# Redis视频笔记

## 简介

1. Redis：Remote Dictionary Server（远程字典服务器），基于内存运行并支持持久化的NoSQL数据库。
2. Redis的三个特点
   1. Redis支持数据持久化，可以将内存中的数据保存到磁盘中，重启的时候可以再次加载进行使用。
   2. Redis不仅仅支持简单的Key-Value类型的数据，同时还支持String、list、set、zset、hash等数据结构的存储。
   3. Redis支持数据的备份，即master-slave模式的数据备份。
3. 传统RDBMS (Relational Database Management System) VS NoSQL

RDBMS(关系型数据库管理系统)

1. 高度组织化结构化数据；
2. 结构化查询语言（SQL）；
3. 数据和关系都存在单独的表中；
4. 数据操纵语言（DML）、数据定义语言（DDL）、数据控制语言（DCL）；
5. 严格的一致性；
6. 基础事务----事务的四大特性(ACID)：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Duration)；

NoSQL

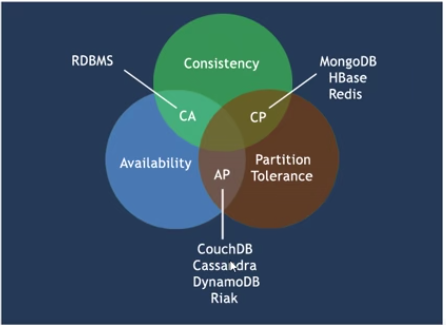
1. 代表着不仅仅是SQL
2. 没有声明性查询语言
3. 没有预定义的模式
4. 键、值对存储，列存储、文档存储、图形数据库
5. 最终一致性，而非ACID属性
6. 非结构化和不可预知的数据
7. CAP定理（Consistency一致性、Availablity可用性、Partition tolerance分区容错）
8. 高性能、高可用性和可伸缩性。
9. Redis、Tair、Memcache和MongDB

Redis和Tair：数据类型丰富。（Redis支持五种数据类型）

Memcache：单一性存储性能更高（字符串类型）

MongDB：是一个“存储数据”的系统，增删改查可以添加很多条件，就像SQL数据库一样灵活。而redis定位是用来做缓存。

1. Redis：KV（键值对存储）、Cache（缓存）、Persistent（持续性）
2. NoSQL数据库的四大分类
   1. KV键值对
   2. 文档型数据库（bson格式比较多）
   3. 列存储数据库
   4. 图关系数据库
3. NoSQL数据库中CAP原理CAP+BASE
   1. 传统ACID分别是什么：（Atomicity）原子性、（Consistency）一致性、（Isolation）独立性、（Durability）持久性
   2. CAP：C（Consistency）强一致性、A（Availability）可用性、P（Partition tolerance）分区容错性也叫分布式容忍性（其实就是高可用性，一个节点崩了，并不影响我们其它的节点）
      1. CAP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、满足AP原则三大类；
      2. 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。传统ORACLE数据库。关系型数据库。
      3. CP - 满足一致性、分区容错性的系统，通常性能不是特别高。大多数网络架构的选择。Redis\MongoGB
      4. AP – 满足可用性、分区容错性的系统，通常可能对一致性要求低一些。



* 1. BASE就是为了解决关系型数据库强一致性引起的问题而引起的可用性降低提出的解决方案
     1. BASE其实是一下三个属于的缩写
        1. 基本可用（Basically Available）
        2. 软状态（Soft state）
        3. 最终一致性（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的改观，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方法。

* 1. 分布式和集群

分布式系统（distributed system）

 由多台计算机和通信的软件组件通过计算机网络连接（本地网络或广域网）组成。分布式系统是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。因此，网络和分布式系统之间的区别更多的在于高层软件（特别是操作系统），而不是硬件。分布式系统可以应用在在不同的平台上如：Pc、工作站、局域网和广域网上等。

* + 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），它们之间通过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（Nginx）进行统一的调度，对外提供服务和访问。

## Redis数据类型

五大数据类型：String(字符串)、Hash(哈希，类似java里的Map)、List（列表）、Set(集合)、Zset(sorted set 有序集合)

### String(字符串)

String类型时二进制安全的，意思是Redis的String可以包含任何数据，比如图片或者序列化的对象。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，是一个String类型的field和value的映射表（类似java里的Map<String, Object>），Hash特别适合存储对象。

Hash的KV模式不变，但是V是一个键值对。

Hset 🡪 给Hash设置值 例hset user name jack

Hget 🡪 获取Hash的值 例hget user name （输出jack）

Hmset 🡪 设置多个值 例：hmset user id 1 name jack

Hmget 🡪 获取多个值 例：hmget user id name （输出 1 jack）

Hgetall 🡪 获取全部值 例：hgetall user （输出 id 1 name jack）

Hlen key 🡪 获取key的长度

Hexists key 在key中的某个value 🡪 判断value是否在key中存在

Hkeys key 🡪 获取key中所有的键 例：hkeys user 输出 id name

Hvals key 🡪 获取key中所有的值 例：hvals user 输出 1 jack

### List(列表)

Redis列表是一个简单的字符串表，按照插入顺序排序，你可以添加一个元素到列表的头部（左边），或者尾部（右边），它的底层实际上是一个链表。

Lpush 🡪 先进后出 lpop 🡪 从左边出栈

Rpush 🡪 先进先出 rpop 🡪 从右边出栈

Lrange 🡪 输出数组长度 例： lrange list01 0 -1 （list01全部输出）

### Set（集合）

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

Sadd 🡪 给set设置值 例： sad set01 1 2 3（给set01放1，2，3三个值）

Smember 🡪 查看set的值 例：smember set01 （会展示出1，2，3）

Scard 🡪 获取set的元素的个数

Srem key value 🡪 删除set中的元素

Spop key 🡪 随机出栈

Smove key1 key2 key1里的某个值 （将key1的某个值赋给key2）

### Zset（sorted set无序集合）

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sorted set的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。

在set基础上，加了一个score值。之前set是k1 v1 v2 v3现在zset是k1 sccore v1 score v2

Zadd 🡪 给zset设置值 例：zadd zset01 60 v1 70 v2 80 v3

Zrange 🡪 查看zset 例：zrange zset01 0 -1 （输出 v1 v2 v3）

例：zrange zset01 0 -1 withscores(输出v1 60 v2 70 v3 80)

## Redis配置文件解析（redis.conf）

### 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 <dbid>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 <seconds> <changes>

  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 <masterip> <masterport>

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

  masterauth <master-password>

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 <password>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 <bytes>

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

## 持久化

### RDB(Redis DataBase)

#### RDB是什么

在指定的时间间隔内将内存中的数据集快照写入磁盘，也就是行话讲的Snapshot快照，它恢复时是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失。

#### Fork

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

RDB是将备份的数据以后缀为.rdb的文件形式保存在磁盘上的。

#### 触发RDB快照策略

Redis的RDB快照默认策略为（redis.conf的默认配置）

1. 在1分钟内改了10000次。
2. 或5分钟内改了10次。
3. 或15分钟内改了1次。
4. 命令save或者bgsave。
   1. Save：save时只管保存，其他不管，全部阻塞。
   2. Bgsave：Redis会在后台异步执行快照操作，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。
5. 执行flushall命令，也会产生dump.rdb文件，但是里面是空的，无意义。

#### 如何恢复

将备份文件（dump.rdb）移动到redis安装目录并启动服务即可。

Config get dir获取目录

Redis重启默认只读取文件名为dump.rdb的文件（配置文件里面的配置）

#### 优势

1. 适合大规模的数据恢复。
2. 对数据的完整性和一致性要求不高。

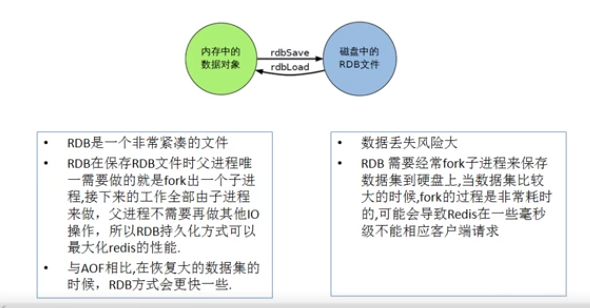
#### [劣势](https://hanyu.baidu.com/zici/s?wd=%E5%8A%A3%E5%8A%BF&query=%E4%BC%98%E5%8A%BF%E7%9A%84%E5%8F%8D%E4%B9%89%E8%AF%8D&srcid=28236&from=kg0)

1. 在一定间隔时间做一次备份，所以redis意外down掉的话就会丢失最后一次快照后的所有修改。
2. Fork的时候，内存的数据被克隆了一份，大致两倍的膨胀性需要考虑。

#### 如何停止(一般不会用到)

1. 设置redis.conf的参数为save “”
2. 动态执行命令：redis-cli config set save “”

#### 小结



### AOF(Append Only File)

#### 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。

#### Aof保存的是appendonly.aof文件

#### AOF配置（redis.conf配置）

1. appendonly no 默认是no若改成yes则表示启动AOF。
2. appenfilename "appendonly.aof"表示默认的AOF快照的文件名
3. appendfsync everysec 有三个参数
   1. always：同步持久化，每次发生数据变更会被立即记录到磁盘，性能较差，但是数据完整性较好。
   2. everysec：出场默认推荐，异步操作，每秒记录，如果一秒内宕机，有数据丢失。
   3. no
4. no-appendfsync-on-rewrite no重写时是否可以运行appendfsync，用默认no即可，保证数据安全性。
5. auto-aof-rewrite-min-size 64mb设置重写的基准值。
6. auto-aof-rewrite-percentage 100设置重写的基准值。

#### AOF启动/修复/恢复

1. 正常启动
   1. 修改redis.conf的默认配置appendonly no为yes；
   2. 重启redis即可启动AOF。
2. 正常恢复
   1. 将有数据的AOF（appendonly.aof）文件复制到对应目录（config get dir）；
   2. 重启redis，即可恢复数据。
3. 异常恢复
   1. 备份坏的AOF（appendonly.aof）文件;
   2. 执行命令redis-check-aof --fix appendonly.aof，即可修复；
   3. 重启redis，即可恢复数据。

#### rewrite重写

1. 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。

1. 重写原理

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，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。

1. 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。

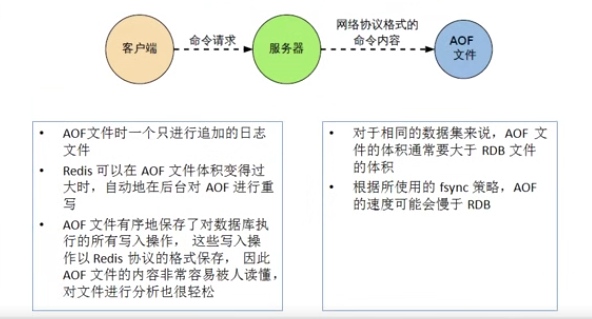
#### 优势

1. 修改同步：appendfsync always 同步持久化 每次发生数据变更会被立即记录到磁盘性能较差但数据完整性比较好。
2. 每秒同步：appendfsync everysec 异步操作，每秒记录，如果一秒内宕机，有数据丢失
3. 不同步：appendfsync no 从不同步。

#### 劣势

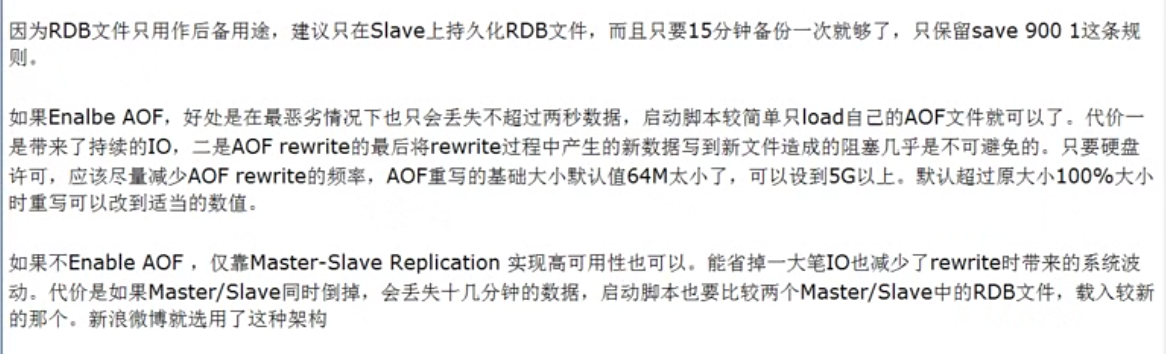
1. 相同数据集的数据而言aof文件要远大于rdb文件，恢复速度慢于rdb。
2. aof运行效率要慢于rdb,每秒同步策略效率较好，不同步效率和rdb相同。

#### 小结



### 总结

1. RDB持久化方式能够在指定的时间间隔能对你的数据进行快照存储。
2. AOF持久化方式记录每次对服务器写的操作，当服务器重启的时候会重新执行这些命令来恢复原始的数据，AOF命令以redis协议追加保存每次写的操作到文件末尾。Redis还能对AOF文件进行后台重写,使得AOF文件的体积不至于过大。
3. 只做缓存：如果你只希望你的数据在服务器运行的时候存在,你也可以不使用任何持久化方式.
4. 同时开启两种持久化方式
   1. 在这种情况下，当redis重启的时候会优先载入AOF文件来恢复原始的数据，因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整。
   2. RDB的数据不实时，同时使用两者时服务器重启也只会找AOF文件。那要不要只使用AOF呢？作者建议不要，因为RDB更适合用于备份数据库(AOF在不断变化不好备份)，快速重启，而且不会有AOF可能潜在的bug，留着作为一个万一的手段。
5. 性能建议



## Redis事务

### Redis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，按顺序的串行化的执行而不会被其他命令插入，不许加塞。

### Redis事务能干什么

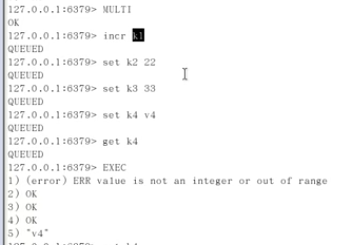
一个队列中、一次性、顺序性、排他性的执行一系列命令。

### Redis事务的使用

#### 常用命令

1. DISCARD 取消事务，放弃执行事务块内的所有命令。【如果正在使用 WATCH 命令监视某个(或某些) key，那么取消所有监视，等同于执行命令 UNWATCH 。】
2. EXEC执行所有事务块内的命令。【假如某个(或某些) key 正处于 [WATCH](http://redisdoc.com/transaction/watch.html#watch) 命令的监视之下，且事务块中有和这个(或这些) key 相关的命令，那么 [EXEC](http://redisdoc.com/transaction/exec.html#exec) 命令只在这个(或这些) key 没有被其他命令所改动的情况下执行并生效，否则该事务被打断(abort)。】
3. MULTI 标记一个事务块的开始。【事务块内的多条命令会按照先后顺序被放到一个队列当中，最后由EXEC命令执行原子性（atomic）的执行。】
4. UNWATCH 取消 [WATCH](http://redisdoc.com/transaction/watch.html#watch) 命令对所有 key 的监视。【如果在执行 [WATCH](http://redisdoc.com/transaction/watch.html#watch) 命令之后，[EXEC](http://redisdoc.com/transaction/exec.html#exec) 命令或 [DISCARD](http://redisdoc.com/transaction/discard.html#discard) 命令先被执行了的话，那么就不需要再执行 [UNWATCH](http://redisdoc.com/transaction/unwatch.html#unwatch) 了。因为 [EXEC](http://redisdoc.com/transaction/exec.html#exec) 命令会执行事务，因此 [WATCH](http://redisdoc.com/transaction/watch.html#watch) 命令的效果已经产生了；而 [DISCARD](http://redisdoc.com/transaction/discard.html#discard) 命令在取消事务的同时也会取消所有对 key 的监视，因此这两个命令执行之后，就没有必要执行 [UNWATCH](http://redisdoc.com/transaction/unwatch.html#unwatch) 了。】
5. WATCH key [key …] 监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。

#### Redis支持部分事务

1. 开启事务后，只要命令放入队列当中，在EXEC执行事务后若因命令正确而执行逻辑不正确而执行失败（命令已经放入队列，执行时失败），则执行失败后不影响该事务后面的命令（不遵守原子性）。所以说Redi支持部分事务。
2. 开启事务后，若命令放入队列时就报错（命令放入队列时就失败），则在EXEC执行事务后，报错之前的命令执行成功，报错之后的命令不会执行。

#### WATCH监控

1. 悲观锁/乐观锁/CAS（Check and Set）
   1. 悲观锁

  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

* 1. 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

* 1. CAS

1. Watch指令，类似乐观锁，事务提交时，如果Key的值已被别的客户端改变，比如某个list已被别的客户端push/pop过了，整个事务队列都不会被执行通过。
2. WATCH命令在事务执行之前监控了多个Keys，倘若在WATCH之后有任何Key的值发生了变化，EXEC命令执行的事务都将被放弃，同时返回Nullmulti-bulk应答以通知调用者事务执行失败。

#### 三阶段

1. 开启：以MULTI开始一个事务。
2. 入队：将多个命令入队到事务中，接到这些命令并不会立即执行，而是放到等待执行的事务队列里面。
3. 执行：由EXEC命令触发事务。

#### 三特性

1. 单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
2. 没有隔离级别的概念：队列中的命令没有提交之前都不会实际的被执行，因为事务提交前任何指令都不会被实际执行，也就不存在”事务内的查询要看到事务里的更新，在事务外查询不能看到“这个让人万分头痛的问题。
3. 不保证原子性：redis同一个事务中如果有一条命令执行失败，其后的命令仍然会被执行，没有回滚。

## Redis消息订阅发布

1. 什么是消息订阅发布

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。

1. 实例



## Redis主从复制读写分离（Master/Slave）

### 主从复制是什么

主机数据更新后，根据配置和策略自动同步到备机的Master/Slave机制，Master以写为主，Slave以读为主。

### 主从复制的作用

1. 读写分离
2. 容灾恢复

### 主从复制的使用

1. 配从（库）不配主（库）
2. 从库配置：slaveof 主库IP 主库端口

每次与Master断开之后都需要重新连接，除非你配置redis.conf文件。

查看配置情况命令：info replication

1. 修改配置文件细节操作

配置一个新的redis的配置（修改redis.conf配置文件）

拷贝多个redis.conf文件

开启daemonize yes

pid文件名字

指定端口

log文件名字

dump.rdb名字

1. 常用3招
   1. 一主而仆
      1. Init
      2. 一个Master，两个Slave
      3. 日志查看
      4. 主从问题演示
         1. 若主机down掉后，两个从机正常运行，则从机依旧是从机；主机重启好了后，主机依旧是主机。
         2. 若一个从机down后，主机和另一台从机正常运行；down掉的从机重启后，这台从机脱离主从复制模式，成为自己的主机；原主机和另一台从机的关系不变。
         3. 主机设置了一个KV从机不能再设置该K，否则会报错。
         4. 从机在任何时候设置成从机，都可以获取主机的所有值。
   2. 薪火相传
      1. 上一个Slave可以是下一个slave的Master，Slave同样可以接收其他slaves的连接和同步请求，那么该slave作为了链条中下一个的master,可以有效减轻master的写压力。
      2. 中途变更转向:会清除之前的数据，重新建立拷贝最新的。
      3. slaveof 新主库IP 新主库端口
   3. 反客为主
      1. SLAVEOF no one：使当前数据库停止与其他数据库的同步，转成主数据库。

### 复制原理

* + - 1. slave启动成功连接到master后会发送一个sync命令。
      2. Master接到命令启动后台的存盘进程，同时收集所有接收到的用于修改数据集命令，在后台进程执行完毕之后，master将传送整个数据文件到slave，以完成一次完全同步。
      3. 全量复制：slave服务在接收到数据库文件数据后，将其存盘并加载到内存中。
      4. 增量复制：Master继续将新的所有收集到的修改命令依次传给slave,完成同步，但是只要是重新连接master，一次完全同步（全量复制)将被自动执行。

### 哨兵模式（sentinel）

#### 哨兵模式是什么

反客为主的自动版，能够后台监控主机是否故障，如果故障了根据投票数自动将从库转换为主库。

#### 哨兵模式的使用

* 1. 调整结构，6379（主）带着6380（从）、6381（从）。
  2. 自定义的/myredis目录下新建sentinel.conf文件，名字绝不能错。
  3. 配置哨兵(sentinel.conf配置文件),填写内容

sentinel monitor 被监控数据库名字(自己起名字) 127.0.0.1 6379 1

* 1. 上面最后一个数字1，表示主机挂掉后salve投票看让谁接替成为主机，得票数多少后成为主机。
  2. 启动哨兵

redis-sentinel /myredis/sentinel.conf

上述目录依照各自的实际情况配置，可能目录不同

* 1. 正常主从演示

原主机6379挂了，哨兵模式开启投票机制，6380当选，成为新的主机，6381成为6380的从机，原主机6379启动好后，也成为6380的从机。

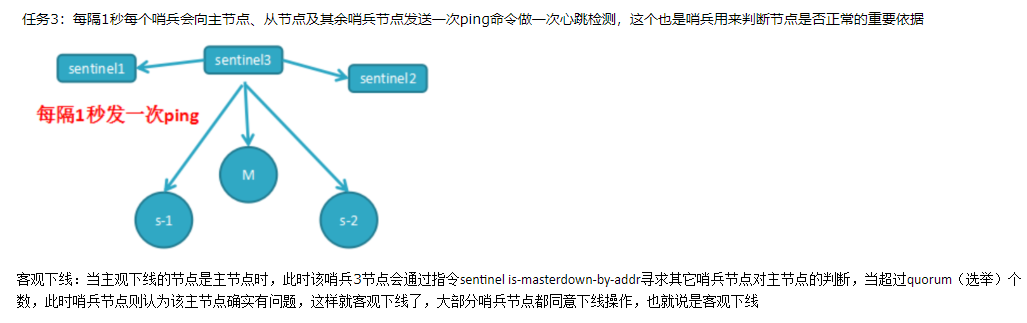
#### 一组sentinel能同时监控多个Master。

#### 哨兵模式的缺点

复制延时

由于所有写操作都是现在Master上操作，然后同步到Slave上，所以从Master同步到Slave上有一定的延迟系统很繁忙的时候，延迟问题会更加严重，Slave机器的增加也会使这个问题更加严重。





## Jedis创建redis连接池

