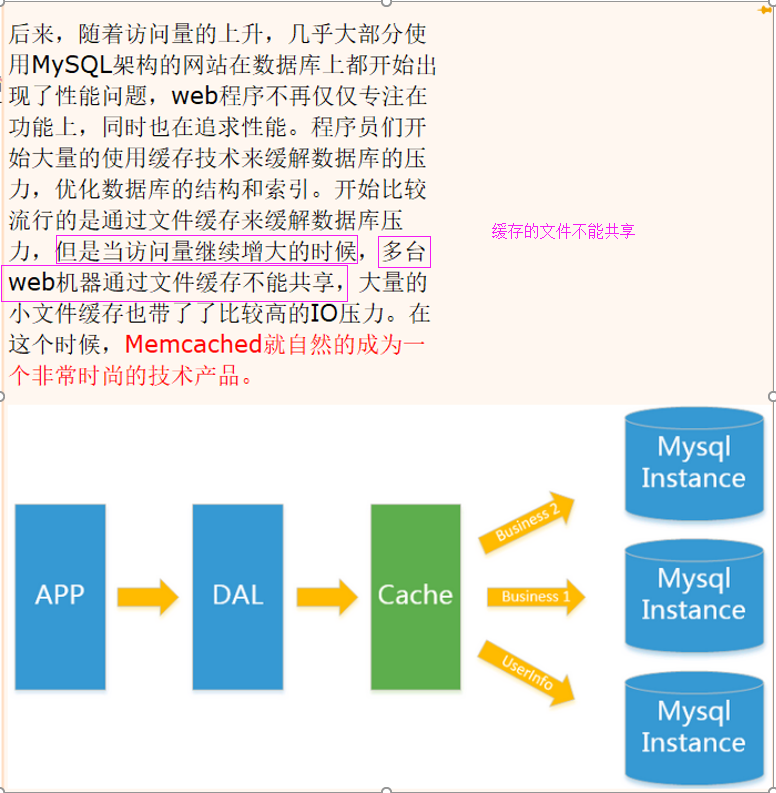
# 01redis\_NoSql入门概述

## 互联网时代背景下大机遇，为什么用nosql

### 1.单机MySql的美好年代



### 2.Memcached(缓存)+MySQL+垂直拆分

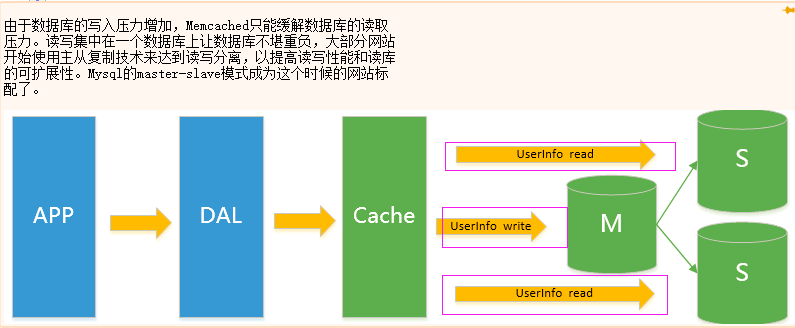


Memcached作为一个独立的分布式的缓存服务器，为多个web服务器提供了一个共享的高性能缓存服务，在Memcached服务器上，又发展了根据hash算法来进行多台Memcached缓存服务的扩展，然后又出现了一致性hash来解决增加或减少缓存服务器导致重新hash带来的大量缓存失效的弊端

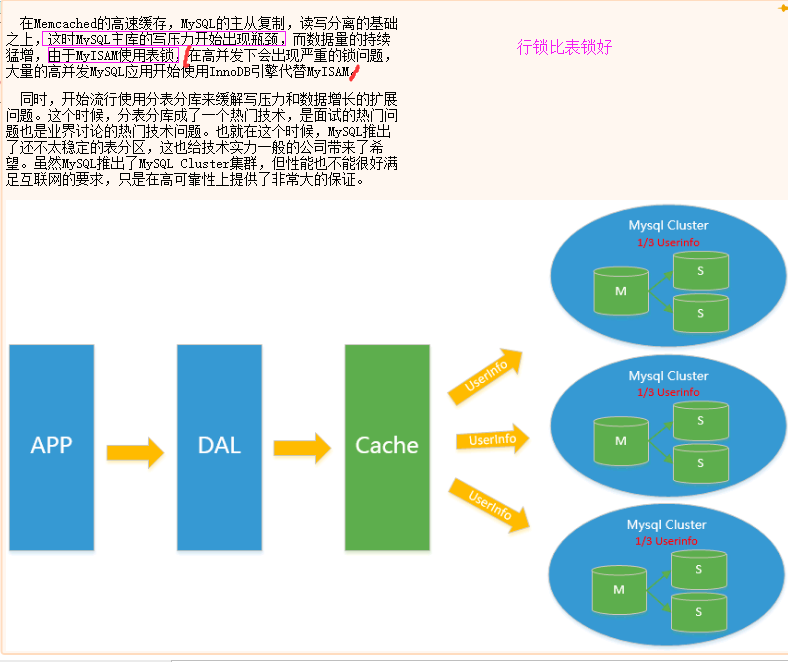
### 3.Mysql主从读写分离

主从复制（为了容灾备份）

读写分离



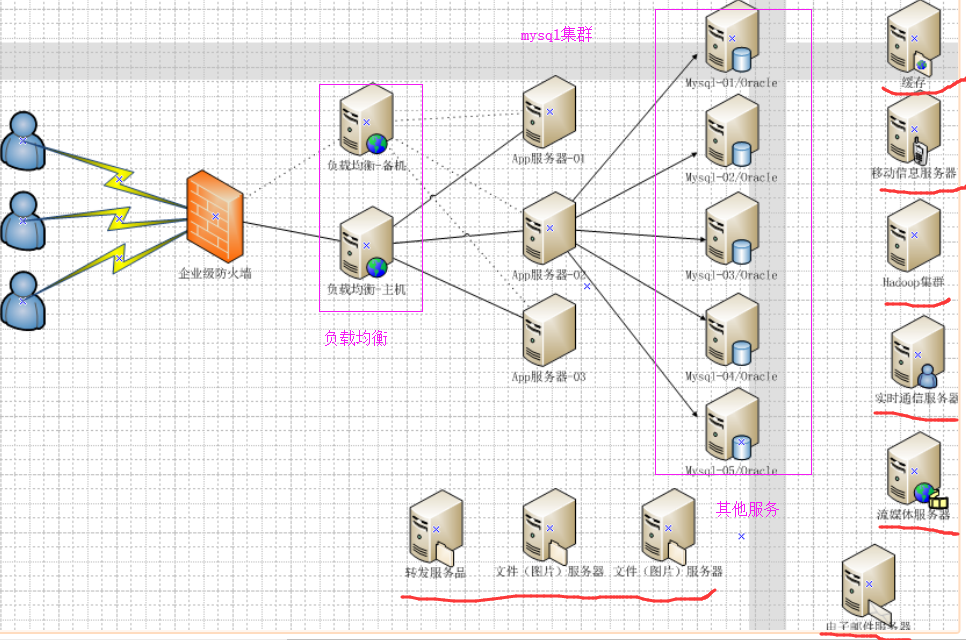
### 4.分表分库+水平拆分+mysql集群



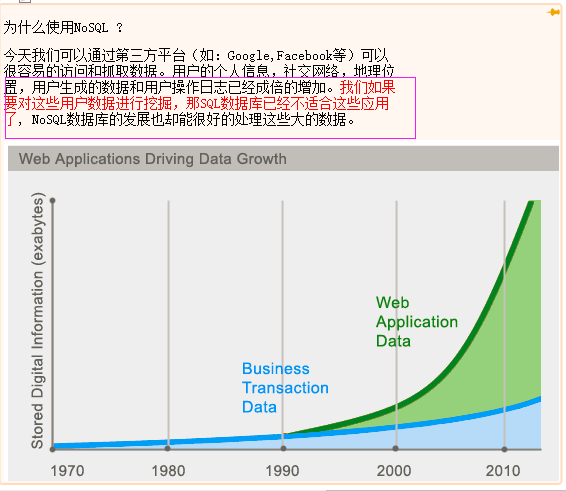
### 5.MySQL的扩展性瓶颈（大文本，数据，视频，不合适）

MySQL数据库也经常存储一些大文本字段，导致数据库表非常的大，在做数据库恢复的时候就导致非常的慢，不容易快速恢复数据库。比如1000万4KB大小的文本就接近40GB的大小，如果能把这些数据从MySQL省去，MySQL将变得非常的小。关系数据库很强大，但是它并不能很好的应付所有的应用场景。MySQL的扩展性差（需要复杂的技术来实现），大数据下IO压力大，表结构更改困难，正是当前使用MySQL的开发人员面临的问题。

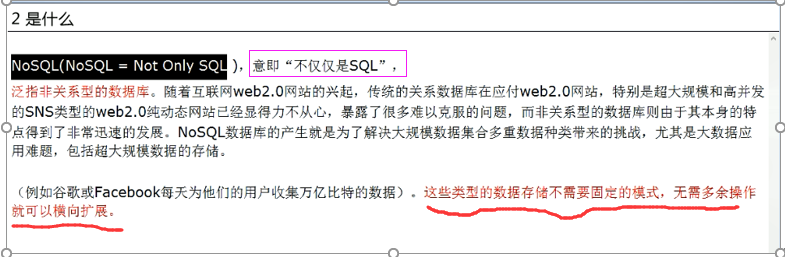
### 6.今天是什么样子



### 7.为什么用NoSql



## Nosql是什么



# 02 NoSQL入门概述下

对redis的理解=kv+cache+persistence

## NoSql的特性

### 易扩展

NoSQL数据库种类繁多，但是一个共同的特点都是去掉关系数据库的关系型特性。

数据之间无关系，这样就非常容易扩展。也无形之间，在架构的层面上带来了可扩展的能力。

### 大数据量高性能

NoSQL数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。

这得益于它的无关系性，数据库的结构简单。

一般MySQL使用Query Cache，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

### 多样灵活的数据模型

NoSQL无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，

增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。

### 传统EDBMS VS NOSQL

RDBMS vs NoSQL

RDBMS

- 高度组织化结构化数据

- 结构化查询语言（SQL）

- 数据和关系都存储在单独的表中。

- 数据操纵语言，数据定义语言

- 严格的一致性

- 基础事务

NoSQL

- 代表着不仅仅是SQL

- 没有声明性查询语言

- 没有预定义的模式

-键 - 值对存储，列存储，文档存储，图形数据库

- 最终一致性，而非ACID属性

- 非结构化和不可预知的数据

- CAP定理

- 高性能，高可用性和可伸缩性

## 3V+3高

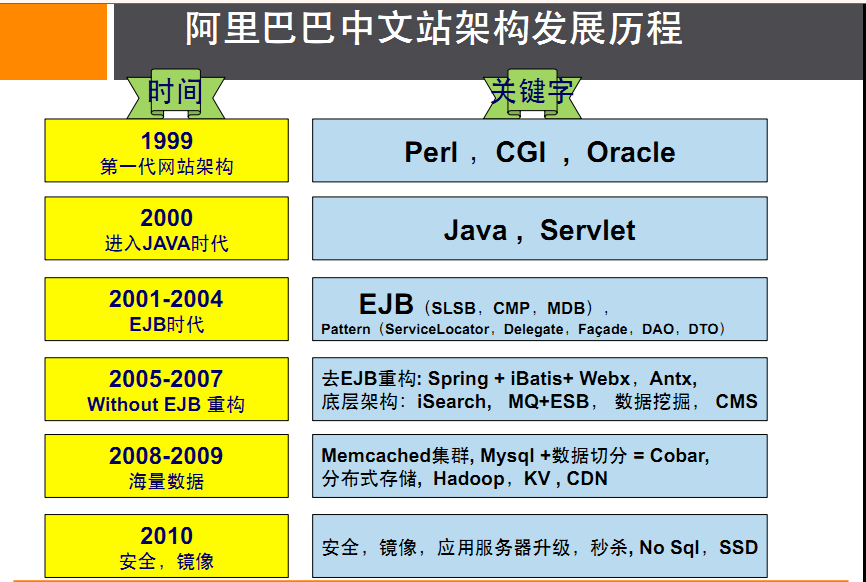


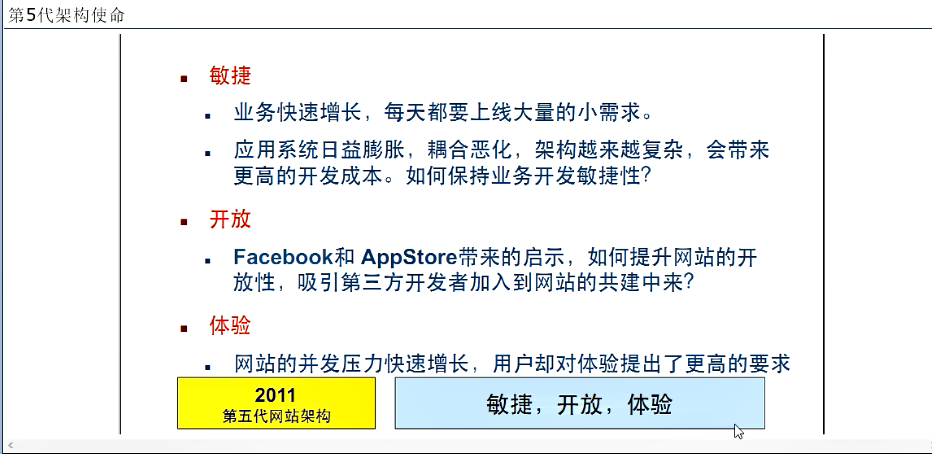
# 03当下nosql应用场景（当下的应用是sql和nosql一起用）

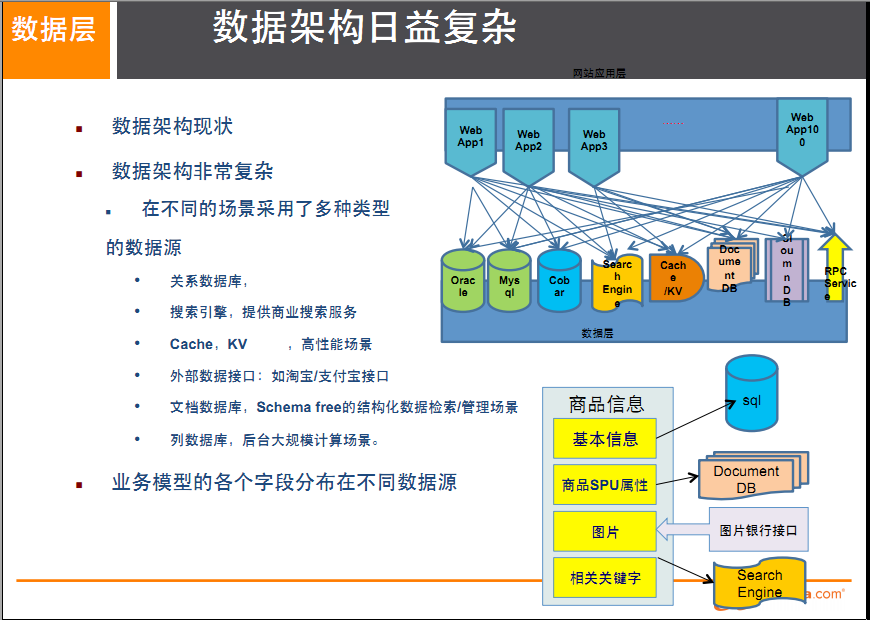
书：淘宝技术10年

2016年4月1日下，淘宝推出全新购物方式Buy+。Buy+使用VirtualReality(虚拟现实)技术，利用计算机图形系统和辅助传感器，生成可交互的三维购物环境。

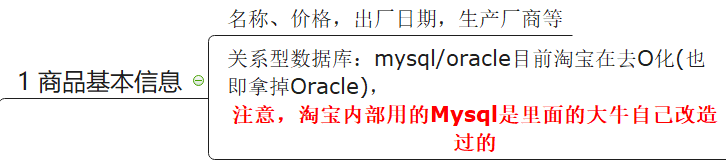
## \*\*\*看看阿里巴巴中文网站首页以女装/女包包为例







### 1商品的基本信息



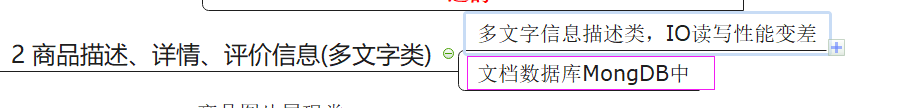
#### 去IOE(王坚)

2008年，王坚加盟阿里巴巴成为集团首席架构师，即现在的首席技术官。这位前微软亚洲研究院常务副院长被马云定位为：将帮助阿里巴巴集团建立世界级的技术团队，并负责集团技术架构以及基础技术平台搭建。

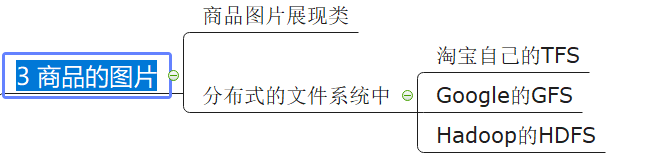
在加入阿里后，带着技术基因和学者风范的王坚就在阿里巴巴集团提出了被称为“去IOE”（在IT建设过程中，去除IBM小型机、Oracle数据库及EMC存储设备）的想法，并开始把云计算的本质，植入阿里IT基因。

王坚这样概括“去IOE”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。

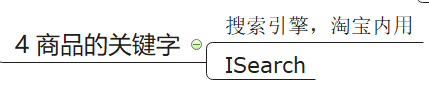
### 2 商品描述、详情、评价信息(多文字类)



### 3 商品的图片



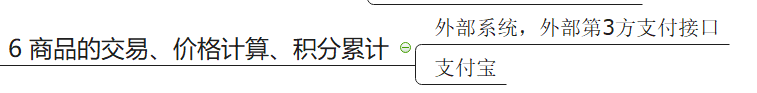
### 4 商品的关键字



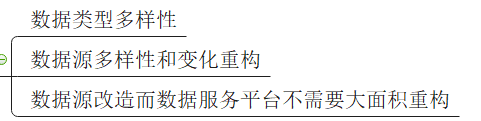
### 5 商品的波段性的热点高频信息(要搜的数据你要准备好)



### 6.商品的交易、价格计算、积分累计



## 难点



我们不要大面积重构

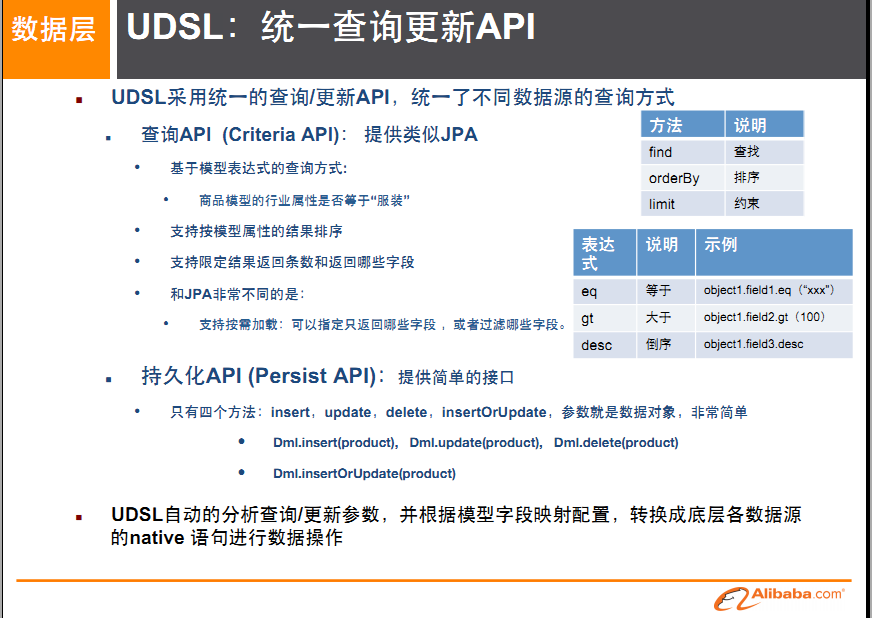
## \*\*\*解决方法(介绍EAI和统一数据平台服务层)

面对这么多数据库（redis Isearch mongodb Memcached oracle）的调用，阿里整了一套统一数据平台服务层（UDSL Unified data services layer）。

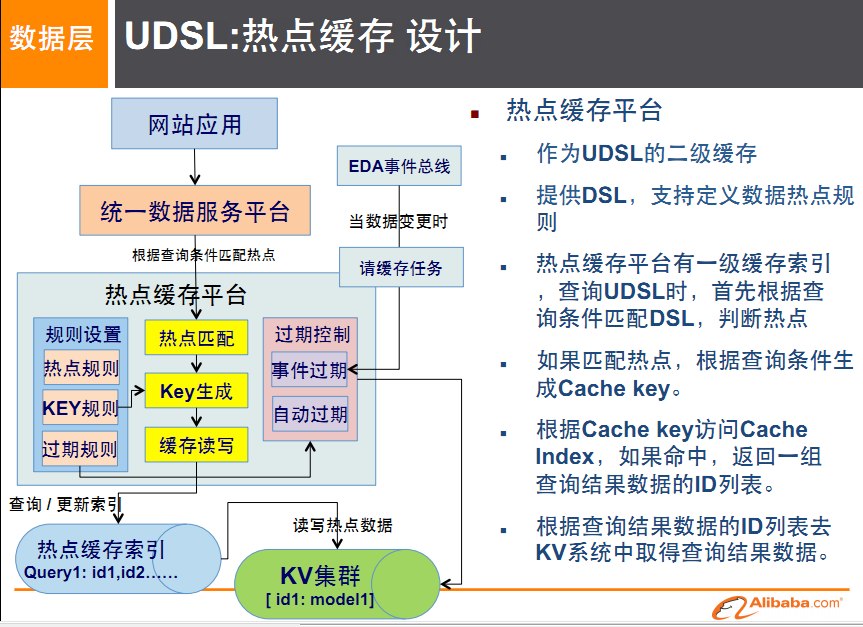
### 映射



### API



### 热点缓存



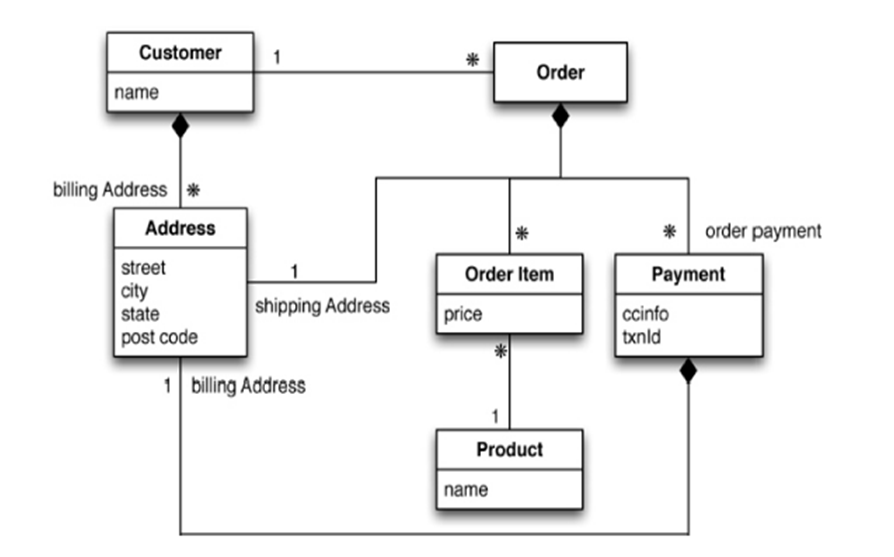
# 04Redis\_NoSql数据模型简介

## 什么是BSON

BSON（）是一种类json的一种二进制形式的存储格式，简称Binary JSON，

它和JSON一样，支持内嵌的文档对象和数组对象

## 传统关系型数据库



## 使用BSon画出的数据模型

{

"customer":{

"id":1136,

"name":"Z3",

"billingAddress":[{"city":"beijing"}],

"orders":[

{

"id":17,

"customerId":1136,

"orderItems":[{"productId":27,"price":77.5,"productName":"thinking in java"}],

"shippingAddress":[{"city":"beijing"}]

"orderPayment":[{"ccinfo":"111-222-333","txnid":"asdfadcd334","billingAddress":{"city":"beijing"}}],

}

]

}

}

## 提示：

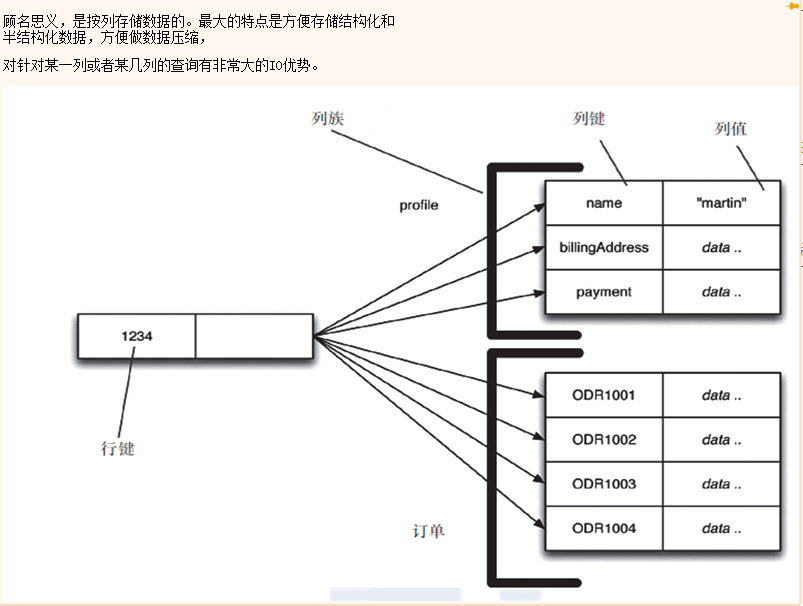
高并发的操作是不太建议有关联查询的，互联网公司用冗余数据来避免关联查询。分布式事务是支持不了太多的的并发的（分布式，一个模块对应一个数据库，表的查询不可能跨数据库）

## 聚合模型（关系数据库中有char int等数据类型）

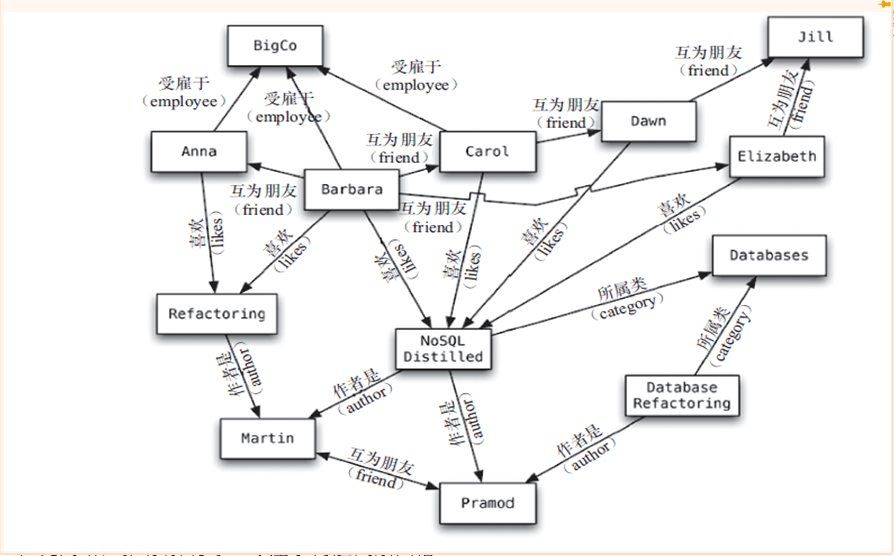
KV键值

bson

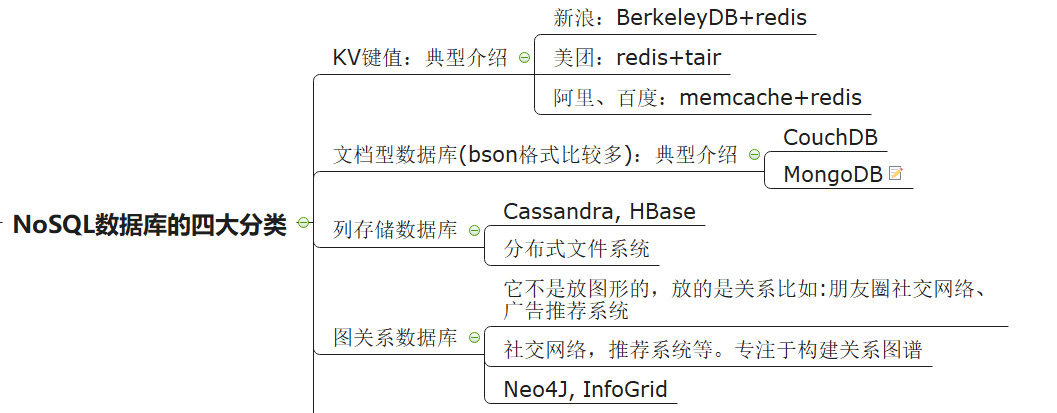
### 列族



### 图形



# 05\*\*NoSql数据库四大分类



## 四者对比



# 06\*\*\*\*\*分布式数据库CAP原理

## 传统的数据库特性是ACID

关系型数据库遵循ACID规则

事务在英文中是transaction，和现实世界中的交易很类似，它有如下四个特性：

1、A (Atomicity) 原子性

原子性很容易理解，也就是说事务里的所有操作要么全部做完，要么都不做，事务成功的条件是事务里的所有操作都成功，只要有一个操作失败，整个事务就失败，需要回滚。比如银行转账，从A账户转100元至B账户，分为两个步骤：1）从A账户取100元；2）存入100元至B账户。这两步要么一起完成，要么一起不完成，如果只完成第一步，第二步失败，钱会莫名其妙少了100元。

2、C (Consistency) 一致性

一致性也比较容易理解，也就是说数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束。

3、I (Isolation) 独立性

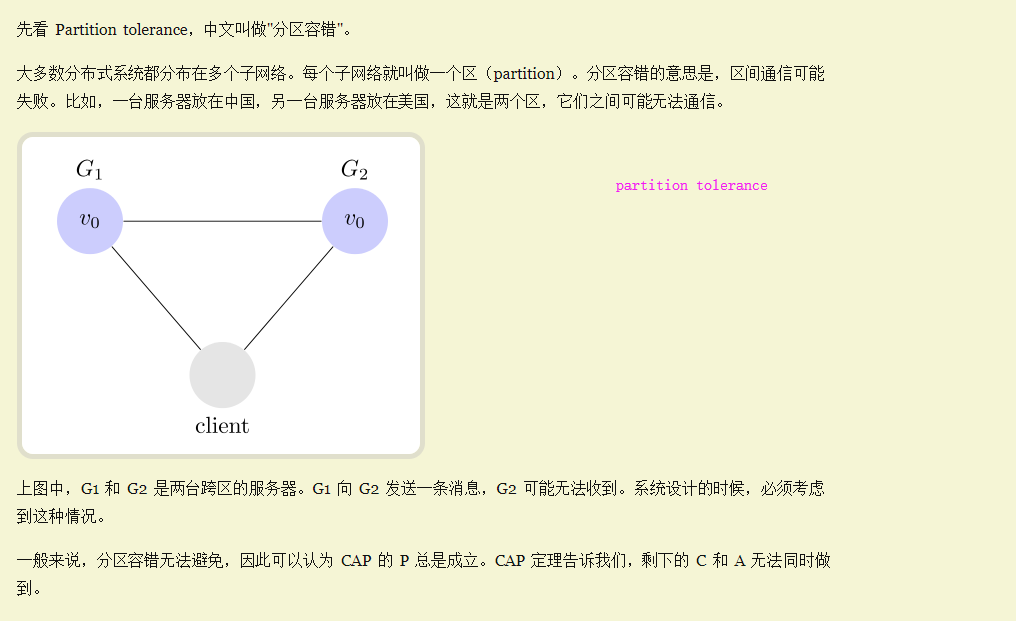
所谓的独立性是指并发的事务之间不会互相影响，如果一个事务要访问的数据正在被另外一个事务修改，只要另外一个事务未提交，它所访问的数据就不受未提交事务的影响。比如现有有个交易是从A账户转100元至B账户，在这个交易还未完成的情况下，如果此时B查询自己的账户，是看不到新增加的100元的

4、D (Durability) 持久性

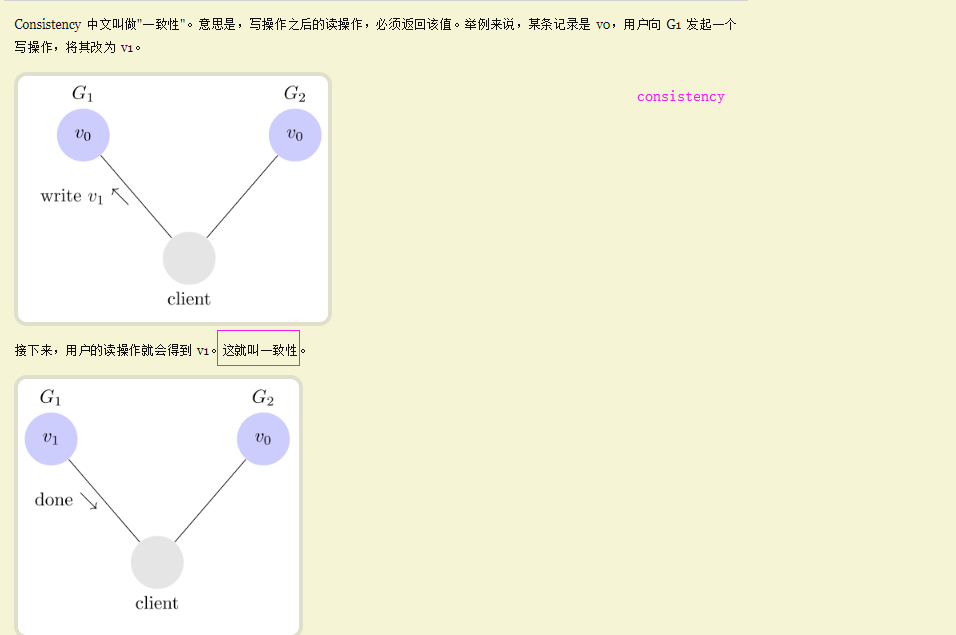
持久性是指一旦事务提交后，它所做的修改将会永久的保存在数据库上，即使出现宕机也不会丢失。

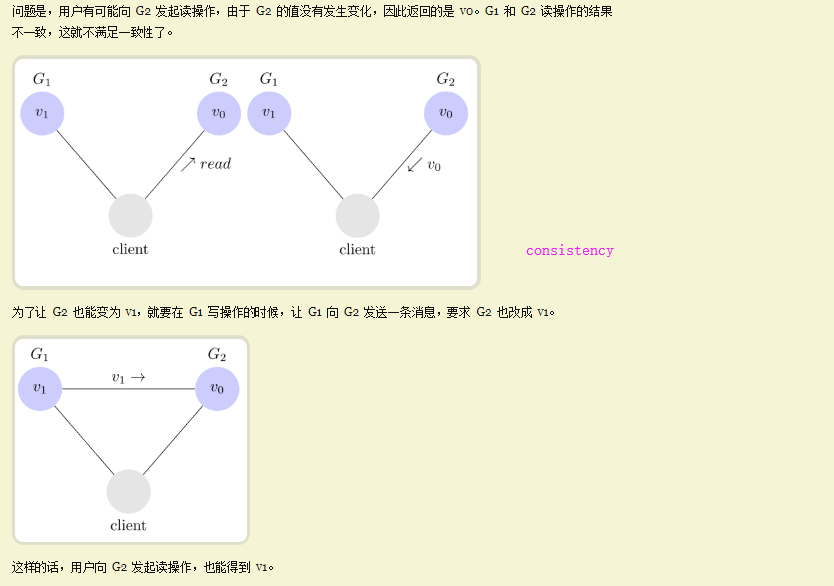
## 非关系数据库是CAP(Consistency强一致性，Availability可用性，Partition tolerance分区容错性)

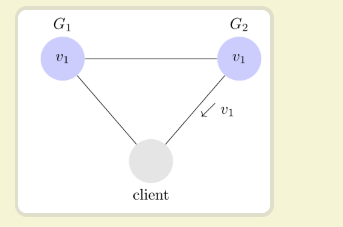
### Partition tolerance



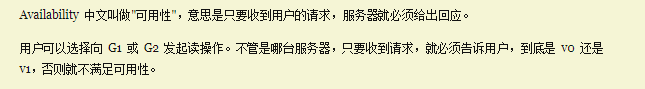
### consistency







### availability



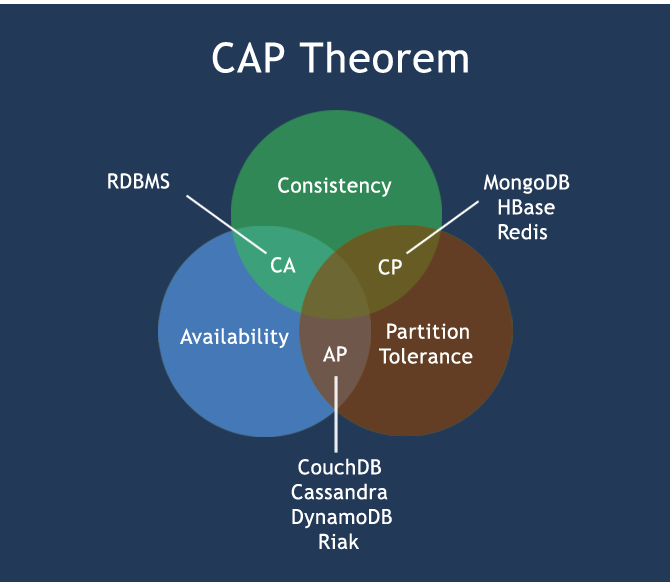
## 经典的CAP图

CAP理论就是说在分布式存储系统中，最多只能实现上面的两点。

而由于当前的网络硬件肯定会出现延迟丢包等问题，所以

分区容忍性是我们必须需要实现的。

所以我们只能在一致性和可用性之间进行权衡，没有NoSQL系统能同时保证这三点。



## CAP的3进2

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，

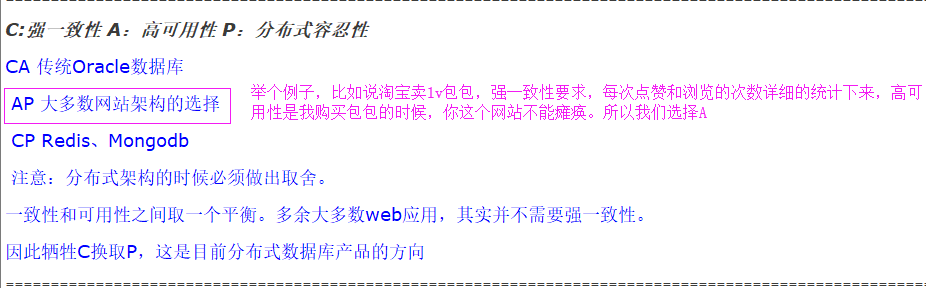
最多只能同时较好的满足两个。

因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

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。

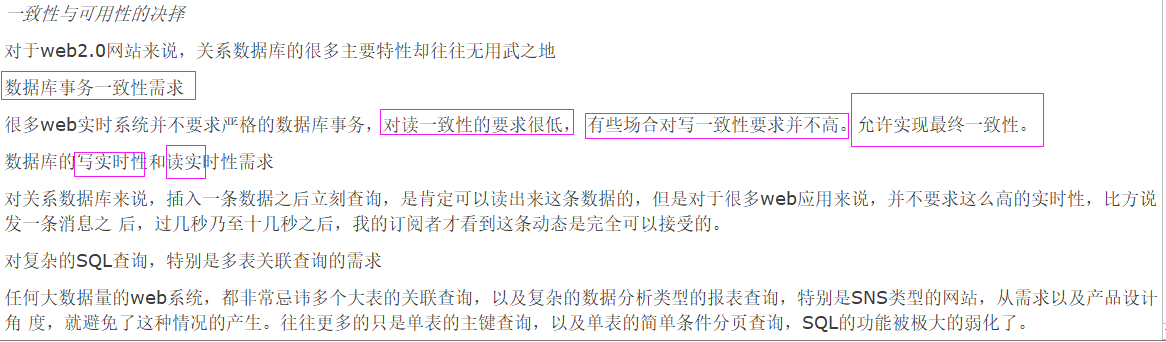
CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。

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。



大多数网站都是弱一致性+ap（确保最终的一致性）

nosql的作用是帮mysql和oracle数据库减负，我们必须确保最终数据的一致性，所以nosql是cp



## \*\*Base

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。

BASE其实是下面三个术语的缩写：

基本可用（Basically Available）

软状态（Soft state）

最终一致（Eventually consistent）

它的思想是通过让系统放松对某一时刻数据一致性的要求来换取系统整体伸缩性和性能上改观。为什么这么说呢，缘由就在于大型系统往往由于地域分布和极高性能的要求，不可能采用分布式事务来完成这些指标，要想获得这些指标，我们必须采用另外一种方式来完成，这里BASE就是解决这个问题的办法

## 分布式+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。

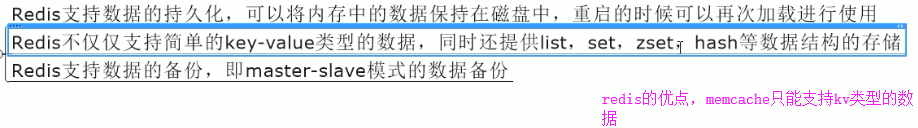
# 07Redis安装

## redis是什么

redis:REmote DIcionary Server（远程字典服务器）

redis是完全开源免费的，用C语言编写的，遵守BSD协议，是一个高性能的(key/value)分布式内存数据库，基于内存运行并支持持久化的NoSQL数据库，是当前最热门的NoSql数据库之一,也被人们称为数据结构服务器

## memcache为什么会被redis取代



## redis能干啥



## 去哪里下

<http://redis.io/> 英文网

<http://www.redis.cn/> 中文网

## 怎么玩

安装过程看思维导图。

# 08Redis\_HelloWorld

测试成功，输入ping回应pong

注意：redis-server是启动服务，redis-cli是测试连通性。

# 09Redis启动后杂项基础知识

redis是一个单进程，单进程处理客户端的请求，对于读写之类的响应，它是通过linux的epoll函数进行包装。意思就是，linux以内核为基础，它为了处理大批量的文件，用epoll包了一层快速的io读写。（单进程，以epoll包装以后，在大批量文件操作中多路复用）



# 10redis\_常用五大数据类型简介

## String（字符串）

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个redis中字符串value最多可以是512M

## Hash（哈希）

Redis hash 是一个键值对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

类似Java里面的Map

## List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素导列表的头部（左边）或者尾部（右边）。

它的底层实际是个链表

## Set（集合）

Redis的Set是string类型的无序集合。它是通过HashTable实现实现的，

## zset(sorted set：有序集合)

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。

redis正是通过分数来为集合中的成员进行从小到大的排序。zset的成员是唯一的,但分数(score)却可以重复。

# \*\*\*\*\*11-16五大数据类型操作看思维导图

redis-server redis.conf的位置 #启动redis-server

redis-cli -h ip -p 端口 #客户端连接测试

redis-cli shutdown #单实例关闭

select id #切换数据库

dbsize #查看当前数据库的key的数量

keys \* #把所有的key罗列一下

keys k? #以通配符的方式

flushdb#清空当前库

flushall #通杀全部库

---------------------

key 键常用

keys \*

exists key的名字#判断某个key是否存在（1就是有，0就是没有）

move key db#移除key到指定库

expire key 秒钟#为给定的key设置过期时间

ttl key #查看还有多少秒过期(-1 标示永不过期，-2表示已过期（过期了就要移除）)

type key#查看你的key是什么类型

-------------------------------------

String 常用

set/get/del/append/strlen

append k value #在k的基础上追加value

strlen k#查看k的value的长度

Incr/decr/incrby/decrby,一定要是数字才能进行加减（k对应的值一定是数字）

getrange k start end #把k对应的value从start到end截取

setrange k index value#在k对应的value的index位置插入value

setex k 秒值 value#设置带过时间的key,动态设置（不能覆盖，k存在返回0，不存在返回1）

mset/mget/msetnx#多个设置

mset k1 v1 k2 v2

mget k1 k2

msetnx k3 v3 k4 v4

msetnx:同时设置一个或多个key-value对，当且仅当所有给定key都不存在（也就是说所有的插入之前是都不存在的）。

getset k newv:将给定key的值设为value,并返回key的旧值

-----------------------------

list常用

lpush mylist1 1 2 3 4#mylist1 中在左边依次插入1 2 3 4

rpush mylist2 1 2 3 4#mylist2 中在右边依次插入1 2 3 4

lrange mylist1 0 -1#mylist1中取出全部 -1表示最后一个

llen mylist1#mylist1的长度

lrem key 2 v1#从left往right删除2个值等于v1的元素，返回的值为实际删除的数量\* LREM list3 0 值，表示删除全部给定的值。零个就是全部值

ltrim key 开始index 结束index#截取指定范围值后再赋值给key

rpoplpush 原列表 目的列表#（原列表rpop,目的列表lpush）

lset key index value#再key中的指定位置做替换

----------------------------

set常用

sadd myset v1 v2 v3#向myset中添加值

smembers myset#查看myset中的元素

sismember myset v1#查看myset中是否有v1,有返回1，没有返回0

scard myset #获取集合里面的元素个数

crem key value#删除集合中元素

srem key value #删除集合中元素

srandmember key# 某个整数（随机出几个数）

spop key 随机出栈

smove key1 key2 value#将key1中的value赋值给key2

sdiff set1 set2#差集

sinter set1 set2#交集

sunion set1 set2#并集

---------------------------

hash常用

kv模式不变，但v是一个键值对 需要根据双key来取值

hset/hget/hmset/hmget/hgetall/hdel

hlen

hexists key

hkeys/hvals

hincrby

hincrbyfloat

hsetnx

------------------

zset（sorted set）常用

在set基础上，加一个score值。之前set是k1 v1 v2 v3，现在zset是k1 score1 v1 score2 v2(score1 v1是一个整体)

zadd zset1 score1 v1 score2 v2#添加

zrange zset1 0 -1 (withscores)#有withscores连值带分一起输出



# 17redis.conf配置文件（熟悉这个配置文件）

linux中配置大于编码

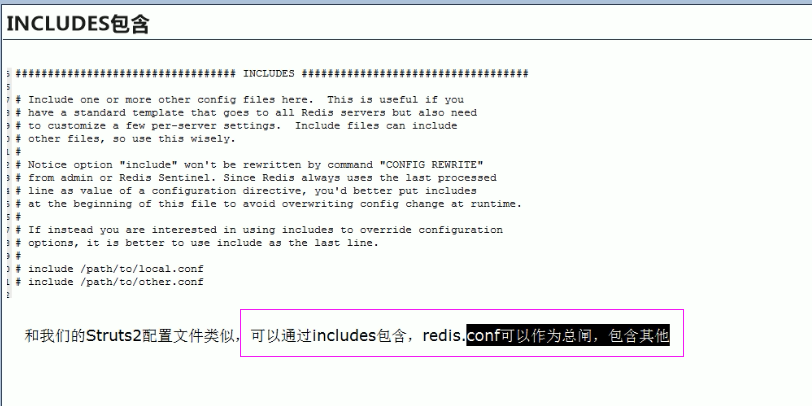


前期在修改配置文件找General，将daemonize 修改为yes

## Units单位

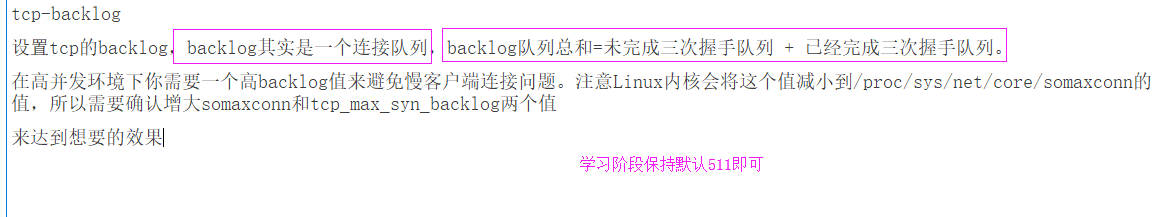


## includes包含



## General通用

### tcp-backlog



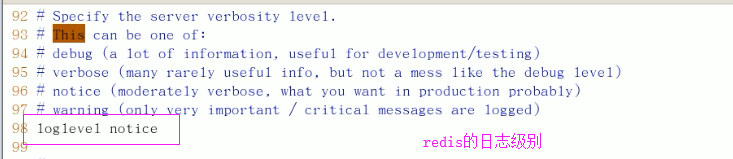
### Tcp-backlog



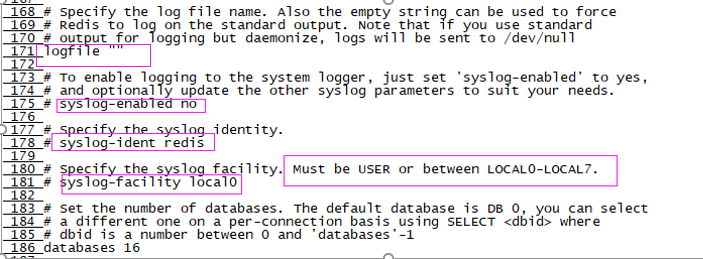
### tcp-keepalive



### loglevel

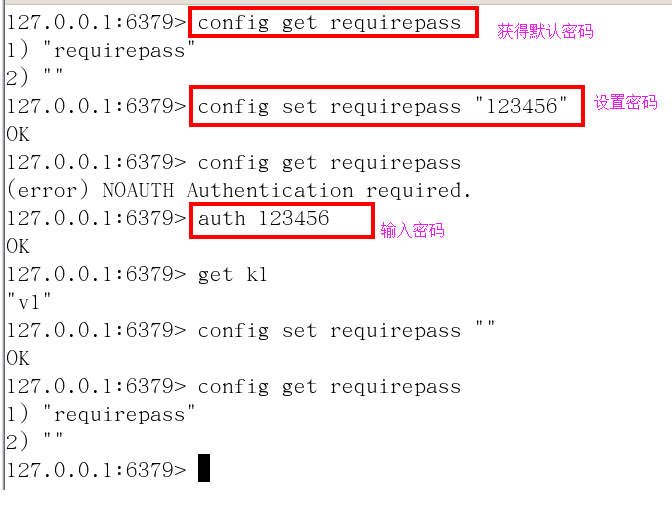


### logfile，syslog-enabled,syslog-ident,syslog-facility



## security安全

我的主要任务是缓存，所以我只要确保我在安全的环境就行



## Limits限制

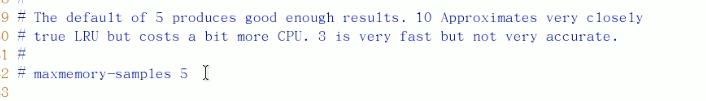
### \*\*\*\*redis的缓存过期配置怎么弄的（面试题）

### 最大内存的缓存清除策略

redis中最大内存达到时，选择那些移除



### Maxmemory-samples



设置样本数量，LRU算法和最小TTL算法都并非是精确的算法，而是估算值，所以你可以设置样本的大小，redis默认会检查这么多个key并选择其中LRU的那个。

## \*\*\*\*常见配置redis.conf介绍

参数说明

redis.conf 配置项说明如下：

1. Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程

daemonize no

2. 当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定

pidfile /var/run/redis.pid

3. 指定Redis监听端口，默认端口为6379，作者在自己的一篇博文中解释了为什么选用6379作为默认端口，因为6379在手机按键上MERZ对应的号码，而MERZ取自意大利歌女Alessia Merz的名字

port 6379

4. 绑定的主机地址

bind 127.0.0.1

5.当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能

timeout 300

6. 指定日志记录级别，Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose

loglevel verbose

7. 日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null

logfile stdout

8. 设置数据库的数量，默认数据库为0，可以使用SELECT 命令在连接上指定数据库id

databases 16

9. 指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合

save

Redis默认配置文件中提供了三个条件：

save 900 1

save 300 10

save 60 10000

分别表示900秒（15分钟）内有1个更改，300秒（5分钟）内有10个更改以及60秒内有10000个更改。

10. 指定存储至本地数据库时是否压缩数据，默认为yes，Redis采用LZF压缩，如果为了节省CPU时间，可以关闭该选项，但会导致数据库文件变的巨大

rdbcompression yes

11. 指定本地数据库文件名，默认值为dump.rdb

dbfilename dump.rdb

12. 指定本地数据库存放目录

dir ./

13. 设置当本机为slav服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步

slaveof

14. 当master服务设置了密码保护时，slav服务连接master的密码

masterauth

15. 设置Redis连接密码，如果配置了连接密码，客户端在连接Redis时需要通过AUTH 命令提供密码，默认关闭

requirepass foobared

16. 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，Redis会关闭新的连接并向客户端返回max number of clients reached错误信息\*\*

maxclients 128

17. 指定Redis最大内存限制，Redis在启动时会把数据加载到内存中，达到最大内存后，Redis会先尝试清除已到期或即将到期的Key，当此方法处理 后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis新的vm机制，会把Key存放内存，Value会存放在swap区

maxmemory

18. 指定是否在每次更新操作后进行日志记录，Redis在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis本身同步数据文件是按上面save条件来同步的，所以有的数据会在一段时间内只存在于内存中。默认为no

appendonly no

19. 指定更新日志文件名，默认为appendonly.aof

appendfilename appendonly.aof

20. 指定更新日志条件，共有3个可选值：

no：表示等操作系统进行数据缓存同步到磁盘（快）

always：表示每次更新操作后手动调用fsync()将数据写到磁盘（慢，安全）

everysec：表示每秒同步一次（折衷，默认值）

appendfsync everysec

21. 指定是否启用虚拟内存机制，默认值为no，简单的介绍一下，VM机制将数据分页存放，由Redis将访问量较少的页即冷数据swap到磁盘上，访问多的页面由磁盘自动换出到内存中（在后面的文章我会仔细分析Redis的VM机制）

vm-enabled no

22. 虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享

vm-swap-file /tmp/redis.swap

23. 将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,所有索引数据都是内存存储的(Redis的索引数据 就是keys),也就是说,当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0

vm-max-memory 0

24. Redis swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，vm-page-size是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page大小最好设置为32或者64bytes；如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值

vm-page-size 32

25. 设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。

vm-pages 134217728

26. 设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4

vm-max-threads 4

27. 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启

glueoutputbuf yes

28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法

hash-max-zipmap-entries 64

hash-max-zipmap-value 512

29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）

activerehashing yes

30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

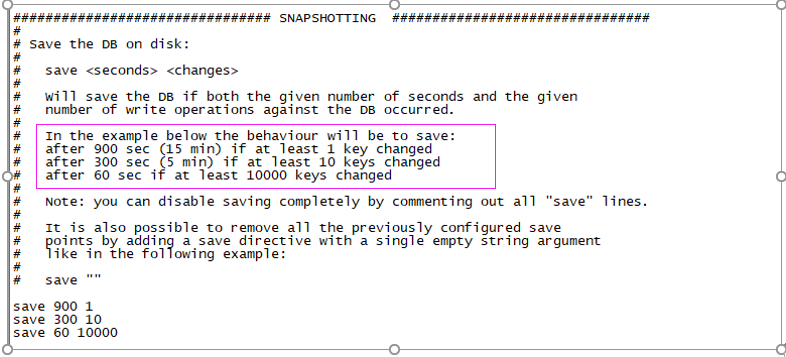
include /path/to/local.conf

# 18Redis\_持久化之RDB(Redis Database面试题)

问持久化答rdb+aof

## rdb是什么

简单的说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



## 做破坏，然后恢复（挖了个坑，执行flushdb清除数据库然后shutdown，虽然rdb文件存在，但是也恢复不了，因为shutdown也会迅速写入rdb,生成新的rdb文件）

所以做flushdb然后shutdown,我们把之前的备份好了rdb文件拿过来就行了。

## 立即备份

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

## 带着问题看

aof和rdb同时存在时冲突还是协作

为什么aof会在rdb后产生

aof有什么优缺点

# 19Redis持久化之AOF

## aof是什么

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。

## 启动aof

修改默认的appendonly no，改为yes.

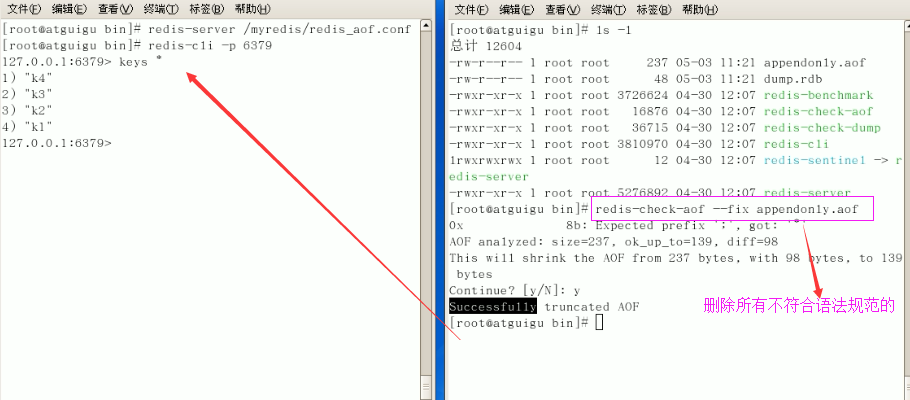
flushdb也会记录到aof中

## 如果一个系统里既有rdb又有aof听谁的

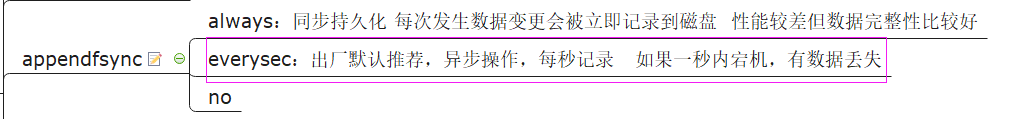
可以这样测试，在aof中做些破坏，同时存在rdb文件（rdb文件时系统写的），如果正常启动说明时rdb起的作用，如果启动不了时aof起的作用。

先加载的时aof

## AOF的修复



## appendfsync



## 两个检查文件的工具



## 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，

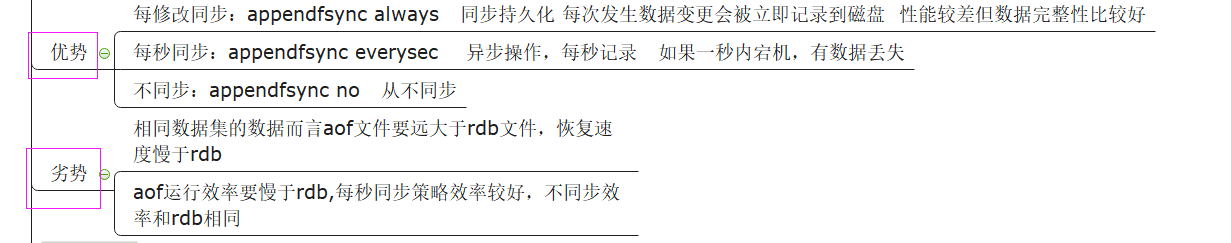
只保留可以恢复数据的最小指令集.可以使用命令bgrewriteaof。

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。

### 触发机制

Redis会记录上次重写时的AOF大小，默认配置是当AOF文件大小是上次rewrite后大小的一倍且文件大于64M时触发

## 优势劣势



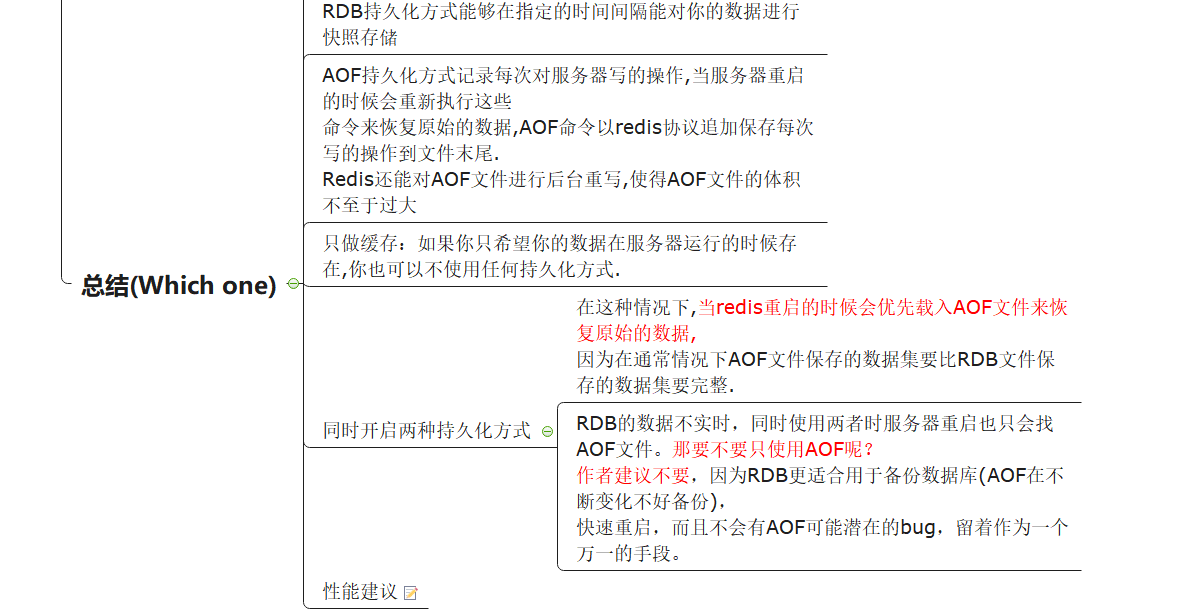
## 性能建议

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

## 总结



# 20Redis\_事务（redis对事务的支持是部分支持case3和case4）

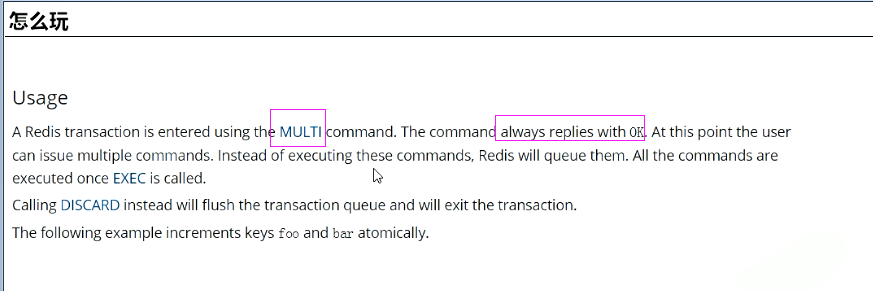
## redis的事务是什么

可以一次执行多个命令，本质是一组命令的集合。一个事务中的所有命令都会序列化，按顺序地串行化执行而不会被其它命令插入，不许加塞。

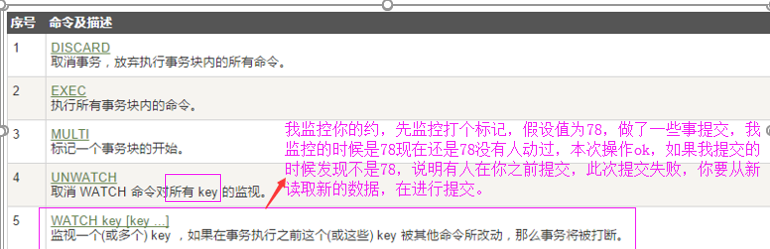
## redis的事务能干啥

一个队列中，一次性、顺序性、排他性的执行一系列命令

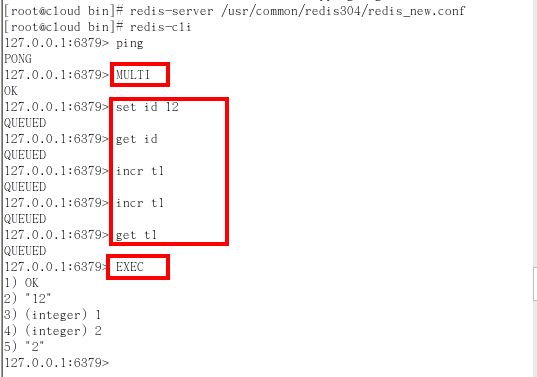
## 怎么使用



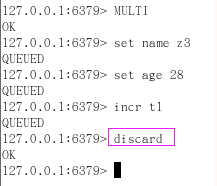
### redis的常用命令



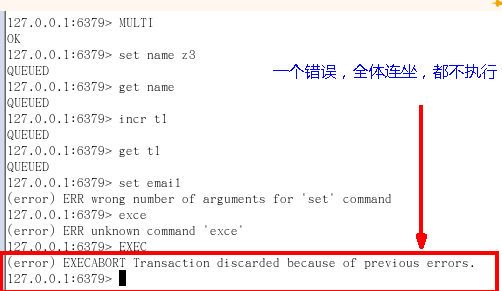
### case1正常执行（批处理）



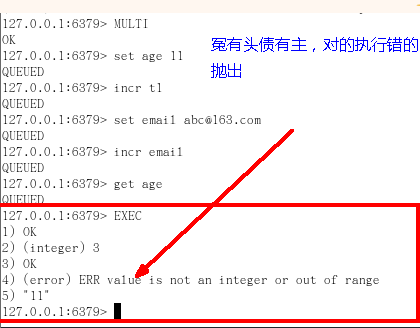
### case2放弃事务



### case3全体连坐(一个错误，全体连坐，都不执行)



### case4冤头债主（对的执行，错的抛出）



case3和case4的区别，case3直接是命令错误，case4是命令对的，加入队列后执行报错

### case5:watch监控

#### 表锁和行锁

表锁，把整个表锁了，只让自己改，（并发行极差，一致性强）

行锁 我只修改表中的某一行，与你不冲突

#### 悲观锁（一致性安全了，并发行不行）

悲观锁(Pessimistic Lock), 顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁

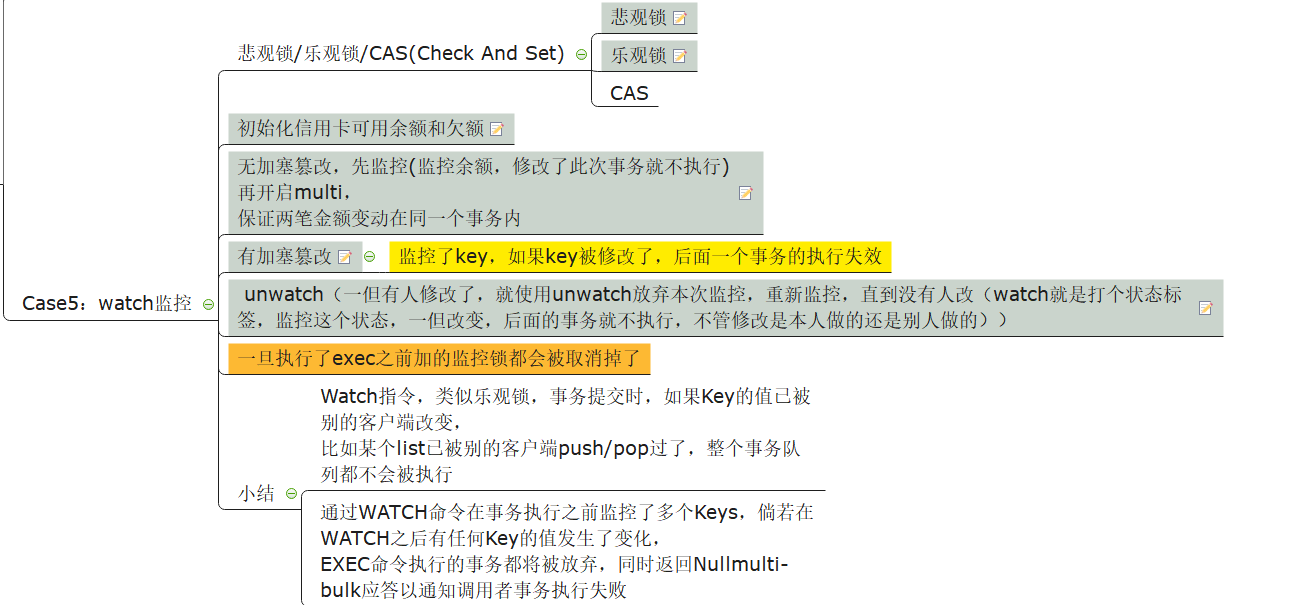
#### 乐观锁（即保证高并发又不锁表，在每行记录后面加个版本号，工作一般用乐观锁）

乐观锁(Optimistic Lock), 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，

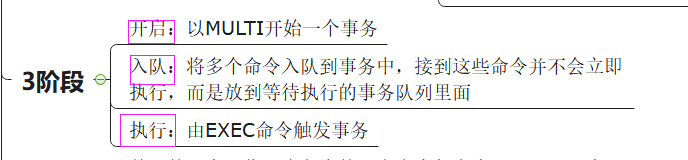
乐观锁策略:提交版本必须大于记录当前版本才能执行更新

#### CAS（check and set）

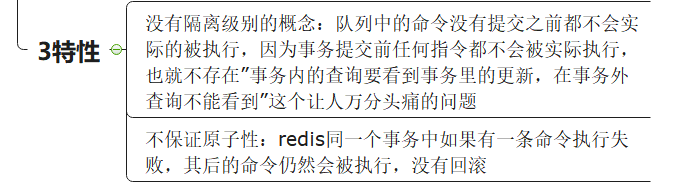
## 例子



## redis事务的三个阶段



## redis的特性

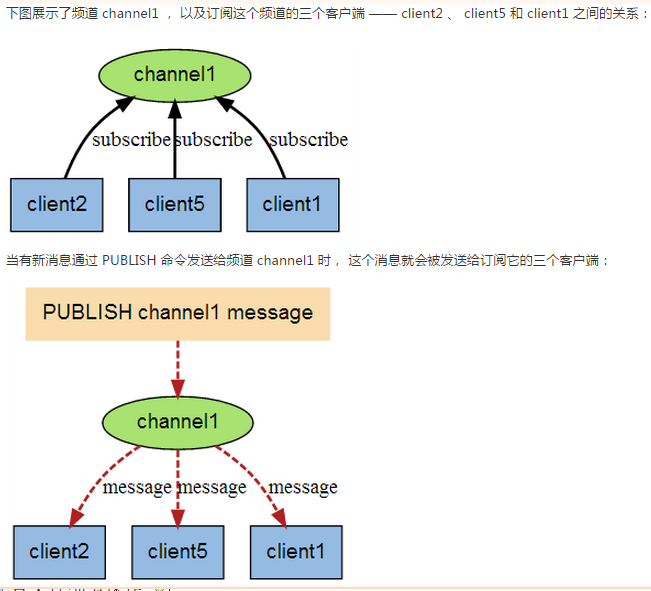


# 21Redis消息订阅发布

## redis的发布订阅是什么

进程间的一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

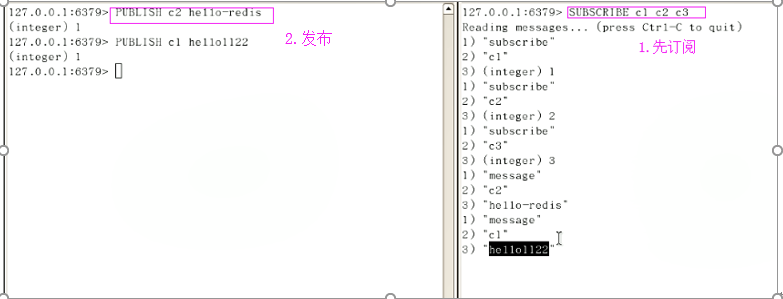
## 订阅/发布消息图



## 命令

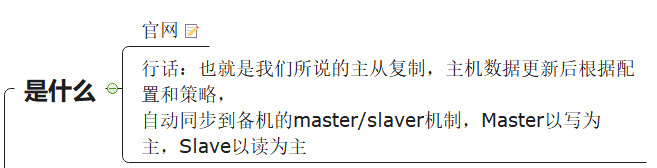


## 案例

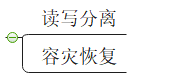


# 22Redis\_主从复制

## 是什么



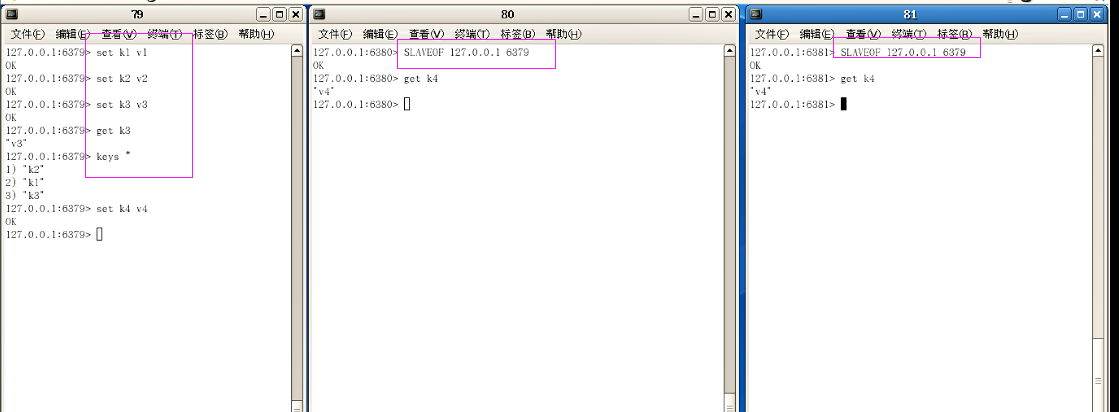
## 能干吗

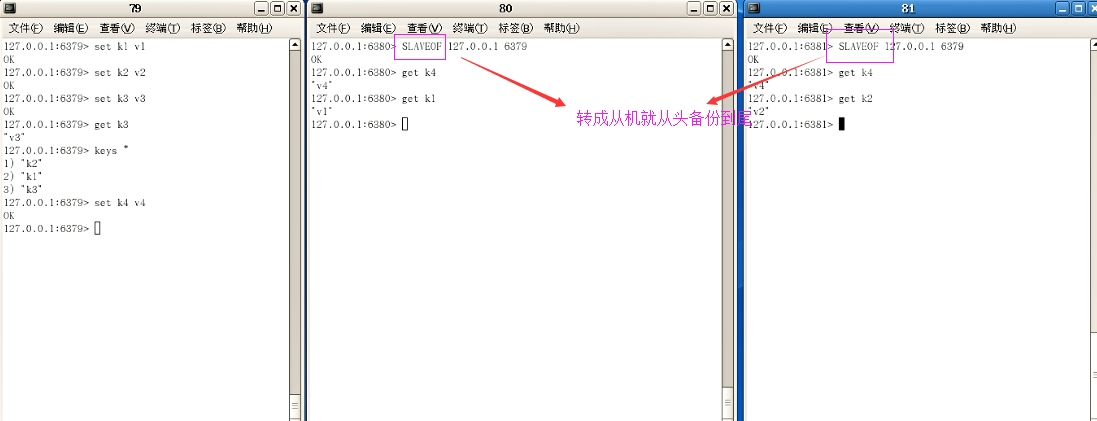


## 怎么玩

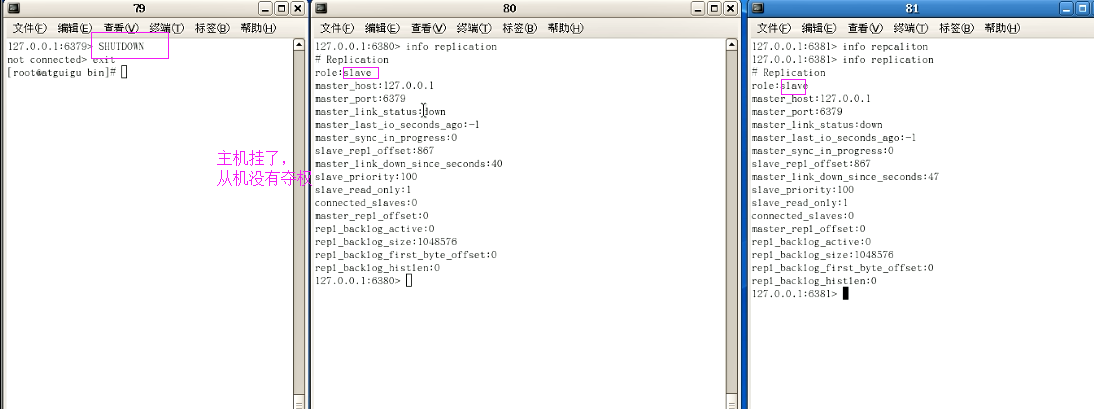
### \*\*\*一主二仆的出现各种情况

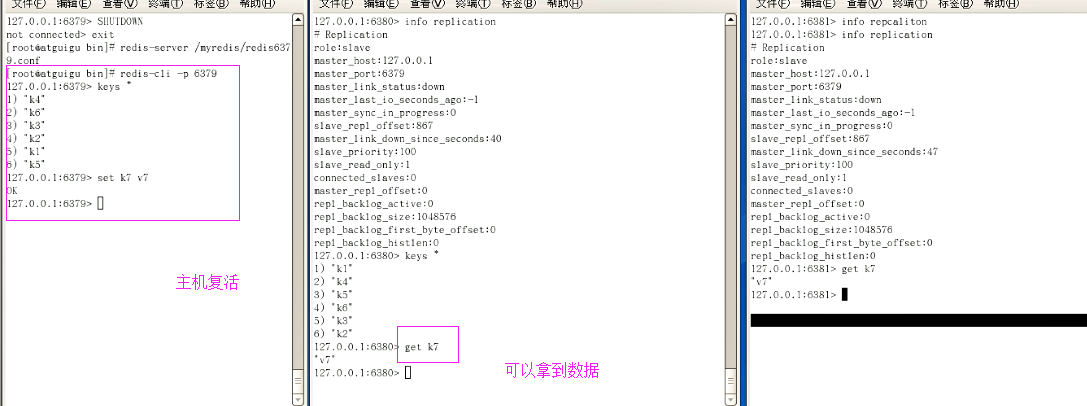


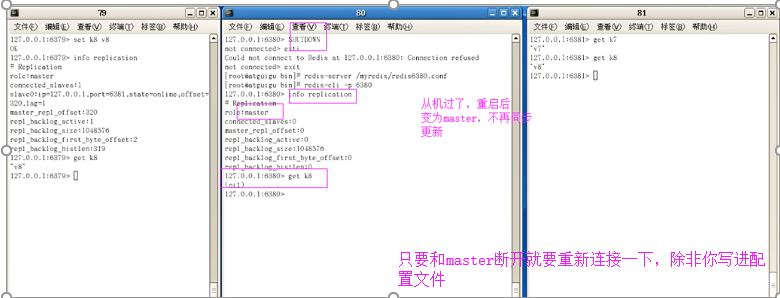


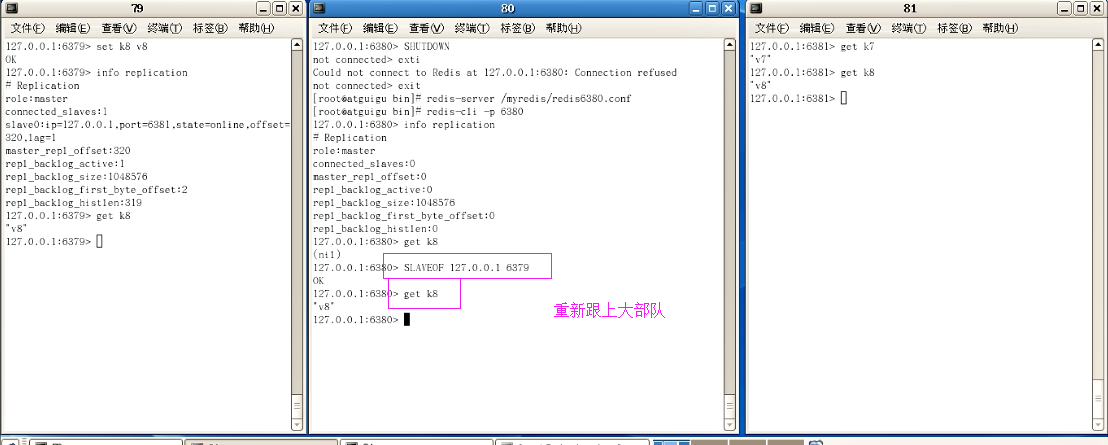






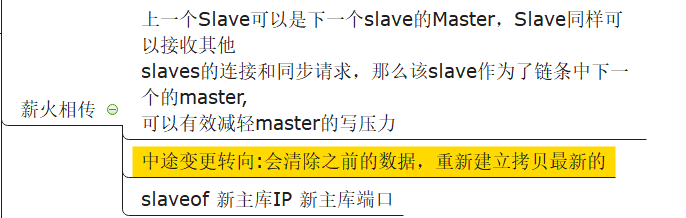


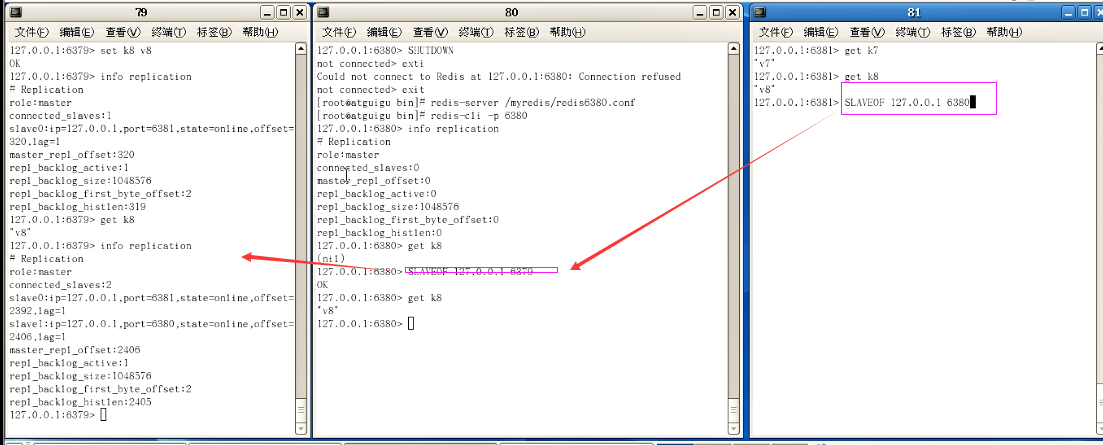


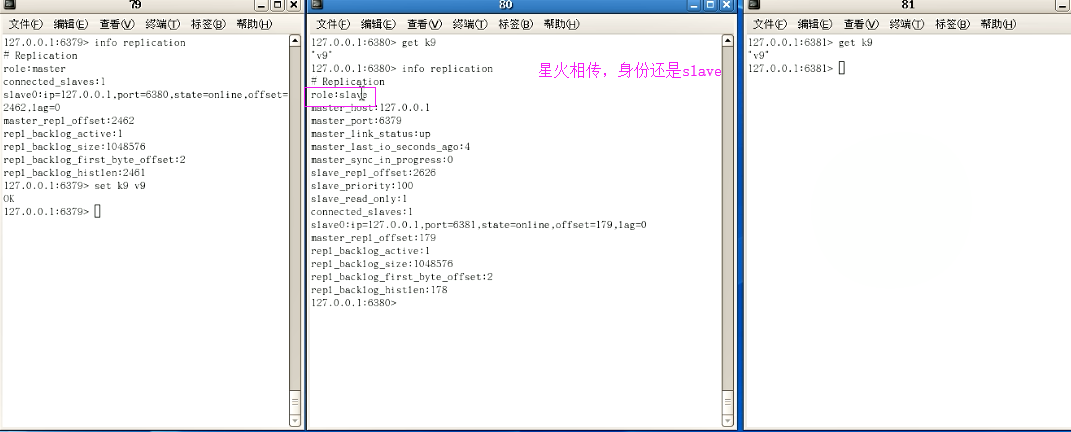


# 23Redis\_主从复制下

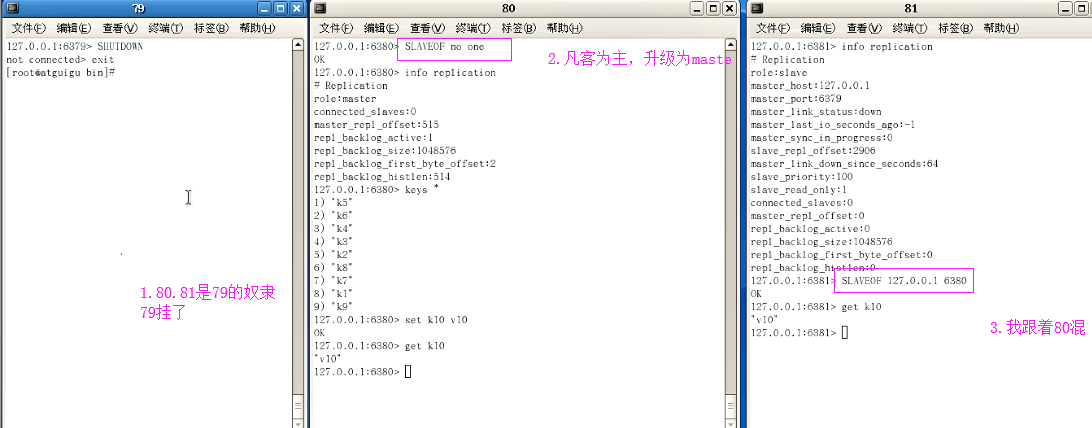
## 薪火相传

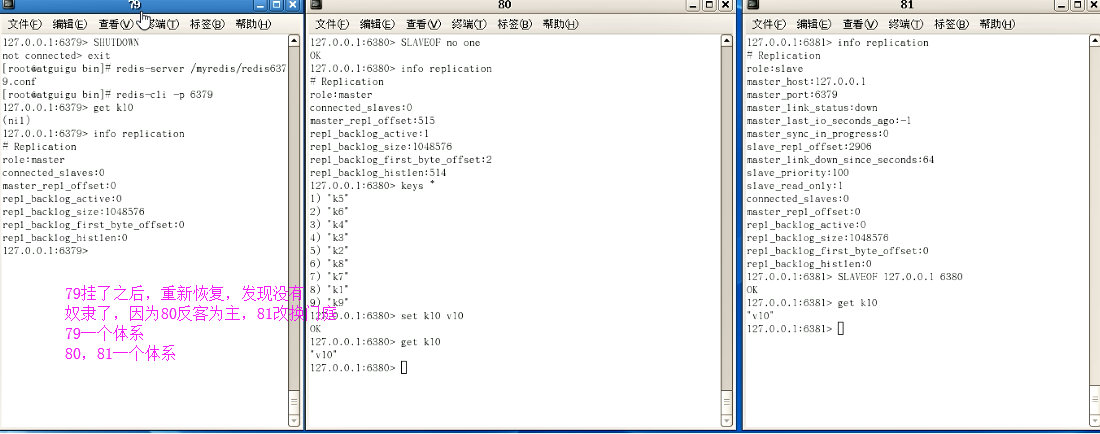






## 反客为主





## 复制原理



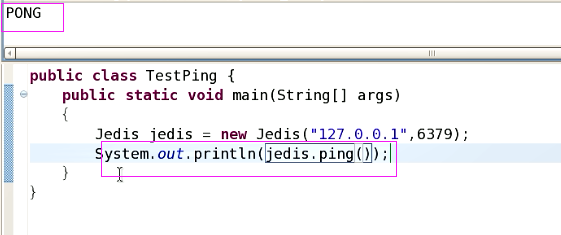
## 哨兵模式（反客为主的自动版 反客为主需要手动执行）



## 复制的缺点

由于所有的写操作都是先在Master上操作，然后同步更新到Slave上，所以从Master同步到Slave机器有一定的延迟，当系统很繁忙的时候，延迟问题会更加严重，Slave机器数量的增加也会使这个问题更加严重。

# 24Redis\_jedis测试联通



# 25Redi\_jedis常用API

# 26Redis\_jedis\_事务

# 27主从复制

# 28Redis\_jedis\_jedisPool