# NoSQL数据库

# 提纲

1 NoSQL简介

2 NoSQL兴起的原因

3 NoSQL与关系数据库的比较

4 NoSQL的四大类型

5 NoSQL的三大基石

6 从NoSQL到NewSQL数据库

7 文档数据库MongoDB

# 1　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

通常，NoSQL数据库具有以下几个特点：

（1）灵活的可扩展性

（2）灵活的数据模型

（3）与云计算紧密融合

# 1　NoSQL简介

现在已经有很多公司使用了NoSQL数据库：

- Google
- Facebook
- Mozilla
- Adobe
- Foursquare
- LinkedIn
- Digg
- McGraw-Hill Education
- Vermont Public Radio
- 百度、腾讯、阿里、新浪、华为……

# 2　NoSQL兴起的原因

一、关系数据库已经无法满足Web2.0的需求。主要表现在以下几个方面：

（1）无法满足海量数据的管理需求

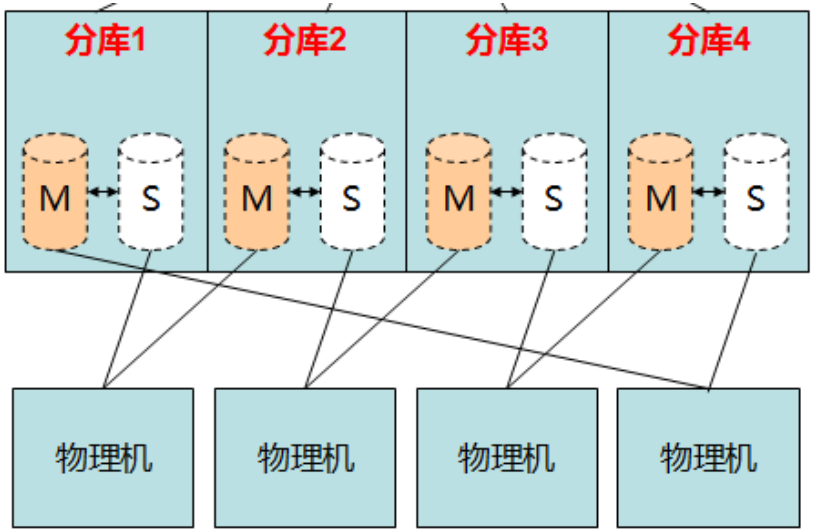
（2）无法满足数据高并发的需求

（3）无法满足高可扩展性和高可用性的需求

# 2　NoSQL兴起的原因

MySQL集群是否可以完全解决问题？

- **复杂性**：部署、管理、配置很复杂
- **数据库复制**：MySQL主备之间采用复制方式，只能是异步复制，当主库压力较大时可能产生较大延迟，主备切换可能会丢失最后一部分更新事务，这时往往需要人工介入，备份和恢复不方便
- **扩容问题**：如果系统压力过大需要增加新的机器，这个过程涉及数据重新划分，整个过程比较复杂，且容易出错
- **动态数据迁移问题**：如果某个数据库组压力过大，需要将其中部分数据迁移出去，迁移过程需要总控节点整体协调，以及数据库节点的配合。这个过程很难做到自动化



# 2 NoSQL兴起的原因

二、“One size fits all”模式很难适用于截然不同的业务场景

- 关系模型作为统一的数据模型既被用于数据分析，也被用于在线业务。但这两者一个强调高吞吐，一个强调低延时，已经演化出完全不同的架构。用同一套模型来抽象显然是不合适的
- Hadoop就是针对数据分析
- MongoDB、Redis等是针对在线业务，两者都抛弃了关系模型

# 2　NoSQL兴起的原因

三、关系数据库的关键特性包括完善的事务机制和高效的查询机制。但是，关系数据库引以为傲的两个关键特性，到了Web2.0时代却成了鸡肋，主要表现在以下几个方面：

（1）Web2.0网站系统通常不要求严格的数据库事务

（2）Web2.0并不要求严格的读写实时性

（3）Web2.0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）

# 3 NoSQL与关系数据库的比较

NoSQL和关系数据库的简单比较

| 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<br>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<br>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<br>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）<br>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，但是，在复杂查询方面的性能仍然不如RDBMS |

# 3 NoSQL与关系数据库的比较

NoSQL和关系数据库的简单比较（续）

| 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<br>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<br>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<br>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<br>大多数NoSQL都能提供较高的可用性 |

# 3　NoSQL与关系数据库的比较

NoSQL和关系数据库的简单比较（续）

| 比较标准 | RDBMS | NoSQL | 备注 |
| --- | --- | --- | --- |
| 标准化 | 是 | 否 | RDBMS已经标准化（SQL）<br>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<br>StoneBraker认为：NoSQL缺乏统一查询语言，将会拖慢NoSQL发展 |
| 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<br>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<br>NoSQL数据库虽然没有DBMS复杂，也难以维护 |

# 3 NoSQL与关系数据库的比较总结

**（1）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

**（2）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，技术尚不成熟，缺乏专业团队的技术支持，维护较困难等

# 3 NoSQL与关系数据库的比较总结

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）

**采用混合架构**

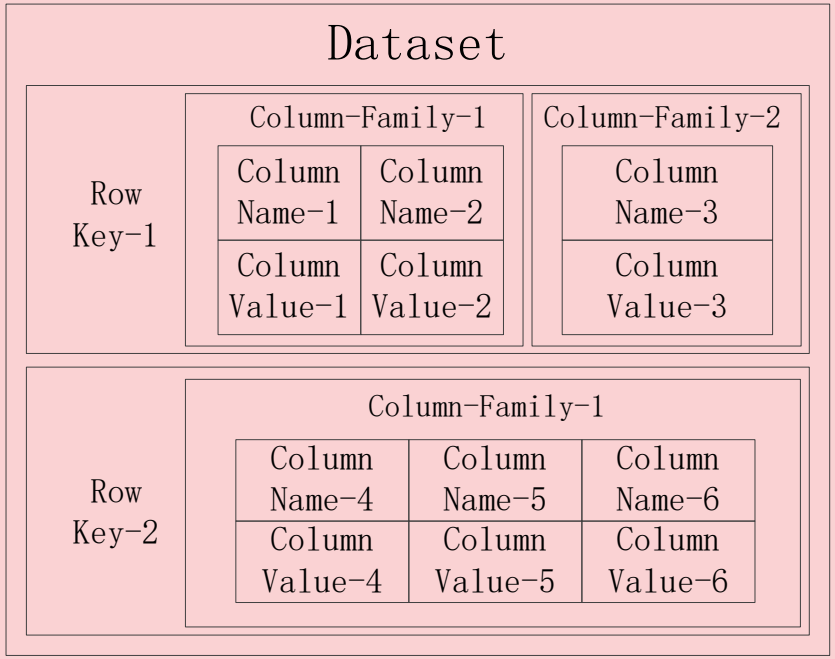
- 案例：亚马逊公司就使用不同类型的数据库来支撑它的电子商务应用
- 对于“购物篮”这种临时性数据，采用键值存储会更加高效
- 当前的产品和订单信息则适合存放在关系数据库中
- 大量的历史订单信息则适合保存在类似MongoDB的文档数据库中

# 4　NoSQL的四大类型

NoSQL数据库虽然数量众多，但是归结起来，典型的NoSQL数据库通常包括键值数据库、列族数据库、文档数据库和图形数据库。

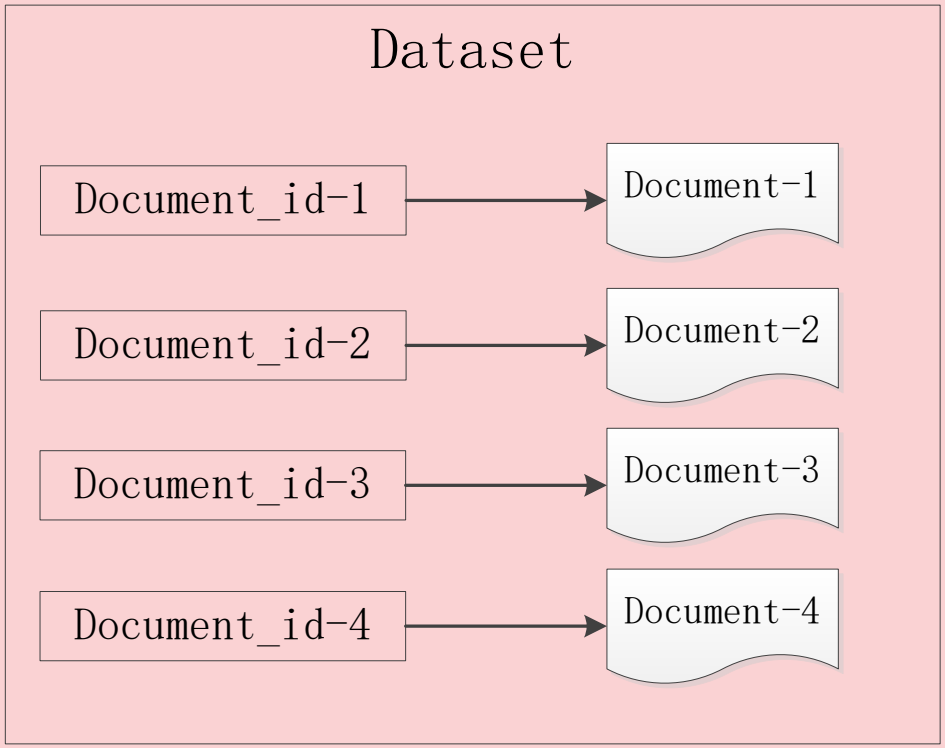
| Key_1 | Value_1 |
|---|---|
| Key_2 | Value_2 |
| Key_3 | Value_1 |
| Key_4 | Value_3 |
| Key_5 | Value_2 |
| Key_6 | Value_1 |
| Key_7 | Value_4 |
| Key_8 | Value_3 |

键值数据库

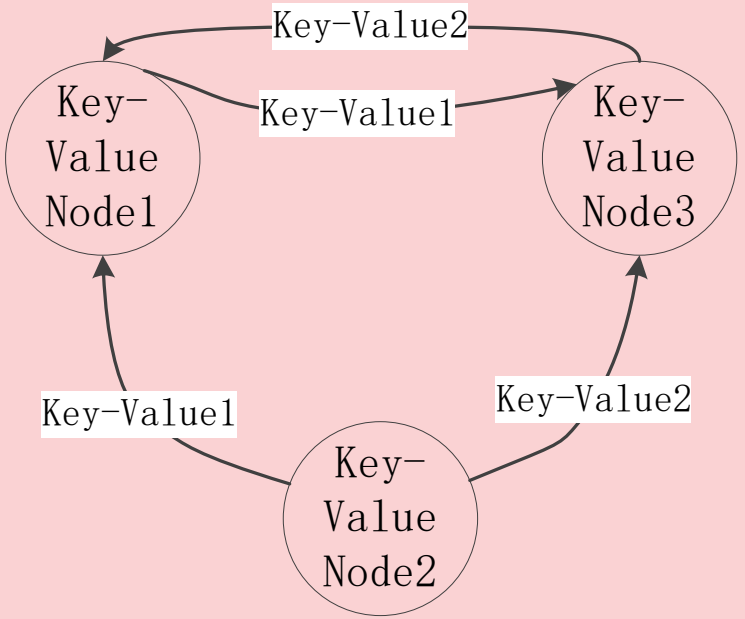



列族数据库

# 4　NoSQL的四大类型




文档数据库




图形数据库

# 4　NoSQL的四大类型

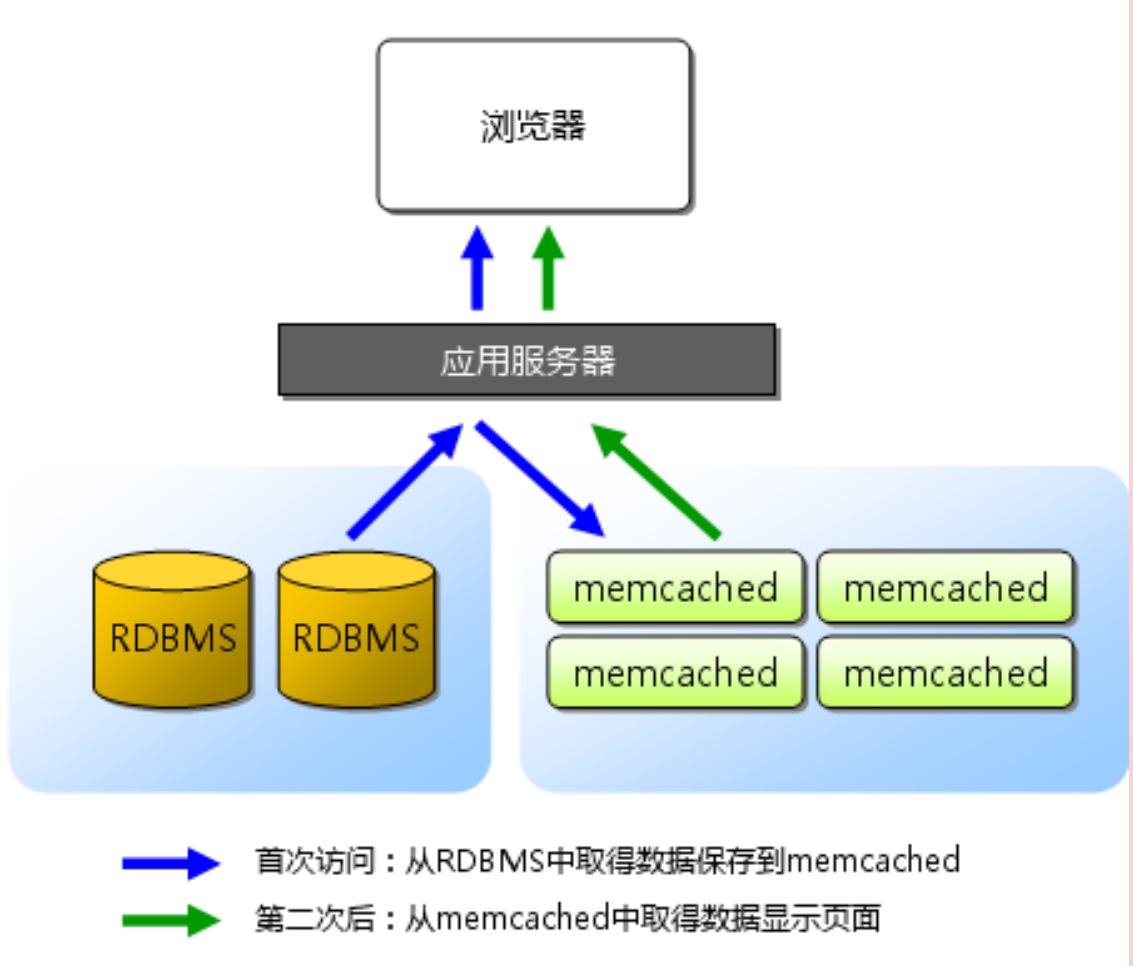
| 文档数据库 | 图数据库 |
| --- | --- |
|  Couchbase MarkLogic mongoDB | Neo4j InfiniteGraph The Distributed Graph Database |
| **键值数据库** | **列族数据库** |
| redis amazon DynamoDB AEROSPIKE riak |  accumulo HYPERTABLE INC Cassandra APACHE HBASE Amazon SimpleDB |

# 4.1 键值数据库

| | |
|---|---|
| 相关产品 | Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached |
| 数据模型 | 键/值对<br>键是一个字符串对象<br>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<br>内容缓存，比如会话、配置文件、参数、购物车等<br>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<br>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<br>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） |

# 4.1 键值数据库

Redis有时候会被人们称为“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

# Memcached

- Memcached是个高性能的基于内存的分布式缓存系统，Memcached的分布式是基于客户端Key的hash来做均衡，是个伪分布式的系统。Memcached有2个组件，一个客户端（java），一个服务器端c编写。
- 基于内存（但是重启了后，数据丢失），提供简单的get/set方法，缓存对象只能小于1M，安装使用比较简单。
- 中小型网站应用中来存放非可靠性的只读数据。例如：用户的信息、网站的计数、数据库DAO前的行记录缓存、全局的代码参数。

# Redis

- Redis 是一个用c语言写的类似Memcached 的key-value的存储系统，它比Memcached提供了更多的API接口和更好的并发性能，可以支持10万并发的读写，建议用Redis代替Memcahed。
- Redis是基于内存的，因此部署Redis的机器对于内存是非常有高的要求的，Redis是会把数据实时写到内存中，再定时同步到文件，不会掉数据。
- Redis 可以当作数据库来用，但是有缺陷，在可靠性上，没有Oracle关系型数据库来的稳定。可以作为持久层的Cache层,缓存下面的数据库结构:计数、排行榜单、最新浏览的数据、队列（订阅关系）。

# Redis 的配置-基本配置

Redis的核心配置文件，只有一个，就是redis.conf 文件，下面是列举了核心的配置项。

| 配置项 | 值 | 字段意义 |
| --- | --- | --- |
| daemonize | no或者yes | 是不是需要在后台运行 |
| pidfile | /var/run/redis.pid | pid文件 |
| port | 6379 | 启动端口 |
| bind | 127.0.0.1 | 绑定IP，只有指定的IP地址才能够访问redis实例 |
| timeout | 0 | 一个客户端空闲多少秒后关闭连接 |
| databases | 16 | 数据库数 |
| loglevel | Debug<br>Verbose<br>Notice<br>warning | debug 开发和测试的时候配置<br>verbose 比debug信息少点<br>notice 基本信息，生产环境建议配置这个<br>warning 严重错误的时候 |

# Redis 的配置-快照配置

| 配置项 | 值 | 字段意义 |
| --- | --- | --- |
| save | 900 1 | 指出在多长时间内，有多少次更新操作，就将数据同步到数据文件rdb,在15分钟内，有1个key被改变就同步到文件中去 |
| rdbcompression | yes | 当导出到 .rdb 数据库时是否用LZF压缩字符串对象 |
| rdbchecksum | yes | 存储和加载rdb文件时校验 |
| dbfilename | dump.rdb | 数据库的文件名 |
| dir | ./ | 工作目录 |

# Redis 的配置-同步配置

| 配置项 | 值 | 字段意义 |
| --- | --- | --- |
| slaveof | 127.0.0.1 6379 | 主服务器地址 |
| masterauth | <master-password> | 从服务器连接主服务器的时候的密码 |
| slave-serve-stale-data | Yes/no | 当从库正在做复制的同步工作的时候，从库的工作模式，配置成yes了后继续相应客户的请求，配置成no了后，不相应客户的请求。 |
| slave-read-only | yes | 配置从库为只读模式，这个属性只需要在主库上配置，在从库上配置没什么意义的 |
| repl-ping-slave-period | 10 | 从库定时向主库发送Ping的命令 |
| repl-timeout | 60 | 从库定时向主库发送Ping的命令的超时时间 |
| slave-priority | 100 | 如果主库挂了的，会寻找找一个slave-priority最小的一个，来变成主库。 |

# Redis 的配置-安全配置

| 配置项 | 值 | 字段意义 |
| --- | --- | --- |
| requirepass | foobared | redis的密码，可以理解成Oracle数据库的连接密码一样的。 |
| maxclients | 10000 | 最大连接数 |
| maxmemory | <bytes> | 最大内存数，给Redis使用的最大内存数。 |
| maxmemory-policy | volatile-lru<br>volatile-random<br>allkeys->random<br>volatile-ttl<br>noeviction | 当Redis已经存到最大内存数的时候，这个时候 Redis使用的策略，下面是5个策略。<br>volatile-lru -> 根据LRU算法生成的过期时间来删除。<br>allkeys-lru -> 根据LRU算法删除任何key。<br>volatile-random -> 根据过期设置来随机删除key。<br>allkeys->random -> 无差别随机删。<br>volatile-ttl -> 根据最近过期时间来删除（辅以TTL）<br>noeviction -> 谁也不删，直接在写操作时返回错误 |

# Redis 的配置-数据追加配置

| 配置项 | 值 | 字段意义 |
| --- | --- | --- |
| appendonly | no | Redis是异步的把数据写到文件中去的，有时候机器重启会导致正在写入的数据丢失掉的，如果appendonly 属性配置成yes了后，Redis会把数据实时的写到appendonly.aof文件中去，一旦Redis重启的时候,就会加载这个文件的,那样的话,数据就不会丢掉了的. |
| appendfilename | appendonly.aof | 累加文件名字 |
| appendfsync | No<br>Always<br>everysec | always 马上立即写到操作系统的文件中去,准确性最高,但是代价也是最高的,<br>速度比较慢了的,并且是IO开销是非常高的，配置成no不是立刻写，everysec每秒写一次 |
| no-appendfsync-on-rewrite | no | 当Aof log进行重写时，是否写日志时fsync |
| auto-aof-rewrite-percentage | 100 | 当日志文件超过原始的日志文件的多少的时候，就开启重写机制 |
| auto-aof-rewrite-min-size | 64M | 这次写入大小，超过多少大小了后，就开启重写机制 |

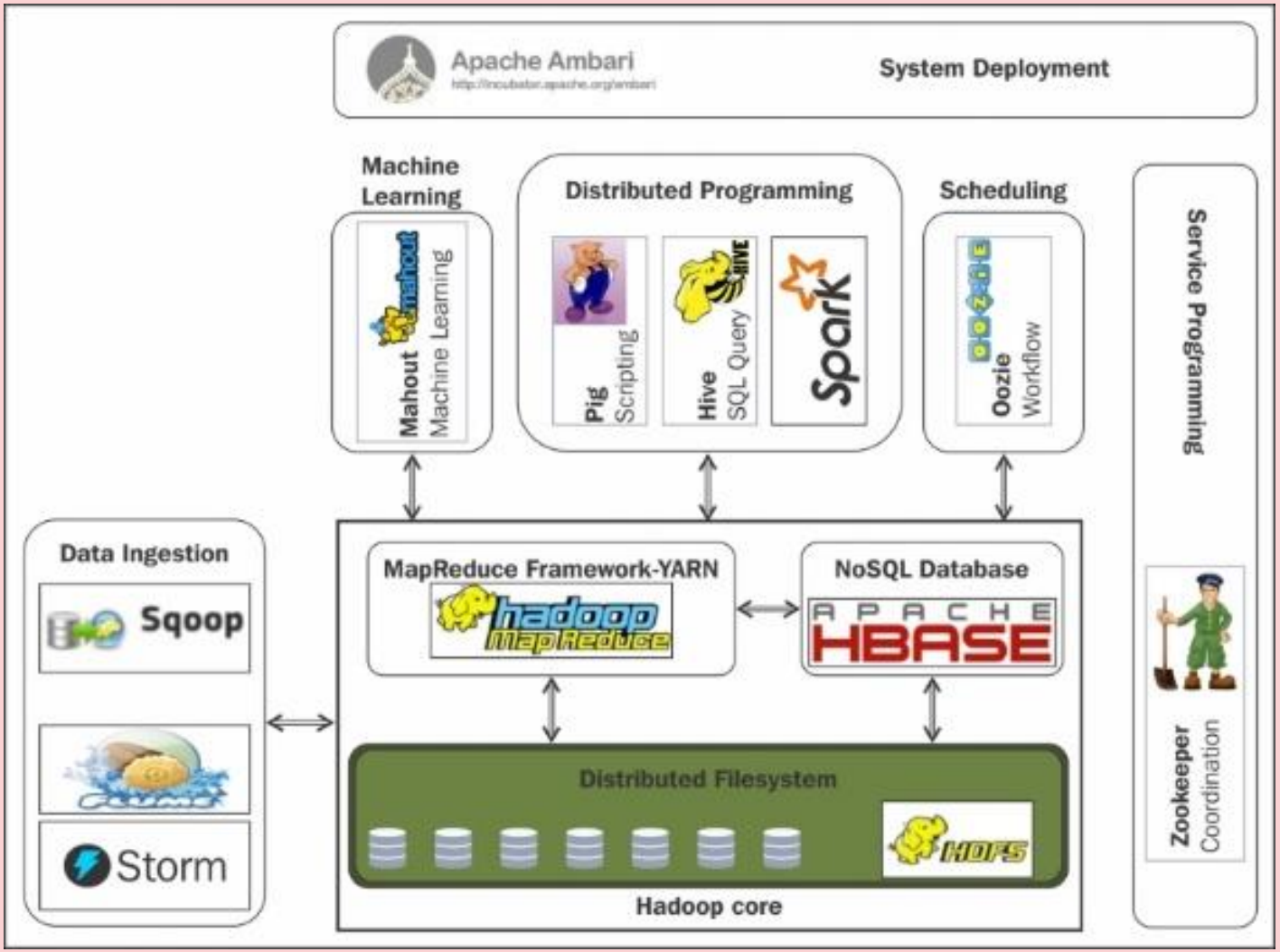
# 4.2 列族数据库

| | |
|---|---|
| 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<br>数据在地理上分布于多个数据中心的应用程序<br>可以容忍副本中存在短期不一致情况的应用程序<br>拥有动态字段的应用程序<br>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） |

# HBase

- Hbase是个分布式的面向列的非结构化数据库，是Hadoop的子项目。
  - HBase是Google Bigtable的开源实现。
  - HBase利用Hadoop HDFS作为其文件存储系统。
  - HBase利用Hadoop MapReduce来处理HBase中的海量数据。
  - HBase利用ZooKeeper跟踪分布式数据的状态。
- 作为存储层来存储非可靠性的海量数据，如日志，评论，访问记录；数据量巨大，千万或者是亿级别以上；实时性要求不高，比如离线计算。
- 特点：高可靠性、高效性、面向列、可伸缩、可在廉价PC Server搭建大规模结构化存储集群。

# Hbase-The Hadoop ecosystem



# 4.3 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<br>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<br>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），灵活性高，复杂性低，数据结构灵活<br>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<br>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） |

# 4.3 文档数据库

“文档”其实是一个数据记录，这个记录能够对包含的数据类型和内容进行“自我描述”。XML文档、HTML文档和JSON 文档就属于这一类。SequoiaDB就是使用JSON格式的文档数据库，它的存储的数据是这样的：

```
{
“ID”:1,
“NAME”: “SequoiaDB”,
“Tel”: {
        “Office”: “123123”, “Mobile”: “132132132”
      }
“Addr”: “China,GZ”
}
```

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）

一个XML文档：

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

# 4.3 文档数据库

```
{
“ID”:1,
“NAME”: “SequoiaDB”,
“Tel”: {
        “Office”: “123123”, “Mobile”: “132132132”
      }
“Addr”: “China,GZ”
}
```

- 数据是不规则的，每一条记录包含了所有的有关“SequoiaDB”的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全移动到其他服务器，因为这条记录的所有信息都包含在里面了，不需要考虑还有信息在别的表没有一起迁移走
- 因为在移动过程中，只有被移动的那一条记录（文档）需要操作，而不像关系型中每个有关联的表都需要锁住来保证一致性，这样一来ACID的保证就会变得更快速，读写的速度也会有很大的提升

# 4.4 图形数据库

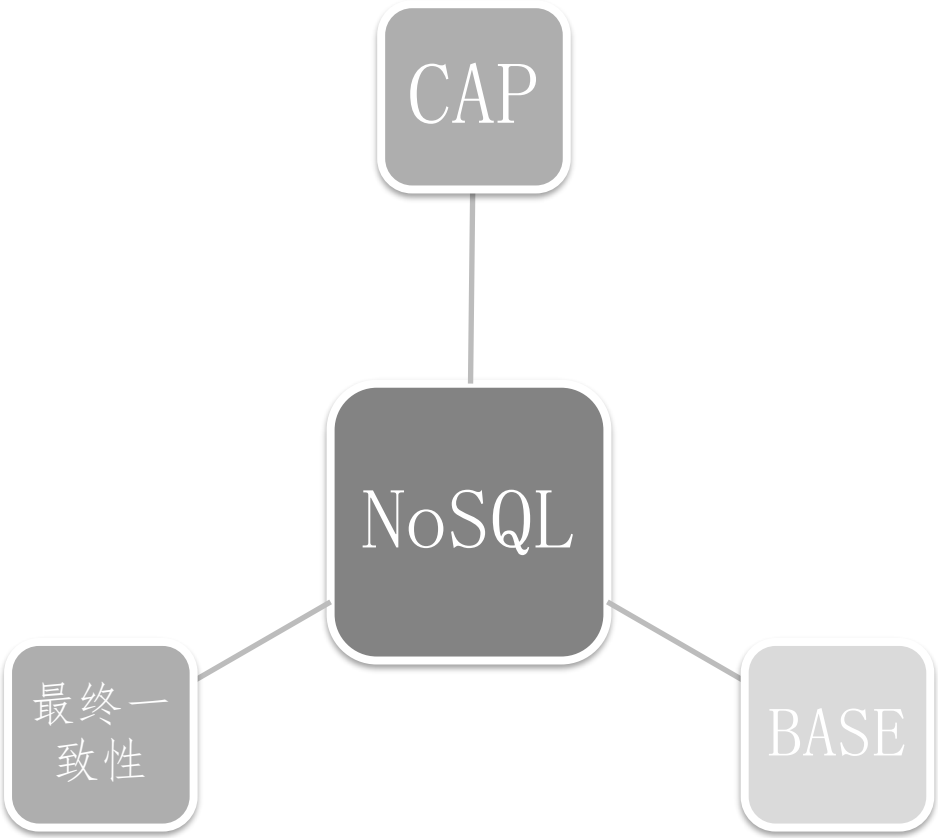
| | |
|---|---|
| 相关产品 | Neo4J、OrientDB、InfoGrid、Infinite Graph、GraphDB |
| 数据模型 | 图结构 |
| 典型应用 | 专门用于处理具有高度相互关联关系的数据，比较适合于社交网络、模式识别、依赖分析、推荐系统以及路径寻找等问题 |
| 优点 | 灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱 |
| 缺点 | 复杂性高，只能支持一定的数据规模 |
| 使用者 | Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J） |

# 4.5 不同类型数据库比较分析

| 中国菜刀 | 瑞士军刀 | 大象兵 | 金箍棒 |
| --- | --- | --- | --- |
| MySQL | MongoDB | HBase | Redis |

- MySQL产生年代较早，而且随着LAMP大潮得以成熟。尽管其没有什么大的改进，但是新兴的互联网使用的最多的数据库
- Redis是键值存储的代表，功能最简单。提供随机数据存储。就像一根棒子一样，没有多余的构造。但是也正是因此，它的伸缩性特别好。就像悟空手里的金箍棒，大可捅破天，小能成缩成针
- HBase是个“仗势欺人”的大象兵。依仗着Hadoop的生态环境，可以有很好的扩展性。但是就像象兵一样，使用者需要养一头大象(Hadoop)，才能驱使他
- MongoDB是个新生事物，提供更灵活的数据模型、异步提交、地理位置索引等五花十色的功能

# 5　NoSQL的三大基石



# 5.1 CAP

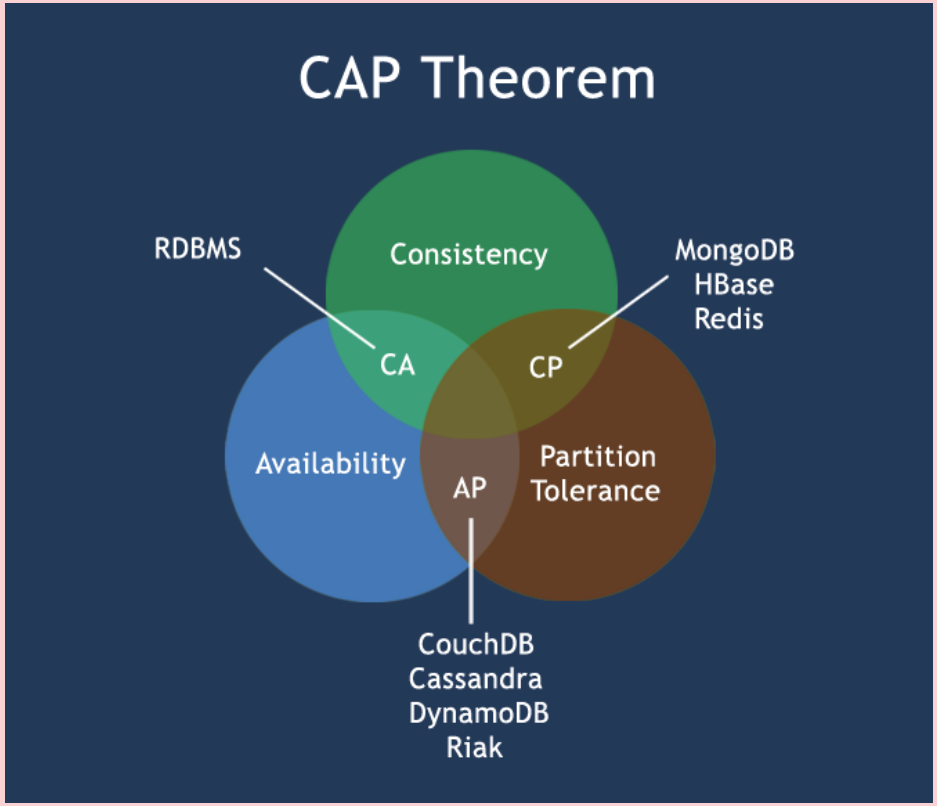
●C（Consistency）：一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据

●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。

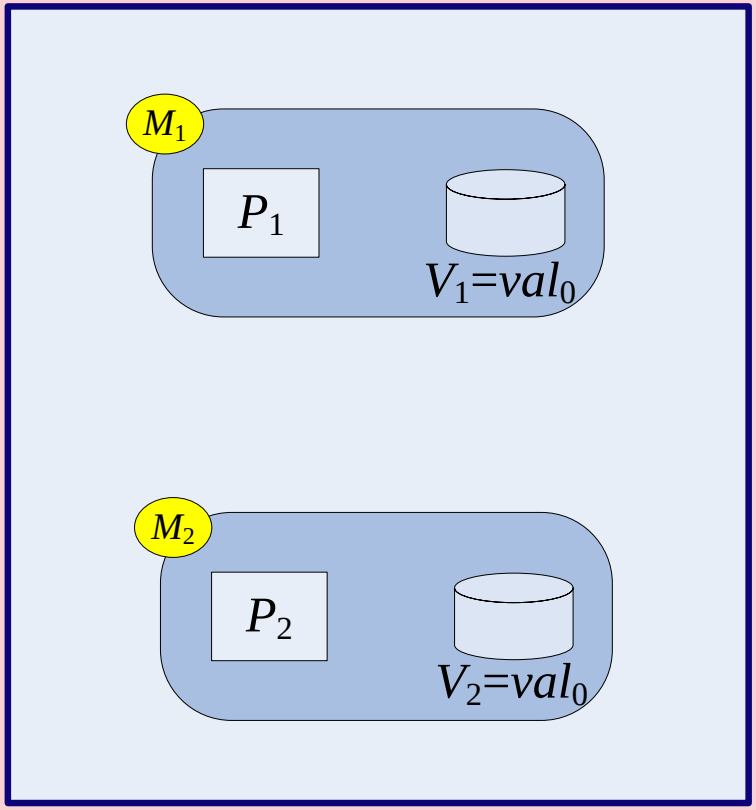
# 5.1 CAP

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”。



# 5.1 CAP

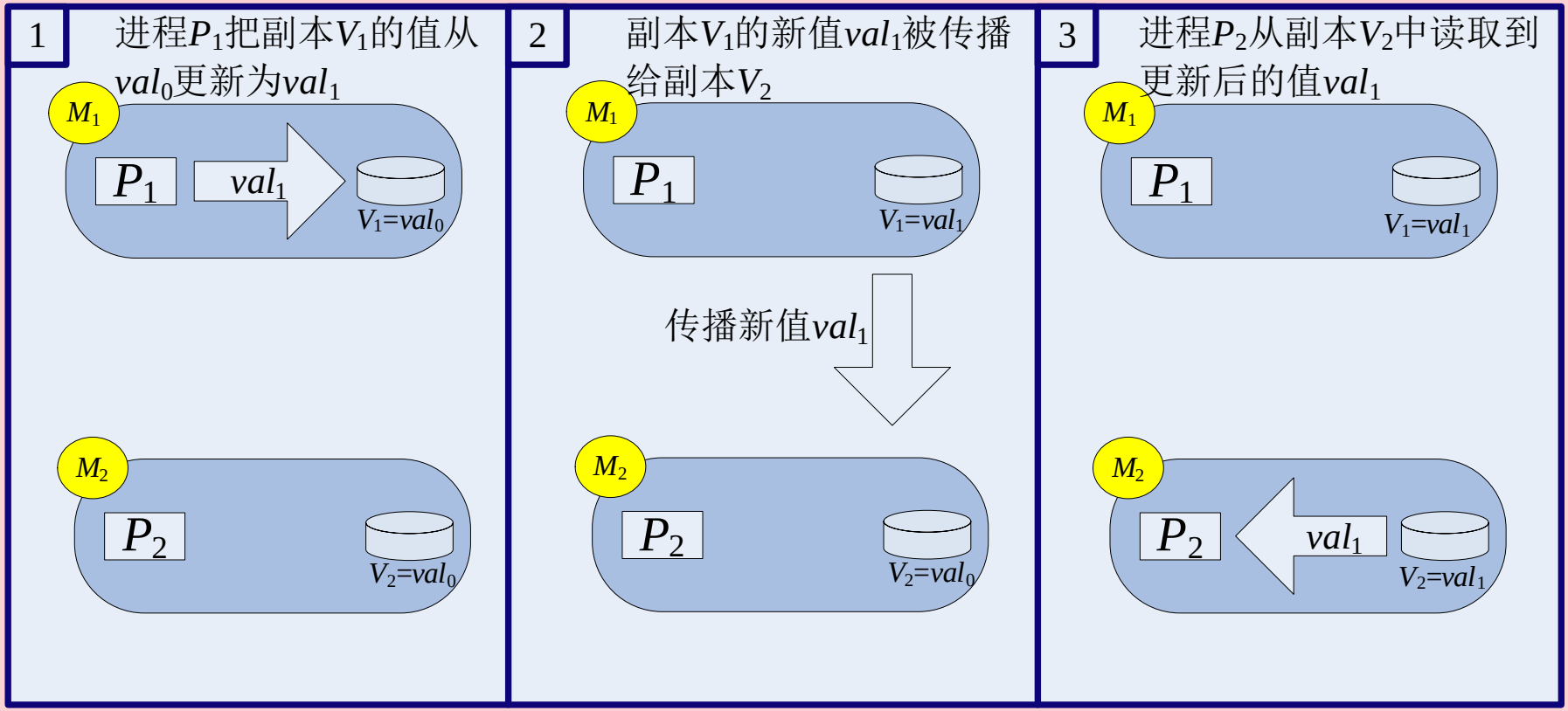
一个牺牲一致性来换取可用性的实例




（a）初始状态

# 5.1 CAP

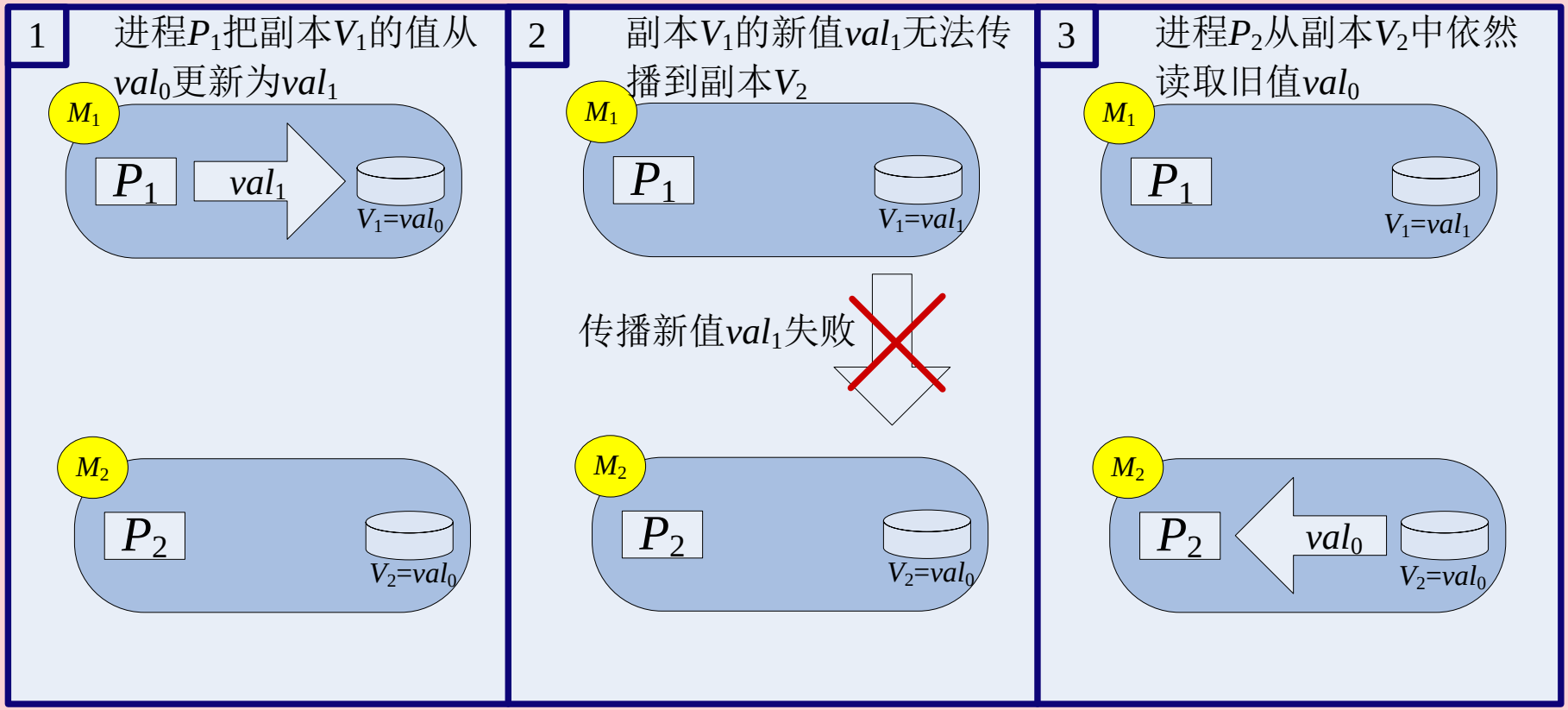
## 一个牺牲一致性来换取可用性的实例




（b）正常执行过程

# 5.1 CAP

## 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

# 5.1 CAP

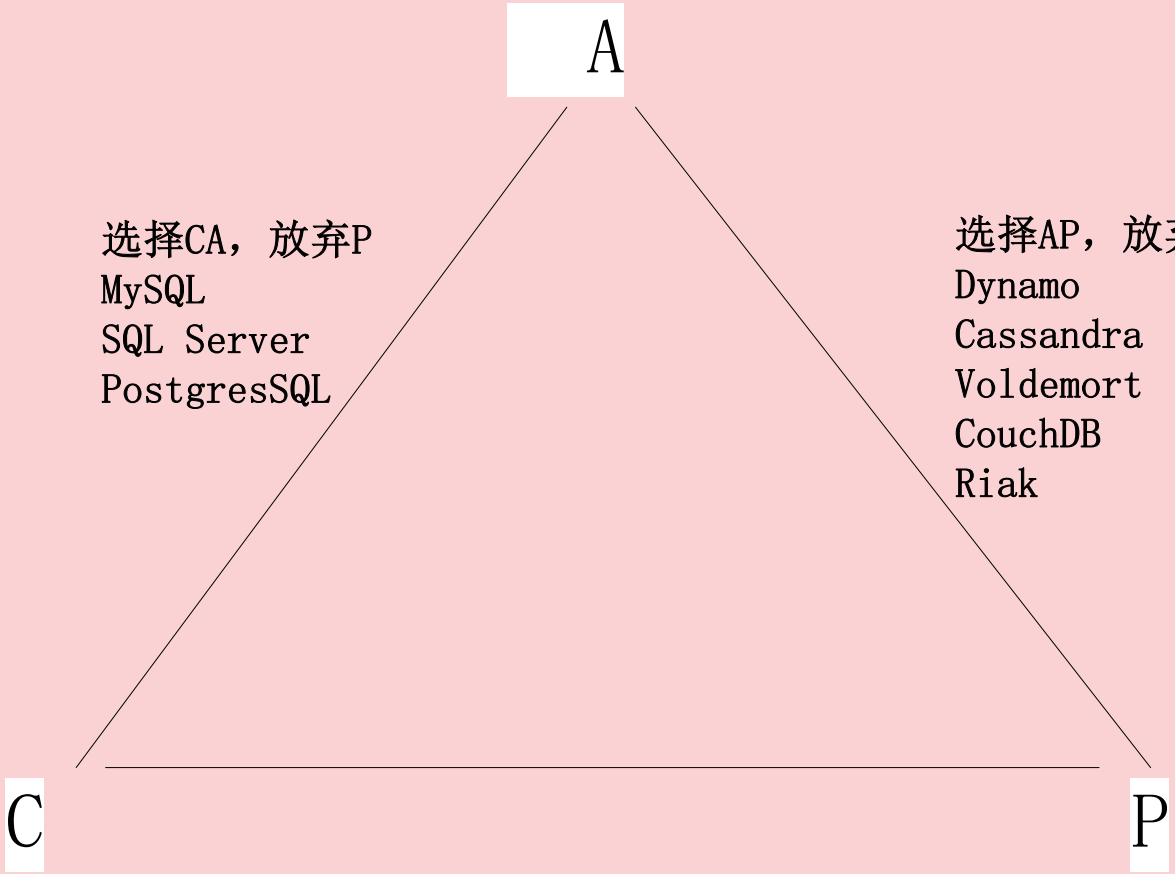
当处理CAP的问题时，可以有几个明显的选择：

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

# 5.1 CAP




不同产品在CAP理论下的不同设计原则

# 5.2 BASE

说起BASE（Basically Availble, Soft-state, Eventual consistency），不得不谈到ACID。

| ACID | BASE |
| --- | --- |
| 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) | |

# 5.2 BASE

●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

# 5.2 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

# 5.2 BASE

- **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。

# 5.3 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值

# 5.3 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“读己之所写”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

# 5.3 最终一致性

**如何实现各种类型的一致性？**

对于分布式数据系统：

- N — 数据复制的份数
- W — 更新数据的时候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

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。

# 5.3 最终一致性

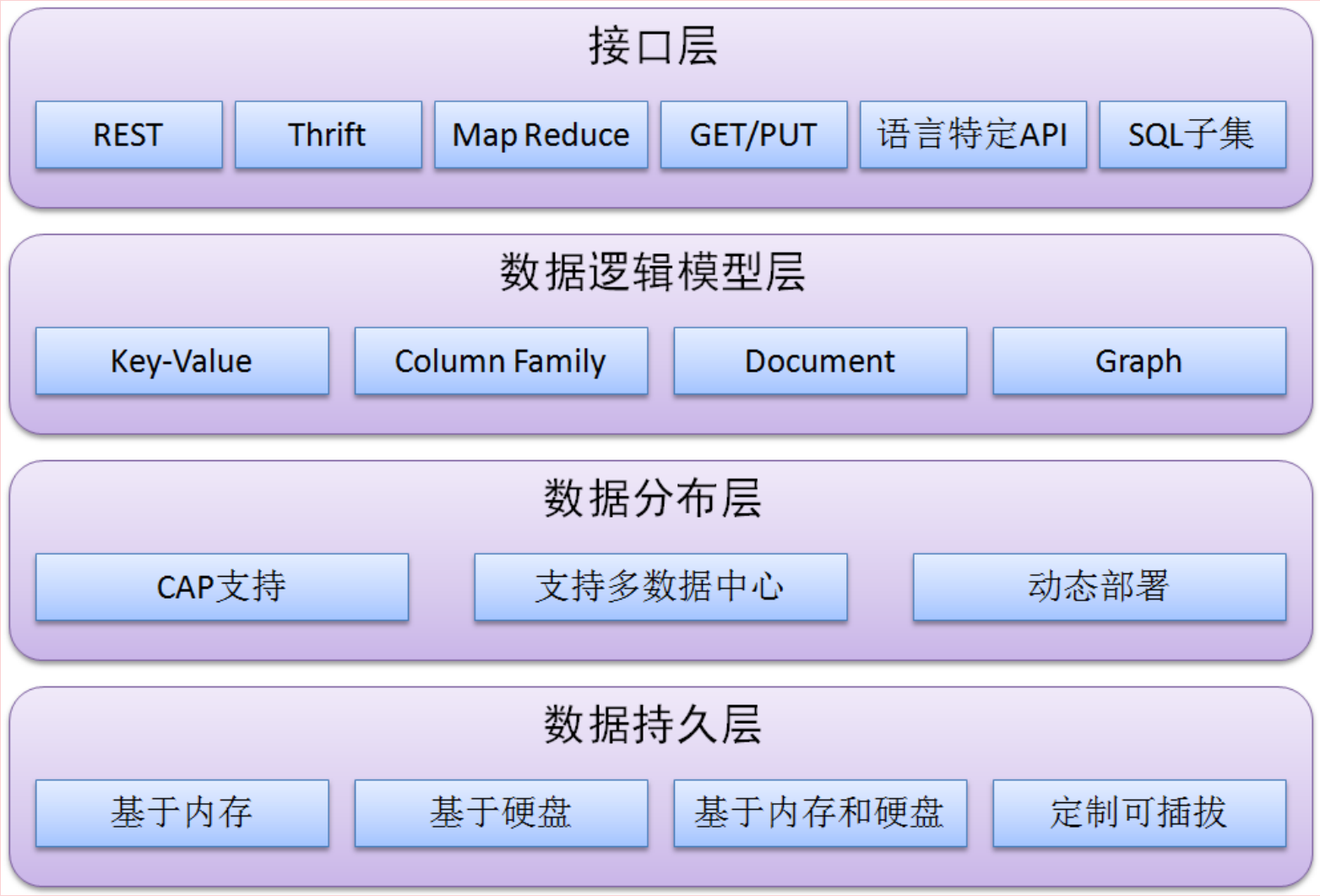
对于分布式系统，为了保证高可用性，一般设置N>=3。不同的N,W,R组合，是在可用性和一致性之间取一个平衡，以适应不同的应用场景。

•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。

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。

•像Voldemort，Cassandra和Riak这些类Dynamo的系统，通常都允许用户按需要设置N，R，W三个值，即使是设置成W＋R<= N也是可以的。也就是说他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。

# 5.4 NoSQL的整体架构



# 接口层（Interfaces）

这层的主要作用是为上层应用提供合适和方便的数据调用接口，而且提供的选择远多于传统的关系型数据库，主要有六大类接口：

•其一是常见的 REST（Representational State Transfer），采用REST的产品有HBase和CouchDB等。

•其二是源自Facebook的RPC协议Thrift，支持Thrift的产品有HBase和Cassandra等。

•其三是用于大规模数据处理的Map Reduce，其相关产品有HBase，CouchDB和MongoDB等。

•其四是类似于Memcached的Get/Put方式，采用Get/Put的产品有Voldemort等。

•其五是提供语言特定（Language Specific）的API，比如Java，在这方面做的不错是MongoDB。

•最后一个是提供SQL的子集，虽然“Join”在NoSQL属于禁忌，但是提供一个SQL的基本子集来方便用户也是一个不错的想法。

# 数据逻辑模型层（Logical Data Model）

这层的主要作用是描述数据的逻辑表现形式，而且与关系型数据库相比NoSQL在逻辑表现形式方面相当灵活，主要有四种形式：

- 最普通的Key-Value形式，这种形式在表现形式是比较单一，但是在扩展方面很有优势，采用Key-Value形式产品的有Voldemort等。
- 列式 （Column Family），这种形式与Key-Value相比其能支持更复杂的数据，但是在扩展方面稍逊一筹，其相关产品有BigTable，HBase和Cassnadra等。
- 文档（Doucument）形式，文档形式源自于著名协作软件Lotus Notes，并且本质上与Key-Value形式非常相似，主要区别在于是那个Value只能存储文档形式的数据，同时文档形式在对复杂数据的支持和扩展 这两方面表现都还可以，采用文档形式的产品有Couch DB和MongoDB等。
- 图（Graph）式，图式的使用场景不是很广，主要是为基于图数据结构的数据“度身定做”的，比如SNS应用中的关系等，其最知名的产品就是Neo4j。

# 数据分布层（Data Distribution Model）

这层的主要作用是定义了数据是如何分布的，和关系型数据库不同的是NoSQL数据库可选择的机制比较多，主要有三种：

•其一是用于水平扩展的CAP机制，支持CAP的产品有HBase，MongoDB和Cassandra等。

•其二是对多数据中心的支持，通过这个机制能够保证横跨多数据中心的NoSQL数据库 能非常平稳地运行，相关的产品有Cassandra等。

•其三是支持动态部署，也就是能在一个生产集群中能动态并且平滑地添加或者删去一个节点

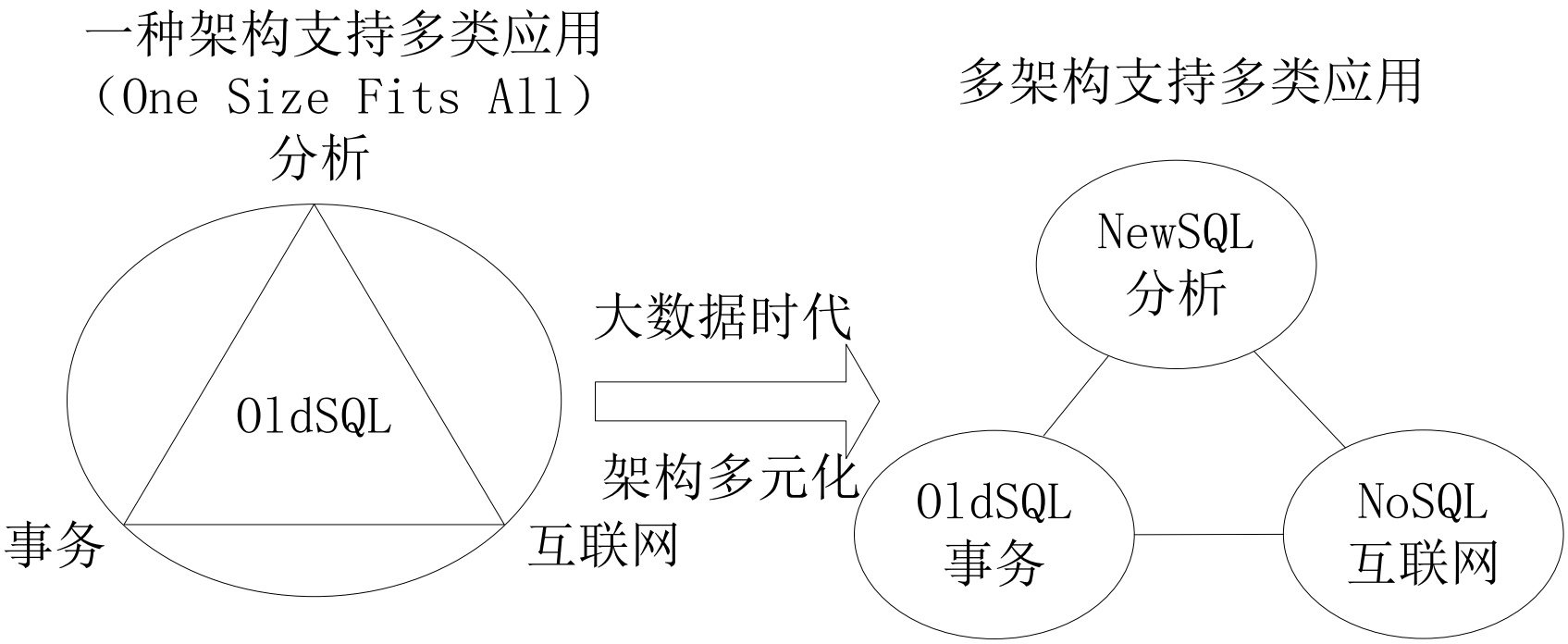
# 数据持久层（Data Persistence）

这层主要作用是定义了数据的存储形式，主要有四种形式：

- 基于内存形式，这种形式速度最快，但是存在丢失数据的可能性，采用内存的有Redis等。
- 基于硬盘形式，这种形式，在数据耐久性方面表现不错，但是在速度方面远不如基于内存的，其相关产品有MongoDB等。
- 基于内存和硬盘的形式，因为这种形式主要结合前面两者的优点，所以其在速度上表现不错，同时数据也不会丢失，而且常被认为是最合适的方案，采用这种形式的产品 有HBase和Cassandra等。
- 定制可插拔（Custom Pluggable）形式，这种形式以灵活著称

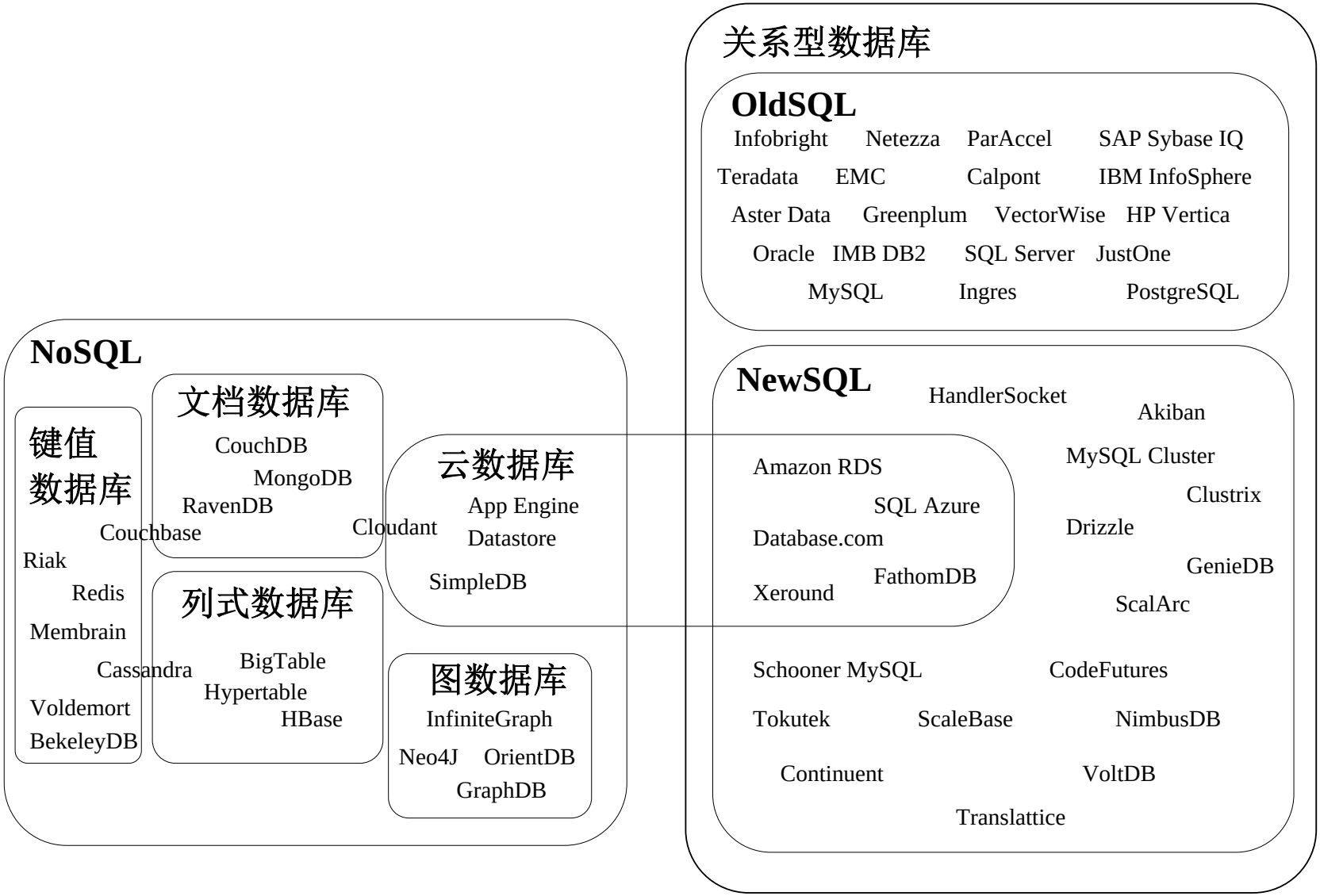
- **注意：**虽然分四层，但并不意味着每个产品只能在每一层选择一个特性，而是可以选择多个特性。比如，在接口层HBase支持REST，Thrift和Map Reduce这三种接口，而在数据分布层，Cassandra即支持CAP，又对多数据中心有所支撑。

# 6　从NoSQL到NewSQL数据库




大数据引发数据处理架构变革

# 6　从NoSQL到NewSQL数据库




关系数据库、NoSQL和NewSQL数据库产品分类图

# 7 文档数据库MongoDB

7.1 MongoDB简介
7.2 MongoDB概念解析
7.3 安装MongoDB
7.4 访问MongoDB


具体请参考网络教程：http://www.runoob.com/mongodb/mongodb-tutorial.html

# 7.1 MongoDB简介

- MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。
- 在高负载的情况下，添加更多的节点，可以保证服务器性能。
- MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。
- 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。

# 7.1 MongoDB简介

## 主要特点

- 提供了一个面向文档存储，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排序
- 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

# 7.2 MongoDB概念解析

在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 |

# 7.2 MongoDB概念解析

通过下图实例，可以更直观的的了解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 "
}
```

# 7.2 MongoDB概念解析

举例：在一个关系型数据库中，一篇博客（包含文章内容、评论、评论的投票）会被打散在多张数据表中。在文档数据库MongoDB中，能用一个文档来表示一篇博客， 评论与投票作为文档数组，放在正文主文档中。这样数据更易于管理，消除了传统关系型数据库中影响性能和水平扩展性的“JOIN”操作。

author:

| pid | tid | name |
|---|---|---|
| 1 | 1 | Jane |

blogposts:

| tid | cid | title |
|---|---|---|
| 1 | 1 | "MyFirstPost" |
| 1 | 2 | "MyFirstPost" |

comments:

| cid | by | text |
|---|---|---|
| 1 | "Abe" | "First" |
| 2 | "Ada" | "Good post" |

# 7.2 MongoDB概念解析

关系数据库中的其中一条记录，在文档数据库MongoDB中的存储方式类似如下：

```
{
 “id”:1,
 “author”:” Jane”,
 “blogposts”:{
                    “tile”:”MyFirstPost”, “comment”:{

 “by”:” Ada”,” text”:”Good post”}
                }
}
```

# 7.2 MongoDB概念解析

## 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

文档是一个键值(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，也是 MongoDB 非常突出的特点。

一个简单的文档例子如下：

{“site”：“dblab.xmu.edu.cn”，“name”：“厦门大学数据库实验室"}

# 7.2 MongoDB概念解析

## 集合

- 集合就是 MongoDB 文档组，类似于 RDBMS 中的表格。
- 集合存在于数据库中，集合没有固定的结构，这意味着你在对集合可以插入不同格式和类型的数据，但通常情况下我们插入集合的数据都会有一定的关联性。

比如，我们可以将以下不同数据结构的文档插入到集合中：

```
{"site":"www.baidu.com"}
{"site":“dblab.xmu.edu.cn”, "name" :“厦门大学数据库实验室"}
{"site":"www.runoob.com","name":"菜鸟教程","num":5}
```

# 7.2 MongoDB概念解析

| 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。 |

# 7.3 安装MongoDB

## Window平台安装MongoDB

MongoDB提供了可用于32位和64位系统的预编译二进制包，你可以从MongoDB官网下载安装，MongoDB预编译二进制包下载地址：
http://www.mongodb.org/downloads
*注意：在 MongoDB2.2 版本后已经不再支持 Windows XP 系统。*

## Linux平台安装MongoDB

MongoDB提供了linux平台上32位和64位的安装包，你可以在官网下载安装包。
下载地址：http://www.mongodb.org/downloads

## 启动 MongoDB服务

只需要在MongoDB安装目录的bin目录下执行'mongod'即可

# 7.4 访问MongoDB

（一） 使用 MongoDB shell访问MongoDB

（二） 使用Java程序访问 MongoDB

# （一）使用 MongoDB shell访问MongoDB

- 使用 MongoDB shell 来连接 MongoDB 服务器

  mongodb://localhost

- 使用用户名和密码连接登陆到指定数据库：

  mongodb://admin:123456@localhost/test

# （一）使用 MongoDB shell访问MongoDB

**MongoDB 创建数据库**

MongoDB 创建数据库的语法格式如下：

```
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。

如果你想查看所有数据库，可以使用 **show dbs** 命令

**创建集合**

MongoDB没有单独创建集合名的shell命令，在插入数据的时候，MongoDB会自动创建对应的集合。

# （一）使用 MongoDB shell访问MongoDB

## MongoDB 插入文档

文档的数据结构和JSON基本一样。
所有存储在集合中的数据都是BSON格式。
BSON是一种类JSON的一种二进制形式的存储格式,简称Binary JSON。

MongoDB 使用 insert() 或 save() 方法向集合中插入文档，语法如下：

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

**实例**

```
>db.col.insert({title: 'MongoDB 教程',
description: 'MongoDB 是一个 Nosql 数据库',
by:  ‘大数据实验室',
url: 'http://bigdb.scnu.edu.cn',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
 })
```

# (二) 使用Java程序访问 MongoDB

MongoDB Java

环境配置

•在Java程序中如果要使用MongoDB，需要确保已经安装了Java环境及MongoDB JDBC 驱动。

•首先必须下载mongo jar包，下载地址：https://github.com/mongodb/mongo-java-driver/downloads，请确保下载最新版本。

•需要将mongo.jar包含在你的 classpath 中

# (二) 使用Java程序访问 MongoDB

## (1) 连接数据库

```
import com.mongodb.MongoClient;
……//这里省略其他需要导入的包

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
                  // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" ,
27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
                System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
                 System.out.println("Authentication: "+auth);
      }catch(Exception e){
              System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
          }
   }}
```

# (二) 使用Java程序访问 MongoDB

（2）创建集合 可以使用com.mongodb.DB类中的createCollection()来创建集合

```
public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
            // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" ,
27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
         System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
         System.out.println("Authentication: "+auth);
         DBCollection coll = db.createCollection("mycol");
         System.out.println("Collection created successfully");
      }catch(Exception e){
            System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
         }
   }
}
```

# (二) 使用Java程序访问 MongoDB

(3) 插入文档 可以使用com.mongodb.DBCollection类的 insert() 方法来插入一个文档

```
public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
          // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
         // 连接到数据库
         DB db = mongoClient.getDB( "test" );
          System.out.println("Connect to database successfully");
         boolean auth = db.authenticate(myUserName, myPassword);
          System.out.println("Authentication: "+auth);
         DBCollection coll = db.getCollection("mycol");
         System.out.println("Collection mycol selected successfully");
         BasicDBObject doc = new BasicDBObject("title", "MongoDB").
            append("description", "database").
            append("likes", 100).
            append("url", "http://www.w3cschool.cc/mongodb/").
            append("by", "w3cschool.cc");
         coll.insert(doc);
         System.out.println("Document inserted successfully");
      }catch(Exception e){
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
        }}}
```

# 本章小结

- 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
- 最后介绍了具有代表性的NoSQL数据库——文档数据库MongoDB