第七次课作业

姓名：张霁雯　学号：16307110435

本次作业：分析总结 NoSQL 数据库与关系数据库的差异和各自特点。请具体介绍两种不同的 NoSQL 数据库系统的架构、特点和应用场景。

1. NoSQL 数据库与关系数据库的差异

NoSQL 是一种不同于关系数据库的数据库管理系统设计方方式，是对非关系型数据库的统称。它所采用的数据模型并非传统的关系数据库的关系模型，而是类似键/值、列族、文档等非关系模型。NoSQL 数据库没有固定的表结构，通常也不存在连接操作，也没有严格遵守 ACID 规则，所以相比起关系数据库，具有更灵活的水平课扩展性，可以支持海量数据存储。

通常 NoSQL 数据库具有以下 3 个特点：

- **灵活的可扩展性**

  传统的关系型数据库由于自身设计机理的问题，通常很难实现“横向扩展”。在面对数据库负载大规模增加时，往往需要通过升级硬件来实现“纵向扩展”。而 NoSQL 数据库在设计之初就是为了满足“横向扩展”的需求，因此天生具备良好的水平扩展能力。

- **灵活的数据模型**

  关系模型是关系数据库的基石，它以完备的关系代数理论为基础，具有规范的定义，遵守各种严格的约束条件。这种做法虽然保证了业务系统对数据一致性的需求，但是过于死板的数据模型，也意味着无法满足各种新兴的业务需求。相反，NoSQL 数据库天生就旨在摆脱关系数据库的各种束缚条件，摈弃了流行多年的关系数据模型，转而采用键/值、列族等非关系模型，允许在一个数据元素里存储不同类型的数据。

  文档数据库 将半结构化数据存储为文档的数据库，例如 CouchDB, MongoDB。每个 NoSQL 文档的架构是不同的，可以让用户更灵活地整理和存储应用程序数据，减少可选值所需的存储。典型应用场景是存储类似 JSON 格式的内容，可对某些字段建立索引功能。这一类的数据库是最像关系型的数据库，常用格式是 JSON 或 XML。

  - 优点:数据结构要求不严格，表结构可变，无需像 RMDBS 一样预定义表结构。
  - 缺点:查询性能不高，而且缺乏统一的查询语法。

  列式数据 按列存储数据的数据库，对应 HBase,BigTable 等。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的 IO 优势。典型应用场景是数据需要按列存储。

  - 优点:查找速度快，可扩展性强，更容易进行分布式扩展。 针对某一列或者某几列的查询有非常大的 IO 优势。
  - 缺点:功能相对局限。

  内存键值存储 可以通过 key 快速查询到 value 的数据库，对应 Redis，Memcached 等。内存缓存可将重要数据存储在内存中以实现低延迟访问，从而提高应用程序性能。一般来说，存储不管 value 的格式，照单全收，因此典型的应用场景是针对读取密集型应用程序工作负载(例如社交网络、游戏、媒体共享和 QA 门户)，内容缓存，主要用于处理大量数据的高访问负载，也用于一些日志系统等等。

  - 优点:查找速度快。
  - 缺点:数据无结构化，通常只被当作字符串或者二进制数据。

图形数据库 顾名思义，可以存储图形数据的数据库。数据模型是图模型，可存储顶点以及称为边缘的直接链路。图形数据库可以在 SQL 和 NoSQL 数据库上构建。顶点和边缘可以拥有各自的相关属性，常用于构建知识图谱。这种数据库善于处理大量复杂、互连接、低结构化、查询频繁的数据，典型应用场景是社交网络，推荐系统等。

- 优点:利用图结构相关算法。比如最短路径寻址，N 度关系查找等。
- 缺点:很多时候需要对整个图做计算才能得出需要的信息；不太好做分布式的集群方案。

- **与云计算紧密结合**

  云计算有很好的水平扩展能力，可以根据资源使用情况进行自由伸缩，各种资源可以动态加入或退出，NoSQL 数据库可以凭借自身良好的横向扩展能力，充分自由利用云计算基础设施， 很好地融入到云计算环境中，构建基于 NoSQL 的云数据库服务。

下图给出了 NoSQL 和关系型数据库（RDBMS）的简单比较。

从表中可以看出，关系数据库的——

优势：以完善的关系代数理论作为基础，有严格的标准，支持事务 ACID 四性，借助索引机制可以实现高效的查询，技术成熟，有专业公司的技术支持。

劣势：可扩展性较差，无法较好地支持海置数据存储，数据模式过于死板，无法较好地支持 Web 2.0 应用，事务机制影响了系统的整体性能等。

NoSQL 数据库的——

优势：可以支持超大规模数据存储，灵活的数据模型可以很好地支持 Web 2.0 应用；具有强大的横向扩展能力等。

劣势：缺乏数学理论基础，复杂查询性能不高，一般都不能实现事务强一致性，很难实现数据完整性，技术尚不成熟，缺乏专业团队的技术支持，维护较困难等。

比较标准	RDBMS	NoSQL	备注
数据库原理	完全支持	部分支持	RDBMS 有关系代数理论作为基础。 NoSQL 没有统一的理论基础。
数据规模	大	超大	RDBMS 很难实现横向扩展，纵向扩展空间较有限，性能会随数据规模的增大而降低。 NoSQL 可以很容易通过添加更多设备来支持更大规模的数据。
数据库模式	固定	灵活	RDBMS 需要定义数据库模式，严格遵守数据定义和相关约束条件。 NoSQL 不存在数据库模式，可以灵活定义并存储不同类型的数据。
查询效率	快	可以实现高效的简单查询，但是不具备高度结构化查询等特性，复杂查询的性能不尽人意	RDBMS 借助索引机制可实现快速查询（包括记录查询和范围查询）。 很多 NoSQL 数据库没有面向复杂查询的索引，虽然 NoSQL 可以使用 MapReduce 来加速查询，但在复杂查询方面的性能仍不如 RDBMS。
一致性	强一致性	弱一致性	RDBMS 严格遵守事务 ACID 模型，可以保证事务强一致性。 很多 NoSQL 数据库放松了对事务 ACID 四性的要求，而是遵守 BASE 模型，只能保证最终一致性。
数据完整性	容易实现	很难实现	任何一个 RDBMS 都可以很容易实现数据完整性，比如通过主键或者非空约束来实现实体完整性，通过主键、外键来实现参照完整性，通过约束或者触发器来实现用户自定义完整性。 但在 NoSQL 数据库却无法实现。
扩展性	一般	好	RDBMS 很难实现横向扩展，纵向扩展的空间也比较有限。 NoSQL 在设计之初就充分考虑了横向扩展的需求，可以很容易通过添加廉价设备实现扩展。
可用性	好	很好	RDBMS 在任何时候都以保证数据一致性为优先目标，其次才是优化系统性能，随着数据规模的增大， RDBMS 为了保证严格的一致性，只能提供相对较弱的可用性。大多数 NoSQL 都能提供较高的可用性
标准化	是	否	RDBMS 已经标准化（SQL）。 NoSQL 还没有行业标准，不同的 NoSQL 数据库都有自己的查询语言，很难规范应用程序接口。
技术支持	高	低	RDBMS 经过几十年的发展，已经非常成熟，Oracle 等大型厂商都可以提供很好的技术支持。 NoSQL 在技术支持方面仍然处于起步阶段，还不成熟，缺乏有力的技术支持。
可维护性	复杂	复杂	RDBMS 需要专门的数据库管理员(DBA)维护。NoSQL 数据库虽然没有 DBMS 复杂，也难以维护。

此外，在数据存储方面，关系型数据库和 NoSQL 数据库还具有以下几点不同：

1) 存储方式

关系型数据库是表格式的，因此存储在表的行和列中。他们之间很容易关联协作存储，提取数据很方便。而 Nosql 数据库则与其相反，他是大块的组合在一起。通常存储在数据集中，就像文档、键值对或者图结构。

2) 存储结构

关系型数据库对应的是结构化数据，数据表都预先定义了结构（列的定义），结构描述了数据的形式和内容。这一点对数据建模至关重要，虽然预定义结构带来了可靠性和稳定性，但是修改这些数据比较困难。而 Nosql 数据库基于动态结构，使用与非结构化数据。因为 Nosql 数据库是动态结构，可以很容易适应数据类型和结构的变化。

3) 存储规范

关系型数据库的数据存储为了更高的规范性，把数据分割为最小的关系表以避免重复，获得精简的空间利用。虽然管理起来很清晰，但是单个操作设计到多张表的时候，数据管理就显得有点麻烦。而 Nosql 数据存储在平面数据集中，数据经常可能会重复。单个数据库很少被分隔开，而是存储成了一个整体，这样整块数据更加便于读写。

4) 规范化

在数据库的设计开发过程中开发人员通常会面对同时需要对一个或者多个数据实体(包括数组、列表和嵌套数据)进行操作，这样在关系型数据库中，一个数据实体一般首先要分割成多个部 分，然后再对分割的部分进行规范化，规范化以后再分别存入到多张关系型数据表中，这是一个复 杂的过程。好消息是随着软件技术的发展，相当多的软件开发平台都提供一些简单的解决方法，例如，可以利用 ORM 层(也就是对象关系映射)来将数据库中对象模型映射到基于 SQL 的关系型 数据库中去以及进行不同类型系统的数据之间的转换。

5) 读写性能

关系型数据库十分强调数据的一致性，并为此降低读写性能付出了巨大的代价，虽然关系型数 据库存储数据和处理数据的可靠性很不错，但一旦面对海量数据的处理的时候效率就会变得很差， 特别是遇到高并发读写的时候性能就会下降的非常厉害。

总体来说，关系数据库和 NoSQL 数据库各有优缺点，彼此无法取代，各自有各自适用的场景。如关系数据库适宜在电信、银行等领域的关键业务系统中使用，因为这些业务需要保证强事务一致性；而 NoSQL 数据库适用于互联网企业、传统企业的非关键业务（比如数据分析）。

2. 介绍两种不同的 NoSQL 数据库系统的架构、特点和应用场景

1. Redis

1.1 Redis 特点

Redis 是一个高性能的 key-value 类型的内存数据库，是完全开源免费的。整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存。因为是纯内存操作, Redis 的性能非常出色, 每秒可以处理超过 10 万次读写操作, 是已知性能最快的 Key-Value DB。

Redis 的出色之处不仅仅是性能，它最大的魅力是支持保存多种数据结构，此外单个 value 的最大限制是 1GB，因此 Redis 可以用来实现很多有用的功能，比方说用 List 来做 FIFO 双向链表，实现一个轻量级的高性 能消息队列服务，用他的 Set 可以做高性能的 tag 系统等等。另外 Redis 也可以对存入的 Key-Value 设置 expire 时间。总体来说 Redis 的特点如下：

- **速度快**，因为数据存在内存中，读写的速度极快；
- **支持丰富的数据类型**，支持 string，list，set，sorted set，hash；
- **原子性：**Redis 的所有操作都是原子性的。
- **支持事务**：事务中任意命令执行失败，其余命令依然被执行。也就是说 Redis 事务不保证原子性，也不支持回滚；事务中的多条命令被一次性发送给服务器，服务器在执行命令期间，不会去执行其他客户端的命令请求。
- **丰富的特性**：可用于缓存，消息（支持 publish/subscribe 通知），按 key 设置过期时间，过期后将会自动删除。具体过期键的策略有：定时删除（缓存过期时间到就删除，创建 timer 耗 CPU），惰性删除（获取的时候检查，不获取一直留在内存，对内存不友好），定期删除（CPU 和内存的折中方案）
- **支持数据持久化**，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用；
- **支持数据备份**，即 master - slave 模式的数据备份。

Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写（在 string 类型上会消耗较多内存），因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

1.2 Redis 应用场景

1) 会话缓存（Session Cache）

最常用的一种使用 Redis 的情景是会话缓存（session cache）。用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，可以保证用户信息不会全部丢失。

2) 队列

Reids 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。

3) 全页缓存

大型互联网公司都会使用 Redis 作为缓存存储数据，提升页面相应速度。即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降。

4) 排行榜/计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单。

1.3 Redis 的高可用架构

1) 持久化

Redis 是内存型数据库，为了保证数据在断电后不会丢失，需要将内存中的数据持久化到硬盘上。Redis 提供了两种持久化的方式，分别是 RDB（Redis DataBase）和 AOF（Append Only File）。

- RDB：简而言之，就是在不同的时间点，将 redis 存储的数据生成快照并存储到磁盘等介质上，可以将快照复制到其他服务器从而创建具有相同数据的服务器副本。如果系统发生故障，将会丢失最后一次创建快照之后的数据。如果数据量大，保存快照的时间会很长。
- AOF：换了一个角度来实现持久化，那就是将 redis 执行过的所有写指令记录下来，在下次 redis 重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复。将写命令添加到 AOF 文件（append only file）末尾。

  使用 AOF 持久化需要设置同步选项，从而确保写命令同步到磁盘文件上的时机。这是因为对文件进行写入并不会马上将内容同步到磁盘上，而是先存储到缓冲区，然后由操作系统决定什么时候同步到磁盘。选项同步频率 always 每个写命令都同步，eyerysec 每秒同步一次，no 让操作系统来决定何时同步，always 选项会严重减低服务器的性能，everysec 选项比较合适，可以保证系统崩溃时只会丢失一秒左右的数据，并且 Redis 每秒执行一次同步对服务器几乎没有任何影响。no 选项并不能给服务器性能带来多大的提升，而且会增加系统崩溃时数据丢失的数量。随着服务器写请求的增多，AOF 文件会越来越大。Redis 提供了一种将 AOF 重写的特性，能够去除 AOF 文件中的冗余写命令。

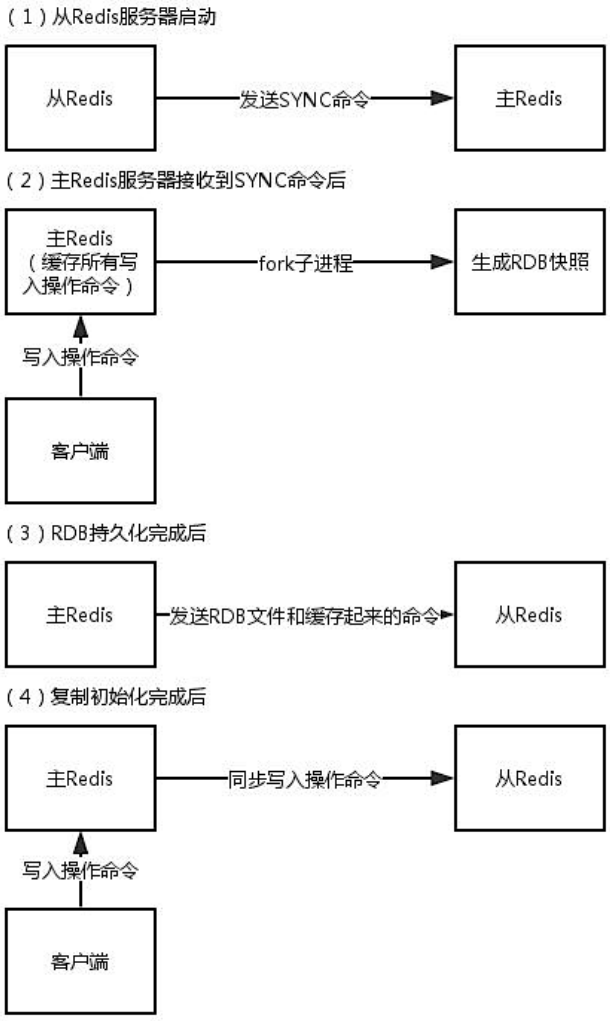
RDB 和 AOF 两种方式可以同时使用，在这种情况下如果 redis 重启，则会优先采用 AOF 方式来进行数据恢复，这是因为 AOF 方式的数据恢复完整度更高。如果没有数据持久化的需求，也完全可以关闭 RDB 和 AOF 方式，这样 redis 将变成一个纯内存数据库。

2) 主从复制

Redis 为了解决单点数据库问题，会把数据复制多个副本部署到其他节点上，通过复制，实现 Redis 的高可用性，实现对数据的冗余备份，保证数据和服务的高度可靠性。

Redis 有主从和主备两种方式解决单点问题，主备（keepalived）模式下主机备机对外提供同一个虚拟 IP，客户端通过虚拟 IP 进行数据操作，正常期间主机一直对外提供服务，宕机后 VIP 自动漂移到备机上。

主从模式下当 Master 宕机后，通过选举算法(Paxos、Raft)从 slave 中选举出新 Master 继续对外提供服务，主机恢复后以 slave 的身份重新加入，此模式下可以使用读写分离，如果数据量比较大，不希望过多浪费机器，还希望在宕机后，做一些自定义的措施，比如报警、记日志、数据迁移等操作，推荐使用主从方式，因为和主从搭配的一般还有个管理监控中心（哨兵）。



①从数据库向主数据库发送 sync(数据同步)命令。

②主数据库接收同步命令后，会保存快照，创建一个 RDB 文件。

③当主数据库执行完保持快照后，会向从数据库发送 RDB 文件，而从数据库会接收并载入该文件。

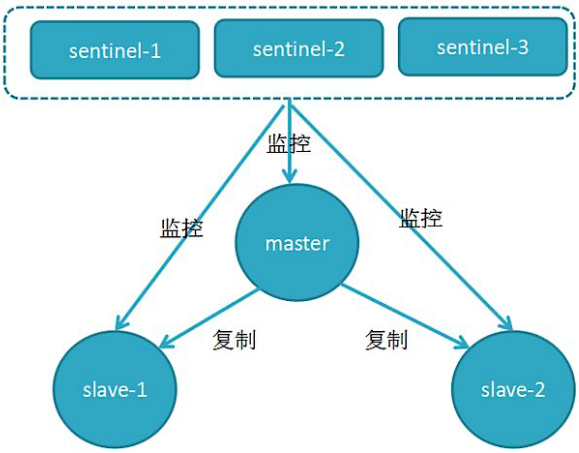
④主数据库将缓冲区的所有写命令发给从服务器执行。

⑤以上处理完之后，之后主数据库每执行一个写命令，都会将被执行的写命令发送给从数据库。可以同步发送也可以异步发送，同步发送可以不用每台都同步，可以配置一台 master，一台 slave，同时这台 salve 又作为其他 slave 的 master。异步方式无法保证数据的完整性，比如在异步同步过程中主机突然宕机了，也称这种方式为数据弱一致性。

3) 哨兵模式

哨兵是 Redis 集群架构中非常重要的一个组件，哨兵的出现主要是解决了主从复制出现故障时需要人为干预的问题。Redis 哨兵主要功能为：

- 集群监控：负责监控 Redis master 和 slave 进程是否正常工作
- 消息通知：如果某个 Redis 实例有故障，哨兵负责发送消息作为报警通知给管理员
- 故障转移：如果 master node 失效，会自动转移到 slave node 上
- 配置中心：如果故障转移发生了，通知 client 客户端新的 master 地址



当主节点出现故障时，由 Redis Sentinel 自动完成故障发现和转移，并通知应用方，实现高可用性。哨兵机制建立了多个哨兵节点(进程)，共同监控数据节点的运行状况。同时哨兵节点之间也互相通信，交换对主从节点的监控状况。每隔 1 秒每个哨兵会向整个集群：Master 主服务器+Slave 从服务器+其他 Sentinel（哨兵）进程，发送一次 ping 命令做一次心跳检测。这就是哨兵用来判断节点是否正常的重要依据，涉及两个新的概念：主观下线和客观下线。一个哨兵节点判定主节点 down 掉是主观下线，只有半数哨兵节点都主观判定主节点 down 掉，此时多个哨兵节点交换主观判定结果，才会判定主节点客观下线。大部分情况下，哪个哨兵节点最先判断出该主节点客观下线，就会在各个哨兵节点中发起投票机制 Raft 算法（选举算法），最终被投为领导者的哨兵节点完成主从自动化切换的过程。

4) 集群模式

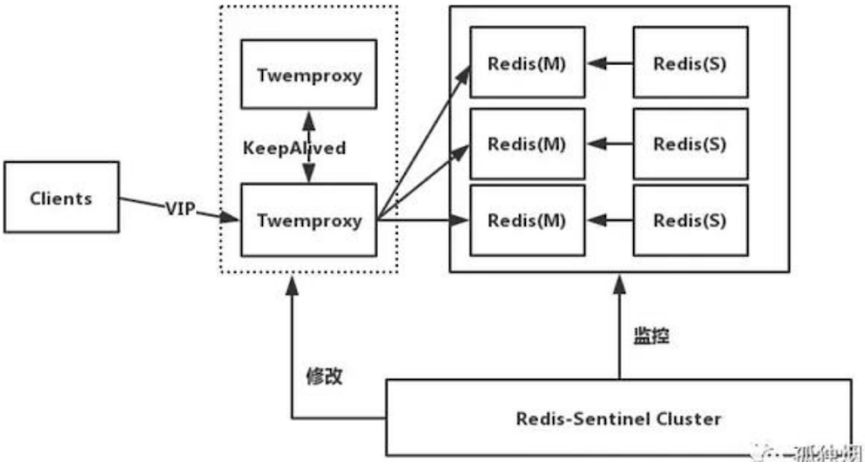
至少部署两台 Redis 服务器构成一个小的集群，主要有 2 个目的：

高可用性：在主机失效后，自动故障转移，使前端服务对用户无影响。

读写分离：将主机读压力分流到从机上。

使用集群可在客户端组件上实现负载均衡，根据不同服务器的运行情况，分担不同比例的读请求压力。

当数据量持续增加时，应用可根据不同场景下的业务申请对应的分布式集群。 最关键的是缓存治理，其中最重要的部分是加入了代理服务（Codis 和 Twemproxy）。 应用通过代理访问真实的 Redis 服务器进行读写，这样做的好处是避免越来越多的客户端直接访问 Redis 服务器难以管理，而造成风险。在代理层可以做对应的安全措施，比如限流、授权、分片，避免客户端越来越多的逻辑代码。代理层无状态的，可任意扩展节点，对于客户端来说，访问代理跟访问单机 Redis 一样。



2. MongoDB

2.1 MongoDB 介绍及特点

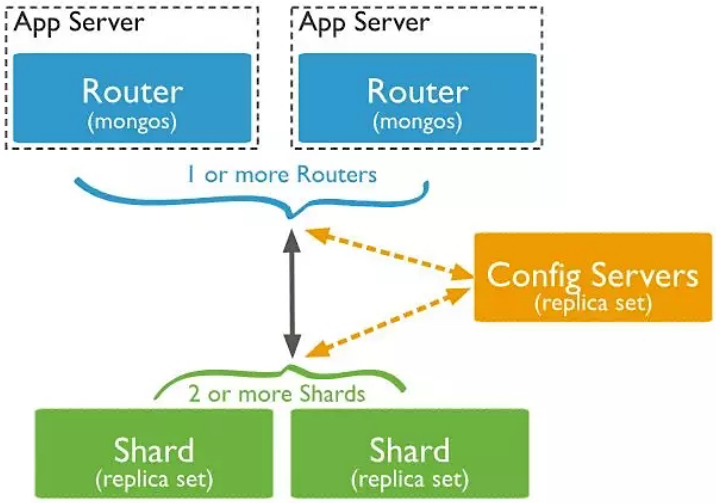
MongoDB 是一个基于分布式文件存储的数据库。由 C++语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。 MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中性能最丰富，最像关系数据库的。它支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。Mongo 最大的特点是支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。MongoDB 有以下特点：

- **更高的写负载**，MongoDB 拥有更高的插入速度。
- 处理很大的规模的单表，当数据表太大的时候可以很容易的分割表。
- **高可用性**，设置 M-S 不仅方便而且很快，MongoDB 还可以快速、安全及自动化的实现节点（数据中心）故障转移。
- **快速的查询，支持二维空间索引**，比如管道，因此可以快速及精确的从指定位置获取数据。MongoDB 在启动后会将数据库中的数据以文件映射的方式加载到内存中。如果内存资源相当丰富的话，这将极大地提高数据库的查询速度。
- **快速处理：**非结构化数据的爆发增长，增加列在有些情况下可能锁定整个数据库，或者增加负载从而导致性能下降，由于 MongoDB 的弱数据结构模式，添加 1 个新字段不会对旧表格有任何影响，整个过程会非常快速。

缺点是：只支持单文档事务；占用空间过大；没有成熟的维护工具。

2.2 Mongodb 分布式架构

MongoDB Sharding 是 MongoDB 数据库一种水平扩展的分布式架构。由 shards（存取数据，计算），config server（数据元信息、控制节点），mongos（路由，分发请求和汇聚结果）三个组件构成。



- shards： 一个 shard 为一组 mongod，通常一组为两台，主从或互为主从。这一组 mongod 中的数据时相同的，具体可见《mongodb 分布式之数据复制》。数据分割按有序分割方式，每个分片上的数据为某一范围的数据块，故可支持指定分片的范围查询，这同 google 的 BigTable 类似。数据块有指定的最大容量，一旦某个数据块的容量增长到最大容量时，这个数据块会切分成为两块；当分片的数据过多时，数据块将被迁移到系统的其他分片中。另外，新的分片加入时，数据块也会迁移。

- mongos：可以有多个，相当于一个控制中心，负责路由和协调操作，使得集群像一个整体的系统。 mongos 可以运行在任何一台服务器上，有些选择放在 shards 服务器上，也有放在 client 服务器上的。mongos 启动时需要从 config servers 上获取基本信息，然后接受 client 端的请求，路由到 shards 服务器上，然后整理返回的结果发回给 client 服务器。
- config server：存储集群的信息，包括分片和块数据信息。主要存储块数据信息，每个 config server 上都有一份所有块数据信息的拷贝，以保证每台 config server 上的数据的一致性。
- shard key :为了分割数据集, 需要制定分片 key 的格式, 类似于用于索引的 key 格式, 通常由一个或多个字段组成以分发数据，比如：

  { name : 1 }
  { _id : 1 }
  { lastname : 1, firstname : 1 }
  { tag : 1, timestamp : -1 }

  mongoDB 的分片为有序存储(1 为升序，-1 为降序)，shard key 相邻的数据通常会存在同一台服务（数据块）上。

2.3 MongoDB 应用场景

MongoDB 的应用场景与 MongoDB 的特性相匹配。由于 MongoDB 目前只支持单文档事务，因此需要复杂事务支持的场景暂时不适合。但 MongoDB 灵活的文档模型、强大的索引性能和灵活的可扩展性，使得它在游戏、物流、电商、内容管理、社交、物联网、视频直播等领域都已经有了广泛的使用。具体来说：

- 游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、更新
- 物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来。
- 社交场景，使用 MongoDB 存储存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能
- 物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析
- 视频直播，使用 MongoDB 存储用户信息、礼物信息等

具体来说，是否选用 MongoDB 可以参照如下表格：

应用特征	Yes/No
应用不需要事务及复杂 join 支持	Yes
新应用，需求会变，数据模型无法确定，想快速迭代开发	?
应用需要 2000-3000 以上的读写 QPS（更高也可以）	?
应用需要 TB 甚至 PB 级别数据存储	?
应用发展迅速，需要能快速水平扩展	?
应用要求存储的数据不丢失	?
应用需要 99.999%高可用	?
应用需要大量的地理位置查询、文本查询	?

如果上述有 1 个及以上的 Yes，就可以考虑使用 MongoDB。