1、概念：

SQL（Structured Query Language）数据库，指关系型数据库。主要代表：SQL Server、Oracle、MySQL、PostgreSQL。

NoSQL（Not Only SQL）泛指非关系型数据库。主要代表：MongoDB、Redis、CouchDB。

2、诞生原因：

随着互联网的不断发展，各类型的应用层出不穷，在这个云计算的时代，对技术提出了更多的需求，主要体现在这四个方面：

①低延迟的读写速度：应用快速的反应能极大地提升用户的满意度。

②海量的数据和流量：对于搜索这样大型应用而言，需要利用 PB 级别的数据和能应对百万级的流量。

③大规模集群的管理：系统管理员希望分布式应用能更简单的部署和管理。

④庞大运营成本的考量：IT 经理们希望在硬件成本、软件成本和人力成本能够有大幅度地降低。

目前世界上主流的存储系统大部分还是采用了关系型数据库，其主要有以下优点：

①事务处理——保持数据的一致性。

②由于以标准化为前提，数据更新的开销很小

③可以进行 join 等复杂查询

虽然关系型数据库已经在业界的数据存储方面占据了不可动摇的地位，但是由于其天生的几个限制，使其很难满足上面这几个要求：

①扩展困难：由于存在类似 join 这要多表查询机制，使得数据库在扩展方面很艰难

②读写慢：这种情况主要发生在数据量达到一定规模时由于关系型数据库的系统逻辑非常复杂，使得其非常容易发生死锁等的并发问题，所以导致其读写速度下滑非常严重。

③成本高：企业级数据库的 License 价格很惊人，并且随着系统的规模而不断上升。

④有限的支撑容量：现有关系型解决方案还无法支撑 Google 这样海量的数据存储。

2、NoSQL 优缺点：

优点：

①简单的扩展：典型例子是 Cassandra，由于其架构是类似于经典的 P2P，所以能通过轻松地添加新的节点来扩展这个集群。

②快速的读写：主要例子有 Redis，由于其逻辑简单，而且纯内存操作，使得其性能非常出色，单节点每秒可以处理超过 10 万次读写操作。

③低廉的成本：这是大多数分布式数据库共有的特点，因为主要都是开源软件，没有昂贵的 License 成本。

缺点：

①不提供对 SQL 的支持：如果不支持 SQL 这样的工业标准，将会对用户产生一定的学习和应用迁移成本。

②支持的特性不够丰富：现有产品所提供的功能都比较有限，大多数 NoSQL 数据库都不支持事务，也不像 MSSQL Server 和 Oracle 那样能提供各种附加功能，比如 BT 和报表等。

③现有产品的不够成熟：大多数产品都还处于初创期。

3、NoSQL 使用场景

①数据库表 schema 经常变化

②数据库表字段是复杂数据类型

③高并发数据库请求

④海量数据的分布式存储。

4、NoSQL 与 SQL 的区别：

SQL 数据库：

在使用之前需要定义表的一个模式

在表中存储相关联的数据

支持 join 多表查询

提供事务

使用一个强声明性语言查询

提供足够的支持，专业技能和工具

NoSQL 数据库：

将相关联的数据存储在类似 JSON 格式，名称-值

可以保存没有指定格式的数据

保证更新一个文档，但不是多个文档

提供出色的性能和可伸缩性

使用 JSON 数据对象查询

二、为什么用 nosql?

传统关系型数据在应付超大规模和高并发的场景下显得力不从心。nosql，不仅仅是 sql。泛指非关系型数据库。nosql 的产生就是为了应付解决超大规模数据集合多种数据类型带来的挑战。

nosql 数据库的种类繁多，但是一个共同的特点都是去掉关系数据库的关系特性。数据之间无关系，这样就非常容易扩展。也无形之间，在架构层面上带来了可扩展的能力。

多样灵活的数据模型。NoSQL 无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦

NoSQL 数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。

这得益于它的无关系性，数据库的结构简单。一般 MySQL 使用 Query Cache，每次表的更新 Cache 就失效，是一种大粒度的 Cache，在针对 web2.0 的交互频繁的应用，Cache 性能不高。而 NoSQL 的 Cache 是记录级的，是一种细粒度的 Cache，所以 NoSQL 在这个层面上来说就要性能高很多了。

RDBMS vs NoSQL

RDBMS

- 高度组织化结构化数据
- 结构化查询语言（SQL）
- 数据和关系都存储在单独的表中。
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

NoSQL

- 代表着不仅仅是 SQL
- 没有声明性查询语言
- 没有预定义的模式

-键 - 值对存储，列存储，文档存储，图形数据库

- 最终一致性，而非 ACID 属性
- 非结构化和不可预知的数据
- CAP 定理
- 高性能，高可用性和可伸缩性

工作中遇到的挑战：
1，高并发读写
Web2.0 网站，数据库并发负载非常高，往往达到每秒上万次的读写请求
2，高容量存储和高效存储
Web2.0 网站通常需要在后台数据库中存储海量数据，如何存储海量数据并进行高效的查询往往是一个挑战
3，高扩展性和高可用性
随着系统的用户量和访问量与日俱增，需要数据库能够很方便的进行扩展、维护

传统关系型数据库的瓶颈：
1，无法应对每秒上万次的读写请求，硬盘 IO 此时也将变为性能瓶颈
2，表中存储记录数量有限，横向可扩展能力有限，纵向数据可承受能力也是有限的，面对海量数据，势必涉及到分库分表，难以维护
大数据查询 SQL 效率极低，数据量到达一定程度时，查询时间会呈指数级别增长
3，难以横向扩展，无法简单地通过增加硬件、服务节点来提高系统性能
对于需要 24 小时不间断提供服务的网站来说，数据库升级、扩展将是一件十分麻烦的事，往往需要停机维护，数据迁移，为了避免服务间断，如果网站使用服务器集群，则根据集群策略，需要相应的考虑主从一致性、集群扩展性等一系列问题

NoSQL 数据库的优点：
1，海量数据下，读写性能优异
2，数据模型灵活
3，数据间无关系，易于扩展

NoSQL 数据库分类：
1，键值存储数据库
代表数据库：Redis
适用场景：会话信息，用户配置信息，购物车
2，列存储数据库
代表数据库：BigTable，Cassandra，HBase
适用场景：事件记录，内容管理，博客平台
不适合需要 ACID 事务的场合
3，文档型数据库
代表数据库：MongoD
适用场景：事件记录，内容管理，博客平台，网站分析，实时分析，电子商务应用
4，图数据库：可以使用图结构相关算法，比如最短路径寻址
代表数据库：Neo4j
适用场景：社交网络，推荐引擎，基于位置的服务

分片简介

分片是指将数据拆分，将其分散存放在不同的机器上的过程。有时也用分区（partitioning）来表示这个概念。

几乎所有数据库软件都能进行**手动分片（manual sharding）**。应用需要维护与若干不同数据库服务器的连接，每个连接还是完全独立的。应用程序管理不同服务器上不同数据的存储，还管理在合适的数据库上查询数据的工作。

Mongodb 支持**自动分片（autosharding）**，可以使数据库架构对应用程序不可见，也可以简化系统管理。Mongodb 自动处理数据在分片上的分布，也更容易添加和删除分片。

理解集群的组件

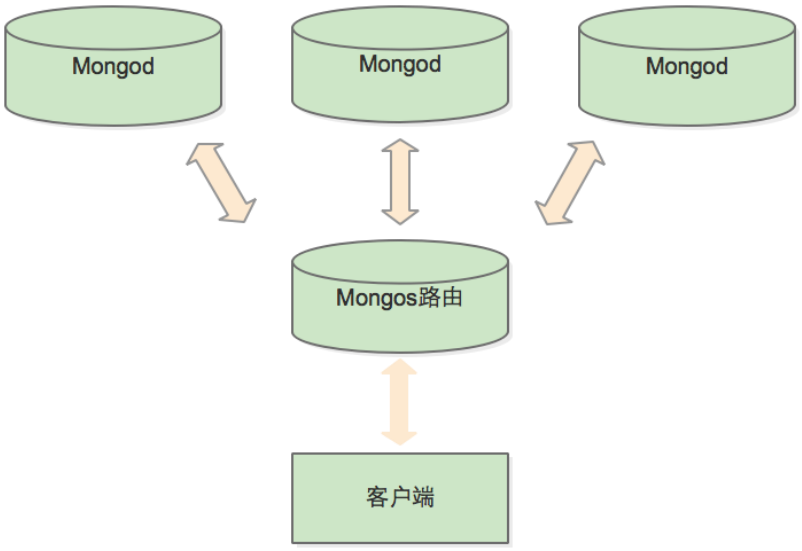
Mongodb 的分片机制允许你创建一个包含许多台机器（分片）的集群。将数据子集分散在集群中，每个分片维护着一个数据集合的子集。与单个服务器和副本集相比，使用集群架构可以使应用程序具有更大的数据处理能力。

复制是让多台服务器都拥有同样的数据副本，每一台服务器都是其它服务器的镜像，而每一个分片和其它分片拥有不同的数据子集。

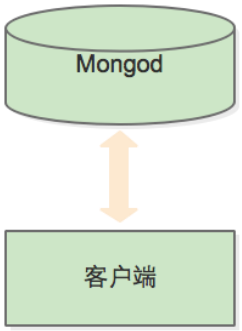
为了对应用程序隐藏数据库架构的细节，在分片之前要先执行 mongos 进行一次路由过程。这个路由服务器维护着一个“内容列表”，指明了每个分片包含什么数据内容。应用程序只需要连接到路由服务器，就可以像使用单机服务器一样进行正常的请求了。路由服务器知道哪些数据位于哪个分片，可以将请求转发给相应的分片。每个分片对请求的响应都会发送给路由服务器，路由服

务器将所有响应合并在一起，返回给应用程序。对应用程序来说，它只知道自己是连接到了一台单机 mongod 服务器。

使用分片的连接:



不使用分片连接:



可以看到，这次查询不得不访问所有 3 个分片，查询出所有数据。通常来说，如果没有在查询中使用片键，mongos 就不得不将查询发送到每个分片。包含片键的查询能够直接被发送到目标分片或者是集群分片的一个子集，这样的查询叫做定向查询（targeted query）。有些查询必须被发送到所有分片，这样的查询叫做分散--聚集查询（scatter-gather query）：mongos 将查询分散到所有分片上，然后将各个分片的查询结果聚集起来。

分片(sharding)是指将数据拆分，将其分散存在不同机器上的过程。
那么何时才考虑分片呢，出现如下问题时就该考虑使用分片：
1). 当数据量达到 T 级别的时候，我们的磁盘，内存不够用了
2). 单个 mongod 进程已经不能满足写数据的性能需要
3). 想将大量的数据放在内存中提高性能