第1章 关系型和非关系型

关系型数据库： mysql oracle pg

非关系型数据库：redis mongo es

NoSQL not only sql

NoSQL，指的是非关系型的数据库。

NoSQL有时也称作Not Only SQL的缩写是对不同于传统的关系型数据库的数据库管理系统的统称。

对NoSQL最普遍的解释是"非关联型的"，强调Key-Value Stores和文档数据库的优点，而不是单纯的RDBMS。

NoSQL用于超大规模数据的存储。

这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

今天我们可以通过第三方平台可以很容易的访问和抓取数据。

用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。

我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了

NoSQL数据库的发展也却能很好的处理这些大的数据。

第2章 Redis重要特性 AK47

1.速度快

c语言编写的

代码优雅简洁

单线程架构

2.支持多种数据结构

字符串，哈希，列表，集合，有序集合

3.丰富的功能

天然计数器

键过期功能

消息队列

4.支持客户端语言多

php,java,go,python

5.支持数据持久化 面试必问

所有在运行的数据都是放在内存里的

支持多种数据久化格式，RDB，AOF，混合持久化

6.**自带多种高可用架构**

主从，哨兵，集群

第3章 Redis应用场景-面试重点

```
1.缓存-键过期
  把session数据缓存在redis里，过期删除
  换用用户信息，缓存mysql部分的数据，用户先访问redis,如果redis没命中，在访问mysql，然后回写给redis
  商城优惠卷过期
  短信验证码过期
2.排行榜-列表&有序集合
  热度/点击量
  直播间礼物打赏排行榜
3.计数器-天然计数器
  帖子浏览数
  视频播放次数
  评论次数
  点赞/点踩
4.社交网络-集合
  粉丝
  共同好友/可能认识的人
  兴趣爱好/标签
  检查用户注册名是否已经被注册
5.消息队列-列表
  ELK缓存日志
  聊天记录
6.发布订阅
  粉丝关注通知
  消息发布
```

第4章 Redis的安装部署

1.Redis官网

https://redis.io/download

2.版本选择

2.x 非常老

3.x 主流 redis-cluster

4.x 混合持久化

5.x 最新稳定版 新增加了流处理类型

3.规划目录

/data/soft

下载目录

/opt/redis_6379/{conf,logs,pid} 安装目录，日志目录，pid目录，配置目录

/data/redis_6379/ 数据目录

4.安装命令

```
yum install gcc libc -y
mkdir /data/soft -p
cd /data/soft/
wget http://download.redis.io/releases/redis-5.0.7.tar.gz
tar zxf redis-5.0.7.tar.gz -C /opt/
cd /opt
ln -s /opt/redis-5.0.7 /opt/redis
cd /opt/redis
make  #如果报错可以执行 make MALLOC=libc
 make install
 [root@db-51 /opt/redis]# redis-cli -v
 redis-cli 5.0.7

```

报错

```
[root@db-51 redis]# make
cd src && make all
make[1]: Entering directory `/opt/redis-5.0.7/src'
    LINK redis-server
cc: error: ../deps/hiredis/libhiredis.a: No such file or directory
cc: error: ../deps/lua/src/liblua.a: No such file or directory
make[1]: *** [redis-server] Error 1
make[1]: Leaving directory `/opt/redis-5.0.7/src'
make: *** [all] Error 2
```

解决

```
cd deps
```

```
make lua hiredis linenoise
make
make install
```

make和make install功能

菜	**二进制命令**
./config	**洗菜和切菜**
make	**炒菜**
make install	**装盘**

5.编写配置文件

```
mkdir -p /opt/redis_6379/{conf,logs,pid}
mkdir -p /data/redis_6379
cat >/opt/redis_6379/conf/redis_6379.conf<<EOF
daemonize yes
bind 127.0.0.1 10.0.0.51
port 6379
pidfile /opt/redis_6379/pid/redis_6379.pid
logfile /opt/redis_6379/logs/redis_6379.log
EOF
```

6.启动命令

```
redis-server /opt/redis_6379/conf/redis_6379.conf
```

7.检查

```
ps -ef|grep redis
netstat -lntup|grep 6379
```

8.连接redis

```
[root@db01 ~]# redis-cli
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379>
```

9.关闭命令

第一种:

```
[root@db01 ~]# redis-cli
127.0.0.1:6379> SHUTDOWN
```

第二种:

```
[root@db01 ~]# redis-cli shutdown
```

第三种:

```
kill
pkill
```

10.system启动配置

```
redis-cli shutdown
groupadd redis -g 1111
useradd redis -u 1111 -g 1111 -M -s /sbin/nologin
chown -R redis:redis /opt/redis*
chown -R redis:redis /data/redis*
cat >/usr/lib/systemd/system/redis.service<<EOF
[Unit]
Description=Redis persistent key-value database
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
ExecStart=/usr/local/bin/redis-server /opt/redis_6379/conf/redis_6379.conf --supervised systemd
ExecStop=/usr/local/bin/redis-cli shutdown
Type=notify
User=redis
Group=redis
RuntimeDirectory=redis
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
systemctl start redis
```

11.优化警告

警告1：maximum open files过低

```
17068:M 23 Jun 2020 10:23:55.707 # You requested maxclients of 10000 requiring
at least 10032 max file descriptors.
17068:M 23 Jun 2020 10:23:55.707 # Server can't set maximum open files to 10032
because of OS error: Operation not permitted.
17068:M 23 Jun 2020 10:23:55.707 # Current maximum open files is 4096.
maxclients has been reduced to 4064 to compensate for low ulimit. If you need
higher maxclients increase 'ulimit -n'
```

解决：systemd启动文件添加参数

```
vim /usr/lib/systemd/system/redis.service
[Service]
..............
LimitNOFILE=65536
```

警告2: overcommit_memory设置 虚拟内存相关

故障案例章节有讲解

```
17068:M 23 Jun 2020 10:23:55.707 # WARNING overcommit_memory is set to 0!
Background save may fail under low memory condition. To fix this issue add
'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the
command 'sysctl vm.overcommit_memory=1' for this to take effect.
```

解决:

```
sysctl vm.overcommit_memory=1
```

警告3: 关闭THP大内存页

```
17068:M 23 Jun 2020 10:23:55.707 # WARNING you have Transparent Huge Pages (THP)
support enabled in your kernel. This will create latency and memory usage issues
with Redis. To fix this issue run the command 'echo never >
/sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your
/etc/rc.local in order to retain the setting after a reboot. Redis must be
restarted after THP is disabled.
17068:M 23 Jun 2020 10:23:55.707 * Ready to accept connections
```

解决:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

警告4:

```
34685:M 23 Jun 2020 10:47:00.901 # WARNING: The TCP backlog setting of 511
cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower
value of 128.
```

解决:

```
echo "511" > /proc/sys/net/core/somaxconn
sysctl net.core.somaxconn= 4096
```

redis优化告警

```
问题1:
# WARNING: The TCP backlog setting of 511 cannot be enforced bec
ause /proc/sys/net/core/somaxconn is set to the lower value of 12
8.
解决方法:
cat /proc/sys/net/core/somaxconn
echo 511 > /proc/sys/net/core/somaxconn

问题2:
# WARNING overcommit_memory is set to 0! Background save may fai
l under low memory condition. To fix this issue add 'vm.overcommit
_memory = 1' to /etc/sysctl.conf and then reboot or run the comman
d 'sysctl vm.overcommit_memory=1' for this to take effect.
解决方法:
vim /etc/sysctl.conf
vm.overcommit_memory = 1
sysctl vm.overcommit_memory=1

问题3:
# WARNING you have Transparent Huge Pages (THP) support enabled
in your kernel. This will create latency and memory usage issues w
ith Redis. To fix this issue run the command 'echo never > /sys/ke
rnel/mm/transparent_hugepage/enabled' as root, and add it to your
/etc/rc.local in order to retain the setting after a reboot. Redis
must be restarted after THP is disabled.
解决方法:
echo never > /sys/kernel/mm/transparent_hugepage/enabled

问题4:
# You requested maxclients of 10000 requiring at least 10032 ma
x file descriptors.
# Server can't set maximum open files to 10032 because of OS er
ror: Operation not permitted.
# Current maximum open files is 4096. maxclients has been reduc
ed to 4064 to compensate for low ulimit. If you need higher maxcli
ents increase 'ulimit -n'.
解决方法:
vim /usr/lib/systemd/system/redis.service
[Service]
```

```
26 ..............
27 LimitNOFILE=65536
28
```

11.配置文件解释

```
1 # daemonize no 默认情况下， redis 不是在后台运行的，如果需要在后台运行，把该项的值更改为 yes
2 daemonize yes
3 # 当redis在后台运行的时候， Redis默认会把pid文件放在/var/run/redis.pid ，你可以配置到其他地址。
4 # 当运行多个redis服务时，需要指定不同的 pid 文件和端口
5 pidfile /var/run/redis_6379.pid
6 # 指定redis运行的端口，默认是 6379
7 port 6379
8 # 在高并发的环境中，为避免慢客户端的连接问题，需要设置一个高速后台日志
9 tcp-backlog 511
10 # 指定 redis 只接收来自于该 IP 地址的请求，如果不进行设置，那么将处理所有请求
11 # bind 192.168.1.100 10.0.0.1
12 # bind 127.0.0.1
13 # 设置客户端连接时的超时时间，单位为秒。当客户端在这段时间内没有发出任何指令，那么关闭该连接
14 # 0 是关闭此设置
15 timeout 0
16 # TCP keepalive
17 # 在 Linux 上，指定值（秒）用于发送 ACKs 的时间。注意关闭连接需要双倍的时间。默认为 0 。
18 tcp-keepalive 0
19 # 指定日志记录级别，生产环境推荐 notice
20 # Redis 总共支持四个级别： debug 、 verbose 、 notice 、 warning ，默认为 verbose
21 # debug 记录很多信息，用于开发和测试
22 # varbose 有用的信息，不像 debug 会记录那么多
23 # notice 普通的 verbose ，常用于生产环境
24 # warning 只有非常重要或者严重的信息会记录到日志
```

```
loglevel notice
# 配置 log 文件地址
# 默认值为 stdout ，标准输出，若后台模式会输出到 /dev/null 。
logfile /var/log/redis/redis.log
# 可用数据库数
# 默认值为 16 ，默认数据库为 0 ，数据库范围在 0- （ database-1 ）之间
databases 16
################################ 快照#################################
# 保存数据到磁盘，格式如下 :
# save
# 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 。
# 相当于条件触发抓取快照，这个可以多个条件配合
# 比如默认配置文件中的设置，就设置了三个条件
# save 900 1 900 秒内至少有 1 个 key 被改变
# save 300 10 300 秒内至少有 300 个 key 被改变
# save 60 10000 60 秒内至少有 10000 个 key 被改变
# save 900 1
# save 300 10
# save 60 10000
# 后台存储错误停止写。
stop-writes-on-bgsave-error yes
# 存储至本地数据库时（持久化到 rdb 文件）是否压缩数据，默认为 yes
rdbcompression yes
# RDB 文件的是否直接偶像 chcksum
rdbchecksum yes
# 本地持久化数据库文件名，默认值为 dump.rdb
dbfilename dump.rdb
# 工作目录
# 数据库镜像备份的文件放置的路径。
# 这里的路径跟文件名要分开配置是因为 redis 在进行备份时，先会将当前数据库的状态写入到一个临时文件中，等备份完成，
# 再把该该临时文件替换为上面所指定的文件，而这里的临时文件和上面所配置的备份文件都会放在这个指定的路径当中。
# AOF 文件也会存放在这个目录下面
```

```
57 # 注意这里必须制定一个目录而不是文件
58 dir /var/lib/redis-server/
59 ################################## 复制 ###################################
60 # 主从复制 . 设置该数据库为其他数据库的从数据库 .
61 # 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
62 # slaveof
63 # 当 master 服务设置了密码保护时 ( 用 requirepass 制定的密码 )
64 # slave 服务连接 master 的密码
65 # masterauth
66 # 当从库同主机失去连接或者复制正在进行，从机库有两种运行方式：
67 # 1) 如果 slave-serve-stale-data 设置为 yes( 默认设置 ) ，从库会继续响应客户端的请求
68 # 2) 如果 slave-serve-stale-data 是指为 no ，出去 INFO 和 SLAVOF 命令之外的任何请求都会返回一个
69 # 错误 "SYNC with master in progress"
70 slave-serve-stale-data yes
71 # 配置 slave 实例是否接受写。写 slave 对存储短暂数据（在同 master 数据同步后可以很容易地被删除）是有用的，但未配置的情况下，客户端写可能会发送问题。
72 # 从 Redis2.6 后，默认 slave 为 read-only
73 slaveread-only yes
74
75 # 从库会按照一个时间间隔向主库发送 PINGs. 可以通过 repl-ping-slave-period 设置这个时间间隔，默认是 10 秒
76 # repl-ping-slave-period 10
77 # repl-timeout 设置主库批量数据传输时间或者 ping 回复时间间隔，默认值是 60 秒
78 # 一定要确保 repl-timeout 大于 repl-ping-slave-period
79 # repl-timeout 60
80 # 在 slave socket 的 SYNC 后禁用 TCP_NODELAY
81 # 如果选择“ yes ” ,Redis 将使用一个较小的数字 TCP 数据包和更少的带宽将数据发送到 slave ， 但是这可能导致数据发送到 slave 端会有延迟 , 如果是 Linux kernel 的默认配置，会达到 40 毫秒 .
82 # 如果选择 "no" ，则发送数据到 slave 端的延迟会降低，但将使用更多的带宽用于复制 .
83 repl-disable-tcp-nodelay no
```

```
84 # 设置复制的后台日志大小。
85 # 复制的后台日志越大， slave 断开连接及后来可能执行部分复制花的时间就越长。
86 # 后台日志在至少有一个 slave 连接时，仅仅分配一次。
87 # repl-backlog-size 1mb
88 # 在 master 不再连接 slave 后，后台日志将被释放。下面的配置定义从最后一个 slave 断开连接后需要释放的时间（秒）。
89 # 0 意味着从不释放后台日志
90 # repl-backlog-ttl 3600
91 # 如果 master 不能再正常工作，那么会在多个 slave 中，选择优先值最小的一个 slave 提升为 master ，优先值为 0 表示不能提升为 master 。
92 slave-priority 100
93 # 如果少于 N 个 slave 连接，且延迟时间 <=M 秒，则 master 可配置停止接受写操作。
94 # 例如需要至少 3 个 slave 连接，且延迟 <=10 秒的配置：
95 # min-slaves-to-write 3
96 # min-slaves-max-lag 10
97 # 设置 0 为禁用
98 # 默认 min-slaves-to-write 为 0 （禁用）， min-slaves-max-lag 为 10
99 ################################## 安全 ###################################
100 # 设置客户端连接后进行任何其他指定前需要使用的密码。
101 # 警告：因为 redis 速度相当快，所以在一台比较好的服务器下，一个外部的用户可以在一秒钟进行 150K 次的密码尝试，这意味着你需要指定非常非常强大的密码来防止暴力破解
102 # requirepass foobared
103 # 命令重命名 .
104 # 在一个共享环境下可以重命名相对危险的命令。比如把 CONFIG 重名为一个不容易猜测的字符。
105 # 举例 :
106 # rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52
107 # 如果想删除一个命令，直接把它重命名为一个空字符 "" 即可，如下：
108 # rename-command CONFIG ""
109 ################################## 约束#####################################
110 #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 10000
# 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 会按照清除策略尝试清除已到期的 Key
# 如果 Redis 依照策略清除后无法提供足够空间，或者策略设置为 "noeviction" ，则使用更多空间的命令将会报错，例如 SET, LPUSH 等。但仍然可以进行读取操作
# 注意： Redis 新的 vm 机制，会把 Key 存放内存， Value 会存放在 swap 区
# 该选项对 LRU 策略很有用。
# maxmemory 的设置比较适合于把 redis 当作于类似 memcached 的缓存来使用，而不适合当做一个真实的 DB 。
# 当把 Redis 当做一个真实的数据库使用的时候，内存使用将是一个很大的开销
# maxmemory
# 当内存达到最大值的时候 Redis 会选择删除哪些数据？有五种方式可供选择
# volatile-lru -> 利用 LRU 算法移除设置过过期时间的 key (LRU: 最近使用 Least RecentlyUsed )
# allkeys-lru -> 利用 LRU 算法移除任何 key
# volatile-random -> 移除设置过过期时间的随机 key
# allkeys->random -> remove a randomkey, any key
# volatile-ttl -> 移除即将过期的 key(minor TTL)
# noeviction -> 不移除任何可以，只是返回一个写错误
# 注意：对于上面的策略，如果没有合适的 key 可以移除，当写的时候 Redis 会返回一个错误
# 默认是 : volatile-lru
# maxmemory-policy volatile-lru
# LRU 和 minimal TTL 算法都不是精准的算法，但是相对精确的算法 ( 为了节省内存 ) ，随意你可以选择样本大小进行检测。
# Redis 默认的灰选择 3 个样本进行检测，你可以通过 maxmemory-samples 进行设置
# maxmemory-samples 3
############################## AOF###############################
##
```

```
136 # 默认情况下， redis 会在后台异步的把数据库镜像备份到磁盘，但是该备份是非常耗时的，而且备份也不能很频繁，如果发生诸如拉闸限电、拔插头等状况，那么将造成比较大范围的数据丢失。
137 # 所以 redis 提供了另外一种更加高效的数据库备份及灾难恢复方式。
138 # 开启 append only 模式之后， redis 会把所接收到的每一次写操作请求都追加到 appendonly.aof 文件中，当 redis 重新启动时，会从该文件恢复出之前的状态。
139 # 但是这样会造成 appendonly.aof 文件过大，所以 redis 还支持了 BGREWRITEAOF 指令，对 appendonly.aof 进行重新整理。
140 # 你可以同时开启 asynchronous dumps 和 AOF
141 appendonly no
142 # AOF 文件名称 ( 默认 : "appendonly.aof")
143 # appendfilename appendonly.aof
144 # Redis 支持三种同步 AOF 文件的策略 :
145 # no: 不进行同步，系统去操作 . Faster.
146 # always: always 表示每次有写操作都进行同步 . Slow, Safest.
147 # everysec: 表示对写操作进行累积，每秒同步一次 . Compromise.
148 # 默认是 "everysec" ，按照速度和安全折中这是最好的。
149 # 如果想让 Redis 能更高效的运行，你也可以设置为 "no" ，让操作系统决定什么时候去执行
150 # 或者相反想让数据更安全你也可以设置为 "always"
151 # 如果不确定就用 "everysec".
152 # appendfsync always
153 appendfsync everysec
154 # appendfsync no
155 # AOF 策略设置为 always 或者 everysec 时，后台处理进程 ( 后台保存或者 AOF 日志重写 ) 会执行大量的 I/O 操作
156 # 在某些 Linux 配置中会阻止过长的 fsync() 请求。注意现在没有任何修复，即使 fsync 在另外一个线程进行处理
157 # 为了减缓这个问题，可以设置下面这个参数 no-appendfsync-on-rewrite
158 no-appendfsync-on-rewrite no
159 # AOF 自动重写
160 # 当 AOF 文件增长到一定大小的时候 Redis 能够调用 BGREWRITEAOF 对日志文件进行重写
161 # 它是这样工作的： Redis 会记住上次进行些日志后文件的大小 ( 如果从开机以来还没进行过重写，那日子大小在开机的时候确定 )
162 # 基础大小会同现在的大小进行比较。如果现在的大小比基础大小大制定的百分比，重写功能将启动
```

```
163 # 同时需要指定一个最小大小用于 AOF 重写，这个用于阻止即使文件很小但是增长幅度很大也去重写 AOF 文件的情况
164 # 设置 percentage 为 0 就关闭这个特性
165 auto-aof-rewrite-percentage 100
166 auto-aof-rewrite-min-size 64mb
167 ################################## LUASCRIPTING ###############################
168 # 一个 Lua 脚本最长的执行时间为 5000 毫秒（ 5 秒），如果为 0 或负数表示无限执行时间。
169 lua-time-limit 5000
170 ################################LOW LOG#######################################
171 # Redis Slow Log 记录超过特定执行时间的命令。执行时间不包括 I/O 计算比如连接客户端，返回结果等，只是命令执行时间
172 # 可以通过两个参数设置 slow log：一个是告诉 Redis 执行超过多少时间被记录的参数 slowlog-log-slower-than( 微妙 )，
173 # 另一个是 slow log 的长度。当一个新命令被记录的时候最早的命令将被从队列中移除
174 # 下面的时间以微妙为单位，因此 1000000 代表一秒。
175 # 注意指定一个负数将关闭慢日志，而设置为 0 将强制每个命令都会记录
176 slowlog-log-slower-than 10000
177 # 对日志长度没有限制，只是要注意它会消耗内存
178 # 可以通过 SLOWLOG RESET 回收被慢日志消耗的内存
179 # 推荐使用默认值 128 ，当慢日志超过 128 时，最先进入队列的记录会被踢出
180 slowlog-max-len 128
181 ################################ 事件通知 ######################################
182 # 当事件发生时， Redis 可以通知 Pub/Sub 客户端。
183 # 可以在下表中选择 Redis 要通知的事件类型。事件类型由单个字符来标识：
184 # K Keyspace 事件，以 _keyspace@_ 的前缀方式发布
185 # E Keyevent 事件，以 _keysevent@_ 的前缀方式发布
186 # g 通用事件（不指定类型），像 DEL, EXPIRE, RENAME, …
187 # $ String 命令
188 # s Set 命令
189 # h Hash 命令
190 # z 有序集合命令
```

```
# x 过期事件（每次 key 过期时生成）
# e 清除事件（当 key 在内存被清除时生成）
# A g$lshzxe 的别称，因此 ”AKE” 意味着所有的事件
# notify-keyspace-events 带一个由 0 到多个字符组成的字符串参数。空字符串意思是通知被禁用。
# 例子：启用 list 和通用事件：
# notify-keyspace-events Elg
# 默认所用的通知被禁用，因为用户通常不需要改特性，并且该特性会有性能损耗。
# 注意如果你不指定至少 K 或 E 之一，不会发送任何事件。
notify-keyspace-events “”
############################## 高级配置 ############################################
# 当 hash 中包含超过指定元素个数并且最大的元素没有超过临界时，
# hash 将以一种特殊的编码方式（大大减少内存使用）来存储，这里可以设置这两个临界值
# Redis Hash 对应 Value 内部实际就是一个 HashMap ，实际这里会有 2 种不同实现，
# 这个 Hash 的成员比较少时 Redis 为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的 HashMap 结构，对应的 valueredisObject 的 encoding 为 zipmap，
# 当成员数量增大时会自动转成真正的 HashMap，此时 encoding 为 ht 。
hash-max-zipmap-entries 512
hash-max-zipmap-value 64
# 和 Hash 一样，多个小的 list 以特定的方式编码来节省空间。
# list 数据类型节点值大小小于多少字节会采用紧凑存储格式。
list-max-ziplist-entries 512
list-max-ziplist-value 64
# set 数据类型内部数据如果全部是数值型，且包含多少节点以下会采用紧凑格式存储。
set-max-intset-entries 512
# 和 hashe 和 list 一样 ，排序的 set 在指定的长度内以指定编码方式存储以节省空间
# zsort 数据类型节点值大小小于多少字节会采用紧凑存储格式。
zset-max-ziplist-entries 128
zset-max-ziplist-value 64
# Redis 将在每 100 毫秒时使用 1 毫秒的 CPU 时间来对 redis 的 hash 表进行重新 hash ，可以降低内存的使用
```

```
# 当你的使用场景中，有非常严格的实时性需要，不能够接受 Redis 时不时的对请求有 2 毫秒的延迟的话，把这项配置为 no 。
# 如果没有这么严格的实时性要求，可以设置为 yes ，以便能够尽可能快的释放内存
activerehashing yes
# 客户端的输出缓冲区的限制，因为某种原因客户端从服务器读取数据的速度不够快，
# 可用于强制断开连接（一个常见的原因是一个发布 / 订阅客户端消费消息的速度无法赶上生产它们的速度）。
# 可以三种不同客户端的方式进行设置：
# normal -> 正常客户端
# slave -> slave 和 MONITOR 客户端
# pubsub -> 至少订阅了一个 pubsub channel 或 pattern 的客户端
# 每个 client-output-buffer-limit 语法 :
# client-output-buffer-limit
# 一旦达到硬限制客户端会立即断开，或者达到软限制并保持达成的指定秒数（连续）。
# 例如，如果硬限制为 32 兆字节和软限制为 16 兆字节 /10 秒，客户端将会立即断开
# 如果输出缓冲区的大小达到 32 兆字节，客户端达到 16 兆字节和连续超过了限制 10 秒，也将断开连接。
# 默认 normal 客户端不做限制，因为他们在一个请求后未要求时（以推的方式）不接收数据，
# 只有异步客户端可能会出现请求数据的速度比它可以读取的速度快的场景。
# 把硬限制和软限制都设置为 0 来禁用该特性
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb60
client-output-buffer-limit pubsub 32mb 8mb60
# Redis 调用内部函数来执行许多后台任务，如关闭客户端超时的连接，清除过期的 Key ，等等。
# 不是所有的任务都以相同的频率执行，但 Redis 依照指定的“ Hz ”值来执行检查任务。
# 默认情况下，“ Hz ”的被设定为 10 。
# 提高该值将在 Redis 空闲时使用更多的 CPU 时，但同时当有多个 key 同时到期会使 Redis 的反应更灵敏，以及超时可以更精确地处理。
# 范围是 1 到 500 之间，但是值超过 100 通常不是一个好主意。
# 大多数用户应该使用 10 这个预设值，只有在非常低的延迟的情况下有必要提高最大到 100 。
```

```
245 hz 10
246 # 当一个子节点重写 AOF 文件时，如果启用下面的选项，则文件每生成 32M 数据进行同步。
247 aof-rewrite-incremental-fsync yes
```

第5章 Redis全局命令

1.Redis数据格式

```
1 key:value
```

键:值

2.写入测试命令

```
1 set k1 v1
2 set k2 v2
3 set k3 v3
```

3.查看所有的key-注意！！！此操作未满28岁，请在父母陪同下操作！！！

线上生产禁止执行！！！

```
keys *
```

4.查看有多少个key

```
1 DBSIZE
```

5.查看某个key是否存在

```
1 EXISTS k1
2
3 状态码:
4 0: 表示这个key不存在
5 1: 表示这个key存在
6 N: 表示有N个key存在
```

6.删除KEY

```
1 DEL k1
2 DEL k1 k2
3
4 状态码:
5 0: 要删除的KEY不存在
6 1: 表示这个key存在，并且被删除成功了
```

```
N：表示N个key存在，并且被删除成功了N个
```

7.键过期　重要

设置过期时间

```
EXPIRE k1 10

状态码：
0：这个key不存在
1：这个key存在，并且设置过期时间成功
```

查看key是否过期

```
TTL k1

状态码：
-1 ：这个key存在，但是没有设置过期时间，永不过期
-2 ：这个key不存在
N ：表示这个key存在，并且还有N秒过期

127.0.0.1:6379> set k8 v8 EX 100
OK
 127.0.0.1:6379> TTL k8
 (integer) 89
 127.0.0.1:6379> TTL k8
 (integer) 88
 127.0.0.1:6379> TTL k8
 (integer) 87

```

取消过期时间

```
第一种方法：
set k1 v1

第二种方法：
PERSIST k1
127.0.0.1:6379> PERSIST k8
(integer) 1
127.0.0.1:6379> TTL k8
(integer) -1
```

取消key

```
127.0.0.1:6379> FLUSHALL
OK
127.0.0.1:6379> keys *
(empty list or set)
```

第6章 字符串操作

1.设置一个key

```
set k1 v1
set k1 v1 EX 10
```

2.查看一个key

```
127.0.0.1:6379> get k1
"v1"
```

3.设置多个key

```
MSET k1 v1 k2 v2 k3 v3
```

4.查看多个key

```
127.0.0.1:6379> mget k1 k2 k3
1) "v1"
2) "v2"
3) "v3"
```

5.天然计数器

加1：

```
SET k1 1
INCR k1
GET k1
127.0.0.1:6379> set k1 1
OK
127.0.0.1:6379> get k1
"1"

127.0.0.1:6379> incr k1
(integer) 2
```

```
127.0.0.1:6379> get k1
"2"
```

加N：

```
INCRBY k1 100
127.0.0.1:6379> INCRBY k1 100
(integer) 102
```

减1：

```
DECR k1
INCRBY k1 -1

减N:
DECRBY k1 N
INCRBY k1 -N

127.0.0.1:6379> DECR k1
(integer) 101
127.0.0.1:6379> INCRBY k1 -1
(integer) 100
```

减N：

```
DECRBY k1 N
INCRBY k1 -N
127.0.0.1:6379> DECRBY k1 100
(integer) 0
127.0.0.1:6379> INCRBY k1 -21
(integer) -21
```

故障案例：

问题背景：

某日，突然在公司办公室集体访问不了公司网站了，访问其他网站都正常，用手机流量访问公司网站却正常

排查过程：

笔记本用手机流量热点，连上了IDC机房的VPN服务器，连上反向代理负载均衡查看，发现公司出口IP地址被防火墙封掉了。

紧急恢复：
先放开规则，恢复访问。再排查问题

排查步骤：
为什么办公室会被封？
防火墙上做了限制访问次数，如果访问超过1分钟200次，就自动封掉这个IP，24小时后再放开。

内网是谁在大量访问呢？
通过路由器查看那个交换机流量大
通过交换机确认哪个端口的流量异常
拔掉网线，然后等待尖叫声

问题真正原因：
供应商软文有指标，需要把热度炒起来，所以同事用浏览器自动刷新网页的插件不断的刷新网页。
从而触发了防火墙的限制，又因为防火墙没有设置白名单，所以导致整个办公室唯一的出口IP被封掉。

解决方案：
开发在后台添加新功能，输入帖子ID和期望的访问数，操作Redis字符串的计数器功能自动添加访问量
防火墙设置白名单，放开公司办公室出口IP

第7章 列表操作

1.插入列表

从左边压入数据：

`LPUSH list1 A`

```
127.0.0.1:6379> LPUSH list1 A
(integer) 1
127.0.0.1:6379> LPUSH list1 B
```

```
(integer) 2
127.0.0.1:6379> LPUSH list1 C
(integer) 3
127.0.0.1:6379> LPUSH list1 D E
```

从右边压入数据：

RPUSH list1 D

```
127.0.0.1:6379> RPUSH list1 F
(integer) 6
127.0.0.1:6379> RPUSH list1 F G
(integer) 8
```

2.查看列表长度

LLEN list1

```
127.0.0.1:6379> LLEN list1
(integer) 8
```

3.查看列表内容

LRANGE list1 0 -1

```
127.0.0.1:6379> LRANGE list1 0 -1
1) "E"
2) "D"
3) "C"
4) "B"
5) "A"
6) "F"
7) "F"
8) "G"

```

4.删除列表元素

左删

LPOP list1

```
127.0.0.1:6379> LPOP list1
"E"
```

右删

RPOP list1

```
127.0.0.1:6379> RPOP list1
"G"
```

5.删除整个列表

DEL list1

```
127.0.0.1:6379> del list1
(integer) 1
127.0.0.1:6379> LLEN list1
(integer) 0
```

第8章 HASH操作

1.mysql数据格式如何缓存到redis

```
mysql数据格式:
user表
id name job age
1 boss it 18
2 wei it 24
3 cookz it 30

hash类型存储格式:
key field1 value field2 value field3 value
 user:1 name boss job it age 18
 user:2 name wei job it age 18
 user:3 name cookz job it age 18
```

2.创建一个HASH数据

```
HMSET user:1 name boss job it age 18
HMSET user:2 name wei job it age 24
HMSET user:3 name cooz job it age 30
```

3.查看hash里的指定字段的值

```
select name from user where id = 1 ;

127.0.0.1:6379> HMGET user:1 name
```

```
1) "boss"

127.0.0.1:6379> HMGET user:1 name job age
1) "boss"
2) "it"
3) "18"

```

4.查看hash里的所有字段的值

```
select * from user where id = 1 ;

127.0.0.1:6379> HGETALL user:1
1) "name"
2) "boss"
3) "job"
4) "it"
5) "age"
6) "18"

```

第9章 集合操作

1.创建集合

```
SADD set1 1 2 3
SADD set2 1 3 5 7
```

2.查看集合成员

```
SMEMBERS set1
127.0.0.1:6379> SMEMBERS set1
1) "1"
2) "2"
3) "3"
127.0.0.1:6379> SMEMBERS set2
1) "1"
2) "3"
3) "5"
```

```
4) "7"
```

3.查看集合的交集

```
127.0.0.1:6379> SINTER set1 set2
1) "1"
2) "3"
127.0.0.1:6379> SINTER set2 set1
1) "1"
2) "3"
```

4.查看集合的差集

```
127.0.0.1:6379> SMEMBERS set1
1) "1"
2) "2"
3) "3"

127.0.0.1:6379> SMEMBERS set2
1) "1"
2) "3"
3) "5"
4) "7"

127.0.0.1:6379> SDIFF set1 set2
1) "2"

127.0.0.1:6379> SDIFF set2 set1
1) "5"
2) "7"
```

5.查看集合的并集

```
127.0.0.1:6379> SUNION set1 set2
1) "1"
2) "2"
3) "3"
4) "5"
5) "7"
```

第10章 有序集合

有序创建集合

1.添加有序集合

```
127.0.0.1:6379> ZADD sz6 90 haitao
(integer) 1
127.0.0.1:6379> ZADD sz6 85 guanru
(integer) 1
127.0.0.1:6379> ZADD sz6 99 junhui
(integer) 1
```

2.查看有序集合成员

```
127.0.0.1:6379> ZCARD sz6
(integer) 3
```

3.计算某个成员分数

```
127.0.0.1:6379> ZSCORE sz6 junhui
"99"
127.0.0.1:6379> ZSCORE sz6 guanru
"85"
```

4.按照升序查看成员名次

```
127.0.0.1:6379> ZRANGE sz6 0 -1
1) "guanru"
2) "haitao"
3) "junhui"
```

```
127.0.0.1:6379> ZRANGE sz6 0 -1 WITHSCORES
1) "guanru"
2) "85"
3) "haitao"
4) "90"
5) "junhui"
6) "99"
```

5.按照降序查看成员名次

```
127.0.0.1:6379> ZREVRANGE sz6 0 -1
1) "junhui"
2) "haitao"
3) "guanru"
127.0.0.1:6379> ZREVRANGE sz6 0 -1 WITHSCORES
1) "junhui"
2) "99"
3) "haitao"
4) "90"
5) "guanru"
6) "85"
```

6.删除成员

```
127.0.0.1:6379> ZREM sz6 junhui
(integer) 1
```

7.返回指定分数范围的成员

```
127.0.0.1:6379> ZRANGEBYSCORE sz6 88 90 WITHSCORES
1) "haitao"
2) "90"
127.0.0.1:6379> ZRANGEBYSCORE sz6 80 90 WITHSCORES
1) "guanru"
2) "85"
3) "haitao"
4) "90"
```

8.增加成员分数

```
127.0.0.1:6379> ZINCRBY sz6 10 haitao
"100"
127.0.0.1:6379> ZINCRBY sz6 -10 haitao
"90"
```

第11章 持久化　很重要

1.RDB持久化和AOF持久化

RDB：**类似于快照，当前内存里的数据的状态持久化到硬盘**

优点：压缩格式/恢复速度快

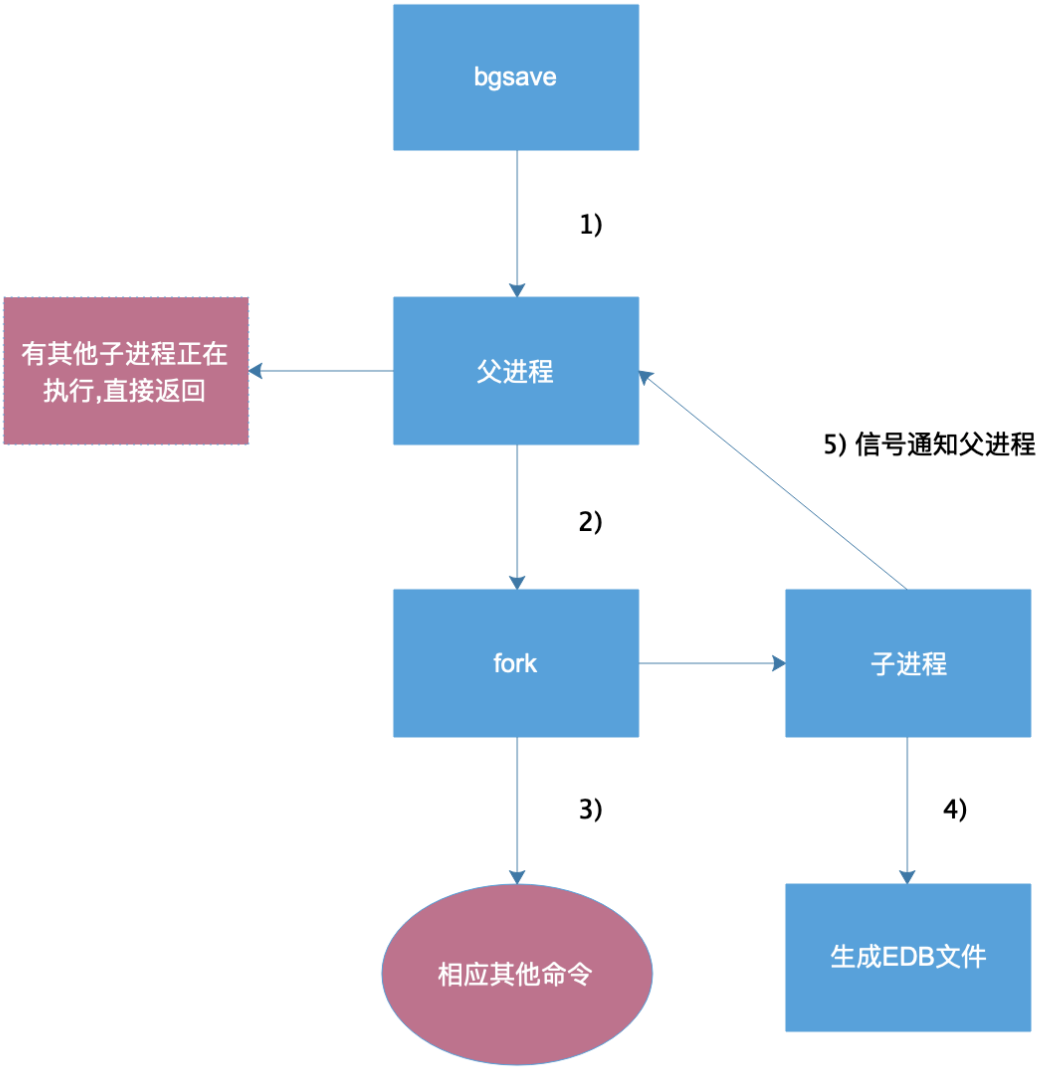
缺点：不是实时的，可能会丢数据，操作比较重量

AOF：类似于mysql的binlog，可以设置成每秒/每次操作都以追加的形式保存在日志文件中

优点：安全，最多只损失1秒的数据，具备一定的可读性

缺点：文件比较大，恢复速度慢

2.RDB持久化流程图



3.配置RDB持久化

```
save 900 1
save 300 10
save 60 10000
dbfilename redis.rdb
dir /data/redis_6379/
```

4.RDB持久化结论：

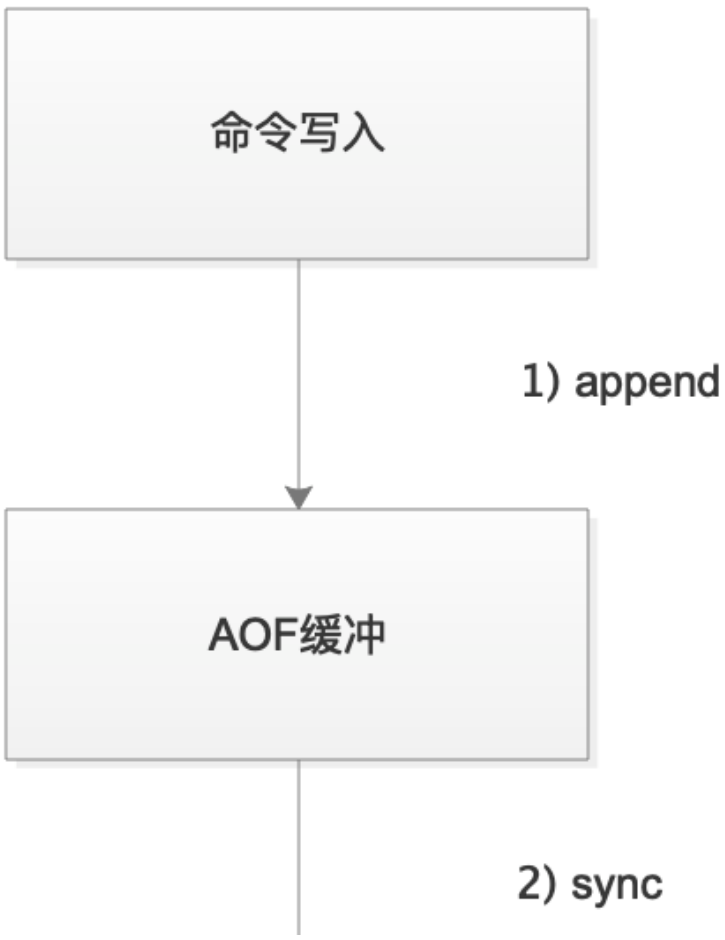
```
没配置save参数时：
1.shutdown/pkill/kill都不会持久化保存
2.可以手动执行bgsave

配置save参数时：
1.shutdown/pkill/kill均会自动触发bgsave持久化保存数据
2.pkill -9 不会触发持久化
```

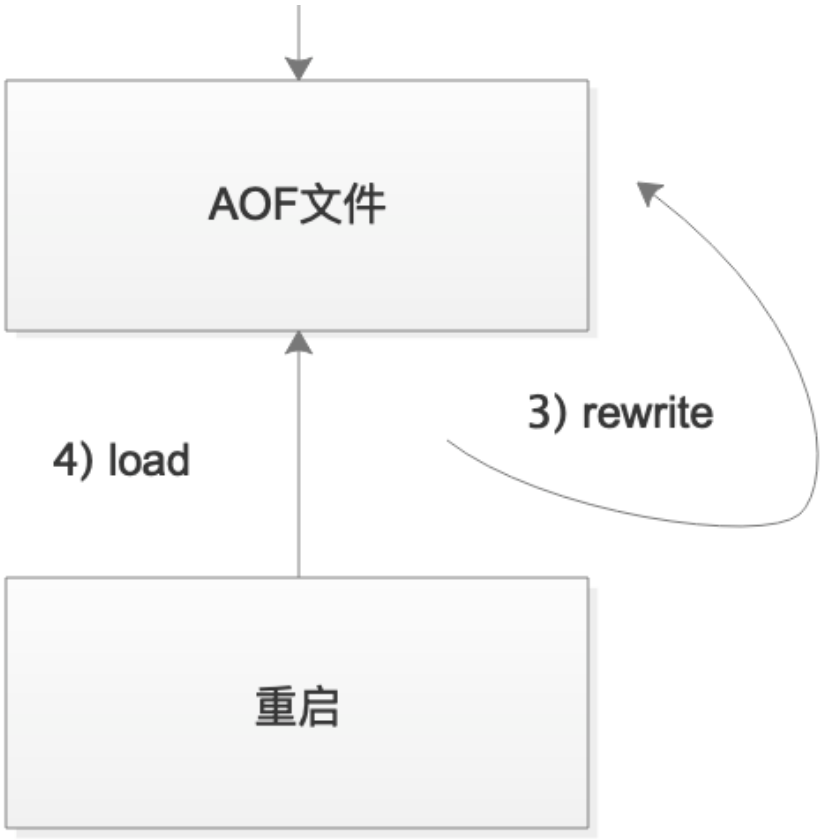
```
恢复时：
1.持久化数据文件名要和配置文件里定义的一样才能被识别
2.RDB文件只有一个数据文件，迁移和备份只要这一个RDB文件即可

注意：
RDB高版本兼容低版本，低版本不能兼容高版本
3.x >> 5.X >> OK
5.x >> 3.x >> NoOK
```

日志内容：

```
8952:M 13 Apr 2020 17:33:12.947 # User requested shutdown...
8952:M 13 Apr 2020 17:33:12.947 * Saving the final RDB snapshot before exiting.
8952:M 13 Apr 2020 17:33:12.947 * DB saved on disk
8952:M 13 Apr 2020 17:33:12.947 * Removing the pid file.
8952:M 13 Apr 2020 17:33:12.947 # Redis is now ready to exit, bye bye...
```

4.AOF流程图





5.AOF持久化配置

```
appendonly yes
appendfilename "redis.aof"
appendfsync everysec #每秒持久化
```

6.AOF重写机制

```
执行的命令 aof记录 redis里的数据
set k1 v1 set k1 k1

set k2 v2 set k1 k1 k2
 set k2

set k3 v3 set k1 k1 k2 k3
 set k2
 set k3

del k1 set k1 k2 k3
```

```
 set k2
 set k3
 del k1

del k2 set k1 k3
 set k2
 set k3
 del k1
 del k2

aof文件里实际有意义的只有一条记录：
set k3
```

7.AOF和RDB读取实验

实验背景：

aof和rdb同时存在，redis重启会读取哪一个数据?

实验步骤：

```
set k1 v1
set k2 v2
bgsave rbd保存 k1 k2
mv redis.rdb /opt/

flushall
set k3 v3
set k4 v4 aof保存 k3 k4
mv redis.aof /opt/

redis-cli shutdown
rm -rf /data/redis_6379/*
mv /opt/redis.aof /data/redis_6379/
mv /opt/redis.rdb /data/redis_6379/

systemctl start redis
```

实验结论： 面试可能问

当aof和rdb同时存在的时候，redis会优先读取aof的内容

8.AOF模拟故障

损坏实验结论：

```
1.aof修复命令不要用，因为他的修复方案非常粗暴，一刀切，从出错的地方到最后全部删除
2.任何操作之前，先备份数据
```

kill -9 实验:

```
for i in {1..10000};do redis-cli set key_${i} v_${i} && echo "${i} is ok";done
ps -ef|grep redis|grep -v grep|awk '{print "kill -9",$2}'
```

结论：

1.aof相对比较安全，最多丢失1秒数据

9.如果设置了过期时间，恢复数据后会如何处理?

```
1.aof文件会记录下过期时间
2.恢复的时候会去对比过期时间和当前时间，如果超过了，就删除key
3.key的过期时间不受备份影响
```

10.AOF和RDB如何选择

```
https://redis.io/topics/persistence
1.开启混合模式
2.开启aof
3.不开启rdb
4.rdb采用定时任务的方式定时备份
5.可以从库开启RDB进行备份
```