# 背景

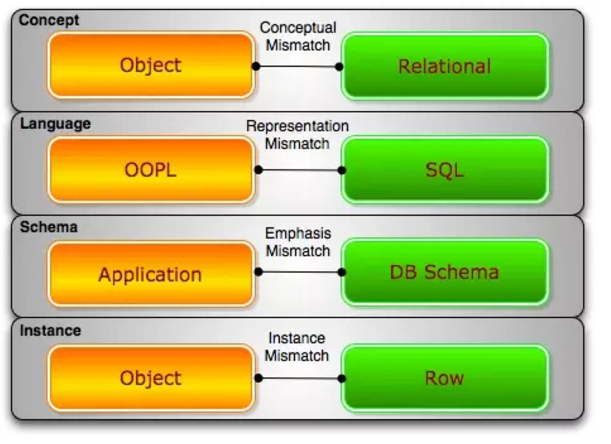
随着互联网网站的兴起，传统的关系型数据库在应对动态网站，特别是超大规模和高并发的纯动态网站已经显得力不从心，暴露了很多难以克服的问题。如商城网站中对商品数据频繁查询、对热搜商品的排名统计、订单超时问题、以及微信朋友圈（音频、视频）存储等相关使用传统的关系型数据库实现就显得非常复杂，虽然能够实现相应功能但是在性能上却不是那么乐观。NoSQL这个技术的出现，更好的解决了这些问题。

在过去几年，关系型数据库一直是数据持久化的唯一选择，数据工作者考虑的也只是在这些传统数据库中做筛选，比如SQL Server、Oracle或者是MySQL。甚至是做一些默认的选择，比如使用.NET的一般会选择SQL Server；使用Java的可能会偏向Oracle，Ruby是MySQL，Python则是PostgreSQL或MySQL等等。

原因很简单，过去很长一段时间内，关系数据库的健壮性已经在多数应用程序中得到证实。我们可以使用这些传统数据库良好的控制并发操作、事务等等。然而如果传统的关系型数据库一直这么可靠，那么还有NoSQL什么事？NoSQL之所以生存并得到发展，是因为它做到了传统关系型数据库做不到的事！

关系型数据库中存在的问题：

1、Impedance Mismatch



我们使用Python、Ruby、Java、.Net等语言编写应用程序，这些语言有一个共同的特性——面向对象。但是我们使用MySQL、PostgreSQL、Oracle以及SQL Server，这些数据库同样有一个共同的特性——关系型数据库。这里就牵扯到了“Impedance Mismatch”这个术语：存储结构是面向对象的，但是数据库却是关系的，所以在每次存储或者查询数据时，我们都需要做转换。类似Hibernate、Entity Framework这样的ORM框架确实可以简化这个过程，但是在对查询有高性能需求时，这些ORM框架就捉襟见肘了。

2、应用程序规模的变大

网络应用程序的规模日渐变大，我们需要储存更多的数据、服务更多的用户以及需求更多的计算能力。为了应对这种情形，我们需要不停的扩展。扩展分为两类：一种是纵向扩展，即购买更好的机器，更多的磁盘、更多的内存等等；另一种是横向扩展，即购买更多的机器组成集群。

在巨大的规模下，纵向扩展发挥的作用并不是很大。首先单机器性能提升需要巨额的开销并且有着性能的上限，在Google和Facebook这种规模下，永远不可能使用一台机器支撑所有的负载。鉴于这种情况，我们需要新的数据库，因为关系数据库并不能很好的运行在集群上。不错你也可能会去搭建关系数据库集群，但是他们使用的是共享存储，这并不是我们想要的类型。于是就有了以Google、Facebook、Amazon这些试图处理更多传输所引领的NoSQL纪元。

# 概述

NoSQL（Not Only SQL），意即不仅仅是SQL，泛指非关系型数据库。

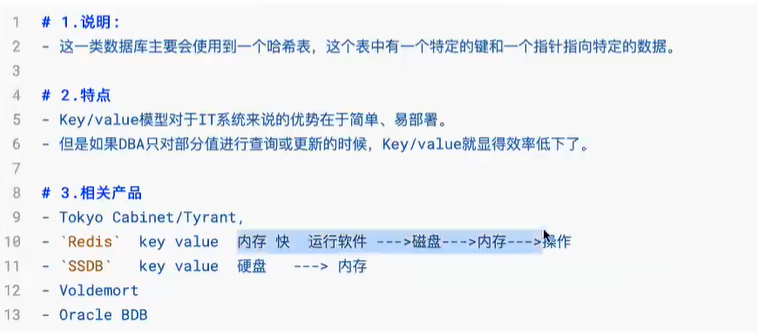
RDBMS：关系型数据库SQL语句

NoSQL：泛指非关系型数据库

# 分类

  对比传统关系型数据库，NoSQL有着更为复杂的分类——键值（Key-value）、面向文档（Document-Oriented）、列存储（Column-Family Databases）以及图数据库（Graph-Oriented Databases）。

## 键值（key-value）存储数据库



键值数据库就像在传统语言中使用的哈希表。你可以通过key来添加、查询或者删除数据，鉴于使用主键访问，所以会获得不错的性能及扩展性。

产品：Riak、Redis、Memcached、Amazon’s Dynamo、Project Voldemort

有谁在使用：GitHub（Riak）、BestBuy（Riak）、Twitter（Redis和Memcached）、StackOverFlow（Redis）、Instagram（Redis）、Youtube（Memcached）、Wikipedia（Memcached）

**适用的场景**

储存用户信息，比如会话、配置文件、参数、购物车等等。这些信息一般都和ID（键）挂钩，这种情景下键值数据库是个很好的选择。

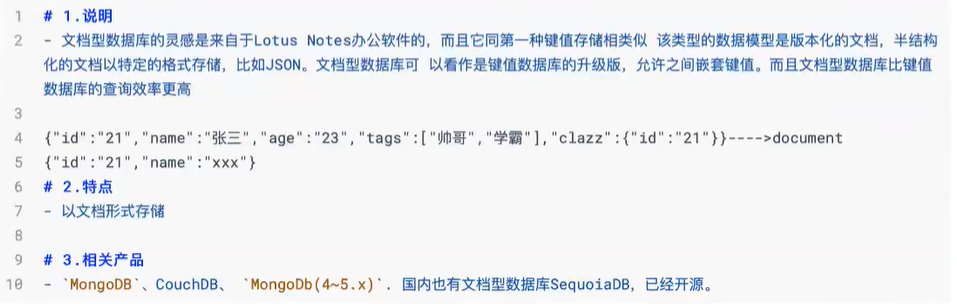
**不适用场景**

1、取代通过键查询，而是通过值来查询。Key-Value数据库中根本没有通过值查询的途径。

2、需要储存数据之间的关系。在Key-Value数据库中不能通过两个或以上的键来关联数据。

3、事务的支持。在Key-Value数据库中故障产生时不可以进行回滚。

## 文档型数据库



 面向文档数据库会将数据以文档的形式储存。每个文档都是自包含的数据单元，是一系列数据项的集合。每个数据项都有一个名称与对应的值，值既可以是简单的数据类型，如字符串、数字和日期等；也可以是复杂的类型，如有序列表和关联对象。数据存储的最小单位是文档，同一个表中存储的文档属性可以是不同的，数据可以使用XML、JSON或者JSONB等多种形式存储。

产品：MongoDB、CouchDB、RavenDB

有谁在使用：SAP（MongoDB）、Codecademy（MongoDB）、Foursquare（MongoDB）、NBC News（RavenDB）

**适用的场景**

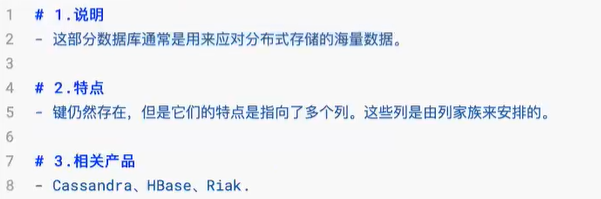
1、日志。企业环境下，每个应用程序都有不同的日志信息。Document-Oriented数据库并没有固定的模式，所以我们可以使用它储存不同的信息。

2、分析。鉴于它的弱模式结构，不改变模式下就可以储存不同的度量方法及添加新的度量。

**不适用场景**

在不同的文档上添加事务。Document-Oriented数据库并不支持文档间的事务，如果对这方面有需求则不应该选用这个解决方案。

## 列存储数据库



 列存储数据库将数据储存在列族（column family）中，一个列族存储经常被一起查询的相关数据。举个例子，如果我们有一个Person类，我们通常会一起查询他们的姓名和年龄而不是薪资。这种情况下，姓名和年龄就会被放入一个列族中，而薪资则在另一个列族中。

产品：Cassandra、HBase

有谁在使用：Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（HBase）、Yahoo!（HBase）

**适用的场景**

1、日志。因为我们可以将数据储存在不同的列中，每个应用程序可以将信息写入自己的列族中。

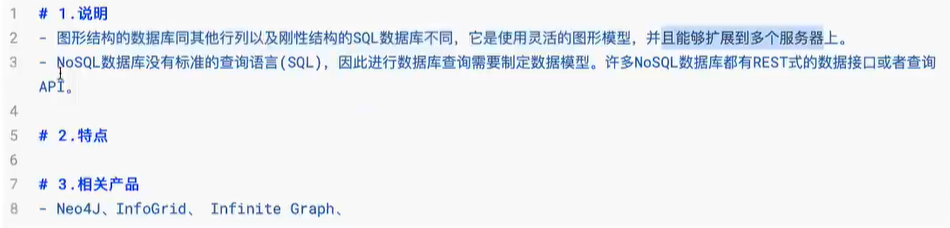
2、博客平台。我们储存每个信息到不同的列族中。举个例子，标签可以储存在一个，类别可以在一个，而文章则在另一个。

**不适用场景**

1、如果我们需要ACID事务。Vassandra就不支持事务。

2、原型设计。如果我们分析Cassandra的数据结构，我们就会发现结构是基于我们期望的数据查询方式而定。在模型设计之初，我们根本不可能去预测它的查询方式，而一旦查询方式改变，我们就必须重新设计列族。

## 图形（Graph）数据库



图数据库允许我们将数据以图的方式储存。实体会被作为顶点，而实体之间的关系则会被作为边。比如我们有三个实体，Steve Jobs、Apple和Next，则会有两个“Founded by”的边将Apple和Next连接到Steve Jobs。

产品：Neo4J、Infinite Graph、OrientDB

有谁在使用：Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

**适用的场景**

1、在一些关系性强的数据中

2、推荐引擎。如果我们将数据以图的形式表现，那么将会非常有益于推荐的制定不适用场景不适合的数据模型。图数据库的适用范围很小，因为很少有操作涉及到整个图。

# 对比

# 应用场景

