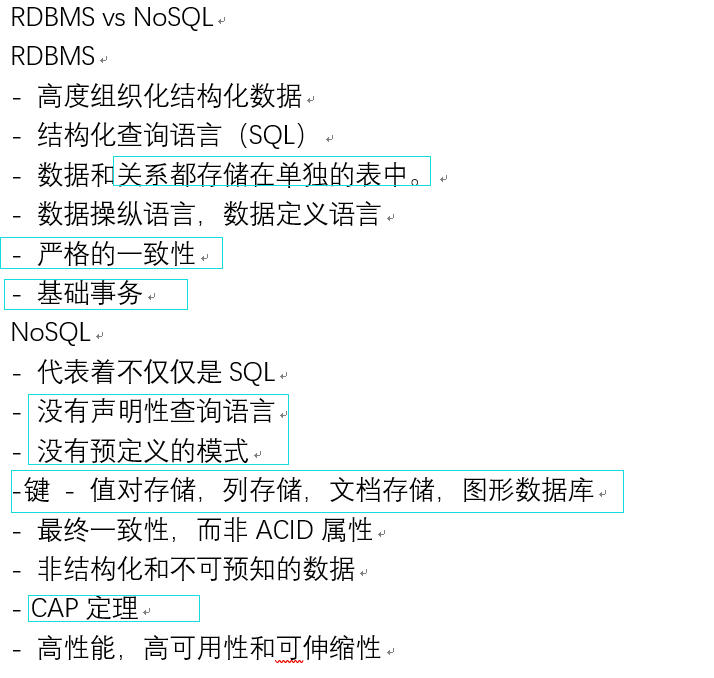
1. MySQL的扩展性瓶颈（大文本，数据，视频，不合适）
2. 
3. 一般MySQL，每次表的更新Cache就失效，是一种大粒度的Cache，在针对web2.0的交互频繁的应用，Cache性能不高。而NoSQL的Cache是记录级的是一种细粒度的Cache，所以NoSQL在这个层面上来说就要性能高很多了
4. 3V+3高（variety,velocity(实时)，volume(海量)）+（高并发+高可扩+高性能）
5. 王坚这样概括“去IOE（去掉IBM的小型机、Oracle数据库、EMC存储设备）”运动和阿里云之间的关系：“去IOE”彻底改变了阿里集团IT架构的基础，是阿里拥抱云计算，产出计算服务的基础。“去IOE”的本质是分布化，让随处可以买到的Commodity PC架构成为可能，使云计算能够落地的首要条件。
6. 各种数据类型的应用场景：

商品的基本信息：mysql

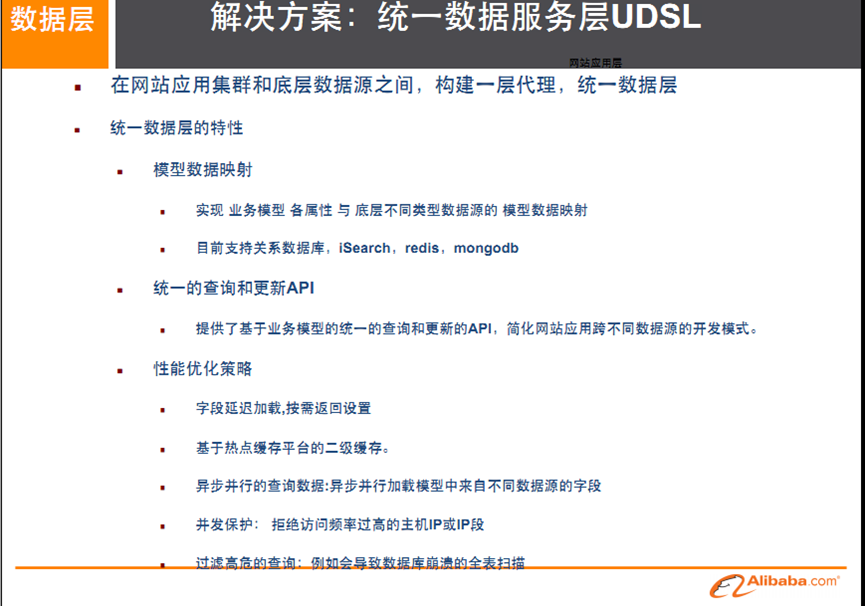
多文字信息描述，文档数据库: mongDB

商品的图片(放在分布式文件系统中)：TFS，GFS，HDFS,hyperbase(即可以存放关系型又可以存放非关系型)

商品关键字：Search

商品的热点高频信息:redis

商品交易，价格计算，积分累计：第三方系统

1. 
2. nosql数据库四大分类（kv（redis），文档(mongodb)，列示（hbase 结构化的分布式系统），图关系数据库（InfoGrid,graph））
3. CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。

CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。

AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。

1. 高并发的操作是不太建议有关联查询的，互联网公司用冗余数据来避免关联查询。分布式事务是支持不了太多的的并发的（分布式，一个模块对应一个数据库，表的查询不可能跨数据库）
2. 列族：最大的特点是方便存储结构化和半结构化数据，方便做数据压缩
3. 
4. base理论（basically available基本可用（响应时间的损失：正常0.5s，基本可用1s）,soft state软状态（硬状态要求多个节点的数据副本都是一致的，软状态允许系统存在中间状态，并认为改状态不影响系统的整体可用性，即允许系统在多个不同节点的数据副本存在数据延时），eventually consistent最终一致性）
5. BASE就是为了解决关系数据库强一致性引起的问题而引起的可用性降低而提出的解决方案。
6. 分布式和集群简介

简单来讲：

1分布式：不同的多台服务器上面部署不同的服务模块（工程），他们之间通过Rpc/Rmi之间通信和调用，对外提供服务和组内协作。

2集群：不同的多台服务器上面部署相同的服务模块，通过分布式调度软件进行统一的调度，对外提供服务和访问。

1. redis内存回收：惰性删除+定时任务删除
2. redis内存溢出策略：默认拒绝写入，LRU删除超时的键，lru删除键，随机删除所有键，随机删除过期键，删除将要过期的键（LRU:最近最少使用(Least Recently Used).替换上次使用距离当前最远的页）
3. rdb就是在指定的时间间隔内将内存中数据集快照写入磁盘，也就是行话讲的snapshot快照，它恢复时时将快照文件直接读到内存里。（Redis会单独创建（fork）一个子进程来进行持久化，会先将数据写入到一个临时文件中，待持久化过程都结束了，再用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何IO操作的，这就确保了极高的性能如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那RDB方式要比AOF方式更加的高效。RDB的缺点是最后一次持久化后的数据可能丢失。（因为最后一次持久化在断电的时候未完成））fork的作用是复制一个与当前进程一样的进程。新进程的所有数据（变量、环境变量、程序计数器等）数值都和原进程一致，但是是一个全新的进程，并作为原进程的子进程，rdb 保存的是dump.rdb文件。（优势：适合大规模数据恢复，对数据完整性和一致性要求不高。劣势：丢失最后一次快照，内存中的数据被克隆了一份）
4. 如何触发rdb快照备份（执行save指令（save时只管保存，其它不管，全部阻塞）

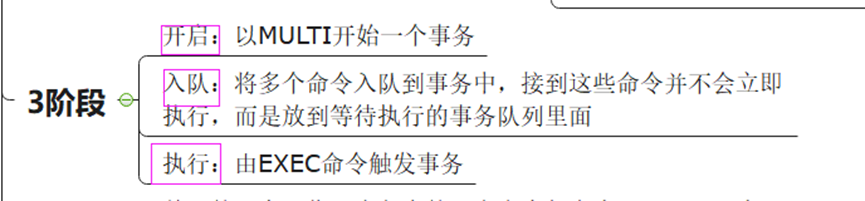
BGSAVE：Redis会在后台异步进行快照操作，快照同时还可以响应客户端请求。可以通过lastsave命令获取最后一次成功执行快照的时间）

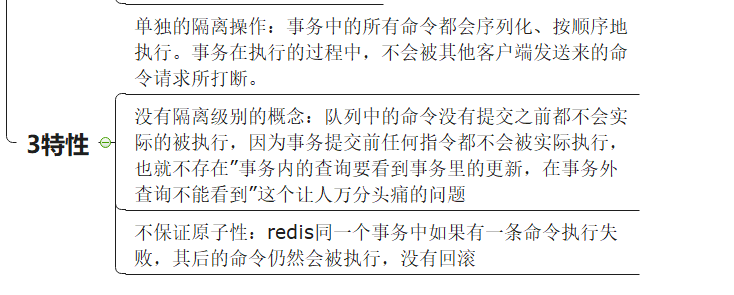
1. aof以日志的形式来记录每个写操作，将Redis执行过的所有写指令记录下来(读操作不记录)，只许追加文件但不可以改写文件，redis启动之初会读取该文件重新构建数据，换言之，redis重启的话就根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作。（劣势:aof文件要远大于rdb文件，恢复速度慢于rdb）
2. rewrite是什么，AOF采用文件追加方式，文件会越来越大为避免出现此种情况，新增了重写机制,当AOF文件的大小超过所设定的阈值时，Redis就会启动AOF文件的内容压缩，只保留可以恢复数据的最小指令集.可以使用命令bgrewriteaof。AOF文件持续增长而过大时，会fork出一条新进程来将文件重写(也是先写临时文件最后再rename)，遍历新进程的内存中数据（意思是AOF重写并不需要对原有AOF文件进行任何的读取，写入，分析等操作，这个功能是通过读取服务器当前的数据库状态来实现的。），每条记录有一条的Set语句。重写aof文件的操作，并没有读取旧的aof文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的aof文件，这点和快照有点类似。
3. 
4. 因为RDB文件只用作后备用途，建议只在Slave上持久化RDB文件，而且只要15分钟备份一次就够了，只保留save 900 1这条规则。

如果Enalbe AOF，好处是在最恶劣情况下也只会丢失不超过两秒数据，启动脚本较简单只load自己的AOF文件就可以了。代价一是带来了持续的IO，二是AOF rewrite的最后将rewrite过程中产生的新数据写到新文件造成的阻塞几乎是不可避免的。只要硬盘许可，应该尽量减少AOF rewrite的频率，AOF重写的基础大小默认值64M太小了，可以设到5G以上。默认超过原大小100%大小时重写可以改到适当的数值。

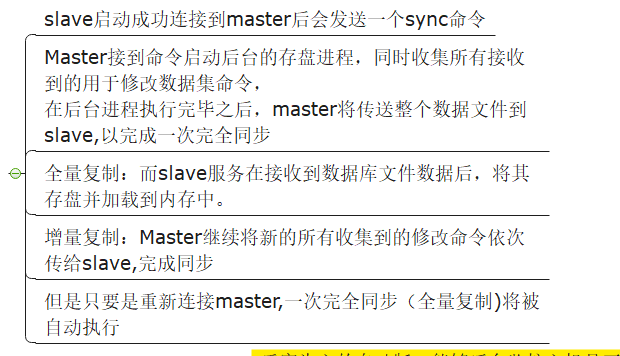
如果不Enable AOF ，仅靠Master-Slave Replication 实现高可用性也可以。能省掉一大笔IO也减少了rewrite时带来的系统波动。代价是如果Master/Slave同时倒掉，会丢失十几分钟的数据，启动脚本也要比较两个Master/Slave中的RDB文件，载入较新的那个。新浪微博就选用了这种架构

1. redis事务是部分支持的（一个队列里的指令只要全部正确可以全部执行成功，如果在输入指令的过程中有一个指令输入错了权力连坐，不执行，但是如果指令对的，执行报错就没办法了）
2. redis事务的三个阶段



1. watch监控相当于乐观锁
2. 
3. redis复制常用三招（一主二从，主机写，从机读，主机挂了只能读不能写，从机不会取代主机）



1. 
2. 主从复制的缺点（复制延时）
3. redis-java代码（jedis）