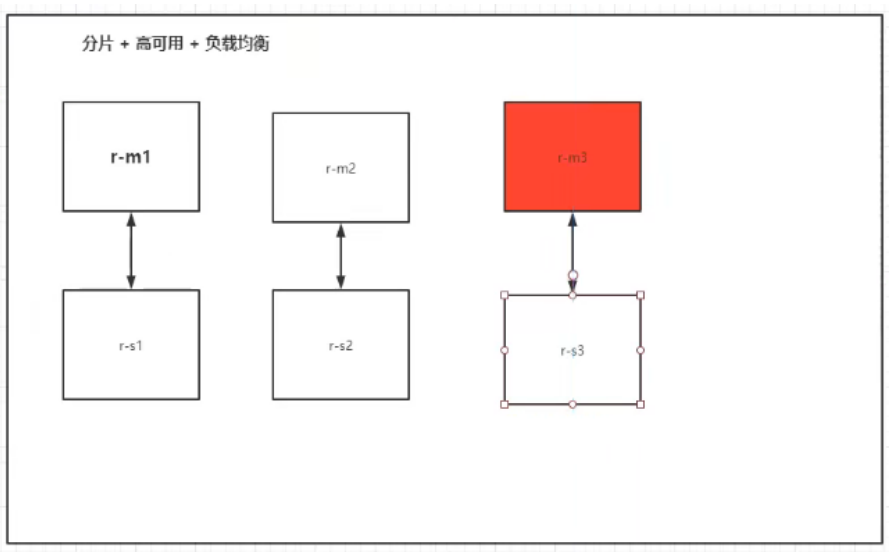
19、使用Docker搭建高可用Redis集群

- 参考：https://zhuanlan.zhihu.com/p/144479894
- 在Redis的服务中，可以有多台服务器，还可以配置主从服务器，通过配置使得从机能够从主机同步数据。在这种配置下，当主Redis服务器出现故障时，只需要执行故障切换（failover）即可，也就是作废当前出故障的主Redis服务器，将从Redis服务器切换为主Redis服务器即可。这个过程可以由人工完成，也可以由程序完成，如果由人工完成，则需要增加人力成本，且容易产生人工错误，还会造成一段时间的程序不可用，所以一般来说，我们会选择使用程序完成。这个程序就是我们所说的哨兵（sentinel），哨兵是一个程序进程，它运行于系统中，通过发送命令去检测各个Redis服务器（包括主从Redis服务器）
- 除了可以使用哨兵模式实现高可用的集群外，还可以使用Redis集群（cluster）技术来实现高可用，不过Redis集群是3.0版本之后才提供的，所以在使用集群前，请注意你的Redis版本。
- 这里我们学习使用Redis集群（cluster）技术来实现高可用。

1、redis集群架构



2、创建一个网络，叫redis（网络模式driver使用default，也就是默认的bridge桥接模式，分配的子网范围：172.38.0.1~172.38.255.255）

```
[root@kuangshen /]# docker network create redis --subnet 172.38.0.0/16
8a16fe0d88708e32490a1496cbc420afe0bd6a865fbee06e603626e3536f8d74
[root@kuangshen /]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
5a008c015cac        bridge              bridge              local
db44649a9bff        composetest_default bridge              local
ae2b6209c2ab        host                host                local
eb21272b3a35        mynet               bridge              local
c037f7ec7e57        none                null                local
8a16fe0d8870        redis               bridge              local
[root@kuangshen /]# docker network inspect redis
[
    {
        "Name": "redis",
        "Id": "8a16fe0d88708e32490a1496cbc420afe0bd6a865fbee06e603626e353
        "Created": "2020-05-15T23:38:28.514181843+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.38.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
[root@kuangshen /]#
```

3、使用shell脚本，来创建6个redis配置文件（3主3从）

```
# 通过脚本创建六个redis配置
for port in $(seq 1 6); \
do \
mkdir -p /mydata/redis/node-${port}/conf
touch /mydata/redis/node-${port}/conf/redis.conf
cat << EOF >/mydata/redis/node-${port}/conf/redis.conf
port 6379
bind 0.0.0.0
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
cluster-announce-ip 172.38.0.1${port}
cluster-announce-port 6379
cluster-announce-bus-port 16379
appendonly yes
EOF
done
```

```
[root@kuangshen /]# for port in $(seq 1 6); \
> do \
> mkdir -p /mydata/redis/node-${port}/conf
> touch /mydata/redis/node-${port}/conf/redis.conf
> cat << EOF >/mydata/redis/node-${port}/conf/redis.conf
> port 6379
> bind 0.0.0.0
> cluster-enabled yes
> cluster-config-file nodes.conf
> cluster-node-timeout 5000
> cluster-announce-ip 172.38.0.1${port}
> cluster-announce-port 6379
> cluster-announce-bus-port 16379
> appendonly yes
> EOF
> done
[root@kuangshen /]#
```

4、查看下使用shell脚本创建好的6个redis配置文件

```
[root@kuangshen /]# cd /mydata/
[root@kuangshen mydata]# ls
redis
[root@kuangshen mydata]# cd redis/
[root@kuangshen redis]# ls
node-1  node-2  node-3  node-4  node-5  node-6
[root@kuangshen redis]#
```

5、基于redis:5.0.9-alpine3.11镜像，运行出redis-1、redis-2.....redis-6等6个容器（6个redis服务节点）

- 记得使用redis网络运行出容器，并且要使用redis网络下的没有被占用的子网ip
- 每个容器，要对外暴露的端口要不同，容器内的端口都是一致的6379
- 每个容器，要挂载目录也是不同的。

```
[root@kuangshen conf]# docker run -p 6371:6379 -p 16371:16379 --name redis-1 \
> -v /mydata/redis/node-1/data:/data \
> -v /mydata/redis/node-1/conf/redis.conf:/etc/redis/redis.conf \
> -d --net redis --ip 172.38.0.11 redis:5.0.9-alpine3.11 redis-server /etc/redis/redis.conf
Unable to find image 'redis:5.0.9-alpine3.11' locally
5.0.9-alpine3.11: Pulling from library/redis
cbdbe7a5bc2a: Pull complete
dc0373118a0d: Pull complete
cfd369fe6256: Pull complete
3e45770272d9: Pull complete
558de8ea3153: Pull complete
a2c652551612: Pull complete
Digest: sha256:83a3af36d5e57f2901b4783c313720e5fa3ecf0424ba86ad9775e06a9a5e35d0
Status: Downloaded newer image for redis:5.0.9-alpine3.11
09f75ccd982b2f50c7c2a466cf69b571510f1ded688c70390c63705cff2567be
[root@kuangshen conf]# docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS
                                NAMES
09f75ccd982b        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   4 seconds ago       Up 3 seconds
6379/tcp, 0.0.0.0:16371->16379/tcp   redis-1
[root@kuangshen conf]# docker run -p 6372:6379 -p 16372:16379 --name redis-2 \
> -v /mydata/redis/node-2/data:/data \
> -v /mydata/redis/node-2/conf/redis.conf:/etc/redis/redis.conf \
> -d --net redis --ip 172.38.0.12 redis:5.0.9-alpine3.11 redis-server /etc/redis/redis.conf
9950ab87ea24e1abb5484567159794d4a152e0c08df3dd0d3fbc85dbf458ca39
[root@kuangshen conf]#
```

```
[root@kuangshen conf]# docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                                NAMES
2b5e15892b29        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   4 seconds ago       Up 3 seconds        0.0.0.0:6376-
>6379/tcp, 0.0.0.0:16376->16379/tcp   redis-6
4d42286d4a2d        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   18 seconds ago      Up 17 seconds       0.0.0.0:6375-
>6379/tcp, 0.0.0.0:16375->16379/tcp   redis-5
9553e7f0568d        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   39 seconds ago      Up 38 seconds       0.0.0.0:6374-
>6379/tcp, 0.0.0.0:16374->16379/tcp   redis-4
b7517798a0cf        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   About a minute ago  Up About a minute   0.0.0.0:6373-
>6379/tcp, 0.0.0.0:16373->16379/tcp   redis-3
9950ab87ea24        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   About a minute ago  Up About a minute   0.0.0.0:6372-
>6379/tcp, 0.0.0.0:16372->16379/tcp   redis-2
09f75ccd982b        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   About a minute ago  Up About a minute   0.0.0.0:6371-
>6379/tcp, 0.0.0.0:16371->16379/tcp   redis-1
[root@kuangshen conf]#
```

6、进入某一个redis容器内部，为6个redis服务节点创建一个redis集群，确定主从关系。

- 主节点的hash槽之和就是总分片数据（16384个槽），只有主节点才提供写服务，当主节点宕机，对应的从节点才会自动替换为主节点，当然主节点数据必须是完整的，因为从节点的数据是同步的主节点的数据。

```
[root@kuangshen conf]# docker exec -it redis-1 /bin/bash
OCI runtime exec failed: exec failed: container_linux.go:349: starting container process caused "exec: \"/bin/bash\": stat /
bin/bash: no such file or directory": unknown
[root@kuangshen conf]# docker exec -it redis-1 /bin/sh
/data # ls
appendonly.aof  nodes.conf
/data #
```

进入redis容器内部，不能使用/bin/bash，而是/bin/sh

```
/data # redis-cli --cluster create 172.38.0.11:6379 172.38.0.12:6379 172.38.0.13:6379 172.38.0.14:6379 172.38.0.15:6379 172.38.0.16:6379 --cluster-replicas 1
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 172.38.0.15:6379 to 172.38.0.11:6379
Adding replica 172.38.0.16:6379 to 172.38.0.12:6379
Adding replica 172.38.0.14:6379 to 172.38.0.13:6379
M: 1f78274584697920a57d573a9e4940bb02470771 172.38.0.11:6379
   slots:[0-5460] (5461 slots) master
M: 86d22fa57e6f53ddd1969e96fd9dadff186312a3 172.38.0.12:6379
   slots:[5461-10922] (5462 slots) master
M: 070b04be2eb89c6131162a05fd5e7d77c6650455 172.38.0.13:6379
   slots:[10923-16383] (5461 slots) master
S: 2308bc39496252d51c6605e3a168fe00267d1915 172.38.0.14:6379
   replicates 070b04be2eb89c6131162a05fd5e7d77c6650455
S: 33dfa95e5b370392a31a8fbaaa694b67a2af3308 172.38.0.15:6379
   replicates 1f78274584697920a57d573a9e4940bb02470771
S: 4e0f40a1409df5400f4d6ffdb1cbc4277501be94 172.38.0.16:6379
   replicates 86d22fa57e6f53ddd1969e96fd9dadff186312a3
Can I set the above configuration? (type 'yes' to accept):
```

确定主节点，并为主分配hash槽，因为主才能写数据

确定每个主的从

7、连接到redis集群，查看下集群信息

-c代表连接到集群，不写是连接到单节点

查看下集群信息

```
/data # redis-cli -c
127.0.0.1:6379> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:6
cluster_my_epoch:1
cluster_stats_messages_ping_sent:84
cluster_stats_messages_pong_sent:88
cluster_stats_messages_sent:172
cluster_stats_messages_ping_received:83
cluster_stats_messages_pong_received:84
cluster_stats_messages_meet_received:5
cluster_stats_messages_received:172
127.0.0.1:6379>
```

8、查看下集群下的主从节点的信息

```
127.0.0.1:6379> cluster nodes
33dfa95e5b370392a31a8fbaaa694b67a2af3308 172.38.0.15:6379@16379 slave 1f78274584697920a57d573a9e4940bb02470771 0 15895575770
00 5 connected
4e0f40a1409df5400f4d6ffdb1cbc4277501be94 172.38.0.16:6379@16379 slave 86d22fa57e6f53ddd1969e96fd9dadff186312a3 0 15895575770
00 6 connected
1f78274584697920a57d573a9e4940bb02470771 172.38.0.11:6379@16379 myself,master - 0 1589557576000 1 connected 0-5460
86d22fa57e6f53ddd1969e96fd9dadff186312a3 172.38.0.12:6379@16379 master - 0 1589557577111 2 connected 5461-10922
070b04be2eb89c6131162a05fd5e7d77c6650455 172.38.0.13:6379@16379 master - 0 1589557577000 3 connected 10923-16383
2308bc39496252d51c6605e3a168fe00267d1915 172.38.0.14:6379@16379 slave 070b04be2eb89c6131162a05fd5e7d77c6650455 0 15895575781
14 4 connected
127.0.0.1:6379>
```

9、向集群里插入一个值，看到插入到了172.39.0.13（redis-3主节点）

```
127.0.0.1:6379> set a b
-> Redirected to slot [15495] located at 172.38.0.13:6379
OK
172.38.0.13:6379>
```

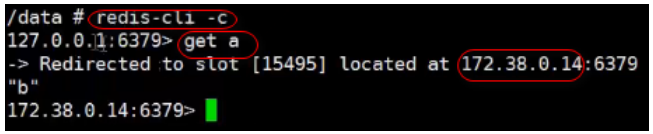
主节点

10、为了测试从节点**自动**替换为主节点（redis的哨兵机制），先停止redis-3主节点（172.39.0.13）

```
[root@kuangshen tomcatlogs]# docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                    NAMES
2b5e15892b29        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   4 minutes ago       Up 4 minutes        0.0.0.0:6376->
6379/tcp, 0.0.0.0:16376->16379/tcp   redis-6
4d42286d4a2d        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   4 minutes ago       Up 4 minutes        0.0.0.0:6375->
6379/tcp, 0.0.0.0:16375->16379/tcp   redis-5
9553e7f0568d        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   4 minutes ago       Up 4 minutes        0.0.0.0:6374->
6379/tcp, 0.0.0.0:16374->16379/tcp   redis-4
b7517798a0cf        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   5 minutes ago       Up 5 minutes        0.0.0.0:6373->
6379/tcp, 0.0.0.0:16373->16379/tcp   redis-3
9950ab87ea24        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   5 minutes ago       Up 5 minutes        0.0.0.0:6372->
6379/tcp, 0.0.0.0:16372->16379/tcp   redis-2
09f75ccd982b        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   6 minutes ago       Up 6 minutes        0.0.0.0:6371->
6379/tcp, 0.0.0.0:16371->16379/tcp   redis-1
[root@kuangshen tomcatlogs]# docker stop redis-3
redis-3
[root@kuangshen tomcatlogs]#
```

11、可以看到从节点172.39.0.14已经自动替换为主节点，对外提供服务，至此docker搭建高可用的redis集群成功。

```
172.38.0.14:6379> cluster nodes
070b04be2eb89c6131162a05fd5e7d77c6650455 172.38.0.13:6379@16379 master,fail - 1589557662128 1589557662028 3 connected
2308bc39496252d51c6605e3a168fe00267d1915 172.38.0.14:6379@16379 myself,master - 0 1589557738000 7 connected 10923-16383
1f78274584697920a57d573a9e4940bb02470771 172.38.0.11:6379@16379 master - 0 1589557738279 1 connected 0-5460
33dfa95e5b370392a31a8fbaaa694b67a2af3308 172.38.0.15:6379@16379 slave 1f78274584697920a57d573a9e4940bb02470771 0 158955773
00 5 connected
86d22fa57e6f53ddd1969e96fd9dadff186312a3 172.38.0.12:6379@16379 master - 0 1589557738000 2 connected 5461-10922
4e0f40a1409df5400f4d6ffdb1cbc4277501be94 172.38.0.16:6379@16379 slave 86d22fa57e6f53ddd1969e96fd9dadff186312a3 0 158955773
80 6 connected
172.38.0.14:6379>
```


/data # redis-cli -c
127.0.0.1:6379> get a
-> Redirected to slot [15495] located at 172.38.0.14:6379
"b"
172.38.0.14:6379>