北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

NoSQL数据库

大数据处理技术

计算机学院

课程提纲

- NoSQL简介
- NoSQL VS. 关系数据库
- NoSQL的四大类型
- NoSQL的三大基石
- 从NoSQL到NewSQL数据库
- 文档数据库MongoDB

NoSQL简介

- NoSQL是对非关系型数据库的统称，具有以下几个特点：
  - 灵活的可扩展性
  - 灵活的数据模型
  - 与云计算紧密融合

概念演变

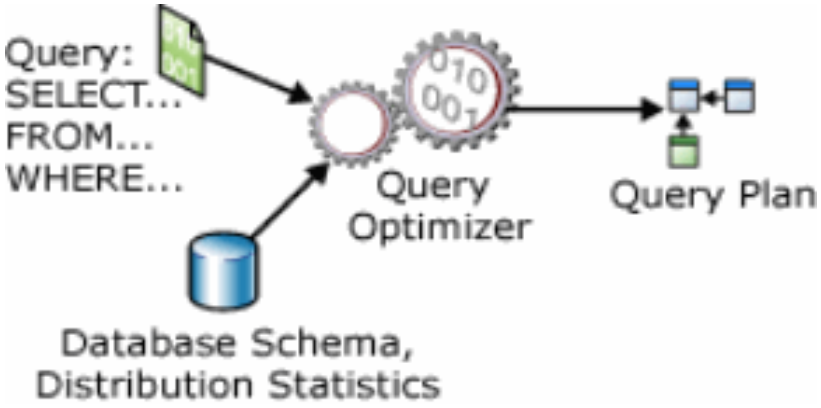
Not only SQL

最初表示“反SQL”运动
用新型的非关系数据库取代关系数据库

现在表示关系和非关系型数据库各有优缺点
彼此都无法互相取代

NoSQL兴起的原因

- 传统关系数据库一度占据商业数据库应用的主流位置
  - 完备的关系理论基础
  - 事务管理机制的支持
  - 高效的查询优化机制

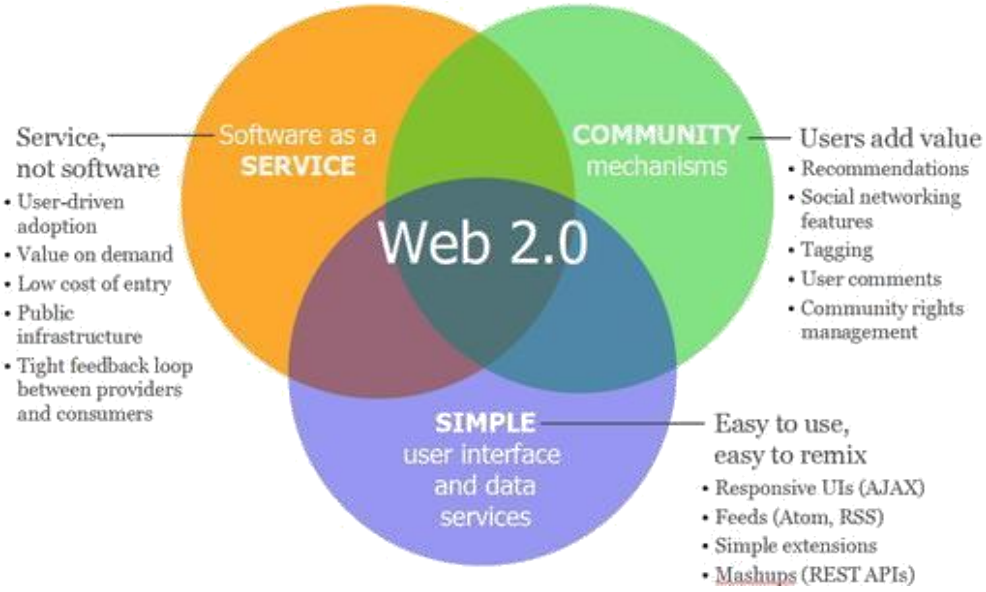


NoSQL兴起的原因

- 关系数据库无法满足Web 2.0和大数据时代的需求
  - 无法满足海量数据的管理需求
  - 无法满足数据高并发的需求
  - 无法满足高可扩展性和高可用性的需求

NoSQL兴起的原因

- 关系数据库中的关键特性包括完善的事务机制和高效的查询机制在Web 2.0时代没有发挥
  - Web 2.0系统通常不要求严格的数据库事务
  - Web 2.0并不要求严格的读写实时性
  - Web 2.0通常不包含大量复杂的SQL查询（去结构化，存储空间换取更好的查询性能）



NoSQL VS. 关系数据库

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，纵向扩展的空间也比较有限，性能会随着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，可以自由灵活定义并存储各种不同类型的数据
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借助于索引机制可以实现快速查询（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

NoSQL VS. 关系数据库

比较标准	RDBMS	NoSQL	备注
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 但是，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

NoSQL VS. 关系数据库

比较标准	RDBMS	NoSQL	备注
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

NoSQL VS. 关系数据库

关系数据库

- 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，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，无法较好支持海量数据存储，数据模型过于死板、无法较好支持Web2.0应用，事务机制影响了系统的整体性能等

NoSQL数据库

- 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Web2.0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，大都不能实现事务强一致性，很难实现数据完整性，技术不够成熟，缺乏专业团队的技术支持，维护较困难等

NoSQL VS. 关系数据库

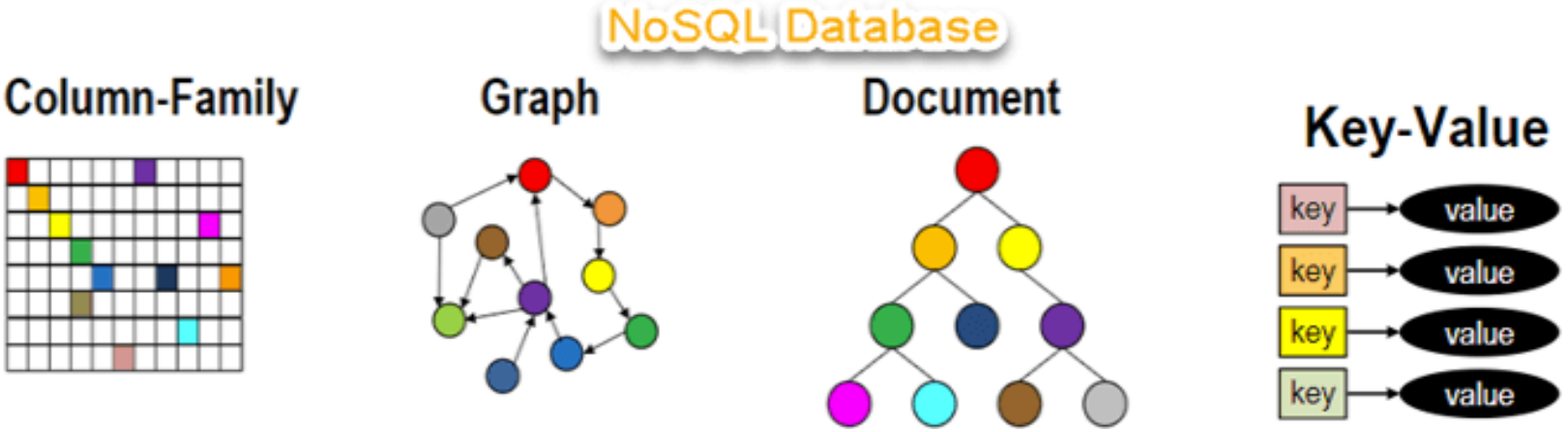
- 关系数据库和NoSQL数据库各有优缺点，彼此无法取代
- 关系数据库应用场景：电信、银行等领域的关键业务系统，需要保证强事务一致性
- NoSQL数据库应用场景：互联网企业、传统企业的非关键业务（比如数据分析）

NoSQL VS. 关系数据库

- 采用混合架构：使用不同类型的数据库来支撑电子商务应用
  - 对于“购物车”这种临时性数据，采用键值存储会更加高效
  - 当前的产品和订单信息则适合存放在关系数据库中
  - 大量的历史订单信息则适合保存在类似MongoDB的文档数据库中

NoSQL的四大类型

- 典型的NoSQL数据库通常包括
  - 键值数据库
  - 列族数据库
  - 文档数据库
  - 图形数据库



NoSQL的四大类型

Document Database	Graph Databases
 Couchbase MarkLogic mongoDB	 InfiniteGraph The Distributed Graph Database
Key-Value Databases	**Column Stores**
 redis amazon DynamoDB  AEROSPIKE riak	 accumulo HYPERTABLE INC  Cassandra APACHE HBASE

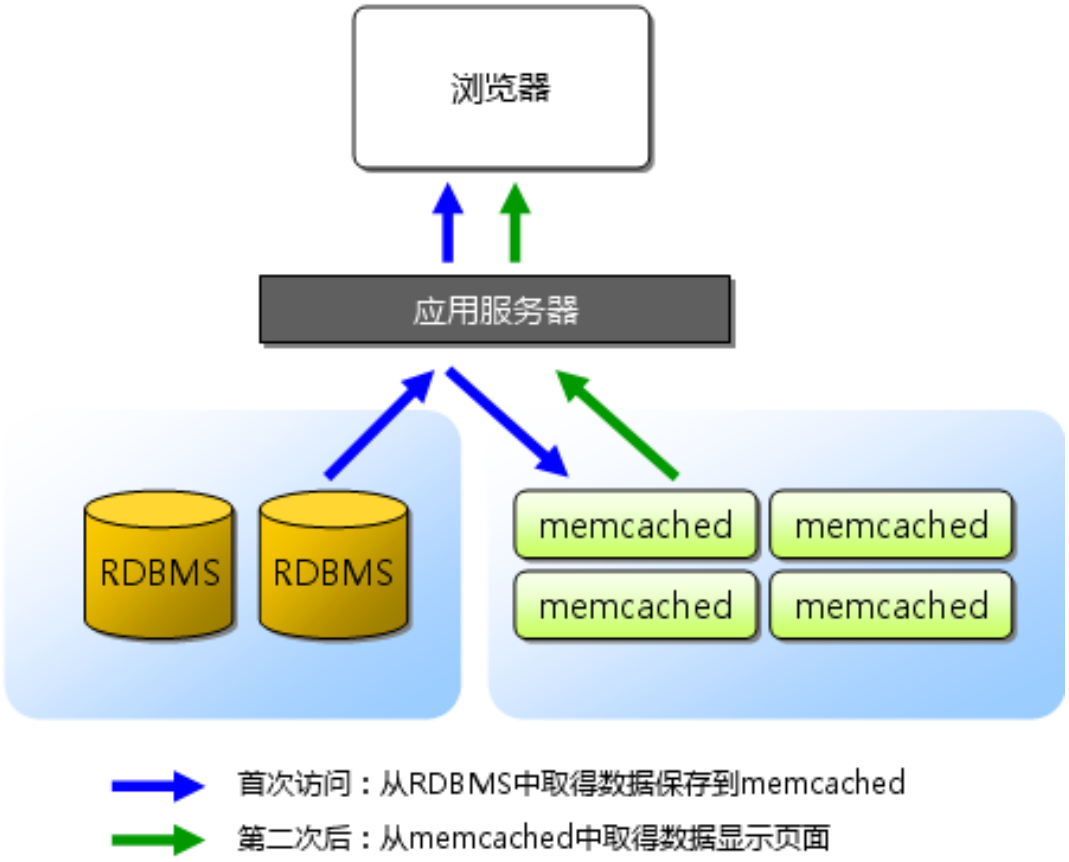
键值数据库

相关产品	Dynamo, Redis、Riak、Memcached, Voldemort, Berkeley DB等
数据模型	键/值对 键是一个字符串对象 值可以是任意类型的数据，比如整型、字符型、数组、列表、集合等
典型应用	拥有简单数据模型的应用，涉及频繁读写 内容缓存，比如会话、配置文件、参数、购物车等
优点	扩展性好，灵活性好，大量写操作时性能高
缺点	无法存储结构化信息，条件查询效率较低
不适用情形	不是通过键而是通过值来查：键值数据库没有通过值查询的途径 需要存储数据之间的关系：在键值数据库中，不能通过两个或两个以上的键来关联数据 需要事务的支持：在一些键值数据库中，产生故障时，不可以回滚
使用者	百度云数据库（Redis）、GitHub（Riak）、BestBuy（Riak）、Twitter（Redis和Memcached）、StackOverFlow（Redis）、Instagram （Redis）、Youtube（Memcached）、Wikipedia（Memcached）

键值数据库

Redis被称为“强化版的Memcached”

- 支持持久化
- 数据恢复
- 更多数据类型



键值数据库成为理想的缓冲层解决方案

列族数据库

相关产品	BigTable、HBase、Cassandra、Hypertable、GreenPlum等
数据模型	列族
典型应用	分布式数据存储与管理 数据在地理上分布于多个数据中心的应用程序 可以容忍副本中存在短期不一致情况的应用程序 拥有动态字段的应用程序 拥有潜在大量数据的应用程序，大到几百TB的数据
优点	查找速度快，可扩展性强，容易进行分布式扩展，复杂性低
缺点	功能较少，大都不支持强事务一致性
不适用情形	需要ACID事务支持的情形，Cassandra等产品就不适用
使用者	Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（Cassandra）、Yahoo!（HBase）

文档数据库

- “文档” 是一个数据记录，这个记录能够对包含的数据类型和内容进行“自我描述”
- 旨在将半结构化数据存储为文档，通常用XML、JSON 等文档格式来封装和编码数据

```
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:9000/hbase</value>
</property>
</configuration>
```

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
  ]
}
```

文档数据库

- 一个文档可以包含非常复杂的数据结构，如嵌套对象，且每个文档可以有完全不同的数据结构
- 每一条记录包含了所有的有关信息而没有任何外部的引用，这条记录就是“自包含”的，这使得记录很容易完成数据迁移
- 既可以根据键来构建索引，也可以根据内容构建索引

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
  ]
}
```

文档数据库

相关产品	MongoDB、CouchDB、Terrastore、MarkLogic、RavenDB
数据模型	键/值 值（value）是版本化的文档
典型应用	存储、索引并管理面向文档的数据或者类似的半结构化数据 比如，用于后台具有大量读写操作的网站、使用JSON数据结构的应用、使用嵌套结构等非规范化数据的应用程序
优点	性能好（高并发），复杂性低，数据结构灵活 提供嵌入式文档功能，将经常查询的数据存储在同一个文档中 既可以根据键来构建索引，也可以根据内容构建索引
缺点	缺乏统一的查询语法
不适用情形	在不同的文档上添加事务。文档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
使用者	百度云数据库（MongoDB）、SAP （MongoDB）、Codecademy （MongoDB）、Foursquare （MongoDB）、NBC News （RavenDB）

图形数据库

相关产品	Neo4J、Infinite Graph、GraphDB
数据模型	图结构
典型应用	专门用于处理具有高度相互关联关系的数据，比较适合于社交网络、模式识别、依赖分析、推荐系统以及路径寻找等问题
优点	灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱
缺点	复杂性高，只能支持一定的数据规模
使用者	Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

不同类型数据库比较

MySQL

功能较稳定强大

满足多样需求

MongoDB

数据模型较灵活

支持较多功能

HBase

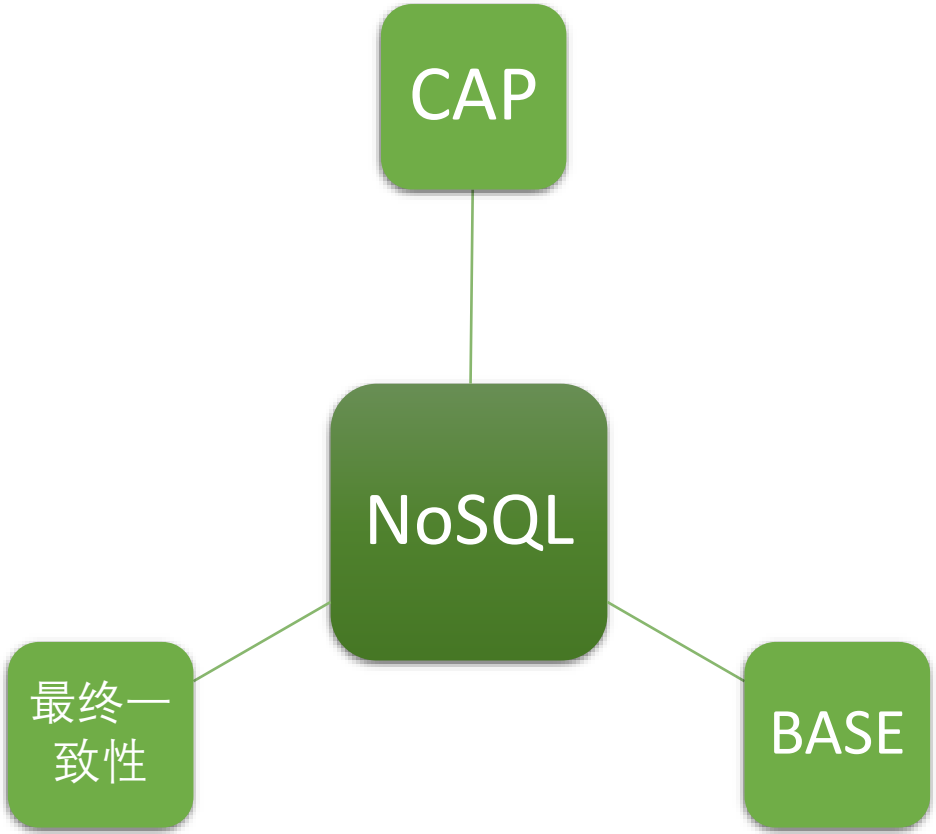
具有很好的扩展性

依赖Hadoop生态环境

Redis

模型相对较为简单，可提供随机数据存储，数据库伸缩性较好

NoSQL的三大基石

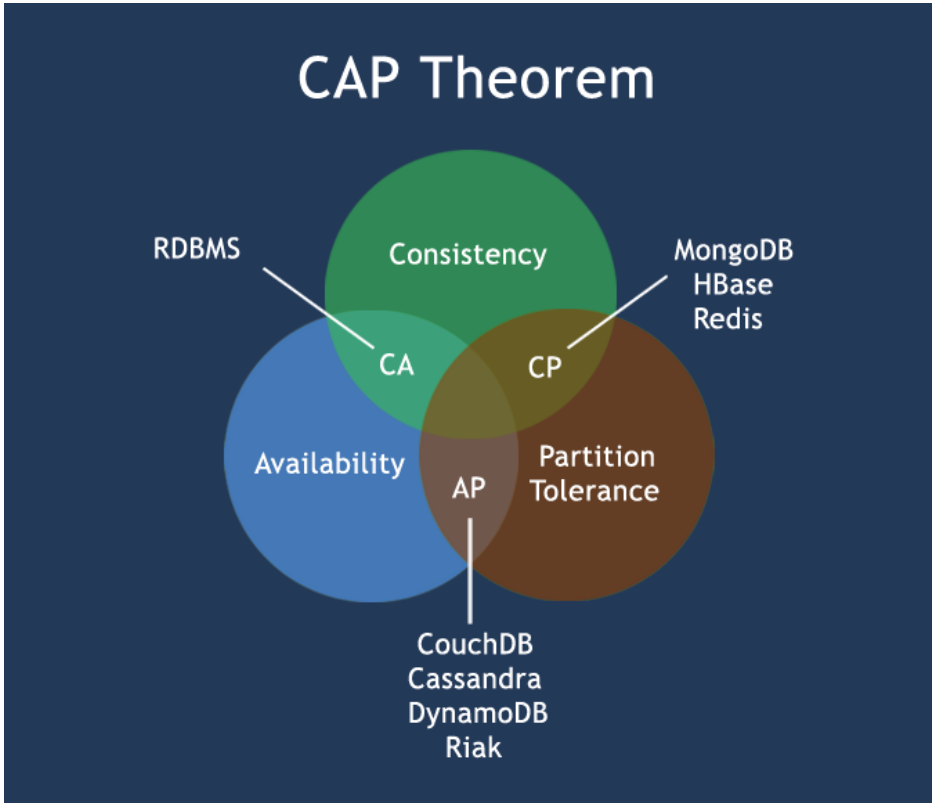


CAP理论

- C（Consistency）：一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据
- A:（Availability）：可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应；
- P（Tolerance of Network Partition）：分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作。

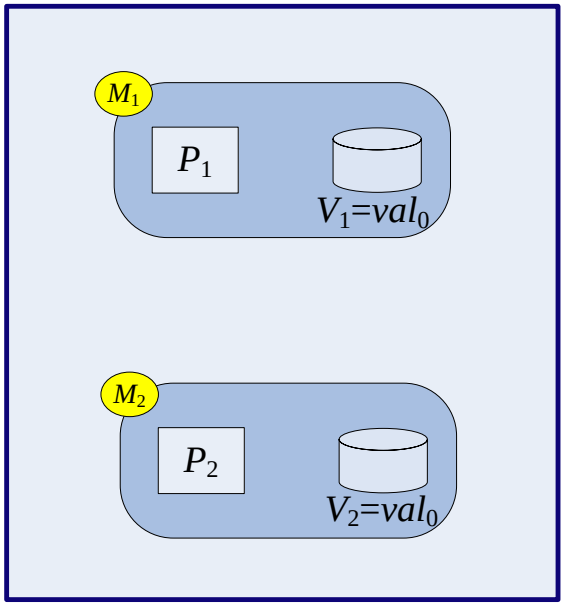
CAP理论

- CAP理论：一个分布式系统不可能同时满足一致性、可用性和分区容忍性这三个需求，最多只能同时满足其中两个



CAP理论

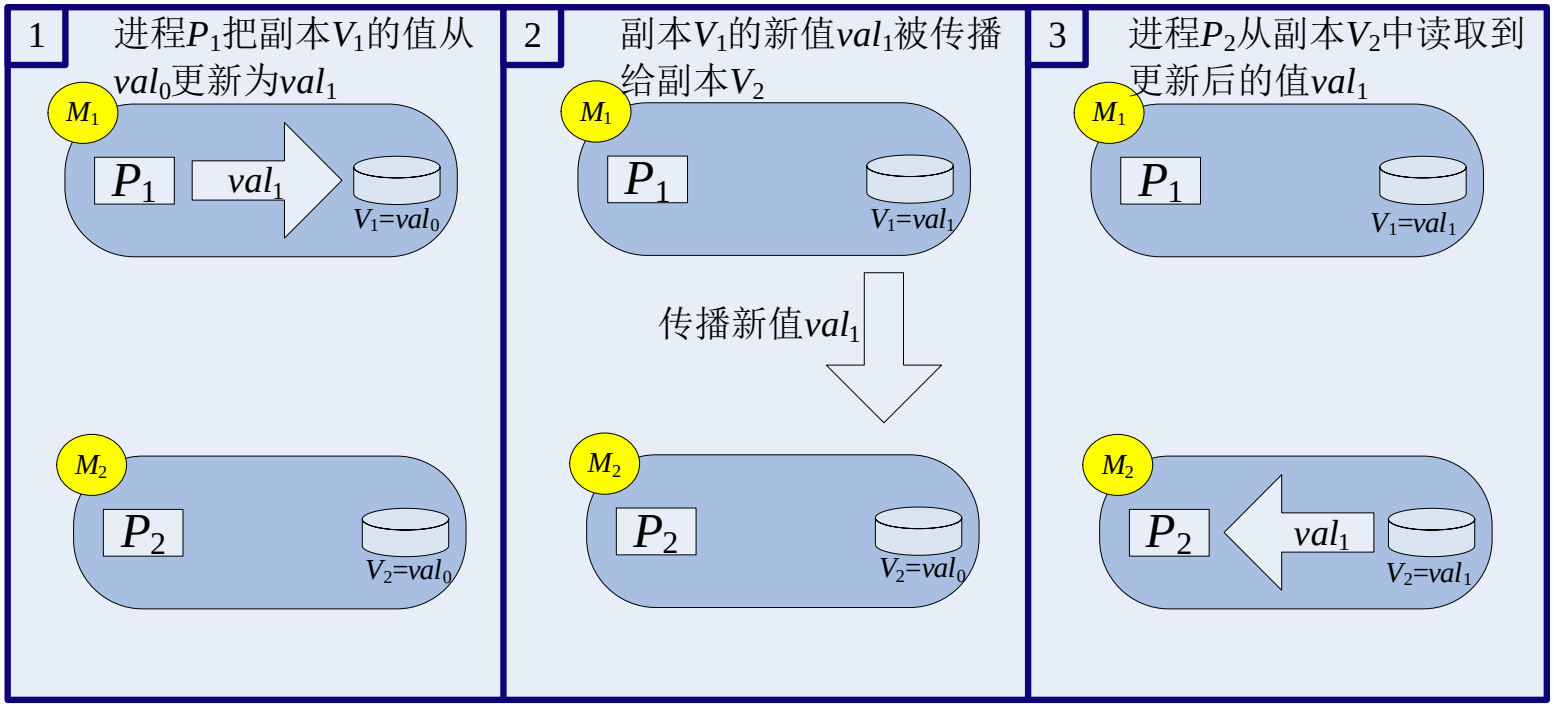
- 牺牲一致性来换取可用性的实例



（a）初始状态

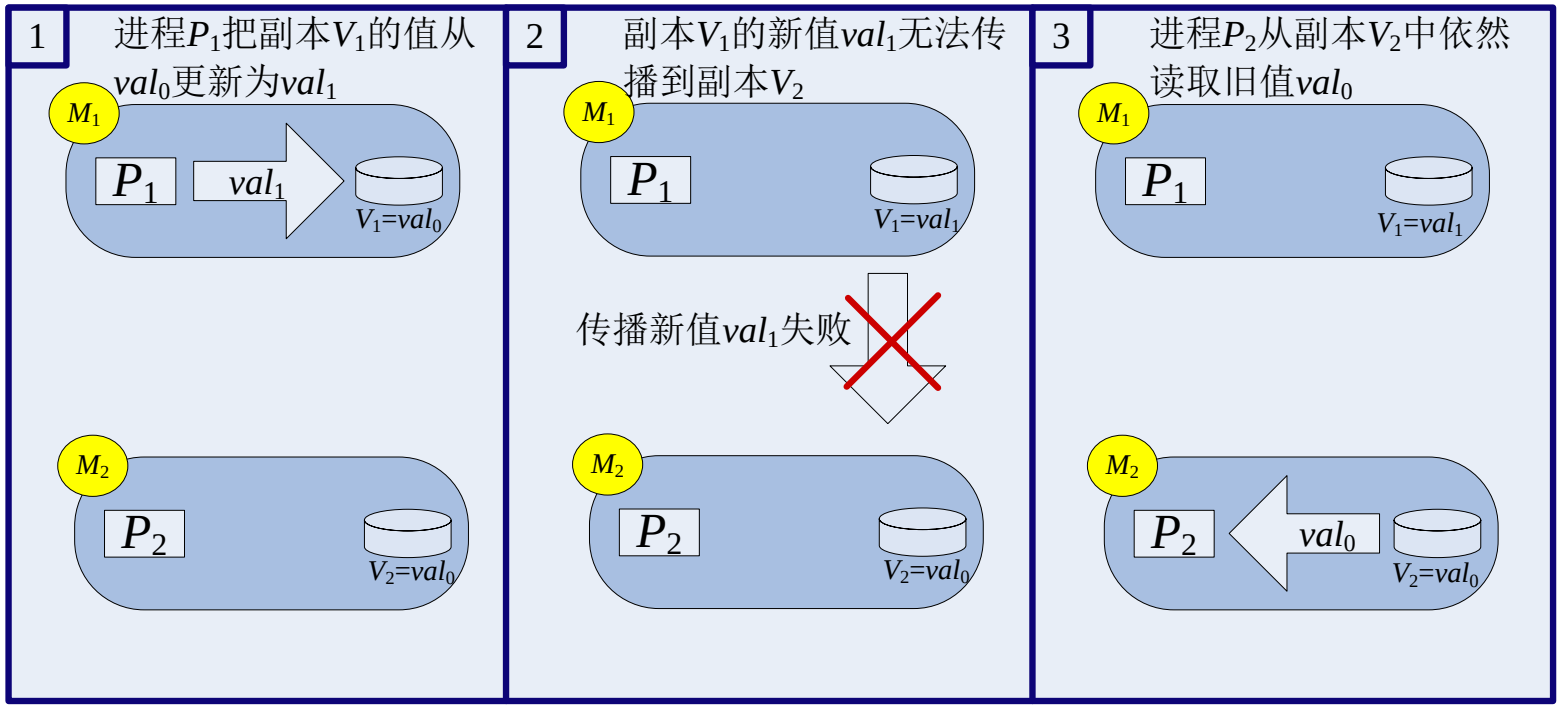
- 分布式环境下存在两个节点M1和M2
- 一个数据V的两个副本V1和V2分别保存在M1和M2上，两个副本值都是val_0
- 现有两进程P1和P2分别对两个副本进行操作，P1向副本V1中写入新值，P2读取副本V2的值

CAP理论



（b）正常执行过程

CAP理论



(c) 更新传播失败时的执行过程

CAP理论

处理CAP的问题时的选择：

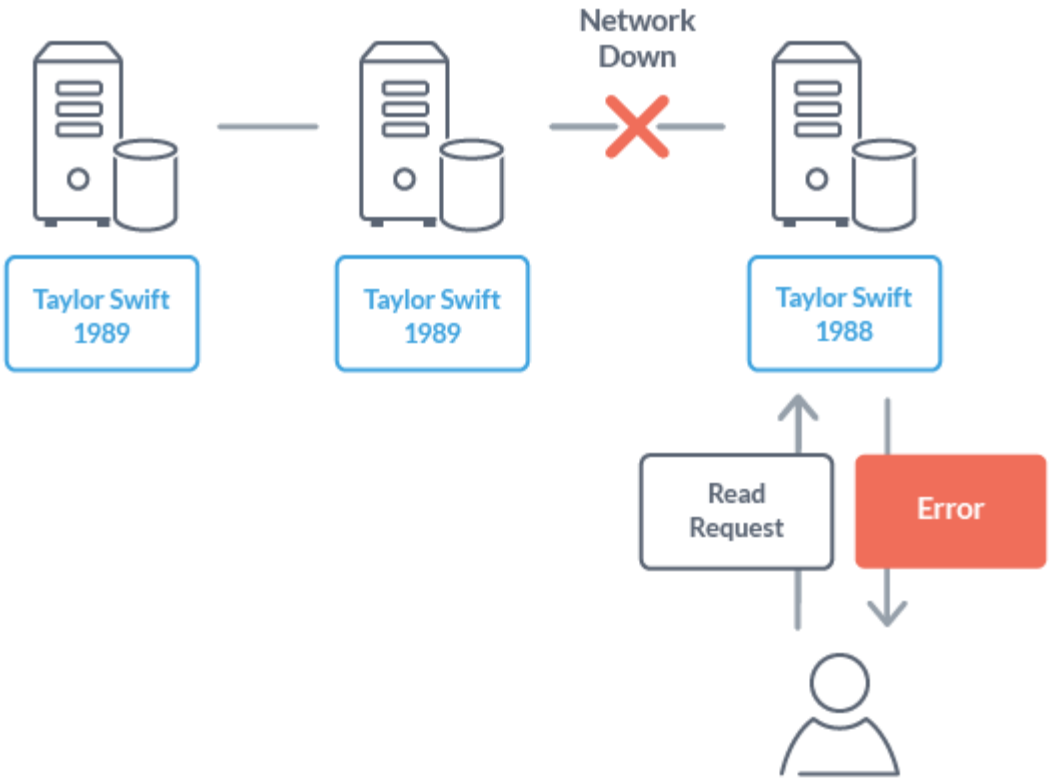
- **CA**：强调一致性（C）和可用性（A），放弃分区容忍性（P），最简单的做法是把所有与事务相关的内容都放到同一台机器上。很显然，这种做法会严重影响系统的可扩展性
- **CP**：强调一致性（C）和分区容忍性（P），放弃可用性（A），当出现网络分区的情况时，受影响的服务需要等待数据一致，因此在等待期间就无法对外提供服务
- **AP**：强调可用性（A）和分区容忍性（P），放弃一致性（C），允许系统返回不一致的数据

CAP Theorem

- Many NoSQL databases which feature scale-out, are distributed computer systems with multiple nodes, so partitioning tolerance is mandatory.
- As a result, a database on a distributed computer system can only achieve one of the following:
  1. **Guarantee Consistency and Partition-tolerance (CP),** and give up Availability (A)
  2. **Guarantee Availability and Partition-tolerance (AP),** and give up consistency (C)

CAP Theorem

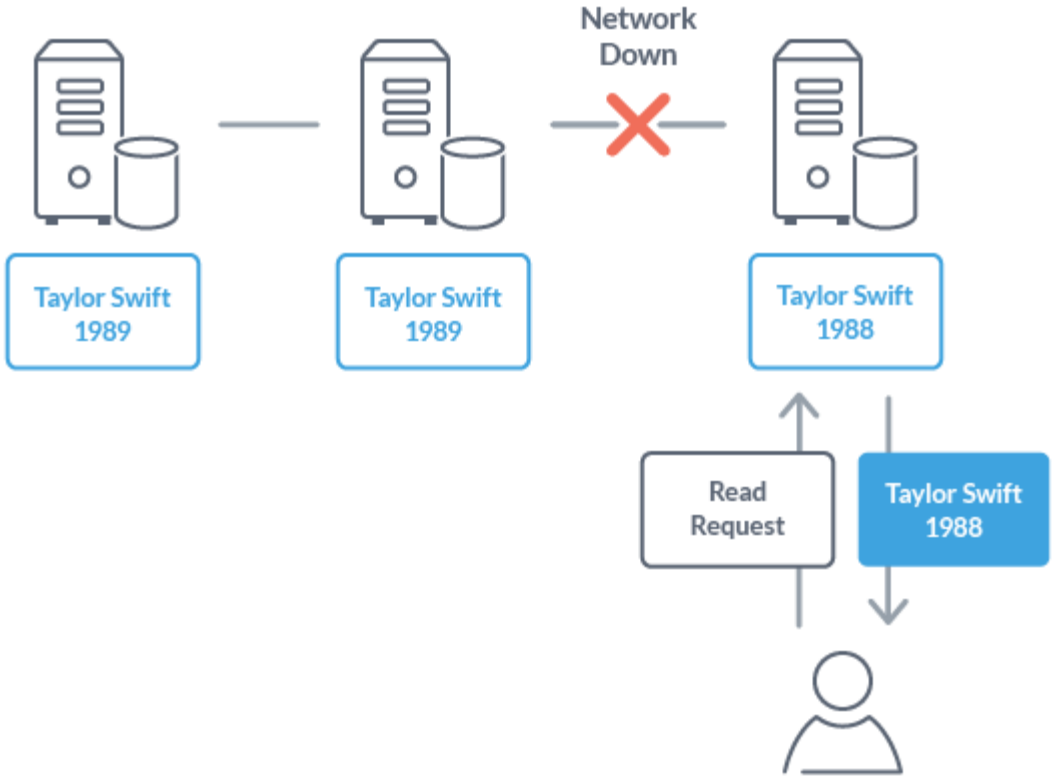
Guarantee Consistency and Partition-tolerance (CP), and give up Availability (A)



Typical examples of NoSQL databases that guarantee CP are MongoDB, HBase, Redis.

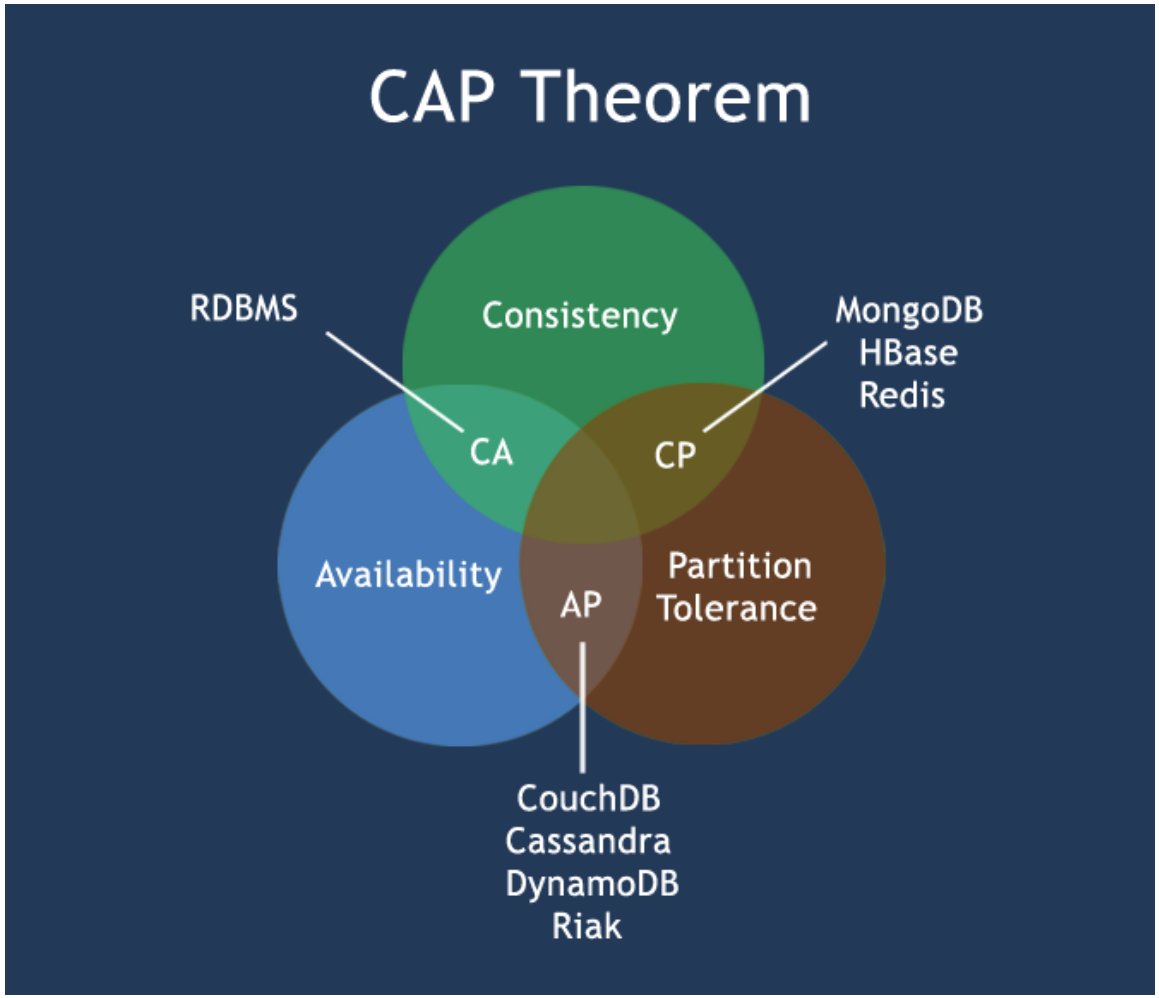
CAP Theorem

Guarantee Availability and Partition-tolerance (AP), and give up consistency (C)

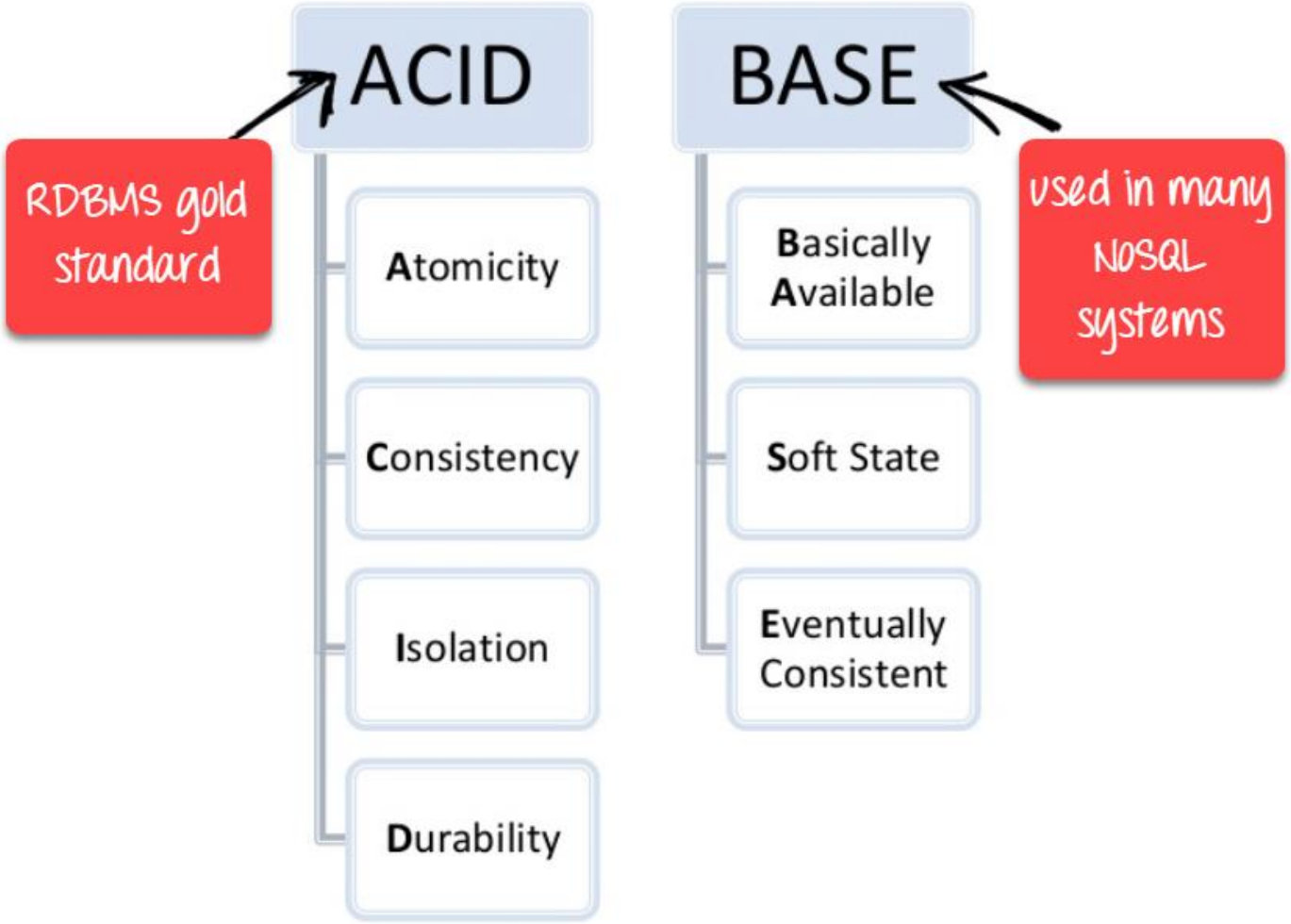


Typical examples of such a NoSQL database that guarantees APs include Cassandra and CouchDB.

CAP理论


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
AP
Partition
Tolerance
CouchDB
Cassandra
DynamoDB
Riak

BASE


ACID
RDBMS gold standard
Atomicity
Consistency
Isolation
Durability
BASE
used in many NOSQL systems
Basically Available
Soft State
Eventually Consistent

BASE

数据库事务的ACID特性

- A（Atomicity）：原子性，是指事务必须是原子工作单元，对于其数据修改，要么全都执行，要么全都不执行
- C（Consistency）：一致性，是指事务在完成时，必须使所有的数据都保持一致状态
- I（Isolation）：隔离性，是指由并发事务所做的修改必须与任何其它并发事务所做的修改隔离
- D（Durability）：持久性，是指事务完成之后，它对于系统的影响是永久性的，该修改即使出现致命的系统故障也将一直保持

BASE

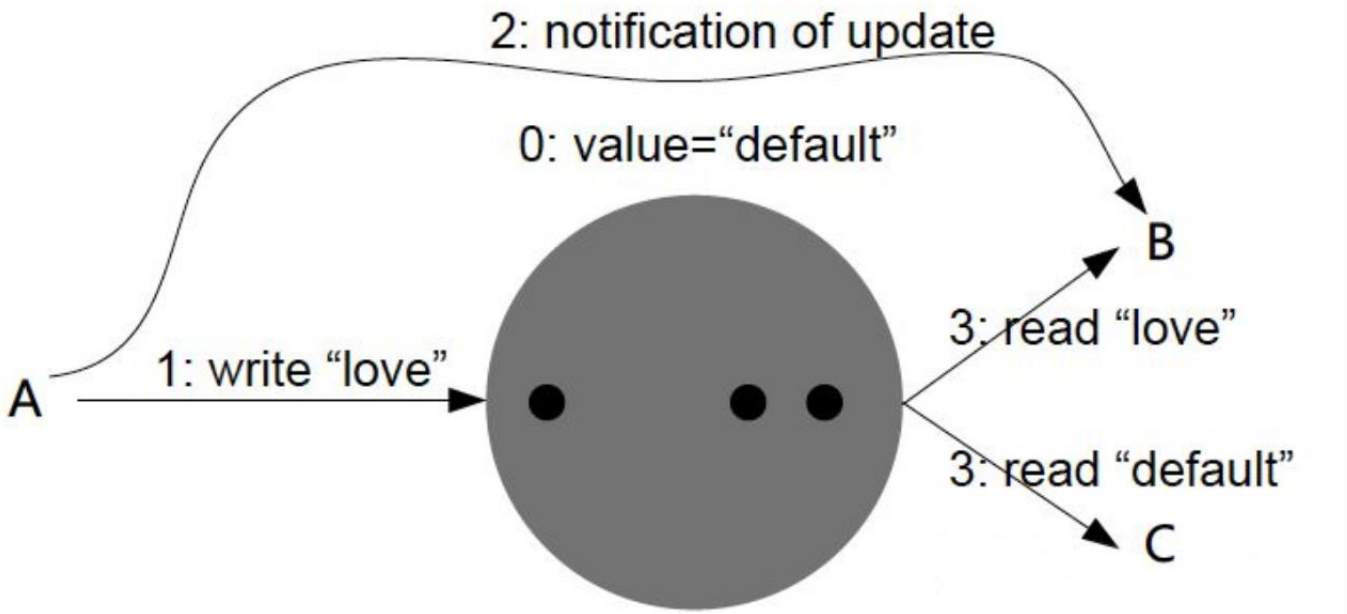
- 基本可用（Basically Available）：是指一个分布式系统的一部分发生问题变得不可用时，其他部分仍然可以正常使用，也就是允许分区失败的情形出现
- 软状态（Soft-state）：是与“硬状态（hard-state）”相对应的一种提法。数据库保存的数据是“硬状态”时，可以保证数据一致性，即保证数据一直是正确的。“软状态”是指状态可以有一段时间不同步，具有一定的滞后性

BASE

- 最终一致性（Eventual consistency）：一致性的类型包括强一致性和弱一致性，二者的主要区别在于高并发的数据访问操作下，后续操作是否能够获取最新的数据。对于强一致性而言，当执行完一次更新操作后，后续的其他读操作就可以保证读到更新后的最新数据；反之，如果不能保证后续访问读到的都是更新后的最新数据，那么就是弱一致性。而最终一致性只不过是弱一致性的一种特例，允许后续的访问操作可以暂时读不到更新后的数据，但是经过一段时间之后，必须最终读到更新后的数据

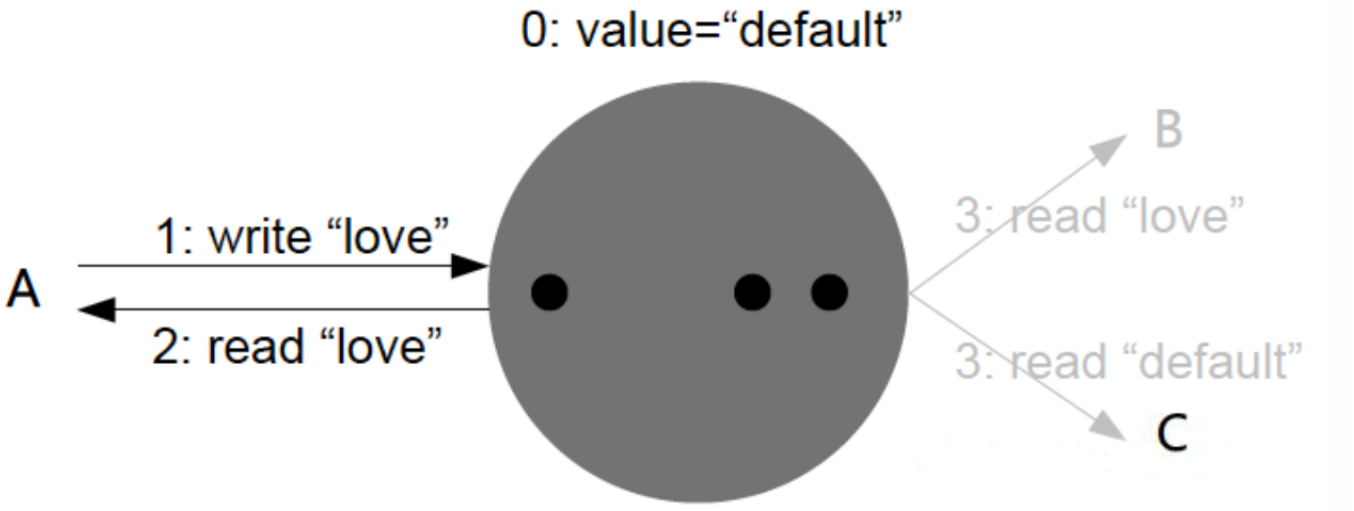
最终一致性

- 最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：
- **因果一致性 (Causal consistency)**：如果进程A通知进程B它已更新了一个数据项，那么进程B的后续访问将获得A写入的最新值。而与进程A无因果关系的进程C的访问，仍然遵守一般的最终一致性规则



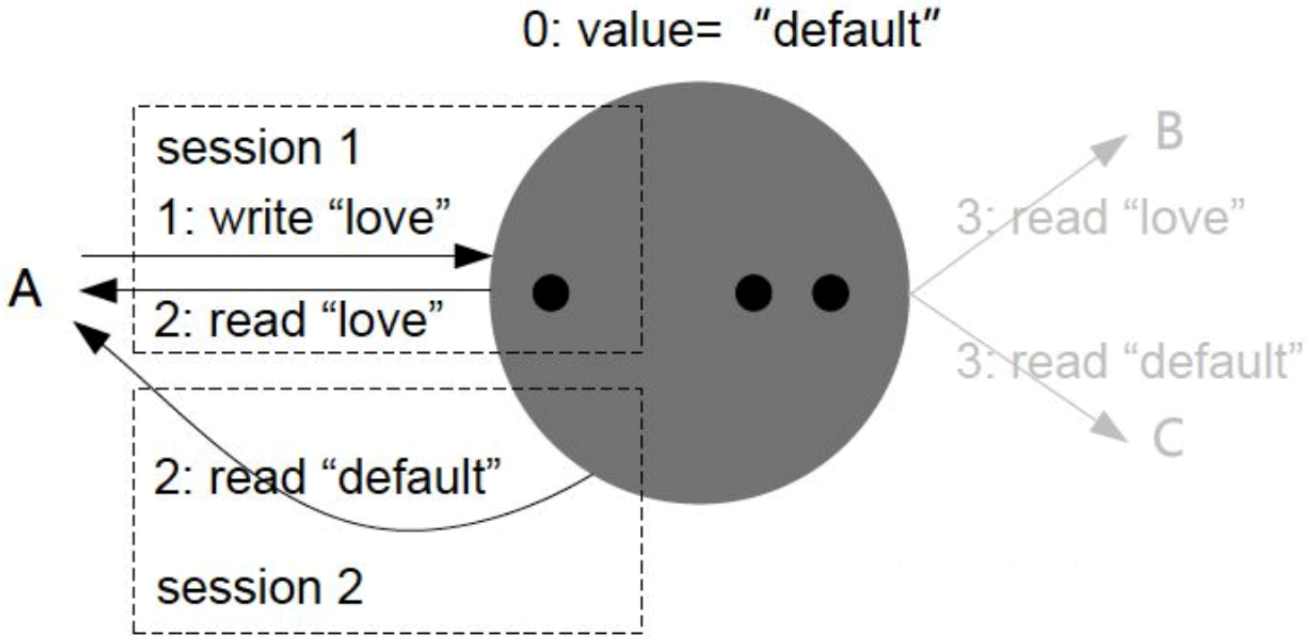
最终一致性

- 最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：
- **“读己之所写”一致性 (Read your writes)**：可以视为因果一致性的一个特例。当进程A自己执行一个更新操作之后，它自己总是可以访问到更新过的值，绝不会看到旧值



最终一致性

- 最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：
- **会话一致性 (Session consistency)**：系统能保证在同一个有效的会话中实现 “读己之所写” 的一致性，也就是说，执行更新操作之后，客户端能够在同一个会话中始终读取到该数据项的最新值



最终一致性

- 最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：
- **单调读一致性**：如果进程已经看到过数据对象的某个值，那么任何后续访问都不会返回在那个值之前的值
- **单调写一致性**：系统保证来自同一个进程的写操作顺序执行，对于多副本系统来说，保证写顺序的一致性（串行化），是很重要的

最终一致性

- 对于分布式数据系统如何实现一致性

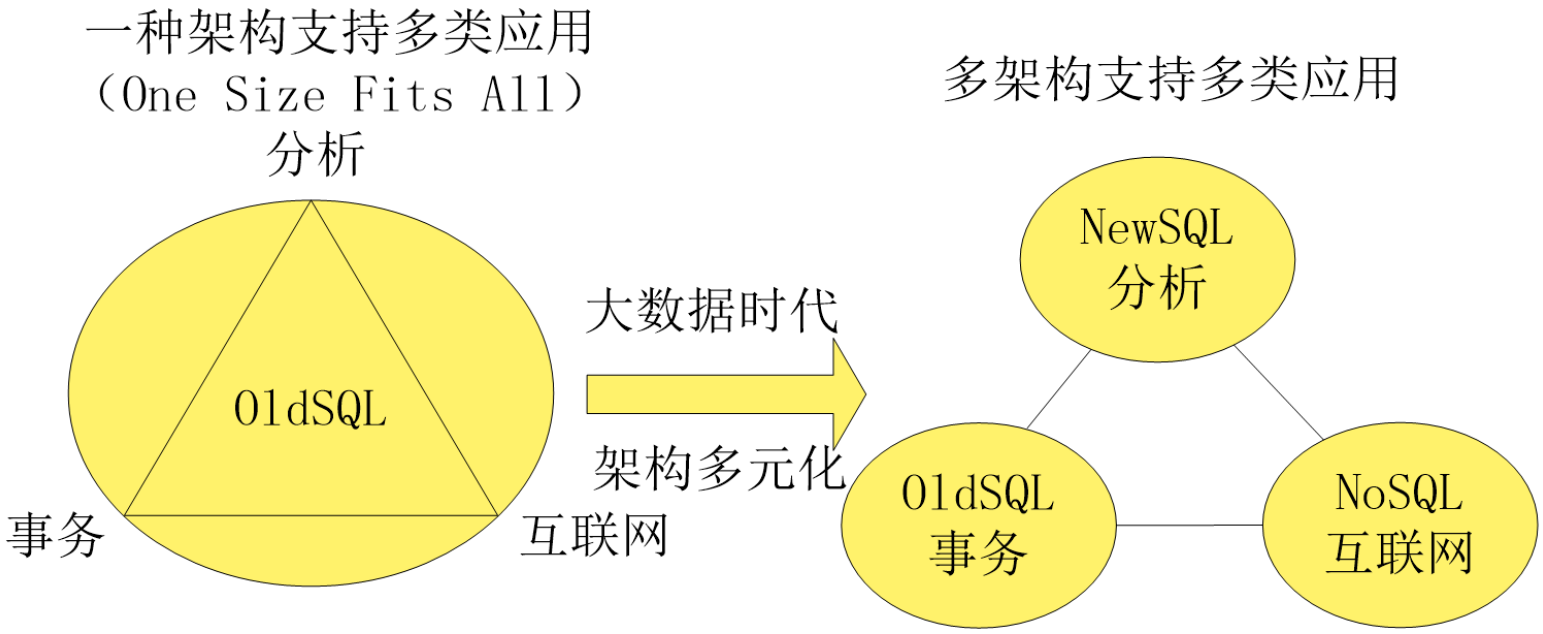
最终一致性

- 如果W+R>N，写的节点和读的节点重叠，则是强一致性。例如对于典型的一主一备同步复制的关系型数据库，N=2,W=2,R=1，则不管读的是主库还是备库的数据，都是一致的。一般设定是R＋W = N+1，这是保证强一致性的最小设定
- 如果W+R<=N，则是弱一致性。例如对于一主一备异步复制的关系型数据库，N=2,W=1,R=1，则如果读的是备库，就可能无法读取主库已经更新过的数据，所以是弱一致性。

最终一致性

- 对于分布式系统，为了保证高可用性，一般设置N>=3。不同的N,W,R组合，是在可用性和一致性之间取一个平衡，以适应不同的应用场景
- 如果N=W,R=1，任何一个写节点失效，都会导致写失败，因此可用性会降低，但是由于数据分布的N个节点是同步写入的，因此可以保证强一致性
- HBase是借助其底层的HDFS来实现其数据冗余备份的。HDFS采用的就是强一致性保证。在数据没有完全同步到N个节点前，写操作不会返回成功。也就是说它的W＝N，而读操作只需要读到一个值即可，也就是说它R＝1
- 而Cassandra等系统，通常都允许用户按需要设置N，R，W三个值，即可设置成W＋R<= N，也就是说允许用户在强一致性和最终一致性之间自由选择。而在用户选择了最终一致性，或者是W<N的强一致性时，则总会出现一段“各个节点数据不同步导致系统处理不一致的时间”。为了提供最终一致性的支持，这些系统会提供一些工具来使数据更新被最终同步到所有相关节点

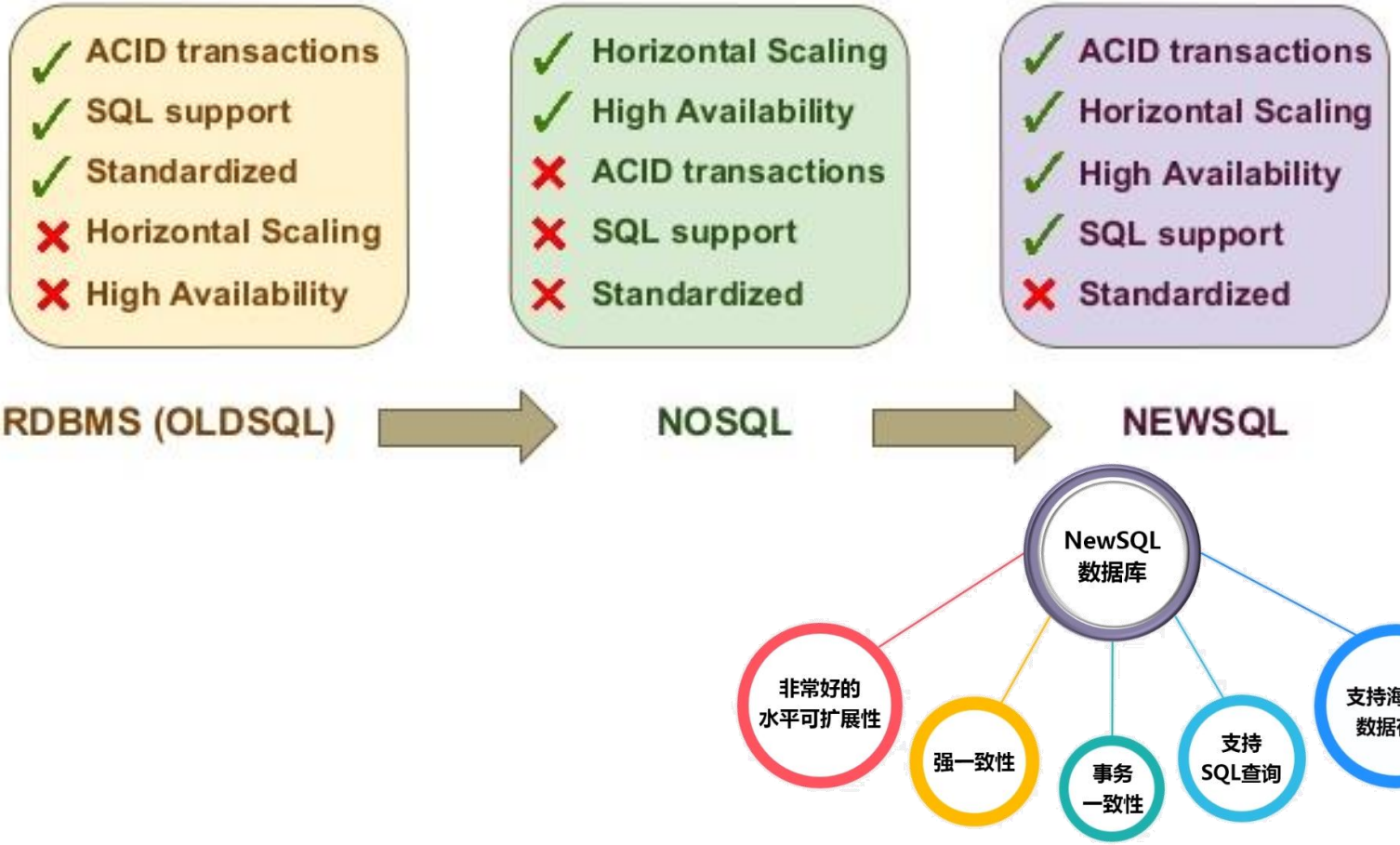
从NoSQL到NewSQL数据库



大数据引发数据处理架构变革

从NoSQL到NewSQL数据库

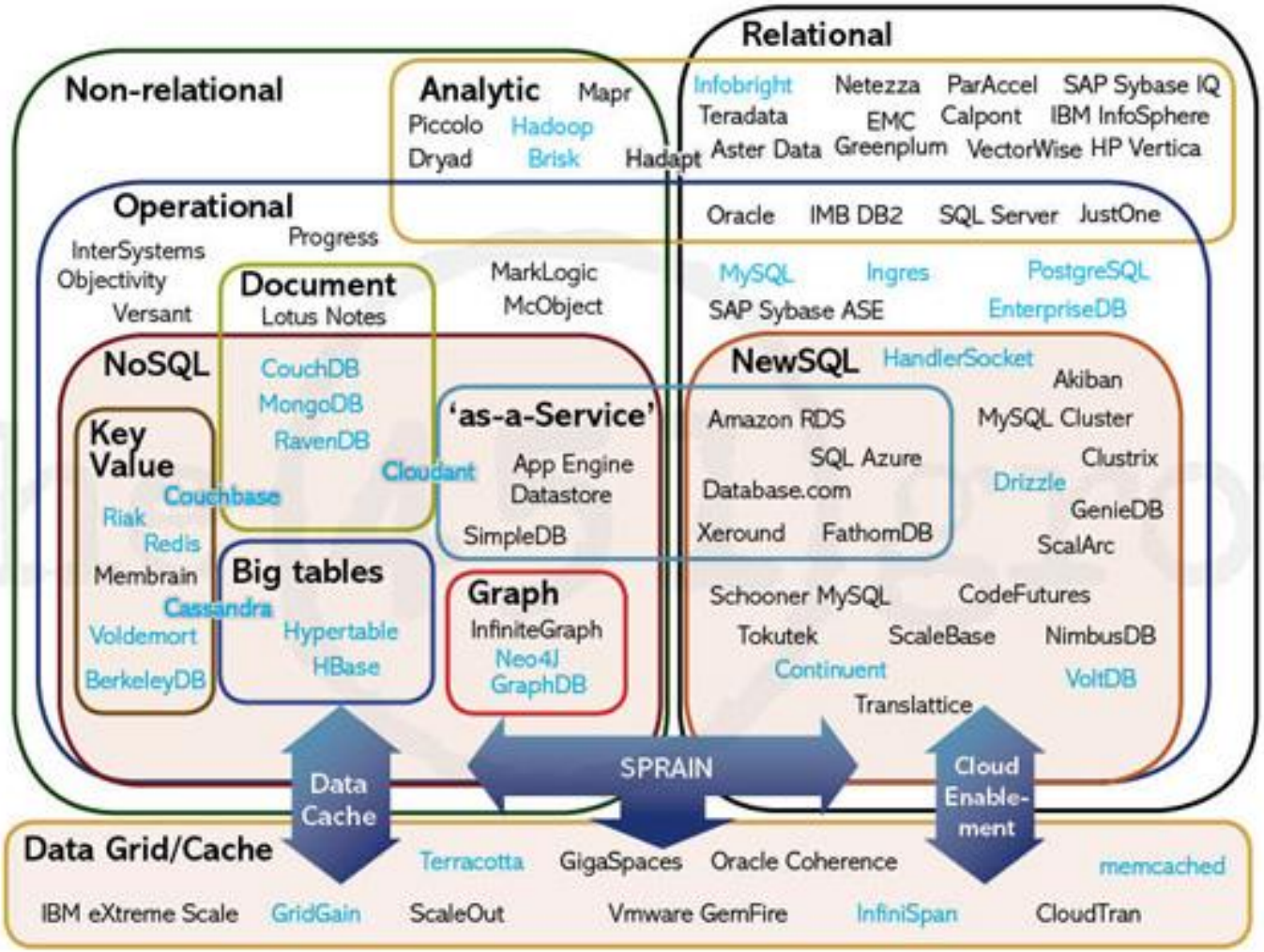
- NewSQL 是对各种新的可扩展/高性能数据库的简称，这类数据库不仅具有NoSQL对海量数据的存储管理能力，还保持了传统数据库支持ACID和SQL等特性。



NoSQL to NewSQL

Parameters for Comparison	RDBMS	NoSQL	NewSQL
SCHEMA	Relational Schema / TABLE	Schema-free	BOTH
SCALABILITY	Scalable reads	Scalable writes/reads Hirizontal Scalable	Scalable writes/reads Hirizontal Scalable
High Availability	Custom High-Availability	Auto High-Availability	Built-in High-Availability
STORAGE	On-Disk + Cache	On-Disk + Cache	On-Disk + Cache
Cloud Support	Not Fully	SUPPORTED	FULLY SUPPORTED
Query Complexity	LOW	High	Very High
ACID-CAP-BASE	ACID	CAP Through BASE	ACID
OLTP	Not Fully Supported	Not Supported	Fully Supported
Performance Overhead	Huge	Moderate	Minimal
Security Concerns	Very Very High	LOW	LOW
Examples	Oracle, MS SQL, MySQL, IBM DB2, PostgreSQL	MongoDB, Cassandra, Redis, HBase	Google Spanner, VoltDB
Use Cases	Financial , CRM, HR Applications	Big Data, IoT, Social Network Applications	Gaming, E-Commerce (High Availability), Telecom industry

从NoSQL到NewSQL数据库



文档数据库MongoDB

- 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功能最丰富，最像关系数据库的。

https://docs.mongodb.com/manual/

https://www.runoob.com/mongodb/mongodb-tutorial.html

文档数据库MongoDB

- MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
             }
```

文档数据库MongoDB

主要特点

- 面向文档存储，操作起来比较简单和容易
- 可以设置任何属性的索引来实现更快的查询
- 具有较好的水平可扩展性
- 支持丰富的查询表达式，可轻易查询文档中内嵌的对象及数组
- 可以实现替换完成的文档（数据）或者一些指定的数据字段
- MongoDB中的Map/Reduce主要是用来对数据进行批量处理和聚合操作
- 支持各种编程语言:RUBY，PYTHON，JAVA，C++，PHP，C#等语言
- MongoDB安装简单

文档数据库MongoDB

- MongoDB中基本的概念是文档、集合、数据库

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

* MongoDB 3.2 介绍了一个新的$lookup操作，这个操作可以提供一个类似于LEFT OUTER JOIN的操作

文档数据库MongoDB

数据库

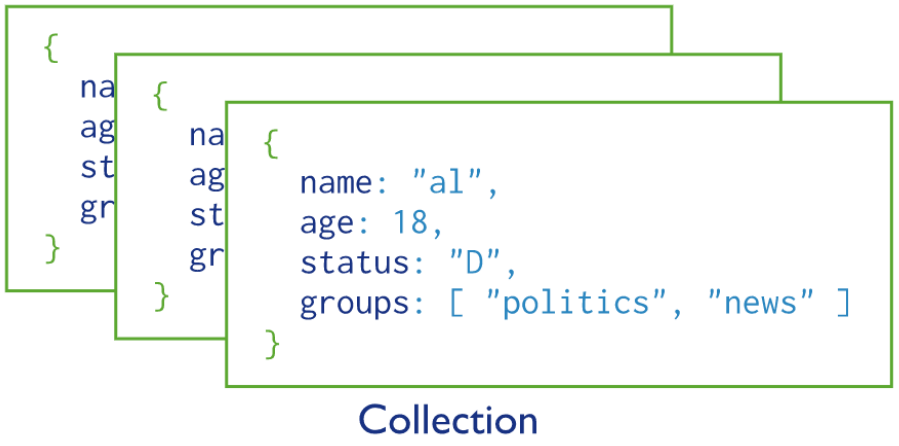
- 一个mongodb中可以建立多个数据库。
- MongoDB的默认数据库为"db"，该数据库存储在data目录中。
- MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中

文档

- 文档是一组键值(key-value)对(即 BSON)。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别

集合

- 文档组，类似于关系数据库中的table



文档数据库MongoDB

- 通过下图实例，我们也可以更直观的了解MongoDB中的一些概念：

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas

```
{
    "_id": ObjectId("5146bb52d8524270060001f3"),
    "age": 25,
    "city": "Los Angeles",
    "email": "mark@abc.com",
    "user_name": "Mark Hanks"
}
{
    "_id": ObjectId("5146bb52d8524270060001f2"),
    "age": 31,
    "city": "Dallas",
    "email": "richard@abc.com",
    "user_name": "Richard Peter"
}
```

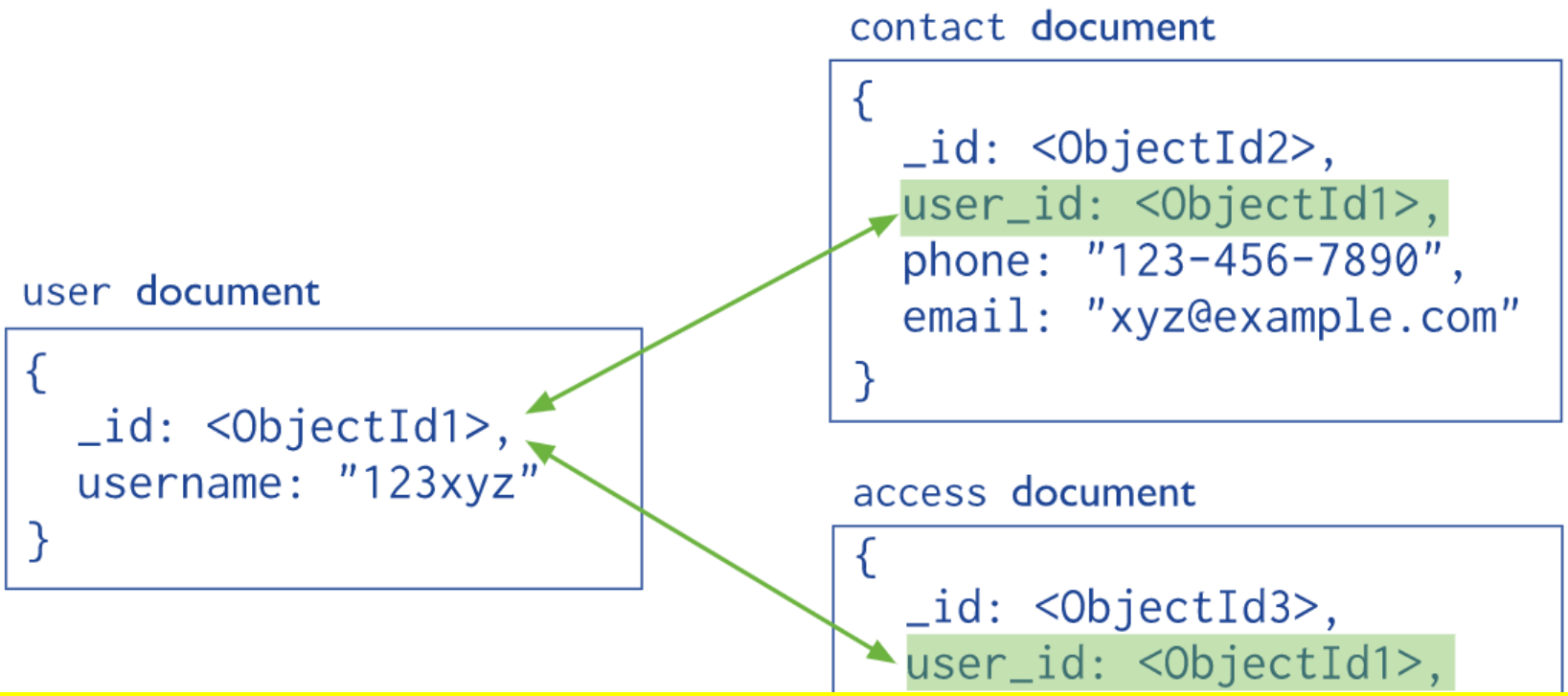
文档数据库MongoDB

- Embedded Data Model

In general, embedding provides **better performance** for read operations, as well as the ability to request and retrieve related data in a single database operation. Embedded data models make it possible to update related data in a single atomic write operation.

文档数据库MongoDB

- Normalized Data Model



References provides more flexibility than embedding. However, client-side applications must issue follow-up queries to resolve the references. In other words, normalized data models can require more round trips to the server.

文档数据库MongoDB

MongoDB 数据类型

数据类型	描述
String	字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。
Integer	整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。
Boolean	布尔值。用于存储布尔值（真/假）。
Double	双精度浮点值。用于存储浮点值。
Min/Max keys	将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。
Arrays	用于将数组或列表或多个值存储为一个键。
Timestamp	时间戳。记录文档修改或添加的具体时间。
Object	用于内嵌文档。
Null	用于创建空值。
Symbol	符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。
Date	日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。
Object ID	对象 ID。用于创建文档的 ID。
Binary Data	二进制数据。用于存储二进制数据。
Code	代码类型。用于在文档中存储 JavaScript 代码。
Regular expression	正则表达式类型。用于存储正则表达式。

文档数据库MongoDB

- MongoDB 中存储的文档必须有一个 _id 键。这个键的值可以是任何类型的，默认是个 ObjectId 对象
- ObjectId 包含 12 bytes，其中前4 bytes表示创建的时间戳
- 由于 ObjectId 中保存了创建的时间戳，所以不需要为文档保存时间戳字段，可以通过 getTimestamp 函数来获取文档的创建时间

```
ObjectId.getTimestamp()
```

安装MongoDB

- Windows 平台安装 MongoDB
- Linux平台安装MongoDB
- Mac OSX 平台安装 MongoDB

https://docs.mongodb.com/manual/installation/

文档数据库MongoDB

- MongoDB 创建数据库

```
use DATABASE_NAME
```

如果数据库不存在，则创建数据库，否则切换到指定数据库

```
> use runoob
switched to db runoob
> db
runoob
>
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
>
```

MongoDB 插入文档

- 在 MongoDB 中，你不需要创建集合。当你插入一些文档时，MongoDB 会自动创建集合。

```
db.COLLECTION_NAME.insert(document)
```

```
>db.col.insert({title: 'MongoDB 教程',
    description: 'MongoDB 是一个 Nosql 数据库',
    by: '菜鸟教程',
    url: 'http://www.runoob.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
})
```

MongoDB 插入文档

- 也可以将数据定义为一个变量，再插入变量

```
> document=({title: 'MongoDB 教程',
    description: 'MongoDB 是一个 Nosql 数据库',
    by: '菜鸟教程',
    url: 'http://www.runoob.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
});
```

```
> db.col.insert(document)
WriteResult({ "nInserted" : 1 })
>
```

MongoDB 插入文档

- 3.2 版本后还有以下几种语法可用于插入文档:
  - db.collection.insertOne()
  - db.collection.insertMany()

```
> var document = db.collection.insertOne({"a": 3})
> document
{
        "acknowledged" : true,
        "insertedId" : ObjectId("571a218011a82a1d94c02333")
}
> var res = db.collection.insertMany([{"b": 3}, {'c': 4}])
> res
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("571a22a911a82a1d94c02337"),
                ObjectId("571a22a911a82a1d94c02338")
        ]
}
```

MongoDB 更新文档

- MongoDB 使用 **update()** 方法来更新集合中的文档

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })   # 输出
信息
> db.col.find().pretty()
{
        "_id" : ObjectId("56064f89ade2f21f36b03136"),
        "title" : "MongoDB",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "菜鸟教程",
        "url" : "http://www.runoob.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
>
```

MongoDB 更新文档

- 以上语句只会修改第一条发现的文档，如果你要修改多条相同的文档，则需要设置 multi 参数为 true

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}},{multi:
true})
```

- 在3.2版本开始，MongoDB提供以下更新集合文档的方法：
  - db.collection.updateOne()
  - db.collection.updateMany()

MongoDB 更新文档

首先我们在test集合里插入测试数据

```
use test
db.test_collection.insert( [
{"name":"abc","age":"25","status":"zxc"},
{"name":"dec","age":"19","status":"qwe"},
{"name":"asd","age":"30","status":"nmn"},
] )
```

更新单个文档

```
> db.test_collection.updateOne({"name":"abc"},{$set:{"age":"28"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.test_collection.find()
{ "_id" : ObjectId("59c8ba673b92ae498a5716af"), "name" : "abc", "age" : "28", "status" : "zxc" }
{ "_id" : ObjectId("59c8ba673b92ae498a5716b0"), "name" : "dec", "age" : "19", "status" : "qwe" }
{ "_id" : ObjectId("59c8ba673b92ae498a5716b1"), "name" : "asd", "age" : "30", "status" : "nmn" }
>
```

更新多个文档

```
> db.test_collection.updateMany({"age":{$gt:"10"}},{$set:{"status":"xyz"}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.test_collection.find()
{ "_id" : ObjectId("59c8ba673b92ae498a5716af"), "name" : "abc", "age" : "28", "status" : "xyz" }
{ "_id" : ObjectId("59c8ba673b92ae498a5716b0"), "name" : "dec", "age" : "19", "status" : "xyz" }
{ "_id" : ObjectId("59c8ba673b92ae498a5716b1"), "name" : "asd", "age" : "30", "status" : "xyz" }
>
```

MongoDB 查询文档

- MongoDB 查询文档使用 find() 方法。
  - query ： 可选，使用查询操作符指定查询条件
  - projection ： 可选，使用投影操作符指定返回的键，默认省略，即返回文档中所有键值
  - 如果需要以易读的方式来读取数据，可以使用 pretty() 方法

```
db.collection.find(query, projection)
```

- 除了 find() 方法之外，还有一个 findOne() 方法，它只返回一个文档。

MongoDB 的条件查询

- MongoDB 与 RDBMS Where 语句比较

操作	格式	范例	RDBMS中的类似语句
等于	{<key>:<value>}	db.col.find({"by":"菜鸟教程"}).pretty()	where by = '菜鸟教程'
小于	{<key>:{$lt:<value>}}	db.col.find({"likes":{$lt:50}}).pretty()	where likes < 50
小于或等于	{<key>:{$lte:<value>}}	db.col.find({"likes":{$lte:50}}).pretty()	where likes <= 50
大于	{<key>:{$gt:<value>}}	db.col.find({"likes":{$gt:50}}).pretty()	where likes > 50
大于或等于	{<key>:{$gte:<value>}}	db.col.find({"likes":{$gte:50}}).pretty()	where likes >= 50
不等于	{<key>:{$ne:<value>}}	db.col.find({"likes":{$ne:50}}).pretty()	where likes != 50

MongoDB 的条件查询

- **MongoDB AND 条件**
- MongoDB 的 find() 方法可以传入多个键(key)，每个键(key)以逗号隔开

```
> db.col.find({"by":"菜鸟教程", "title":"MongoDB 教程"}).pretty()
{
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "菜鸟教程",
        "url" : "http://www.runoob.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

MongoDB 的条件查询

- **MongoDB OR 条件**
- MongoDB OR 条件语句需要使用关键字 **$or**

```
>db.col.find({$or:[{"by":"菜鸟教程"},{"title": "MongoDB 教程"}]}).pretty()
{
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "菜鸟教程",
        "url" : "http://www.runoob.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
>
```

MongoDB 的条件查询

- **AND 和 OR 联合使用**
- 'where likes>50 AND (by = '菜鸟教程' OR title = 'MongoDB 教程')'

```
>db.col.find({"likes": {$gt:50}, $or: [{"by": "菜鸟教程"},{"title": "MongoDB
教程"}]}).pretty()
{
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "菜鸟教程",
        "url" : "http://www.runoob.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

MongoDB 的条件查询

- **关于projection参数**

```
db.collection.find(query, projection)
```

- 若不指定 projection，则默认返回所有键，指定 projection 格式如下，有两种模式
  - _id 键默认返回，需要主动指定 _id:0 才会隐藏
  - 两种模式不可混用，只能全1或全0，除了在inclusion模式时可以指定_id为0

```
db.collection.find(query, {title: 1, by: 1}) // inclusion模式 指定返回的键，不返回其他键
db.collection.find(query, {title: 0, by: 0}) // exclusion模式 指定不返回的键,返回其他键
```

```
db.collection.find(query, {_id:0, title: 1, by: 1}) // 正确
```

MongoDB 删除文档

- MongoDB 中使用 drop() 方法来删除集合
- 查看集合使用`show collections/show tables`命令

```
db.collection.drop()
```

- MongoDB 中使用 dropDatabase() 方法来删除当前数据库，可以使用 db 命令查看当前数据库名
- `show dbs`查看所有数据库

```
db.dropDatabase()
```

MongoDB 删除集合和数据库

- remove() 方法已经过时了，现在官方推荐使用 deleteOne() 和 deleteMany() 方法

```
db.inventory.deleteMany({})
```

```
db.inventory.deleteMany({ status : "A" })
```

```
db.inventory.deleteOne( { status: "D" } )
```

MongoDB 排序

- 使用 sort() 方法对数据进行排序，可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列

```
>db.col.find({},{"title":1,_id:0}).sort({"likes":-1})
{ "title" : "PHP 教程" }
{ "title" : "Java 教程" }
{ "title" : "MongoDB 教程" }
>
```

```
{ "_id" : ObjectId("56066542ade2f21f36b0313a"), "title" : "PHP 教程", "d
escription" : "PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。", "b
y" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [ "php" ], "l
ikes" : 200 }
{ "_id" : ObjectId("56066549ade2f21f36b0313b"), "title" : "Java 教程",
"description" : "Java 是由Sun Microsystems公司于1995年5月推出的高级程序设
计语言。", "by" : "菜鸟教程", "url" : "http://www.runoob.com", "tags" : [
"java" ], "likes" : 150 }
{ "_id" : ObjectId("5606654fade2f21f36b0313c"), "title" : "MongoDB 教程"
, "description" : "MongoDB 是一个 Nosql 数据库", "by" : "菜鸟教程", "url"
: "http://www.runoob.com", "tags" : [ "mongodb" ], "likes" : 100 }
```

MongoDB 其他操作

- $exists 和 $in
- 正则表达式和模糊查询
- Limit() 方法读取指定数量的数据记录
- skip()方法来跳过指定数量的数据
- aggregate()方法执行聚合操作
- createIndex() 方法来创建索引

小结

- 本章介绍了NoSQL数据库的相关知识
- NoSQL数据库较好地满足了大数据时代的各种非结构化数据的存储需求，开始得到越来越广泛的应用。但是，需要指出的是，传统的关系数据库和NoSQL数据库各有所长，彼此都有各自的市场空间，不存在一方完全取代另一方的问题，在很长的一段时期内，二者都会共同存在，满足不同应用的差异化需求
- NoSQL数据库主要包括键值数据库、列族数据库、文档型数据库和图形数据库等四种类型，不同产品都有各自的应用场合。CAP、BASE和最终一致性是NoSQL数据库的三大理论基石，是理解NoSQL数据库的基础
- 介绍了融合传统关系数据库和NoSQL优点的NewSQL数据库
- 本章最后介绍了具有代表性的NoSQL数据库——文档数据库MongoDB