🚚 超详细Redis7.X 安装教程

1. Redis 安装

本教程将演示在 linux 环境下安装 Redis7，给大家最简单，最快捷的安装方式，其中包括单机部署、主从部署、哨兵部署、集群部署的安装以及相应的架构介绍。

1.1. 单机部署

1.1.1. 检查安装 gcc 环境

Redis是由C语言编写的，它的运行需要C环境，因此我们需要先安装gcc。

```
-- 关闭防火墙
systemctl stop firewalld.service
-- 状态
firewall-cmd --state
-- 卸载防火墙
yum remove firewalld

-- 检查版本
gcc --version
-- 安装 gcc
yum install gcc
```

```
[root@localhost ~]# gcc --version
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-44)
Copyright © 2015 Free Software Foundation, Inc.
本程序是自由软件；请参看源代码的版权声明。本软件没有任何担保；
包括没有适销性和某一专用目的下的适用性担保。
```

1.1.2. 下载安装 Redis

```
-- 安装应用养成良好习惯，文件归类
mkdir -p /opt/software/redis

-- 进入redis文件夹，使用wget下载
cd /opt/software/redis
wget https://download.redis.io/redis-stable.tar.g
z

-- 解压下载的redis包
tar -xzf redis-stable.tar.gz

-- 进入redis-stable目录，然后使用make install 编译并安
装，安装完成后 /usr/local/bin 会生成相应的服务
cd redis-stable
make install

-- 检查是否成功生成
ll /usr/local/bin
```

```
[root@localhost redis-stable]# ll /usr/local/bin
总用量 29244
-rwxr-xr-x. 1 root root  6900344 5月  21 17:09 redis-benchmark
lrwxrwxrwx. 1 root root       12 5月  21 17:09 redis-check-aof -> redis-server
lrwxrwxrwx. 1 root root       12 5月  21 17:09 redis-check-rdb -> redis-server
-rwxr-xr-x. 1 root root  7620224 5月  21 17:09 redis-cli
lrwxrwxrwx. 1 root root       12 5月  21 17:09 redis-sentinel -> redis-server
-rwxr-xr-x. 1 root root 15418856 5月  21 17:09 redis-server
```

- 文件介绍：

  redis-benchmark：性能测试工具

  redis-check-aof：修复有问题的 aof 文件

  redis-check-rdb：修复有问题的rdb文件

  redis-sentinel：Redis集群使用

  redis-server：Redis服务器启动命令

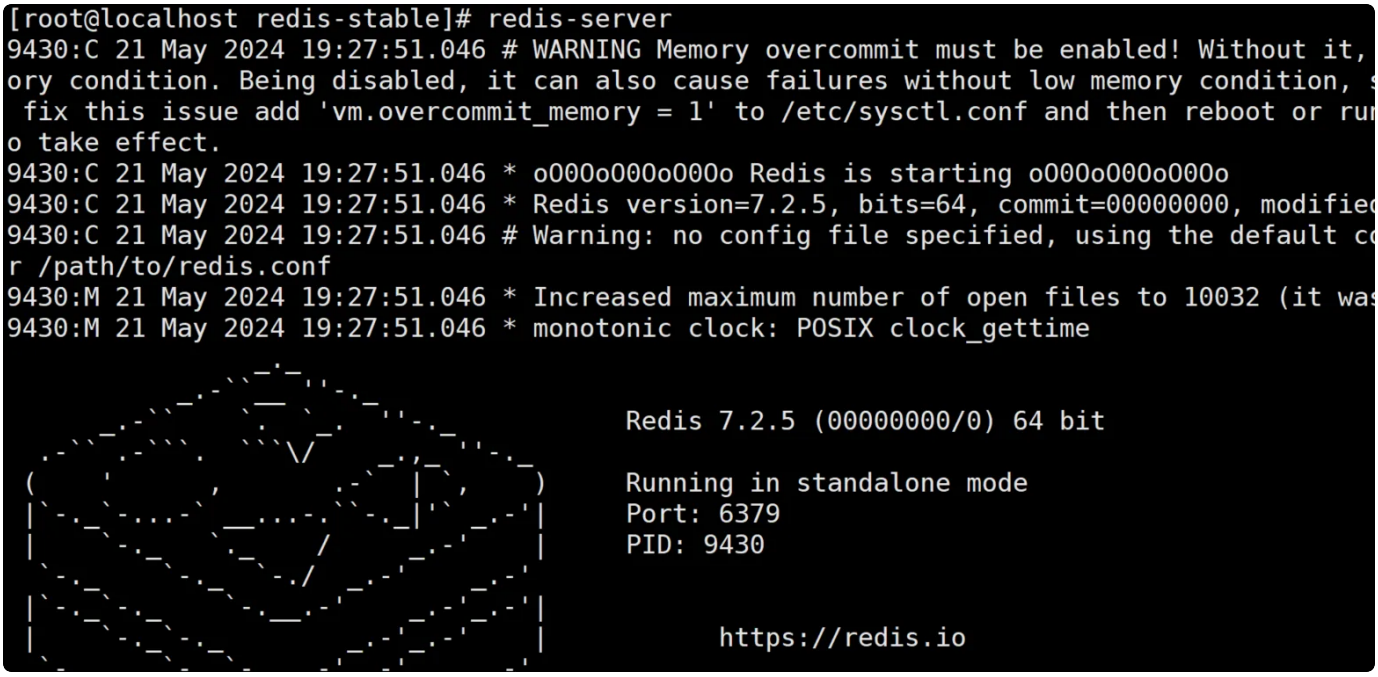
  redis-cli：客户端，操作入口

1.1.3. 启动 Redis

到这里其实我们可以在使用 /opt/software/redis/redis-stable/src 或者 /usr/local/bin 目录下的 redis-server 启动 Redis 服务了。

```
Redis 源码路径下启动
./src/redis-server

使用usr/local/bin 路径下启动（该目录下）
redis-server
```



1.1.4. 配置Redis

前面的启动方式无法再后台运行，退出之后直接关闭了 Redis 服务，所以我们还需要针对 Redis 做一些设置。

```
-- 修改当前Redis目录下的 Reids.conf 文件
vim redis.conf
```

需要修改的内容如下：如果大家使用 vim 打开后没有行号，可以在打开 vim 后输入：“：set number”。

```
bind * -::*                                    #87行，修改bind 项，* -::* 支持远程连接
daemonize yes                                  #309行，开启守护进程，后台运行
logfile /opt/software/redis/redis-stable/redis.log     #355行，指定日志文件目录
dir /opt/software/redis                        #510行，指定工作目录
requirepass 1qaz@WSX                           #1044行，给默认用户设置密码，主要是使用 redis-cli 连接 redis-server 时，需要通过密码校验。自行学习，可以不设置。
protected-mode no                              #111行，允许远程连接        如果不设置密码必须讲此设置关闭。
```

修改完成后，使用配置文件启动 Redis，并使用 redis-cli 连接测试，需要注意由于前面我们配置了安全密码，所以连接后需要先验证密码，否则会报错。

```
redis-server redis.conf
redis-cli
auth 1qaz@WSX
```

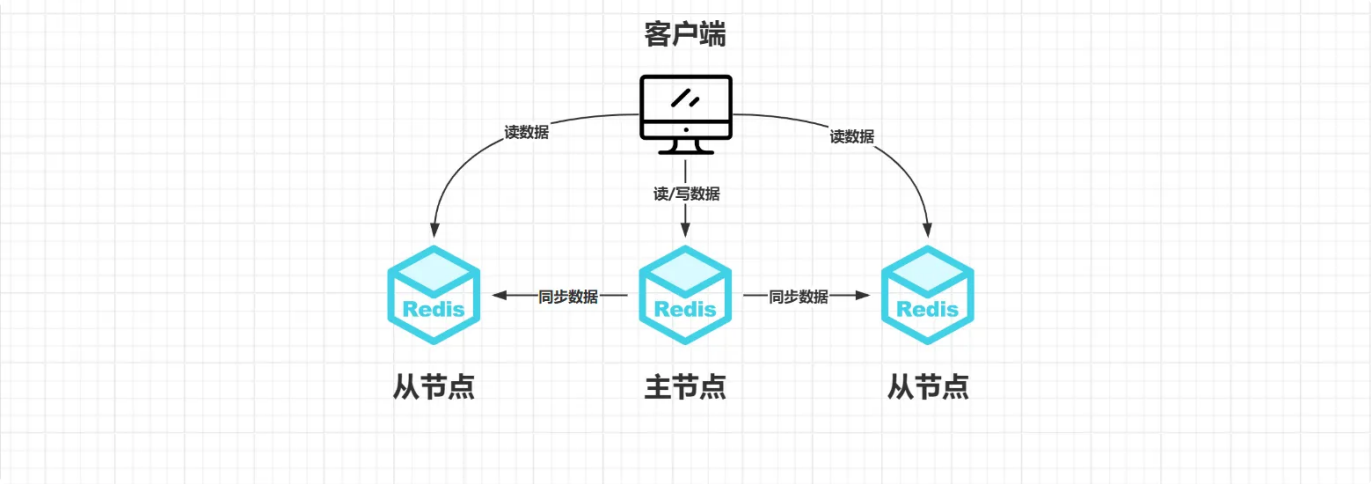
```
[root@localhost redis-stable]# redis-cli
127.0.0.1:6379> keys *
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth 1qaz@WSX
OK
127.0.0.1:6379> keys *
(empty array)
```

1.1.5. 退出 OR 关闭 redis

```
-- 退出redis
quit

-- 关闭redis
redis-cli shutdown
```

1.2. 主从部署（Master-Slave Replication）



主从复制，是指将一台Redis服务器的数据，复制到其他的Redis服务器。前者称为主节点(Master)，后者称为从节点(Slave)；数据的复制是单向的，只能由主节点到从节点。默认情况下，每台Redis服务器都是主节点；且一个主节点可以有多个从节点(或没有从节点)，但一个从节点只能有一个主节点。

1.2.1. 主从复制的作用

a）数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。

b）故障恢复：当主节点出现问题时，可以由从节点提供服务，实

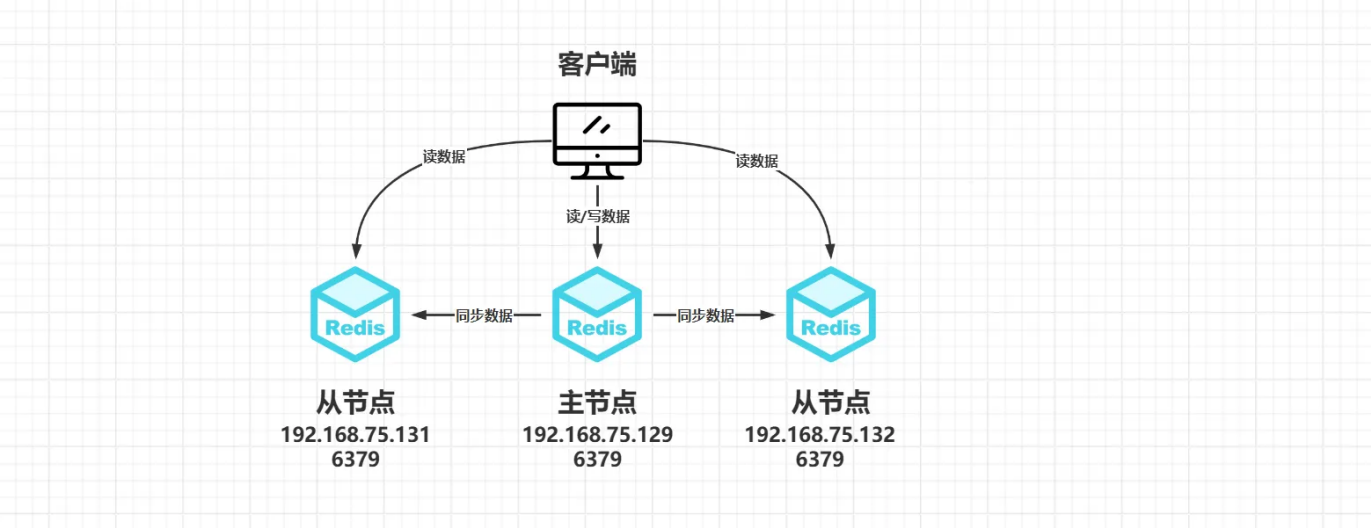
现快速的故障恢复；实际上是一种服务的冗余。

c）负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写Redis数据时应用连接主节点，读Redis数据时应用连接从节点），分担服务器负载；尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高Redis服务器的并发量。

d）高可用基石：除了上述作用以外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是Redis高可用的基础。

1.2.2. 主从复制部署

整体架构图



主节点不需要做任何改变，从节点都需要修改配置加上主节点信息，配置完成后，可以再主库检查从节点信息

```
# 添加主节点信息
replicaof 192.168.75.129 6379
```

```
525 # 2) Redis replicas are able to perform a partial resynchronization with the
526 #    master if the replication link is lost for a relatively small amount of
527 #    time. You may want to configure the replication backlog size (see the next
528 #    sections of this file) with a sensible value depending on your needs.
529 # 3) Replication is automatic and does not need user intervention. After a
530 #    network partition replicas automatically try to reconnect to masters
531 #    and resynchronize with them.
532 #
533 # replicaof <masterip> <masterport>
534 replicaof 192.168.75.129 6379
535 # If the master is password protected (using the "requirepass" configuration
536 # directive below) it is possible to tell the replica to authenticate before
537 # starting the replication synchronization process, otherwise the master will
538 # refuse the replica request.
539 #
```

```
1   -- 主节点查看从节点信息
2   info Replication
```

```
127.0.0.1:6379> info Replication
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.75.131,port=6379,state=online,offset=270,lag=1
slave1:ip=192.168.75.132,port=6379,state=online,offset=270,lag=1
master_failover_state:no-failover
master_replid:8a2a65aba289acecff42673d1097850b0365cf29
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:270
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:270
```

1.2.3. 主从复制缺点

- 复制延时，信号衰减

  由于所有的写操作都是现在master上操作，然后同步更新到slave上，所以从master同步到slave机器上有一定的延迟，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。

- master挂了如何办?

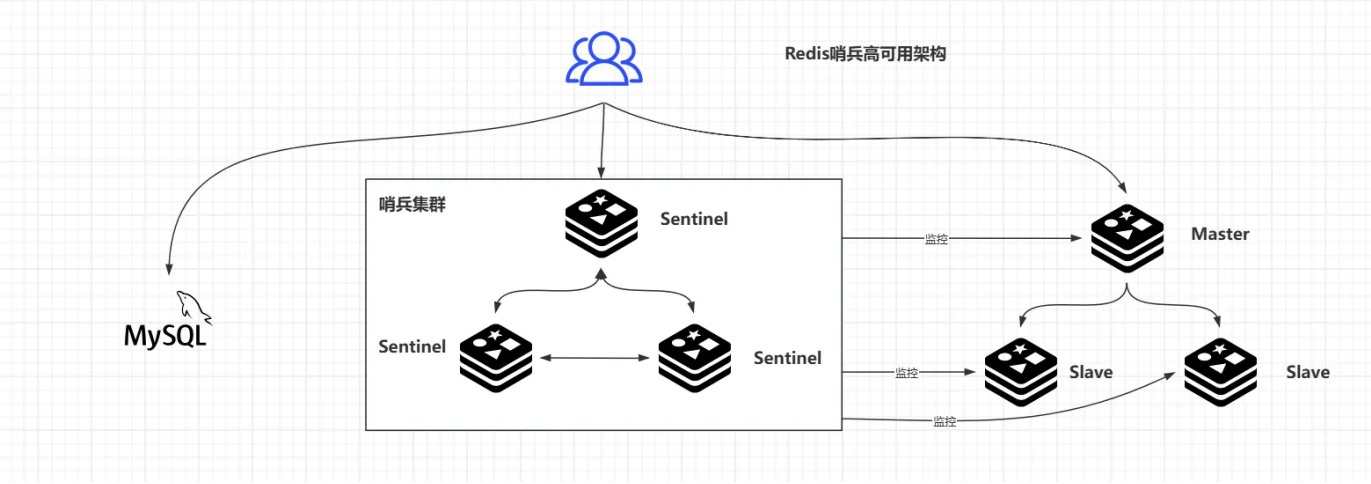
默认情况下，不会在slave节点中自动重选一个master，每次都要人工干预。

1.3. 哨兵部署（Sentinel）

Redis的主从复制主要用于实现数据的冗余备份和读分担，并不是为了提供高可用性。因此在系统高可用方面，单纯的主从架构无法很好的保证整个系统高可用

1.3.1. 哨兵模式的原理

Redis哨兵模式是通过在独立的哨兵节点上运行特定的哨兵进程来实现的。这些哨兵进程监控主从节点的状态，并在发现故障时自动完成故障发现和转移，并通知应用方，实现高可用性。



1.3.2. 哨兵

在启动时，每个哨兵节点会执行选举过程，其中一个哨兵节点被选为领导者（leader），负责协调其他哨兵节点。

- **选举过程：**

每个在线的哨兵节点都可以成为领导者，每个哨兵节点会向其它哨兵发is-master-down-by-addr命令，征求判断并要求将自己设置为领导者；
当其它哨兵收到此命令时，可以同意或者拒绝它成为领导者；
如果哨兵发现自己在选举的票数大于等于num(sentinels)/2+1时，将成为领导者，如果没有超过，继续选举。

- **监控主从节点：**

  哨兵节点通过发送命令周期性地检查主从节点的健康状态，包括主节点是否在线、从节点是否同步等。如果哨兵节点发现主节点不可用，它会触发一次故障转移。

- **故障转移：**

  一旦主节点被判定为不可用，哨兵节点会执行故障转移操作。它会从当前的从节点中选出一个新的主节点，并将其他从节点切换到新的主节点。这样，系统可以继续提供服务而无需人工介入。

- **故障转移过程：**

  由Sentinel节点定期监控发现主节点是否出现了故障：sentinel会向master发送心跳PING来确认master是否存活，如果master在“一定时间范围”内不回应PONG 或者是回复了一个错误消息，那么这个sentinel会主观地(单方面地)认为这个master已经不可用了。

- **确认主节点：**

- 过滤掉不健康的（下线或断线），没有回复过哨兵ping响应的从节点
- 选择从节点优先级最高的
- 选择复制偏移量最大，此指复制最完整的从节点
- 当主节点出现故障， 由领导者负责处理主节点的故障转移。

- **客户端重定向：**

  哨兵节点会通知客户端新的主节点的位置，使其能够与新的主节点建立连接并发送请求。这确保了客户端可以无缝切换到新的主节点，继续进行操作。

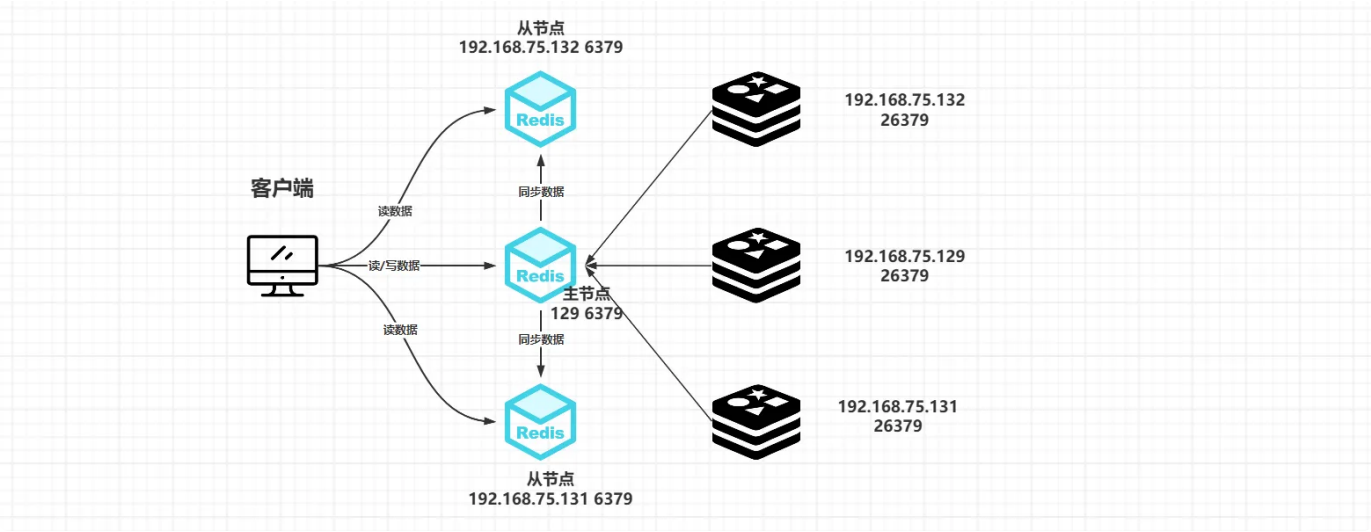
此外，哨兵节点还负责监控从节点的状态。如果从节点出现故障，哨兵节点可以将其下线，并在从节点恢复正常后重新将其加入集群。

1.3.3. 客观下线

当主观下线的节点是主节点时，此时该哨兵3节点会通过指令sentinel is-masterdown-by-addr寻求其它哨兵节点对主节点的判断，当超过quorum（选举）个数，此时哨兵节点则认为该主节点确实有问题，这样就客观下线了，大部分哨兵节点都同意下线操作，也就说是客观下线。

1.3.4. 哨兵模式部署

整体架构图



3 个机器都需要修改 sentinel.conf 配置，配置完成之后先从主节点开始启动哨兵。

```
1  protected-mode no
     #6行，关闭保护模式
2  daemonize yes
     #15行，指定sentinel为后台启动
3  logfile /opt/software/redis/redis-stable/sentin
   el.log                          #34行，指定日志存放路径
4  dir /opt/software/redis
     #73行，指定数据库存放路径
5  sentinel monitor mymaster 192.168.75.129 6379 2
     #93行，修改 指定该哨兵节点监控20.0.0.20:6379这个主
   节点，该主节点的名称是mymaster，最后的2的含义与主节点的
   故障判定有关：至少需要2个哨兵节点同意，才能判定主节点故
   障并进行故障转移
6  sentinel down-after-milliseconds mymaster 30000
     #134行，判定服务器down掉的时间周期，默认30000毫秒
    （30秒）
7  sentinel failover-timeout mymaster 180000
     #234行，故障节点的最大超时时间为180000（180秒）
```

启动后检查哨兵状态：

```
redis-cli -p 26379 info sentinel
```

```
[root@localhost redis-stable]# redis-cli -p 26379 info sentinel
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_tilt_since_seconds:-1
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=192.168.75.129:6379,slaves=2,sentinels=3
```

故障模拟

```
-- 可以杀掉主节点的进程，也可以直接停掉主节点服务
ps aux | grep redis
redis-cli shutdown

-- 观察哨兵日志，129 主节点下线，重新选举131为主节点
tail -f sentinel.log

--重新启动 129 服务 并观察日志，129加入主从，此时主节点为131服务
redis-server redis.conf
tail -f sentinel.log
redis-cli -p 26379 info sentinel
-- 观察哨兵日志
tail -f sentinel.log

-- 停止哨兵
redis-cli -p 26379 shutdown
```

```
3366:X 23 May 2024 14:38:32.046 # +sdown master mymaster 192.168.75.129 6379 主观下线
3366:X 23 May 2024 14:38:32.094 * Sentinel new configuration saved on disk
3366:X 23 May 2024 14:38:32.094 # +new-epoch 4 新一轮选举
3366:X 23 May 2024 14:38:32.096 * Sentinel new configuration saved on disk
3366:X 23 May 2024 14:38:32.096 # +vote-for-leader a04107c0b37ab2b32b852c1d719a57f1a7b89877 4
3366:X 23 May 2024 14:38:32.098 # +odown master mymaster 192.168.75.129 6379 #quorum 3/2 客观下线
3366:X 23 May 2024 14:38:32.098 * Next failover delay: I will not start a failover before Thu May 23 14:44:32 2024
3366:X 23 May 2024 14:38:33.112 # +config-update-from sentinel a04107c0b37ab2b32b852c1d719a57f1a7b89877 192.168.75.13
75.129 6379
3366:X 23 May 2024 14:38:33.112 # +switch-master mymaster 192.168.75.129 6379 192.168.75.131 6379 131为新主节点
3366:X 23 May 2024 14:38:33.112 * +slave slave 192.168.75.132:6379 192.168.75.132 6379 @ mymaster 192.168.75.131 6379
3366:X 23 May 2024 14:38:33.112 * +slave slave 192.168.75.129:6379 192.168.75.129 6379 @ mymaster 192.168.75.131 6379
3366:X 23 May 2024 14:38:33.113 * Sentinel new configuration saved on disk 保存配置
```


```
3366:X 23 May 2024 14:38:32.046 # +sdown master mymaster 192.168.75.129 6379 主观下线
3366:X 23 May 2024 14:38:32.094 * Sentinel new configuration saved on disk
3366:X 23 May 2024 14:38:32.094 # +new-epoch 4 新一轮选举
3366:X 23 May 2024 14:38:32.096 * Sentinel new configuration saved on disk
3366:X 23 May 2024 14:38:32.096 # +vote-for-leader a04107c0b37ab2b32b852c1d719a57f1a7b89877 4
3366:X 23 May 2024 14:38:32.098 # +odown master mymaster 192.168.75.129 6379 #quorum 3/2 客观下线
3366:X 23 May 2024 14:38:32.098 * Next failover delay: I will not start a failover before Thu May 23 14:44:32 2024
3366:X 23 May 2024 14:38:33.112 # +config-update-from sentinel a04107c0b37ab2b32b852c1d719a57f1a7b89877 192.168.75.13
75.129 6379
3366:X 23 May 2024 14:38:33.112 # +switch-master mymaster 192.168.75.129 6379 192.168.75.131 6379 131为新主节点
3366:X 23 May 2024 14:38:33.112 * +slave slave 192.168.75.132:6379 192.168.75.132 6379 @ mymaster 192.168.75.131 6379
3366:X 23 May 2024 14:38:33.112 * +slave slave 192.168.75.129:6379 192.168.75.129 6379 @ mymaster 192.168.75.131 6379
3366:X 23 May 2024 14:38:33.113 * Sentinel new configuration saved on disk 保存配置
```


```
[root@localhost redis-stable]# redis-cli -p 26379 info sentinel
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_tilt_since_seconds:-1
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=192.168.75.131:6379,slaves=2,sentinels=3
```

```
-- 切换到131服务，已经为主节点。
redis-cli info replication
```

```
[root@localhost redis-stable]# redis-cli info replication
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.75.132,port=6379,state=online,offset=12558399,lag=0
slave1:ip=192.168.75.129,port=6379,state=online,offset=12558113,lag=1
master_failover_state:no-failover
master_replid:60161ee69642889c49dac4f9e3f7767ad006fa4f
master_replid2:47a63933569f8fbc2d847d259db9ffc95d501750
master_repl_offset:12558542
second_repl_offset:12403321
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:11500711
repl_backlog_histlen:1057832
```

当触发了哨兵选举之后，会再后台更改 redis.conf 与 sentinel.conf，可以检查每台机器的文件末尾的数据

```
cat redis.conf
cat sentinel.conf
```

```
# Generated by CONFIG REWRITE
save 3600 1
save 300 100
save 60 10000
latency-tracking-info-percentiles 50 99 99.9

user default on nopass sanitize-payload ~* &* +@all

replicaof 192.168.75.131 6379
```

```
# Generated by CONFIG REWRITE
save 3600 1
save 300 100
save 60 10000
latency-tracking-info-percentiles 50 99 99.9

user default on nopass sanitize-payload ~* &* +@all

replicaof 192.168.75.131 6379
```

1.3.5. 哨兵使用建议

- 哨兵节点的数量应为多个，哨兵本身应该集群，保证高可用
- 哨兵节点数应该是奇数
- 各个哨兵结点的配置应一致
- 如果哨兵节点部署在Docker等容器里面，尤其要注意端口号的正确映射

1.3.6. 哨兵模式：并不能保证数据零丢失

1. **复制延迟**：
    - 在主从复制中，从节点的数据是异步复制自主节点的。这意味着在主节点故障时，从节点可能还没有完全同步最新的数据，从而导致数据丢失。

2. **故障检测和转移时间**：
    - Sentinel 检测到主节点故障并执行故障转移需要一定的时间。在这段时间内，主节点可能已经接收了一些写操作，但这些操作尚未被复制到从节点。
3. **网络分区**：
    - 在发生网络分区（网络分裂）的情况下，一部分节点可能与主节点失去联系。如果此时主节点继续处理写操作，那么在网络恢复之前，这些操作可能不会被复制到从节点。
4. **多个从节点同时故障**：
    - 如果所有的从节点同时故障或在故障转移之前与主节点失联，那么在主节点故障时，将没有可用的从节点来提升为主节点。

1.4. 集群部署（Cluster）

Redis 集群是 Redis 的一种分布式运行模式，它通过分片（sharding）来提供数据的自动分区和管理，从而实现数据的高可用性和可扩展性。

在集群模式下，数据被分割成多个部分（称为槽或slots），分布在多个 Redis 节点上。

集群中的节点分为主节点和从节点：**主节点**负责读写请求和集群信息的维护；**从节点**只进行主节点数据和状态信息的复制。

1.4.1. Redis集群的作用

数据分区：数据分区(或称数据分片)是集群最核心的功能。 集群将数据分散到多个节点，一方面突破了Redis单机内存大小的限制，存储容量大大增加；

另一方面每个主节点都可以对外提供读服务和写服务，大大提高了集群的响应能力。 Redis单机内存大小受限问题，在介绍持久化和主从复制时都有提及；

例如，如果单机内存太大，bgsave和bgrewriteaof的fork操作可能导致主进程阻塞，主从环境下主机切换时可能导致从节点长时间无法提供服务，全量复制阶段主节点的复制缓冲区可能溢出。

高可用：集群支持主从复制和主节点的自动故障转移（与哨兵类似）；当任一节点发生故障时，集群仍然可以对外提供服务。

1.4.2. Redis集群的数据分片

Redis集群引入了哈希槽的概念 Redis集群有16384个哈希槽（编号0-16383） 集群的每个节点负责一部分哈希槽 每个Key通过CRC16校验后对16384取余来决定放置哪个哈希槽，

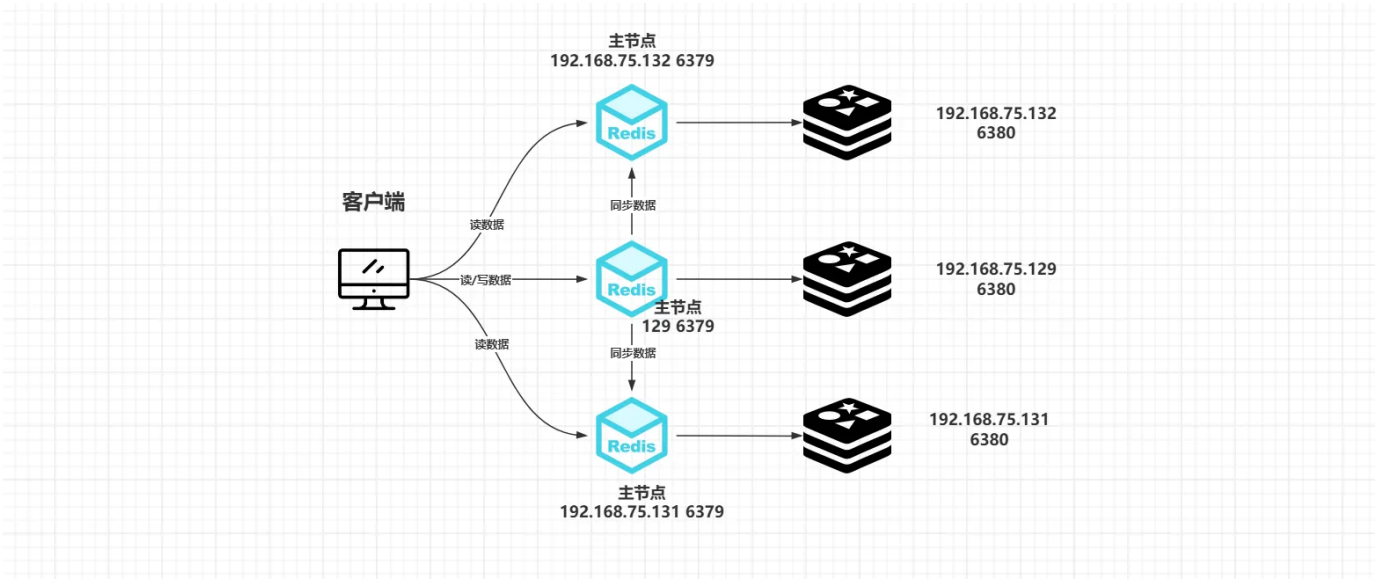
通过这个值，去找到对应的插槽所对应的节点，然后直接自动跳转到这个对应的节点上进行存取操作

- 以3个节点组成的集群为例： 节点A包含0到5460号哈希槽 节点B包含5461到10922号哈希槽 节点C包含10923到16383号哈希槽

- Redis集群的主从复制模型 集群中具有A、B、C三个节点，如果节点B失败了，整个集群就会因缺少5461-10922这个范围的槽而不可以用。

为每个节点添加一个从节点A1、B1、C1整个集群便有三个Master节点和三个slave节点组成，在节点B失败后，集群选举B1位为的主节点继续服务。当B和B1都失败后，集群将不可用

1.4.3. Reids 集群部署



1.4.3.1. redis 环境简述

Redis Cluster被配置为三主三从模式。这意味着每台服务器上的两个Redis节点中，一个节点作为主库（master），另一个作为从库（slave）。

1.4.3.2. redis 集群配置准备

```
-- 创建集群配置文件夹，将下面的配置复制过去，另外两个机器重复这个过程
mkdir -p /opt/software/redis/redis-stable/cluster
mkdir -p /opt/software/redis/cluster
vim ./cluster/redis_6379.conf
vim ./cluster/redis_6380.conf

-- 配置文件准备完成之后，启动所有redis服务，用cluster配置文件
redis-server ./cluster/redis_6379.conf
redis-server ./cluster/redis_6380.conf

-- 检查服务
ps aux | grep redis

-- 创建三主三从集群模式，每一个主节点带一个从节点
redis-cli --cluster create --cluster-replicas 1 192.168.75.129:6379 192.168.75.129:6380 192.168.75.131:6379 192.168.75.131:6380 192.168.75.132:6379 192.168.75.132:6380

-- 查看集群信息
redis-cli cluster info

-- 查看单个节点信息
redis-cli info replication

-- 查看集群节点身份信息
redis-cli cluster nodes
```

```
-- 停止redis服务
redis-cli -p 6379 shutdown
redis-cli -p 6380 shutdown
```

6379配置 Shell

```
# 允许所有的IP地址
bind * -::*
# 后台运行
daemonize yes
# 允许远程连接
protected-mode no
# 开启集群模式
cluster-enabled yes
# 集群节点超时时间
cluster-node-timeout 5000
# 配置数据存储目录
dir "/opt/software/redis/cluster"
# 开启AOF持久化
appendonly yes

# 端口
port 6379
# log日志
logfile "/opt/software/redis/redis-stable/cluster/redis6379.log"
# 集群配置文件
cluster-config-file nodes-6379.conf
# AOF文件名
appendfilename "appendonly6379.aof"
# RBD文件名
dbfilename "dump6379.rdb"
```

6380配置 Shell

```
# 允许所有的IP地址
bind * -::*
# 后台运行
daemonize yes
# 允许远程连接
protected-mode no
# 开启集群模式
cluster-enabled yes
# 集群节点超时时间
cluster-node-timeout 5000
# 配置数据存储目录
dir "/opt/software/redis/cluster"
# 开启AOF持久化
appendonly yes

# 端口
port 6380
# log日志
logfile "/opt/software/redis/redis-stable/cluster/redis6380.log"
# 集群配置文件
cluster-config-file nodes-6380.conf
# AOF文件名
appendfilename "appendonly6380.aof"
# RBD文件名
dbfilename "dump6380.rdb"
```

```
[root@localhost redis-stable]# redis-server ./cluster/redis_6379.conf
[root@localhost redis-stable]# redis-server ./cluster/redis_6380.conf
[root@localhost redis-stable]# ps aux | grep redis
root      13466  0.0  0.4 169172  8636 ?        Ssl  16:24   0:00 redis-server *:6379 [cluster]
root      13472  0.1  0.4 166100  8496 ?        Ssl  16:24   0:00 redis-server *:6380 [cluster]
root      13486  0.0  0.0 112824   988 pts/0    S+   16:24   0:00 grep --color=auto redis
```

```
[root@localhost redis-stable]# redis-cli --cluster create --cluster-replicas 1 192.168.75.129:6
68.75.131:6380 192.168.75.132:6379 192.168.75.132:6380
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 192.168.75.131:6380 to 192.168.75.129:6379
Adding replica 192.168.75.132:6380 to 192.168.75.131:6379
Adding replica 192.168.75.129:6380 to 192.168.75.132:6379
M: 4793d4aec67ba3992d5f44d5c2b3b83b90567151 192.168.75.129:6379
   slots:[0-5460] (5461 slots) master
S: b04fc68b7292a6351cd009e0ab42c34eba028286 192.168.75.129:6380
   replicates 95fb243869704b33d9703f62ca9e7e8af7429fe0
M: 97bc0dabc1af69e5b5e2898e6a7ba27850d75864 192.168.75.131:6379
   slots:[5461-10922] (5462 slots) master
S: f2c8ba007e209f8e17dd17c6fb28b7a790a3e8e4 192.168.75.131:6380
   replicates 4793d4aec67ba3992d5f44d5c2b3b83b90567151
M: 95fb243869704b33d9703f62ca9e7e8af7429fe0 192.168.75.132:6379
   slots:[10923-16383] (5461 slots) master
S: a9f2662d5daef9376f28475f1afacf84234e1699 192.168.75.132:6380
   replicates 97bc0dabc1af69e5b5e2898e6a7ba27850d75864
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
```

节点槽位分配

主从节点分布

1.4.3.3. Redis 集群数据读写

```
-- 连接一个主节点进行写数据
redis-cli info replication
-- 直接连接读写可能会出现以下问题，是因为不同的节点的槽位不同，图中就是提示我们去132:6379进行写入数据
```

```
[root@localhost redis-stable]# redis-cli
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=192.168.75.131,port=6380,state=online,offset=2058,lag=1
master_failover_state:no-failover
master_replid:8cda874e13377b4fac9fa0e105f31a02f9111f3a
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:2058
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:2058
127.0.0.1:6379> set k1 v1
(error) MOVED 12706 192.168.75.132:6379
127.0.0.1:6379> set k2 v2
OK
```

```
-- 不过我们也可以开启路由规则 -c，进行处理
redis-cli -c
-- 重新写入数据，恢复正常。
set k1 b1
```

```
[root@localhost redis-stable]# redis-cli -c
127.0.0.1:6379> keys *
1) "k2"
127.0.0.1:6379> get k2
"v2"
127.0.0.1:6379> set k1 b1
-> Redirected to slot [12706] located at 192.168.75.132:6379
OK
192.168.75.132:6379> keys *
1) "k1"
```

1.4.3.4. 模拟故障转移

```
-- 注意机器ip的区分
-- 将129机器的主节点给干掉(129的6379服务)
redis-cli -p 6379 shutdown
-- 查看129机器从节点工作日志(131的6380日志)
cat redis6380.log
-- 在切换到132机器上查看当前集群节点信息，131:6380已经升为主节点
redis-cli cluster nodes
```

```
7725:S 12 Jun 2024 20:00:31.997 * Start of election delayed for 921 milliseconds (ran
k #0, offset 350).
7725:S 12 Jun 2024 20:00:31.998 # Cluster state changed: fail
7725:S 12 Jun 2024 20:00:32.501 * Connecting to MASTER 192.168.75.129:6379
7725:S 12 Jun 2024 20:00:32.501 * MASTER <-> REPLICA sync started
7725:S 12 Jun 2024 20:00:32.501 # Error condition on socket for SYNC: Connection refu
sed
7725:S 12 Jun 2024 20:00:33.005 * Starting a failover election for epoch 7.
7725:S 12 Jun 2024 20:00:33.008 * Failover election won: I'm the new master.
7725:S 12 Jun 2024 20:00:33.008 * configEpoch set to 7 after successful failover
7725:M 12 Jun 2024 20:00:33.008 * Discarding previously cached master state.
7725:M 12 Jun 2024 20:00:33.008 * Setting secondary replication ID to 5df7787b6c326ea
eafd6dcebf77b9f0941297292, valid up to offset: 351. New replication ID is c6b855e90e0
b7e95f76438874b95bdee9ec25ebd
7725:M 12 Jun 2024 20:00:33.008 * Cluster state changed: ok
```

```
[root@localhost redis-stable]# redis-cli cluster nodes
f3d425596537fef191518fba881d2d56c93d97fd 192.168.75.132:6379@16379 myself,master - 0
1718193667000 5 connected 10923-16383
f2b8a5f82fefd5a3862be7268e5fbc9f262922cb 192.168.75.131:6379@16379 master - 0 1718193
668588 3 connected 5461-10922
0bdb7f61bfec8ae78a3e42e456572a9ac10be4a1 192.168.75.129:6380@16380 slave f3d425596537
fef191518fba881d2d56c93d97fd 0 1718193669000 5 connected
2d1e533201748b23d81da8f3c7bda07a78134eb1 192.168.75.132:6380@16380 slave f2b8a5f82fef
d5a3862be7268e5fbc9f262922cb 0 1718193669596 3 connected
66c5b2450017624a1f8e93684be5315b35de8fb4 192.168.75.131:6380@16380 master - 0 1718193
668588 7 connected 0-5460
31a2f8fc07eda2b5a1a28e0a99fa0e30196ec255 192.168.75.129:6379@16379 master,fail - 1718
193626232 1718193624720 1 disconnected
```

```
-- 在重新启动129.6379服务
redis-server ./cluster/redis_6379.conf
-- 查看129.6379的节点信息，主节点变为从节点
redis-cli -p 6379 info replication
-- 观察131.6380日志，129.6379 重新加入集群
```

```
[root@localhost redis-stable]# redis-server ./cluster/redis_6379.conf
[root@localhost redis-stable]# redis-cli -p 6379 info replication
# Replication
role:slave
master_host:192.168.75.131
master_port:6380
master_link_status:up
master_last_io_seconds_ago:2
master_sync_in_progress:0
slave_read_repl_offset:392
slave_repl_offset:392
slave_priority:100
slave_read_only:1
```

```
7725:M 12 Jun 2024 20:04:53.705 * Replica 192.168.75.129:6379 asks for synchronizatio
n
7725:M 12 Jun 2024 20:04:53.706 * Partial resynchronization not accepted: Replication
 ID mismatch (Replica asked for 'd11a3c0a76115c911be82817829f132e6061957e', my replic
ation IDs are 'c6b855e90e0b7e95f76438874b95bdee9ec25ebd' and '5df7787b6c326eaeafd6dce
bf77b9f0941297292')
7725:M 12 Jun 2024 20:04:53.706 * Delay next BGSAVE for diskless SYNC
7725:M 12 Jun 2024 20:04:53.717 * Clear FAIL state for node 31a2f8fc07eda2b5a1a28e0a9
9fa0e30196ec255 ():master without slots is reachable again.
7725:M 12 Jun 2024 20:04:58.737 * Starting BGSAVE for SYNC with target: replicas sock
ets
7725:M 12 Jun 2024 20:04:58.739 * Background RDB transfer started by pid 7902
```

至此 Redis 部署篇章结束，完结撒花~~~~~

2. 完整的文件目录与配置文件与使用过程中的命令

2.1. 文件目录

手工创建 Shell

```
/opt/software/redis/    -- Redis应用
/opt/software/redis/redis-stable -- Redis应用根目录
/opt/software/redis/cluster  -- Redis集群应用文件目录(日志，快照等信息)
/opt/software/redis/redis-stable/cluster  -- Redis集群配置文件存放路径
```

```
[root@localhost redis]# pwd
/opt/software/redis
[root@localhost redis]# ll
总用量 3412
drwxr-xr-x. 2 root  root      103 5月  22 15:51 appendonlydir
drwxr-xr-x. 3 root  root      113 5月  24 17:08 cluster
-rw-r--r--. 1 root  root      232 5月  23 16:21 dump.rdb
drwxrwxr-x. 9 baili baili    4096 5月  23 15:45 redis-stable
-rw-r--r--. 1 root  root  3483771 5月  19 14:18 redis-stable.tar.gz
```

2.2. 配置文件

2.2.1. 单机Redis配置文件

所在目录：/opt/software/redis/redis-stable

6379配置 Shell

```
bind * -::*
protected-mode no
port 6379
tcp-backlog 511
timeout 0
tcp-keepalive 300
daemonize yes
pidfile /var/run/redis_6379.pid
loglevel notice
logfile /opt/software/redis/redis-stable/redis.log
databases 16
always-show-logo no
set-proc-title yes
proc-title-template "{title} {listen-addr} {server-mode}"
locale-collate ""
stop-writes-on-bgsave-error yes
rdbcompression yes
rdbchecksum yes
dbfilename dump.rdb
rdb-del-sync-files no
dir /opt/software/redis
replica-serve-stale-data yes
replica-read-only yes
repl-diskless-sync yes
repl-diskless-sync-delay 5
repl-diskless-sync-max-replicas 0
repl-diskless-load disabled
```

```
repl-disable-tcp-nodelay no
replica-priority 100
acllog-max-len 128
lazyfree-lazy-eviction no
lazyfree-lazy-expire no
lazyfree-lazy-server-del no
replica-lazy-flush no
lazyfree-lazy-user-del no
lazyfree-lazy-user-flush no
oom-score-adj no
oom-score-adj-values 0 200 800
disable-thp yes
appendonly no
appendfilename "appendonly.aof"
appenddirname "appendonlydir"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
aof-use-rdb-preamble yes
aof-timestamp-enabled no
slowlog-log-slower-than 10000
slowlog-max-len 128
latency-monitor-threshold 0
notify-keyspace-events ""
hash-max-listpack-entries 512
hash-max-listpack-value 64
list-max-listpack-size -2
list-compress-depth 0
```

```
set-max-intset-entries 512
set-max-listpack-entries 128
set-max-listpack-value 64
zset-max-listpack-entries 128
zset-max-listpack-value 64
hll-sparse-max-bytes 3000
stream-node-max-bytes 4096
stream-node-max-entries 100
activerehashing yes
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit replica 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
hz 10
dynamic-hz yes
aof-rewrite-incremental-fsync yes
rdb-save-incremental-fsync yes
jemalloc-bg-thread yes
```

2.2.2. 主从节点配置

所在目录：/opt/software/redis/redis-stable

大家可以将不同服务器的端口设置不同的值，以方便区分。

129.6379配置-主节点 Shell

```
与单机主节点配置一样
```

131.6379配置-从节点 Shell

```
bind * -::*
protected-mode no
port 6379
tcp-backlog 511
timeout 0
tcp-keepalive 300
daemonize yes
pidfile /var/run/redis_6379.pid
loglevel notice
logfile /opt/software/redis/redis-stable/redis.log
databases 16
always-show-logo no
set-proc-title yes
proc-title-template "{title} {listen-addr} {server-mode}"
locale-collate ""
stop-writes-on-bgsave-error yes
rdbcompression yes
rdbchecksum yes
dbfilename dump.rdb
rdb-del-sync-files no
dir /opt/software/redis
replicaof 192.168.75.129 6379
replica-serve-stale-data yes
replica-read-only yes
repl-diskless-sync yes
repl-diskless-sync-delay 5
repl-diskless-sync-max-replicas 0
```

```
repl-diskless-load disabled
repl-disable-tcp-nodelay no
replica-priority 100
acllog-max-len 128
lazyfree-lazy-eviction no
lazyfree-lazy-expire no
lazyfree-lazy-server-del no
replica-lazy-flush no
lazyfree-lazy-user-del no
lazyfree-lazy-user-flush no
oom-score-adj no
oom-score-adj-values 0 200 800
disable-thp yes
appendonly no
appendfilename "appendonly.aof"
appenddirname "appendonlydir"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
aof-use-rdb-preamble yes
aof-timestamp-enabled no
slowlog-log-slower-than 10000
slowlog-max-len 128
latency-monitor-threshold 0
notify-keyspace-events ""
hash-max-listpack-entries 512
hash-max-listpack-value 64
list-max-listpack-size -2
```

```
list-compress-depth 0
set-max-intset-entries 512
set-max-listpack-entries 128
set-max-listpack-value 64
zset-max-listpack-entries 128
zset-max-listpack-value 64
hll-sparse-max-bytes 3000
stream-node-max-bytes 4096
stream-node-max-entries 100
activerehashing yes
client-output-buffer-limit normal 0 0 0
client-output-buffer-limit replica 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60
hz 10
dynamic-hz yes
aof-rewrite-incremental-fsync yes
rdb-save-incremental-fsync yes
jemalloc-bg-thread yes
```

132.6379配置-从节点 Shell

同131.6379配置一样

2.2.3. 哨兵模式

所在目录：/opt/software/redis/redis-stable

主从配置无需修改，直接配置 sentinel 文件，3 个机器配置相同

26379 Shell

```
protected-mode no
port 26379
daemonize yes
pidfile /var/run/redis-sentinel.pid
loglevel notice
logfile /opt/software/redis/redis-stable/sentine
l.log
dir /opt/software/redis
sentinel monitor mymaster 192.168.75.129 6379 2
sentinel down-after-milliseconds mymaster 30000
acllog-max-len 128
sentinel parallel-syncs mymaster 1
sentinel failover-timeout mymaster 180000
sentinel deny-scripts-reconfig yes
SENTINEL resolve-hostnames no
SENTINEL announce-hostnames no
SENTINEL master-reboot-down-after-period mymaste
r 0
```

2.2.4. 集群

所在目录：/opt/software/redis/redis-stable/cluster

3 个机器配置相同

6379 Shell

```
# 允许所有的IP地址
bind * -::*
# 后台运行
daemonize yes
# 允许远程连接
protected-mode no
# 开启集群模式
cluster-enabled yes
# 集群节点超时时间
cluster-node-timeout 5000
# 配置数据存储目录
dir "/opt/software/redis/cluster"
# 开启AOF持久化
appendonly yes

# 端口
port 6379
# log日志
logfile "/opt/software/redis/redis-stable/cluster/redis6379.log"
# 集群配置文件
cluster-config-file nodes-6379.conf
# AOF文件名
appendfilename "appendonly6379.aof"
# RBD文件名
dbfilename "dump6379.rdb"
```

6380 Shell

```
# 允许所有的IP地址
bind * -::*
# 后台运行
daemonize yes
# 允许远程连接
protected-mode no
# 开启集群模式
cluster-enabled yes
# 集群节点超时时间
cluster-node-timeout 5000
# 配置数据存储目录
dir "/opt/software/redis/cluster"
# 开启AOF持久化
appendonly yes

# 端口
port 6380
# log日志
logfile "/opt/software/redis/redis-stable/cluste
r/redis6380.log"
# 集群配置文件
cluster-config-file nodes-6380.conf
# AOF文件名
appendfilename "appendonly6380.aof"
# RBD文件名
dbfilename "dump6380.rdb"
```

2.2.5. 命令

命令汇总 Shell

```
---------------------------------- Redis基础常见命令 ----------------------------------
keys *: 查看当前库所有的key
exists key: 判断某个key是否存在
type key: 查看key值是什么类型
del key: 删除指定的key数据
unlink key: 非阻塞删除，仅仅将keys从keyspace元数据中删除，真正的删除会在后续异步中操作
ttl key: 查看还有多少秒过期，-1表示永不过期，-2表示已过期
expire key: 秒钟，为给定的key设置过期时间
move key dbindex[0-15]: 将当前数据库的key移动到给定的数据库db当中
select dbindex: 切换数据库[0-15]，默认值为0
dbsize: 查看当前数据库key的数量
flushdb: 清空当前库
flusshall: 通杀全部库
---------------------------------- 完整的操作命令 ----------------------------------
-- 关闭防火墙
systemctl stop firewalld.service
-- 状态
firewall-cmd --state
-- 卸载防火墙
yum remove firewalld

---------------------------- 单机部署 ----------------------------

-- 检查版本
gcc --version
```

```
-- 安装 gcc
yum install gcc

-- 安装应用养成良好习惯，文件归类
mkdir -p /opt/software/redis

-- 进入redis文件夹，使用wget下载
cd /opt/software/redis
wget https://download.redis.io/redis-stable.tar.g
z

-- 解压下载的redis包
tar -xzf redis-stable.tar.gz

-- 进入redis-stable目录，然后使用make install 编译并安
装，安装完成后 /usr/local/bin 会生成相应的服务
cd redis-stable
make install

-- 检查是否成功生成
ll /usr/local/bin

Redis 源码路径下启动
./src/redis-server

使用usr/local/bin 路径下启动（该目录下）
redis-server

-- 修改当前Redis目录下的 Reids.conf 文件
vim redis.conf
```

```
-- 启动Redis,使用密码认证登录
redis-server redis.conf
redis-cli -a 1qaz@WSX

-- 退出redis
quit

-- 关闭redis
redis-cli shutdown

--------------------------- 主从部署 ----------------
-----------
-- 主节点查看从节点信息
info Replication

--------------------------- 哨兵部署 ----------------
-----------
-- 可以杀掉主节点的进程，也可以直接停掉主节点服务
ps aux | grep redis
redis-cli shutdown

-- 观察哨兵日志，129 主节点下线，重新选举131为主节点
tail -f sentinel.log

--重新启动 129 服务 并观察日志，129加入主从，此时主节点为
131服务
redis-server redis.conf

tail -f sentinel.log
```

```
redis-cli -p 26379 info sentinel

-- 观察哨兵日志
tail -f sentinel.log

-- 停止哨兵
redis-cli -p 26379 shutdown

-- 切换到131服务，已经为主节点。
redis-cli info replication

-- 查看文件内容
cat redis.conf
cat sentinel.conf

---------------------------- 集群部署 ----------------
-----------
-- 创建集群配置文件夹，将下面的配置复制过去，另外两个机器重复这个过程
mkdir -p /opt/software/redis/redis-stable/cluster
mkdir -p /opt/software/redis/cluster
vim ./cluster/redis_6379.conf
vim ./cluster/redis_6380.conf

-- 配置文件准备完成之后，启动所有redis服务，用cluster配置文件
redis-server ./cluster/redis_6379.conf
redis-server ./cluster/redis_6380.conf
```

```
-- 检查服务
ps aux | grep redis

-- 创建三主三从集群模式，每一个主节点带一个从节点
redis-cli --cluster create --cluster-replicas 1 1
92.168.75.129:6379 192.168.75.129:6380 192.168.7
5.131:6379 192.168.75.131:6380 192.168.75.132:637
9 192.168.75.132:6380

-- 查看集群信息
redis-cli cluster info

-- 查看单个节点信息
redis-cli info replication

-- 查看集群节点身份信息
redis-cli cluster nodes

-- 停止redis服务
redis-cli -p 6379 shutdown
redis-cli -p 6380 shutdown

-- 连接一个主节点进行写数据
redis-cli info replication

-- 注意机器ip的区分
-- 将129机器的主节点给干掉(129的6379服务)
redis-cli -p 6379 shutdown
-- 查看129机器从节点工作日志(131的6380日志)
```

```
cat redis6380.log
-- 在切换到132机器上查看当前集群节点信息，131:6380已经升为主节点
redis-cli cluster nodes

-- 在重新启动129.6379服务
redis-server ./cluster/redis_6379.conf
-- 查看129.6379的节点信息，主节点变为从节点
redis-cli -p 6379 info replication
-- 观察131.6380日志，129.6379 重新加入集群
```