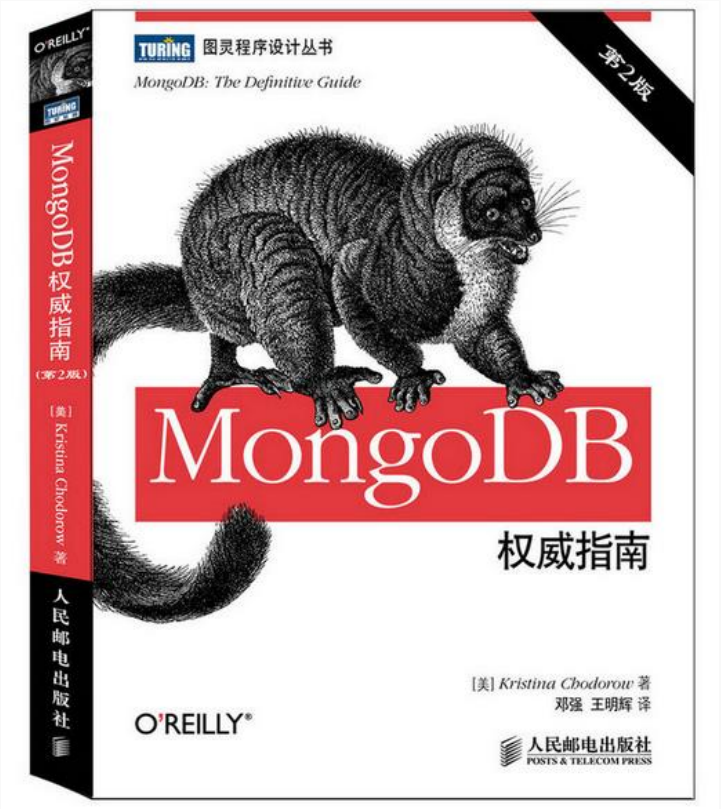
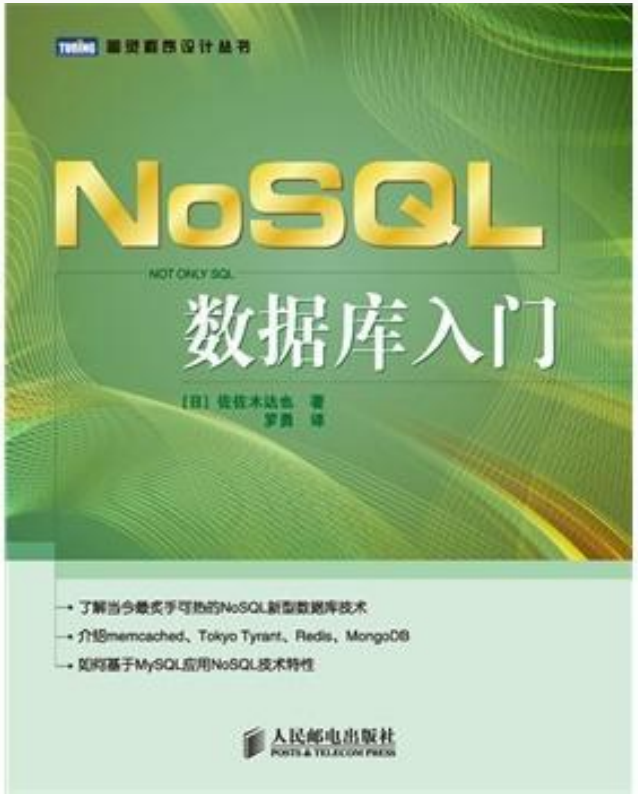
课程目标

- 了解NoSQL类数据库的原理和应用场景，能进行初步选型
- 熟练Redis、MongoDB等NoSQL产品
- 懂得如何使用上述NOSQL产品，并能用于解决实际应用问题
- 组建分布式集群并进行调试

参考书


O'REILLY
TURING 图灵程序设计丛书
MongoDB: The Definitive Guide
第2版
MongoDB权威指南
(第2版)
[美] Kristina Chodorow 著
人民邮电出版社
MongoDB
权威指南
[美] Kristina Chodorow 著
邓强 王明辉 译
O'REILLY
人民邮电出版社
POSTS & TELECOM PRESS


NoSQL
NOT ONLY SQL
数据库入门
了解当今最炙手可热的NoSQL新型数据库技术
介绍memcached、Tokyo Tyrant、Redis、MongoDB
如何基于MySQL应用NoSQL技术特性
人民邮电出版社
POSTS & TELECOM PRESS

NoSQL概述

李焕贞

河北师范大学软件学院

本章大纲

- NoSQL兴起的原因
- NoSQL与关系数据库的比较
- NoSQL的四大类型
- NoSQL的三大基石
- 从NoSQL到NewSQL数据库

NoSQL简介

概念演变

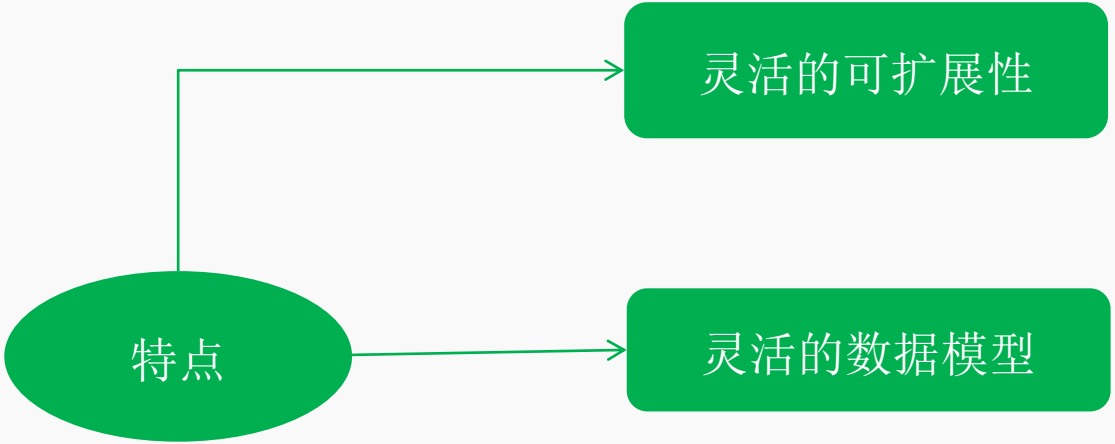
Not only SQL

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简介

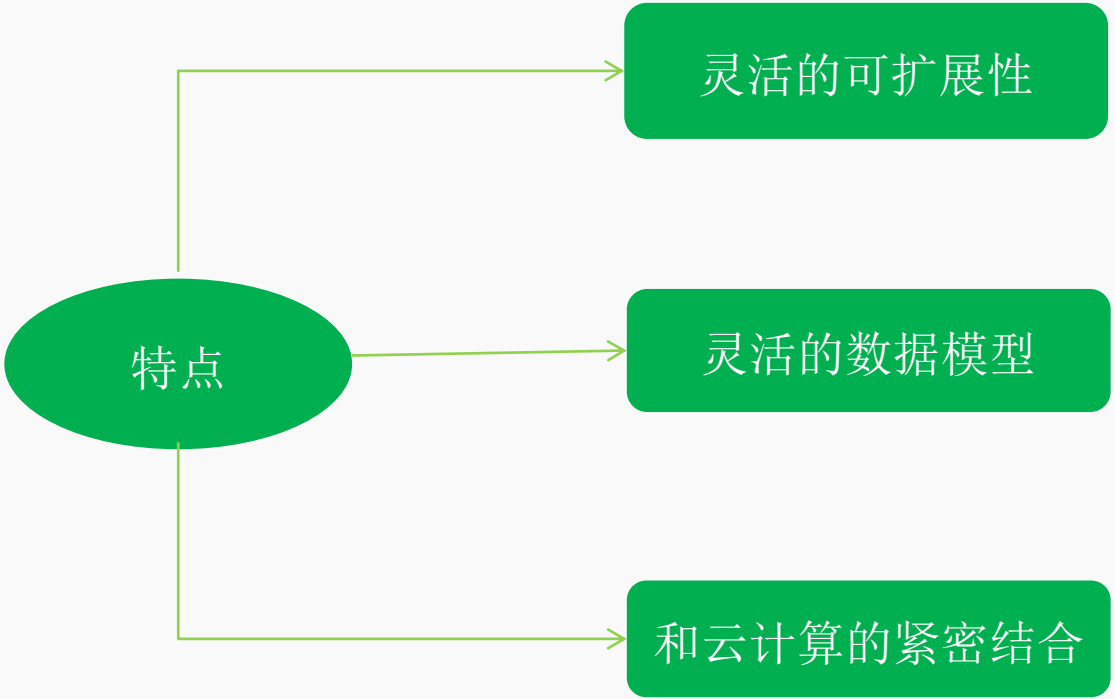
NoSQL数据库具有以下几个特点



NoSQL简介

NoSQL简介

NoSQL数据库具有以下几个特点



NoSQL简介

云计算

NoSQL数据库

传统的关系数据库

非常完备的关系理论基础

具有事务性机制的支持

高效的查询优化机制

传统的关系数据库性能上缺陷

无法满足海量数据的管理需求

NoSQL简介

无法满足海量数据的管理需求

到了Web2.0时代以后，数据的产生速度非常快

传统的关系数据库性能上缺陷

无法满足海量数据的管理需求

无法满足高并发需求

NoSQL简介

无法满足高并发的需求

动态数据是没有办法提前生成一个静态网页让用户来访问，只能实时地根据用户的请求来实时的生成数据

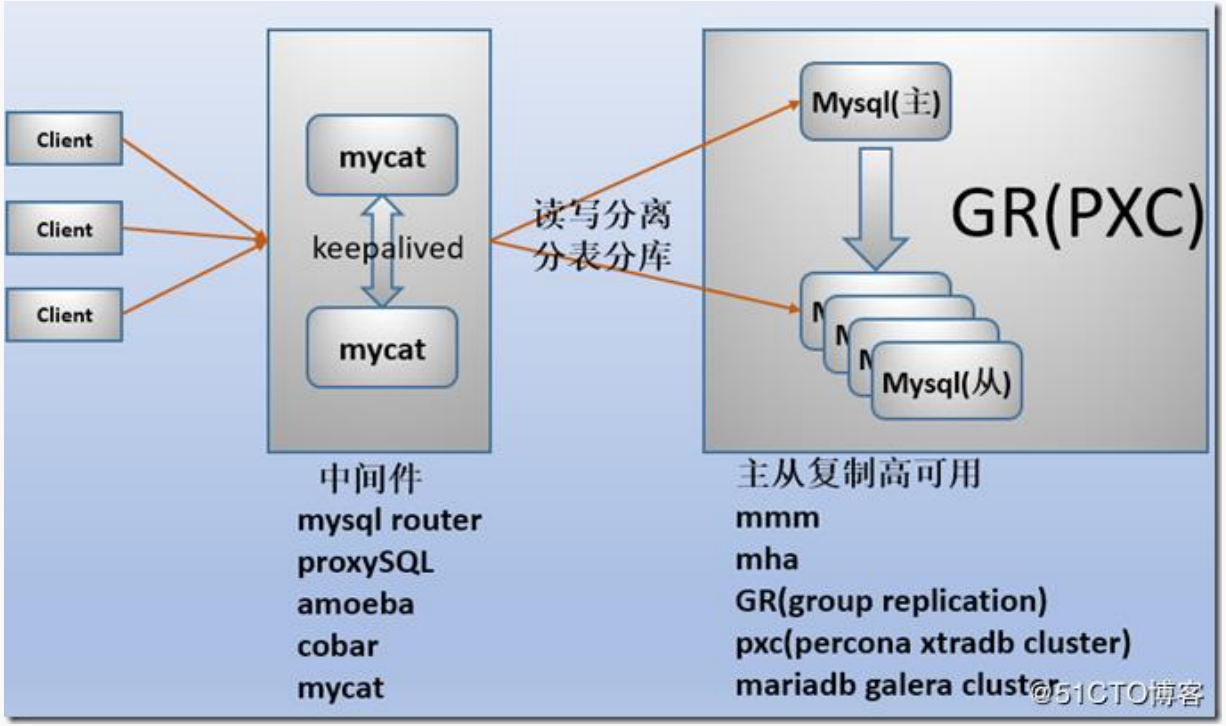
这种实时生成的数据，对数据库的负载非常高

基本上用关系数据库是没有办法满足这种高并发需求的

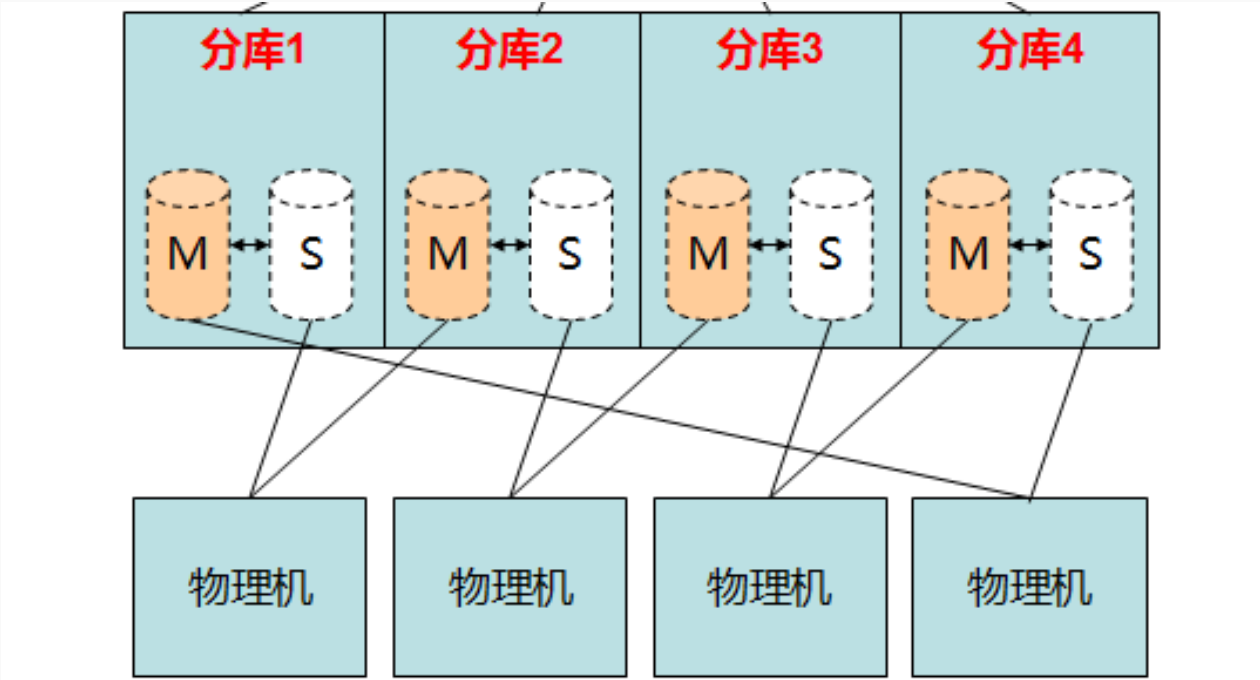
传统的关系数据库性能上缺陷

- 无法满足海量数据的管理需求
- 无法满足高并发需求
- 无法满足高扩展性和高可用性的需求

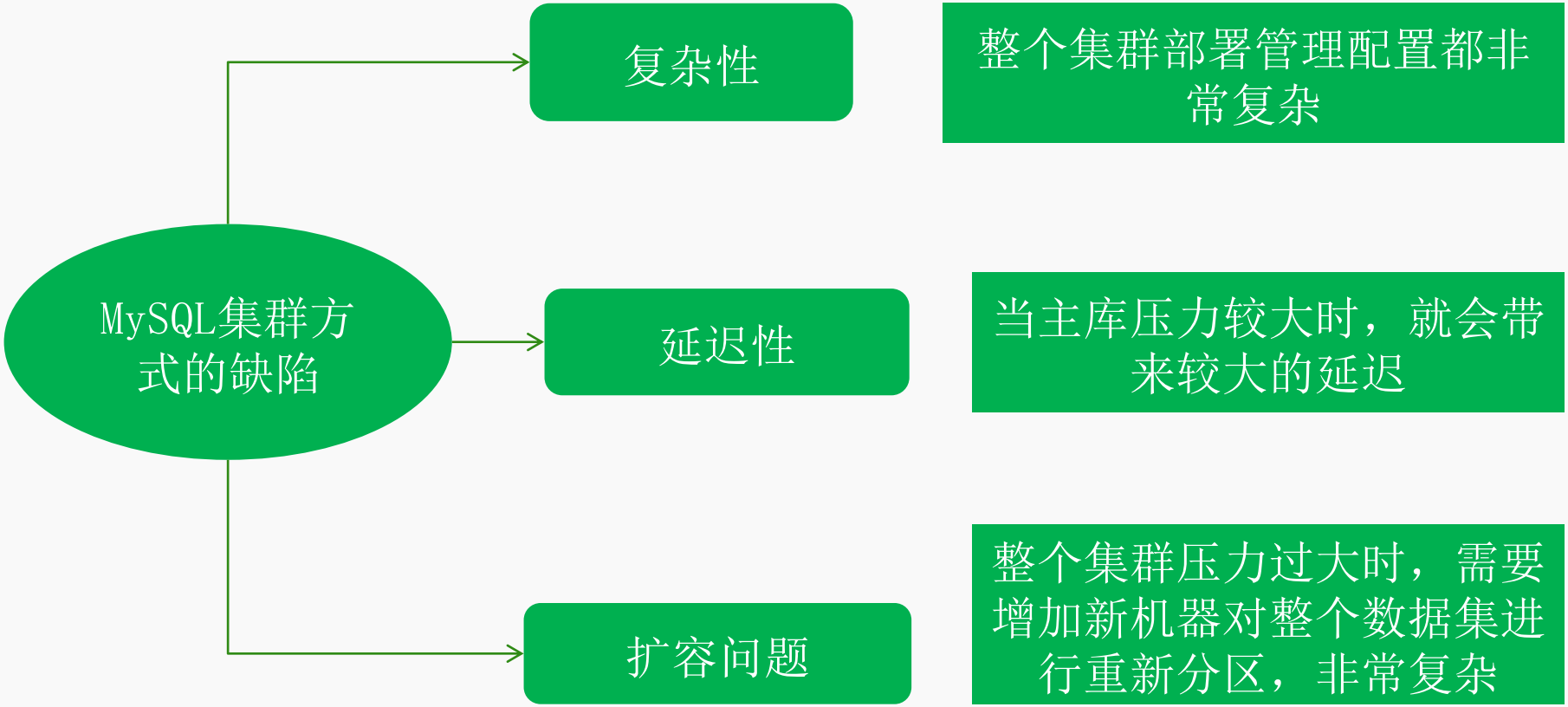
无法满足高扩展性和高可用性的需求



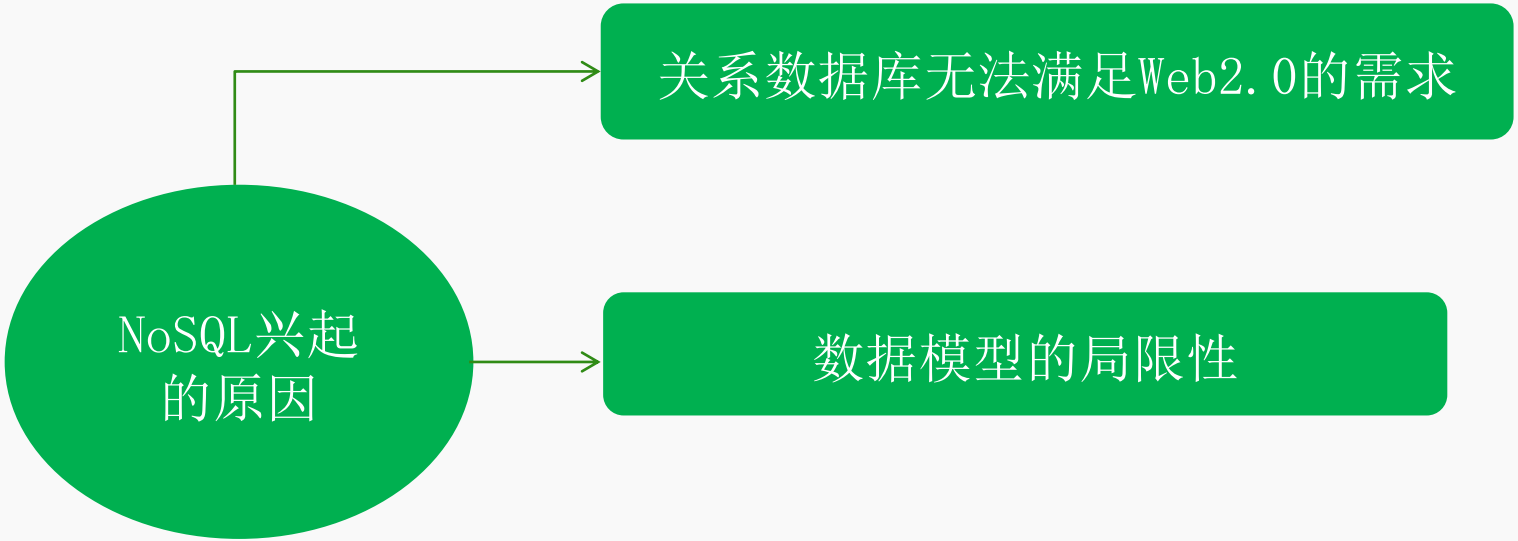
无法满足高扩展性和高可用性的需求



NoSQL简介


MySQL集群方式的缺陷
复杂性
整个集群部署管理配置都非常复杂
延迟性
当主库压力较大时，就会带来较大的延迟
扩容问题
整个集群压力过大时，需要增加新机器对整个数据集进行重新分区，非常复杂

NoSQL简介



NoSQL简介

NoSQL简介


NoSQL兴起
的原因
关系数据库无法满足Web2.0的需求
数据模型的局限性
Web2.0关系数据库许多特性没有发挥

NoSQL简介

01
Web2.0通常是不要求严格数据库事务

02
Web2.0不需要严格的读写实时性

03
Web2.0不包含复杂的SQL查询

本章大纲

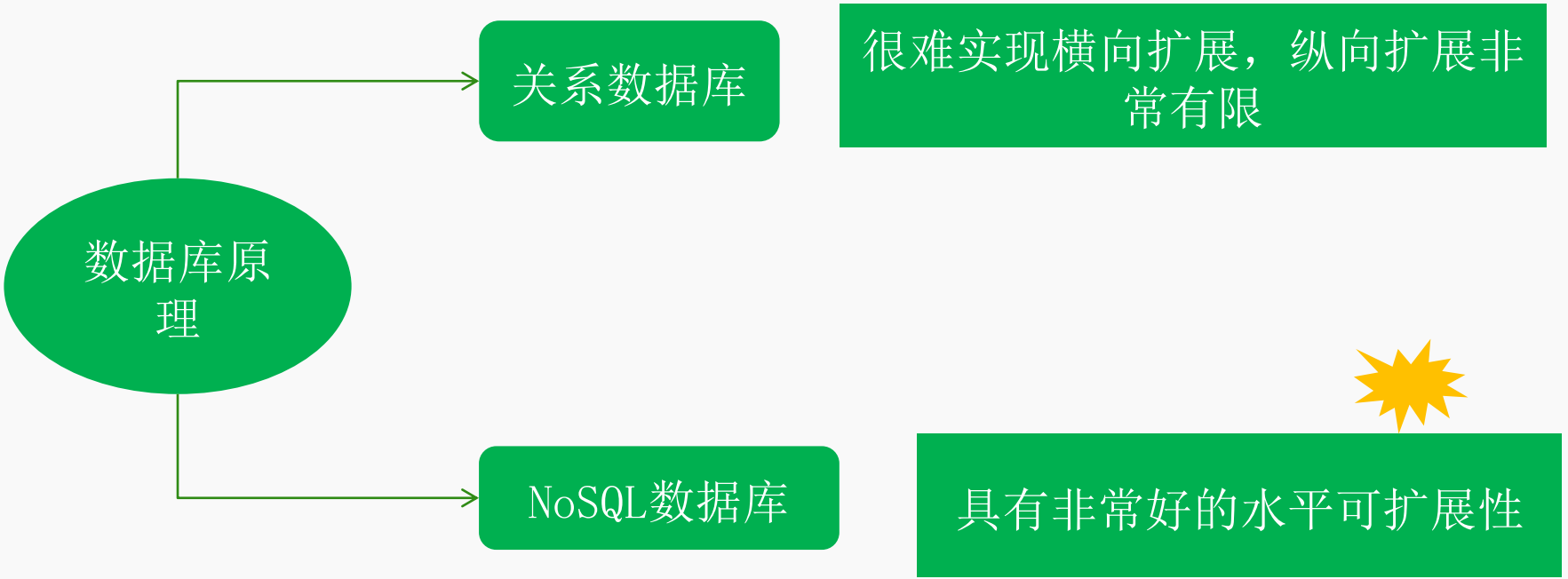
- NoSQL简介
- NoSQL与关系数据库的比较
- NoSQL的四大类型
- NoSQL的三大基石
- 从NoSQL到NewSQL数据库

NoSQL与关系数据库的比较

一、在数据库原理方面

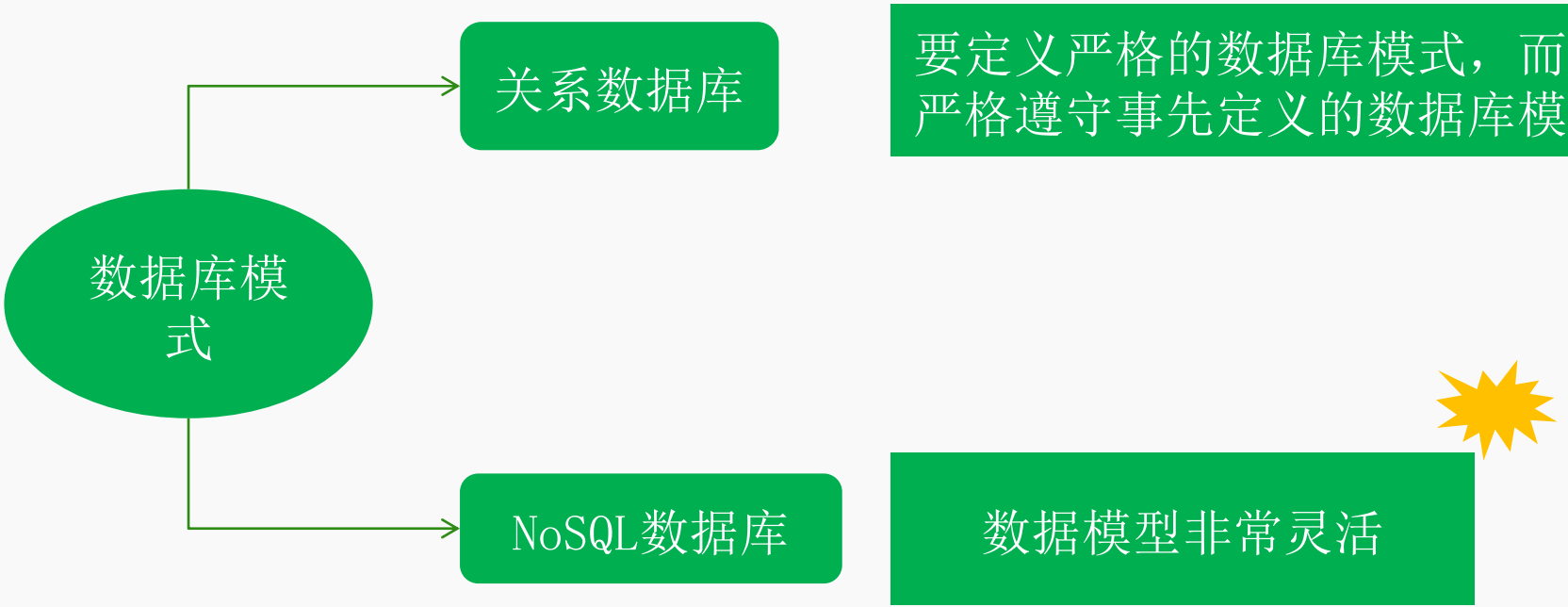
NoSQL与关系数据库的比较

二、在数据规模方面

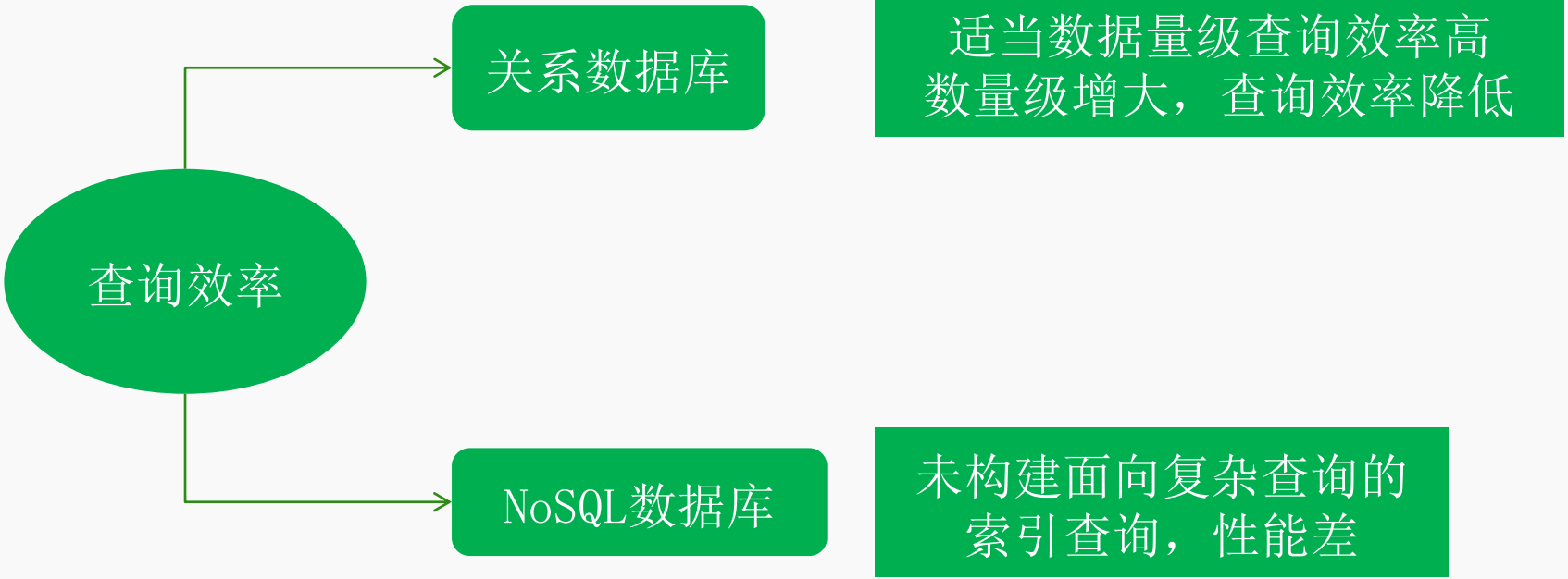


NoSQL与关系数据库的比较

三、在数据库模式方面

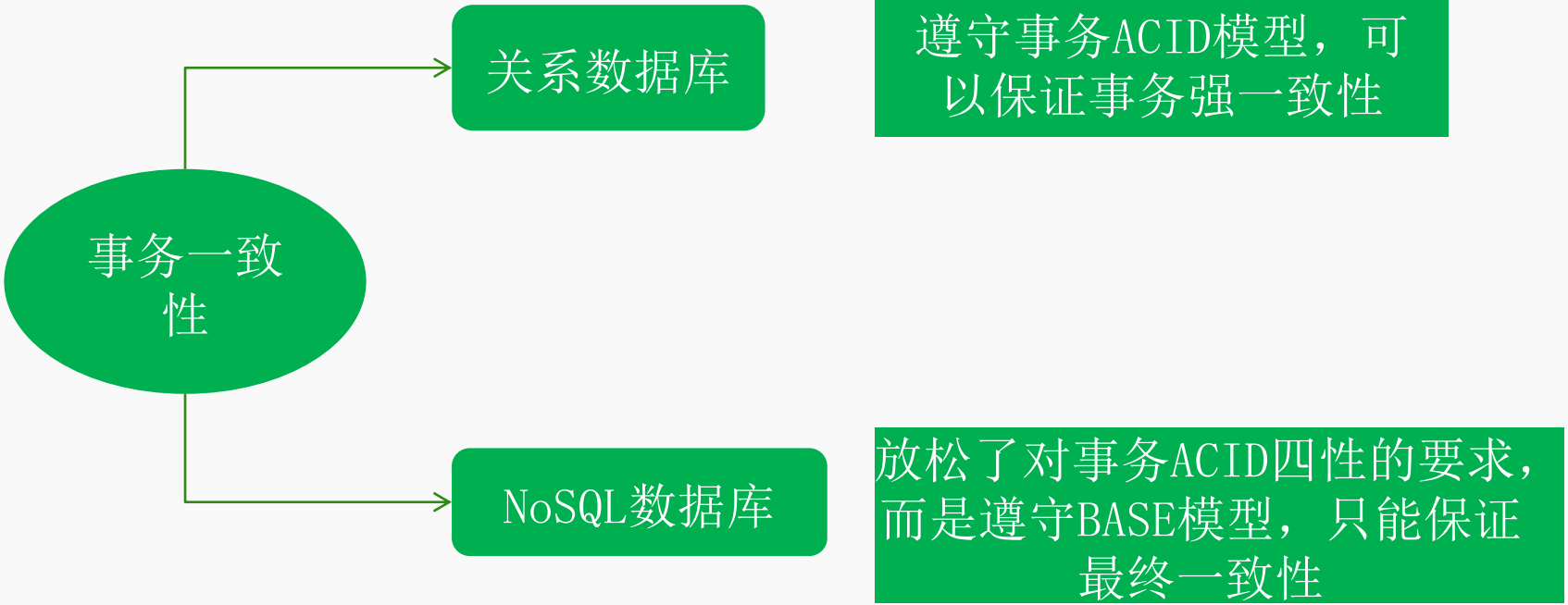


四、在查询效率方面

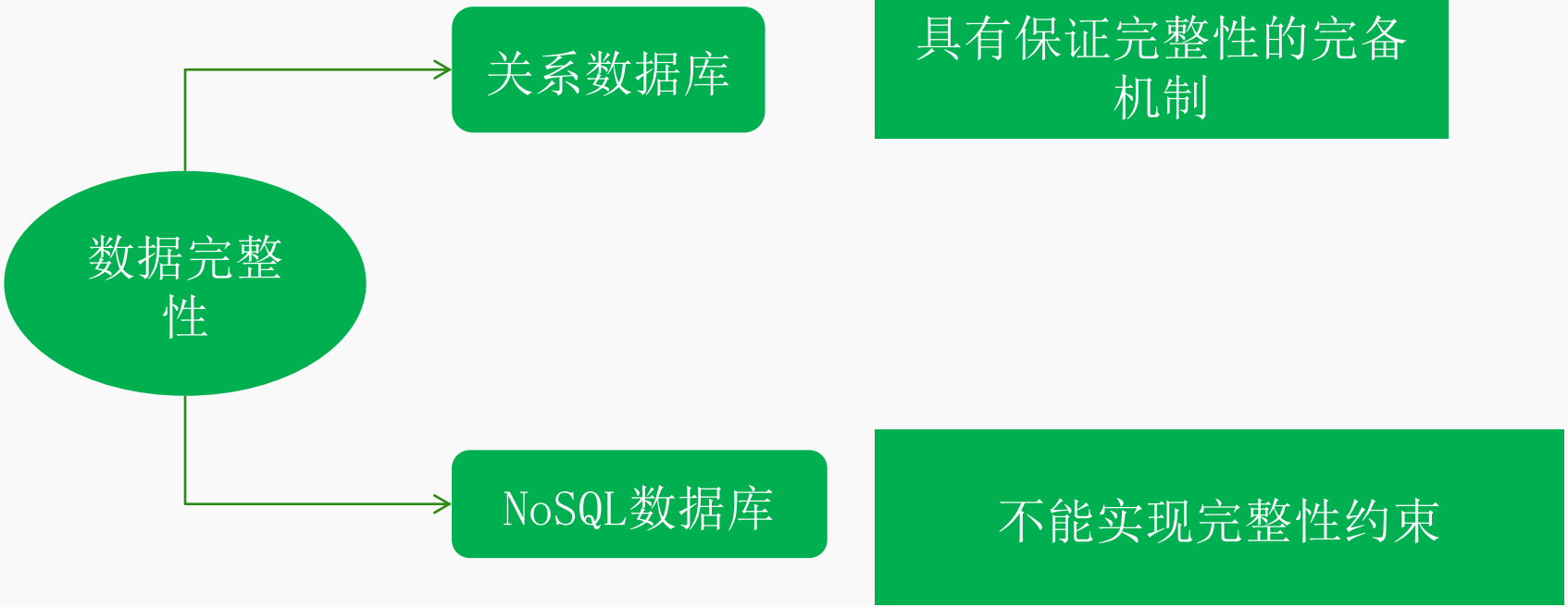

查询效率
关系数据库
适当数据量级查询效率高
数量级增大，查询效率降低
NoSQL数据库
未构建面向复杂查询的
索引查询，性能差

NoSQL与关系数据库的比较

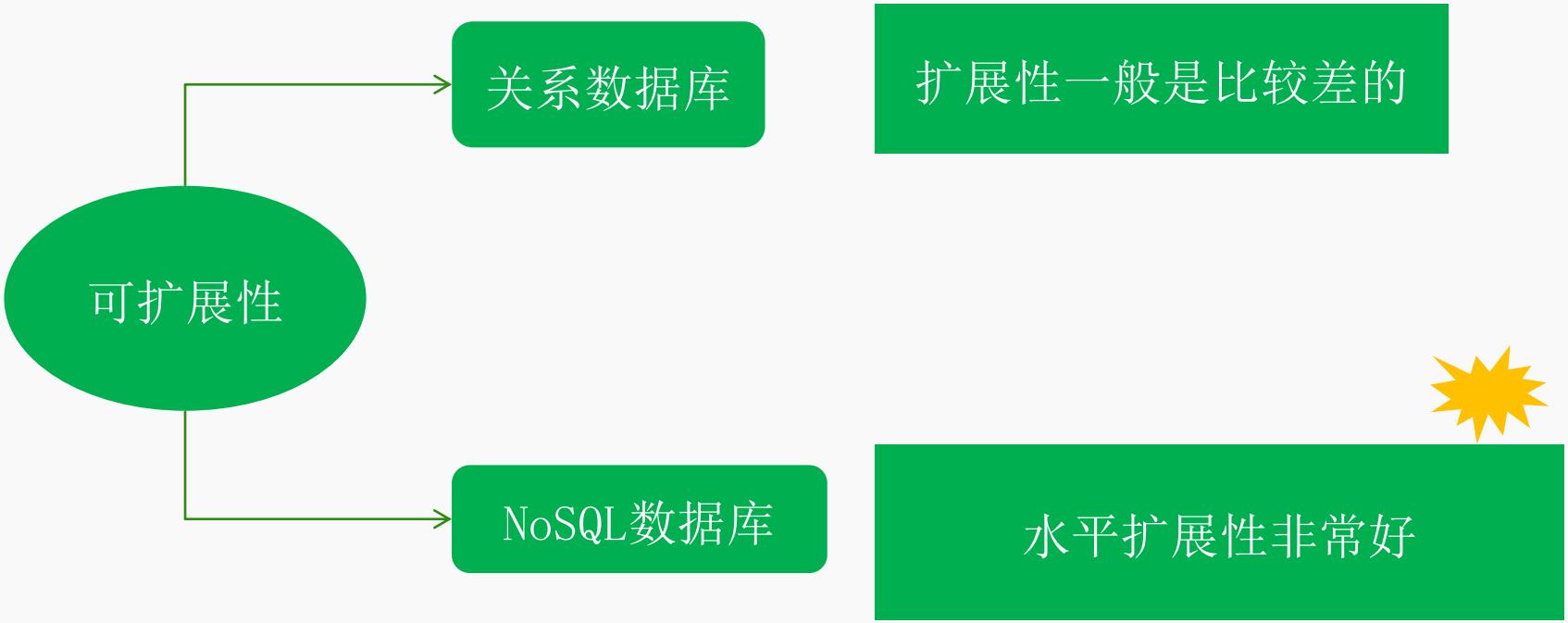
五、在事务一致性方面



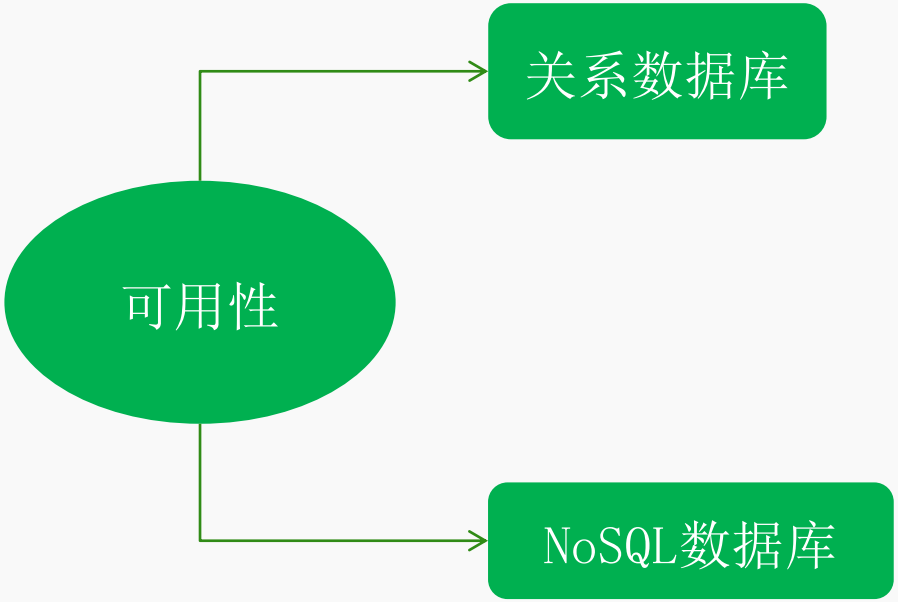
六、在数据完整性方面



七、在可扩展性方面



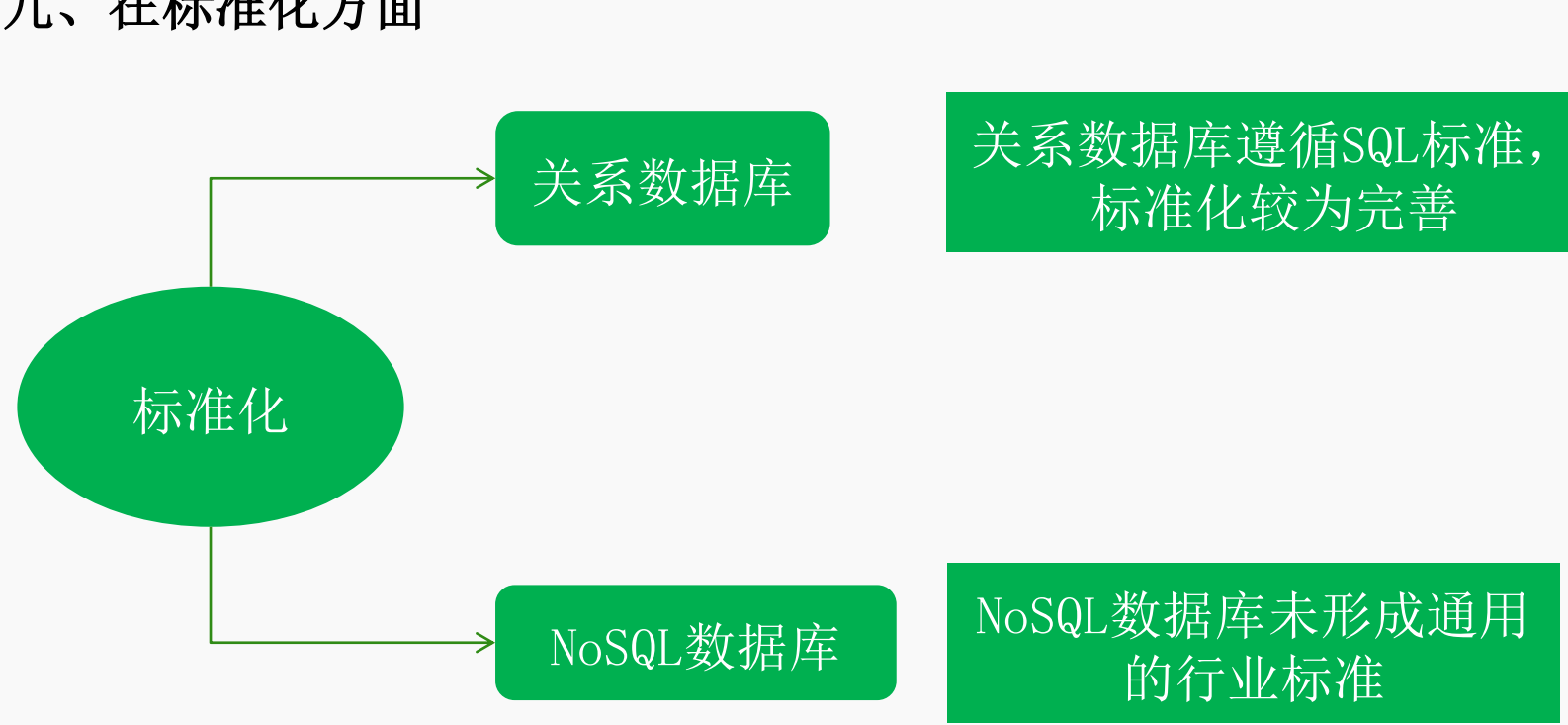
八、在可用性方面



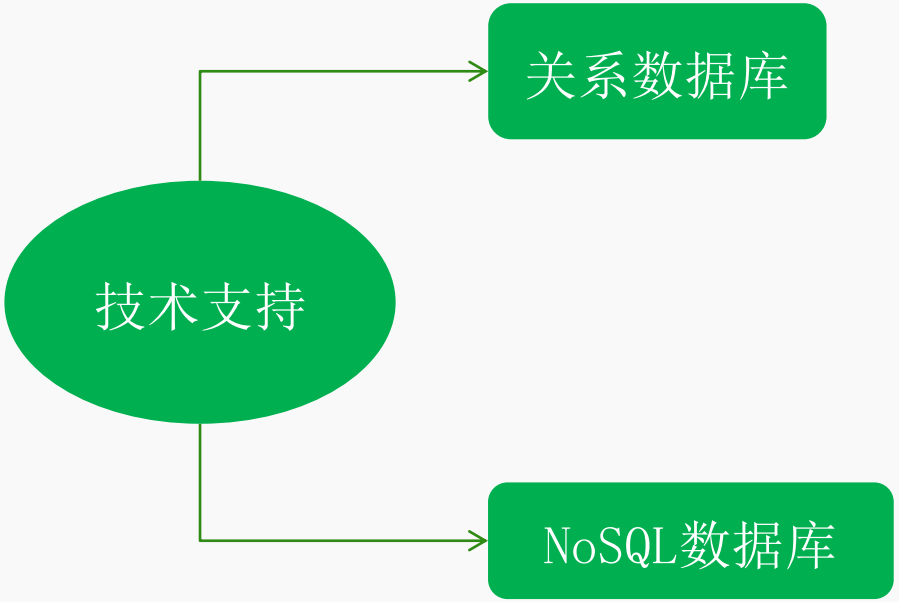
随着规模增大，为了保证严格的一致性，可用性方面就被削弱

具有非常好的可用性，能够在短时间内迅速返回所需的结果

九、在标准化方面


关系数据库
关系数据库遵循SQL标准，
标准化较为完善
标准化
NoSQL数据库
NoSQL数据库未形成通用
的行业标准

十、在技术支持方面

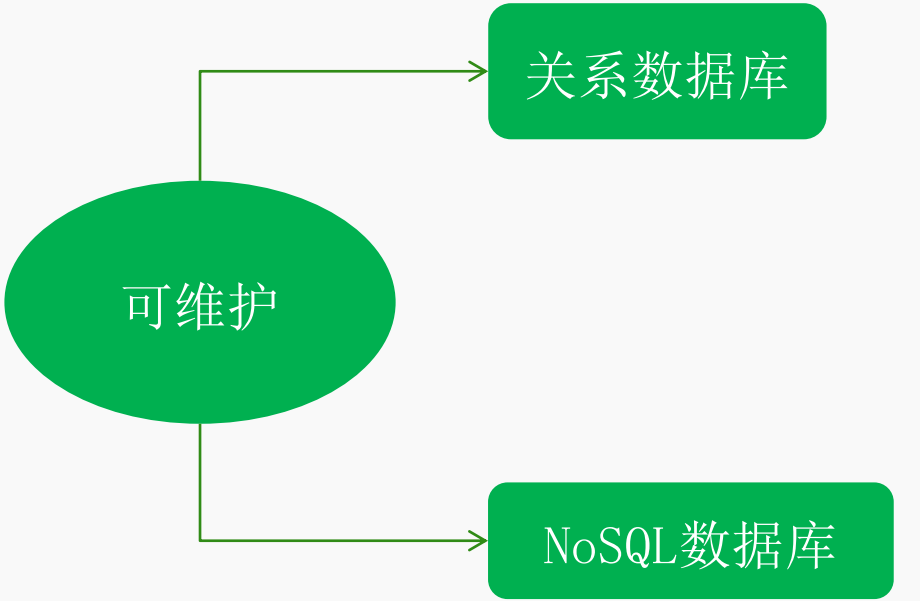


关系数据库很多都是商业数据库，可获得非常强大的技术和后续服务支持

NoSQL数据库很多都属于开源产品，处于整个发展的初级阶段

NoSQL与关系数据库的比较

十一、在可维护方面



关系数据库需要管理员维护

没有成熟的基础和实践操作规范维护较为复杂

NoSQL与关系数据库的比较

关系数据库的优势

- 具有非常完备的关系代数理论作为基础
- 有非常严格的标准
- 支持事务一致性
- 可以借助索引机制实现非常高效的查询

NoSQL与关系数据库的比较

关系数据库的劣势

- 可扩展性非常差
- 不具备水平可扩展性，无法较好支持海量数据存储
- 数据模型定义严格，无法较好满足新型Web2.0应用需求

NoSQL与关系数据库的比较

NoSQL数据库的优势

- 支持超大规模的数据存储
- 数据模型比较灵活

NoSQL数据库的劣势

- 缺乏底层基础理论做支撑
- 很多NoSQL数据库都不支持事务的强一致性

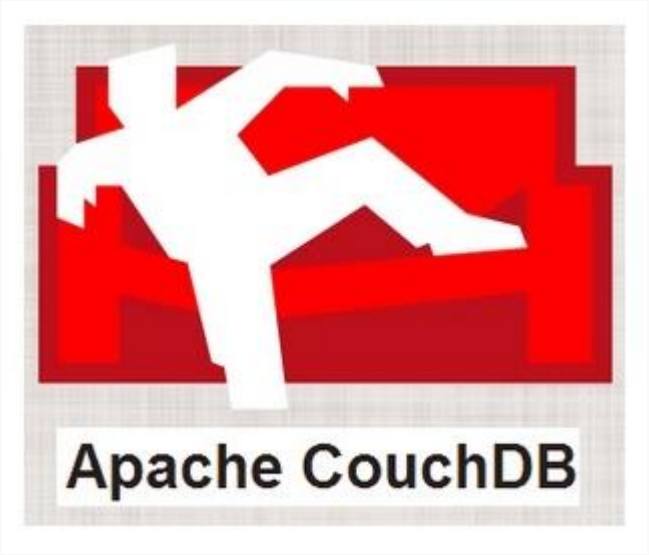
NoSQL与关系数据库的比较

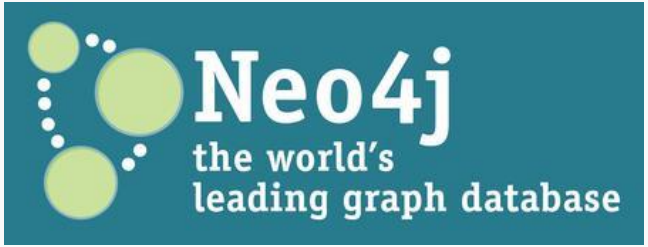
两种数据库的应用场景

NoSQL与关系数据库的比较

Figure 1: Service-oriented architecture of Amazon's

NoSQL与关系数据库的比较





本章大纲

- NoSQL简介
- NoSQL与关系数据库的比较
- NoSQL的四大类型
- NoSQL的三大基石
- 从NoSQL到NewSQL数据库

NoSQL的四大类型

列族数据库

Hbase根据列族进行垂直划分
根据行键进行水平划分

键值数据库

就是一堆键值对

文档数据库

可看作键值数据库
它的值是文档而非标量

图数据库

以图结构方式存储相关信息

NoSQL的四大类型

文档数据库

Couchbase

MarkLogic

mongoDB

图数据库

Neo4j

InfiniteGraph

The Distributed Graph Database

键值数据库

redis

amazon DynamoDB

AEROSPIKE

riak

列族数据库

accumulo

HYPERTABLE INC

Cassandra

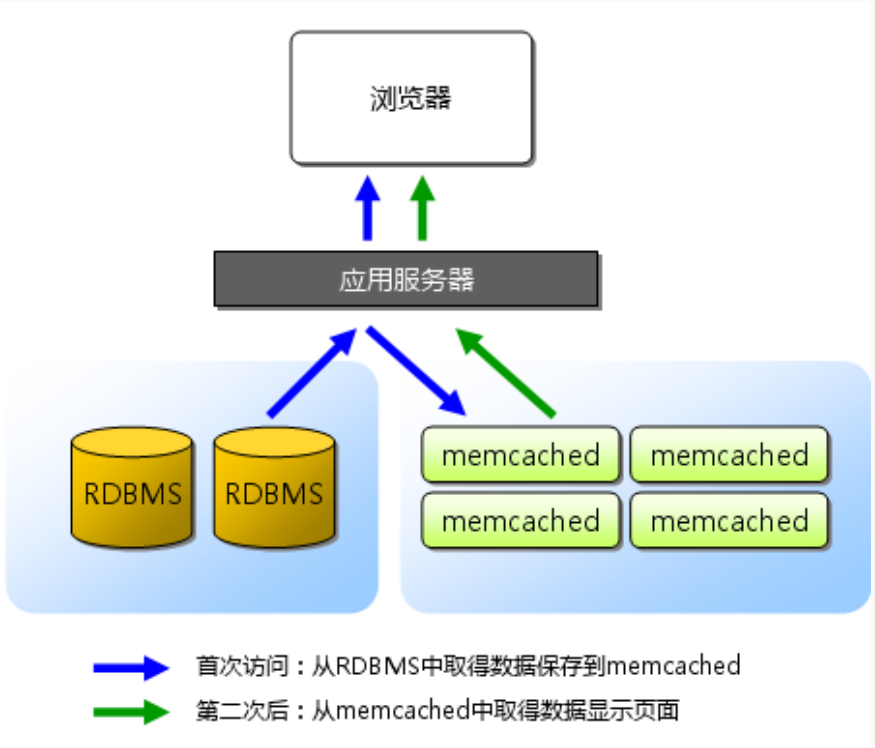
APACHE HBASE

Amazon SimpleDB

键值数据库

相关产品	Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached
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	涉及频繁读写、拥有简单数据模型的应用 内容缓存，比如会话、配置文件、参数、购物车等 存储配置和用户数据信息的移动应用
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：键值数据库根本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）、Twitter（Redis和Memcached）

键值数据库



Redis“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、HadoopDB、GreenPlum、PNUTS
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（HBase）、Yahoo!（HBase）

文档数据库

文档数据库的数据结构：JSON数据格式

```
{
        “ID”:1,
        “NAME”:”SequioaDB”,
        “Tel”:{“Office”,”123123”,”Mobile”:”13811009977”},
        “Addr”:”China,GZ”
}
```

更好的并发性：文档数据库可以完整包含在一个文档里，具有与较好的并发性。在对数据进行更新时，只需要锁定一个文档就可以把相关数据修改掉

文档数据库

特性：

能够对包含的数据类型和内容进行“自我描述”。

一个XML文档：

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

关系数据库：

必须有schema信息才能理解数据的含义

学生（学号，姓名，性别，年龄，系，年级）

（1001，张三，男，20，计算机，2002）

文档数据库

相关产品	MongoDB、CouchDB、Terrastore、ThruDB、RavenDB、SisoDB、RaptorDB、CloudKit、Perservere、Jackrabbit
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
优点	性能好（高并发），灵活性高，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、OrientDB、InfoGrid、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分析

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

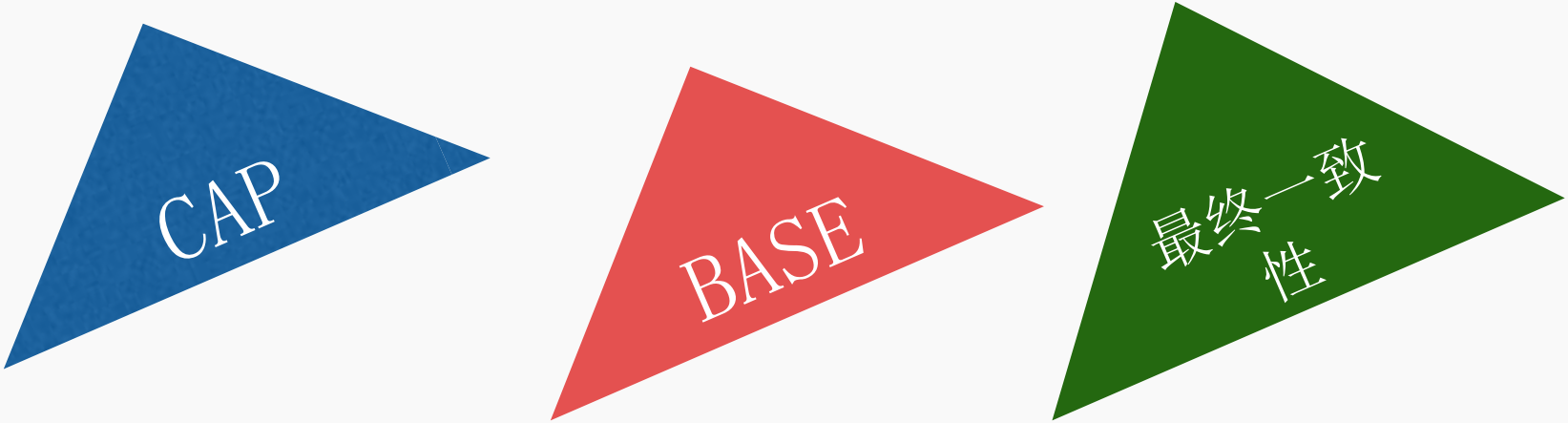
模型相对较为简单，可提供随机数据存储，数据库伸缩性较好

本章大纲

- NoSQL简介
- NoSQL与关系数据库的比较
- NoSQL的四大类型
- NoSQL的三大基石
- 从NoSQL到NewSQL数据库

CAP理论

NoSQL数据库的的三大理论基石



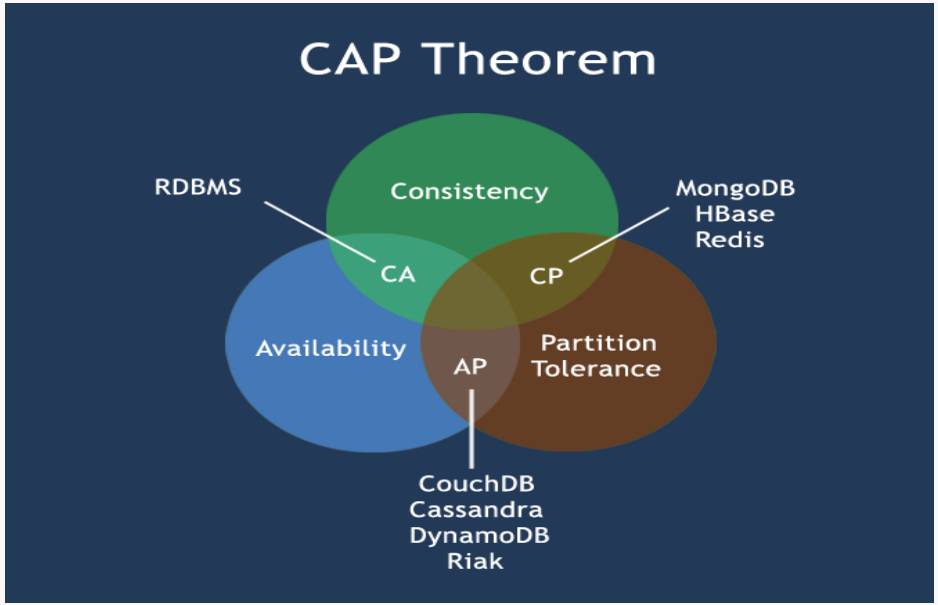
CAP理论

所谓的CAP指的是：

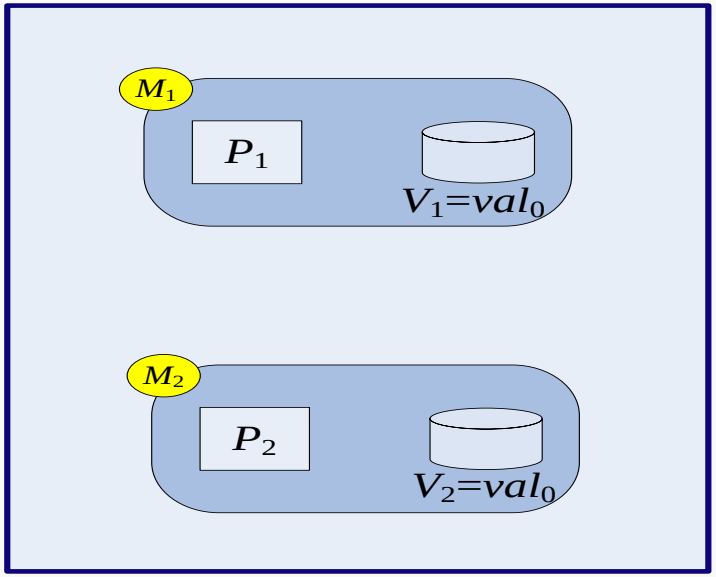
- C（Consistency）：一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据
- A:（Availability）：可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应；
- P（Tolerance of Network Partition）：分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作。

CAP理论

CAP理论告诉我们，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，正所谓“鱼和熊掌不可兼得”。



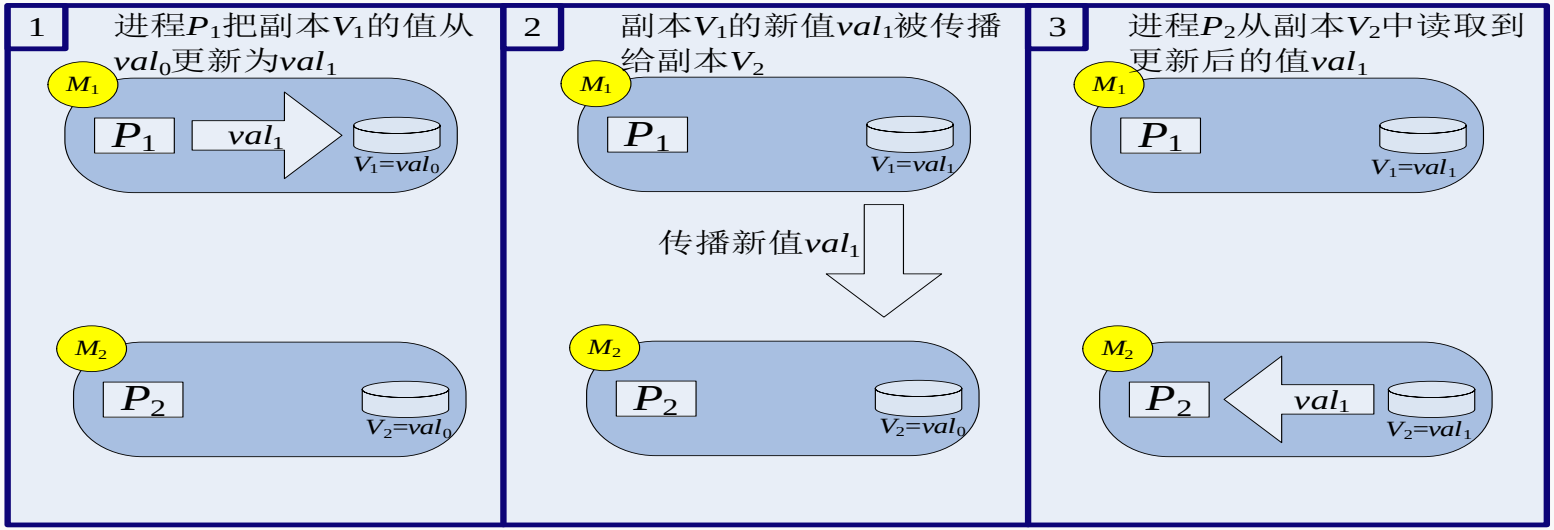
一个牺牲一致性来换取可用性的实例



（a）初始状态

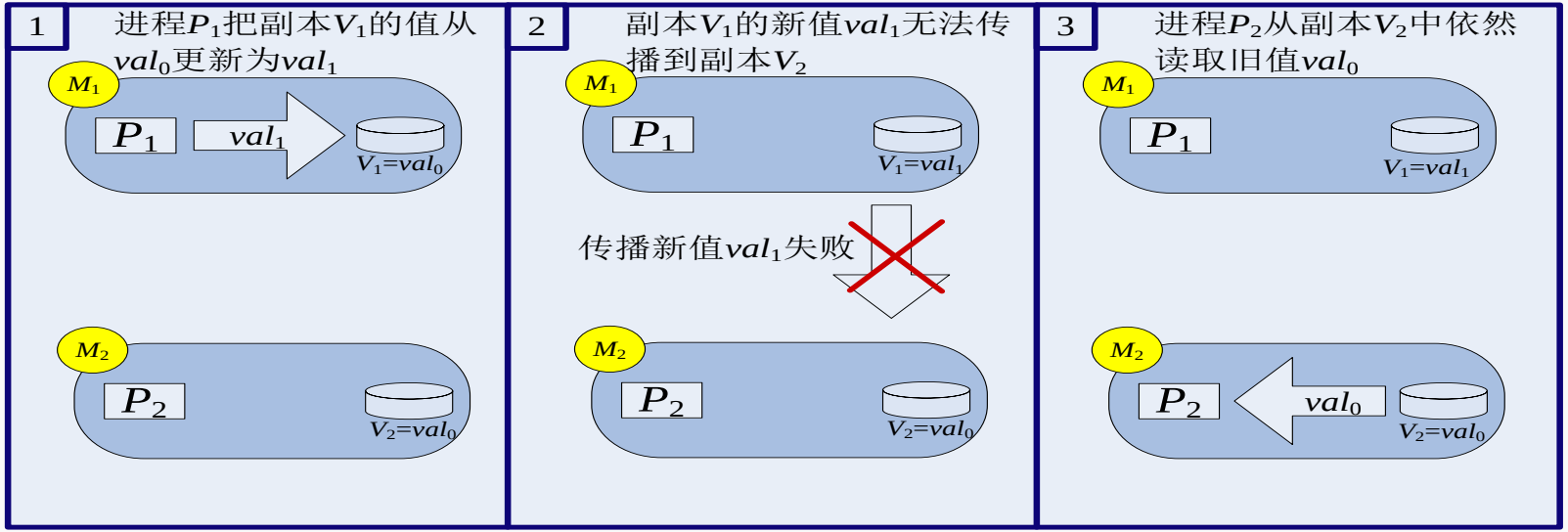
CAP理论

一个牺牲一致性来换取可用性的实例



（b）正常执行过程

一个牺牲一致性来换取可用性的实例



（c）更新传播失败时的执行过程

CAP理论

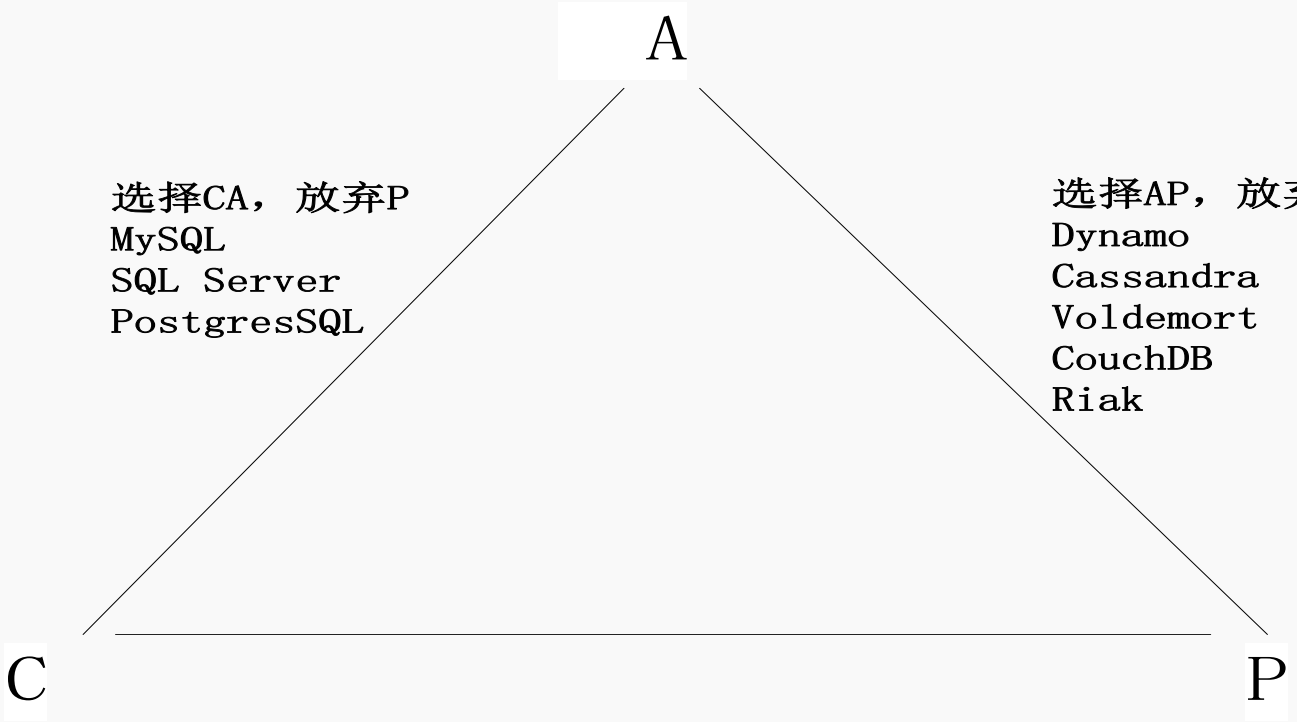
当处理CAP的问题时，可以有几个明显的选择：

1.CA：也就是强调一致性（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。传统的关系数据库（MySQL、SQL Server和PostgreSQL），都采用了这种设计原则，因此，扩展性都比较差

2.CP：也就是强调一致性（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

3.AP：也就是强调可用性（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理论



不同产品在CAP理论下的不同设计原则

BASE

BASE（**B**asically **A**vailble, **S**oft-state, **E**ventual consistency），不得不谈到ACID。

ACID（酸）	BASE（碱）
原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durable)	

BASE

一个数据库事务具有ACID四性：

- 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

BASE的基本含义是基本可用（Basically Availble）、软状态（Soft-state）和最终一致性（Eventual consistency）：

- **基本可用**

基本可用，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**

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

BASE的基本含义是基本可用（Basically Availble）、软状态（Soft-state）和最终一致性（Eventual consistency）：

- **最终一致性**

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。

最常见的实现最终一致性的系统是DNS（域名系统）。一个域名更新操作根据配置的形式被分发出去，并结合有过期机制的缓存；最终所有的客户端可以看到最新的值。

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：

- **因果一致性：**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
- **“读己之所写”一致性：**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
- **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**：它把访问存储系统的进程放到会话（session）的上下文中，只要会话还存在，系统就保证“读已之所写”一致性。如果由于某些失败情形令会话终止，就要建立新的会话，而且系统保证不会延续到新的会话
- **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。系统必须保证这种程度的一致性，否则就非常难以编程了

最终一致性

如何实现各种类型的一致性？

假设有一个分布式数据系统：

- N — 数据复制的份数
- W — 更新数据是需要保证写完成的节点数
- R — 读取数据的时候需要读取的节点数

最终一致性

如何实现各种类型的一致性？

W+R>N
强一致性
写的节点和读的节点重叠，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

W+R<=N
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。

N=W,R=1
强一致性
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。

最终一致性

实例

HBase是借助其底层的HDFS来实现其数据冗余备份的。

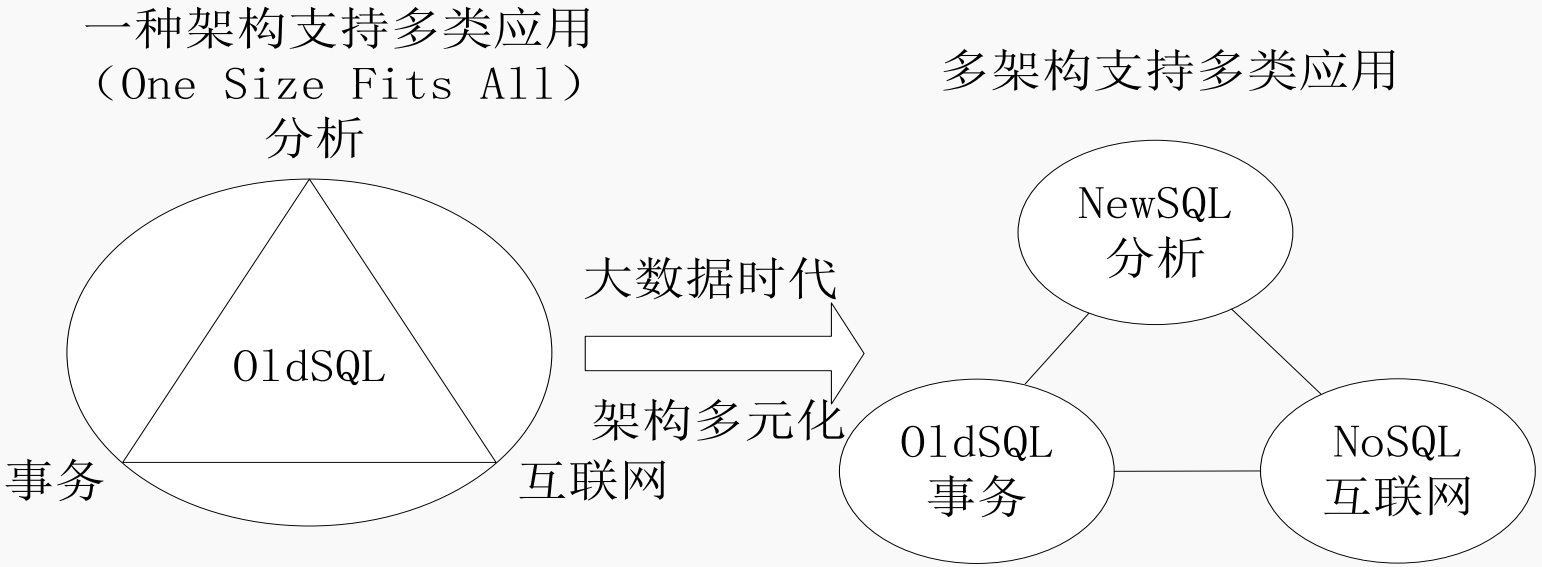
HDFS采用的就是强一致性保证。在数据没有完全同步到N个节点前，写操作是不会返回成功的。也就是说它的W＝N，而读操作只需要读到一个值即可，也就是说它R＝1。

本章大纲

- NoSQL简介
- NoSQL与关系数据库的比较
- NoSQL的四大类型
- NoSQL的三大基石
- 从NoSQL到NewSQL数据库

从NoSQL到NewSQL数据库

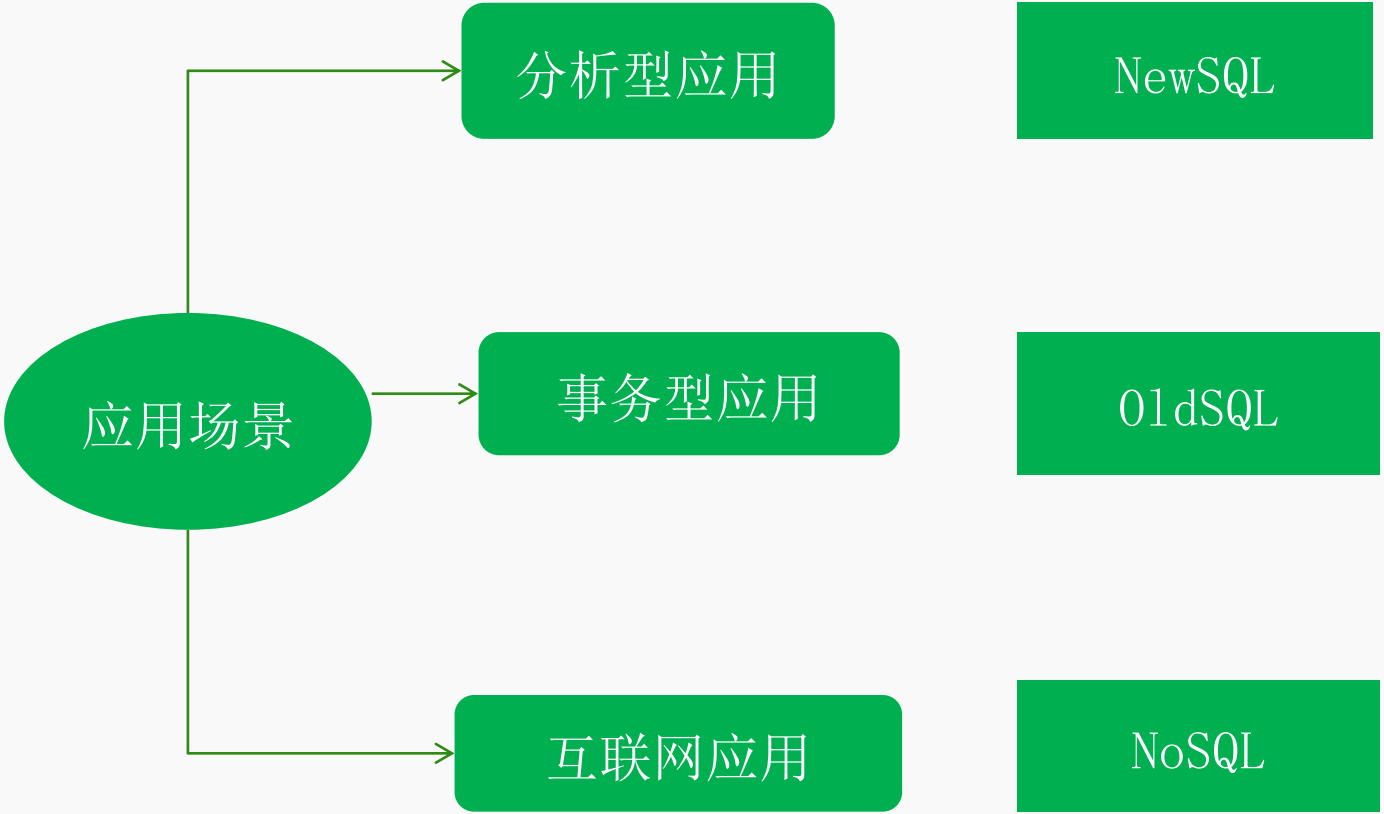
数据库的发展



大数据引发数据处理架构变革

从NoSQL到NewSQL数据库

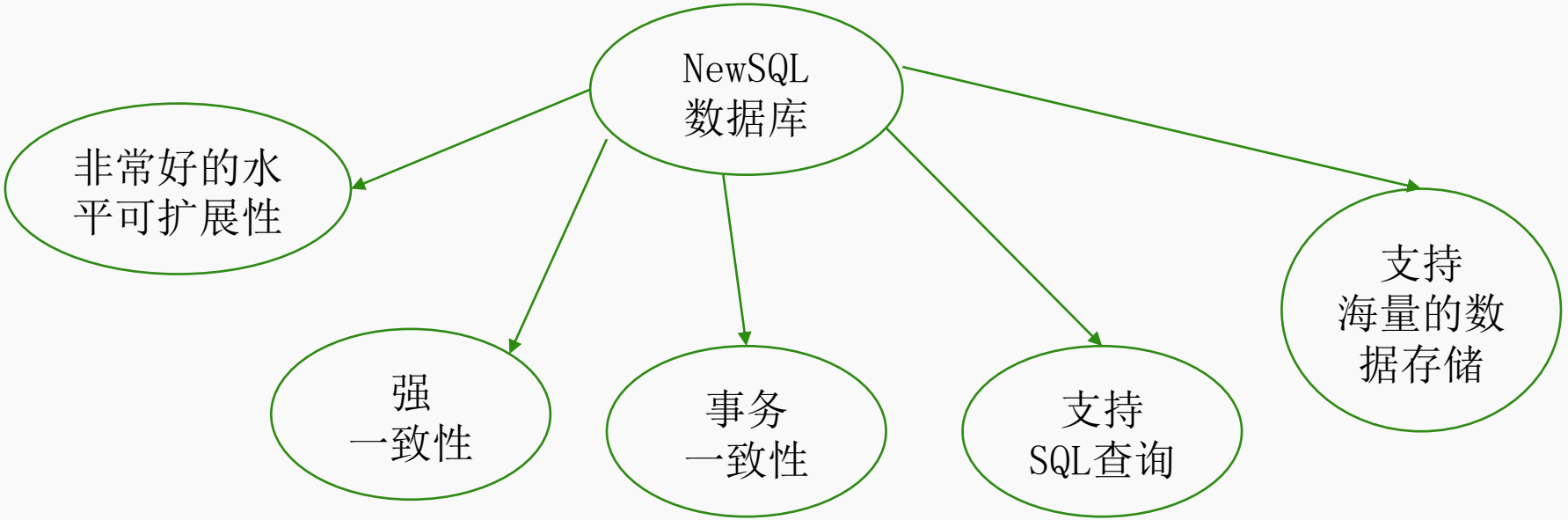
应用场景



从NoSQL到NewSQL数据库

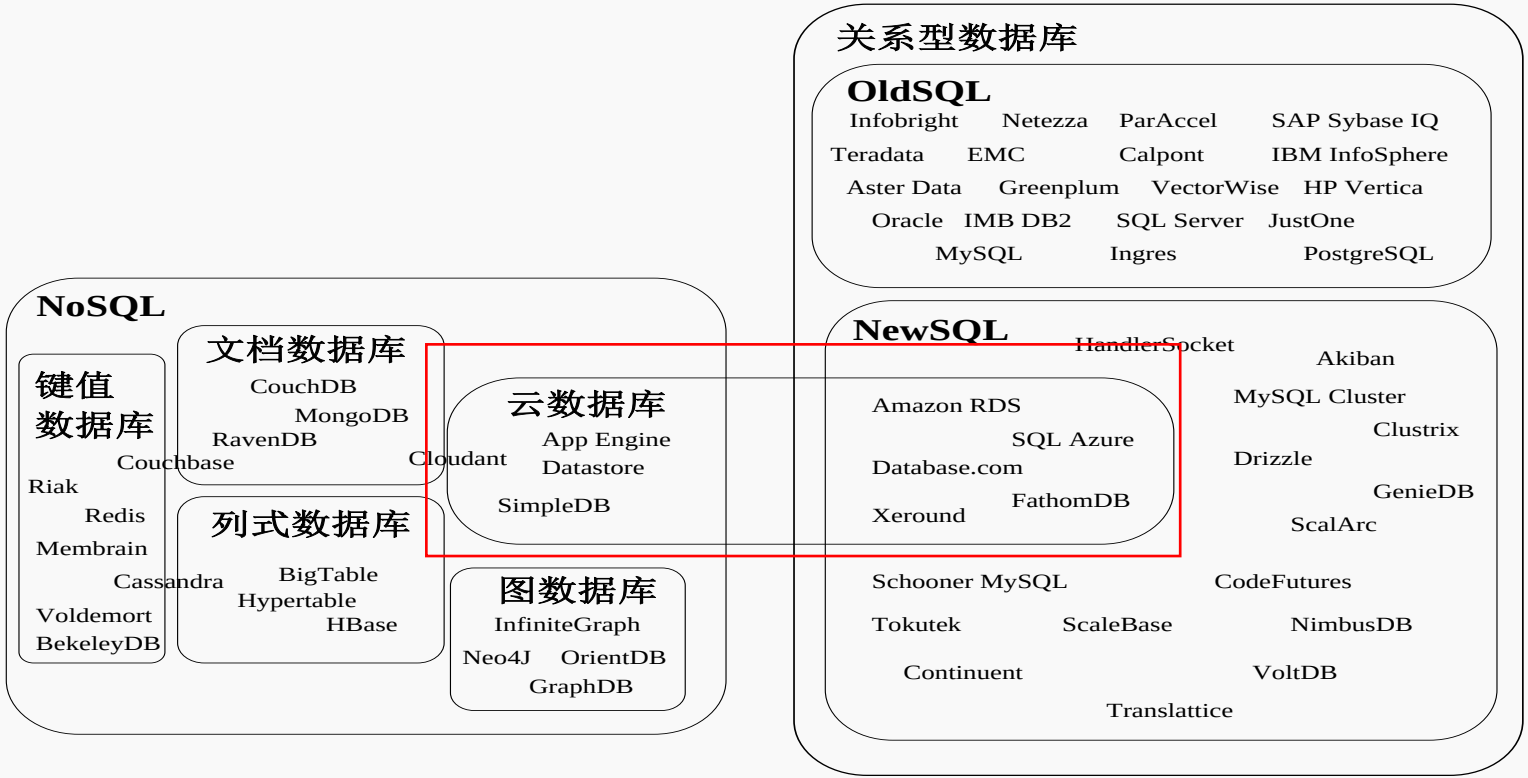
NewSQL数据库

NewSQL同时具备OldSQL数据库和NoSQL数据库各自的优点



从NoSQL到NewSQL数据库

关系数据库、NoSQL和NewSQL数据库产品分类图



课后题

- 关系数据库在哪些方面无法满足web2.0的需求?
- 为什么说关系数据库的一些关键特性在web2.0时代称为肋“鸡肋”?
- 比较关系数据库与NoSQL的优缺点
- 论述NoSQL数据库的四大类型，适用场合和优缺点
- 论述CAP理论的具体含义
- 论述数据库ACID四性的含义
- 论述BASE的含义
- 什么是最终一致性?
- 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，又可以区分为哪些不同类型的一致性?