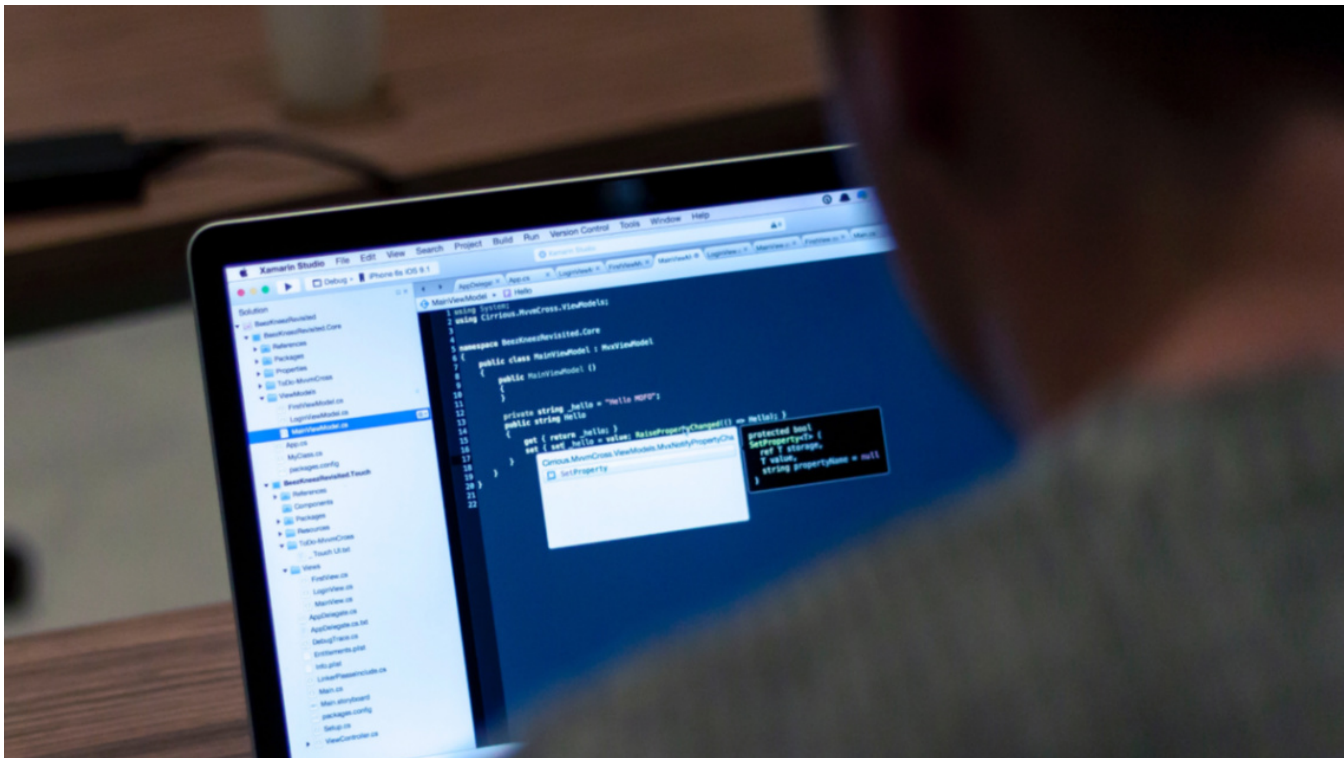


16 | 高性能NoSQL

2018-06-02 李运华



16 | 高性能NoSQL

朗读人：黄洲君 16'05" | 7.37M

关系数据库经过几十年的发展后已经非常成熟，强大的 SQL 功能和 ACID 的属性，使得关系数据库广泛应用于各式各样的系统中，但这并不意味着关系数据库是完美的，关系数据库存在如下缺点。

- 关系数据库存储的是行记录，无法存储数据结构

以微博的关注关系为例，“我关注的人”是一个用户 ID 列表，使用关系数据库存储只能将列表拆成多行，然后再查询出来组装，无法直接存储一个列表。

- 关系数据库的 schema 扩展很不方便

关系数据库的表结构 schema 是强约束，操作不存在的列会报错，业务变化时扩充列也比较麻烦，需要执行 DDL (data definition language, 如 CREATE、ALTER、DROP 等) 语句修改，而且修改时可能会长时间锁表（例如，MySQL 可能将表锁住 1 个小时）。

- 关系数据库在大数据场景下 I/O 较高

如果对一些大量数据的表进行统计之类的运算，关系数据库的 I/O 会很高，因为即使只针对其中某一列进行运算，关系数据库也会将整行数据从存储设备读入内存。

- 关系数据库的全文搜索功能比较弱

关系数据库的全文搜索只能使用 like 进行整表扫描匹配，性能非常低，在互联网这种搜索复杂的场景下无法满足业务要求。

针对上述问题，分别诞生了不同的 NoSQL 解决方案，这些方案与关系数据库相比，在某些应用场景下表现更好。但世上没有免费的午餐，NoSQL 方案带来的优势，本质上是牺牲 ACID 中的某个或者某几个特性，因此我们不能盲目地迷信 NoSQL 是银弹，而应该将 NoSQL 作为 SQL 的一个有力补充，NoSQL != No SQL，而是 NoSQL = Not Only SQL。

常见的 NoSQL 方案分为 4 类。

- K-V 存储：解决关系数据库无法存储数据结构的问题，以 Redis 为代表。
- 文档数据库：解决关系数据库强 schema 约束的问题，以 MongoDB 为代表。
- 列式数据库：解决关系数据库大数据场景下的 I/O 问题，以 HBase 为代表。
- 全文搜索引擎：解决关系数据库的全文搜索性能问题，以 Elasticsearch 为代表。

今天，我来介绍一下各种高性能 NoSQL 方案的典型特征和应用场景。

K-V 存储

K-V 存储的全称是 Key-Value 存储，其中 Key 是数据的标识，和关系数据库中的主键含义一样，Value 就是具体的数据。

Redis 是 K-V 存储的典型代表，它是一款开源（基于 BSD 许可）的高性能 K-V 缓存和存储系统。Redis 的 Value 是具体的数据结构，包括 string、hash、list、set、sorted set、bitmap 和 hyperloglog，所以常常被称为数据结构服务器。

以 List 数据结构为例，Redis 提供了下面这些典型的操作（更多请参考链接：<http://redis.cn/commands.html#list>）：

- LPOP key 从队列的左边出队一个元素。
- LINDEX key index 获取一个元素，通过其索引列表。
- LLEN key 获得队列（List）的长度。
- RPOP key 从队列的右边出队一个元素。

以上这些功能，如果用关系数据库来实现，就会变得很复杂。例如，LPOP 操作是移除并返回 key 对应的 list 的第一个元素。如果用关系数据库来存储，为了达到同样目的，需要进行下面的操作：

- 每条数据除了数据编号（例如，行 ID），还要有位置编号，否则没有办法判断哪条数据是第一条。注意这里不能用行 ID 作为位置编号，因为我们会往列表头部插入数据。
- 查询出第一条数据。
- 删除第一条数据。
- 更新从第二条开始的所有数据的位置编号。

可以看出关系数据库的实现很麻烦，而且需要进行多次 SQL 操作，性能很低。

Redis 的缺点主要体现在并不支持完整的 ACID 事务，Redis 虽然提供事务功能，但 Redis 的事务和关系数据库的事务不可同日而语，Redis 的事务只能保证隔离性和一致性（I 和 C），无法保证原子性和持久性（A 和 D）。

虽然 Redis 并没有严格遵循 ACID 原则，但实际上大部分业务也不需要严格遵循 ACID 原则。以上面的微博关注操作为例，即使系统没有将 A 加入 B 的粉丝列表，其实业务影响也非常小，因此我们在设计方案时，需要根据业务特性和要求来确定是否可以用 Redis，而不能因为 Redis 不遵循 ACID 原则就直接放弃。

文档数据库

为了解决关系数据库 schema 带来的问题，文档数据库应运而生。文档数据库最大的特点就是 no-schema，可以存储和读取任意的数据。目前绝大部分文档数据库存储的数据格式是 JSON（或者 BSON），因为 JSON 数据是自描述的，无须在使用前定义字段，读取一个 JSON 中不存在的字段也不会导致 SQL 那样的语法错误。

文档数据库的 no-schema 特性，给业务开发带来了几个明显的优势。

1. 新增字段简单

业务上增加新的字段，无须再像关系数据库一样要先执行 DDL 语句修改表结构，程序代码直接读写即可。

2. 历史数据不会出错

对于历史数据，即使没有新增的字段，也不会导致错误，只会返回空值，此时代码进行兼容处理即可。

3. 可以很容易存储复杂数据







JSON 是一种强大的描述语言，能够描述复杂的数据结构。例如，我们设计一个用户管理系统，用户的信息有 ID、姓名、性别、爱好、邮箱、地址、学历信息。其中爱好是列表（因为可以有多个爱好）；地址是一个结构，包括省市区楼盘地址；学历包括学校、专业、入学毕业年份信息等。如果我们用关系数据库来存储，需要设计多张表，包括基本信息（列：ID、姓名、性别、邮箱）、爱好（列：ID、爱好）、地址（列：省、市、区、详细地址）、学历（列：入学时间、毕业时间、学校名称、专业），而使用文档数据库，一个 JSON 就可以全部描述。

```
{
  "id": 10000,
  "name": "James",
  "sex": "male",
  "hobbies": [
    "football",
    "playing",
    "singing"
  ],
  "email": "user@google.com",
  "address": {
    "province": "GuangDong",
    "city": "GuangZhou",
    "district": "Tianhe",
    "detail": "PingYun Road 163"
  },
  "education": [
    {
      "begin": "2000-09-01",
      "end": "2004-07-01",
      "school": "UESTC",
      "major": "Computer Science & Technology"
    },
    {
      "begin": "2004-09-01",
      "end": "2007-07-01",
      "school": "SCUT",
      "major": "Computer Science & Technology"
    }
  ]
}
```

```
    J
  }
```

通过这个样例我们看到，使用 JSON 来描述数据，比使用关系型数据库表来描述数据方便和容易得多，而且更加容易理解。

文档数据库的这个特点，特别适合电商和游戏这类的业务场景。以电商为例，不同商品的属性差异很大。例如，冰箱的属性和笔记本电脑的属性差异非常大，如下图所示。

门款式:	对开门 		十字对开门 		多门 		三门 		双门 		单门 									
总容积:	561以上		451-560升		281-450升		231-280升		211-230升		150-210升		150L以下							
制冷方式:	风冷（无霜）		风直冷（混冷）				直冷													
压缩机:	变频（节能）		定频																	
能效等级:	一级	二级	三级																	
宽度:	60cm以下		60-65.9cm		66-70.9cm		71-75.9cm		76-80.9cm		81-85.9cm		86-90.9cm		91-95.9cm		96cm及以上			
深度:	50cm以下		51-55cm		56-60cm		61-65cm		66-70cm		71-75cm		76-80cm		80cm以上					
高度:	1m以下		1-1.4m		1.41-1.5m		1.51-1.6m		1.61-1.7m		1.71-1.8m		1.81-1.9m		1.9m以上					
控温方式:	电脑控温		机械控温																	
特色推荐:	原装进口		智能APP		多循环		吧台		制冰		门中门		干湿分储		零度保鲜					
面板材质:	钢化玻璃		彩钢		不锈钢															
面板颜色:	白色	银色	金色	印花	其它															
大家说:	冰箱不错		制冷效果好		冷冻室不错		空间大		压缩机不错		冷藏室够大		噪音很小		节能省电		两人用刚好		零度保鲜	

屏幕尺寸:	11.6英寸		12.5英寸		13.3英寸		14.0英寸		15.6英寸		17.3英寸		18.4英寸		其他									
处理器:	Intel CoreM		Intel i3		Intel i5低功耗版		Intel i5标准电压版		Intel i7低功耗版		Intel i7标准电压版		Intel 其他		AMD系列									
内存容量:	2G	4G	8G	16G	32G	其他																		
分类:	游戏本		轻薄本		二合一笔记本		常规笔记本		加固笔记本		其它													
硬盘容量:	500G		1T		128G固态		192G固态		256G固态		512G固态		128G+500G		128G+1T		256G+1T		512G+1T		混合硬盘		其他	
分辨率:	标准屏（1366×768）				高分屏（1600×900）				全高清屏（1920×1080）				超高清屏（2K/3K/4K）				其他							
显卡型号:	GTX1050		GTX1050Ti		GTX1060		GTX1070		GTX1080		GTX950M		GTX960M		GTX965M		GTX970M		GTX980M		MX150			
显卡类别:	集成显卡		入门级游戏独立显卡				高性能游戏独立显卡																	
显存容量:	1G	2G	3G	4G	6G	其他																		
系统:	Windows 7		Windows 8		Windows 10		MAC		DOS/Linux		其他													
裸机重量:	小于1KG		1-1.5KG		1.5-2kg		2-2.5kg		大于2.5KG															
特性:	窄边框		触控屏		机械键盘		背光键盘		指纹识别		3D实感摄像头		其他											
待机时长:	小于5小时		5-7小时		7-9小时		9小时以上																	
厚度:	10.0mm以下		10.0mm—15.0mm				15.1mm—20.0mm				20.0mm以上													
游戏性能:	入门级		发烧级		骨灰级																			
大家说:	东西不错		配置不错		散热很好		外观漂亮		屏幕大		性能不错		键盘不错		看电影不错		速度快		性价比高		开机速度			

即使是同类商品也有不同的属性。例如，LCD 和 LED 显示器，两者有不同的参数指标。这种业务场景如果使用关系数据库来存储数据，就会很麻烦，而使用文档数据库，会简单、方便许多，扩展新的属性也更加容易。

文档数据库 no-schema 的特性带来的这些优势也是有代价的，最主要的代价就是不支持事务。例如，使用 MongoDB 来存储商品库存，系统创建订单的时候首先需要减扣库存，然后再创建订单。这是一个事务操作，用关系数据库来实现就很简单，但如果用 MongoDB 来实现，就无

法做到事务性。异常情况下可能出现库存被扣减了，但订单没有创建的情况。因此某些对事务要求严格的业务场景是不能使用文档数据库的。

文档数据库另外一个缺点就是无法实现关系数据库的 join 操作。例如，我们有一个用户信息表和一个订单表，订单表中有买家用户 id。如果要查询“购买了苹果笔记本用户中的女性用户”，用关系数据库来实现，一个简单的 join 操作就搞定了；而用文档数据库是无法进行 join 查询的，需要查两次：一次查询订单表中购买了苹果笔记本的用户，然后再查询这些用户哪些是女性用户。

列式数据库

顾名思义，列式数据库就是按照列来存储数据的数据库，与之对应的传统关系数据库被称为“行式数据库”，因为关系数据库是按照行来存储数据的。

关系数据库按照行式来存储数据，主要有以下几个优势：

- 业务同时读取多个列时效率高，因为这些列都是按行存储在一起的，一次磁盘操作就能够把一行数据中的各个列都读取到内存中。
- 能够一次性完成对一行中的多个列的写操作，保证了针对行数据写操作的原子性和一致性；否则如果采用列存储，可能会出现某次写操作，有的列成功了，有的列失败了，导致数据不一致。

我们可以看到，行式存储的优势是在特定的业务场景下才能体现，如果不存在这样的业务场景，那么行式存储的优势也将不复存在，甚至成为劣势，典型的场景就是海量数据进行统计。例如，计算某个城市体重超重的人员数据，实际上只需要读取每个人的体重这一列并进行统计即可，而行式存储即使最终只使用一列，也会将所有行数据都读取出来。如果单行用户信息有 1KB，其中体重只有 4 个字节，行式存储还是会将整行 1KB 数据全部读取到内存中，这是明显的浪费。而如果采用列式存储，每个用户只需要读取 4 字节的体重数据即可，I/O 将大大减少。

除了节省 I/O，列式存储还具备更高的存储压缩比，能够节省更多的存储空间。普通的行式数据库一般压缩率在 3:1 到 5:1 左右，而列式数据库的压缩率一般在 8:1 到 30:1 左右，因为单个列的数据相似度相比行来说更高，能够达到更高的压缩率。

同样，如果场景发生变化，列式存储的优势又会变成劣势。典型的场景是需要频繁地更新多个列。因为列式存储将不同列存储在磁盘上不连续的空间，导致更新多个列时磁盘是随机写操作；而行式存储时同一行多个列都存储在连续的空间，一次磁盘写操作就可以完成，列式存储的随机写效率要远远低于行式存储的写效率。此外，列式存储高压缩率在更新场景下也会成为劣势，因为更新时需要将存储数据解压后更新，然后再压缩，最后写入磁盘。

基于上述列式存储的优缺点，一般将列式存储应用在离线的大数据分析和统计场景中，因为这种场景主要是针对部分列单列进行操作，且数据写入后就无须再更新删除。

全文搜索引擎

传统的关系型数据库通过索引来达到快速查询的目的，但是在全文搜索的业务场景下，索引也无能为力，主要体现在：

- 全文搜索的条件可以随意排列组合，如果通过索引来满足，则索引的数量会非常多。
- 全文搜索的模糊匹配方式，索引无法满足，只能用 like 查询，而 like 查询是整表扫描，效率非常低。

我举一个具体的例子来看看关系型数据库为何无法满足全文搜索的要求。假设我们做一个婚恋网站，其主要目的是帮助程序员找朋友，但模式与传统婚恋网站不同，是“程序员发布自己的信息，用户来搜索程序员”。程序员的信息表设计如下：

ID	姓名	性别	地点	单位	爱好	语言	自我介绍	
1	多隆	男	北京	猫厂	写代码、旅游、马拉松	Java、C++、PHP	技术专家，简单，为人热情	
2	如花	女	上海	鹅厂	旅游、美食、唱歌	PHP、Java	美女如花，风华绝代，貌美如花	
3	小宝	男	广州	熊厂	泡吧、踢球	Python、Go、C	我是一匹来自北方的狼	

我们来看一下这个简单业务的搜索场景：

- 美女 1：听说 PHP 是世界上最好的语言，那么 PHP 的程序员肯定是钱最多的，而且我妈一定要我找一个上海的。

美女 1 的搜索条件是“性别 + PHP + 上海”，其中“PHP”要用模糊匹配查询“语言”列，“上海”要查询“地点”列，如果用索引支撑，则需要建立“地点”这个索引。

- 美女 2：我好崇拜这些技术哥哥啊，要是能找一个鹅厂技术哥哥陪我旅游就更好了。

美女 2 的搜索条件是“性别 + 鹅厂 + 旅游”，其中“旅游”要用模糊匹配查询“爱好”列，“鹅厂”需要查询“单位”列，如果要用索引支撑，则需要建立“单位”索引。

- 美女 3：我是一个“女程序员”，想在北京找一个猫厂的 Java 技术专家。

美女 3 的搜索条件是“性别 + 猫厂 + 北京 + Java + 技术专家”，其中“猫厂 + 北京”可以通过索引来查询，但“Java”“技术专家”都只能通过模糊匹配来查询。

- 帅哥 4：程序员妹子有没有漂亮的呢？试试看看。

帅哥 4 的搜索条件是“性别 + 美丽 + 美女”，只能通过模糊匹配搜索“自我介绍”列。

以上只是简单举个例子，实际上搜索条件是无法列举完全的，各种排列组合非常多，通过这个简单的样例我们就可以看出关系数据库在支撑全文搜索时的不足。

1. 全文搜索基本原理

全文搜索引擎的技术原理被称为“倒排索引”（Inverted index），也常被称为反向索引、置入档案或反向档案，是一种索引方法，其基本原理是建立单词到文档的索引。之所以被称为“倒排”索引，是和“正排”索引相对的，“正排索引”的基本原理是建立文档到单词的索引。我们通过一个简单的样例来说明这两种索引的差异。

假设我们有一个技术文章的网站，里面收集了各种技术文章，用户可以在网站浏览或者搜索文章。

正排索引示例：

文章 ID	文章名称	文章内容
1	敏捷架构设计原则	省略具体内容，文档内容包含：架构、设计、架构师等单词
2	Java 编程必知必会	省略具体内容，文档内容包含：Java、编程、面向对象、类、架构、设计等单词
3	面向对象葵花宝典是什么	省略具体内容，文档内容包含：设计、模式、对象、类、Java 等单词

（注：文章内容仅为示范，文章内容实际上存储的是几千字的内容。）

正排索引适用于根据文档名称来查询文档内容。例如，用户在网站上单击了“面向对象葵花宝典是什么”，网站根据文章标题查询文章的内容展示给用户。

倒排索引示例：

单词	文档 ID 列表
架构	1, 2
设计	1, 2, 3
Java	2, 3

（注：表格仅为示范，不是完整的倒排索引表格，实际上的倒排索引有成千上万行，因为每个单词就是一个索引。）

倒排索引适用于根据关键词来查询文档内容。例如，用户只是想看看“设计”相关的文章，网站需要将文章内容中包含“设计”一词的文章都搜索出来展示给用户。

2. 全文搜索的使用方式

全文搜索引擎的索引对象是单词和文档，而关系数据库的索引对象是键和行，两者的术语差异很大，不能简单地等同起来。因此，为了让全文搜索引擎支持关系型数据的全文搜索，需要做一些转换操作，即将关系型数据转换为文档数据。

目前常用的转换方式是将关系型数据按照对象的形式转换为 JSON 文档，然后将 JSON 文档输入全文搜索引擎进行索引。我同样以程序员的基本信息表为例，看看如何转换。

将前面样例中的程序员表格转换为 JSON 文档，可以得到 3 个程序员信息相关的文档，我以程序员 1 为例：

```
{
  "id": 1,
  " 姓名 ": " 多隆 ",
  " 性别 ": " 男 ",
  " 地点 ": " 北京 ",
  " 单位 ": " 猫厂 ",
  " 爱好 ": " 写代码, 旅游, 马拉松 ",
  " 语言 ": "Java、C++、PHP",
  " 自我介绍 ": " 技术专家, 简单, 为人热情 "
}
```

全文搜索引擎能够基于 JSON 文档建立全文索引，然后快速进行全文搜索。以 Elasticsearch 为例，其索引基本原理如下：

Elasticsearch 是分布式的文档存储方式。它能存储和检索复杂的数据结构——序列化成为 JSON 文档——以实时的方式。

在 Elasticsearch 中，每个字段的所有数据都是默认被索引的。即每个字段都有为了快速检索设置的专用倒排索引。而且，不像其他多数的数据库，它能在相同的查询中使用所有倒排索引，并以惊人的速度返回结果。

(<https://www.elastic.co/guide/cn/elasticsearch/guide/current/data-in-data-out.html>)

小结

今天我为你讲了为了弥补关系型数据库缺陷而产生的 NoSQL 技术，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，因为 NoSQL 的方案功能都很强大，有人认为 NoSQL = No SQL，架构设计的时候无需再使用关系数据库，对此你怎么看？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）



版权归极客邦科技所有，未经许可不得转载

精选留言



鹅米豆发

30

关于NoSQL，看过一张图，挺形象：“1970, We have no SQL” -> “1980, Know SQL” -> “2000, NoSQL” -> “2005, Not only SQL” -> “2015, No, SQL”。目前，一些新型数据库，同时具备了NoSQL的扩展性和关系型数据库的很多特性。关系型和NoSQL数据库的选型。考虑几个指标，数据量、并发量、实时性、一致性要求、读写分布和类型、安全性、运维性等。根据这些指标，软件系统可分成几类。

- 1.管理型系统，如运营类系统，首选关系型。
- 2.大流量系统，如电商单品页的某个服务，后台选关系型，前台选内存型。
- 3.日志型系统，原始数据选列式，日志搜索选倒排索引。
- 4.搜索型系统，指站内搜索，非通用搜索，如商品搜索，后台选关系型，前台选倒排索引。
- 5.事务型系统，如库存、交易、记账，选关系型+缓存+一致性协议，或新型关系数据库。
- 6.离线计算，如大量数据分析，首选列式，关系型也可以。
- 7.实时计算，如实时监控，可以选时序数据库，或列式数据库。

2018-06-04

作者回复

求原图😁

2018-06-04



公号-Java大后端

👍 9

需求驱动架构，无论选用RDB/NoSQL/DRDB，一定是以需求为导向，最终的数据存储方案也必然是各种权衡的设计妥协。

没有银弹，因为不同的应用需求对数据的要求也各不相同。当前我们不可能找到一个存储方案能满足所有的应用需求，但是我们可以通过认识到各种存储方案的优点与缺点（陷阱），在实际应用中，根据应用场景的不同要求（比如对可用性、一致性、易用性、支持事务、响应延迟、伸缩性等），按照上述特性要求的优先级排序，找到最合适的方案并“混合搭配”使用各种数据存储方案。

多种方案搭配混用必然会增加应用的复杂性与增加运维成本，但同时也带来了系统更多的灵活性。

举例：传统/互联网金融领域目前就不可能离开RDB。

2018-06-03



空档滑行

👍 5

NoSQL出现的原因是针对解决某一特定问题，这个问题用关系型数据库无法很好的解决。所以就注定了NO SQL不会是个大而全的东西，虽然这几年特性不断增加，使得部分业务场景可以完全不依赖关系数据库，但是关系数据库仍然是最容易理解，适用场景最广泛的数据库。就像OLAP无法完全代替离线计算，HTAP无法完全代替OLAP+OLTP一样，NOSQL是很好的补充，不是为了代替

2018-06-02



曹铮

👍 5

看过一些资料，RDB当年（比我还大好多）也是在数据库大战的血雨腥风中一条血路杀出来的。现在回魂的document db当前也是RDB对手之一。如果真有这么问题，怎么还能脱颖而出，主宰软件开发领域这么久。只能说互联网和物联网的兴起使得应用场景向海量数据，线下上分析，读多写也多这些情况偏移了，这些场景对ACID的要求很低。

但现在大多数应用的也许还是直接面对用户，要求数据有一定可靠性，数据总量和并发量并没那么高。RDB技术成熟，人才易得，声明式SQL语法让开发者忽略底层实现，关系型的模型也符合人的思考模式。而且现在大多数一流的RDB都集成了基本的文档存储和索引，空间存储，全文检索，数据分析等功能。在没达到一定规模和深度前，完全可以用个RDB来做MVP，甚至搞定中小型也许场景。

从码农的角度看，我还是更崇拜关系型数据库，因为其底层实现里包罗了算法，系统，网络，分布式，数学统计学各种绝世武功。

前几年在NoSQL炒起来没多久，NewSQL的概念又被提出了。现在各路牛人都投入到RDB的研发中，成型的也有不少。虽然不太可能完全取代现在的各种NoSQL，但也许能收复不少失地。

历史就是个循环，天下大势分久必合合久必分...

2018-06-02

作者回复

RDB的功能是最强大的

2018-06-04



铃兰Neko

4

认真的看到现在，有两点疑问。

- 1.es和lucene这种，一般当做搜索引擎，也划在nosql里面合适吗？
- 2.能否给一些常见nosql性能和数据量的数据，网上搜不大到，唯一知道的都是从ppt里面扣的

另外，nosql那个图原图应该在reddit上，地址是

<http://i.imgur.com/lkG9Vm8.jpg>

2018-06-04

作者回复

1. 只要是为了弥补关系数据库的缺陷的方案，都可算nosql
2. 量级一般都是上万，但和测试硬件测试用例相关，如果要用，最好自己测试

2018-06-06



Snway

4

No SQL并非银弹，如ACID方面就无法跟关系型数据库相比，实际运用中，需要根据业务场景来分析，比较好的做法是，No SQL+关系型数据库结合使用，取长补短。如我们之前的做法是将商品/订单/库存等相关基本信息放在关系型数据库中(如MySQL，业务操作上支持事务，保证逻辑正确性)，缓存可以用Redis(减少DB压力)，搜索可以用Elasticsearch(提升搜索性能，可通过定时任务定期将DB中的信息刷到ES中)。

2018-06-03



幻影霸者

3

mongodb4.0不是支持acid事务吗？

2018-06-03

作者回复

官方宣称会实现事务，但需要看最终实现，实现事务不是完全没有代价的，要么性能降低，要么灵活性降低

2018-06-04



fiseasky

3

跨库操作如何做事物呢？看到这期了还是没有答案吗？

2018-06-03

作者回复

跨库操作不建议事物，效率低，容易出错，用最终一致性

2018-06-04

**monkay**

👍 3

有个商品搜索的需求，怎么评估elasticsearch和solr哪个更合适？

2018-06-02

作者回复

功能上其实都可以，关键看实时性和性能，如果都满足，按照简单原则，你熟悉哪个就用哪个

2018-06-02

**Gotta**

👍 2

根据实际需求来决定需要使用NoSQL或关系型数据库，或者结合使用，取长补短。实际的场景中，一般都是两者相互结合:例如我司，关系型数据库为主，因为其提供了强大的SQL功能以及ACID事务属性；同时，也是用了Redis，其主要作用是缓存，缓解关系型数据库的访问压力。一个字，怎么使用或者结合使用，都是需要根据实际的生产需求来决定，而不是拍着屁股决定NoSQL就真的等于No SQL嘿嘿嘿。

2018-06-02

**王磊**

👍 2

个人理解，关系型数据库也有列式存储，如Greenplum, Redshift, 所以这个不是NoSQL和关系数据库的区别吧。

2018-06-02

**明翼**

👍 1

文档数据库和es有何区别，es我看也是对scheme不敏感的啊

2018-06-14

作者回复

文档数据库定位于存储和访问，es定位于搜索，但目前差别并不是很大，因为系统边界的扩充，就像mongodb也要支持ACID，mysql也要支持文档存储

2018-06-14

**Otto**

👍 1

你好，我有找到一张 history of nosql图档，分享给您。

https://www.reddit.com/r/ProgrammerHumor/comments/2mk8sb/history_of_nosql/

2018-06-11

**Michael**

👍 1

老师您好，问个题外话😁

如今各种新技术层出不穷，像那种底层的大块头之类的书籍，比如深入理解计算机系统，编译原理这样的，还有必要深入学习吗？

像深入理解计算机系里面的反汇编分析在实际工作中确是用不到，更别提开发个编译器了。面对这些看似不错的书籍，但又觉得里面的内容无法运用到实际中，有些迷茫，望老师指点。

2018-06-05

作者回复

请到聊聊架构公众号搜索《当我们聊技术实力的时候，我们到底在聊什么》，我写了一篇文章来回答你这个问题。

2018-06-05



ncicheng

1

对于需要存储大量非文本格式的文档（如doc），项目中一般采取blob，但是搜索文档内容不容易，如果换成NoSQL来存储是不是比现在好？谢谢~

2018-06-05

作者回复

赶紧换es，谁用谁知道😄

2018-06-06



姜哥

1

华仔，我们公司刚起步的时候有2个应用，数据库用的mysql，日志也存储在mysql，用户增长很快，没有专职架构师，现在准备重构以适应业务增长，考虑hipster on kubernetes，昨天同事之间交流了一下方案：gateway + registry + k8s traefik ingress，通过k8s实现hipster技术栈的高可用，所有微服务容器化，第一阶段准备拆成5个service，在gateway做路由规则和鉴权，日志部分用mongodb，mysql暂时不分库，订单rocketMQ队列化，这样合理吗？有什么更好的建议？

2018-06-02

作者回复

我感觉你们一下子引入这么多对你们来说是新的技术，风险有点大。

2018-06-02



yankerzhu

1

记得有文章介绍workday，系统中用到关系型数据库和非关系型数据库，但是关系型数据库中只有三个表，用来存储元数据，主要在系统启动初始化时用到，业务数据全部在非关系型数据库中，如果有人用过这样架构的，请分享经验！

2018-06-02



caison

0

redis的操作应该是原子性的吧，因为redis的单线程的

2018-07-25

作者回复

是的

2018-07-26



bubble

0

跨库操作不建议事物，效率低，容易出错，用最终一致性，如何保证最终一致性

2018-07-19

| 作者回复

最常用的就是定时校验，记录很多关键日志？，或者用zk这种分布式协调系统

2018-07-20



YangXinjie

👍 0

所谓倒排索引其实就是传统索引，只是特定场景下的查询顺序不同罢了。那个词语索引到文档号的例子，把文档号看成rowid就是普通索引了

2018-07-12

| 作者回复

应该说都是索引，不是传统索引

2018-07-13