五月的仓颉

Sql Or NoSql，看完这一篇你就懂了

前言

你是否在为系统的数据库来一波大流量就几乎打满CPU，日常CPU居高不下烦恼？你是否在各种NoSql间纠结不定，到底该选用那种最好？今天的你就是昨天的我，这也是写这篇文章的初衷。

这篇文章是我好几个月来一直想写的一篇文章，也是一直想学习的一个内容，作为互联网从业人员，我们要知道关系型数据库（MySql、Oracle）无法满足我们对存储的所有要求，因此对底层存储的选型，对每种存储引擎的理解非常重要。同时也由于过去一段时间的工作经历，对这块有了一些更多的思考，想通过自己的总结把这块写出来分享给大家。

结构化数据、非结构化数据与半结构化数据

文章的开始，聊一下结构化数据、非结构化数据与半结构化数据，因为数据特点的不同，将在技术上直接影响存储引擎的选型。

首先是结构化数据，根据定义**结构化数据指的是由二维表结构来逻辑表达和实现的数据，严格遵循数据格式与长度规范，也称作为行数据**，特点为：数据以行为单位，一行数据表示一个实体的信息，每一行数据的属性是相同的。例如：

id	Name	age	phone	address
1	张三	18	12345	浙江
2	李四	19	23456	上海
3	王五	20	34567	北京

因此关系型数据库完美契合结构化数据的特点，关系型数据库也是关系型数据最主要的存储与管理引擎。

非结构化数据，指的是**数据结构不规则或不完整，没有任何预定义的数据模型，不方便用二维逻辑表来表现的数据**，例如办公文档（Word）、文本、图片、HTML、各类报表、视频音频等。

介于结构化与非结构化数据之间的数据就是半结构化数据了，它是结构化数据的一种形式，虽然**不符合二维逻辑这种数据模型结构，但是包含相关标记，用来分割语义元素以及对记录和字段进行分层**。常见的半结构化数据有XML和JSON，例如：

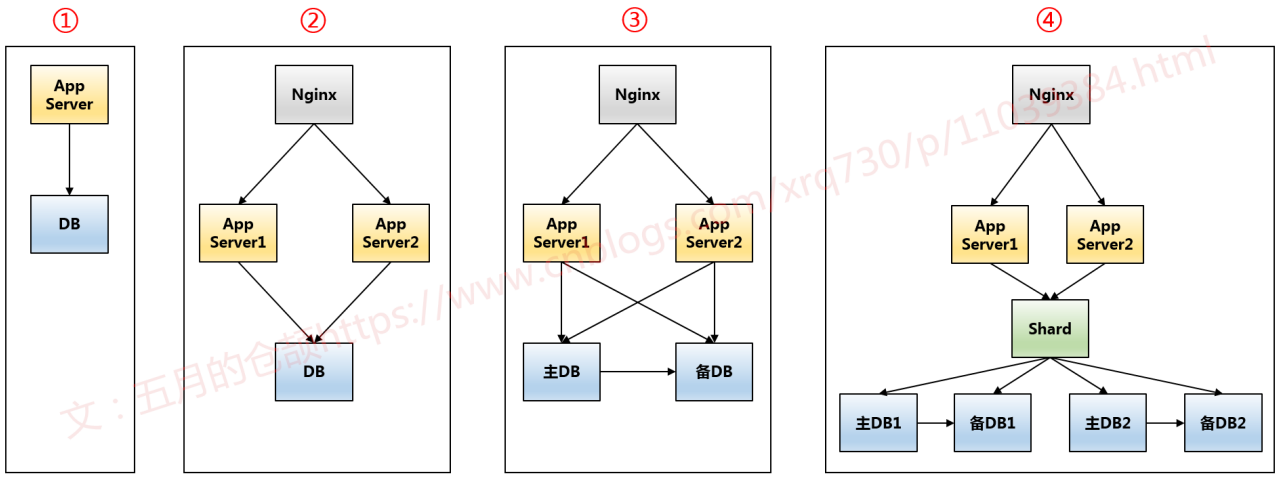
```
<person>
    <name>张三</name>
    <age>18</age>
```

```
    <phone>12345</phone>
</person>
```

这种结构也被成为自描述的结构。

以关系型数据库的方式做存储的架构演进

首先，我们看一下使用关系型数据库的方式，企业一个系统发展的几个阶段的架构演进（由于本文写的是Sql与NoSql，因此只以存储方式作为切入点，不会涉及类似MQ、ZK这些中间件内容）：



阶段一：企业刚发展的阶段，最简单，一个应用服务器配一个关系型数据库，每次读写数据库。

阶段二：无论是使用MySQL还是Oracle还是别的关系型数据库，数据库通常不会先成为性能瓶颈，通常随着企业规模的扩大，一台应用服务器扛不住上游过来的流量且一台应用服务器会产生单点故障的问题，因此**加应用服务器并且在流量入口使用Nginx做一层负载均衡**，保证把流量均匀打到应用服务器上。

阶段三：随着企业规模的继续扩大，此时由于读写都在同一个数据库上，数据库性能出现一定的瓶颈，此时简单地做一层**读写分离**，每次写主库，读备库，主备库之间通过binlog同步数据，就能很大程度上解决这个阶段的数据库性能问题

阶段四：企业发展越来越好了，业务越来越大了，做了读写分离数据库压力还是越来越大，这时候怎么办呢，一台数据库扛不住，那我们就分几台吧，做**分库分表**，对表做垂直拆分，对库做水平拆分。以扩数据库为例，扩出两台数据库，以一定的单号（例如交易单号），以一定的规则（例如取模），交易单号对2取模为0的丢到数据库1去，交易单号对2取模为1的丢到数据库2去，通过这样的方式将写数据库的流量均分到两台数据库上。一般分库分表会使用Shard的方式，通过一个中间件，便于连接管理、数据监控且客户端无需感知数据库ip

关系型数据库的优点

上面的方式，看似可以解决问题（实际上确实也能解决很多问题），正常对关系型数据库做一下读写分离 + 分库分表，支撑个1W+的读写QPS还是问题不大的。但是受限于关系型数据库本身，这套架构方案依然有着明显的不足，下面对利用关系型数据库方式做存储的方案的优点先进行一下分析，后一部分再分析一下缺点，对某个技术的优缺点的充分理解是技术选型的前提。

- **易理解**

因为行 + 列的二维表逻辑是非常贴近逻辑世界的一个概念，关系模型相对网状、层次等其他模型更加容易被理解

- **操作方便**

通用的SQL语言使得操作关系型数据库非常方便，支持join等复杂查询，Sql + 二维关系是关系型数据库最无可比拟的优点，这种易用性非常贴近开发者

- **数据一致性**

支持ACID特性，可以维护数据之间的一致性，这是使用数据库非常重要的一个理由之一，例如同银行转账，张三转给李四100元钱，张三扣100元，李四加100元，而且必须同时成功或者同时失败，否则就会造成用户的资损

- **数据稳定**

数据持久化到磁盘，没有丢失数据风险，支持海量数据存储

- **服务稳定**

最常用的关系型数据库产品MySql、Oracle服务器性能卓越，服务稳定，通常很少出现宕机异常

关系型数据库的缺点

紧接着的，我们看一下关系型数据库的缺点，也是比较明显的。

- **高并发下IO压力大**

数据按行存储，即使只针对其中某一列进行运算，也会将整行数据从存储设备中读入内存，导致IO较高

- **为维护索引付出的代价大**

为了提供丰富的查询能力，通常热点表都会有多个二级索引，一旦有了二级索引，数据的新增必然伴随着所有二级索引的新增，数据的更新也必然伴随着所有二级索引的更新，这不可避免地降低了关系型数据库的读写能力，且索引越多读写能力越差。有机会的话可以看一下自己公司的数据库，除了数据文件不可避免地占空间外，索引占的空间其实也并不少

- **为维护数据一致性付出的代价大**

数据一致性是关系型数据库的核心，但是同样为了维护数据一致性的代价也是非常大的。我们都知道SQL标准为事务定义了不同的隔离级别，从低到高依次是读未提交、读已提交、可重复度、串行化，事务隔离级别越低，可能出现的并发异常越多，但是通常而言能提供的并发能力越强。那么为了保证事务一致性，数据库就需要提供并发控制与故障恢复两种技术，前者用于减少并发异常，后者可以在系统异常的时候保证事务与数据库状态不会被破坏。对于并发控制，其核心思想就是加锁，无论是乐观锁还是悲观锁，只要提供的隔离级别越高，那么读写性能必然越差

- **水平扩展后带来的种种问题难处理**

前文提过，随着企业规模扩大，一种方式是对数据库做分库，做了分库之后，数据迁移（1个库的数据按照一定规则打到2个库中）、跨库join（订单数据里有用户数据，两条数据不在同一个库中）、分布式事务处理都是需要考虑的问题，尤其是分布式事务处理，业界当前都没有特别好的解决方案

- **表结构扩展不方便**

由于数据库存储的是结构化数据，因此表结构schema是固定的，扩展不方便，如果需要修改表结构，需要执行DDL（data definition language）语句修改，修改期间会导致锁表，部分服务不可用

- **全文搜索功能弱**

例如like "%中国真伟大%"，只能搜索到"2019年中国真伟大，爱祖国"，无法搜索到"中国真是太伟大了"这样的文本，即不具备分词能力，且like查询在"%中国真伟大"这样的搜索条件下，无法命中索引，将会导致查询效率大大降低

写了这么多，我的理解核心还是前三点，它反映出的一个问题是**关系型数据库在高并发下的能力是有瓶颈的**，尤其是写入/更新频繁的情况下，出现瓶颈的结果就是数据库CPU高、Sql执行慢、客户端报数据库连接池不够等错误，因此例如万人秒杀这种场景，我们绝对不可能通过数据库直接去扣减库存。

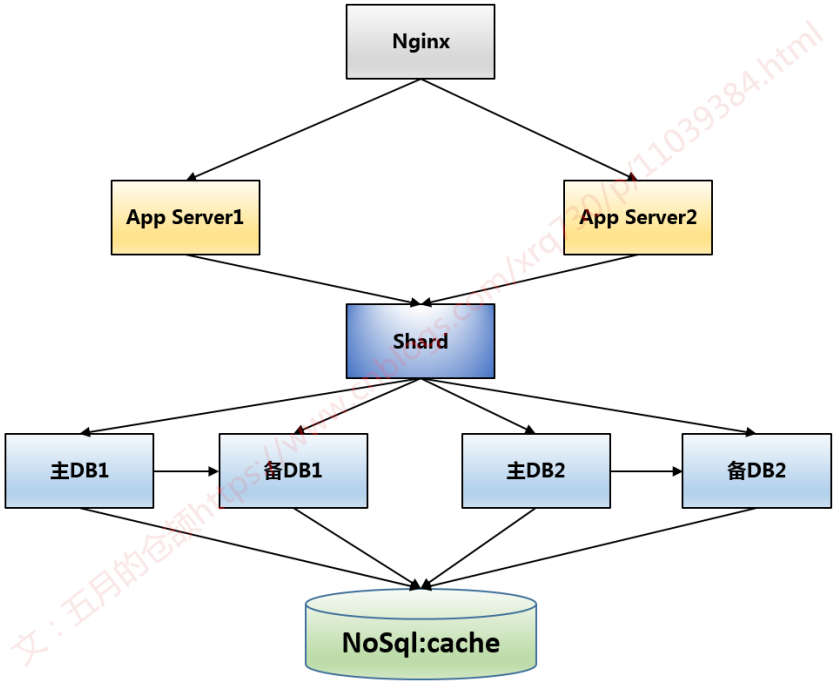
可能有朋友说，数据库在高并发下的能力有瓶颈，我公司有钱，加CPU、换固态硬盘、继续买服务器加数据库做分库不就好了，问题是这是一种性价比非常低的方式，花1000万达到的效果，换其他方式可能100万就达到了，不考虑人员、服务器投入产出比的Leader就是个不合格的Leader，且关系型数据库的方式，受限于它本身的特点，可能花了钱都未必能达到想要的效果。至于什么是花100万就能达到花1000万效果的方式呢？可以继续往下看，这就是我们要说的NoSql。

结合NoSql的方式做存储的架构演进

像上文分析的，数据库作为一种关系型数据的存储引擎，存储的是关系型数据，它有优点，同时也有明显的缺点，因此通常在企业规模不断扩大的情况下，不会一味指望通过增强数据库的能力来解决数据存储问题，而是会引入其他存储，也就是我们说的NoSql。

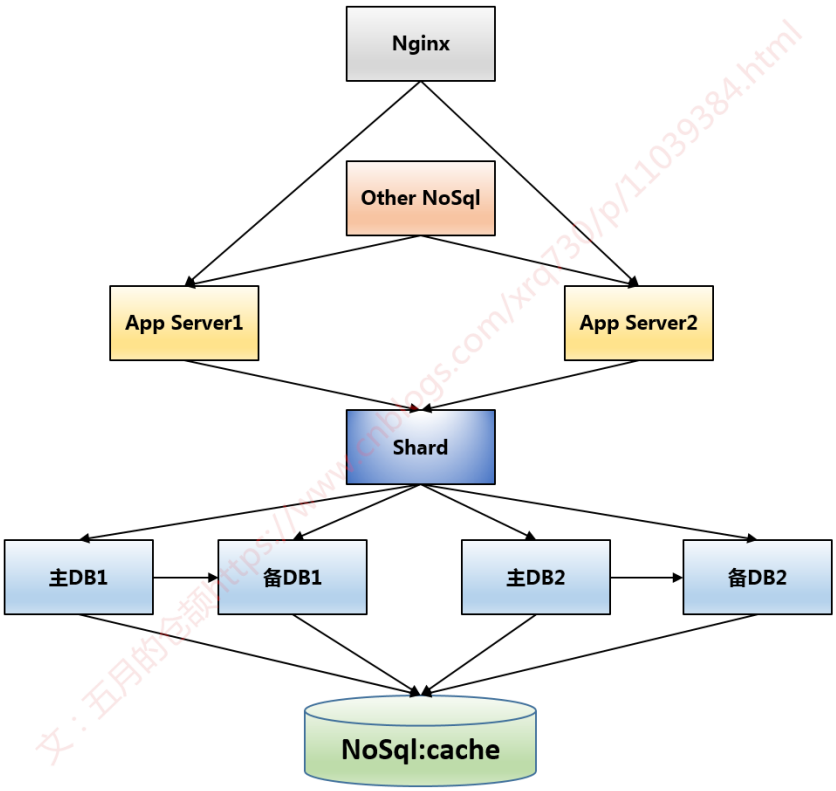
NoSql的全称为Not Only SQL，泛指非关系型数据库，是对关系型数据库的一种**补充**，特别注意补充这两个字，这意味着NoSql与关系型数据库并不是对立关系，二者各有优劣，取长补短，在合适的场景下选择合适的存储引擎才是正确的做法。

比较简单的NoSql就是缓存：



针对那些读远多于写的数据，引入一层缓存，每次读从缓存中读取，缓存中读取不到，再去数据库中取，取完之后再写入到缓存，对数据做好失效机制通常就没有大问题了。通常来说，缓存是性能优化的第一选择也是见效最明显的方案。

但是，缓存通常都是KV型存储且容量有限（基于内存），无法解决所有问题，于是再进一步的优化，我们继续引入其他NoSql：



数据库、缓存与其他NoSql并行工作，充分发挥每种NoSql的特点。当然NoSql在性能方面大大优于关系挺数据库的同时，往往也伴随着一些特性的缺失，比较常见的就是事务功能的缺失。

下面看一下常用的NoSql及他们的代表产品，并对每种NoSql的优缺点和适用场景做一下分析，便于熟悉每种NoSql的特点，方便技术选型。

KV型NoSql（代表----Redis）

KV型NoSql顾名思义就是以键值对形式存储的非关系型数据库，是最简单、最容易理解也是大家最熟悉的一种NoSql，因此比较快地带过。Redis、MemCache是其中的代表，Redis又是KV型NoSql中应用最广泛的NoSql，KV型数据库以Redis为例，最大的优点我总结下来就两点：

- 数据基于内存，读写效率高
- KV型数据，时间复杂度为O(1)，查询速度快

因此，KV型NoSql最大的优点就是**高性能**，利用Redis自带的BenchMark做基准测试，TPS可达到10万的级别，性能非常强劲。同样的Redis也有所有KV型NoSql都有的比较明显的缺点：

- 只能根据K查V，无法根据V查K
- 查询方式单一，只有KV的方式，不支持条件查询，多条件查询唯一的做法就是数据冗余，但这会极大的浪费存储空间
- 内存是有限的，无法支持海量数据存储
- 同样的，由于KV型NoSql的存储是基于内存的，会有丢失数据的风险

综上所述，KV型NoSql最合适的场景就是**缓存**的场景：

- 读远多于写
- 读取能力强
- 没有持久化的需求，可以容忍数据丢失，反正丢了再查询一把写入就是了

例如根据用户id查询用户信息，每次根据用户id去缓存中查询一把，查到数据直接返回，查不到去关系型数据库里面根据id查询一把数据写到缓存中去。

搜索型NoSql（代表----ElasticSearch）

传统关系型数据库主要通过索引来达到快速查询的目的，但是在全文搜索的场景下，索引是无能为力的，like查询一来无法满足所有模糊匹配需求，二来使用限制太大且使用不当容易造成慢查询，**搜索型NoSql的诞生正是为了解决关系型数据库全文搜索能力较弱的问题**，ElasticSearch是搜索型NoSql的代表产品。

全文搜索的原理是**倒排索引**，我们看一下什么是倒排索引。要说倒排索引我们先看下什么是正排索引，传统的正排索引是**文档-->关键字**的映射，例如"Tom is my friend"这句话，会将其切分为"Tom"、"is"、"my"、"friend"四个单词，在搜索的时候对文档进行扫描，符合条件的查出来。这种方式原理非常简单，但是由于其检索效率太低，基本没什么实用价值。

倒排索引则完全相反，它是**关键字-->文档**的映射，我用张表格展示一下就比较清楚了：

	Tom is Tom	Tom is my friend	Thank you , Betty	Tom is Betty's husband
Tom	2	1	0	1
is	1	1	0	1
my	0	1	0	0
friend	0	1	0	0
Thank	0	0	1	0
you	0	0	1	0
Betty	0	0	1	1
husband	0	0	0	1

意思是我现在这里有四个短句：

- "Tom is Tom"
- "Tom is my friend"
- "Thank you, Betty"
- "Tom is Betty's husband"

搜索引擎会根据一定的切分规则将这句话切成N个关键字，并以关键字的维度维护关键字在每个文本中的出现次数。这样下次搜索"Tom"的时候，由于Tom这个词语在"Tom is Tom"、"Tom is my friend"、"Tom is Betty's husband"三句话中都有出现，因此这三条记录都会被检索出来，且由于"Tom is Tom"这句话中"Tom"出现了2次，因此这条记录对"Tom"这个单词的匹配度最高，最先展示。这就是搜索引擎倒排索引的基本原理，假设某个关键字在某个文档中出现，那么倒排索引中有两部分内容：

- 文档ID
- 在该文档中出现的位置情况

可以举一反三，我们搜索"Betty Tom"这两个词语也是一样，搜索引擎将"Betty Tom"切分为"Tom"、"Betty"两个单词，根据开发者指定的满足率，比如满足率=50%，那么只要记录中出现了两个单词之一的记录都会被检索出来，再按照匹配度进行展示。

搜索型NoSql以ElasticSearch为例，它的优点为：

- 支持分词场景、全文搜索，这是区别于关系型数据库最大特点
- 支持条件查询，支持聚合操作，类似关系型数据库的Group By，但是功能更加强大，适合做数据分析
- 数据写文件无丢失风险，在集群环境下可以方便横向扩展，可承载PB级别的数据
- 高可用，自动发现新的或者失败的节点，重组和重新平衡数据，确保数据是安全和可访问的

同样，ElasticSearch也有比较明显的缺点：

- 性能全靠内存来顶，也是使用的时候最需要注意的点，非常吃硬件资源、吃内存，大数据量下64G + SSD基本是标配，算得上是数据库中的爱马仕了。为什么要专门提一下内存呢，因为内存这个东

西是很值钱的，相同的配置多一倍内存，一个月差不多就要多花几百块钱，至于ElasticSearch内存用在什么地方，大概有如下这些：

  - Indexing Buffer----ElasticSearch基于Luence，Lucene的倒排索引是先在内存里生成，然后定期以Segment File的方式刷磁盘的，每个Segment File实际就是一个完整的倒排索引
  - Segment Memory----倒排索引前面说过是基于关键字的，Lucene在4.0后会将所有关键字以FST这种数据结构的方式将所有关键字在启动的时候全量加载到内存，加快查询速度，官方建议至少留系统一半内存给Lucene
  - 各类缓存----Filter Cache、Field Cache、Indexing Cache等，用于提升查询分析性能，例如Filter Cache用于缓存使用过的Filter的结果集
  - Cluter State Buffer----ElasticSearch被设计为每个Node都可以响应用户请求，因此每个Node的内存中都包含有一份集群状态的拷贝，一个规模很大的集群这个状态信息可能会非常大

- 读写之间有延迟，写入的数据差不多1s样子会被读取到，这也正常，写入的时候自动加入这么多索引肯定影响性能
- 数据结构灵活性不高，ElasticSearch这个东西，字段一旦建立就没法修改类型了，假如建立的数据表某个字段没有加全文索引，想加上，那么只能把整个表删了再重建

因此，搜索型NoSql最适用的场景就是**有条件搜索尤其是全文搜索的场景**，作为关系型数据库的一种替代方案。

另外，搜索型数据库还有一种特别重要的应用场景。我们可以想，一旦对数据库做了分库分表后，原来可以在单表中做的聚合操作、统计操作是否统统失效？例如我把订单表分16个库，1024张表，那么订单数据就散落在1024张表中，我想要统计昨天浙江省单笔成交金额最高的订单是哪笔如何做？我想要把昨天的所有订单按照时间排序分页展示如何做？**这就是搜索型NoSql的另一大作用了，我们可以把分表之后的数据统一打在搜索型NoSql中，利用搜索型NoSql的搜索与聚合能力完成对全量数据的查询**。

至于为什么把它放在KV型NoSql后面作为第二个写呢，因为通常搜索型NoSql也会作为一层前置缓存，来对关系型数据库进行保护。

列式NoSql（代表----HBase）

列式NoSql，大数据时代最具代表性的技术之一了，以HBase为代表。

列式NoSql是基于列式存储的，那么什么是列式存储呢，列式NoSql和关系型数据库一样都有主键的概念，区别在于关系型数据库是按照行组织的数据：

id	name	phone
1	张三	12345
2	李四	

看到每行有name、phone、address三个字段，这是行式存储的方式，且可以观察id = 2的这条数据，即使phone字段没有，它也是占空间的。

列式存储完全是另一种方式，它是按每一列进行组织的数据：

id	name
1	张三
2	李四

id	phone
1	12345

这么做有什么好处呢？大致有以下几点：

- 查询时只有指定的列会被读取，不会读取所有列
- 存储上节约空间，Null值不会被存储，一列中有时候会有很多重复数据（尤其是枚举数据，性别、状态等），这类数据可压缩，行式数据库压缩率通常在3:1~5:1之间，列式数据库的压缩率一般在8:1~30:1左右
- 列数据被组织到一起，一次磁盘IO可以将一列数据一次性读取到内存中

第二点说到了数据压缩，什么意思呢，以比较常见的字典表压缩方式举例：

通过字典表压缩数据

自己看图理解一下，应该就懂了。

接着继续讲讲优缺点，列式NoSql，以HBase为代表的，优点为：

- 海量数据无限存储，PB级别数据随便存，底层基于HDFS（Hadoop文件系统），数据持久化

- 读写性能好，只要没有滥用造成数据热点，读写基本随便玩
- 横向扩展在关系型数据库及非关系型数据库中都是最方便的之一，只需要添加新机器就可以实现数据容量的线性增长，且可用在廉价服务器上，节省成本
- 本身没有单点故障，可用性高
- 可存储结构化或者半结构化的数据
- 列数理论上无限，HBase本身只对列族数量有要求，建议1~3个

说了这么多HBase的优点，又到了说HBase缺点的时候了：

- HBase是Hadoop生态的一部分，因此它本身是一款比较重的产品，依赖很多Hadoop组件，数据规模不大没必要用，运维还是有点复杂的
- KV式，不支持条件查询，或者说条件查询非常非常弱吧，HBase在Scan扫描一批数据的情况下还是提供了前缀匹配这种API的，条件查询除非定义多个RowKey做数据冗余
- 不支持分页查询，因为统计不了数据总数

因此**HBase比较适用于那种KV型的且未来无法预估数据增长量的场景**，另外HBase使用还是需要一定的经验，主要体现在RowKey的设计上。

文档型NoSql（代表----MongoDB）

坦白讲，根据我的工作经历，文档型NoSql我只有比较浅的使用经验，因此这部分只能结合之前的使用与网上的文章大致给大家介绍一下。

什么是文档型NoSql呢，文档型NoSql指的是将半结构化数据存储为文档的一种NoSql，文档型NoSql通常以JSON或者XML格式存储数据，因此文档型NoSql是没有Schema的，由于没有Schema的特性，我们可以随意地存储与读取数据，因此文档型NoSql的出现是**解决关系型数据库表结构扩展不方便的问题的**。

MongoDB是文档型NoSql的代表产品，同时也是所有NoSql产品中的明星产品之一，因此这里以MongoDB为例。按我的理解，作为文档型NoSql，MongoDB是一款完全和关系型数据库对标的产品，就我们从存储上来看：

id	name	age	phone	address
1	张三	18	12345	浙江
2	李四	19	23456	上海

```
{"address":"浙江","phone":"12345","name":"张三","id":1,"age":18}
{"address":"上海","phone":"23456","name":"李四","id":2,"age":19}
```

看到，关系型数据库是按部就班地每个字段一列存，在MongDB里面就是一个JSON字符串存储。关系型数据可以为name、phone建立索引，MongoDB使用createIndex命令一样可以为列建立索

引，建立索引之后可以大大提升查询效率。其他方面而言，就大的基本概念，二者之间基本也是类似的：

关系型数据库	Mongdb
表	集合
行	文档
列	字段
join	嵌入文档或链接

因此，对于MongDB，我们只要理解成一个Free-Schema的关系型数据库就完事了，它的优缺点比较一目了然，优点：

- 没有预定义的字段，扩展字段容易
- 相较于关系型数据库，读写性能优越，命中二级索引的查询不会比关系型数据库慢，对于非索引字段的查询则是全面胜出

缺点在于：

- 不支持事务操作，虽然Mongodb4.0之后宣称支持事务，但是效果待观测
- 多表之间的关联查询不支持（虽然有嵌入文档的方式），join查询还是需要多次操作
- 空间占用较大，这个是MongDB的设计问题，空间预分配机制 + 删除数据后空间不释放，只有用db.repairDatabase()去修复才能释放
- 目前没发现MongoDB有关系型数据库例如MySql的Navicat这种成熟的运维工具

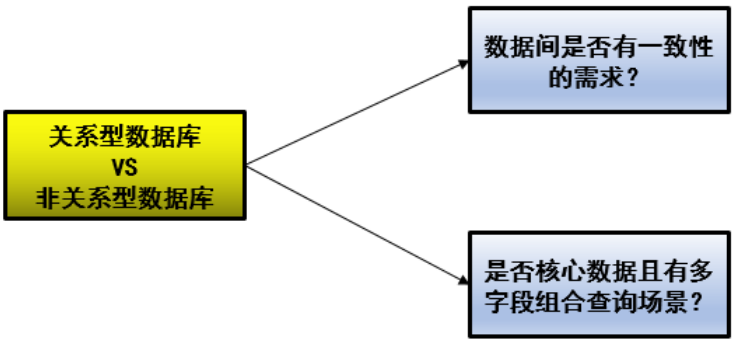
总而言之，MongDB的使用场景很大程度上可以对标关系型数据库，但是比较适合处理那些没有join、没有强一致性要求且表Schema会常变化的数据。

总结：数据库与NoSql及各种NoSql间的对比

最后一部分，做一个总结，本文归根到底是两个话题：

- 何时选用关系型数据库，何时选用非关系型数据库
- 选用非关系型数据库，使用哪种非关系型数据库

首先是第一个话题，关系型数据库与非关系型数据库的选择，在我理解里面无非就是两点考虑：



第一点，不多解释应该都理解，非关系型数据库都是通过牺牲了ACID特性来获取更高的性能的，假设两张表之间有比较强的一致性需求，那么这类数据是不适合放在非关系型数据库中的。

第二点，核心数据不走非关系型数据库，例如用户表、订单表，但是这有一个前提，就是这一类核心数据会有多种查询模式，例如用户表有ABCD四个字段，可能根据AB查，可能根据AC查，可能根据D查，假设核心数据，但是就是个KV形式，比如用户的聊天记录，那么HBase一存就完事了。

这几年的工作经验来看，非核心数据尤其是日志、流水一类中间数据千万不要写在关系型数据库中，这一类数据通常有两个特点：

- 写远高于读
- 写入量巨大

一旦使用关系型数据库作为存储引擎，将大大降低关系型数据库的能力，正常读写QPS不高的核心服务会受这一类数据读写的拖累。

接着是第二个问题，如果我们使用非关系型数据库作为存储引擎，那么如何选型？其实上面的文章基本都写了，这里只是做一个总结（所有的缺点都不会体现事务这个点，因为这是所有NoSql相比关系型数据库共有的一个问题）：

	代表产品	解决关系型数据库的什么问题	优点	缺点
KV型NoSql	Redis	（1）热点KV型数据读写QPS高，读性能受高QPS写数据的直接影响	（1）KV方式读写性能非常强劲 （2）原子命令的特点可用于实现分布式锁	（1）数据无法持久化 （2）不支持条件查询 （3）基于内存，存储空间有限
搜索型NoSql	ElasticSearch	（1）条件查询性能高度依赖索引、不支持全文搜索 （2）分库分表后无法对表做全量查询与聚合	（1）支持丰富的条件查询，支持全文搜索 （2）支持Group By，有强大的数据分析能力 （3）支持海量数据存储	（1）硬件成本昂贵 （2）读取相较于写入有一定延时 （3）修改字段类型需重建表
列式NoSql	HBase	（1）海量数据存储后的读写性能低	（1）支持海量数据存储，特别适合数据增长不可预估的场景 （2）KV方式读写，性能强劲 （3）水平扩容方便 （4）可部署于廉价服务器，成本低	（1）基本没有条件查询能力 （2）不支持分页查询 （3）运维复杂 （4）比较依赖开发者水平，RowKey设计能力
文档型NoSql	MongoDB	（1）表Schema扩展不方便	（1）没有预定义的Schema，可随意变更字段 （2）命中二级索引情况下性能不比关系型数据库差，非索引字段查询对比关系型数据库则是全面胜出	（1）没有连表能力 （2）运维工具还不成熟 （3）空间占用大

但是这里特别说明，**选型一定要结合实际情况而不是照本宣科**，比如：

- 企业发展之初，明明一个关系型数据库就能搞定且支撑一年的架构，搞一套大而全的技术方案出来
- 有一些数据条件查询多，更适合使用ElasticSearch做存储降低关系型数据库压力，但是公司成本有限，这种情况下这类数据可以尝试继续使用关系型数据库做存储
- 有一类数据格式简单，就是个KV类型且增长量大，但是公司没有HBase这方面的人才，运维上可能会有一定难度，出于实际情况考虑，可先用关系型数据库顶一阵子

所以，如果不考虑实际情况，虽然合适有些存储引擎更加合适，但是强行使用反而适得其反，总而言之，适合自己的才是最好的。

==========

我不能保证写的每个地方都是对的，但是至少能保证不复制、不黏贴，保证每一句话、每一行代码都经过了认真的推敲、仔细的斟酌。每一篇文章的背后，希望都能看到自己对于技术、对于生活的态度。

我相信乔布斯说的，只有那些疯狂到认为自己可以改变世界的人才能真正地改变世界。面对压力，我可以挑灯夜战、不眠不休；面对困难，我愿意迎难而上、永不退缩。

其实我想说的是，我只是一个程序员，这就是我现在纯粹人生的全部。

==========

分类: 架构

五月的仓颉
关注 - 0
粉丝 - 4976
+加关注

« 上一篇： 最近学习了HBase
» 下一篇： Java设计模式14：建造者模式

posted @ 2019-08-12 23:29 五月的仓颉 阅读(4937) 评论(17) 编辑 收藏

评论列表

#1楼 2019-08-13 09:31 何甜甜在吗

NoSQL存储架构中作为缓存的redis为什么是在最底层啊，不应该架在db上面吗，先去查询nosql，在去db查询？mongo的可视化工具robo 3T我觉得还不错使用起来

支持(0) 反对(0)

#2楼 [楼主] 2019-08-13 09:46 五月的仓颉

@ 甜甜咿呀咿呀哟

引用

> NoSQL存储架构中作为缓存的redis为什么是在最底层啊，不应该架在db上面吗，先去查询nosql，在去db查询？mongo的可视化工具robo 3T我觉得还不错使用起来

哦那个地方是美观原因，画在DB上面感觉不好看所以就放下面了，这么画只是为了表达会把DB的数据放在缓存中的意思

至于MongoDB的可视化工具确实不熟悉，之前都是使用命令行进行操作的，我把有的一些可视化工具试用一下再更新文章内容

支持(0) 反对(0)

#3楼 2019-08-13 09:54 何甜甜在吗

@ 五月的仓颉
公众号是还不能开启评论吗，专门跑到博客园提出自己的疑问【逃】

支持(1) 反对(0)

#4楼 [楼主] 2019-08-13 10:08 五月的仓颉

@ 甜甜咿呀咿呀哟

> 引用
> @五月的仓颉
> 公众号是还不能开启评论吗，专门跑到博客园提出自己的疑问【逃】

是啊，还没法开启评论。。。确实不方便的，麻烦你了^_^

支持(0) 反对(0)

#5楼 2019-08-13 10:21 luzemin

学习了

支持(0) 反对(0)

#6楼 2019-08-14 14:46 | 核桃牛奶

文章写得很好，读下来有几个问题请教下
『这就是文档型NoSql的另一大作用了，我们可以把分表之后的数据统一打在文档型NoSql中，利用文档型NoSql的搜索与聚合能力完成对全量数据的查询。』
这里不知道是不是笔误应该是搜索型NoSql?

『对于并发控制，其核心思想就是加锁，无论是乐观锁还是悲观锁，只要提供的隔离级别越高，那么读写性能必然越差』
这里私以为关系型db针对并发控制的核心思想并非加锁而是MVCC机制，所以觉得这里这样写有些欠妥

支持(0) 反对(0)

#7楼 2019-08-14 14:57 | 核桃牛奶

还有最后针对KV型NoSql的总结
『原子命令的特点可用于实现分布式锁』
不知道这句话的具体所指是什么？是说setnx指令么？我认为原子命令并非实现分布式锁的必要条件，所以这里有些存疑
『数据无法持久化』
这个的话Redis已经有很成熟的支持了，RDB和AOF都可以

支持(0) 反对(0)

#8楼 [楼主] 2019-08-14 15:13 五月的仓颉

@ | 核桃牛奶
引用
还有最后针对KV型NoSql的总结
『原子命令的特点可用于实现分布式锁』
不知道这句话的具体所指是什么？是说setnx指令么？我认为原子命令并非实现分布式锁的必要条件，所以这里有些存疑
『数据无法持久化』
这个的话Redis已经有很成熟的支持了，RDB和AOF都可以

（1）是笔误，原文已修改
（2）MVCC是MySql InnoDB针对并发控制的一种高性能实现方式，如果以MVCC去说整个关系型数据库的并发控制核心思想我觉得不合适，且MVCC机制中也有一个加锁的过程
（3）MemCache的cas、Redis的setnx都是原子命令，基于命令原子性的特点去实现简单的分布式锁应该没什么问题
（4）持久化不能这么理解，AOF和RDB都只是数据备份的一种方式并不是数据存储的方式，Redis的读写数据本身都是基于内存的不像HBase、ES都是基于文件系统的，且AOF与RDB为了性能的提升是可以关闭的，所以如果数据有持久化的需求指望Redis的AOF与RDB不靠谱

支持(0) 反对(0)

#9楼 2019-08-19 11:55 春娇

插句题外话 博主也是五迷啊hhh

支持(0) 反对(0)

#10楼 [楼主] 2019-08-20 10:37 五月的仓颉

@ 春娇
引用
插句题外话 博主也是五迷啊hhh

五迷自然懂^_^

支持(0) 反对(0)

#11楼 2019-08-22 11:46 春娇

@ 五月的仓颉
10月底上海有演唱会，记得来看呀

支持(0) 反对(0)

#12楼 2019-09-11 21:16 shyoldman

实习期间曾在ATA拜读过这篇文章，特来膜拜

支持(0) 反对(0)

#13楼 [楼主] 2019-09-12 15:01 五月的仓颉

@ shyoldman
引用
实习期间曾在ATA拜读过这篇文章，特来膜拜

希望明年可以在阿里巴巴这个舞台见到你

支持(0) 反对(0)

#14楼 2019-09-29 17:41 悲惨世界

文中提到“(mongodb)对于非索引字段的查询则是全面胜出”, 请问这是基于什么原理？mongo对于非索引字段的查询在处理上相对于mysql有什么优化?

支持(0) 反对(0)

#15楼 [楼主] 2019-09-29 17:54 五月的仓颉

@ 悲惨世界
引用
文中提到“(mongodb)对于非索引字段的查询则是全面胜出”, 请问这是基于什么原理？mongo对于非索引字段的查询在处理上相对于mysql有什么优化?

Mysql索引使用B+树，数据都在叶节点上，每次查询都需要访问到叶节点
MongoDB索引使用B-树，所有节点都有Data域，只要找到指定索引就可以进行访问

无疑单次查询平均快于Mysql

我没有对这块研究很深，但是大致来说是这样的

支持(0) 反对(0)

#16楼 2019-09-29 17:57 悲惨世界

原文不是“非索引字段”么？

支持(0) 反对(0)

#17楼 [楼主] 2019-09-29 18:07 五月的仓颉

@ 悲惨世界

引用

原文不是“非索引字段”么？

哦看错了

非索引字段查询我的理解是，MongoDB充分使用系统内存作为缓存，这样大部分操作只读内存，只要内存越大，MongoDB的查询速度就越快

网上我看过MongoDB和MySql基准性能测试的对比，在读取的数据规模不大的情况下，MongoDB的查询速度远高于MySql

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。