSON 数据和二进制文档存储，比如图像、pdf 文档等，为了高效的访问文档数据，这里需要为文档建立适当的索引，比如，全文索引 (Full-Text Index)，单词索引 (Word Index)、值索引（Value Index）、短语索引 (Phrase Index) 以及强大的通用索引 (Universal Index)、路径索引（Path Index）、安全索引（Security Index）等等. 这里也需要为数据访问建立缓存机制（Cache），Joural 是用来整合事务处理的组件. 最上层为事务控制层（XTN Controller）用来管理上一层发来的事务处理请求.

接下来搜索层主要负责将上一层发过来的搜索请求进行格式转化和转发. 其中 BROADCASTER 主要负责将转化后搜索请求转发给 MarkLogic Server 或者 Cluster，而 AGGREGATOR 主要负责将各个 MarkLogic Server 检索后的数据进行整合（比如：map-reduce），同样地，为了访问的高效，需要建立缓存为搜索层，增加了重复搜索的检索速度. 最上层为 EVALUATOR，主要是将上层的搜索请求，解析为内部使用的 XSL 语言格式，并且对其优化.

最上层是 API 层，MakrLogic 内部有一个 HTTP 接口，支持 REST 这样的 SOA 编程范式和软件架构，使得上层可以支持 Node 和 Java 语言的编程接口，并且 MarkLogic 支持原生的 XML 检索语言 XQuery 并且在此基础进行了改良.

2.4 MarkLogic 主要特性

作为 NoSQL 中的 Multi-Model 数据库，MarkLogic 不仅像 NoSQL 那样支持灵活的数据模型和可扩展等特点，它还具备企业级数据库所应该具备的特点，比如 ACID，安全，MVCC 等. 本小节主要从如下几部分进行重点介绍：

- 灵活的数据模型
- 强大查询和搜索（同一个意思）
- 语义（Semantics）
- ACID 事务

2.4.1 灵活的数据模型

Marklogic 可以直接存储多种数据类型，JSON、XML、RDF、坐标、二进制数据（PDF、图片、视频）. 因此，用户可以方便的存储数据，并在之后对其作出修改. 不用像关系型数据库那样需要提前定义好数据库对象和所使用的数据仓库技术（ETL[1]）. 使用 Marklogic，用户可以按照数据原始格式进行存储，而免于繁重的 ETL 过程.Marklogic 会把结构化数据和非结构化数据（数据与元数据）存储在同一个数据库中. 如果在之后存储来自不同数据源的不同结构数据，都是可以的.

图 2: MarkLogic 灵活的数据模型.

Marklogic 主要存储 JSON 和 XML 文档数据. 在 NoSQL 数据库中，文档模型最流行，它可以解决许多关系型数据库无法处理的问题. 文档对于多样而复杂的数据来说，很有帮助，它是可读的，可以很接近业务模型，从而避免关系型数据的阻抗失衡（impedance mismatch）问题. 由于文档模型可以很容易地将多种不同数据结构的数据存储在同一数据库中，因此数据集成变得容易而高效. 因为 Marklogic 的“Ask Anything”通用索引，用户可以很方便的在整个数据库中及时查询到数据的结构和内容.

2.4.2 强大的搜索机制

大多数数据库将搜索和查询分为两个不同的功能. 而 MarkLogic 将搜索和查询放在一起，它们是等价的. 这意味着用户不需要固定在单独的搜索解决方案上，也不必担心

[1]ETL，是英文 Extract-Transform-Load 的缩写，用来描述将数据从来源端经过抽取（extract）、交互转换（transform）、加载（load）至目的端的过程. ETL 一词较常用在数据仓库，但其对象并不限于数据仓库.

何时以及如何构建正确的索引，或者如何利用这些索引来执行某些查询. MarkLogic 设计有超过 30 个复杂的索引，可以对其进行调整和优化，以尽可能快地完成最复杂的查询，而不需要重复数据，数据按原样进行采集并立即搜索.

复杂的索引意味着开发人员可以提出更难的问题并获得更快的解决方案. MarkLogic 对每种数据类型使用多种查询语言（用于 JSON 的 JavaScript，用于 XML 的 XQuery 和用于 RDF 的 SPARQL）. 这些查询语言支持跨非结构化内容的全文搜索，支持复杂查询所需的丰富查询功能，多种格式和类型（包括与 ESRI ArcGIS 和 Google Maps 的连接）的地理空间搜索，跨链接数据的语义搜索（类似于图搜索）以及用于运行大规模并行查询的 MapReduce 功能. MarkLogic 的独特功能之一就是设计索引，以便开发人员可以编写跨多个索引运行的复杂查询，而不会导致性能瓶颈. 使用 MarkLogic，用户可以按照数据的原始格式进行查询，或者就地转换和管理数据，而这些都可以通过保持完整 ACID 属性的事务处理系统来实现.

图 3: MarkLogic 强大的搜索引擎.

2.4.3 多模型的语义搜索

语义提供了一个通用框架来描述和关联不同的数据，以便更好地理解和全面搜索，从而使开发人员和计算机能够观察并发现数据中的关系. MarkLogic 提供了存储和查询关联数据的功能，包括一个本地 RDF 三重存储，用于存储和管理数百亿可以用 SPARQL 查询的三元组，所有这一切都在 MarkLogic 内部进行. 不仅如此，MarkLogic 将三重存储与其文档存储结合在一起，提供了将文档，数据和三元组一起存储和管理的功能，以便用户可以在上下文中发现，理解和做出决策.

2.4.4 ACID 事务

Marklogic 是一个完全支持事务的 NoSQL 数据库，从一开始就支持 ACID 事务. 包括多文档事务（multi-document transaction）、多语句事务（multi-statement transaction）、XA 事务（transactions across a cluster）. 这一特性使 Marklogic 不同于其他文档数据库（MongoDB），这也使得它能够稳定地运行大规模任务密集型任务.

在关系型数据库领域，ACID 事务早已经成为标准，可以保证数据不丢失. 对于一些涉及到金融数据、健康数据、国家安全数据这样的关键应用，数据一致性显得格外重要. 当然，就算不是数据敏感性应用，事务性数据库还是能帮助开发者节省开发时间，还能避免一些数据丢失的麻烦. 很多全球性企业选择 Marklogic，正是因为其 ACID 特性. 当然还有他的政府级安全、高可用性、灾备机制等等.

Marklogic 实现 ACID 使用的是 MVCC（multi-version concurrency control）. 对 MVCC 来说，每一次更新都会对应有时间戳到相应的文档，数据库根据这些时间戳去确保数据的一致性. 这样的设计还有附加的好处：能保证高的数据吞吐量，更少的锁，可以避免数据冲突，还能方便快捷地数据恢复. ACID 的一致性还能提供更为强大的功能：高可用性、灾难恢复、分层存储.

2.4.5 增量备份

在 Marklogic 上，通过增量备份机制，用户可以更快速地执行备份，并且只需要备份自上次增量备份或完全备份以来的更改，从而可以使用更少的存储空间. 更频繁的备份也意味着更小的备份窗口（backup window），从而减少 CPU 和存磁盘 I/O 的消耗，使基础架构能够更快地用于其他工作负载. 这也减少了灾难发生时数据库的恢复点目标. 使用 MarkLogic，用户可以每天进行多项增量备份，并且对数据库性能影响很小.

日志归档是一个现有的功能，可与备份机制一起使用，以实现细粒度的时间点恢复. 但是，日志档案占用了相当多的存储空间. 现在，通过增量备份，可以自动清除日志以及增量备份，从而以更低的成本实现时间点恢复. 例如，将每日增量备份与 24 小时日志归档保留结合使用时，可以在过去 24 小时内恢复到任意时间点.

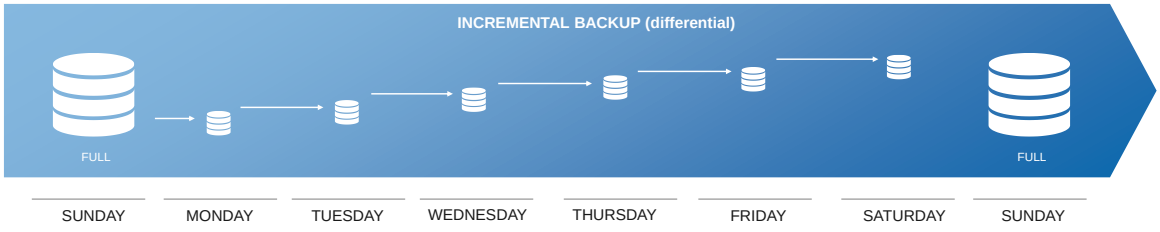

图 4: MarkLogic Server 的增量备份.

3 MarkLogic 中的索引

Marklogic 数据库本身支持搜索引擎，它可以从数据仓库中加载数据并在整个数据库中查询. 对于标准的查询，Marklogic 只需要花费少量的时间，便可以构建索引，而不像有些数据库，需要额外的搜索引擎来支持全文检索. 在 MarkLogic 内部建立了 30 多个索引，其中包括通用索引、范围索引、地理空间索引和安全索引等等，本部分主要描述 MarkLogic 内部的索引.

3.1 通用索引

通用索引是主要的 MarkLogic 索引, 也被称作自由发问式（Ask Anything）通用索引, 默认情况下始终打开. 它使用倒排索引和术语列表来标记数据库中的所有内容，包括结构化数据和非结构化数据. 通用索引使得 MarkLogic 的搜索和查询功能变得十分强大. 另外通用索引对其他元素也是有用的，比如集合，目录和安全规则. 通用索引的内部技术实现参考 Inside MarkLogic Server[3].

通用索引有如下优点：

- 范围广：利用索引使查询贯穿整个数据库
- 规模大：数据量大到拍字节/亿级别文档
- 新数据获取时，不限制次数的及时提醒
- 快：亚秒级出结果

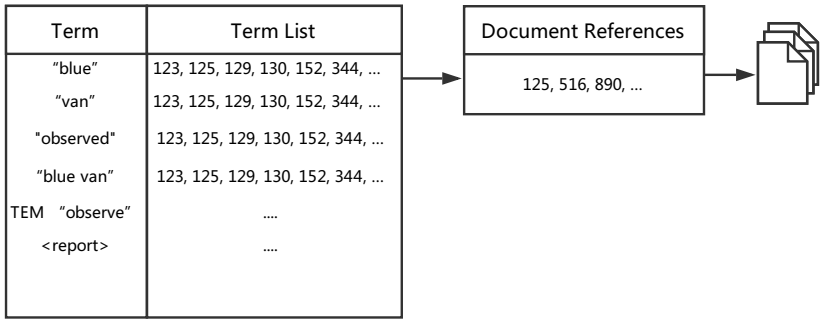

图 5: MarkLogic Universal Index.

3.2 范围索引

范围索引对于各种事情都很有用，包括简单地快速搜索日期，并返回结果或从结果集的文档里提取信息. 对于每个范围索引，MarkLogic 创建的数据结构可以轻松做到两

件事：对于任何片段 ID 获取片段的范围索引值，或者对于任何范围索引值获取具有该值的片段 ID. 它工作速度非常快，不会浪费内存或磁盘.

另外，如上所示，范围查询可以与 MarkLogic 中的任何其他类型的查询结合使用. 假设您想要将结果限制为上述日期范围内的结果，但也要使用特定的元数据标记. MarkLogic 使用范围索引获取范围中的片段 ID 集合，使用术语列表获取带有元数据标签的片段 ID 的集合，并与 ID 集合相交以确定与 两个约束匹配的结果集合. MarkLogic 中的所有索引都是完全可组合的.

除了组织日期和文档 ID 之外，范围索引对于连接文档也很有用，这通过查找一组文档的 ID，然后将其作为约束提供给第二个查询来完成. 而且，这对分类信息也很有用，并且是启用方面的索引，这是 MarkLogic 搜索功能的关键之一. 例如，如果我们想按类别组织搜索结果并在左侧显示结果（类似于亚马逊显示产品的方式），但在 MarkLogic 中，您可以全部实时构建.

关于范围索引的另外一件事是，它也非常快，不会浪费任何空间，因为它实际上是由两个数据结构实现的，一个写入磁盘，另一个映射为高效访问. 另外范围索引对 order by 进行了优化.

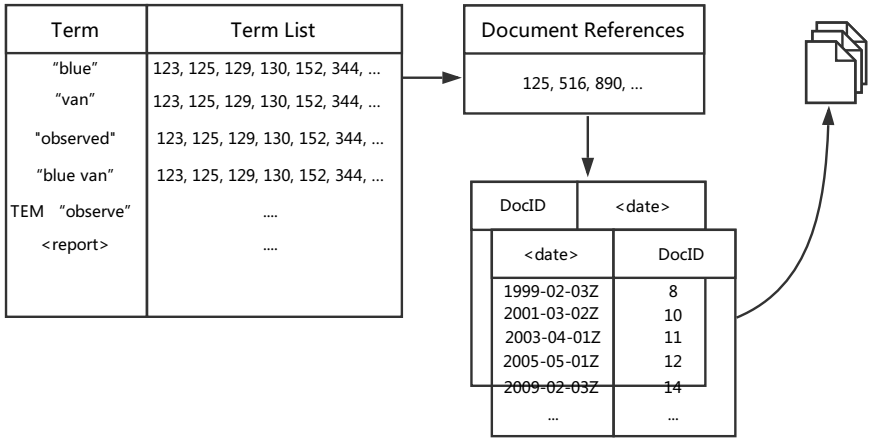

图 6: MarkLogic Range Index.

3.3 地理空间索引

地理空间索引与范围索引相似，内置对点，框，圆，线条和复杂多边形的支持. MarkLogic 还支持多种地理空间数据类型，如 GML，KML 和 GeoRSS. 在图7中，地理空间索引可以与其他索引一起使用，以进一步缩小结果集的范围.

如果用户需要可视化数据，则可以使用多层数据生成显示信息的地图，并包含弹出窗口，让用户可以进一步理解数据和文档. 用户可以对数百万个文档进行全文搜索，然

后对数据进行过滤和分层以创建新的视图，从而使用户的思维更加开阔. MarkLogic 集成了多种地理意识产品，如 Esri ArcGIS，Google Earth，Google Maps，Yahoo Maps 和 Microsoft Bing Maps.

地理空间索引另一个功能是实时警报. 如果导入的新数据与用户保存的查询匹配时，它将会实时推送警报，提供甚至可以通过文本或电子邮件发送给用户的信息流，这是非常有用的功能. 启用后，地理空间查询将被编入索引，并在新信息到达时立即路由. 随着警报查询数量的增加，检查新信息所需的时间保持不变，这可以使得 MarkLogic 能够同时处理数百万个不同的警报，并仍保持高性能.

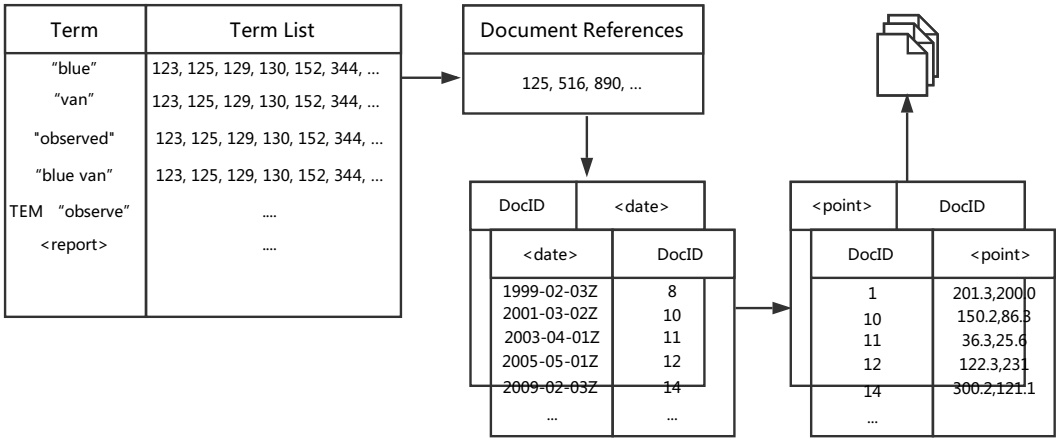

图 7: MarkLogic 地理空间索引.

3.4 三重索引

除了支持通用索引、范围索引和地理空间索引外，MarkLogic 还支持三重索引（triple index），该索引用于驱动 MarkLogic 更好的存储 RDF 三元组和管理 RDF 三元组的语义动能.RDF（资源描述框架）三元组就是语义学里面的主语、谓语和宾语，RDF 三元组本身就是 MarkLogic 的一种数据模型，直接可以存储在 MarkLogic 数据库中，并且用户可以使用 SPARQL[2]进行查询.

三元索引用于索引在文档中任何位置找到的模式有效的 sem:triple 元素。当包含三元组的文档被导入 MarkLogic 或数据库重新索引期间时，将会对三元组建立索引。三重索引在字典中仅存储一次，并且唯一。字典给每个值一个 ID，然后三元组使用该 ID 来引用字典中的值。

[2]SPARQL 是用于资源描述框架的查询语言，被认为是语义网科技的一个关键. 2008 年 1 月 15 日，SPARQL 正式成为一项 W3C 推荐标准. 同时资源描述框架（RDF）也是用于描述网络资源的 W3C 标准.

sem:triple 元素的有效性是通过检查文档中的元素和属性是否与/MarkLogic/Config/semantics.xsd 定义 sem:triple 的格式是否一致来确定的。如果 sem:triple 元素有效，则在三重索引中创建一个条目，否则该元素将被跳过。与范围索引不同，三重索引不需要装入内存，所以前期内存分配很少。

4 MarkLogic 的 API

Marklogic 从 Server-Side JavaScript、Server-Side XQuery、REST 、Java、Node.js 五个维度分别介绍用各自风格开发 Marklogic 应用的 API[1]. 在这里主要说明 Java 客户端和 XQuery 服务端部分的知识. 为了能够更好的理解这些 API，这里给出一些基础知识、MarkLogic 基本概念和如何利用官方的 API 快速的实现数据的增删改查任务，主要内容有：

- 基础知识
- XQuery 服务端 API
- Java 客户端 API

4.1 基础知识

4.1.1 XML

XML（Extensible Markup Language）可扩展标记语言，设计之初就是为了存储和传输数据，标签本身没有被预定义，符合 w3c 标准. 同时为了让 XML 便于解析，XML 语法有如下规定：

- 任何起始标签都应有结束标签. Marklogic
- 可以采用另一种简化语法，可以在一个标签中同时表示起始和结束标签.
- 标签必须按合适的顺序进行嵌套，所以结束标签必须按镜像顺序匹配起始标签.
- 所有的属性都必须有值.<input name=”text”>
- 所有的属性都必须在值的周围加上双引号.

另外 XML 文档必须要有声明，声明这是一个 XML 文件，最简单的声明：<?xml version=”1.0” ?>，目前只有 1.0 版本，还可以规定文档编码 <?xml version=”1.0” encoding=”utf-8” ?>

比如下面就是一段 XML 文档.

```
<?xml version="1.0" encoding="utf-8" ?>
<books xmlns='http://www.csdn.net/example/book'>
    <book id="1010596200" category="编程">
```

```
        <title>Java编程思想</title>
        <author>埃克尔</author>
        <pubDate>2007-6-1</pubDate>
        <pages>880</pages>
        <price>30.0</price>
    </book>
    <book id="1010696100" category="文学">
        <title>红楼梦</title>
        <author>
            <firstAuthor>曹雪芹</firstAuthor>
            <secondAuthor>高鹗</secondAuthor>
        </author>
        <pubDate>2012-9-1</pubDate>
        <pages>1606</pages>
        <price>41.5</price>
    </book>
</books>
```

4.1.2 XPath

XPath 即为 XML 路径语言（XML Path Language），它是一种用来确定 XML 文档中某部分位置的语言. XPath 基于 XML 的树状结构，提供在数据结构树中找寻节点的能力.

XPath 的节点类型说明和语法的使用，分别见表2和表3：

节点	英文名	例子
元素	element	<title>
属性	attribute	id='1010596200'
文本	text	红楼梦
命名空间	namespace	xmlns='http://www.csdn.net/example/book'
处理指令	processing	强大的函数库
注释	comment	<!–这是另一本书的信息–>
文档节点	root	即根节点 <books>

表 2: XPath 的七种类型的节点.

4.1.3 XQuery

Oracle, SQL, data $\Longleftrightarrow$ MarkLogic, XQuery, content

表达式	描述	实例
nodename	选取此节点的所有子节点.	books
/	从根节点选取.	/books
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置.	books//title
.	选取当前节点	./book
..	选取当前节点的父节点.	..
@	选取属性	//@category

表 3: XPath 语法.

XQuery 相对于 XML 的关系，等同于 SQL 相对于数据库表的关系. 具体使用请参考http://www.w3school.com.cn/xquery/.

4.2 XQuery 在 MarkLogic 数据库中的应用

MarkLogic 将 XQuery 作为其本地接口查询语言，因此安装完 MarkLogic 后，可以在 MarkLogic 控制台上使用 XQuery 对数据库的数据进行管理本节将会简单的分析和应用官方给出的 API，这里主要讲解如何使用 Xquery 进行插入数据.

可以通过查询官方的 XQuery 开发文档获得 API, 下面是官方给出的利用 XQuery 插入数据的 API.

```
xdmp:document-insert(
  $uri as xs:string,
  $root as node(),
  [$permissions as
   element(sec:permission)*],
  [$collections as xs:string*],
  [$quality as xs:int?],
  [$forest-ids as xs:unsignedLong*]
) as empty-sequence()
```

比如下面一段程序就是使用 XQuery 向 MarkLogic 数据库插入数据.

```
xquery version "1.0-ml";
declare namespace html = "http://www.w3.org/1999/xhtml";
xdmp:document-insert("/books.xml",
<books>
    <book id="1010596200" category="编程">
        <title>Java编程思想</title>
        <author>埃克尔</author>
        <pubDate>2007-6-1</pubDate>
```

表达式	描述
$uri	文档存入的路径
$root	插入文档的内容
[]	表示该参数为可选参数.
$permissions	插入的文档该有的权限. 默认取 xdmp:default-permissions() 权限. 文档的创建者至少拥有文档的 update 权限.
$collections	插入的文档所属的 collection. 默认取 xdmp:default-collections()
$quality	搜索时，该值影响搜索结果的相关性，正数增加相关性，负数减小相关性. 默认值为 0
$forest-ids	指定文档插入的 forest. 指定多个时，只插入其中一个，所指定的 forest 中至少有一个可以做 update. 默认插入该数据库所 merge 的 forest

表 4: XQuery 插入数据解释.

```
        <pages>880</pages>
        <price>30.0</price>
    </book>
    <book id="1010596100" category="编程">
        <title>Scala编程思想</title>
        <author>埃克尔</author>
        <pubDate>2015-6-1</pubDate>
        <pages>880</pages>
        <price>30.0</price>
    </book>
</books>,
xdmp:default-permissions(),
xdmp:default-collections()
)
```

4.3 Java Clinet API 介绍

使用 MarkLogic 的 Java API 接口可以很简单的对数据库的数据进行增删改查操作 (CRUD)，为了很快进入到 MarkLogic 的 Java 开发，本小节将使用典型的数据库持久层操作 CRUD 来对其 Java API 进行展开讲解[6].

Java Client API 为 MarkLogic 提供了开箱即用的数据管理，搜索和警报等功能. 它提供了一个纯 Java 查询构建器，并为 Java 开发人员提供了熟悉的 POJO 操作，使得操纵 JSON，XML 和二进制对象提供了便利. 对于需要运行服务器端代码的额外性能优势的应用程序，它包含内置的可扩展性，用于将关键性能的代码移至数据库. Java 客户

端 API 是开源的，并托管在 GitHub[7] 上.

4.3.1 Example: 向数据库插入 XML 文档

首先要初始化一个客户端对象，并传递认证信息，将连接信息存储在 com.marklogic.client.DatabaseClient 对象中.

```
DatabaseClient client = DatabaseClientFactory.newClient(host, port, user,
    password, authType);
```

创建完成后，使用 DatabaseClient 对象创建一个文档管理对象，即 com.marklogic.client.document.DocumentManager 对象，用来访问数据库保存的 XML，文本，JSON，二进制文件等文档.

```
XMLDocumentManager docMgr = client.newXMLDocumentManager();
```

接下来，从本地或者其他的文件系统中获取用户要插入到 MarkLogic 的文档，比如在本例，使用 InputStream 获取本地文件系统的文件内容.

```
FileInputStream docStream = new
    FileInputStream("data"+File.separator+filename);
```

然后初始化 InputStreamHandle 对象，即为 docStream 创建一个句柄，用于接收文档内容.

```
InputStreamHandle handle = new InputStreamHandle(docStream);
```

可以通过调用 DocumentManager 对象的 write() 方法向数据库写数据，接受的参数分别为文档的 URI（数据库中存储的地址）和句柄.

```
docMgr.write(docId, handle);
```

最后写入完成后，将会通过调用 DatabaseClient 的 release 方法释放掉数据库连接.

```
client.release();
```

完整的程序如下：

```
import com.marklogic.client.DatabaseClient;
import com.marklogic.client.DatabaseClientFactory;
import com.marklogic.client.document.XMLDocumentManager;
import com.marklogic.client.io.InputStreamHandle;

import javax.xml.xpath.XPathExpressionException;
import java.io.File;
```

```java
import java.io.FileInputStream;
import java.io.IOException;

public class DocumentWriter {
    public static void main(String[] args) throws XPathExpressionException,
        IOException {
        run(HOST, PORT, YOURUSERNAME, YOURPASSWORD);
    }
    public static void run(String host, int port, String username, String
            password) throws XPathExpressionException, IOException {
        DatabaseClient client = DatabaseClientFactory.newClient(
            host, port, new DatabaseClientFactory.DigestAuthContext(username,
                password));
        XMLDocumentManager docMgr = client.newXMLDocumentManager();
        String docId = "/book2.xml";
        String filename = "book1.xml";
        FileInputStream docStream = new FileInputStream("data" + File.separator
            + filename);
        if (docStream == null)
            throw new IOException("Could not read document example");
        InputStreamHandle handle = new InputStreamHandle(docStream);
        docMgr.write(docId, handle);
        client.release();
    }
}
```

4.3.2 Example: 从数据库读取 XML 文档

从 MarkLogic Server 读取文档的基本步骤如下：

1. 创建一个数据库客户端
2. 创建一个文档管理对象，用于处理 XML 文档
3. 创建一个句柄，用于接收文档内容
4. 读取文档内容
   (a) 对文档内容应用 XPath 解析
   (b) 访问解析后的数据
5. 释放数据库连接资源

完整的程序如下：

```java
import com.marklogic.client.DatabaseClient;
import com.marklogic.client.DatabaseClientFactory;
import com.marklogic.client.document.XMLDocumentManager;
import com.marklogic.client.io.DOMHandle;

import org.w3c.dom.Document;
import javax.xml.xpath.XPathExpressionException;
public class DocumentReader{
    public static void main(String[] args) throws XPathExpressionException {
        run(HOST, PORT, YOURUSERNAME, YOURPASSWORD);
    }
    public static void run(String host, int port, String username, String
        password) throws XPathExpressionException {
        DatabaseClient client = DatabaseClientFactory.newClient(host, port, new
            DatabaseClientFactory.DigestAuthContext(username, password));
        XMLDocumentManager docMgr = client.newXMLDocumentManager();
        String docId = "/books.xml";

        DOMHandle handle = new DOMHandle();
        docMgr.read(docId, handle);
        Document document = handle.get();

        String bookName = handle.evaluateXPath("string(/books/book/title)",
            String.class);
        String rootName = document.getDocumentElement().getTagName();
        System.out.println("(Strong Typed) Read " + docId + " content with the
            <"
              + rootName + "/> root element for the " + bookName + " book");
        client.release();
    }
}
```

4.3.3 Example: 从数据库删除 XML 文档

从 MarkLogic Server 删除指定的文档的基本步骤如下：

1. 创建一个数据库客户端
2. 创建一个文档管理对象，用于处理 XML 文档
3. 创建一个句柄，用于接收文档内容

4. 删除指定的文档（需要提供提供文档的 URI）
5. 释放数据库连接资源

完整的程序如下：

```
import com.marklogic.client.DatabaseClient;
import com.marklogic.client.DatabaseClientFactory;
import com.marklogic.client.document.XMLDocumentManager;
import javax.xml.xpath.XPathExpressionException;
public class DocumentDelete {
    public static void main(String[] args) throws XPathExpressionException {
        run(HOST, PORT, YOURUSERNAME, YOURPASSWORD);
    }
    public static void run(String host, int port, String username, String
            password) throws XPathExpressionException {
        String filename = "books.xml";
        DatabaseClient client = DatabaseClientFactory.newClient(host, port, new
            DatabaseClientFactory.DigestAuthContext(username, password));
        XMLDocumentManager docMgr = client.newXMLDocumentManager();
        String docId = "/" + filename;
        docMgr.delete(docId);
        System.out.println("Delete " + docId + " document successfully");
        client.release();
    }
}
```

4.4 搜索处理模型

（1）使用 MarkLogic 提供的组合构造器构造指定的搜索，MarkLogic 有如下几种构造器

```
cts:and-query(("wrist", "injury"))
cts:or-query((cts:and-query(("cat", "scratch"))
          cts:and-query(("dog", "bite")) ))
cts:and-not-query( “United States” , "Texas")
cts:element-query(xs:QName("Year"),
          cts:or-query(("1980", "1981")))
```

（2）定义要搜索的节点位置/URI

```
//MedlineCitation[Journal/JournalIssue/PubDate/Year = "1980"]
```

（3）将搜索应用到节点上

```
cts:search(//MedlineCitation,
           cts:and-query(("wrist", "injury")))
```

（4）按搜索结果的关联度返回结果

4.4.1 文本搜索

Marklogic 的查询功能相当丰富，API cts 的大部分方法都为查询服务，还有 search, 方法不多，但是有些很重要. 本节主要介绍在文本搜索中 cts 的部分 query 函数和 search 下比较重要的函数. 下面是 cts:word-query 的 API.

```
cts:word-query( $text as xs:string,
                [$options as xs:string*],
                [$weight as xs:double] )
as  cts:word-query
```

文本查询主要有两个个参数，第一个参数是要匹配的字符串，如果指定了多个字符串，那么只要有一个匹配到，那么查询也就匹配到了结果. 第二个参数是 weight. 即此查询的权重.

4.4.2 布尔查询

- cts:and-query() 所有子查询的交集
- cts:or-query() 所有子查询的并集
- cts:and-not-query() 两个子查询的差集
- cts:not-query() 单个子查询的补集

4.4.3 邻近查询

```
cts:near-query($queries, $distance, $ordered, $weight)
```

如果匹配到了结果并且结果与查询之间的距离小于等于指定的距离，则结果匹配. 如果指定距离为 0，那么只有当文本完全匹配时才能匹配. 默认值是 100.

4.4.4 XML 结构查询

主要有三种查询，分别为查询符合某元素值的 cts:element-value-query，查询符合某属性的 cts:element-attribute-value-query，查询不符合某条件的 cts:not-query 等等. 下面 3 个函数使用的时候需要注意各自的不同：

```
cts:element-value-query(
```

```
  $element-name as xs:QName*,
  [$text as xs:string*],
  [$options as xs:string*],
  [$weight as xs:double?]
) as cts:element-value-query
```

这里指定了元素的值，说明只有元素的值完全符合条件才会返回数据，可以理解为此时为“exact”匹配，当然 $options 可以设置是否大小写敏感，是否空格敏感. 比如 cts:search(//book,cts:element-value-query(xs:QName(”title”),”java 核心技术”)) 会返回 id=1010596101 的 book，但是将其中的“java 核心技术”改为“Java 核心技术”, 只是有一个大小写不同，则查询不到结果. 如果只指定元素名，而不指定其值，则返回包含该元素的所有文档.

```
cts:element-word-query(
  $element-name as xs:QName*,
  $text as xs:string*,
  [$options as xs:string*],
  [$weight as xs:double?]
) as cts:element-word-query
```

这里必须指定元素名和一个 word, 只要该词出现在这一元素的值中，则会返回数据. cts:search(//book,cts:element-word-query(xs:QName(”title”),”java”)) 这里会返回 id=1010596101 和 id= 1010596200 两条数据. 后者的标题中”java”有大写字母，但是可以识别到.

```
cts:element-range-query(
  $element-name as xs:QName*,
  $operator as xs:string,
  $value as xs:anyAtomicType*,
  [$options as xs:string*],
  [$weight as xs:double?]
) as cts:element-range-query
```

这里还要传一个操作符 $operator（支持的操作符 [<, <=, >, >= ,!=, =]），这个函数可以返回符合某一范围的所有数据，较为常用的是找出某一时间点之前或之后的数据，或者在某一数值范围内的数据. cts:search(//book,cts:element-range-query(xs:QName(”price”), ”>”,50.0)) 可以找出价格大于 50 元的书籍信息.

5 数据库比较

5.1 MarkLogic 与 MongoDB 的对比

MarkLogic 主要是文档数据库，而 MongoDB 是文档型数据库的典型代表，但是 MongoDB 是开源的，免费的，如果要使用 MarkLogic 的全部功能，需要购买其商业授权. 在本篇主要会对面向文档数据的两个重要 NoSQL 数据库进行对比，并通过相应的性能测试，利用测试数据对两者进行客观的比较[5][2].

5.1.1 对事务的支持

MarkLogic 作为一个商业类型的 NOSQL 数据库，它是完全支持事务 ACID 的. Marklogic 使用 MVCC 多版本并发控制来实现的 ACID，因此可以无锁访问数据. 例如：当文档正在写的时候，可以直接读文档，不会因为读操作而使写操作暂停. 而 MongoDB 采用最终一致性，如果多个 MongoDB servers 在一个 Cluster 中运行时，MongoDB 的默认设置无法保证数据更新的稳定性，也无法保证更新数据时读操作得到的数据在所有 server 上都一致.

5.1.2 分片和集群

企业级 NoSQL 数据库应该是可以通过新增服务器到集群来扩大规模. Marklogic 可以利用云服务器或者实体服务器实现，它的自动化工具保证服务的稳定性，而且 Marklogic 是集合了应用服务器，数据库，搜索引擎于一体，对于管理员来说，拓扑更为容易.MongoDB 此种情况下对硬件的要求就比较高. 创建集群需要考虑片键，片键可以使数据进入分布环境下的不同服务器，这样可以优化读和写操作. 但问题是，片键一旦设置好以后，就不能更改，想要改变就需要把数据复制到另一片空间. 这样的分区影响了基于时间点的数据恢复，使文档孤立等.

5.1.3 索引

当插入一个文档到 Marklogic，索引将自动创建，且 MarkLogic 内部有 30 多个索引，对于快速查找和搜索数据库的信息十分高效. 使用索引可快速访问数据库中的特定信息. Marklogic 提供的通用索引可以检索文档中是否包含某个单词或者短语，范围索引可以检索满足代数条件的结果，地理空间索引可以检索出地理坐标位置. MongoDB 只有一个唯一索引和稀疏索引，查询最多同时使用两个索引，复杂查询时就需要混合索引，这样不利于结果的排序.

5.1.4 集合

基于文档的 NoSQL 解决方案需要适当的方式去存储并组织文档，关系型数据库将数据存在表结构中，非关系型则存在集合中. MongoDB 只允许用户将文档存储在一个

集合中，如果想要两个或以上的集合，就不行了. Marklogic 中的数据支持集合查询，更便捷.

5.1.5 检索

Marklogic 有强大的搜索功能，查询结果通过相关度排序，这使得用户最想要的结果出显示在最上面，当然这一结果是基于整个数据集的复杂算法而来. MongoDB 的查询接口有点像数据库的查询语句，仅支持简单的查询. 如果想实现强大的搜索，就需要第三方工具了，这样当数据改变的时候就需要考虑重建索引的开销. 而且这样的结果没有相关性，它只能硬性的符合检索条件，排序也需要手动指定.

5.1.6 安全

安全性是 MarkLogic 最为突出的一个特点.Marklogic 是六个拥有此认证的数据库管理系统供应商之一，也是唯一一个商用 NoSQL 公司，因此 MarkLogic 拥有市场领先的安全特性，足以确保数据安全.

5.1.7 其他

对于系统特性的比较，这里总结成一张表，如表5.

Name	MarkLogic	MongoDB
数据模型	Document store;Native XML DBMS;RDF store;Search engine	Document store;Key-value store
实现语言	C++	C++
API 和访问方式	Java Node ODBC RESTful SPARQL WebDAV XDBC XQuery XSLT	JSON
是否支持触发器	支持	不支持
分区方式	分片	分片
复制方式	主从复制	主从复制
是否支持 MapReduce	支持	支持
一致性原则	立即一致	最终一致
是否支持 ACID	支持	不支持
是否支持并发	支持	支持
是否支持持久性存储	支持	支持

表 5: MongoDB 与 MarkLogic 的对比.

5.2 MarkLogic 与 Redis 的对比 (可选)

这里将会对 MarkLogic 与 Redis 进行比较，我将使用 MarkLogic Server 作为比较. MarkLogic 是支持文档存储，搜索和三重存储等操作的混合 NoSQL 数据库. 它也可以用来存储任意的二进制数据以及文本数据，json 和 XML. 另外存储的许多源文档都是二进制的 - 比如图片，PDF. 这意味着它具有存储和检索任意数据的核心功能，就像 Key-Value 存储一样.

5.2.1 存储和获取

MarkLogic 可通过简单地使用它的 REST API 存储二进制数据. 通过设置 POST 请求头的 Content-Type 到地址/v1/documents?uri=/some/doc/uri.bin 进行存储二进制数据 uri.bin，另外通过需求设置 GET 请求的 MIME 类型. 使用 GET /v1/documents?uri=whatever 获取文档的原始内容.

MarkLogic 中的 URI 是我们的文档的标识符. 可以自己指定一个，或者让服务器随机生成一个. 在这种情况下，URI 会在 POST 响应中返回.

而 Redis 作为一个 kv 存储的典型数据库代表它往往通过 GET 和 PUT 命令获取和存储数据.

5.2.2 分片

在 MarkLogic 服务器集群中，接收新文档的机器会将文档随机放入集群服务器中的碎片（森林）中. MarkLogic Server 还有自动重新平衡的功能，如果不小心将所有文档添加到一台服务器，MarkLogic 整个群集会自动的重新平衡所有文档. 而 Redis 集群的重新分片操作可以将任意数量已经指派给某个节点的槽改为指派给另一个节点，并且相关槽所属的数据也会迁移到另一个节点上，另外重新分片时，集群的服务器可以线上重新分片，并且目标节点和源节点都可以继续处理请求，Redis 并没有自动平衡节点数据的功能需要手动指派需要分派的节点名字.

5.2.3 对值增加/减少

MarkLogic 可以存储 Json 和 XML 数据，这些文档可以很简单也可以非常复杂，但是每一个文档都有一点的开销，大约是 750bytes，所以存储 2bytes 的数据是非常浪费的，因此最好可以创建一个有很多值的文档，并且可以改变其中的某些部分，用户不要仅仅创建一个键值的文档，比如只有商品名字的商品，而是添加一些新的 KV，比如商品的生产日期，过期时间，条形码等等相关的数据.

为了管理和添加删除这些键值对，MarkLogic 内部的 REST API 提供了 PATCH 操作，用于对文档数据或者元数据进行更新，比如可以用于将值增加或减少 1 或其他任何值，也可以使用 Replace PATCH 机制，通过使用 ml.add 方法将值增加，MarkLogic 也提供了一些其他的操作，比如减、乘、除、表达式搜索与替换、连接字符串等等. 而

Redis 可以通过 INCREASE 或者 DECREASE 方法实现.

5.2.4 检查键是否存在

如果在 MarkLogic 中检索某个文档是否存在，可以简单请求/v1/documents?uri=whatever 的 HEAD 信息. 如果收到 404 响应，则表示该文档不存在. 但是如果要检索文档中某些值是否存在，稍微复杂些，由于 MarkLogic 会对加载到数据库文档的每个元素建立通用索引，因此如果要检查某个键是否存在，需要通过搜索之后的结果才能验证，但是这样搜索的往往是整个数据库的所有文档数据. 实际上需要做的是对特定的文档进行检查，因此需要通过 element-query 来检查，而对于 Redis 可以直接通过 GET KEY 检查，如果返回 KEY NOT EXIST 表示键不存在.

5.2.5 追加数据到列表

MarkLogic 也支持 Insert 操作，也可以支持特定位置进行追加数据类似于 Redis 的 APPEND 操作.

5.2.6 栈操作

MarkLogic 的插入操作允许用户可以将插入位置指定为'before' 或'after'，以允许将节点数据放置在列表的前端和末尾. MarkLogic 的删除模式可以删除列表的第一个元素 (/list/parent/child/fn:first()) 或者最后一个元素 (/list/parent/child/fn:last()) . 也可以获取一个特定的元素，而不是一个完整的文档. 为此，需要用户创建一个非常简单的转换（XQuery 或 XSLT 或 Server JavaScript），并将其与 GET /v1/documents?transform= mytransform 请求一起使用. 因此，不仅可以实现堆栈（push 和 pop 等同于'after' 和'fn:last' 操作），还可以实现 FIFO 和 FILO 队列！而 Redis 内部已经实现了堆栈的数据结构和相关的操作，所以相比 MarkLogic，Redis 是很简单的.

5.2.7 哈希操作

Redis 的哈希数据结构类似于 Java 中的 HashMap 数据结构，对于 KV 存储来说，其中的 Hash 数据结构应用到实际文档中就是 JSON 文档，对于 JSON 文档数据，MarkLogic 已经有现成的 API 来对其进行操作.

5.2.8 发布与订阅

MarkLogic 服务器包含一个' 反向查询' 索引. 这使用户可以进行任何搜索 - 包括对单个文档的任何更改或文档的某一部分的搜索 - 并对满足所有搜索条件的数据进行触发警报动作（alert）. 这些警报动作可以是 XQuery 脚本或 JavaScript 脚本，可以执行任何操作. 比如可以通过电子邮件服务器发送邮件或者发送到邮件服务器或者发布到 Web 服务上等等，因此可以实现发布与订阅功能. 比如使用 Node.js 创建了一个 W3C

WebSockets 服务器，它保持对客户端开放连接.Node.js 服务器也有一个 Web 服务端点. 当警报被触发时，MarkLogic 向该端点发送消息，Node.js 使用 WebSocket 将其传递给 Web 应用程序客户端. 但是与 Redis 相比，不是特别容易.

6 性能测试

6.1 Multi-Model 的基准测试设计

基准测试是评估数据库系统的常用方法，随着越来越多的平台被提出来能够处理多模型数据，具有可用于评估下一代多模型的数据库的性能和可用性的基准变得非常重要. 已经提出了许多可用于评估大数据系统（例如 YCSB，BigBench，TPCx-BB，Bigframe，BigDataBench）的基准. 不幸的是，那些通用的大数据基准并不适用于评估多模型数据库. 在这节中，我们认为，对多模型数据库系统进行全面评估会带来一些需要克服的新挑战. 首先，现有的多模型数据库的输入和输出是多种多样的. 由于现在没有可用的标准多模式查询语言，因此应开发，共享，统一和优化公开可用的基准数据和不同系统查询的实现. 其次，与关系世界不同，NoSQL 系统遵循*"data first, schema later or never"* 的范式. 对于严格的评估，必须有可能控制（并系统地改变）输入模式以及多模式数据的模式演变的复杂性. 最后，多模型数据库应该支持跨模型事务和一致性. 因此，必须精确地提出描述不同数据模型的一致性行为的新颖一致性度量.

本次测试主要对硬件的性能以及 MarkLogic 软件的功能进行评价. 测试的工作负载主要包括两个方面:

- 导入阶段 (Ingestion phase). 将一个大数据集插入到 MarkLogic 数据库索引中.
- 查询阶段 (Query phase). 对数据库进行搜索、视图更新以及删除数据操作.

另外本次基准测试将比较 MarkLogic 与 MongoDB 在数据库 CURD 的性能，涉及到非常简单的增删改查，主要测试数据为 json 数据.

6.2 测试环境

该 benchmark 是在单机环境下进行测试的，机器配置如下

- Intel(R) Core(TM) i5-4210 CPU 1.70GHZ 2.4GHZ
- 16GB of RAM

6.3 MongoDB 的 Benchmark

该项测试只测服务器的吞吐率，也就是客户端在服务器上读或者写数据到数据库或者从数据库中删除数据. 总数据条数 1million. 测试过程中可以改变不同参数的取值，来

测试特定参数对性能的影响. 使用测试工具 mongo-bench 来测试.mongo-bench 参数说明如下.

--host	测试目标（如127.0.0.1）
--port	测试端口（default 27017）
--db	操作数据库名称（默认mongobench）
--collection	操作数据库表（默认data_test）
--userName	用户名（如果有）
--passWord	密码（如果有）
--cpunum	多核设定（默认1）
--procnum	测试并发个数（默认4）
--datanum	每个线程插入数据条数（默认10000）
--logpath	日志路径（默认./log.log）
--jsonfile	希望插入document路径（不选用该参数则使用默认的插入格式）
--operation	测试模式（insert,prepare,query,tps,update）prepare 模式会在插入完成后为查询会用的项添加索引
--queryall	测试模式为query的时候，是否返回所有查询到的结果（默认false，即db.xx.findOne()）
--clean	是否清理数据(默认false，如果为true将drop数据库mongobench)

6.3.1 插入测试

首先清理数据库：

```
./mongo-bench --host 127.0.0.1 --clean true
```

再来进行插入测试：设置不同的线程/并发数插入相同量的数据，查看性能变化.

使用 4 核 cpu，2 个并发，每个并发插入 100000 条数据，日志输入为/tmp/log.log，插入的每条数据为./test_data.json 中的内容.

```
./mongo-bench --host 127.0.0.1 --datanum 100000 \
  --procnum 2 --cpunum 4 --jsonfile \
  ./test_data.json --operation insert
```

使用 4 核 cpu，4 个并发，每个并发插入 100000 条数据，日志输入为/tmp/log.log，插入的每条数据为./test_data.json 中的内容

```
./mongo-bench --host 127.0.0.1 --datanum 100000 \
  --procnum 4 --cpunum 4 --jsonfile \
  ./test_data.json --operation insert
```

使用 4 核 cpu，8 个并发，每个并发插入 100000 条数据，日志输入为/tmp/log.log，插入的每条数据为./test_data.json 中的内容

```
./mongo-bench --host 127.0.0.1 --datanum 100000 \
  --procnum 1 --cpunum 4 --jsonfile \
  ./test_data.json --operation insert
```

6.3.2 查询测试

首先清理数据库：

```
./mongo-bench --host 127.0.0.1 --clean true
```

接下来进行测试（limit one 的）：使用 4 核 cpu，8 个并发

```
./mongo-bench  --host 127.0.0.1 --datanum 100000 --procnum 1 \
    --cpunum 4 --operation query
```

在进行非 limit one 的：使用 4 核 cpu，8 个并发

```
./mongo-bench --host 127.0.0.1 --datanum 100000 --procnum 1 \
    --cpunum 4 --operation query  --queryall true
```

6.3.3 更新测试

进行更新测试

```
./mongo-bench --host 127.0.0.1 --datanum 1 --procnum 1 \
    --operation update
```

6.4 MarkLogic 的 Benchmark

该项测试只测服务器的吞吐率，也就是客户端在服务器上读或者写数据到数据库或者从数据库中删除数据. 总数据条数 1million. 测试过程中可以改变不同参数的取值，来测试特定参数对性能的影响. 使用测试工具 ml-bench 来测试.ml-bench 参数说明如下.

--host	测试目标（如127.0.0.1）
--port	测试端口（default 27017）
--userName	用户名（默认情况下为admin）
--passWord	密码（默认情况下为admin）
--auth	MarkLogic REST验证类型（默认为digest）
--cpunum	多核设定（默认1）
--procnum	测试并发个数（默认4）
--datanum	每个线程插入数据条数（默认10000）
--logpath	日志路径（默认./log.log）
--jsonfile	希望插入document路径（不选用该参数则使用默认的插入格式）

--operation	测试模式（insert,read,delete,query,update）
--queryall	测试模式为query的时候，是否返回所有查询到的结果（默认true）
--queryStr	要搜索的文本（结构化查询）
--clean	是否清理数据(默认false，如果为true将drop数据库Documents)

6.4.1 插入测试

首先清理数据库：

```
./ml-bench --host 127.0.0.1 --clean true
```

再来进行插入测试：设置不同的线程/并发数插入相同量的数据，查看性能变化.

使用 4 核 cpu，2 个并发，每个并发插入 100000 条数据，日志输入为/tmp/log.log，插入的每条数据为./test_data.json 中的内容.

```
./ml-bench --host 127.0.0.1 --datanum 100000 \
  --procnum 2 --cpunum 4 --jsonfile \
  ./test_data.json --operation insert
```

使用 4 核 cpu，4 个并发，每个并发插入 100000 条数据，日志输入为/tmp/log.log，插入的每条数据为./test_data.json 中的内容

```
./ml-bench --host 127.0.0.1 --datanum 100000 \
  --procnum 4 --cpunum 4 --jsonfile \
  ./test_data.json --operation insert
```

使用 4 核 cpu，8 个并发，每个并发插入 100000 条数据，日志输入为/tmp/log.log，插入的每条数据为./test_data.json 中的内容

```
./ml-bench --host 127.0.0.1 --datanum 100000 \
  --procnum 1 --cpunum 4 --jsonfile \
  ./test_data.json --operation insert
```

6.4.2 查询测试

首先清理数据库：

```
./ml-bench --host 127.0.0.1 --clean true
```

接下来进行测试（limit one 的）：使用 4 核 cpu，8 个并发

```
./ml-bench  --host 127.0.0.1 --datanum 100000 --procnum 1 \
    --cpunum 4 --operation query --queryStr edison
```

在进行非 limit one 的：使用 4 核 cpu，8 个并发

```
./ml-bench --host 127.0.0.1 --datanum 100000 --procnum 1 \
    --cpunum 4 --operation query  --queryall true
```

6.4.3 更新测试

进行更新测试

```
./ml-bench --host 127.0.0.1 --datanum 1 --procnum 1 \
    --operation update
```

6.5 BenchMark 测试结果

	插入操作	查询操作	删除操作	更新操作
MongoDB	9s	67s	≃1s	70s
MarkLogic	39s	277s	≃1s	46s

表 6: BenchMark 测试结果 (10000 条数据).

	插入操作	查询操作	删除操作	更新操作
MongoDB	18s	130s	≃1s	141s
MarkLogic	75s	600s	≃1s	93s

表 7: BenchMark 测试结果 (20000 条数据).

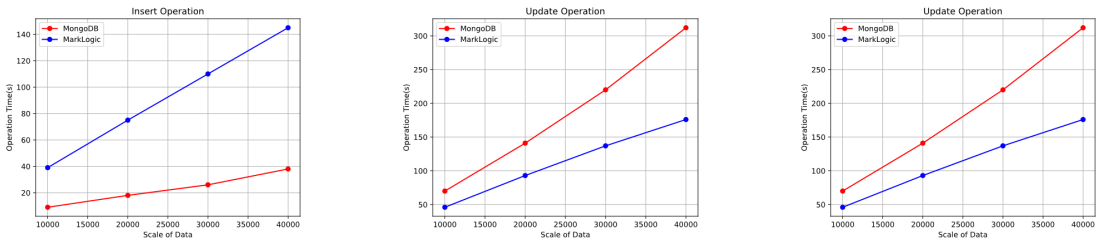

图 8: MarkLogic 与 MongoDB 的性能对比图

7 结论

本报告主要分析了 MarkLogic 的发展背景、技术背景、内部系统实现与底层的架构设计，除了上述的介绍外，还针对 MarkLogic 的编程接口进行必要的整理和介绍以及对 MarkLogic 与 Redis、MongoDB 进行了系统的对比，比如支持的数据模型、数据操

作等等，最后通过设计简单的 Benchmark 对支持文档数据库 MarkLogic 和 MongoDB 进行了性能测试，通过测试结果，可以得出在文档数据库的插入删除更新以及搜索方面，两者的表现几乎持平，但是 MarkLogic 在查询与搜索方面更加的灵活，另外安全机制比 MongoDB 高，又因为 MarkLogic 数据库是企业级数据库，在事务处理等方面都要比 NoSQL 数据库 MongoDB 要好很多. 但是 MarkLogic 不是开源的数据库系统，并且 MarkLogic 由于对事务的完美支持，使得它在性能方面要稍差于其他的 NoSQL 数据库.

通过本次调研，了解到多模型数据库是数据库发展的趋势，通过多数据模型与数据库平台的有效整合，可以使得开发人员更加的关注于业务，而非数据模型本身.

参考文献

[1] CSDN 用户 cj96248. Xquery 增删改查. `https://blog.csdn.net/jiangchao858/article/details/50361880`. Accessed June 1, 2018.

[2] db engines.com. Marklogic vs. mongodb comparison. `https://db-engines.com/en/system/MarkLogic%3BMongoDB`. Accessed June 1, 2018.

[3] Jason Hunter and Mike Wooldridge. Inside marklogic server, 2011.

[4] Jiaheng Lu and Irena Holubová. Multi-model data management: What's new and what's next? In *EDBT*, pages 602–605, 2017.

[5] MarkLogic. How is marklogic different from mongodb? `https://developer.marklogic.com/blog/how-is-marklogic-different-from-mongodb`. Accessed June 1, 2018.

[6] MarkLogic. Java application developer's guide —marklogic 9 product documentation. `http://docs.marklogic.com/guide/java`. Accessed June 1, 2018.

[7] MarkLogic. Marklogic java client api. `https://github.com/marklogic/java-client-api`. Accessed June 1, 2018.

[8] MarkLogic. Reference application architecture guide. 2017.

[9] Dan McCreary and Ann Kelly. Making sense of nosql. *Shelter Island: Manning*, pages 19–20, 2014.

[10] Microsoft. Marklogic recognized as a visionary in the 2017 gartner magic quadrant for operational database management systems. 2018.

[11] Noel. Yuhanna. The forrester wave™:big data nosql, q3 2016. In *Forrester. Forrester Research*, 2017.