

## Redis 集群



JAVA架...

“编程开发工作者，更像是一名艺术家”

5 人赞同了该文章

## Redis 集群

### 1.1 集群的概念

所谓的集群，就是通过添加服务器的数量，提供相同的服务，从而让服务器达到一个稳定、高效的状态（高可用）。

### 1.2 使用redis集群的必要性

问题：我们已经部署好了redis，并且能启动一个redis，实现数据的读写，为什么还要学习redis集群？

1. 单个redis存在不稳定性。当redis服务宕机了，就没有可用的服务了。
2. 单个redis的读写能力是有限的。

redis集群是为了强化redis的读写能力。

### 1.3 如何学习redis集群

1. redis集群中，每一个redis称之为一个节点。
2. redis集群中，有两种类型的节点：主节点(master)、从节点(slave)。
3. redis集群，是基于redis主从复制实现。

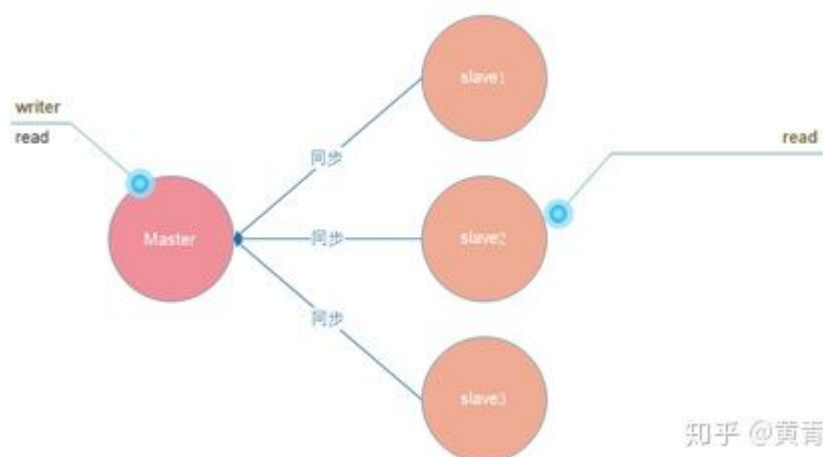
所以，学习redis集群，就是从学习redis主从复制模型开始的。

## 2 redis主从复制

### 2.1 概念



▲ 赞同 5



## 2.2 特点

1. 主节点Master可读、可写
2. 从节点Slave只读。（read-only）

因此，主从模型可以提高读的能力，在一定程度上缓解了写的能力。因为能写仍然只有Master节点一个，可以将读的操作全部移交到从节点上，变相提高了写能力。

## 2.3 基于配置实现

### 2.3.1 需求

节点	端口
主节点	6380
从节点（两个）	6381、6382

### 2.3.2 配置步骤

1. 在 /usr/local 目录下，创建一个 /redis/master-slave 目录
2. 在 master-slave 目录下，创建三个子目录 6380、6381、6382
3. 依次拷贝redis解压目录下的 redis.conf 配置文件，到这三个子目录中
4. 进入 6380 (master) 目录，修改 redis.conf，将 port 端口修改成 6380 即可
5. 进入 6381 (slave) 和 6382 目录，分别修改 redis.conf，将 port 端口改成 6381 和 6382



```

90 # Accept connections on the specified port, default is 6379 (IANA #815344)
91 # If port 0 is specified Redis will not listen on a TCP socket.
92 port 6381

```

```

284 # and resynchronize with them.
285 #
286 # replicaof <masterip> <masterport>
287 replicaof 127.0.0.1 6380

```

知乎 @黄青

### 2.3.3 测试

依次启动主从节点，主节点的日志中会显示从节点的连入。经测试可以看到，主节点可以读写，从节点默认只能读不能写。

```

15787:S 07 Aug 2019 14:24:29.526 # 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.
15787:S 07 Aug 2019 14:24:29.526 # 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 after THP is disabled.
15787:S 07 Aug 2019 14:24:29.526 * Ready to accept connections
15787:S 07 Aug 2019 14:24:29.526 * Connecting to MASTER 127.0.0.1:6380
15787:S 07 Aug 2019 14:24:29.527 * MASTER <-> REPLICATION sync started
15787:S 07 Aug 2019 14:24:29.527 * Non blocking connect for SYNC fired the event.
15787:S 07 Aug 2019 14:24:29.528 * Master replied to PING, replication can continue...
15787:S 07 Aug 2019 14:24:29.528 * Partial resynchronization not possible (no cached master)
15787:S 07 Aug 2019 14:24:29.571 * Full resync from master: ba76c6258e3b31bb1f8c95557dc9673e8512566618
15787:S 07 Aug 2019 14:24:29.673 * MASTER <-> REPLICATION sync: receiving 175 bytes from master
15787:S 07 Aug 2019 14:24:29.673 * MASTER <-> REPLICATION sync: Flushing old data
15787:S 07 Aug 2019 14:24:29.673 * MASTER <-> REPLICATION sync: Loading DB in memory
15787:S 07 Aug 2019 14:24:29.674 * MASTER <-> REPLICATION sync: Finished with success

15793:S 07 Aug 2019 14:25:06.647 # 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.
15793:S 07 Aug 2019 14:25:06.647 # 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 after THP is disabled.
15793:S 07 Aug 2019 14:25:06.647 * DB loaded from disk: 0.000 seconds
15793:S 07 Aug 2019 14:25:06.647 * Before turning into a replica, using my master parameters to synthesize a cached master: I may be able to synchronize with the new master with just a partial transfer.
15793:S 07 Aug 2019 14:25:06.647 * Ready to accept connections
15793:S 07 Aug 2019 14:25:06.647 * Connecting to MASTER 127.0.0.1:6380
15793:S 07 Aug 2019 14:25:06.649 * MASTER <-> REPLICATION sync started
15793:S 07 Aug 2019 14:25:06.649 * Non blocking connect for SYNC fired the event.
15793:S 07 Aug 2019 14:25:06.649 * Master replied to PING, replication can continue...
15793:S 07 Aug 2019 14:25:06.649 * Trying a partial resynchronization (request ba76c6258e3b31bb1f8c95557dc9673e8512566611).
15793:S 07 Aug 2019 14:25:06.649 * Successful partial resynchronization with master.
15793:S 07 Aug 2019 14:25:06.649 * MASTER <-> REPLICATION sync: Master accepted a Partial Resynchronization.

[root@localhost master-slave]# redis-cli -p 6381
-bash: redis-cli: command not found
[root@localhost master-slave]# clear
[root@localhost master-slave]# redis-cli -p 6381
127.0.0.1:6381> set test check
(error) READONLY You can't write against a read only replica.
127.0.0.1:6381> keys *
1) "test"
127.0.0.1:6381> get test
"check"
127.0.0.1:6381>

[root@localhost master-slave]# redis-cli -p 6380
127.0.0.1:6380> set test check
OK
127.0.0.1:6380>

15783:M 07 Aug 2019 14:24:17.148 # Server initialized
15783:M 07 Aug 2019 14:24:17.148 # 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.
15783:M 07 Aug 2019 14:24:17.149 # 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 after THP is disabled.
15783:M 07 Aug 2019 14:24:17.149 * Ready to accept connections
15783:M 07 Aug 2019 14:24:29.528 * Replica 127.0.0.1:6381 asks for synchronization
15783:M 07 Aug 2019 14:24:29.528 * Full resync requested by replica 127.0.0.1:6381
15783:M 07 Aug 2019 14:24:29.528 * Starting BGSAVE for SYNC with target: disk
15783:M 07 Aug 2019 14:24:29.570 * Background saving started by pid 15791
15791:C 07 Aug 2019 14:24:29.572 * DB saved on disk
15791:C 07 Aug 2019 14:24:29.572 * I/O: 4 MB of memory used by copy-on-write
15783:M 07 Aug 2019 14:24:29.672 * Background saving terminated with success
15783:M 07 Aug 2019 14:24:29.673 * Synchronization with target: disk
15783:M 07 Aug 2019 14:25:06.649 * Replica 127.0.0.1:6381 asks for synchronization
15783:M 07 Aug 2019 14:25:06.649 * Partial resynchronization request from 127.0.0.1:6381 accepted. Sending 42 bytes of backlog starting from offset 1.

```

知乎 @黄青

## 3 Sentinel哨兵模式

### 3.1 主从模式的缺陷

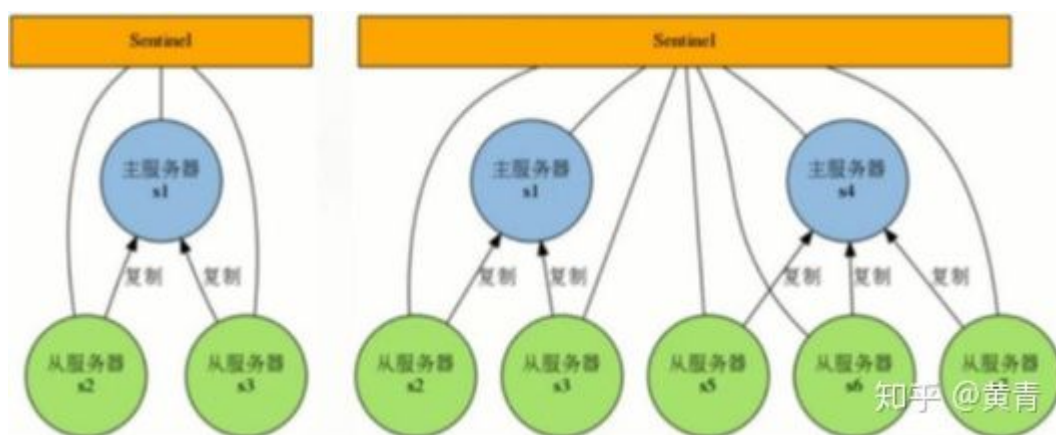
当主节点宕机了，整个集群就没有可写的节点了。由于从节点上备份了主节点的所有数据，在主节点宕机的情况下，如果能够将从节点变成一个主节点，是不是就可以解决这个问题？这个就是Sentinel哨兵的作用。

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance），该系统执行以下三个任务：

- 监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。
- 提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。
- 自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作，它会进行选举，将其中一个从服务器升级为新的主服务器，并让失效主服务器的其他从服务器改为复制新的主服务器；当客户端试图连接失效的主服务器时，集群也会向客户端返回新主服务器的地址，使得集群可以使用新主服务器代替失效服务器。

### 3.2.1 监控（Monitoring）

1. Sentinel可以监控任意多个Master和该Master下的Slaves。（即多个主从模式）
2. 同一个哨兵下的、不同主从模型，彼此之间相互独立。
3. Sentinel会不断检查Master和Slaves是否正常。

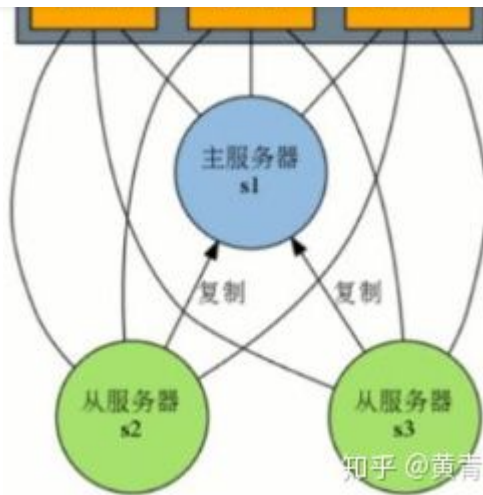


### 3.2.2 自动故障切换（Automatic failover）

#### 3.2.2.1 Sentinel网络

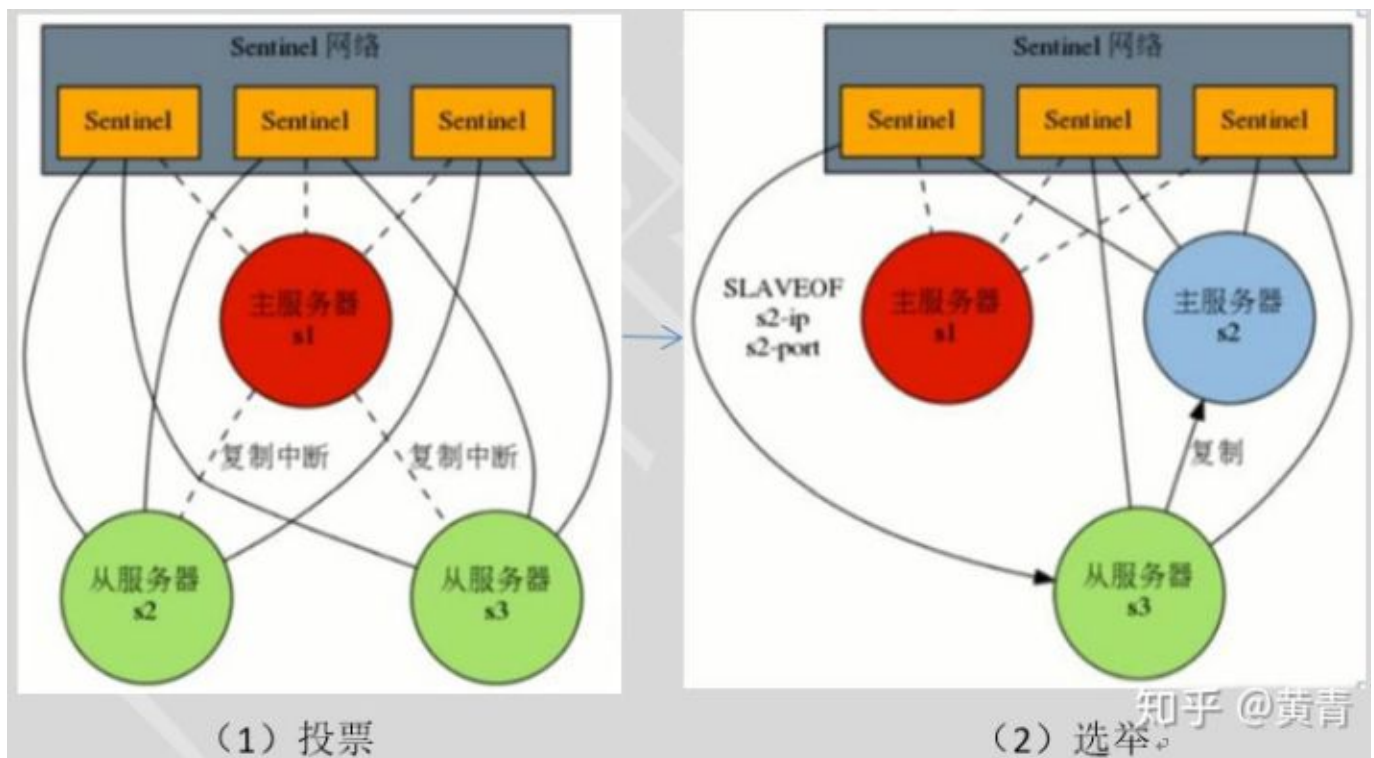
监控同一个Master的Sentinel会自动连接，组成一个分布式的Sentinel网络，互相通信并交换彼此关于被监视服务器的信息。下图中，三个监控s1的Sentinel，自动组成Sentinel网络结构。



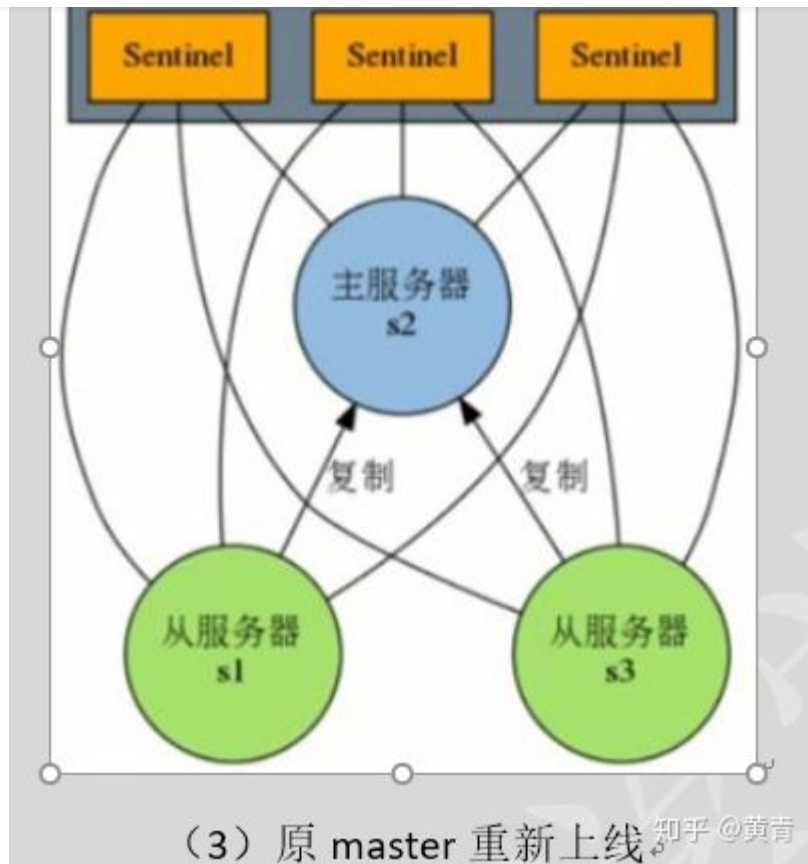


### 3.2.2.2 故障切换的过程

1. 投票（半数原则）：当任何一个Sentinel发现被监控的Master下线时，会通知其它的Sentinel开会，投票确定该Master是否下线（半数以上，所以sentinel通常配奇数个）。
2. 选举：当Sentinel确定Master下线后，会在所有的Slaves中，选举一个新的节点，升级成Master节点。其它Slaves节点，转为该节点的从节点。
3. 当原Master节点重新上线后，自动转为当前Master节点的从节点。







### 3.3 哨兵模式部署

#### 3.3.1 需求:

前提：已经存在一个正在运行的主从模式。

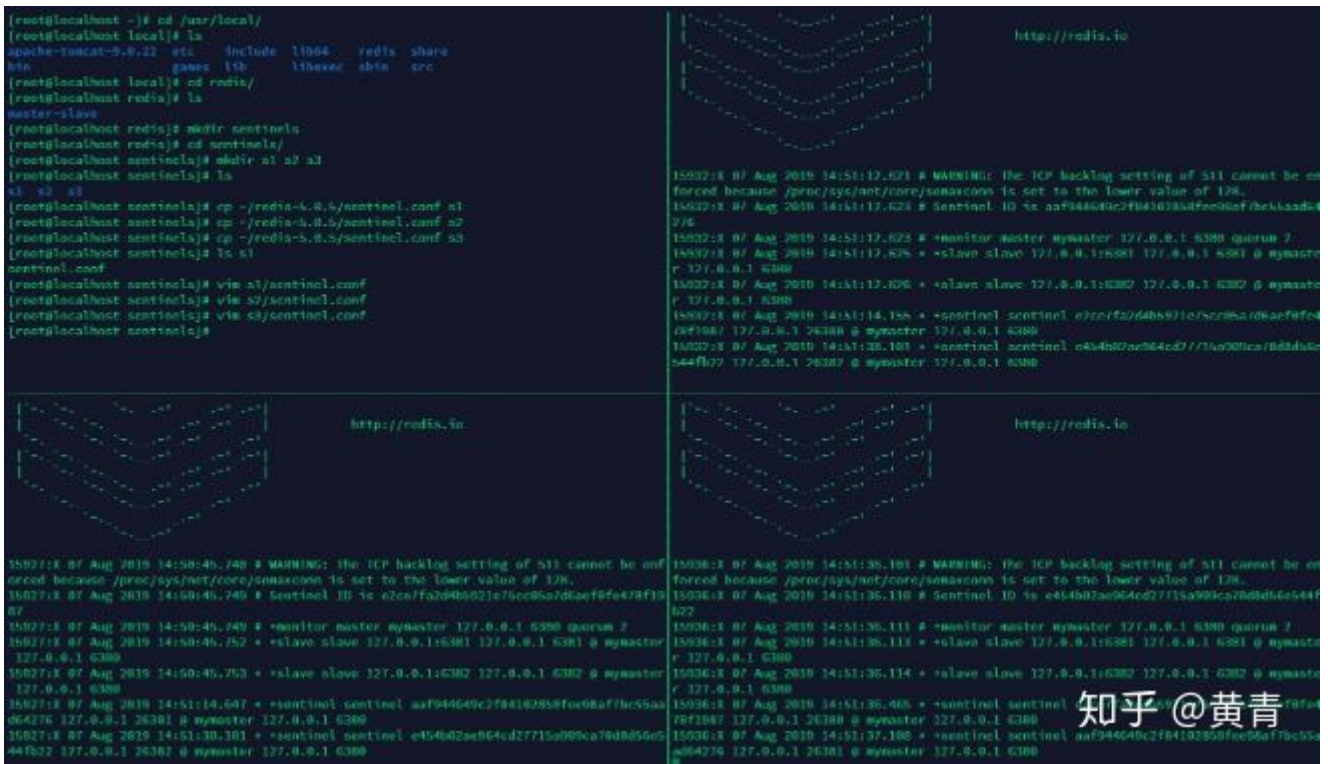
配置三个Sentinel实例，监控同一个Master节点。

#### 3.3.2 配置Sentinel

1. 在 `/usr/local` 目录下，创建 `/redis/sentinels/` 目录
2. 在 `/sentinels` 目录下，依次创建 `s1`、`s2`、`s3` 三个子目录
3. 依次拷贝 `redis` 解压目录下的 `sentinel.conf` 文件，到这三个子目录中
4. 依次修改 `s1`、`s2`、`s3` 子目录中的 `sentinel.conf` 文件，修改端口，并指定要监控的主节点。（从节点不需要指定，sentinel会自动识别）

```
9 # port <sentinel-port>
10 # The port that this sentinel instance will run on
11 port 26381 运行端口
12
```

依次启动三个哨兵后，可以看到日志输出



```

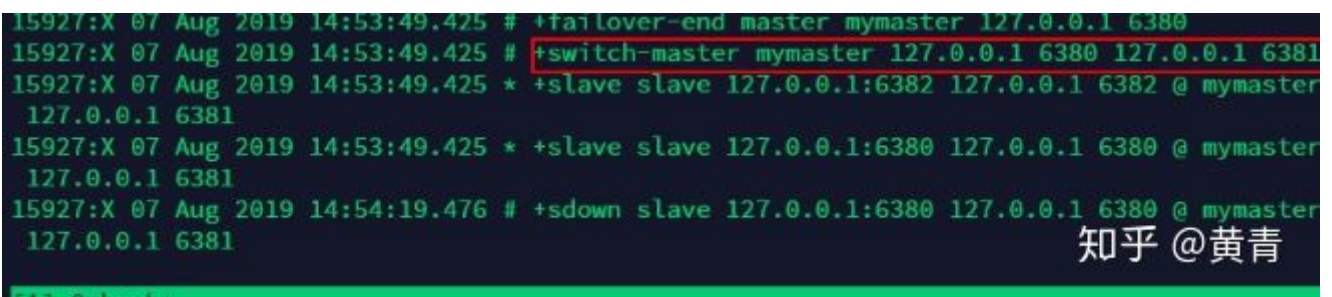
[root@localhost ~]# cd /usr/local/
[root@localhost local]# ls
apache-tomcat-9.0.22  etc  include  lib64  redis  share
bin  games  lib  libexec /sbin  src
[root@localhost local]# cd redis/
[root@localhost redis]# ls
master-slave
[root@localhost redis]# mkdir sentinel
[root@localhost redis]# cd sentinel/
[root@localhost sentinel]# mkdir s1 s2 s3
[root@localhost sentinel]# ls
s1  s2  s3
[root@localhost sentinel]# cp -r /redis-4.0.6/sentinel.conf s1
[root@localhost sentinel]# cp -r /redis-4.0.6/sentinel.conf s2
[root@localhost sentinel]# cp -r /redis-4.0.6/sentinel.conf s3
[root@localhost sentinel]# ls
s1  sentinel.conf
[root@localhost sentinel]# vim s1/sentinel.conf
[root@localhost sentinel]# vim s2/sentinel.conf
[root@localhost sentinel]# vim s3/sentinel.conf
[root@localhost sentinel]#

15927:1 07 Aug 2019 14:53:49.425 # +failover-end master mymaster 127.0.0.1 6380
15927:1 07 Aug 2019 14:53:49.425 # +switch-master mymaster 127.0.0.1 6380 127.0.0.1 6381
15927:1 07 Aug 2019 14:53:49.425 * +slave slave 127.0.0.1:6382 127.0.0.1 6382 @ mymaster
127.0.0.1 6381
15927:1 07 Aug 2019 14:53:49.425 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster
127.0.0.1 6381
15927:1 07 Aug 2019 14:54:19.476 # +sdown slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster
127.0.0.1 6381

```

### 3.3.3 测试

1. 手动关闭 6380 节点后，发现重新指定新主节点，并将另外两个节点作为从节点加入

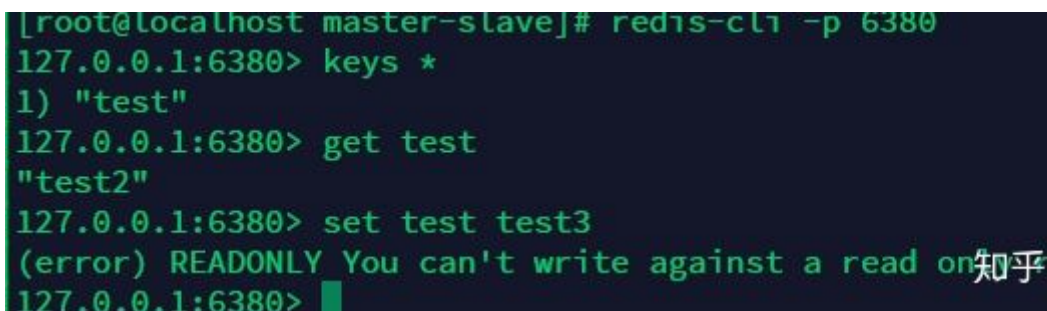


```

15927:1 07 Aug 2019 14:53:49.425 # +failover-end master mymaster 127.0.0.1 6380
15927:1 07 Aug 2019 14:53:49.425 # +switch-master mymaster 127.0.0.1 6380 127.0.0.1 6381
15927:1 07 Aug 2019 14:53:49.425 * +slave slave 127.0.0.1:6382 127.0.0.1 6382 @ mymaster
127.0.0.1 6381
15927:1 07 Aug 2019 14:53:49.425 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster
127.0.0.1 6381
15927:1 07 Aug 2019 14:54:19.476 # +sdown slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster
127.0.0.1 6381

```

再次上线 6380，发现被指定为从节点，只能读不能写



```

[root@localhost master-slave]# redis-cli -p 6380
127.0.0.1:6380> keys *
1) "test"
127.0.0.1:6380> get test
"test2"
127.0.0.1:6380> set test test3
(error) READONLY You can't write against a read-only replica
127.0.0.1:6380>

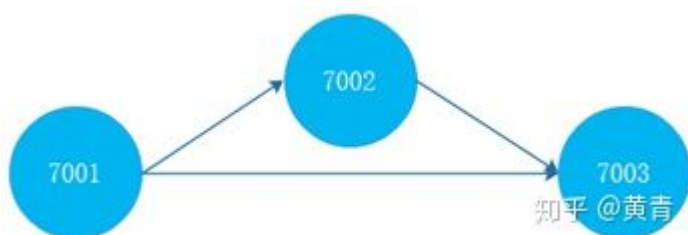
```

## 4.1 哨兵模式的缺陷

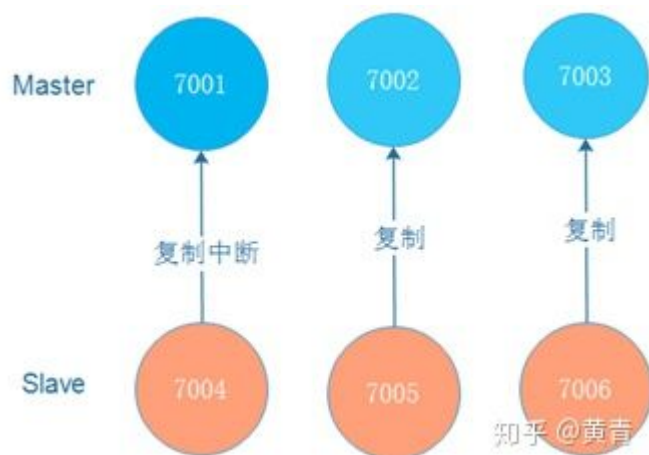
在哨兵模式中，仍然只有一个Master节点。当并发写请求较大时，哨兵模式并不能缓解写压力。我们知道只有主节点才具有写能力，那如果在一个集群中，能够配置多个主节点，是不是就可以缓解写压力了呢？是的。这个就是redis-cluster集群模式。

## 4.2 Redis-cluster集群概念

1. 由多个Redis服务器组成的分布式网络服务集群；
2. 集群之中有多个Master主节点，每一个主节点都可读可写；
3. 节点之间会互相通信，两两相连；
4. Redis集群无中心节点。



## 4.3 集群节点复制



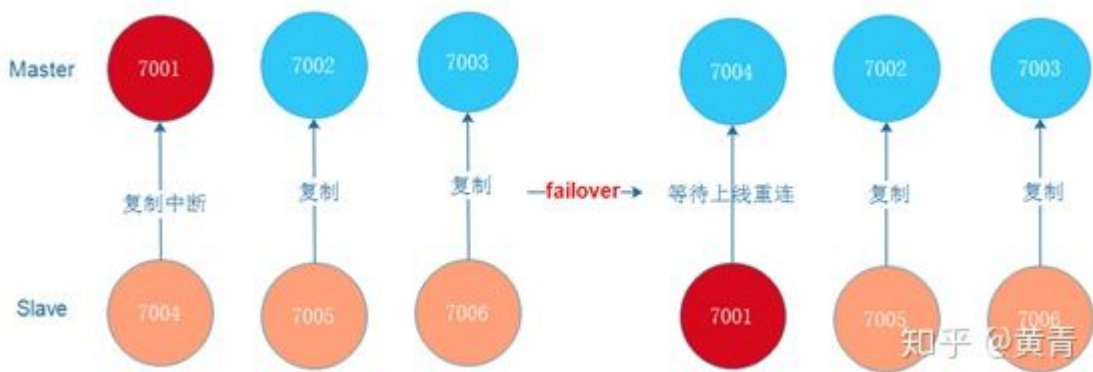
在Redis-Cluster集群中，可以给每一个主节点添加从节点，主节点和从节点直接遵循主从模型的特性。当用户需要处理更多读请求的时候，添加从节点可以扩展系统的读性能。

## 4.4 故障转移





移。



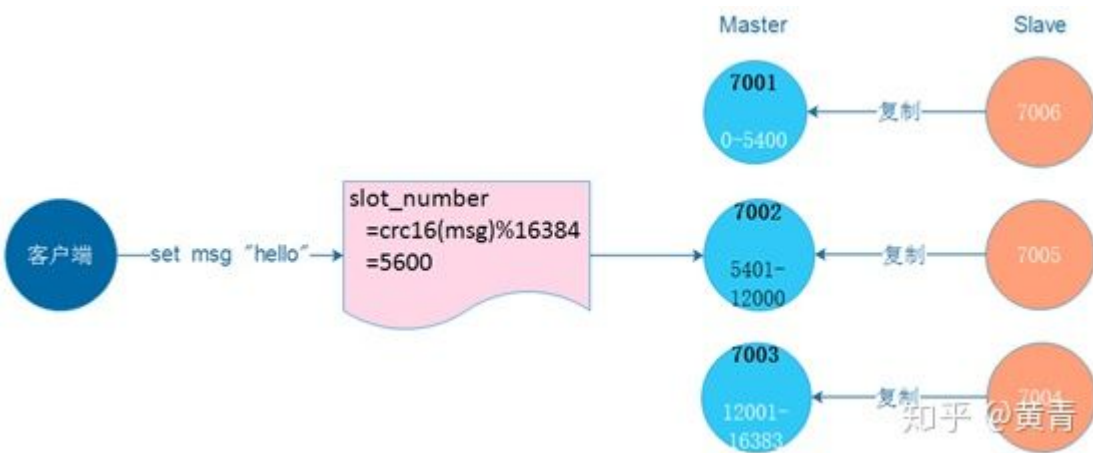
集群进行故障转移的方法和Redis Sentinel进行故障转移的方法基本一样，不同的是，在集群里面，故障转移是由集群中其他在线的主节点负责进行的，所以集群不必另外使用Redis Sentinel。

4.5 集群分片策略

Redis-cluster分片策略，是用来解决key存储位置的。

集群将整个数据库分为16384个槽位slot，所有key-value数据都存储在这些slot中的某一个上。一个slot槽位可以存放多个数据，key的槽位计算公式为： $slot\_number=crc16(msg)\%16384$ ，其中crc16为16位的循环冗余校验和函数。

集群中的每个主节点都可以处理0个至16383个槽，当16384个槽都有某个节点在负责处理时，集群进入上线状态，并开始处理客户端发送的数据命令请求。



4.6 集群redirect转向

端口重新向正确的负责的主节点发起命令请求。

## 4.7 集群搭建

### 4.7.1 准备工作

#### 安装ruby环境

redis 集群管理工具 redis-trib.rb 依赖 ruby 环境，首先需要安装 ruby 环境：

```
yum -y install ruby
yum -y install rubygems
```

#### 安装ruby和redis的接口程序

拷贝redis-3.0.0.gem至/usr/local下，执行安装：

```
gem install /usr/local/redis-3.0.0.gem
```

或者直接用 gem 在线安装

```
gem install reids
```



## redis-trib.rb 工具

官方提供了此工具用于挂历 redis 集群，该工具就在解压目录的 src 目录下：

### 4.7.2 集群规划

Redis集群最少需要6个节点，可以分布在一台或者多台主机上。以下测试为在一台主机上创建伪分布式集群，不同的端口表示不同的redis节点，如下：

- 主节点：192.168.56.3:7001 192.168.56.3:7002 192.168.56.3:7003
- 从节点：192.168.56.3:7004 192.168.56.3:7005 192.168.56.3:7006

在 /usr/local/redis 下创建 redis-cluster 目录，其下创建7001、7002...7006目录，复制 redis.conf 配置文件到每个文件夹，并配置：

```
# 必选配置
port 700X
bind 192.168.X.X
```



```
daemonized yes  
logfile /usr/local/redis/redis-cluster/700X/node.log
```

### 4.7.3 启动每个结点redis服务

依次以700X下的redis.conf，启动redis节点。（必须指定redis.conf文件）

注意，需要分别进入各个文件夹启动，不然会报 cluster config file 已经被使用的错误

### 4.7.4 执行创建集群命令

进入到 redis 源码存放目录 src 目录下，执行redis-trib.rb，此脚本是ruby脚本，它依赖ruby环境。

```
./redis-trib.rb create --replicas 1 192.168.163.88:7001  
192.168.163.88:7002 192.168.163.88:7003 192.168.163.88:7004 192.168.163.88:7005 192.1
```

这里发现，最新版 redis 已经取消对 redis-trib.rb 的支持，采用示例方法创建集群：





## 4.7.5 查询集群信息

集群创建成功登陆任意redis结点查询集群中的节点情况：

- -c：表示以集群方式连接redis
- -h：指定host ip地址
- -p：指定端口号
- cluster nodes：查询集群结点信息
- cluster info：查询集群状态信

## 4.8 集群管理

### 4.8.1 添加主节点

#### 4.8.1.1 节点规划

集群创建成功后可以向集群中添加节点，下面是添加一个master主节点，添加7007节点。



执行下边命令添加节点(第一个地址为新节点, 第二个地址为 cluster 集群中的任意一个节点地址):

```
./redis-trib.rb add-node 192.168.23.3:7007 192.168.23.3:7001 //  
已过时 redis-cli --cluster add-node 192.168.163.88:7007 192.168.163.88:7002
```

运行 `redis-cli --cluster check 192.168.163.88:7001` 检查状态, 发现新节点作为主节点加入, 但没有 slot 分配给它。

## 4.8.1.2 slot槽重新分配



▲ 赞同 5

## 第一步：连接上集群

连接集群中任意一个可用节点都行

```
redis-cli --cluster reshard 192.168.163.88:7001
```

## 第二步：输入要分配的槽数量

输入 500表示要分配500个槽

## 第三步：输入接收槽的结点id

这里输入的是新加入的节点 7007



## 第四步：输入源结点id

这里我选择从 7001 作为源节点获取 500 个 slot，也可以输入 all 表示从所有主节点中平均获取。  
可以输入多个源节点，输入done 表示输入结束。



▲ 赞同 5



第五步：检查结果



▲ 赞同 5

如果只是从一个源里转移，可以使用一句语句完成操作

```
redis-cli --cluster reshard 192.168.163.88:7001 --cluster-from  
95252ffbf34bb114b859ed7da8a312e28347d5c1 --cluster-to  
e272188208df9d9080d41a89a0fffd49e503879c --cluster-slots 500
```

## 4.8.2 添加从节点

为新增的主节点添加从节点，将 7008 作为 7007 的从节点。

```
redis-cli --cluster add-node 192.168.163.88:7008 192.168.163.88:7001 --cluster-slave
```

若不指定 `--cluster-master-id` 同时声明了 `--cluster-slave`，则默认会添加为第二个地址的从节点

若 7008 下面已有 nodes.conf，添加时可能会报错，解决方法是删除该文件后再添加

## 4.8.3 删除节点

使用命令

```
redis-cli --cluster del-node 192.168.163.88:7001 d5d9af031a714c4fe334e8950de46add16c0
```

第一个地址为 cluster 任一节点，后面 id 为需要删除的节点 id

需要注意的是，若删除的节点为主节点，需要将其所拥有的 slot 分配出去后才能删除，不然会报如下错误

将 7007 的 slot 转移回 7001 后删除



▲ 赞同 5

## 5 java程序连接redis集群

### 5.1 连接步骤

#### 5.1.1 第一步：创建项目，导入jar包

#### 5.1.2 第二步：创建redis集群的客户端





```
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisCluster;

import java.util.HashSet;
import java.util.Set;

public class ClusterTest {

    @Test
    public void ClusterConnectionTest() {
        //创建 set 集合封装所有节点信息
        Set<HostAndPort> nodes = new HashSet<>();

        //只需要添加一个节点即可，会自动搜索其它节点
        nodes.add(new HostAndPort("192.168.163.88", 7001));
        /*nodes.add(new HostAndPort("192.168.163.88", 7002));
        nodes.add(new HostAndPort("192.168.163.88", 7003));
        nodes.add(new HostAndPort("192.168.163.88", 7004));
        nodes.add(new HostAndPort("192.168.163.88", 7005));
        nodes.add(new HostAndPort("192.168.163.88", 7006));*/

        //使用节点创建一个 JedisCluster 对象
        JedisCluster jedisCluster = new JedisCluster(nodes);

        //测试连接结果
        System.out.println(jedisCluster.get("wtf"));
    }

    @Test
    public void RedisConnectionTest(){
        Jedis jedis = new Jedis("192.168.163.88", 7001);
        System.out.println(jedis.ping());
        String hello = jedis.get("hello"); //hello 存在 7001
        System.out.println(hello);

        //采用普通方式连接，若数据不是存在此节点，会报错
        /*String wtf = jedis.get("wtf"); //wtf 存在 7002
        System.out.println(wtf);*/
    }
}
```



连接Redis集群时，需要修改防火墙，开方每一个redis节点的端口。

说明：如果要开发一个范围的端口，可以使用冒号来分割，即： 7001:7008，表示开发7001-7008之间所有的端口，或者使用 setup 工具设置

发布于 2019-08-16

[Redis](#) [分布式系统](#) [Java](#)

## 推荐阅读

### Redis的持久化和集群搭建

Redis 简介Remote Dictionary Server(Redis)是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。它通常被称为数据结...

争青

发表于java ...

## 还没有评论

写下你的评论...



▲ 赞同 5