《七周七数据库》读书报告

李博杰 PB10000603

《七周七数据库》（英文：Seven Databases in Seven Weeks）出版于2012年，介绍了时下最热门的7种开源数据库，包括关系数据库（PostgreSQL）、key-value数据库（Riak、Redis）、列式数据库（HBase）、面向文档的数据库（MongoDB、CouchDB）、图数据库（Neo4j）。除了PostgreSQL以外，其他6种数据库可以统称为NoSQL，即它们不使用关系模型，也不把SQL作为查询语言。

这本书模仿了《七周七语言》的体例，每种数据库一章，划分为三节，名为第一天、第二天、第三天。我没有时间来尝试每种数据库，不过我通读了全书，并总结了每种数据库的特性。

多年来，以Oracle、MySQL为代表的关系数据库一直是事实上的选择，也是我们数据库课程所学习的内容。然而，随着Web和大数据的发展，关系模型暴露出了一系列问题：

1. 模式不易修改。关系模型要求事先指定实体、属性和关系，而在复杂多变的Web环境中，经常需要增加或修改属性和关系，甚至不同的实体需要不同的属性集，这在关系模型中并不容易实现。
2. 严格的一致性模型和持久性牺牲了速度。一些Web应用只要求所谓的“最终一致性”，即可以容忍事务处理过程中短暂的数据不一致，此时关系数据库严格的一致性模型影响了速度。类似地，一些数据库只是作为缓存，或者对少量数据丢失不敏感，此时大量数据可以保存在内存中而不必随时写入磁盘。一些NoSQL数据库利用了不严格的持久性需求，将大量操作放在内存中完成，吞吐量提升了一个数量级。
3. 严格的一致性模型牺牲了可用性或分区容忍性。分布式数据库的经典定理是“CAP定理”：一致性（Consistency）、可用性（Availability）、分区容忍性（Partition Tolerance）三者只能居其二。关系数据库往往最关注一致性，而在大数据条件下分区容忍性是首要考虑的。因此在NoSQL中，往往需要牺牲一定程度的一致性来保证可用性和分区容忍性。

由于PostgreSQL是经典的关系数据库，《七周七数据库》在介绍关系模型之余，只是讲解了一些PostgreSQL特有的全文检索和多维查询功能，本文略去。

# Riak

 Riak是一种分布式的key-value数据库。Riak的一个实例有若干bucket（类似关系数据库中的表），每个bucket中可以有很多key（类似关系数据库中的行），value是任何类型（由HTTP header中的Content-Type指定）的数据。

 查询语言是HTTP REST，即通过HTTP GET（类比SQL中的SELECT）、POST（INSERT）、PUT（UPDATE）、DELETE（DELETE）实现CRUD操作。数据类型和一些选项也是通过HTTP header指定的。

 当然，作为NoSQL，Riak不支持关系数据库中最重要的关系操作。

### 容错性

 容错性是Riak最重要的特色。Riak的灵感来源于Amazon Dynamo的论文，它的关键贡献是在CAP（一致性、可用性、分区容忍性）之间实现tradeoff。Riak有三个参数：

* N：集群中的副本数量，即一次写入最终复制到的节点数量。注意，N可以小于集群中的节点数量。
* W：一次写入在返回响应之前，必须成功写入的节点数量。如果W小于N，则成功写入了W个节点后就会对客户端返回成功，此时剩下N-W个节点还在复制数据。
* R：成功读出一项数据所需的节点数量，即从R个节点读出数据，并选择时间戳最新的（稍后详述）。为什么要从R个节点（而非1个）读出数据呢？因为当W小于N时，有可能写操作事实上还没有完成就返回成功了，此时开始了读操作，读到的可能恰好是旧值。如果R+W>N，可以证明读出的一致的数据一定不会是旧值。

N、R、W三个参数有一些经典的配置：

* W=N，R=1：这就是关系数据库的做法，通过确保写操作在返回之前完成来保证一致性，但写入比较慢。
* W=1，R=N：使写入最快，牺牲读取速度。
* W = R = N/2+1：分担介于读写之间的延迟。

值得一提的是，N、R、W三个参数可以对每个key采取不同的设置，甚至可以在读请求（GET）的时候传入参数指定R。这个Amazon Dynamo论文的深刻观点使得Riak可以足够灵活地处理不同一致性需求和读写比例的数据。

### 向量时钟

前面提到，R>1时要决定哪个读出的版本是最新的。这是分布式系统中的根本问题。最简单的解决方案当然是所有机器的时间精确同步，给数据打上时间戳就行了。Google的Spanner（OSDI 2012 Best Paper）就是这样做的，它利用原子钟和NTP，实现了全球任意两台机器的时间差不超过10ms。当然，《七周七数据库》的英文版出版时，Spanner可能还没有发表。

Riak所采用的向量时钟采用了跟Git类似的技术。每次更新时，就会在这个key的向量时钟（vector clock）的末尾附加上一个伪随机的戳，相当于向量的长度增加了1。当两个不同节点进行同步时，如果双方的向量相同，当然就不用同步了；如果一个是另一个的子集，只需要单向复制（类似Git中的fast-forward update）；如果两个向量互不为子集，那么客户端查询这个key时，会返回两个值。客户端可以自行解决冲突，并把新的value和向量提交上去（类似Git中的merge conflict）。

随着更新次数越来越多，向量会不断增长。因此Riak提供了修剪选项，可以将老的向量时钟（如一天前的）修剪掉。

### MapReduce

本书中介绍的好几个数据库都有MapReduce框架。MapReduce将问题分解为两部分。一、通过map方法，将一列数据转换成另一列不同类型的数据；二、通过reduce方法，将map生成的那列数据转换成一个或多个标量值。这种模式允许系统将任务分成更小的组件任务，然后跨大规模集群并行运行这些任务。

在Riak中，可以用JavaScript语言编写MapReduce任务。还可以把作为map函数的JavaScript代码保存进数据库，作为存储过程，以后GET这个key时就会执行这段存储过程代码。

# HBase

HBase是面向列的数据库，基于Google BigTable的思想，主要用于解决大数据问题。HBase基于Hadoop，Hadoop是一个可扩放（scalable）的计算平台，提供了分布式文件系统和MapReduce。Hbase适用于后台OLTP系统，在单个操作方面比较慢，但擅长扫描巨大的数据集。

### 数据结构

Hbase将数据存放在表中，每条记录是一行，一行可以有若干列族，每个列族是一个hashmap，hashmap的每一项称为cell。这里比关系数据库多了一层：关系数据库的基本特征是每行每列交叉处的“单元格”是原子的，而Hbase中行和列族交叉处的数据是一个hashmap，Hbase会对hashmap中的key进行索引。

这种结构很适合对文本进行倒排索引：行是文章，列族只有一个，hashmap是文本中每个单词作为key，单词的出现次数作为value。Hbase进行索引后，就很容易查到某个单词在哪篇文章中出现的次数最多。

Web中常用的链接信息也很适合这种结构：行是页面，hashmap中每个链接的锚文本（anchor text）作为key，链接目标作为value。

为什么需要多个列族呢？这是为了存储不同类型的信息而避免建立多个表。例如，一篇文章有作者、标题、内容和评论，作者和标题可以放在同一个列族中，但内容和评论如果需要做倒排索引，放在原来的列族中就会造成混乱，因此最好是建立三个列族，基本信息（作者、标题）、内容索引、评论索引。

如果不需要经常取出整行的数据，建议将不同的列族分到不同的表，以便集群更好地分割区域。

Hbase内置支持数据的版本管理，用精确到毫秒的时间戳标识数据的版本，因此很适合wiki之类需要保存历史版本的应用。

### 数据分区

Hbase可以通过Hadoop的Java API进行各种操作，书中使用的是JRuby。

与Riak类似，Hbase为每个数据生成了一个唯一ID（hash值）。Hbase按照hash值的范围分成若干个不重叠的区域，并将每个区域指派给集群中一个特定的区域服务器。特殊的 .META. 表记录了哪个区域服务器负责为每个表的哪个区域服务的信息。

如果一台区域服务器失效，主服务器就会重新分配原来属于失效节点的那些区域到其他区域服务器。如果主服务器失效，编号顺延的区域服务器会接管它的职责。

相比Riak，Hbase的可用性没有那么高。通过合理的参数配置，Riak可以在仅有一个节点存活时仍然正常工作，而Hbase只能在一两个节点当机的灾难中存活下来。

另外值得注意的是，Hbase可以轻松scale-up，但难以scale-down，因此它就像是“射钉枪”，能做大事，但要小心伤到自己的手指。

# MongoDB

关系数据库有强大的查询能力，Riak和Hbase这样的数据库具有分布式的特点，MongoDB在这两者之间找到了最佳结合点。因此，MongoDB是很多新兴Web应用采用的持久存储。

### 数据结构

MongoDB由若干个集合（collection）组成（类似关系数据库的表），每个集合中有若干文档（document）（类似关系数据库中的行），每个文档是一个JSON对象。与关系数据库不同的是，JSON对象没有“模式”的概念，而且值可以嵌套任意深度（即JSON对象中key对应的value可以是一个JSON对象）

每个文档都包含一个 \_id 字段作为这个文档的唯一标识符。这个\_id是自动生成的，由4字节时间戳、3字节机器ID、2字节进程ID、3字节增量计数器构成，保证了 \_id 的唯一性。

关系数据库中需要表间连接（JOIN）的数据结构，在MongoDB中往往可以通过子文档来表达。以文章包含评论为例，在关系模型中需要有一张文章表和一张评论表，再建立外键；在MongoDB中，只需建立一个文章的集合，每篇文章作为一个document，其中一个字段是子文档，包含了这篇文章的所有评论。

这种数据结构显然比关系模型更自然。如果需要打破树形结构，查询某个作者曾经发表的所有评论，则可以为它建立索引，查找起来并不会比关系数据库慢。MongoDB可以使用B树、二维索引或球形地理空间索引。

### 查询

MongoDB的母语是JavaScript，JavaScript原生支持JSON，使得数据的操作显得很自然。

MongoDB最基本的查询方法是find，它可以像SQL一样：

* 指定相等、大于、小于、在集合中等条件
* 支持正则表达式（这是SQL没有的特性）
* 用布尔运算符（与、或、非）连接上述条件
* 过滤结果集中包含哪些字段
* 使用聚合函数（如count，distinct）
* GROUP BY，指定自定义的reduce函数

当然，由于JavaScript语言的限制，查询条件要写成JSON对象的形式，不如SQL直观。

在查询语句后附加 .explain()，可以像MySQL的EXPLAIN语句一样获知是否使用了索引等信息。

MongoDB提供了原子增减值的指令，以及更新并返回旧值的原子指令。

MongoDB同样提供了MapReduce的JavaScript API。

### 复制与分片

MongoDB采用与关系数据库热备类似的方法增加冗余。MongoDB集群会自动“选举”出一个主节点承担所有的读写操作，从节点自动从主节点同步数据。为保证选举正常进行，总是应该有奇数个节点，否则如果集群的网络被隔断成相等数量的两边，则两边都不能选举出主节点，导致服务中断。当只剩最后一个节点时，它会选举自己。

当数据集非常大到一个节点无法处理时，可以按照值的范围进行横向分片（sharding）。需要一个mongos服务器作为“反向代理”，作为接受用户请求的单点，同时维护各mongod节点的数据集划分和分发请求。

就像RAID一样，复制和分片可以同时使用，它们的目的分别是增加冗余和提高性能。例如，可以3台服务器构成一个副本集，2个副本集构成2个分片，再加上配置服务器和mongos“反向代理”服务器，就构成了8台服务器的MongoDB集群。

# CouchDB

CouchDB与MongoDB很相似。它们的两个主要区别是，

1. CouchDB更轻量级，它可以像SQLite一样嵌入在应用程序中。
2. CouchDB的数据带有版本号。

### 事务

MongoDB和CouchDB都不提供“锁”和“事务”的功能。在MongoDB中，往往通过内置的“原子操作”来实现；CouchDB避免冲突的秘诀则是确保只修改最新版本的文档。CouchDB给每个文档分配一个名为 \_rev 的域（格式为顺序递增的版本号+伪随机字符串），在每次文档发生变化时，\_rev 域被更新。在更新（POST）和删除（DELETE）操作时，除了提供 \_id，还要提供 \_rev，如果 \_rev 不匹配，则拒绝操作。拒绝操作可能是由于上次读出此文档与进行更新、删除操作之间，此文档被修改了，使得版本号被更新。

在删除文档时，CouchDB并不就地修改文档，而是用一个新的空白文档“更新”它。

MongoDB中特殊的键用 $ 开头，CouchDB中特殊的键用 \_ 开头。

CouchDB除了用JavaScript API访问，也可以用类似Riak的HTTP REST API。

### 视图

CouchDB中的视图（view）是通过JavaScript函数实现的。通常，这样的函数执行一次”全表扫描“，对每个文档执行map函数，得到map结果的集合。Hbase也是用的类似方法生成视图。这种方法固然灵活，但全表扫描实在太耗时了，因此Hbase在map函数中提供了筛选条件，可以先将符合条件的文档初步筛选出来，再送进map函数；CouchDB没有筛选条件的功能，但它可以缓存map的结果，因此每次查询视图时，只需对改变过的文档执行map函数。

除了基于map的视图，CouchDB还支持reduce。

### 变更推送

在Web应用中，时常需要得知某种特定的值是否出现变化。在关系数据库中，这往往用触发器（trigger）来实现。

CouchDB可以编写JavaScript代码来监听某个集合（collection），一旦这个集合上发生任何写操作，监听代码就会被调用。在监听代码中加入过滤器，就能关注变化的某个子集，例如只监听文档删除，或者只关注符合某种条件的文档。

CouchDB的变更推送与Node.js的事件驱动机制在一起工作锝很好，采用长轮询的方法：向MongoDB发出一个监听请求，有事件发生时会调用回调函数，回调函数从MongoDB接收一块块的数据，直到数据接收完毕，解析收到的数据，遍历文档调用map函数。

### 复制

CouchDB提供冗余的方式与Riak类似，每个节点都可以独立进行读写。出现冲突的症状就是同一个 \_id，两个节点的 \_rev 不同。CouchDB的处理方法非常简单粗暴：在两个不同的 \_rev 间二选一，当然这个选择算法是确定的。由于CouchDB保存了文档的历史版本，如果客户端发现不对头，可以读出历史版本进行恢复。

# Neo4j

Neo4j是”图数据库“，即数学意义上的有向图。Neo4j的重点在于存储数据间的关系。与关系数据库不同，Neo4j不需要事先指定模式和关系，从而非常方便关系的修改和保存一些”特例“关系。

Neo4j推荐的编程语言是Gremlin，它是用Groovy实现的一种领域特定的图遍历语言。它使用数学上通用的”图“术语。假设g是图对象，那么

* g.V 表示全部顶点
* g.E 表示全部边
* g.v(0) 表示0号顶点
* g.v(0).map() 可以列出0号顶点的属性
* g.V.filter(it.name==’riesling’) 可以选出符合给定条件的顶点
* g.v(0).outE 得到0号顶点的出边集合
* g.v(0).outE.inV 可以找到0号顶点所指向的顶点集合
* g.v(0).inE.outV 是指向0号顶点的顶点集合
* g.v(0).inE.outV.next() 得到上述集合的第1个顶点
* alice.bothE(‘friends’).bothV.except([alice]).name 得到Alice所有好友（不论好友的方向）的名字
* alice.bothE(‘friends’).bothV.except([alice]).loop(3){ it.loops<=2 }.dedup.name 得到Alice朋友的朋友的朋友（其中loop(3)表示对前面的三步bothE(‘friends’).bothV.except([alice])进行循环，loops<=2表示循环两次，dedup表示集合去重）

如上可以看到，Gremlin的强大之处是对集合进行”管道“操作，酷似jQuery对DOM元素的操作。

Neo4j在单节点模式下支持ACID事务。在多节点高可用性模式下，由于对一个节点的写入操作不会立即同步到其他节点，事务无法遵循ACID。

# Redis

 Redis最大的特点就是快。它本质上是一个key-value数据库，还提供了集合、队列、栈、发布者-订阅者等高级数据结构。Redis一般被用作缓存。

 Redis采用SET设置key，GET获取key，MSET一次设置多个key，MGET一次获取多个key。Redis的原子数据类型是字符串，但它能识别整数：INCR可以原子地对整数key进行自增。

 类似PostgreSQL和Neo4j，Redis也支持事务。用MULTI进入事务，EXEC提交事务，DISCARD撤销事务，用法跟关系数据库的事务很相似。

 Redis还支持高级数据结构：

### 哈希表

 有点像MongoDB或CouchDB的嵌套文档，也就是一个键所对应的值又是一个key-value存储。当然，不能再嵌套了。

### 列表

 列表包含多个有序值，既可以作为队列，又可以作为栈，因为它的左右两端都可以PUSH和POP。可以使用LRANGE取得列表的任何部分，LREM删除列表中匹配的值。

 列表的强大之处在于阻塞的弹出操作，这可以用来支持消息系统。当队列空时，BRPOP会一直阻塞，直到队列里插入元素。

### 集合

 集合与列表的区别是，集合中存储了一些无序的非重复值。Redis支持在集合之间进行交集、并集、差集等操作。

### 有序集合

 哈希表、列表、集合在高级编程语言中都是内置的了。有序集合则不然。例如，我们需要记录每个帖子的访问量，并把访问量最大的排在前面。用关系数据库当然能做到，但关系数据库修改操作的性能往往不敢恭维。自己写个二叉排序树或者堆当然也行，不过没有数据持久化。

 使用Redis，只要用ZADD命令，提供两个参数得分和成员，就能实现这样的功能。还能通过ZRANGE命令返回已排序序列的某个范围。

有序集合更强大的地方是，可以创建类似”视图“的东西，包含若干个有序集合的并集或交集，每个有序集合可以指定权重（用于乘以得分），将加权得分的最小值、最大值或者总和作为“视图”中这一项的得分。这有助于实现简单排序系统，如“热门帖子”可以综合考虑访问量、回帖数、赞的次数等。

### 发布-订阅系统

Linux原生不提供发布-订阅系统，而这种任务模式又很常见，因此现在的做法要么是实现个socket服务器，要么是使用zeroMQ之类的消息库。Redis也提供了优雅的解决方案：SUBSCRIBE订阅消息，PUBLISH发布消息。每条发布的消息都会被推送到正在接收消息的订阅者。

发布-订阅系统与前面说的阻塞队列弹出操作的区别是，如果同时有多个订阅者，发布-订阅系统会把消息发给每个订阅者，阻塞队列会把消息发给任意一个“订阅者”。

### 持久化

Redis之所以快，是因为它的所有操作都是在内存中进行的。Redis提供两种方式进行持久化：只追加的操作日志，以及定期将内存变更刷新到磁盘。

由于Redis常被用作缓存服务器，还可以给已有的key指定TTL（生存期），一旦到期，Redis就会自动清除这个key。

# 结语

前面介绍的6种非关系数据库，经过合理的组合，可以得到高可用、可扩放（scalable）、高性能的存储系统。

* MongoDB / CouchDB作为一般数据的持久存储；
* Redis作为MongoDB / CouchDB 的缓存和Session数据的临时存储；
* Neo4j 存储复杂的实体间关系；
* Hbase 用于OLTP之类的大数据分析；
* Riak 用于需要高可用性的数据。

最后，我们列表比较书中所述7种数据库的优缺点：

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| MongoDB | 复杂查询 | 嵌入能力 |
| CouchDB | 易于嵌入 | 复杂查询 |
| Riak | 高可用性 | 复杂查询 |
| Redis | 快 | 复杂数据结构 |
| PostgreSQL | 严密的关系模型 | 分布式可用性 |
| Neo4j | 灵活地表示关系 | 大数据 |
| Hbase | 大数据，与Hadoop集成 | 快速查询 |

读完本书，我们能够了解当今数据库的发展趋势。在Web大潮下，为了提高分布式可用性，除了关系模型之外，涌现出了这么多种各具特色的数据库。希望在今后的开发中，不再受到一种或两种数据库范式的束缚，而是根据项目的实际需求，选用合适的数据库组合。