Redis 主从集群搭建及容灾部署(哨兵 sentinel)

Redis 也用了一段时间了，记录一下相关集群搭建及配置详解，方便后续使用查阅。

1、提纲

- Redis 安装
- 整体架构
- Redis 主从结构搭建
- Redis 容灾部署（哨兵 sentinel）
- Redis 常见问题

2、Redis 安装

发行版：CentOS-6.6 64bit
内核：2.6.32-504.el6.x86_64
CPU：intel-i7 3.6G
内存：2G

2.1、下载 redis，选择合适的版本

```
[root@rocket software]# wget
http://download.redis.io/releases/redis-2.8.17.tar.gz
[root@rocket software]# cd redis-2.8.17
[root@rocket redis-2.8.17]# make
[root@rocket redis-2.8.17]# make test
cd src && make test
make[1]: Entering directory `/home/software/redis-2.8.17/src'
You need tcl 8.5 or newer in order to run the Redis test
make[1]: *** [test] Error 1
make[1]: Leaving directory `/home/software/redis-2.8.17/src'
make: *** [test] Error 2
```

2.2、make test 报错，安装 tcl

```
[root@rocket software]# wget
http://prdownloads.sourceforge.net/tcl/tcl8.5.18-src.tar.gz
[root@rocket software]# tar -zxvf tcl8.5.18-src.tar.gz
[root@rocket software]# cd tcl8.5.18
```

```
[root@rocket tcl8.5.18]# cd unix/
[root@rocket unix]# ./configure;make;make test;make install
```

2.3、tcl 安装成功，继续测试 redis 的安装情况

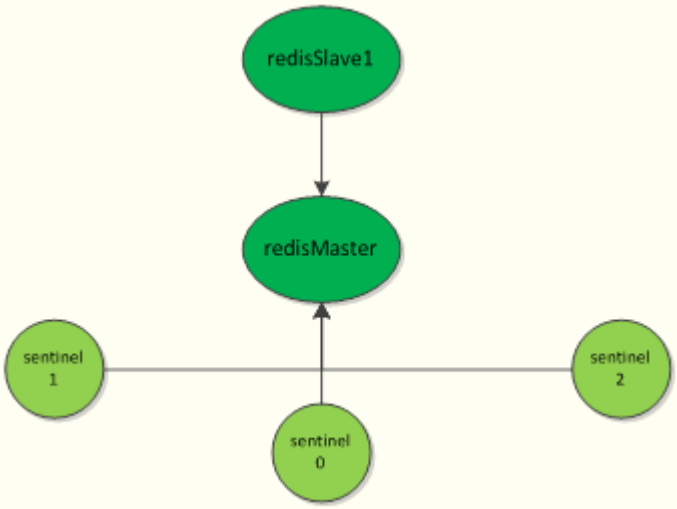
```
[root@rocket redis-2.8.17]# make test
……
Cleanup: may take some time... OK
make[1]: Leaving directory `/home/software/redis-2.8.17/src'
```

说明 redis 安装正常

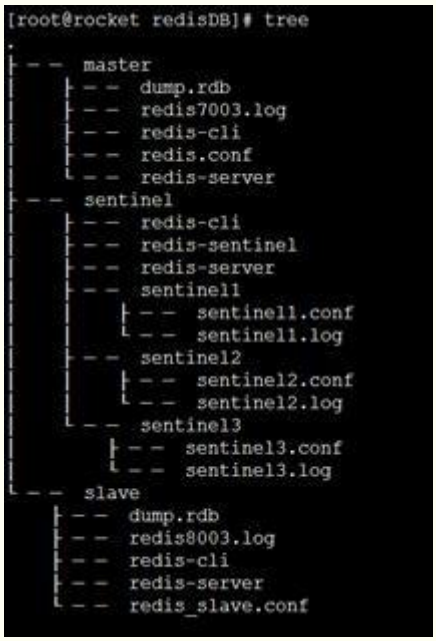
3、整体架构

3.1、整体架构图

这里是本文所搭建集群的整体架构，使用主从结构+哨兵（sentinel）来进行容灾。



3.2、目录结构



4、Redis 主从结构搭建

4.1、搭建 redis master

4.1.1、拷贝可执行文件

```
[root@rocket master]# pwd
/usr/local/redisDB/master
[root@rocket master]# cp /home/software/redis-2.8.17/src/redis-cli .
[root@rocket master]# cp /home/software/redis-2.8.17/src/redis-server .
```

4.1.2、配置文件 redis.conf

```
# 守护进程模式
daemonize yes

# pid file
pidfile /var/run/redis.pid

# 监听端口
port 7003
```

```
# TCP 接收队列长度，受/proc/sys/net/core/somaxconn 和 tcp_max_syn_backlog 这两个内核参数的影响
tcp-backlog 511

# 一个客户端空闲多少秒后关闭连接(0 代表禁用，永不关闭)
timeout 0

# 如果非零，则设置 SO_KEEPALIVE 选项来向空闲连接的客户端发送 ACK
tcp-keepalive 60

# 指定服务器调试等级
# 可能值：
# debug （大量信息，对开发/测试有用）
# verbose （很多精简的有用信息，但是不像 debug 等级那么多）
# notice （适量的信息，基本上是你生产环境中需要的）
# warning （只有很重要/严重的信息会记录下来）
loglevel notice

# 指明日志文件名
logfile "./redis7003.log"

# 设置数据库个数
databases 16

# 会在指定秒数和数据变化次数之后把数据库写到磁盘上
# 900 秒（15 分钟）之后，且至少 1 次变更
# 300 秒（5 分钟）之后，且至少 10 次变更
# 60 秒之后，且至少 10000 次变更
save 900 1
save 300 10
save 60 10000



# 默认如果开启 RDB 快照(至少一条 save 指令)并且最新的后台保存失败，Redis 将会停止接受写操作
# 这将使用户知道数据没有正确的持久化到硬盘，否则可能没人注意到并且造成一些灾难
stop-writes-on-bgsave-error yes

# 当导出到 .rdb 数据库时是否用 LZF 压缩字符串对象
rdbcompression yes

# 版本 5 的 RDB 有一个 CRC64 算法的校验和放在了文件的最后。这将使文件格式更加可靠。
rdbchecksum yes
```

```
# 持久化数据库的文件名
dbfilename dump.rdb

# 工作目录
dir ./

# 当 master 服务设置了密码保护时，slav 服务连接 master 的密码
masterauth 0234kz9*l

# 当一个 slave 失去和 master 的连接，或者同步正在进行中，slave 的行为可以有两种：
#
# 1) 如果 slave-serve-stale-data 设置为 "yes" (默认值)，slave 会继续响应客户端请求，
# 可能是正常数据，或者是过时了的数据，也可能是还没获得值的空数据。
# 2) 如果 slave-serve-stale-data 设置为 "no"，slave 会回复"正在从 master 同步
# （SYNC with master in progress）"来处理各种请求，除了 INFO 和 SLAVEOF 命令。
slave-serve-stale-data yes

# 你可以配置 salve 实例是否接受写操作。可写的 slave 实例可能对存储临时数据比较有用(因为写入 salve
# 的数据在同 master 同步之后将很容易被删除
slave-read-only yes

# 是否在 slave 套接字发送 SYNC 之后禁用 TCP_NODELAY?
# 如果你选择“yes”Redis 将使用更少的 TCP 包和带宽来向 slaves 发送数据。但是这将使数据传输到 slave
# 上有延迟，Linux 内核的默认配置会达到 40 毫秒
# 如果你选择了 "no" 数据传输到 salve 的延迟将会减少但要使用更多的带宽
repl-disable-tcp-nodelay no

# slave 的优先级是一个整数展示在 Redis 的 Info 输出中。如果 master 不再正常工作了，哨兵将用它来
# 选择一个 slave 提升=升为 master。
# 优先级数字小的 salve 会优先考虑提升为 master，所以例如有三个 slave 优先级分别为 10，100，25，
# 哨兵将挑选优先级最小数字为 10 的 slave。
# 0 作为一个特殊的优先级，标识这个 slave 不能作为 master，所以一个优先级为 0 的 slave 永远不会被
# 哨兵挑选提升为 master
slave-priority 100



# 密码验证
```

```
# 警告：因为 Redis 太快了，所以外面的人可以尝试每秒 150k 的密码来试图破解密码。这意味着你需要
# 一个高强度的密码，否则破解太容易了
requirepass 0234kz9*l

# redis 实例最大占用内存，不要用比设置的上限更多的内存。一旦内存使用达到上限，Redis 会根据选定的回收策略（参见：
# maxmemmory-policy）删除 key
maxmemory 3gb

# 最大内存策略：如果达到内存限制了，Redis 如何选择删除 key。你可以在下面五个行为里选：
# volatile-lru -> 根据 LRU 算法删除带有过期时间的 key。
# allkeys-lru -> 根据 LRU 算法删除任何 key。
# volatile-random -> 根据过期设置来随机删除 key，具备过期时间的 key。
# allkeys->random -> 无差别随机删，任何一个 key。
# volatile-ttl -> 根据最近过期时间来删除（辅以 TTL），这是对于有过期时间的 key
# noeviction -> 谁也不删，直接在写操作时返回错误。
maxmemory-policy volatile-lru

# 默认情况下，Redis 是异步的把数据导出到磁盘上。这种模式在很多应用里已经足够好，但 Redis 进程
# 出问题或断电时可能造成一段时间的写操作丢失(这取决于配置的 save 指令)。
#
# AOF 是一种提供了更可靠的替代持久化模式，例如使用默认的数据写入文件策略（参见后面的配置）
# 在遇到像服务器断电或单写情况下 Redis 自身进程出问题但操作系统仍正常运行等突发事件时，Redis
# 能只丢失 1 秒的写操作。
#
# AOF 和 RDB 持久化能同时启动并且不会有问题。
# 如果 AOF 开启，那么在启动时 Redis 将加载 AOF 文件，它更能保证数据的可靠性。
appendonly no

# aof 文件名
appendfilename "appendonly.aof"

# fsync() 系统调用告诉操作系统把数据写到磁盘上，而不是等更多的数据进入输出缓冲区。
# 有些操作系统会真的把数据马上刷到磁盘上；有些则会尽快去尝试这么做。
#
# Redis 支持三种不同的模式：
#
# no：不要立刻刷，只有在操作系统需要刷的时候再刷。比较快。
```

```
# always：每次写操作都立刻写入到 aof 文件。慢，但是最安全。
# everysec：每秒写一次。折中方案。
appendfsync everysec

# 如果 AOF 的同步策略设置成 "always" 或者 "everysec"，并且后台的存储进程（后台存储或写入 AOF
# 日志）会产生很多磁盘 I/O 开销。某些 Linux 的配置下会使 Redis 因为 fsync()系统调用而阻塞很久。
# 注意，目前对这个情况还没有完美修正，甚至不同线程的 fsync() 会阻塞我们同步的 write(2)调用。
#
# 为了缓解这个问题，可以用下面这个选项。它可以在 BGSAVE 或 BGREWRITEAOF 处理时阻止主进程进行 fsync()。
#
# 这就意味着如果有子进程在进行保存操作，那么 Redis 就处于"不可同步"的状态。
# 这实际上是说，在最差的情况下可能会丢掉 30 秒钟的日志数据。（默认 Linux 设定）
#
# 如果你有延时问题把这个设置成"yes"，否则就保持"no"，这是保存持久数据的最安全的方式。
no-appendfsync-on-rewrite yes

# 自动重写 AOF 文件
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb

# AOF 文件可能在尾部是不完整的（这跟 system 关闭有问题，尤其是 mount ext4 文件系统时
# 没有加上 data=ordered 选项。只会发生在 os 死时，redis 自己死不会不完整）。
# 那 redis 重启时 load 进内存的时候就有问题了。
# 发生的时候，可以选择 redis 启动报错，并且通知用户和写日志，或者 load 尽量多正常的数据。
# 如果 aof-load-truncated 是 yes，会自动发布一个 log 给客户端然后 load（默认）。
# 如果是 no，用户必须手动 redis-check-aof 修复 AOF 文件才可以。
# 注意，如果在读取的过程中，发现这个 aof 是损坏的，服务器也是会退出的，
# 这个选项仅仅用于当服务器尝试读取更多的数据但又找不到相应的数据时。
aof-load-truncated yes

# Lua 脚本的最大执行时间，毫秒为单位
lua-time-limit 5000

# Redis 慢查询日志可以记录超过指定时间的查询
slowlog-log-slower-than 10000

# 这个长度没有限制。只是要主要会消耗内存。你可以通过 SLOWLOG RESET 来回收内存。
```

```
slowlog-max-len 128

# redis 延时监控系统在运行时会采样一些操作，以便收集可能导致延时的数据根源。
# 通过 LATENCY 命令 可以打印一些图样和获取一些报告，方便监控
# 这个系统仅仅记录那个执行时间大于或等于预定时间（毫秒）的操作，
# 这个预定时间是通过 latency-monitor-threshold 配置来指定的，
# 当设置为 0 时，这个监控系统处于停止状态
latency-monitor-threshold 0

# Redis 能通知 Pub/Sub 客户端关于键空间发生的事件，默认关闭
notify-keyspace-events ""

# 当 hash 只有少量的 entry 时，并且最大的 entry 所占空间没有超过指定的限制时，会用一种节省内存的
# 数据结构来编码。可以通过下面的指令来设定限制
hash-max-ziplist-entries 512
hash-max-ziplist-value 64

# 与 hash 似，数据元素较少的 list，可以用另一种方式来编码从而节省大量空间。
# 这种特殊的方式只有在符合下面限制时才可以用
list-max-ziplist-entries 512
list-max-ziplist-value 64

# set 有一种特殊编码的情况：当 set 数据全是十进制 64 位有符号整型数字构成的字符串时。
# 下面这个配置项就是用来设置 set 使用这种编码来节省内存的最大长度。
set-max-intset-entries 512

# 与 hash 和 list 相似，有序集合也可以用一种特别的编码方式来节省大量空间。
# 这种编码只适合长度和元素都小于下面限制的有序集合
zset-max-ziplist-entries 128
zset-max-ziplist-value 64

# HyperLogLog 稀疏结构表示字节的限制。该限制包括
# 16 个字节的头。当 HyperLogLog 使用稀疏结构表示
# 这些限制，它会被转换成密度表示。
# 值大于 16000 是完全没用的，因为在该点
# 密集的表示是更多的内存效率。
# 建议值是 3000 左右，以便具有的内存好处，减少内存的消耗
hll-sparse-max-bytes 3000

# 启用哈希刷新，每 100 个 CPU 毫秒会拿出 1 个毫秒来刷新 Redis 的主哈希表（顶级键值映射表）
activerehashing yes
```

```
# 客户端的输出缓冲区的限制,可用于强制断开那些因为某种原因从服务器读取数据的速度不够快的客户端
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60

# 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
hz 10

# 当一个子进程重写 AOF 文件时，如果启用下面的选项，则文件每生成 32M 数据会被同步
aof-rewrite-incremental-fsync yes
```

4.1.3、启动 master

```
[root@rocket master]# ./redis-server ./redis.conf
[root@rocket master]# ps axu|grep redis
root      24000  0.1  0.7
137356  7440 ?        Ssl  23:28   0:00 ./redis-server *:7003
```

4.1.4、使用客户端连接测试

```
[root@rocket master]# ./redis-cli -a 0234kz9*l -p 7003
127.0.0.1:7003> select 1
OK
127.0.0.1:7003[1]> set name zhangsan
OK
127.0.0.1:7003[1]> get name
"zhangsan"
127.0.0.1:7003[1]> quit
```

可以看到，redis 启动成功并且可以开始读写数据。

4.2、搭建 redis slave

slave 的配置和 master 基本一致，只需要修改相应的 pidfile，端口，日志文件名，并配上 master 的地址和认证密码。

4.2.1、配置文件 redis_slave.conf（和 redis master 差异的地方）

```
# pid file
pidfile /var/run/redis_slave.pid
```

```
# 监听端口
port 8003

# 指明日志文件名
logfile "./redis8003.log"

# 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 127.0.0.1 7003

# 当 master 服务设置了密码保护时，slav 服务连接 master 的密码
masterauth 0234kz9*l
```

4.2.2、启动 slave 并查看数据同步情况

```
[root@rocket slave]# ./redis-server ./redis_slave.conf
[root@rocket slave]# ./redis-cli -a 0234kz9*l -p 8003
127.0.0.1:8003> select 1
OK
127.0.0.1:8003[1]> get name
"zhangsan"
可以看到，master 中设置的 key-value 已经成功同步过来。
```

5、Redis 容灾部署（哨兵 Sentinel）

5.1、哨兵的作用

```
1. 监控：监控主从是否正常
2. 通知：出现问题时，可以通知相关人员
3. 故障迁移：自动主从切换
4. 统一的配置管理：连接者询问 sentinel 取得主从的地址
```

5.2、Raft 分布式算法

5.2.1、Raft 分布式算法简介

```
1. 主要用途：用于分布式系统，系统容错，以及选出领头羊
2. 作者：Diego Ongaro，毕业于哈佛
3. 目前用到这个算法的项目有：
a. CoreOS ： 见下面
```

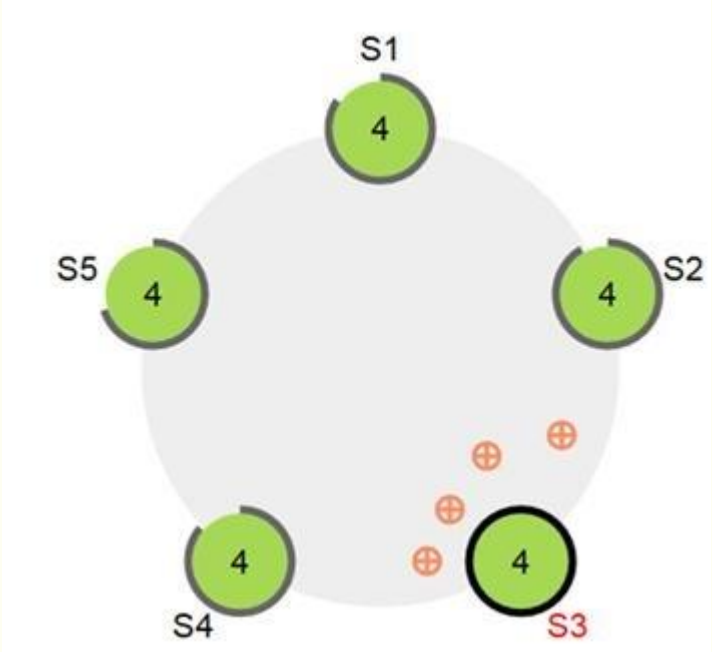
b. ectd : a distributed, consistent shared configuration
c. LogCabin : 分布式存储系统
d. redis sentinel : redis 的监控系统

5.2.2、Sentinel 使用的 Raft 算法核心：原则

1. 所有 sentinel 都有选举的领头羊的权利
2. 每个 sentinel 都会要求其他 sentinel 选举自己为领头羊(主要由发现 redis 客观下线的 sentinel 先发起选举)
3. 每个 sentinel 只有一次选举的机会
4. 采用先到先得的原则
5. 一旦加入到系统了，则不会自动清除(这一点很重要，why?)
6. 每个 sentinel 都有唯一的 uid，不会因为重启而变更
7. 达到领头羊的条件是 N/2 + 1 个 sentinel 选择了自己
8. 采用配置纪元，如果一次选举出现脑裂，则配置纪元会递增，进入下一次选举，所有 sentinel 都会处于统一配置纪元，以最新的为标准。

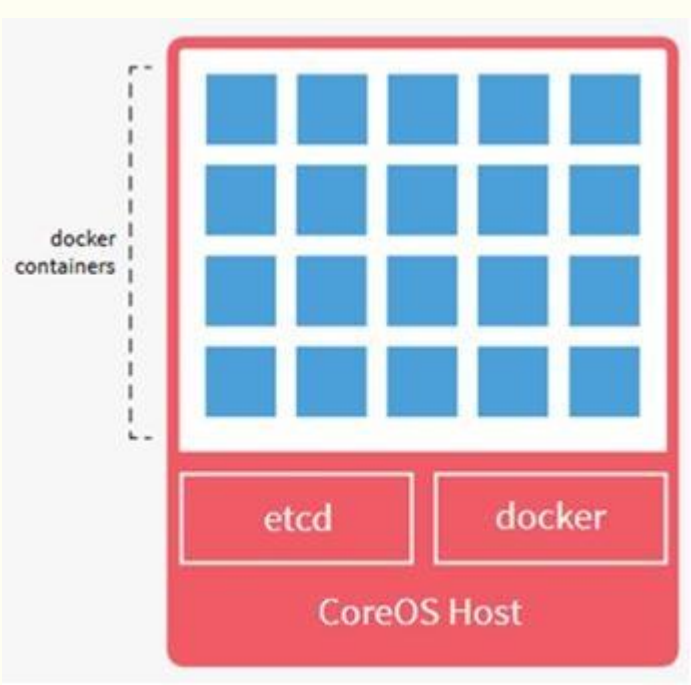
5.2.3、Raft 算法核心(可视图)

Raft Visualization (算法演示)



5.2.4、Raft 分布式算法的应用

coreos:云计算新星 Docker 正在以火箭般的速度发展，与它相关的生态圈也渐入佳境，CoreOS 就是其中之一。CoreOS 是一个全新的、面向数据中心设计的 Linux 操作系统，在 2014 年 7 月发布了首个稳定版本，目前已经完成了 800 万美元的 A 轮融资。



5.3、Sentinel 实现 Redis 容灾部署

5.3.1、三哨兵架构

```
[root@rocket sentinel]# tree
.
├── redis-cli
├── redis-sentinel
├── redis-server
├── sentinel1
│   ├── sentinel1.conf
│   └── sentinel1.log
├── sentinel2
│   ├── sentinel2.conf
│   └── sentinel2.log
└── sentinel3
    ├── sentinel3.conf
└── sentinel3.log
```

哨兵一配置 sentinel1.conf

```
# Example sentinel.conf

# port <sentinel-port>
port 26371
```

```
# 守护进程模式
daemonize yes

# 指明日志文件名
logfile "./sentinel1.log"

# 工作路径，sentinel 一般指定/tmp 比较简单
dir ./

# 哨兵监控这个 master，在至少 quorum 个哨兵实例都认为 master down 后把 master 标记为 odown
# （objective down 客观 down；相对应的存在 sdown，subjective down，主观 down）状态。
# slaves 是自动发现，所以你没必要明确指定 slaves。
sentinel monitor TestMaster 127.0.0.1 7003 1

# master 或 slave 多长时间（默认 30 秒）不能使用后标记为 s_down 状态。
sentinel down-after-milliseconds TestMaster 1500

# 若 sentinel 在该配置值内未能完成 failover 操作(即故障时 master/slave 自动切换)，则认为本次 failover 失败。
sentinel failover-timeout TestMaster 10000

# 设置 master 和 slaves 验证密码
sentinel auth-pass TestMaster 0234kz9*l

sentinel config-epoch TestMaster 15
sentinel leader-epoch TestMaster 8394

# #除了当前哨兵，还有哪些在监控这个 master 的哨兵
sentinel known-sentinel TestMaster 127.0.0.1 26372
0aca3a57038e2907c8a07be2b3c0d15171e44da5
sentinel known-sentinel TestMaster 127.0.0.1 26373
ac1ef015411583d4b9f3d81cee830060b2f29862

sentinel current-epoch 8394
```

哨兵二配置 sentinel2.conf

```
# Example sentinel.conf

# port <sentinel-port>
```

```
port 26372

# 守护进程模式
daemonize yes

# 指明日志文件名
logfile "./sentinel2.log"

# 工作路径，sentinel 一般指定/tmp 比较简单
dir ./

# 哨兵监控这个 master，在至少 quorum 个哨兵实例都认为 master down 后把 master 标记为 odown
# （objective down 客观 down；相对应的存在 sdown，subjective down，主观 down）状态。
# slaves 是自动发现，所以你没必要明确指定 slaves。
sentinel monitor TestMaster 127.0.0.1 7003 1

# master 或 slave 多长时间（默认 30 秒）不能使用后标记为 s_down 状态。
sentinel down-after-milliseconds TestMaster 1500

# 若 sentinel 在该配置值内未能完成 failover 操作(即故障时 master/slave 自动切换)，则认为本次 failover 失败。
sentinel failover-timeout TestMaster 10000

# 设置 master 和 slaves 验证密码
sentinel auth-pass TestMaster 0234kz9*l

sentinel config-epoch TestMaster 15
sentinel leader-epoch TestMaster 8394

# #除了当前哨兵，还有哪些在监控这个 master 的哨兵
sentinel known-sentinel TestMaster 127.0.0.1 26371
b780bbc20fdea6d3789637053600c5fc58dd0690
sentinel known-sentinel TestMaster 127.0.0.1 26373
ac1ef015411583d4b9f3d81cee830060b2f29862

sentinel current-epoch 8394
```

哨兵三配置 sentinel3.conf

```
# Example sentinel.conf
```

```
# port <sentinel-port>
port 26373

# 守护进程模式
daemonize yes

# 指明日志文件名
logfile "./sentinel3.log"

# 工作路径，sentinel 一般指定/tmp 比较简单
dir ./

# 哨兵监控这个 master，在至少 quorum 个哨兵实例都认为 master down 后把 master 标记为 odown
# （objective down 客观 down；相对应的存在 sdown，subjective down，主观 down）状态。
# slaves 是自动发现，所以你没必要明确指定 slaves。
sentinel monitor TestMaster 127.0.0.1 7003 1

# master 或 slave 多长时间（默认 30 秒）不能使用后标记为 s_down 状态。
sentinel down-after-milliseconds TestMaster 1500

# 若 sentinel 在该配置值内未能完成 failover 操作(即故障时 master/slave 自动切换)，则认为本次 failover 失败。
sentinel failover-timeout TestMaster 10000

# 设置 master 和 slaves 验证密码
sentinel auth-pass TestMaster 0234kz9*l

sentinel config-epoch TestMaster 15
sentinel leader-epoch TestMaster 8394

# #除了当前哨兵，还有哪些在监控这个 master 的哨兵
sentinel known-sentinel TestMaster 127.0.0.1 26371
b780bbc20fdea6d3789637053600c5fc58dd0690
sentinel known-sentinel TestMaster 127.0.0.1 26372
0aca3a57038e2907c8a07be2b3c0d15171e44da5

sentinel current-epoch 8394
```

5.3.2、在 sentinel 中查看所监控的 master 和 slave

```
[root@rocket sentinel]# ./redis-cli -p 26371
127.0.0.1:26371> SENTINEL masters
```

```
1)  1) "name"
    2) "TestMaster"
    3) "ip"
    4) "127.0.0.1"
    5) "port"
    6) "7003"
    7) "runid"
    8) "de0896e3799706bda49cb92048776e233841e25d"
    9) "flags"
   10) "master"
127.0.0.1:26371> SENTINEL slaves TestMaster
1)  1) "name"
    2) "127.0.0.1:8003"
    3) "ip"
    4) "127.0.0.1"
    5) "port"
    6) "8003"
    7) "runid"
    8) "9b2a75596c828d6d605cc8529e96edcf951de25d"
    9) "flags"
   10) "slave"
```

查看当前的 master

```
127.0.0.1:26371> SENTINEL get-master-addr-by-name TestMaster
1) "127.0.0.1"
2) "7003"
```

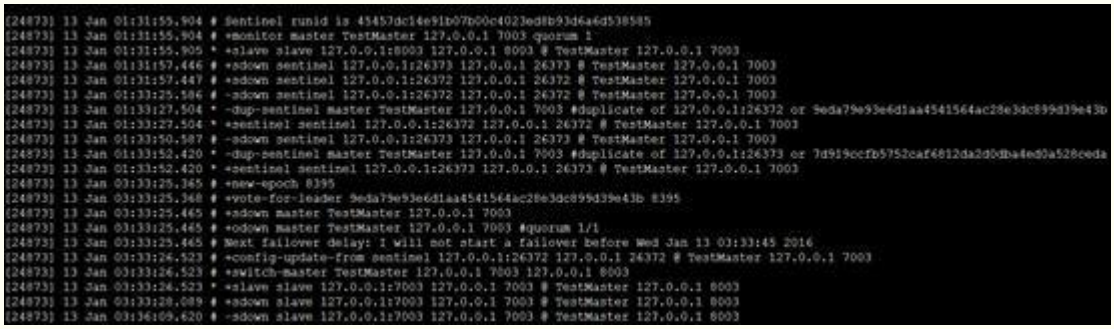
5.3.3、停掉 master，查看容灾切换情况

```
[root@rocket master]# ps axu|grep redis
root      24000  0.2  0.9
137356  9556 ?        Ssl  Jan12   0:30 ./redis-server *:7003
root      24240  0.2  0.7
137356  7504 ?        Ssl  Jan12   0:26 ./redis-server *:8003
root      24873  0.3  0.7
137356  7524 ?        Ssl  01:31   0:25 ../redis-sentinel *:26371
root      24971  0.3  0.7
137356  7524 ?        Ssl  01:33   0:25 ../redis-sentinel *:26372
root      24981  0.3  0.7
137356  7520 ?        Ssl  01:33   0:25 ../redis-sentinel *:26373
root      24995  0.0  0.5  19404  5080
pts/2    S+   01:34   0:00 ./redis-cli -p 26371
root      25969  0.0  0.0  103252   844  pts/0    S+   03:33   0:00  grep
redis
[root@rocket master]# kill -QUIT 24000
```

再查看 master，发现已经 master 已经切换为原来的 slave

```
127.0.0.1:26371> SENTINEL get-master-addr-by-name TestMaster
1) "127.0.0.1"
2) "8003"
```

查看 sentinel 日志



启动原来的 master，发现变成了 slave

```
[root@rocket master]# ./redis-server ./redis.conf
127.0.0.1:26371> SENTINEL slaves TestMaster
1)  1) "name"
    2) "127.0.0.1:7003"
    3) "ip"
    4) "127.0.0.1"
    5) "port"
6) "7003"
```

发现主从发生了对调。

5.3.4、sentinel 自动发现

每个Sentinel 都订阅了被它监视的所有主服务器和从服务器的__sentinel__:hello 频道，查找之前未出现过的 sentinel（looking for unknown sentinels）。当一个 Sentinel 发现一个新的 Sentinel 时，它会将新的 Sentinel 添加到一个列表中，这个列表保存了 Sentinel 已知的，监视同一个主服务器的所有其他 Sentinel。

```
127.0.0.1:7003[1]> SUBSCRIBE __sentinel__:hello
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "__sentinel__:hello"
3) (integer) 1
1) "message"
2) "__sentinel__:hello"
3)
"127.0.0.1,26373,7d919ccfb5752caf6812da2d0dba4ed0a528ceda,8436,TestMaster,127.0.0.1,7003,8436"
1) "message"
2) "__sentinel__:hello"
3)
```

```
"127.0.0.1,26372,9eda79e93e6d1aa4541564ac28e3dc899d39e43b,8436,TestMast
er,127.0.0.1,7003,8436"
1) "message"
2) "__sentinel__:hello"
3)
"127.0.0.1,26371,8d63bebfbca9e1205a43bc13b52079de6015758e,8436,TestMast
er,127.0.0.1,7003,8436"
```

6、Redis 常见问题

最大内存问题：要设置好最大内存，以防不停的申请内存，造成系统内存都被用完。
Fork 进程问题：'vm.overcommit_memory = 1'这一个选项要加到系统的配置中，防止 fork 因内存不足而失败。
密码问题：需要设置复杂一些，防止暴力破解。