Redis集群

本章是基于CentOS7下的Redis集群教程，包括：

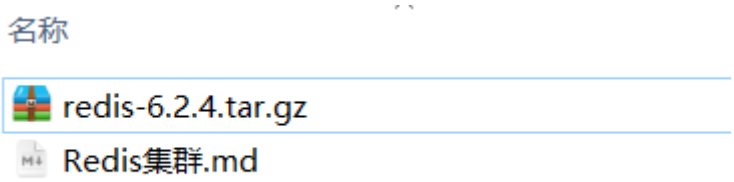
- 单机安装Redis
- Redis主从
- Redis分片集群

1.单机安装Redis

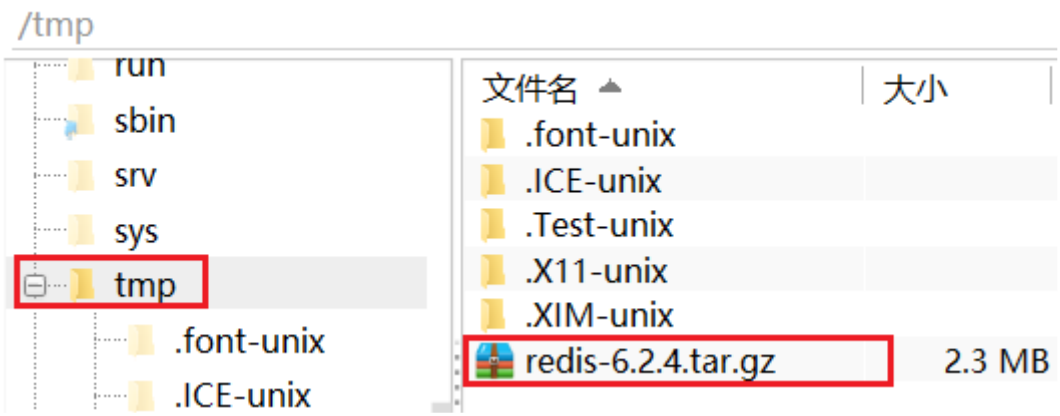
首先需要安装Redis所需要的依赖：

```
yum install -y gcc tcl
```

然后将课前资料提供的Redis安装包上传到虚拟机的任意目录：



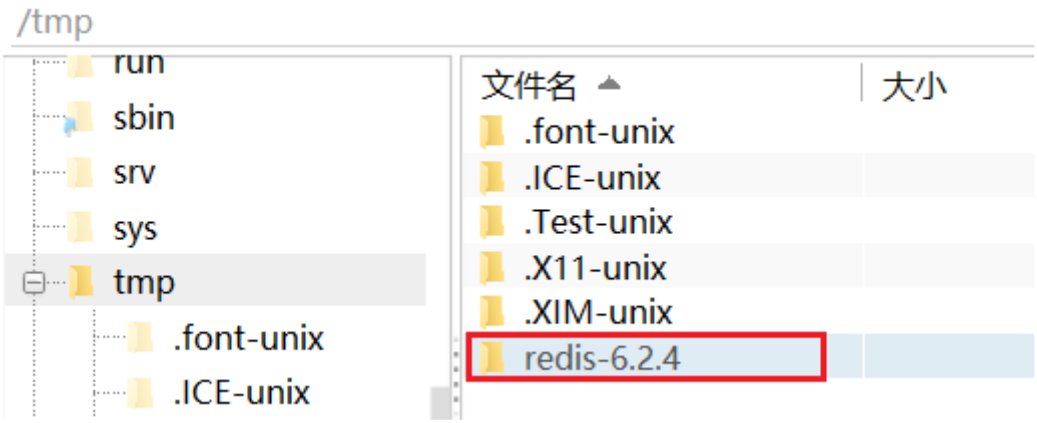
例如，我放到了/tmp目录：



解压缩：

```
tar -xvf redis-6.2.4.tar.gz
```

解压后：



进入redis目录：

```
cd redis-6.2.4
```

运行编译命令：

```
make && make install
```

如果没有出错，应该就安装成功了。

然后修改redis.conf文件中的一些配置：

```
# 绑定地址，默认是127.0.0.1，会导致只能在本地访问。修改为0.0.0.0则可以在任意IP访问
bind 0.0.0.0
# 数据库数量，设置为1
databases 1
```

启动Redis：

```
redis-server redis.conf
```

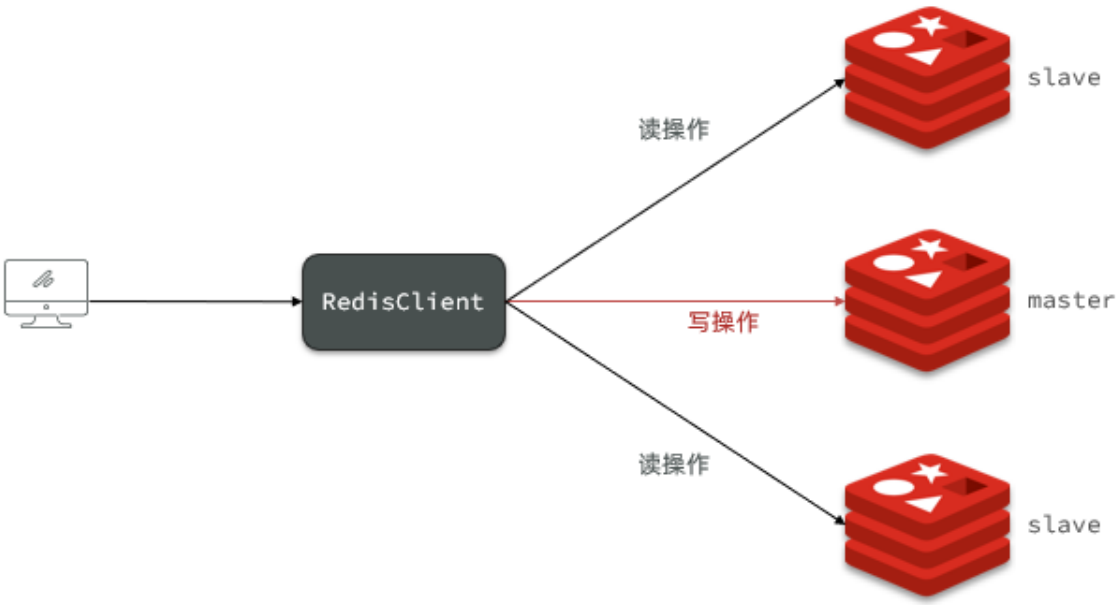
停止redis服务：

```
redis-cli shutdown
```

2.Redis主从集群

2.1.集群结构

我们搭建的主从集群结构如图：



共包含三个节点，一个主节点，两个从节点。

这里我们会在同一台虚拟机中开启3个redis实例，模拟主从集群，信息如下：

IP	PORT	角色
192.168.150.101	7001	master
192.168.150.101	7002	slave
192.168.150.101	7003	slave

2.2.准备实例和配置

要在同一台虚拟机开启3个实例，必须准备三份不同的配置文件和目录，配置文件所在目录也就是工作目录。

1）创建目录

我们创建三个文件夹，名字分别叫7001、7002、7003：

```
# 进入/tmp目录
cd /tmp
# 创建目录
mkdir 7001 7002 7003
```

如图：

```
[root@localhost tmp]# pwd
/tmp
[root@localhost tmp]# ll
总用量 4
drwxr-xr-x. 2 root root    6 6月   30 03:25 7001
drwxr-xr-x. 2 root root    6 6月   30 03:25 7002
drwxr-xr-x. 2 root root    6 6月   30 03:25 7003
drwxrwxr-x. 7 root root 4096 6月   30 03:25 redis-6.2.4
```

2）恢复原始配置

修改redis-6.2.4/redis.conf文件，将其中的持久化模式改为默认的RDB模式，AOF保持关闭状态。

```
# 开启RDB
# save ""
save 3600 1
save 300 100
save 60 10000

# 关闭AOF
appendonly no
```

3）拷贝配置文件到每个实例目录

然后将redis-6.2.4/redis.conf文件拷贝到三个目录中（在/tmp目录执行下列命令）：

```
# 方式一：逐个拷贝
cp redis-6.2.4/redis.conf 7001
cp redis-6.2.4/redis.conf 7002
cp redis-6.2.4/redis.conf 7003
# 方式二：管道组合命令，一键拷贝
echo 7001 7002 7003 | xargs -t -n 1 cp redis-6.2.4/redis.conf
```

4）修改每个实例的端口、工作目录

修改每个文件夹内的配置文件，将端口分别修改为7001、7002、7003，将rdb文件保存位置都修改为自己所在目录（在/tmp目录执行下列命令）：

```
sed -i -e 's/6379/7001/g' -e 's/dir .\//dir \/tmp\/7001\//g' 7001/redis.conf
sed -i -e 's/6379/7002/g' -e 's/dir .\//dir \/tmp\/7002\//g' 7002/redis.conf
sed -i -e 's/6379/7003/g' -e 's/dir .\//dir \/tmp\/7003\//g' 7003/redis.conf
```

5）修改每个实例的声明IP

虚拟机本身有多个IP，为了避免将来混乱，我们需要在redis.conf文件中指定每一个实例的绑定ip信息，格式如下：

```
# redis实例的声明 IP
replica-announce-ip 192.168.150.101
```

每个目录都要改，我们一键完成修改（在/tmp目录执行下列命令）：

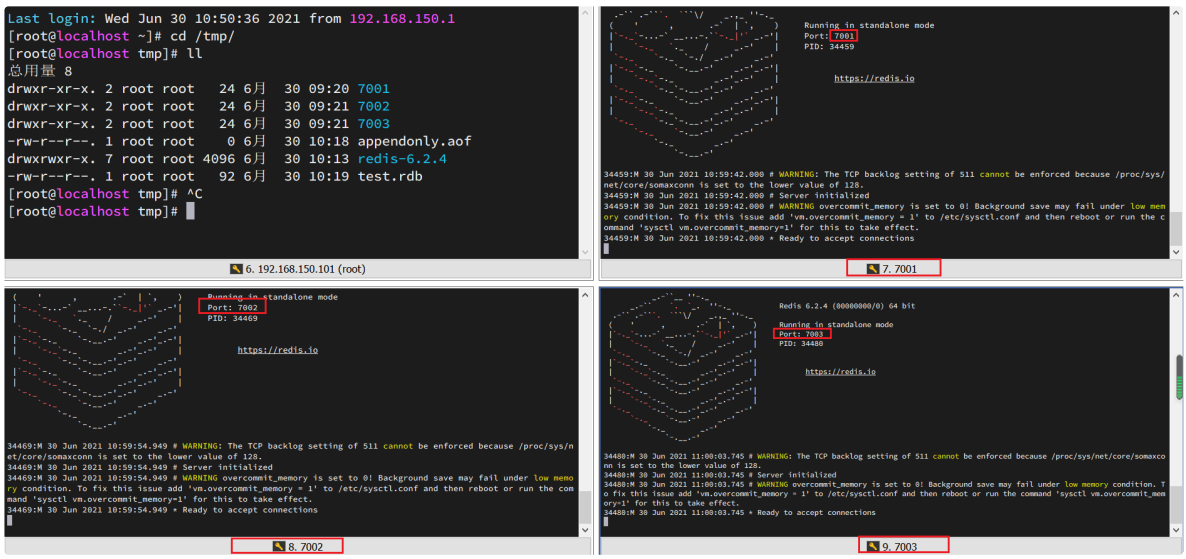
```
# 逐一执行
sed -i '1a replica-announce-ip 192.168.150.101' 7001/redis.conf
sed -i '1a replica-announce-ip 192.168.150.101' 7002/redis.conf
sed -i '1a replica-announce-ip 192.168.150.101' 7003/redis.conf

# 或者一键修改
printf '%s\n' 7001 7002 7003 | xargs -I{} -t sed -i '1a replica-announce-ip 192.168.150.101' {}/redis.conf
```

2.3.启动

为了方便查看日志，我们打开3个ssh窗口，分别启动3个redis实例，启动命令：

```
# 第1个
redis-server 7001/redis.conf
# 第2个
redis-server 7002/redis.conf
# 第3个
redis-server 7003/redis.conf
```

启动后：



如果要一键停止，可以运行下面命令：

```
printf '%s\n' 7001 7002 7003 | xargs -I{} -t redis-cli -p {} shutdown
```

2.4.开启主从关系

现在三个实例还没有任何关系，要配置主从可以使用replicaof 或者slaveof（5.0以前）命令。

有临时和永久两种模式：

- 修改配置文件（永久生效）
  - 在redis.conf中添加一行配置：slaveof <masterip> <masterport>
- 使用redis-cli客户端连接到redis服务，执行slaveof命令（重启后失效）：

```
slaveof <masterip> <masterport>
```

注意：在5.0以后新增命令replicaof，与salveof效果一致。

这里我们为了演示方便，使用方式二。

通过redis-cli命令连接7002，执行下面命令：

```
# 连接 7002
redis-cli -p 7002
# 执行slaveof
slaveof 192.168.150.101 7001
```

通过redis-cli命令连接7003，执行下面命令：

```
# 连接 7003
redis-cli -p 7003
# 执行slaveof
slaveof 192.168.150.101 7001
```

然后连接 7001节点，查看集群状态：

```
# 连接 7001
redis-cli -p 7001
# 查看状态
info replication
```

结果：

```
127.0.0.1:7001> INFO replication
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.150.101,port=7002,state=online,offset=322,lag=1
slave1:ip=192.168.150.101,port=7003,state=online,offset=322,lag=0
master_failover_state:no-failover
master_replid:f1c9374c417136bc462fd61a25ebef77bb7a41b7
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:336
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:336
```

2.5.测试

执行下列操作以测试：

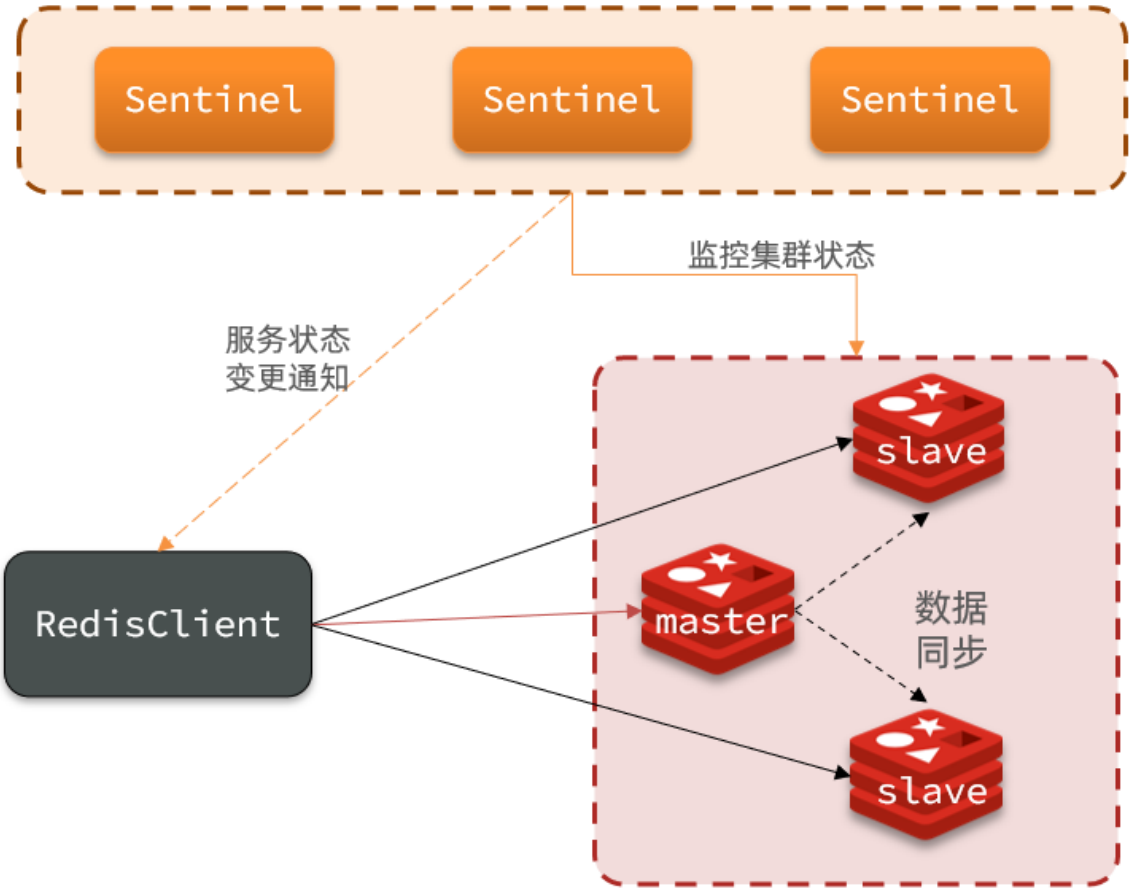
- 利用redis-cli连接7001，执行 `set num 123`
- 利用redis-cli连接7002，执行 `get num` ，再执行 `set num 666`
- 利用redis-cli连接7003，执行 `get num` ，再执行 `set num 888`

可以发现，只有在7001这个master节点上可以执行写操作，7002和7003这两个slave节点只能执行读操作。

3.搭建哨兵集群

3.1.集群结构

这里我们搭建一个三节点形成的Sentinel集群，来监管之前的Redis主从集群。如图：



三个sentinel实例信息如下：

节点	IP	PORT
s1	192.168.150.101	27001
s2	192.168.150.101	27002
s3	192.168.150.101	27003

3.2.准备实例和配置

要在同一台虚拟机开启3个实例，必须准备三份不同的配置文件和目录，配置文件所在目录也就是工作目录。

我们创建三个文件夹，名字分别叫s1、s2、s3：

```
# 进入/tmp目录
cd /tmp
# 创建目录
mkdir s1 s2 s3
```

如图：

```
[root@localhost tmp]# ll
总用量 8
drwxr-xr-x. 2 root root   40 7月    1 07:11 7001
drwxr-xr-x. 2 root root   40 7月    1 07:11 7002
drwxr-xr-x. 2 root root   40 7月    1 07:11 7003
-rw-r--r--. 1 root root  114 7月    1 07:21 a.txt
drwxrwxr-x. 7 root root 4096 7月    1 07:20 redis-6.2.4
drwxr-xr-x. 2 root root   27 7月    1 07:14 s1
drwxr-xr-x. 2 root root   27 7月    1 07:14 s2
drwxr-xr-x. 2 root root   27 7月    1 07:14 s3
```

然后我们在s1目录创建一个sentinel.conf文件，添加下面的内容：

```
port 27001
sentinel announce-ip 192.168.150.101
sentinel monitor mymaster 192.168.150.101 7001 2
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout mymaster 60000
dir "/tmp/s1"
```

解读：

- `port 27001`：是当前sentinel实例的端口
- `sentinel monitor mymaster 192.168.150.101 7001 2`：指定主节点信息
  - `mymaster`：主节点名称，自定义，任意写
  - `192.168.150.101 7001`：主节点的ip和端口
  - `2`：选举master时的quorum值

然后将s1/sentinel.conf文件拷贝到s2、s3两个目录中（在/tmp目录执行下列命令）：

```
# 方式一：逐个拷贝
cp s1/sentinel.conf s2
cp s1/sentinel.conf s3
# 方式二：管道组合命令，一键拷贝
echo s2 s3 | xargs -t -n 1 cp s1/sentinel.conf
```

修改s2、s3两个文件夹内的配置文件，将端口分别修改为27002、27003：

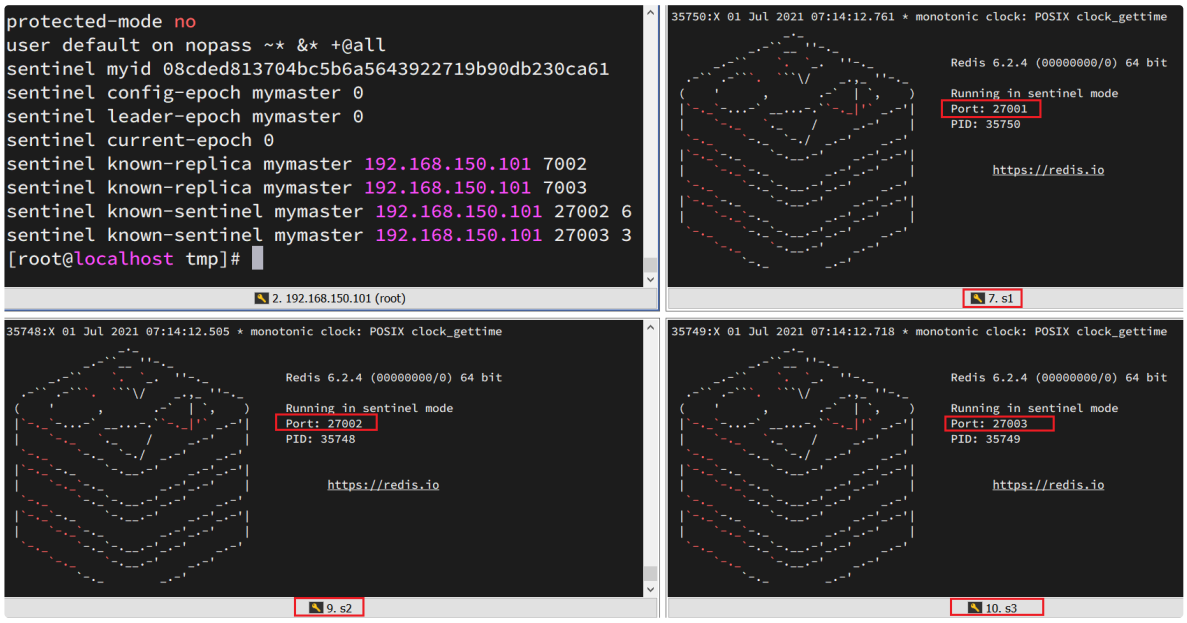
```
sed -i -e 's/27001/27002/g' -e 's/s1/s2/g' s2/sentinel.conf
sed -i -e 's/27001/27003/g' -e 's/s1/s3/g' s3/sentinel.conf
```

3.3.启动

为了方便查看日志，我们打开3个ssh窗口，分别启动3个redis实例，启动命令：

```
# 第1个
redis-sentinel s1/sentinel.conf
# 第2个
redis-sentinel s2/sentinel.conf
# 第3个
redis-sentinel s3/sentinel.conf
```

启动后：



3.4.测试

尝试让master节点7001宕机，查看sentinel日志：

```
^[35750:X 01 Jul 2021 07:50:00.855 # +sdown master mymaster 192.168.150.101 7001                主观认为7001下线
35750:X 01 Jul 2021 07:50:00.962 # +odown master mymaster 192.168.150.101 7001 #quorum 2/2          quorum达标，客观认为7001下线，实锤
35750:X 01 Jul 2021 07:50:00.962 # +new-epoch 1
35750:X 01 Jul 2021 07:50:00.962 # +try-failover master mymaster 192.168.150.101 7001 尝试等待7001
35750:X 01 Jul 2021 07:50:00.965 # +vote-for-leader 08cded813704bc5b6a5643922719b90db230ca
61 1                                  sentinel内部选一个leader，选中的sentinel实例去执行故障切换
35750:X 01 Jul 2021 07:50:00.972 # 6e1af76882ce5a09282256994ed6f68ced0a2bfc voted for 08cd
ed813704bc5b6a5643922719b90db230ca61 1
35750:X 01 Jul 2021 07:50:00.973 # 37dcedcaf7a8df807c3918b0b348484c0e6bd922 voted for 08cd
ed813704bc5b6a5643922719b90db230ca61 1
35750:X 01 Jul 2021 07:50:01.024 # +elected-leader master mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:01.024 # +failover-state-select-slave master mymaster 192.168.15 准备选一个slave，作为新master
0.101 7001
35750:X 01 Jul 2021 07:50:01.094 # +selected-slave slave 192.168.150.101:7003 192.168.150.  选中了7003这个实例
101 7003 @ mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:01.094 * +failover-state-send-slaveof-noone slave 192.168.150.10 让7003执行slaveof noone，成为新的master了
1:7003 192.168.150.101 7003 @ mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:01.167 * +failover-state-wait-promotion slave 192.168.150.101:70  7003等待提升，其实就是让其它slave执行slaveof 192.168.150.101 7003
03 192.168.150.101 7003 @ mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:01.998 # +promoted-slave slave 192.168.150.101:7003 192.168.150.  7003正式提升为master
101 7003 @ mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:01.998 # +failover-state-reconf-slaves master mymaster 192.168.1 修改下线的7001实例的配置，让它标记为7003的slave
50.101 7001
35750:X 01 Jul 2021 07:50:02.077 * +slave-reconf-sent slave 192.168.150.101:7002 192.168.1
50.101 7002 @ mymaster 192.168.150.101 7001                                  修改7002实例的配置，标记为7003的slave节点
35750:X 01 Jul 2021 07:50:03.000 * +slave-reconf-inprog slave 192.168.150.101:7002 192.168
.150.101 7002 @ mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:03.000 * +slave-reconf-done slave 192.168.150.101:7002 192.168.1
50.101 7002 @ mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:03.081 # -odown master mymaster 192.168.150.101 7001          事件处理结束
35750:X 01 Jul 2021 07:50:03.081 # +failover-end master mymaster 192.168.150.101 7001
35750:X 01 Jul 2021 07:50:03.081 # +switch-master mymaster 192.168.150.101 7001 192.168.15   主切换完成
0.101 7003
```

查看7003的日志：

```
35726:S 01 Jul 2021 07:50:00.745 # Error condition on socket for SYNC: Conn
ection refused
35726:M 01 Jul 2021 07:50:01.167 * Discarding previously cached master stat
e.
35726:M 01 Jul 2021 07:50:01.167 # Setting secondary replication ID to 4266
806f42cd665e4abb7bddfeb341c5bf3d751a, valid up to offset: 459681. New repli
cation ID is 781702d350fc5b86f8d926404b341cb8c60c3122
35726:M 01 Jul 2021 07:50:01.167 * MASTER MODE enabled (user request from '
id=9 addr=192.168.150.101:59416 laddr=192.168.150.101:7003 fd=12 name=senti
nel-08cded81-cmd age=2149 idle=0 flags=x db=0 sub=0 psub=0 multi=4 qbuf=202
 qbuf-free=40752 argv-mem=4 obl=45 oll=0 omem=0 tot-mem=61468 events=r cmd=
exec user=default redir=-1')
35726:M 01 Jul 2021 07:50:01.170 # CONFIG REWRITE executed with success.
35726:M 01 Jul 2021 07:50:02.081 * Replica 192.168.150.101:7002 asks for sy
nchronization
35726:M 01 Jul 2021 07:50:02.081 * Partial resynchronization request from 1
92.168.150.101:7002 accepted. Sending 313 bytes of backlog starting from of
fset 459681.
```

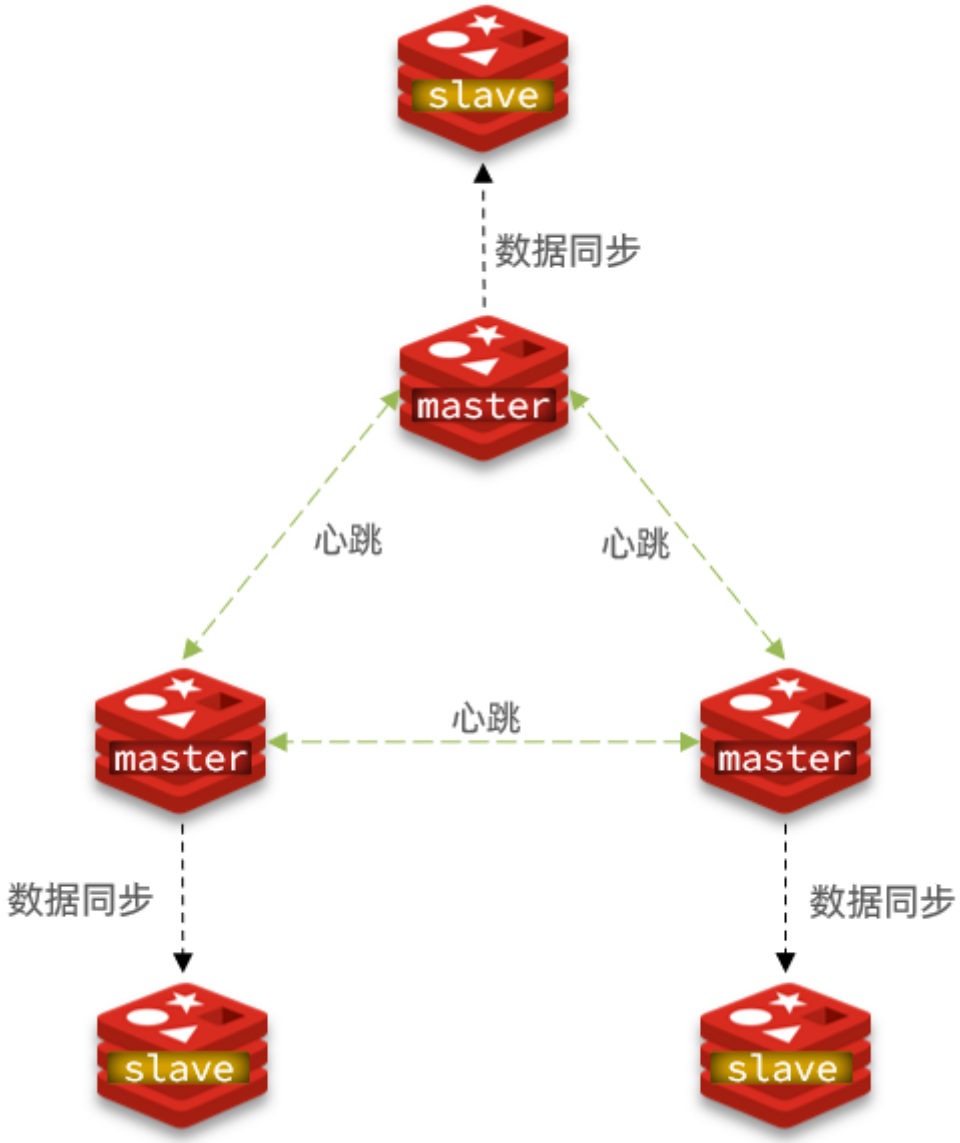
查看7002的日志：

```
35727:S 01 Jul 2021 07:50:01.598 # Error condition on socket for SYNC: Connection refused
35727:S 01 Jul 2021 07:50:02.078 * Connecting to MASTER 192.168.150.101:7003
35727:S 01 Jul 2021 07:50:02.078 * MASTER <-> REPLICA sync started
35727:S 01 Jul 2021 07:50:02.078 * REPLICAOF 192.168.150.101:7003 enabled (user request fr
om 'id=9 addr=192.168.150.101:48788 laddr=192.168.150.101:7002 fd=12 name=sentinel-08cded8
1-cmd age=2150 idle=0 flags=x db=0 sub=0 psub=0 multi=4 qbuf=348 qbuf-free=40606 argv-mem=
4 obl=45 oll=0 omem=0 tot-mem=61468 events=r cmd=exec user=default redir=-1')
35727:S 01 Jul 2021 07:50:02.081 # CONFIG REWRITE executed with success.
35727:S 01 Jul 2021 07:50:02.081 * Non blocking connect for SYNC fired the event.
35727:S 01 Jul 2021 07:50:02.081 * Master replied to PING, replication can continue...
35727:S 01 Jul 2021 07:50:02.081 * Trying a partial resynchronization (request 4266806f42c
d665e4abb7bddfeb341c5bf3d751a:459681).                          重新执行psync
35727:S 01 Jul 2021 07:50:02.081 * Successful partial resynchronization with master.
35727:S 01 Jul 2021 07:50:02.081 # Master replication ID changed to 781702d350fc5b86f8d926
404b341cb8c60c3122
35727:S 01 Jul 2021 07:50:02.081 * MASTER <-> REPLICA sync: Master accepted a Partial Resy
nchronization.
```

4.搭建分片集群

4.1.集群结构

分片集群需要的节点数量较多，这里我们搭建一个最小的分片集群，包含3个master节点，每个master包含一个slave节点，结构如下：



这里我们会在同一台虚拟机中开启6个redis实例，模拟分片集群，信息如下：

IP	PORT	角色
192.168.150.101	7001	master
192.168.150.101	7002	master
192.168.150.101	7003	master
192.168.150.101	8001	slave
192.168.150.101	8002	slave
192.168.150.101	8003	slave

4.2.准备实例和配置

删除之前的7001、7002、7003这几个目录，重新创建出7001、7002、7003、8001、8002、8003目录：

```
# 进入/tmp目录
cd /tmp
# 删除旧的，避免配置干扰
rm -rf 7001 7002 7003
# 创建目录
mkdir 7001 7002 7003 8001 8002 8003
```

在/tmp下准备一个新的redis.conf文件，内容如下：

```
port 6379
# 开启集群功能
cluster-enabled yes
# 集群的配置文件名称，不需要我们创建，由redis自己维护
cluster-config-file /tmp/6379/nodes.conf
# 节点心跳失败的超时时间
cluster-node-timeout 5000
# 持久化文件存放目录
dir /tmp/6379
# 绑定地址
bind 0.0.0.0
# 让redis后台运行
daemonize yes
# 注册的实例ip
replica-announce-ip 192.168.150.101
# 保护模式
protected-mode no
# 数据库数量
databases 1
# 日志
logfile /tmp/6379/run.log
```

将这个文件拷贝到每个目录下：

```
# 进入/tmp目录
cd /tmp
# 执行拷贝
echo 7001 7002 7003 8001 8002 8003 | xargs -t -n 1 cp redis.conf
```

修改每个目录下的redis.conf，将其中的6379修改为与所在目录一致：

```
# 进入/tmp目录
cd /tmp
# 修改配置文件
printf '%s\n' 7001 7002 7003 8001 8002 8003 | xargs -I{} -t sed -i
's/6379/{}/g' {}/redis.conf
```

4.3.启动

因为已经配置了后台启动模式，所以可以直接启动服务：

```
# 进入/tmp目录
cd /tmp
# 一键启动所有服务
printf '%s\n' 7001 7002 7003 8001 8002 8003 | xargs -I{} -t redis-server
{}/redis.conf
```

通过ps查看状态：

```
ps -ef | grep redis
```

发现服务都已经正常启动：

```
[root@localhost tmp]# ps -ef | grep redis
root       2362      1  0 09:21 ?        00:00:00 redis-server 0.0.0.0:7001 [cluster]
root       2368      1  0 09:21 ?        00:00:00 redis-server 0.0.0.0:7002 [cluster]
root       2374      1  0 09:21 ?        00:00:00 redis-server 0.0.0.0:7003 [cluster]
root       2380      1  0 09:21 ?        00:00:00 redis-server 0.0.0.0:8001 [cluster]
root       2386      1  0 09:21 ?        00:00:00 redis-server 0.0.0.0:8002 [cluster]
root       2392      1  0 09:21 ?        00:00:00 redis-server 0.0.0.0:8003 [cluster]
```

如果要关闭所有进程，可以执行命令：

```
ps -ef | grep redis | awk '{print $2}' | xargs kill
```

或者（推荐这种方式）：

```
printf '%s\n' 7001 7002 7003 8001 8002 8003 | xargs -I{} -t redis-cli -p {}
shutdown
```

4.4.创建集群

虽然服务启动了，但是目前每个服务之间都是独立的，没有任何关联。

我们需要执行命令来创建集群，在Redis5.0之前创建集群比较麻烦，5.0之后集群管理命令都集成到了redis-cli中。

1）Redis5.0之前

Redis5.0之前集群命令都是用redis安装包下的src/redis-trib.rb来实现的。因为redis-trib.rb是有ruby语言编写的所以需要安装ruby环境。

```
# 安装依赖
yum -y install zlib ruby rubygems
gem install redis
```

然后通过命令来管理集群：

```
# 进入redis的src目录
cd /tmp/redis-6.2.4/src
# 创建集群
./redis-trib.rb create --replicas 1 192.168.150.101:7001
192.168.150.101:7002 192.168.150.101:7003 192.168.150.101:8001
192.168.150.101:8002 192.168.150.101:8003
```

2）Redis5.0以后

我们使用的是Redis6.2.4版本，集群管理以及集成到了redis-cli中，格式如下：

```
redis-cli --cluster create --cluster-replicas 1 192.168.150.101:7001
192.168.150.101:7002 192.168.150.101:7003 192.168.150.101:8001
192.168.150.101:8002 192.168.150.101:8003
```

命令说明：

- `redis-cli --cluster` 或者 `./redis-trib.rb`：代表集群操作命令
- `create`：代表是创建集群
- `--replicas 1` 或者 `--cluster-replicas 1` ：指定集群中每个master的副本个数为1，此时 `节点总数 ÷ (replicas + 1)` 得到的就是master的数量。因此节点列表中的前n个就是master，其它节点都是slave节点，随机分配到不同master

运行后的样子：

```
[root@localhost tmp]# redis-cli --cluster create --cluster-replicas 1 192.168.150.101:7001 192.168.150.101:7002
168.150.101:7003 192.168.150.101:8001 192.168.150.101:8002 192.168.150.101:8003
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 192.168.150.101:8002 to 192.168.150.101:7001
Adding replica 192.168.150.101:8003 to 192.168.150.101:7002
Adding replica 192.168.150.101:8001 to 192.168.150.101:7003
>>> Trying to optimize slaves allocation for anti-affinity
[WARNING] Some slaves are in the same host as their master
M: f5fc58defbebb957e47fb0d8327a09dc4f1678f5 192.168.150.101:7001
   slots:[0-5460] (5461 slots) master
M: afaaa70d6528fc72490e0f3f7b32731a12c12bb8 192.168.150.101:7002
   slots:[5461-10922] (5462 slots) master
M: 1c00e5f9e158b169f199f15884ab43bc433b1a06 192.168.150.101:7003
   slots:[10923-16383] (5461 slots) master
S: 7b6d5ffc9a985d614dc5aeb2ee3abac1adfd3e22 192.168.150.101:8001
   replicates afaaa70d6528fc72490e0f3f7b32731a12c12bb8
S: 6ec60fb5afd950a465f05c8024bf8f75d809b014 192.168.150.101:8002
   replicates 1c00e5f9e158b169f199f15884ab43bc433b1a06
S: 1fa6d68d590827c24c237b1c490b78e5c7fe2ca9 192.168.150.101:8003
   replicates f5fc58defbebb957e47fb0d8327a09dc4f1678f5
Can I set the above configuration? (type 'yes' to accept):
```

询问是否要按照上面列出的方式来创建集群？如果没问题，输入yes

这里输入yes，则集群开始创建：

```
>>> 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
.
>>> Performing Cluster Check (using node 192.168.150.101:7001)
M: f5fc58defbebb957e47fb0d8327a09dc4f1678f5 192.168.150.101:7001
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
M: afaaa70d6528fc72490e0f3f7b32731a12c12bb8 192.168.150.101:7002
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
S: 1fa6d68d590827c24c237b1c490b78e5c7fe2ca9 192.168.150.101:8003
   slots: (0 slots) slave
   replicates f5fc58defbebb957e47fb0d8327a09dc4f1678f5
S: 6ec60fb5afd950a465f05c8024bf8f75d809b014 192.168.150.101:8002
   slots: (0 slots) slave
   replicates 1c00e5f9e158b169f199f15884ab43bc433b1a06
S: 7b6d5ffc9a985d614dc5aeb2ee3abac1adfd3e22 192.168.150.101:8001
   slots: (0 slots) slave
   replicates afaaa70d6528fc72490e0f3f7b32731a12c12bb8
M: 1c00e5f9e158b169f199f15884ab43bc433b1a06 192.168.150.101:7003
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

通过命令可以查看集群状态：

```
1 redis-cli -p 7001 cluster nodes
```

```
[root@localhost tmp]# redis-cli -p 7001 cluster nodes
afaaa70d6528fc72490e0f3f7b32731a12c12bb8 192.168.150.101:7002@17002 master - 0 1625191451299 2 connected 5461-10922
f5fc58defbebb957e47fb0d8327a09dc4f1678f5 192.168.150.101:7001@17001 myself,master - 0 1625191450000 1 connected 0-5460
1fa6d68d590827c24c237b1c490b78e5c7fe2ca9 192.168.150.101:8003@18003 slave f5fc58defbebb957e47fb0d8327a09dc4f1678f5 0 1625191450245 1 connected
6ec60fb5afd950a465f05c8024bf8f75d809b014 192.168.150.101:8002@18002 slave 1c00e5f9e158b169f199f15884ab43bc433b1a06 0 1625191451084 3 connected
7b6d5ffc9a985d614dc5aeb2ee3abac1adfd3e22 192.168.150.101:8001@18001 slave afaaa70d6528fc72490e0f3f7b32731a12c12bb8 0 1625191452000 2 connected
1c00e5f9e158b169f199f15884ab43bc433b1a06 192.168.150.101:7003@17003 master - 0 1625191452351 3 connected 10923-16383
```

4.5.测试

尝试连接7001节点，存储一个数据：

```
# 连接
redis-cli -p 7001
# 存储数据
set num 123
# 读取数据
get num
# 再次存储
set a 1
```

结果悲剧了：

```
[root@localhost tmp]# redis-cli -p 7001
127.0.0.1:7001> set num 123
OK
127.0.0.1:7001> get num
"123"
127.0.0.1:7001> set a 1
(error) MOVED 15495 192.168.150.101:7003
127.0.0.1:7001>
```

集群操作时，需要给 `redis-cli` 加上 `-c` 参数才可以：

```
redis-cli -c -p 7001
```

这次可以了：

```
[root@localhost tmp]# redis-cli -c -p 7001
127.0.0.1:7001>
127.0.0.1:7001>
127.0.0.1:7001> get num
"123"
127.0.0.1:7001> set a 1
-> Redirected to slot [15495] located at 192.168.150.101:7003
OK
192.168.150.101:7003> get a
"1"
192.168.150.101:7003>
```