

原 redis哨兵、集群

1. 安装Redis3.0

```
yum -y install cpp binutils glibc glibc-kernheaders glibc-common glibc-devel gcc  
make gcc-c++ libstdc++-devel tcl
```

```
mkdir -p /usr/local/src/Redis
```

```
cd /usr/local/src/redis
```

```
wget http://download.redis.io/releases/redis-3.0.2.tar.gz 或者 rz 上传
```

```
tar -xvf redis-3.0.2.tar.gz
```

```
cd redis-3.0.2
```

```
make
```

```
make test #这个就不要执行了，需要很长时间
```

```
make install
```

```
cp redis.conf /etc/
```

```
vi /etc/redis.conf
```

```
# 修改如下，默认为no
```

```
daemonize yes
```

```
#启动
```

```
redis-server /etc/redis.conf
```

```
#测试
```

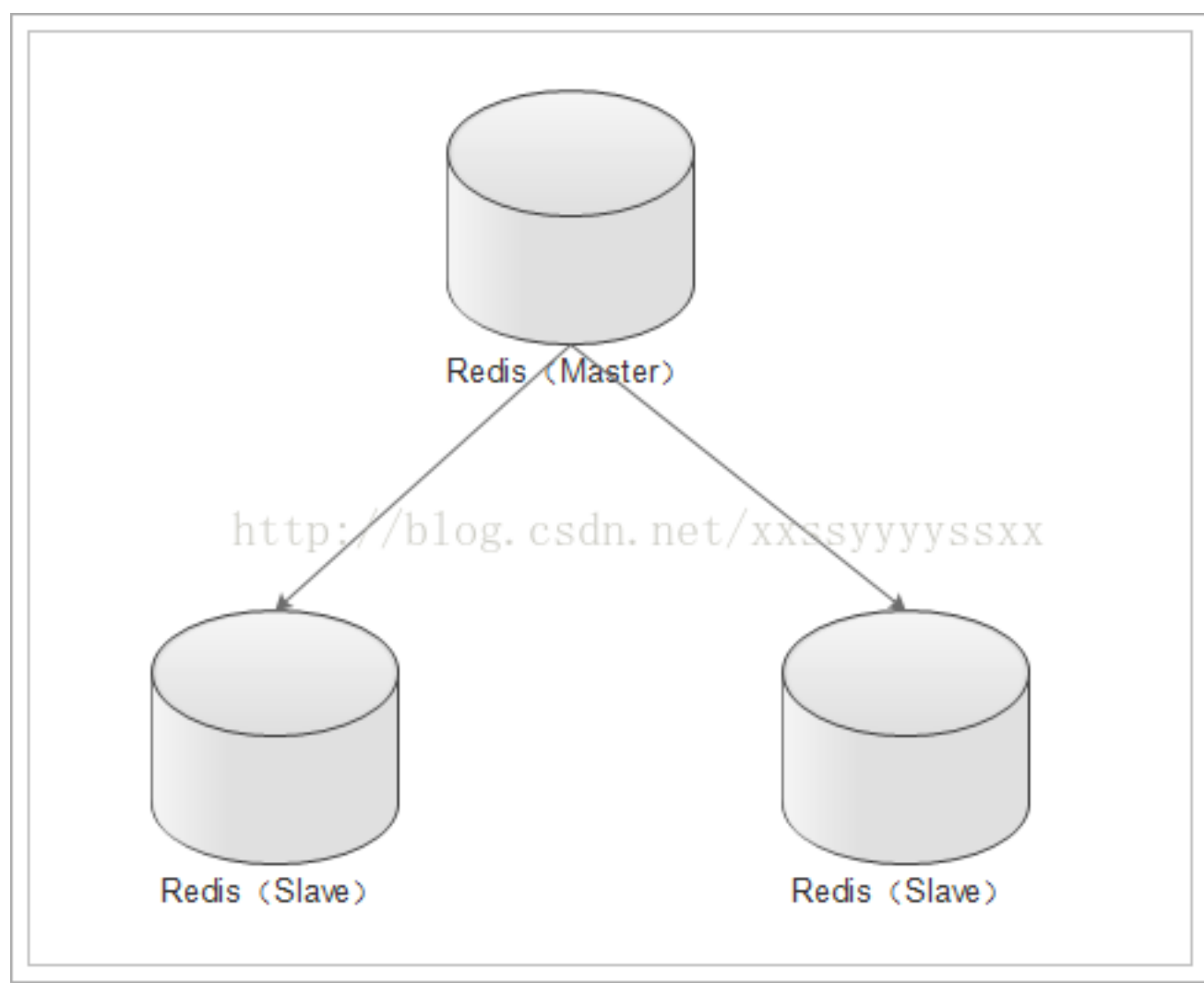
```
redis-cli
```

2. 主从复制（读写分离）

主从复制的好处有2点：

- 1、 避免redis单点故障
- 2、 构建读写分离[架构](#)，满足读多写少的应用场景

2.1. 主从架构



2.1.1. 启动实例

创建6379、6380、6381目录，分别将安装目录下的redis.conf拷贝到这三个目录下。

```
drwxr-xr-x. 2 root root 4096 6月 3 11:30 6379
drwxr-xr-x. 2 root root 4096 6月 3 11:31 6380
drwxr-xr-x. 2 root root 4096 6月 3 11:31 6381
```

分别进入这三个目录，分别修改配置文件，将端口分别设置为：6379（Master）、6380（Slave）、6381（Slave）。同时要设置pidfile文件为不同的路径。

分别启动三个redis实例：

```
[root@taotao2 6381]# ps -ef|grep redis
root      1931      1  0 17:25 ?        00:00:00 redis-server *:6379
root      1941      1  0 17:26 ?        00:00:00 redis-server *:6380
root      1971      1  0 17:27 ?        00:00:00 redis-server *:6381
root      1976    1428  0 17:28 pts/0    00:00:00 grep redis
```

2.1.2. 设置主从

在redis中设置主从有2种方式：

1、 在redis.conf中设置slaveof

a) slaveof <masterip> <masterport>

2、 使用redis-cli客户端连接到redis服务，执行slaveof命令

a) slaveof <masterip> <masterport>

第二种方式在重启后将失去主从复制关系。

查看主从信息： INFO replication

主：

```
127.0.0.1:6379> INFO replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=99,lag=1
slave1:ip=127.0.0.1,port=6381,state=online,offset=99,lag=1
master_repl_offset:99
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:98
127.0.0.1:6379> █
```

role: 角色

connected_slaves: 从库数量

slave0: 从库信息

从:

```
127.0.0.1:6380> INFO replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:10
master_sync_in_progress:0
slave_repl_offset:393
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

2.1.3. 测试

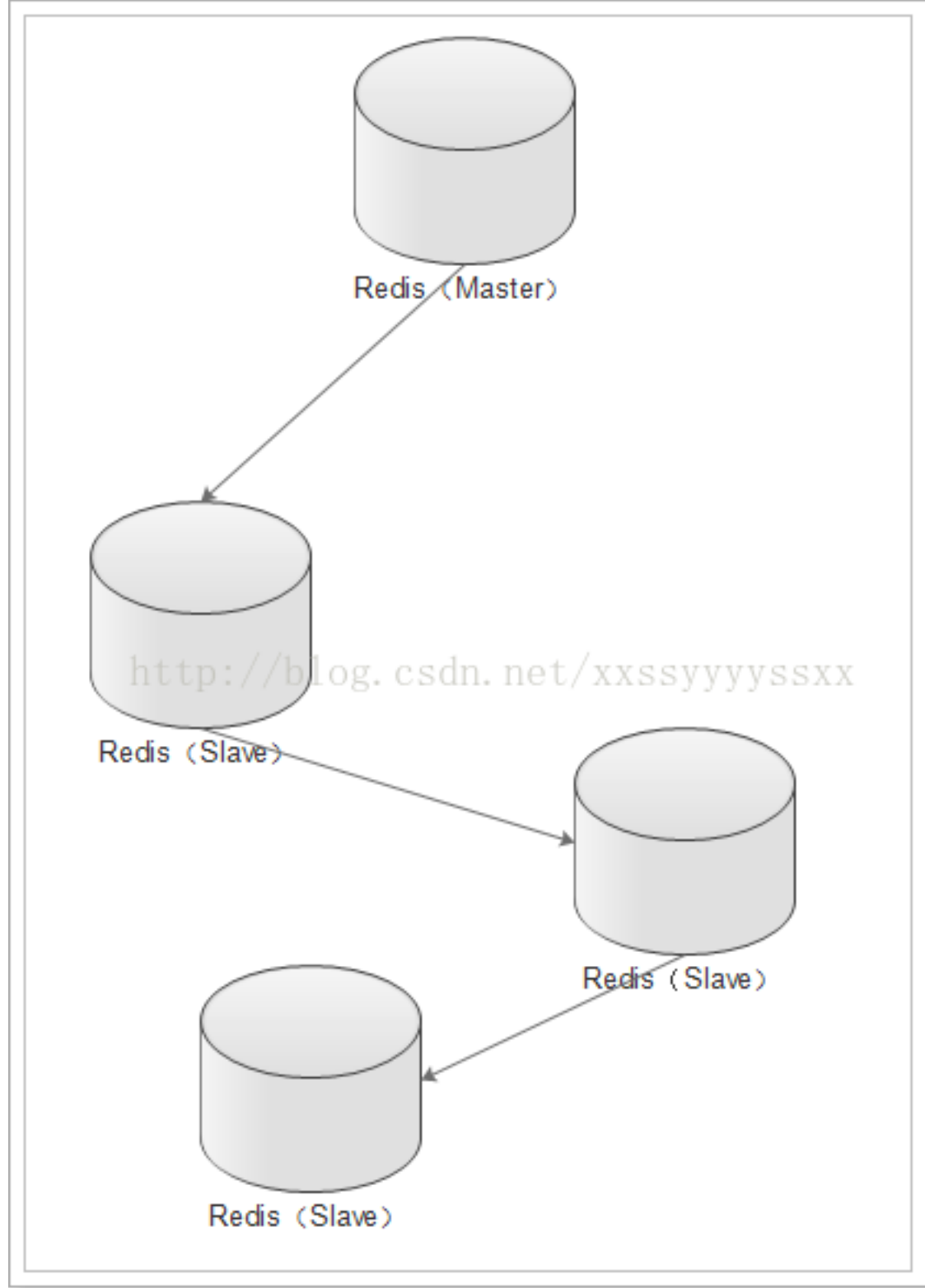
在主库写入数据:

```
127.0.0.1:6379> set abc  
OK  
127.0.0.1:6379> get abc  
"123"  
127.0.0.1:6379> █
```

在从库读取数据：

```
[root@taotao2 redis]# redis-cli -p 6380  
127.0.0.1:6380>  
127.0.0.1:6380>  
127.0.0.1:6380>  
127.0.0.1:6380> keys *  
1) "abc"  
127.0.0.1:6380> get abc  
"123"  
..... █
```

2.2. 主从从架构



2.2.1. 启动实例

```
[root@taotao2 redis]# ps -ef|grep redis
root      2526      1   0 18:27 ?        00:00:00 redis-server *:6379
root      2536      1   0 18:28 ?        00:00:00 redis-server *:6380
root      2546      1   0 18:29 ?        00:00:00 redis-server *:6381
root      2557  1428   0 18:30 pts/0    00:00:00 grep redis
```

设置主从：

```
127.0.0.1:6380> INFO replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:7
master_sync_in_progress:0
slave_repl_offset:15
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

设置从从:

```
127.0.0.1:6381> INFO replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6380
master_link_status:up
master_last_io_seconds_ago:1
master_sync_in_progress:0
slave_repl_offset:15
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

2.2.2. 测试

在主库设置数据:

```
127.0.0.1:6379> set abc 123
OK
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> get abc
"123"
127.0.0.1:6379> █
```

在6380获取数据：

```
[root@taotao2 redis]# redis-cli -p 6380
127.0.0.1:6380>
127.0.0.1:6380>
127.0.0.1:6380>
127.0.0.1:6380> get abc
"123"
127.0.0.1:6380> █
```

在6381获取数据：

```
[root@taotao2 redis]# redis-cli -p 6381
127.0.0.1:6381> get abc
"123"
127.0.0.1:6381> █
```

2.3. 从库只读

默认情况下redis数据库充当slave角色时是只读的不能进行写操作。

```
127.0.0.1:6380> set a 1
(error) READONLY You can't write against a read only slave.
127.0.0.1:6380> █
```

可以在配置文件中开启非只读：slave-read-only no

2.4. 复制的过程原理

- 1、 当从库和主库建立MS关系后，会向主数据库发送SYNC命令；
- 2、 主库接收到SYNC命令后会开始在后台保存快照（RDB持久化过程），并将期间接收到的写命令缓存起来；
- 3、 当快照完成后，主Redis会将快照文件和所有缓存的写命令发送给从Redis；

- 4、从Redis接收到后，会载入快照文件并且执行收到的缓存的命令；
- 5、之后，主Redis每当接收到写命令时就会将命令发送从Redis，从而保证数据的一致；

2.5. 无磁盘复制

通过前面的复制过程我们了解到，主库接收到SYNC的命令时会执行RDB过程，即使在配置文件中禁用RDB持久化也会生成，那么如果主库所在的服务器磁盘IO性能较差，那么这个复制过程就会出现瓶颈，庆幸的是，Redis在2.8.18版本开始实现了无磁盘复制功能（不过该功能还是处于试验阶段）。

原理：

Redis在与从数据库进行复制初始化时将不会将快照存储到磁盘，而是直接通过网络发送给从数据库，避免了IO性能差问题。

开启无磁盘复制：`repl-diskless-sync yes`

2.6. 复制架构中出现宕机情况，怎么办？

如果在主从复制架构中出现宕机的情况，需要分情况看：

1、从Redis宕机

- a) 这个相对而言比较简单，在Redis中从库重新启动后会自动加入到主从架构中，自动完成同步数据；
- b) 问题？ 如果从库在断开期间，主库的变化不大，从库再次启动后，主库依然会将所有的数据做RDB操作吗？还是增量更新？（从库有做持久化的前提下）
 - i. 不会的，因为在Redis2.8版本后就实现了，主从断线后恢复的情况下实现增量复制。

2、主Redis宕机

a) 这个相对而言就会复杂一些，需要以下2步才能完成

i. 第一步，在从数据库中执行SLAVEOF NO ONE命令，断开主从关系并且提升为主库继续服务；

ii. 第二步，将主库重新启动后，执行SLAVEOF命令，将其设置为其他库的从库，这时数据就能更新回来；

b) 这个手动完成恢复的过程其实是比较麻烦的并且容易出错，有没有好办法解决呢？当前有的，Redis提高的哨兵（sentinel）的功能。

3. 哨兵（sentinel）

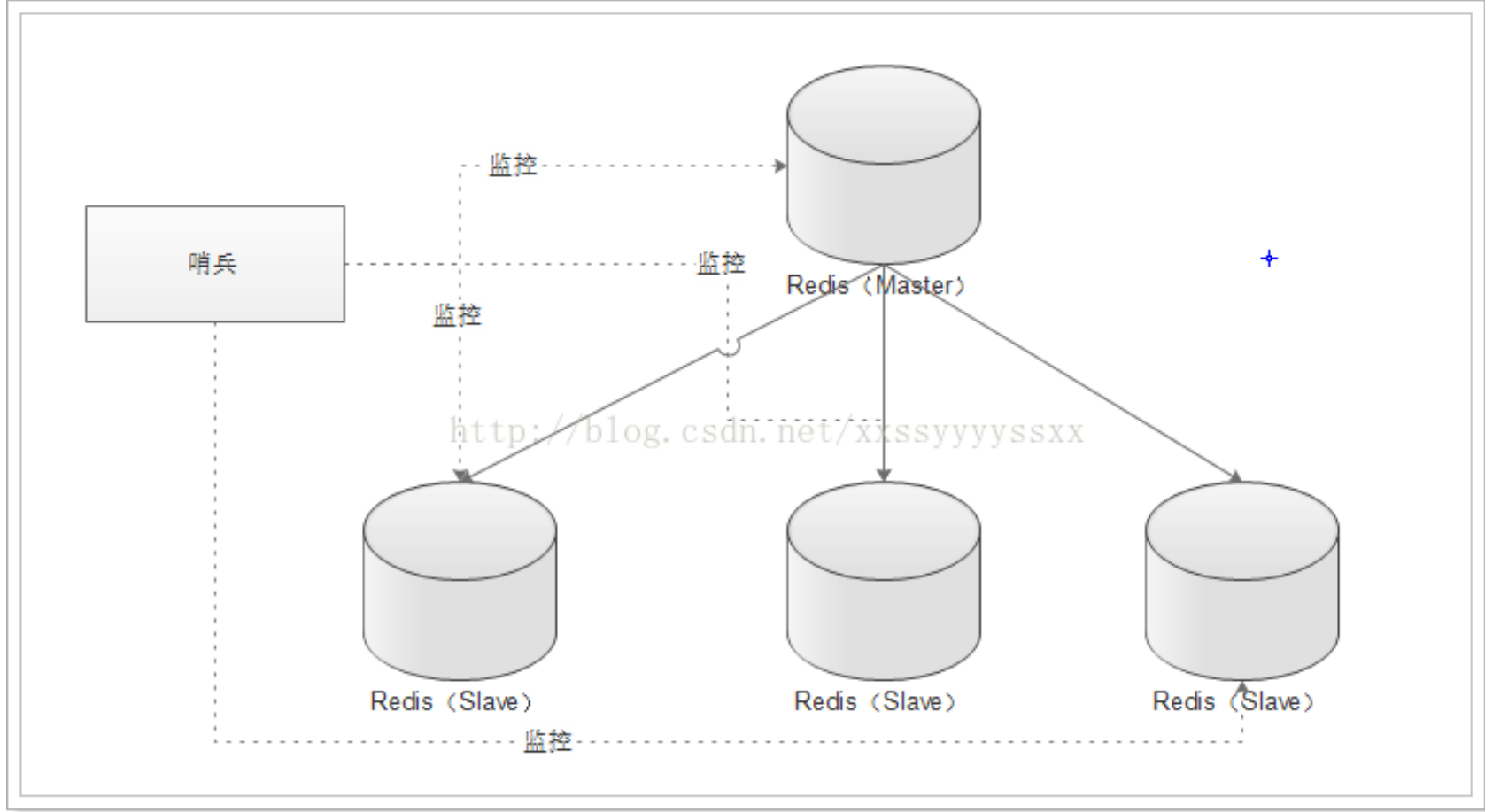
3.1. 什么是哨兵

顾名思义，哨兵的作用就是对Redis的系统的运行情况的监控，它是一个独立进程。它的功能有2个：

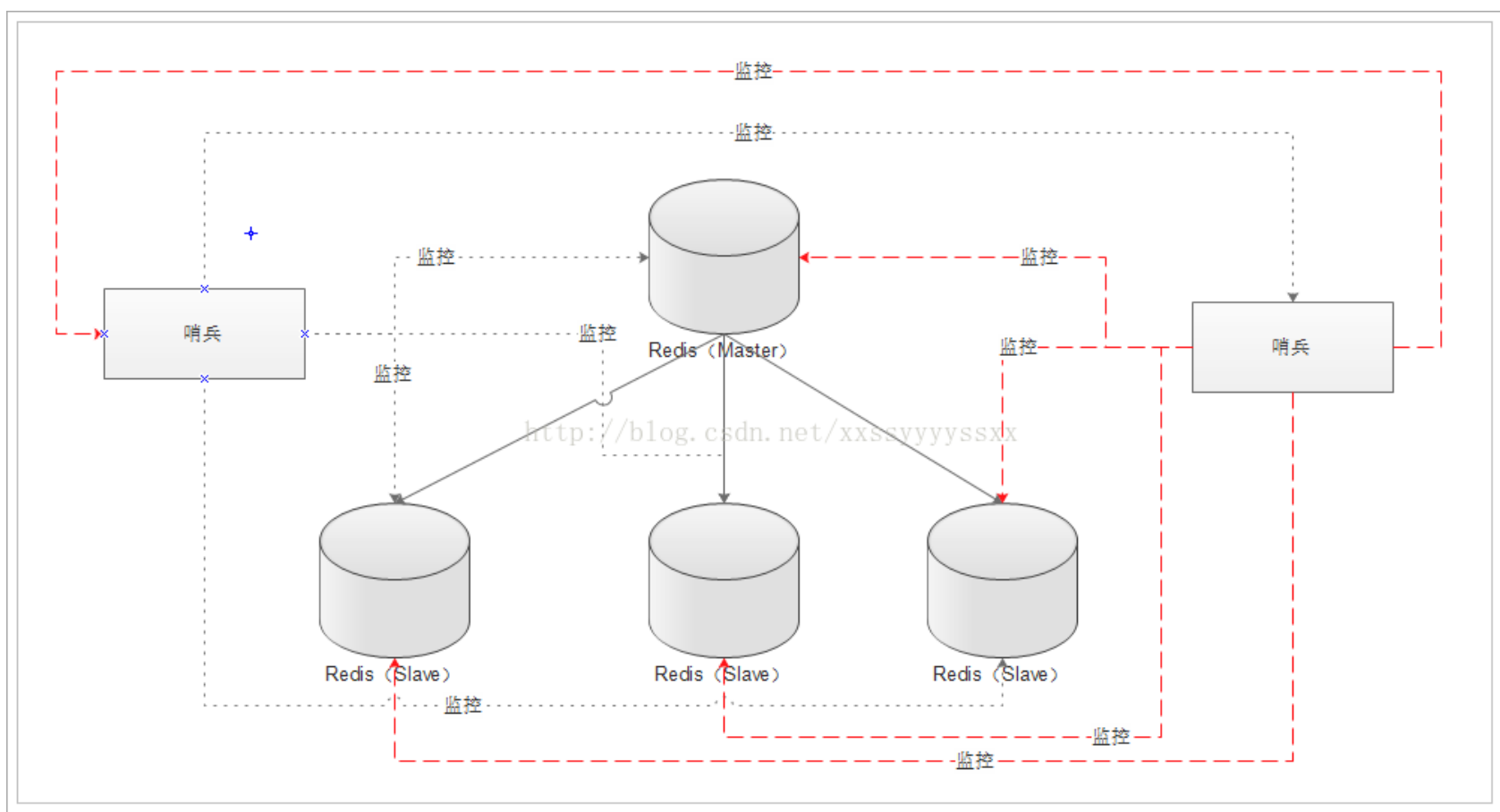
- 1、 监控主数据库和从数据库是否运行正常；
- 2、 主数据出现故障后自动将从数据库转化为主数据库；

3.2. 原理

单个哨兵的架构：



多个哨兵的架构：



多个哨兵，不仅同时监控主从数据库，而且哨兵之间互为监控。

3.3. 环境

当前处于一主多从的环境中：

```
127.0.0.1:6379> INFO replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=5279,lag=0
slave1:ip=127.0.0.1,port=6381,state=online,offset=5279,lag=0
master_repl_offset:5279
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:5278
127.0.0.1:6379>
```

3.4. 配置哨兵

启动哨兵进程首先需要创建哨兵配置文件：

```
vim sentinel.conf
```

输入内容：

```
sentinel monitor taotaoMaster 127.0.0.1 6379 1
```

说明：

taotaoMaster： 监控主数据的名称，自定义即可，可以使用大小写字母和“.-_”符号

127.0.0.1： 监控的主数据库的IP

6379： 监控的主数据库的端口

1： 最低通过票数

启动哨兵进程：

```
redis-sentinel ./sentinel.conf
```

```
[root@taotao2 redis]# redis-sentinel ./sentinel.conf
2989:X 05 Jun 19:56:55.053 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.0.1 (00000000/0) 64 bit

Running in sentinel mode
Port: 26379
PID: 2989

http://redis.io
http://blog.csdn.net/xxssyyyyssxx

2989:X 05 Jun 19:56:55.053 # Sentinel runid is 9059917216012421e8e89a4aa02f15b75346d2b7
2989:X 05 Jun 19:56:55.053 # +monitor master taotaoMaster 127.0.0.1 6379 quorum 1
2989:X 05 Jun 19:56:55.055 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ taotaoMaster 127.0.0.1 6379
2989:X 05 Jun 19:56:55.066 * +slave slave 127.0.0.1:6381 127.0.0.1 6381 @ taotaoMaster 127.0.0.1 6379
```

由上图可以看到：

- 1、 哨兵已经启动，它的id为9059917216012421e8e89a4aa02f15b75346d2b7
- 2、 为master数据库添加了一个监控
- 3、 发现了2个slave（由此可以看出，哨兵无需配置slave，只需要指定master，哨兵会自动发现slave）

3.5. 从数据库宕机

```
[root@taotao2 ~]# ps -ef|grep redis
root      2615      1   0  18:43 ?        00:00:04 redis-server *:6379
root      2626      1   0  18:44 ?        00:00:04 redis-server *:6380
root      2863      1   0  19:32 ?        00:00:01 redis-server *:6381
root      2989    1428   0  19:56 pts/0    00:00:01 redis-sentinel *:26379 [sentinel]
root      3076    3056   0  20:06 pts/1    00:00:00 grep redis
[root@taotao2 ~]#
```

kill掉2826进程后，30秒后哨兵的控制台输出：

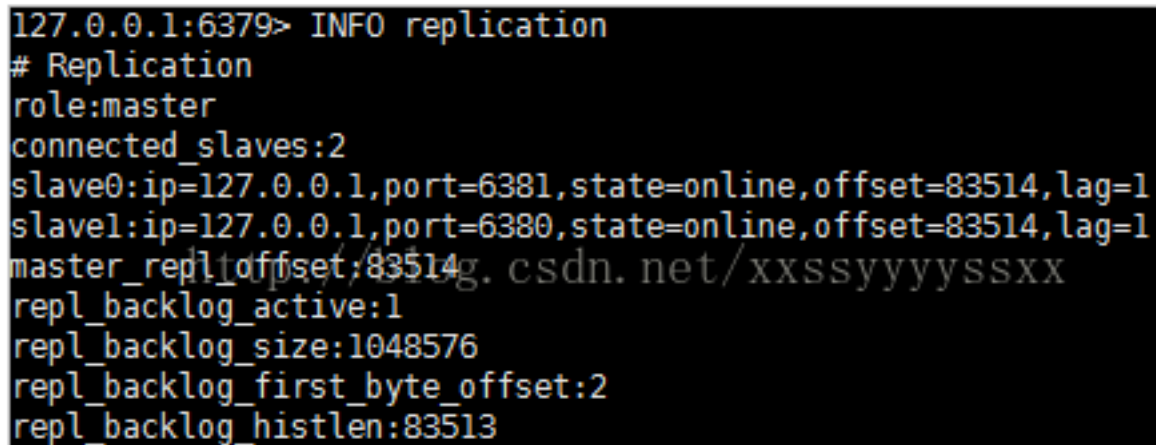
```
2989:X 05 Jun 20:09:33.509 # +sdown slave 127.0.0.1:6380 127.0.0.1 6380 @
taotaoMaster 127.0.0.1 6379
```

说明已经监控到slave宕机了，那么，如果我们将3380端口的redis实例启动后，会自动加入到主从复制吗？

```
2989:X 05 Jun 20:13:22.716 * +reboot slave 127.0.0.1:6380 127.0.0.1 6380 @  
taotaoMaster 127.0.0.1 6379
```

```
2989:X 05 Jun 20:13:22.788 # -sdown slave 127.0.0.1:6380 127.0.0.1 6380 @  
taotaoMaster 127.0.0.1 6379
```

可以看出，slave从新加入到了主从复制中。-sdown：说明是恢复服务。



```
127.0.0.1:6379> INFO replication  
# Replication  
role:master  
connected_slaves:2  
slave0:ip=127.0.0.1,port=6381,state=online,offset=83514,lag=1  
slave1:ip=127.0.0.1,port=6380,state=online,offset=83514,lag=1  
master_repl_offset:83514  
repl_backlog_active:1  
repl_backlog_size:1048576  
repl_backlog_first_byte_offset:2  
repl_backlog_histlen:83513
```

3.6. 主库宕机

哨兵控制台打印出如下信息：

```
2989:X 05 Jun 20:16:50.300 # +sdown master taotaoMaster 127.0.0.1 6379 说  
明master服务已经宕机
```

```
2989:X 05 Jun 20:16:50.300 # +odown master taotaoMaster 127.0.0.1 6379  
#quorum 1/1
```

```
2989:X 05 Jun 20:16:50.300 # +new-epoch 1
```

```
2989:X 05 Jun 20:16:50.300 # +try-failover master taotaoMaster 127.0.0.1 6379  
开始恢复故障
```

```
2989:X 05 Jun 20:16:50.304 # +vote-for-leader
```

```
9059917216012421e8e89a4aa02f15b75346d2b7 1 投票选举哨兵leader，现在就  
一个哨兵所以leader就自己
```

```
2989:X 05 Jun 20:16:50.304 # +elected-leader master taotaoMaster 127.0.0.1 6379  
选中leader
```

2989:X 05 Jun 20:16:50.304 # +failover-state-select-slave master taotaoMaster
127.0.0.1 6379 选中其中的一个slave当做master

2989:X 05 Jun 20:16:50.357 # +selected-slave slave 127.0.0.1:6381 127.0.0.1 6381
@ taotaoMaster 127.0.0.1 6379 选中6381

2989:X 05 Jun 20:16:50.357 * +failover-state-send-slaveof-noone slave
127.0.0.1:6381 127.0.0.1 6381 @ taotaoMaster 127.0.0.1 6379 发送slaveof no one
命令

2989:X 05 Jun 20:16:50.420 * +failover-state-wait-promotion slave 127.0.0.1:6381
127.0.0.1 6381 @ taotaoMaster 127.0.0.1 6379 等待升级master

2989:X 05 Jun 20:16:50.515 # +promoted-slave slave 127.0.0.1:6381 127.0.0.1 6381
@ taotaoMaster 127.0.0.1 6379 升级6381为master

2989:X 05 Jun 20:16:50.515 # +failover-state-reconf-slaves master taotaoMaster
127.0.0.1 6379

2989:X 05 Jun 20:16:50.566 * +slave-reconf-sent slave 127.0.0.1:6380 127.0.0.1
6380 @ taotaoMaster 127.0.0.1 6379

2989:X 05 Jun 20:16:51.333 * +slave-reconf-inprog slave 127.0.0.1:6380 127.0.0.1
6380 @ taotaoMaster 127.0.0.1 6379

2989:X 05 Jun 20:16:52.382 * +slave-reconf-done slave 127.0.0.1:6380 127.0.0.1
6380 @ taotaoMaster 127.0.0.1 6379

2989:X 05 Jun 20:16:52.438 # +failover-end master taotaoMaster 127.0.0.1
6379 故障恢复完成

2989:X 05 Jun 20:16:52.438 # +switch-master taotaoMaster 127.0.0.1 6379
127.0.0.1 6381 主数据库从6379转变为6381

2989:X 05 Jun 20:16:52.438 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @
taotaoMaster 127.0.0.1 6381 添加6380为6381的从库

2989:X 05 Jun 20:16:52.438 * +slave slave 127.0.0.1:6379 127.0.0.1 6379 @

taotaoMaster 127.0.0.1 6381 添加6379为6381的从库

2989:X 05 Jun 20:17:22.463 # +sdown slave 127.0.0.1:6379 127.0.0.1 6379 @

taotaoMaster 127.0.0.1 6381 发现6379已经宕机，等待6379的恢复

```
[root@taotao2 6380]# redis-cli -p 6381
127.0.0.1:6381> INFO replication
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=6380,state=online,offset=69815,lag=1
master_repl_offset:69952
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:69951
127.0.0.1:6381>
```

可以看出，目前，6381位master，拥有一个slave为6380。

接下来，我们恢复6379查看状态：

2989:X 05 Jun 20:35:32.172 # -sdown slave 127.0.0.1:6379 127.0.0.1 6379 @

taotaoMaster 127.0.0.1 6381 6379已经恢复服务

2989:X 05 Jun 20:35:42.137 * +convert-to-slave slave 127.0.0.1:6379 127.0.0.1 6379

@ taotaoMaster 127.0.0.1 6381 将6379设置为6381的slave

```
127.0.0.1:6381> INFO replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=82845,lag=0
slave1:ip=127.0.0.1,port=6379,state=online,offset=82845,lag=0
master_repl_offset:82845
```

3.7. 配置多个哨兵

vim sentinel.conf

输入内容：

sentinel monitor taotaoMaster 127.0.0.1 6381 2

sentinel monitor taotaoMaster2 127.0.0.1 6381 1

3451:X 05 Jun 21:05:56.083 # +sdown master taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:56.083 # +odown master taotaoMaster2 127.0.0.1 6381
#quorum 1/1

3451:X 05 Jun 21:05:56.083 # +new-epoch 1

3451:X 05 Jun 21:05:56.083 # +try-failover master taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:56.086 # +vote-for-leader
3f020a35c9878a12d2b44904f570dc0d4015c2ba 1

3451:X 05 Jun 21:05:56.086 # +elected-leader master taotaoMaster2 127.0.0.1
6381

3451:X 05 Jun 21:05:56.086 # +failover-state-select-slave master taotaoMaster2
127.0.0.1 6381

3451:X 05 Jun 21:05:56.087 # +sdown master taotaoMaster 127.0.0.1 6381

3451:X 05 Jun 21:05:56.189 # +selected-slave slave 127.0.0.1:6380 127.0.0.1 6380
@ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:56.189 * +failover-state-send-slaveof-noone slave
127.0.0.1:6380 127.0.0.1 6380 @ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:56.252 * +failover-state-wait-promotion slave 127.0.0.1:6380
127.0.0.1 6380 @ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:57.145 # +promoted-slave slave 127.0.0.1:6380 127.0.0.1 6380
@ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:57.145 # +failover-state-reconf-slaves master taotaoMaster2
127.0.0.1 6381

```
3451:X 05 Jun 21:05:57.234 * +slave-reconf-sent slave 127.0.0.1:6379 127.0.0.1
6379 @ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:58.149 * +slave-reconf-inprog slave 127.0.0.1:6379 127.0.0.1
6379 @ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:58.149 * +slave-reconf-done slave 127.0.0.1:6379 127.0.0.1
6379 @ taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:58.203 # +failover-end master taotaoMaster2 127.0.0.1 6381

3451:X 05 Jun 21:05:58.203 # +switch-master taotaoMaster2 127.0.0.1 6381
127.0.0.1 6380

3451:X 05 Jun 21:05:58.203 * +slave slave 127.0.0.1:6379 127.0.0.1 6379 @
taotaoMaster2 127.0.0.1 6380

3451:X 05 Jun 21:05:58.203 * +slave slave 127.0.0.1:6381 127.0.0.1 6381 @
taotaoMaster2 127.0.0.1 6380
```

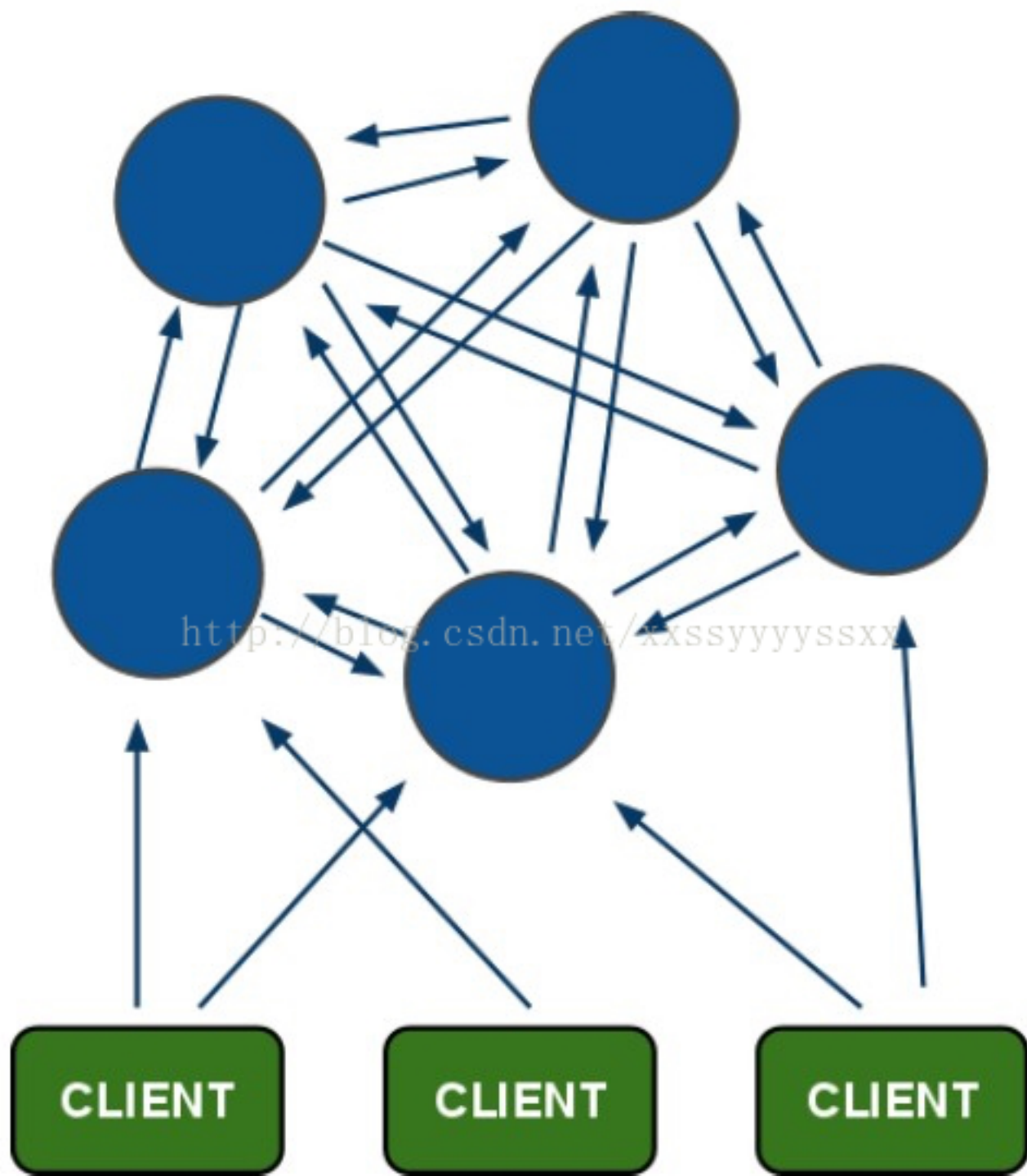
4. 集群

即使有了主从复制，每个数据库都要保存整个集群中的所有数据，容易形成木桶效应。

使用Jedis实现了分片集群，是由客户端控制哪些key数据保存到哪个数据库中，如果在水平扩容时就必须手动进行数据迁移，而且需要将整个集群停止服务，这样做非常不好的。

Redis3.0版本的一大特性就是集群（Cluster），接下来我们一起学习集群。

4.1. 架构



(1)所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.

(2)节点的fail是通过集群中超过半数的节点检测失效时才生效.

(3)客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster把所有的物理节点映射到[0-16383]slot（插槽）上,cluster 负责维护node<->slot<->value

4.2. 修改配置文件

1、 设置不同的端口，6379、6380、6381

2、开启集群，cluster-enabled yes

3、指定集群的配置文件，cluster-config-file "nodes-xxxx.conf"

```
[root@taotao2 6381]# ps -ef|grep redis
root      3568      1   0 21:30 ?        00:00:00 redis-server *:6379 [cluster]
root      3576      1   0 21:31 ?        00:00:00 redis-server *:6380 [cluster]
root      3582      1   0 21:31 ?        00:00:00 redis-server *:6381 [cluster]
root      3587    3056   0 21:31 pts/1    00:00:00 grep redis
```

4.3. 创建集群

4.3.1. 安装ruby环境

因为redis-trib.rb是有ruby语言编写的所以需要安装ruby环境。

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

```
gem install redis
```

手动安装：

rz上传redis-3.2.1.gem

```
gem install -l redis-3.2.1.gem
```

4.3.2. 创建集群

首先，进入redis的安装包路径下：

```
cd /usr/local/src/redis/redis-3.0.1/src/
```

```
[root@taotao2 src]# ls
adlist.c  anet.o      cluster.h  debug.c    intset.c  Makefile  object.o  redisassert.h  redis-cli.c  rio.c      shal.c    syncio.c  t_zset.o  zmalloc.c
adlist.h  aof.c      config.h   debug.o    intset.h  Makefile.dep  pqsort.c  redis-benchmark  redis-cli.o  rio.h     shal.h    syncio.o  util.c    zmalloc.h
adlist.o  aof.o      config.o   dict.c     intset.o  memtest.c  pqsort.h  redis-benchmark.c  redis.h     rio.o     shal.o    testhelp.h  util.o    zmalloc.o
ae.c      asciilogo.h  config.h   dict.o     latency.c  memtest.o  pqsort.o  redis-benchmark.o  redis.o     scripting.c  slowlog.c  t_hash.c  util.o
ae_epoll.c  bio.c      crc16.c   endianconv.c  latency.h  multi.c    pubsub.o  redis.c         redis-sentinel  scripting.o  slowlog.h  t_hash.o  valgrind.sup
ae_evport.c  bio.h     crc16.o   endianconv.h  latency.o  multi.o    pubsub.o  redis-check-aof  redis-server  sds.c     solarisfixes.h  t_list.c  version.h
ae.h       bio.o     crc16.o   endianconv.o  lzf.c.c   multi.o    rand.o    redis-check-aof.c  redis-trib.rb  sds.h     sort.c     t_list.o  zipmap.h
ae_kqueue.c  bitops.c  crc64.c   endianconv.o  lzf.d.c   networking.c  rand.o    redis-check-aof.o  release.c     sds.o     sort.o     t_set.c   zipmap.o
ae.o       bitops.o  crc64.h   fmacros.h    lzf.d.o   notify.c   rdb.o     redis-check-dump  release.h     sentinel.c  sparkline.c  t_string.c  zipmap.c
ae_select.c  blocked.c  crc64.o   help.h       lzf.h     notify.o   rdb.h     redis-check-dump.o  replication.c  sentinel.o  setproctitle.c  t_string.o  zipmap.h
anet.c     blocked.o  db.o      hyperloglog.c  lzfP.h    object.c   rdb.o     replication.o     setproctitle.o  sparkline.o  t_zset.c   zipmap.o
anet.h     cluster.c  db.o      hyperloglog.o  lzfP.h    object.c   rdb.o     replication.o     setproctitle.o  sparkline.o  t_zset.c   zipmap.o
```

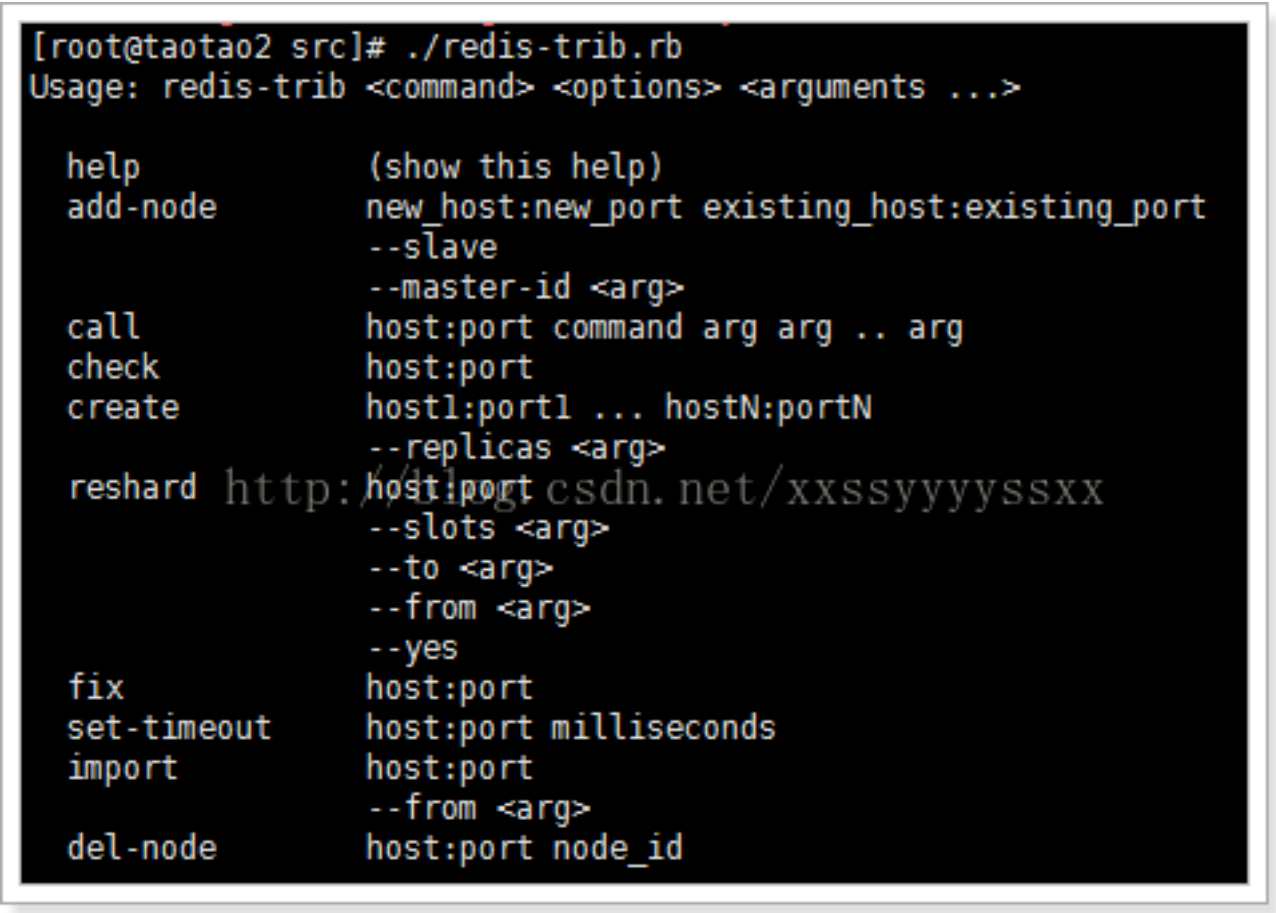
执行命令：

```
./redis-trib.rb create --replicas 0 192.168.56.102:6379 192.168.56.102:6380  
192.168.56.102:6381
```

--replicas 0：指定了从数据的数量为0

注意：这里不能使用127.0.0.1，否则在Jedis客户端使用时无法连接到！

redis-trib用法：



```
[root@taotao2 src]# ./redis-trib.rb  
Usage: redis-trib <command> <options> <arguments ...>  
  
help          (show this help)  
add-node      new_host:new_port existing_host:existing_port  
              --slave  
              --master-id <arg>  
call          host:port command arg arg .. arg  
check         host:port  
create        host1:port1 ... hostN:portN  
              --replicas <arg>  
reshard       http://blog.csdn.net/xxssyyyysxx  
              --slots <arg>  
              --to <arg>  
              --from <arg>  
              --yes  
fix           host:port  
set-timeout   host:port milliseconds  
import        host:port  
              --from <arg>  
del-node      host:port node_id
```

```
[root@taotao2 src]# ./redis-trib.rb create --replicas 0 127.0.0.1:6379 127.0.0.1:6380 127.0.0.1:6381
>>> Creating cluster
Connecting to node 127.0.0.1:6379: OK
Connecting to node 127.0.0.1:6380: OK
Connecting to node 127.0.0.1:6381: OK
>>> Performing hash slots allocation on 3 nodes...
Using 3 masters:
127.0.0.1:6379
127.0.0.1:6380
127.0.0.1:6381
M: 69ab799e215371510cd1d43b14eeda72e688d612 127.0.0.1:6379
slots:0-5460 (5461 slots) master
M: d05b95c38868013dd89172032dec3530ccd1f0eb 127.0.0.1:6380
slots:5461-10922 (5462 slots) master
M: e7753ee684626fd39fb9faf34cf532869e848de4 127.0.0.1:6381
slots:10923-16383 (5461 slots) master
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.
>>> Performing Cluster Check (using node 127.0.0.1:6379)
M: 69ab799e215371510cd1d43b14eeda72e688d612 127.0.0.1:6379
slots:0-5460 (5461 slots) master
M: d05b95c38868013dd89172032dec3530ccd1f0eb 127.0.0.1:6380
slots:5461-10922 (5462 slots) master
M: e7753ee684626fd39fb9faf34cf532869e848de4 127.0.0.1:6381
slots:10923-16383 (5461 slots) master
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@taotao2 src]#
```

确认信息

集群以及槽信息

4.3.3. 测试

```
[root@taotao2 src]# redis-cli
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> set abc 123
(error) MOVED 7638 127.0.0.1:6380
127.0.0.1:6379> █
```

什么情况? ? (error) MOVED 7638 127.0.0.1:6380

因为abc的hash槽信息是在6380上，现在使用redis-cli连接的6379，无法完成set操作，需要客户端跟踪重定向。

redis-cli -c

```
[root@taotao2 src]# redis-cli -c
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set abc 123
-> Redirected to slot [7638] located at 127.0.0.1:6380
OK
127.0.0.1:6380> █
```

看到由6379跳转到了6380，然后再进入6379看能否get到数据

```
[root@taotao2 src]# redis-cli -c
127.0.0.1:6379> get abc
-> Redirected to slot [7638] located at 127.0.0.1:6380
"123"
127.0.0.1:6380> █
```

还是被重定向到了6380，不过已经可以获取到数据了。

4.4. 使用Jedis连接到集群

添加依赖，要注意jedis的版本为2.7.2

```
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>2.7.2</version>
  <type>jar</type>
  <scope>compile</scope>
</dependency>
```


Jedis Cluster

Redis cluster [specification](#) (still under development) is implemented

```
Set<HostAndPort> jedisClusterNodes = new HashSet<HostAndPort>();  
//Jedis Cluster will attempt to discover cluster nodes automatically  
jedisClusterNodes.add(new HostAndPort("127.0.0.1", 7379));  
JedisCluster jc = new JedisCluster(jedisClusterNodes);  
jc.set("foo", "bar");  
String value = jc.get("foo");
```

说明：这里的jc不需要关闭，因为内部已经关闭连接了。

4.5. 插槽的分配

通过cluster nodes命令可以查看当前集群的信息：

```
192.168.56.102:6381> cluster nodes  
4a9b8886ba5261e82597f5590fdb49ea47c4c6c 192.168.56.102:6380 master - 0 1433550616849 2 connected 5461-10922  
9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379 master - 0 1433550615337 1 connected 0-5460  
b22a802b9025b7460f81cfc72f5967cfcba390a5 192.168.56.102:6381 myself master - 0 0 3 connected 10923-16383  
192.168.56.102:6381>
```

该信息反映出了集群中的每个节点的id、身份、连接数、插槽数等。

当我们执行set abc 123命令时，redis是如何将数据保存到集群中的呢？执行步骤：

- 1、接收命令set abc 123
- 2、通过key (abc) 计算出插槽值，然后根据插槽值找到对应的节点。（abc的插槽值为：7638）
- 3、重定向到该节点执行命令

整个Redis提供了16384个插槽，也就是说集群中的每个节点分得的插槽数总和为16384。

./redis-trib.rb 脚本实现了是将16384个插槽平均分配给了N个节点。

注意：如果插槽数有部分是没有指定到节点的，那么这部分插槽所对应的key将不能使用。

4.6. 插槽和key的关系

计算key的插槽值：

key的有效部分使用CRC16[算法](#)计算出哈希值，再将哈希值对16384取余，得到插槽值。

什么是有效部分？

1、如果key中包含了{符号，且在{符号后存在}符号，并且{和}之间至少有一个字符，则有效部分是指{和}之间的部分；

a) key={hello}_taotao的有效部分是hello

2、如果不满足上一条情况，整个key都是有效部分；

a) key=hello_taotao的有效部分是全部

4.7. 新增集群节点

再开启一个实例的端口为6382

```
[root@taotao2 6382]# ps -ef|grep redis
root      1791      1  0  08:21 ?        00:00:01 redis-server *:6379 [cluster]
root      1796      1  0  08:21 ?        00:00:02 redis-server *:6380 [cluster]
root      1800      1  0  08:21 ?        00:00:01 redis-server *:6381 [cluster]
root      2173      1  0  08:51 ?        00:00:00 redis-server *:6382 [cluster]
root      2177    1476  0  08:51 pts/0    00:00:00 grep redis
[root@taotao2 6382]#
```

执行脚本：

./redis-trib.rb add-node 192.168.56.102:6382 192.168.56.102:6379

```
[root@taotao2 src]# ./redis-trib.rb add-node 192.168.56.102:6382 192.168.56.102:6379
>>> Adding node 192.168.56.102:6382 to cluster 192.168.56.102:6379
Connecting to node 192.168.56.102:6379: OK
Connecting to node 192.168.56.102:6380: OK
Connecting to node 192.168.56.102:6381: OK
>>> Performing Cluster Check (using node 192.168.56.102:6379)
M: 9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379
  slots:0-5460 (5461 slots) master
  0 additional replica(s)
M: 4a9b8886ba5261e82597f5590fcdb49ea47c4c6c 192.168.56.102:6380
  slots:5461-10922 (5462 slots) master
  0 additional replica(s)
M: b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381
  slots:10923-16383 (5461 slots) master
  0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
Connecting to node 192.168.56.102:6382: OK
>>> Send CLUSTER MEET to node 192.168.56.102:6382 to make it join the cluster.
[OK] New node added correctly.
[root@taotao2 src]#
```

已经添加成功！查看集群信息：

```
[root@taotao2 src]# redis-cli
127.0.0.1:6379> cluster nodes
4a9b8886ba5261e82597f5590fcdb49ea47c4c6c 192.168.56.102:6380 master - 0 1433552201445 2 connected 5461-10922
9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379 myself,master - 0 0 1 connected 0-5460
b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381 master - 0 1433552201950 3 connected 10923-16383
82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382 master - 0 1433552200940 0 connected 
127.0.0.1:6379>
```

发现没有插槽数。

接下来需要给6382这个服务分配插槽，将6379的一部分（1000个）插槽分配给6382：

```
[root@taotao2 src]# ./redis-trib.rb reshard 192.168.56.102:6379
Connecting to node 192.168.56.102:6379: OK
Connecting to node 192.168.56.102:6380: OK
Connecting to node 192.168.56.102:6381: OK
Connecting to node 192.168.56.102:6382: OK
>>> Performing Cluster Check (using node 192.168.56.102:6379)
M: 9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379
  slots:0-5460 (5461 slots) master
  0 additional replica(s)
M: 4a9b8886ba5261e82597f5590fcd49ea47c4c6c 192.168.56.102:6380
  slots:5461-10922 (5462 slots) master
  0 additional replica(s)
M: b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381
  slots:10923-16383 (5461 slots) master
  0 additional replica(s)
M: 82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382
  slots:(0 slots) master
  0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
How many slots do you want to move (from 1 to 16384)? 1000
What is the receiving node ID? 82ed0d63cfa6d19956dca833930977a87d6ddf74
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1:all
```

<http://blog.csdn.net/xxssyyyyssxx>

输入要移动的插槽数

输入接收的节点ID

all表示从所有的master重新分配

或者数据要提取slot的master节点id,最后用done结束

```
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1:all
```

Ready to move 1000 slots.

Source nodes:

```
M: 9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379
  slots:0-5460 (5461 slots) master
  0 additional replica(s)
M: 4a9b8886ba5261e82597f5590fcd49ea47c4c6c 192.168.56.102:6380
  slots:5461-10922 (5462 slots) master
  0 additional replica(s)
M: b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381
  slots:10923-16383 (5461 slots) master
  0 additional replica(s)
```

Destination node:

```
M: 82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382
  slots:(0 slots) master
  0 additional replica(s)
```

Resharding plan:

```
Moving slot 5461 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5462 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5463 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5464 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5465 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5466 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5467 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5468 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5469 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5470 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5471 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5472 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
Moving slot 5473 from 4a9b8886ba5261e82597f5590fcd49ea47c4c6c
```

已经计算好的插槽值

查看节点情况:


```
127.0.0.1:6379> cluster nodes
4a9b8886ba5261e82597f5590fcd49ea47c4c6c 192.168.56.102:6380 master - 0 1433553060110 2 connected 5795-10922
9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379 myself,master - 0 0 1 connected 333-5460
b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381 master - 0 1433553060110 3 connected 11256-16383
82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382 master - 0 1433553059102 4 connected 0-332 5461-5794 10923-11255
127.0.0.1:6379>
```

4.8. 删除集群节点

想要删除集群节点中的某一个节点，需要严格执行2步：

1、将这个节点上的所有插槽转移到其他节点上；

a) 假设我们想要删除6380这个节点

b) 执行脚本： `./redis-trib.rb reshard 192.168.56.102:6380`

c) 选择需要转移的插槽的数量，因为6380有5128个，所以转移5128个

```
[root@taotao2 src]# ./redis-trib.rb reshard 192.168.56.102:6380
Connecting to node 192.168.56.102:6380: OK
Connecting to node 192.168.56.102:6379: OK
Connecting to node 192.168.56.102:6382: OK
Connecting to node 192.168.56.102:6381: OK
>>> Performing Cluster Check (using node 192.168.56.102:6380)
M: 4a9b8886ba5261e82597f5590fcd49ea47c4c6c 192.168.56.102:6380
  slots:5795-10922 (5128 slots) master
  0 additional replica(s)
M: 9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379
  slots:333-5460 (5128 slots) master
  0 additional replica(s)
M: 82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382
  slots:0-332,5461-5794,10923-11255 (1000 slots) master
  0 additional replica(s)
M: b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381
  slots:11256-16383 (5128 slots) master
  0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
How many slots do you want to move (from 1 to 16384)? 5128
What is the receiving node ID?
```

d) 输入转移的节点的id，我们转移到6382节

点：82ed0d63cfa6d19956dca833930977a87d6ddf7

e) 输入插槽来源id，也就是6380的id

f) 输入done, 开始转移

```
What is the receiving node ID? 82ed0d63cfa6d19956dca833930977a87d6ddf74
Please enter all the source node IDs.
Type 'all' to use all the nodes as source nodes for the hash slots.
Type 'done' once you entered all the source nodes IDs.
Source node #1:4a9b8886ba5261e82597f5590fcdb49ea47c4c6c
Source node #2:done
```

g) 查看集群信息, 可以看到6380节点已经没有插槽了。

```
[root@taotao2 src]# redis-cli cluster nodes
82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382 master - 0 1433573335036 4 connected 0-2041 5461-12964
b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381 master - 0 1433573336544 3 connected 12965-16383
9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379 myself,master - 0 0 1 connected 2042-5460
4a9b8886ba5261e82597f5590fcdb49ea47c4c6c 192.168.56.102:6380 master - 0 1433573336041 2 connected
[root@taotao2 src]#
```

2、使用redis-trib.rb删除节点

a) ./redis-trib.rb del-node 192.168.56.102:6380

4a9b8886ba5261e82597f5590fcdb49ea47c4c6c

b) del-node host:port node_id

```
[root@taotao2 src]# ./redis-trib.rb del-node 192.168.56.102:6380 4a9b8886ba5261e82597f5590fcdb49ea47c4c6c
>>> Removing node 4a9b8886ba5261e82597f5590fcdb49ea47c4c6c from cluster 192.168.56.102:6380
Connecting to node 192.168.56.102:6380: OK
Connecting to node 192.168.56.102:6379: OK
Connecting to node 192.168.56.102:6382: OK
Connecting to node 192.168.56.102:6381: OK
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.
[root@taotao2 src]#
```

c)

d) 查看集群信息, 可以看到已经没有6380这个节点了。

```
[root@taotao2 src]# redis-cli cluster nodes
82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382 master - 0 1433573572879 4 connected 0-2041 5461-12964
b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381 master - 0 1433573571871 3 connected 12965-16383
9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379 myself,master - 0 0 1 connected 2042-5460
[root@taotao2 src]#
```

4.9. 故障转移

如果集群中的某一节点宕机会出现什么状况? 我们这里假设6381宕机。

```
[root@taotao2 src]# ps -ef|grep redis
root      1456      1  0 14:31 ?        00:00:04 redis-server *:6379 [cluster]
root      1465      1  0 14:31 ?        00:00:04 redis-server *:6381 [cluster]
root      1469      1  0 14:31 ?        00:00:05 redis-server *:6382 [cluster]
root      1589    1430  0 14:58 pts/0    00:00:00 grep redis
[root@taotao2 src]#
[root@taotao2 src]# kill -9 1465
```

```
[root@taotao2 src]# ps -ef|grep redis
root      1456      1  0 14:31 ?        00:00:04 redis-server *:6379 [cluster]
root      1469      1  0 14:31 ?        00:00:05 redis-server *:6382 [cluster]
root      1592    1430  0 14:59 pts/0    00:00:00 grep redis
[root@taotao2 src]#
```

我们尝试连接下集群，并且查看集群信息，发现6381的节点断开连接：

```
[root@taotao2 src]# redis-cli
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> cluster nodes
82ed0d63cfa6d19956dca833930977a87d6ddf74 192.168.56.102:6382 master - 0 1433574010425 4 connected 0-2041 5461-12964
b22a802b9025b7460f81cfc72f5967cfcca390a5 192.168.56.102:6381 master,fail - 1433573958671 1433573957967 3 disconnected 12965-16383
9dec39b8441c55658572b290f413200eaa76691f 192.168.56.102:6379 myself,master - 0 0 1 connected 2042-5460
127.0.0.1:6379>
```

我们尝试执行set命令，结果发现无法执行：

```
127.0.0.1:6379>
127.0.0.1:6379> set abc 1
(error) CLUSTERDOWN The cluster is down
127.0.0.1:6379>
```

什么情况？集群不可用了？？ 这集群也太弱了吧？？

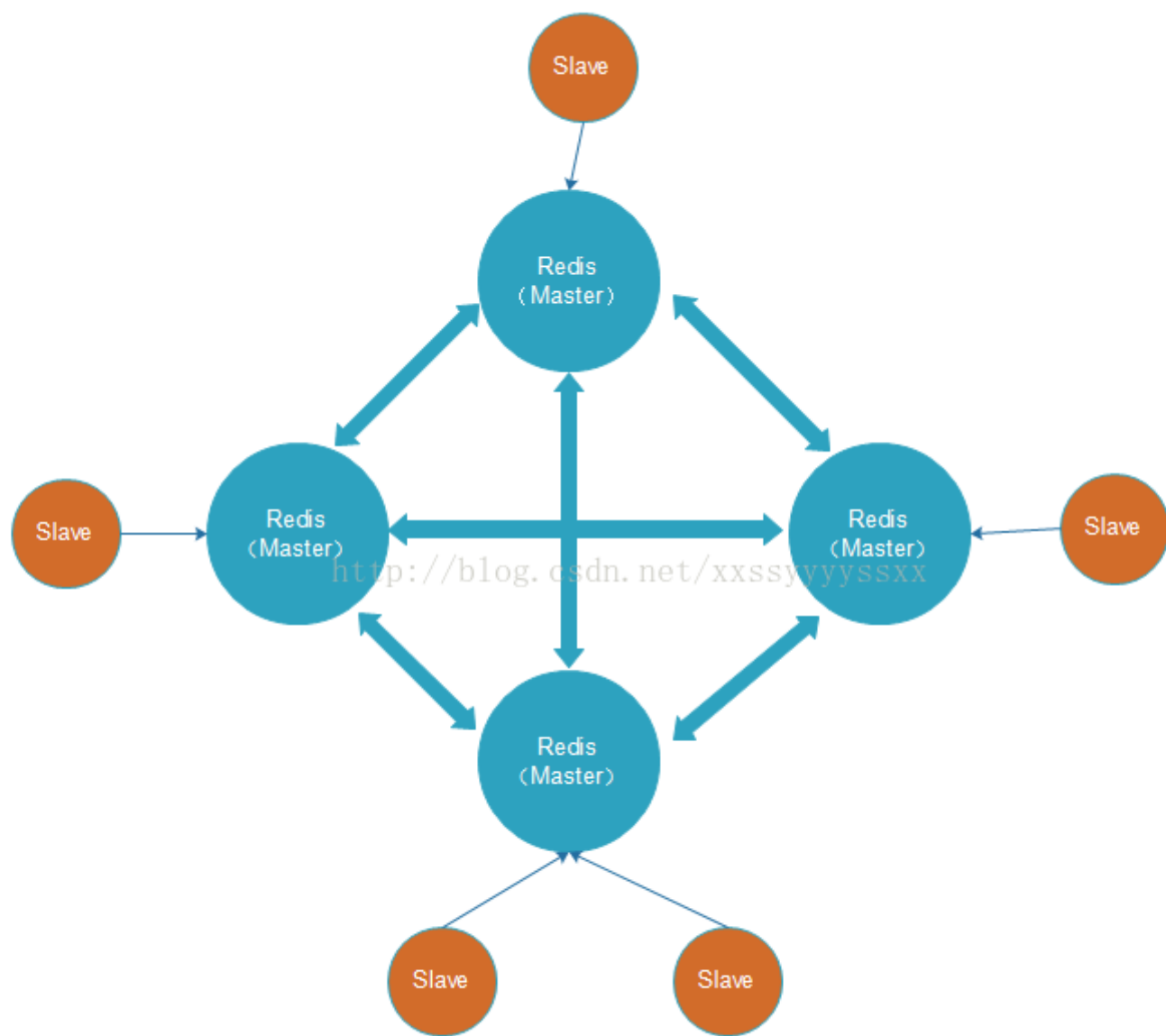
4.9.1. 故障机制

- 1、 集群中的每个节点都会定期的向其它节点发送PING命令，并且通过有没有收到回复判断目标节点是否下线；
- 2、 集群中每一秒就会随机选择5个节点，然后选择其中最久没有响应的节点放PING命令；

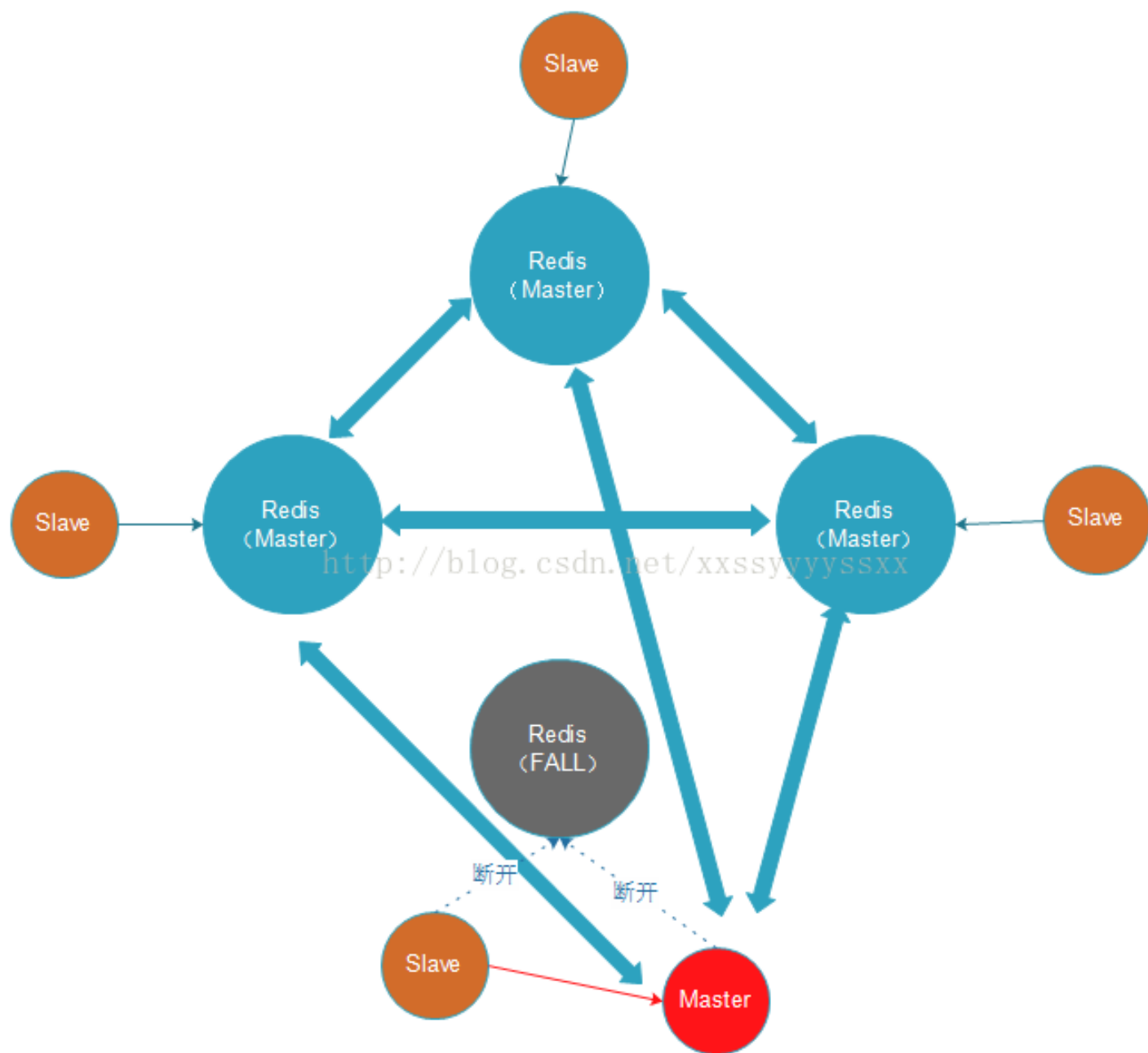
- 3、 如果一定时间内目标节点都没有响应，那么该节点就认为目标节点疑似下线；
- 4、 当集群中的节点超过半数认为该目标节点疑似下线，那么该节点就会被标记为下线；
- 5、 当集群中的任何一个节点下线，就会导致插槽区有空档，不完整，那么该集群将不可用；
- 6、 如何解决上述问题？
 - a) 在Redis集群中可以使用主从模式实现某一个节点的高可用
 - b) 当该节点（master）宕机后，集群会将该节点的从数据库（slave）转变为（master）继续完成集群服务；

4.9.2. 集群中的主从复制架构

架构：



出现故障：



4.9.3. 创建主从集群

需要启动6个redis实例，分别是：

6379（主） 6479（从）

6380（主） 6480（从）

6381（主） 6481（从）

```
drwxr-xr-x. 2 root root 4096 6月 6 15:50 6379
drwxr-xr-x. 2 root root 4096 6月 6 15:50 6380
drwxr-xr-x. 2 root root 4096 6月 6 15:50 6381
drwxr-xr-x. 2 root root 4096 6月 6 15:52 6479
drwxr-xr-x. 2 root root 4096 6月 6 15:47 6480
drwxr-xr-x. 2 root root 4096 6月 6 15:48 6481
```

http://blog.csdn.net/xxssyyyssxx

启动redis实例：

```
cd 6379/ && redis-server ./redis.conf && cd ..
```

```
cd 6380/ && redis-server ./redis.conf && cd ..
```

```
cd 6381/ && redis-server ./redis.conf && cd ..
```

```
cd 6479/ && redis-server ./redis.conf && cd ..
```

```
cd 6480/ && redis-server ./redis.conf && cd ..
```

```
cd 6481/ && redis-server ./redis.conf && cd ..
```

```
[root@taotao2 6481]# ps -ef|grep redis
root      1858      1  0 15:50 ?        00:00:00 redis-server *:6379 [cluster]
root      1862      1  0 15:50 ?        00:00:00 redis-server *:6380 [cluster]
root      1864      1  0 15:50 ?        00:00:00 redis-server *:6381 [cluster]
root      1910      1  0 15:57 ?        00:00:00 redis-server *:6479 [cluster]
root      1935      1  0 15:58 ?        00:00:00 redis-server *:6480 [cluster]
root      1943      1  0 15:59 ?        00:00:00 redis-server *:6481 [cluster]
root      1947    1430  0 15:59 pts/0    00:00:00 grep redis
```

创建集群，指定了从库数量为1，创建顺序为主库（3个）、从库（3个）：

```
./redis-trib.rb create --replicas 1 192.168.56.102:6379 192.168.56.102:6380
```

```
192.168.56.102:6381 192.168.56.102:6479 192.168.56.102:6480 192.168.56.102:6481
```

```
[root@taotao2 6481]# cd /usr/local/src/redis/redis-3.0.1/src/
[root@taotao2 src]# ./redis-trib.rb create --replicas 1 192.168.56.102:6379 192.168.56.102:6380 192.168.56.102:6381 192.168.56.102:6479 192.168.56.102:6480 192.168.56.102:6481
>>> Creating cluster
Connecting to node 192.168.56.102:6379: OK
Connecting to node 192.168.56.102:6380: OK
Connecting to node 192.168.56.102:6381: OK
Connecting to node 192.168.56.102:6479: OK
Connecting to node 192.168.56.102:6480: OK
Connecting to node 192.168.56.102:6481: OK
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
192.168.56.102:6379
192.168.56.102:6380
192.168.56.102:6381
Adding replica 192.168.56.102:6479 to 192.168.56.102:6379
Adding replica 192.168.56.102:6480 to 192.168.56.102:6380
Adding replica 192.168.56.102:6481 to 192.168.56.102:6381
M: e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379
slots:0-5460 (5461 slots) master
M: 4f4ffa2d77f27a94e830eade0601cd207462a08 192.168.56.102:6380
slots:5461-10922 (5462 slots) master
M: 615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381
slots:10923-16383 (5461 slots) master
S: 6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479
replicates e901103802fe41256217971350d56ea9f831e222
S: 3fc265c57ba04c45203e538adf04234548d28884 192.168.56.102:6480
replicates 4f4ffa2d77f27a94e830eade0601cd207462a08
S: dbfabf223345d7cbd68ba9a820fedf8f6a5aec2a 192.168.56.102:6481
replicates 615bd451b7ec656e5088ece832eae87faac3f97a
Can I set the above configuration? (type 'yes' to accept):
```

3个主库

3个从库

```

>>> 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 192.168.56.102:6379)
M: e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379
  slots:0-5460 (5461 slots) master
M: 4f4ffa2d77f27a94e830eaede0601cd207462a08 192.168.56.102:6380
  slots:5461-10922 (5462 slots) master
M: 615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381
  slots:10923-16383 (5461 slots) master
M: 6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479
  slots: (0 slots) master
  replicates e901103802fe41256217971350d56ea9f831e222
M: 3fc265c57ba04c45203e538adf04234548d28884 192.168.56.102:6480
  slots: (0 slots) master
  replicates 4f4ffa2d77f27a94e830eaede0601cd207462a08
M: dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 192.168.56.102:6481
  slots: (0 slots) master
  replicates 615bd451b7ec656e5088ece832eae87faac3f97a
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@taotao2 src]#

```

创建成功！查看集群信息：

```

[root@taotao2 src]# redis-cli cluster nodes
4f4ffa2d77f27a94e830eaede0601cd207462a08 192.168.56.102:6380 master - 0 1433577866269 2 connected 5461-10922
3fc265c57ba04c45203e538adf04234548d28884 192.168.56.102:6480 slave 4f4ffa2d77f27a94e830eaede0601cd207462a08 0 1433577866269 5 connected
6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479 slave e901103802fe41256217971350d56ea9f831e222 0 1433577865261 4 connected
dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 192.168.56.102:6481 slave 615bd451b7ec656e5088ece832eae87faac3f97a 0 1433577865764 6 connected
615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381 master 0 1433577864254 3 connected 10923-16383
e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379 myself,master - 0 0 1 connected 0-5460
[root@taotao2 src]#

```

4.9.4. 测试

```

[root@taotao2 src]# redis-cli -c
127.0.0.1:6379> set abc 123
-> Redirected to slot [7638] located at 192.168.56.102:6380
OK
192.168.56.102:6380> get abc
"123"
192.168.56.102:6380>

```

保存、读取数据OK！

查看下6480的从库数据：

```
[root@taotao2 src]# redis-cli -c -p 6480
127.0.0.1:6480> keys *
1) "abc"
127.0.0.1:6480> get abc
-> Redirected to slot [17638] located at 192.168.56.102:6380
"123"
192.168.56.102:6380> █
```

看到从6480查看数据也是被重定向到6380.

说明集群一切运行OK！

4.9.5. 测试集群中slave节点宕机

我们将6480节点kill掉，查看情况。

```
192.168.56.102:6380>
[root@taotao2 src]# ps -ef|grep redis
root      1858      1  0 15:50 ?        00:00:00 redis-server *:6379 [cluster]
root      1862      1  0 15:50 ?        00:00:00 redis-server *:6380 [cluster]
root      1864      1  0 15:50 ?        00:00:01 redis-server *:6381 [cluster]
root      1910      1  0 15:57 ?        00:00:00 redis-server *:6479 [cluster]
root      1935      1  0 15:58 ?        00:00:00 redis-server *:6480 [cluster]
root      1943      1  0 15:59 ?        00:00:00 redis-server *:6481 [cluster]
root      2031    1430  0 16:13 pts/0    00:00:00 grep redis
[root@taotao2 src]# kill -9 1935
[root@taotao2 src]# ps -ef|grep redis
root      1858      1  0 15:50 ?        00:00:00 redis-server *:6379 [cluster]
root      1862      1  0 15:50 ?        00:00:00 redis-server *:6380 [cluster]
root      1864      1  0 15:50 ?        00:00:01 redis-server *:6381 [cluster]
root      1910      1  0 15:57 ?        00:00:00 redis-server *:6479 [cluster]
root      1943      1  0 15:59 ?        00:00:00 redis-server *:6481 [cluster]
root      2033    1430  0 16:13 pts/0    00:00:00 grep redis
[root@taotao2 src]# █
```

查看集群情况：

```
[root@taotao2 src]# redis-cli -c cluster nodes
4f4ffa2d77f27a94e830eade0601cd207462a08 192.168.56.102:6380 master - 0 1433578465276 2 connected 5461-10922
3fc265c57ba04c45203e538adf04234548d2888 192.168.56.102:6480 slave,fail 4f4ffa2d77f27a94e830eade0601cd207462a08 1433578423043 1433578421433 5 disconnected
6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479 slave e901103802fe41256217971350d56ea9f831e222 0 1433578464265 4 connected
dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 192.168.56.102:6481 slave 615bd451b7ec656e5088ece832eae87faac3f97a 0 1433578465782 6 connected
615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381 master - 0 1433578463759 3 connected 10923-16383
e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379 myself,master - 0 0 1 connected 0-5460
[root@taotao2 src]# █
```


发现6480节点不可用。

那么整个集群可用吗?

```
[root@taotao2 src]# redis-cli
127.0.0.1:6379>
[root@taotao2 src]# redis-cli -c
127.0.0.1:6379> get abc
-> Redirected to slot [7638] located at 192.168.56.102:6380
"123"
192.168.56.102:6380> set abc 456
OK
192.168.56.102:6380> get abc
"456"
192.168.56.102:6380> █
```

发现集群可用，可见从数据库宕机不会影响集群正常服务。

恢复6480服务：

```
[root@taotao2 6480]# redis-cli cluster nodes
4f4ffa2d77f27a94e830eaede0601cd207462a08 192.168.56.102:6380 master - 0 1433578691033 2 connected 5461-10922
3fc265c57ba04c45203e538adf04234548d28884 192.168.56.102:6480 slave 4f4ffa2d77f27a94e830eaede0601cd207462a08 0 1433578690029 5 connected
6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479 slave e901103802fe41256217971350d56ea9f831e222 0 1433578689024 4 connected
dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 192.168.56.102:6481 slave 615bd451b7ec656e5088ece832eae87faac3f97a 0 1433578690029 6 connected
615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381 master - 0 1433578691033 3 connected 10923-16383
e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379 myself,master - 0 0 1 connected 0-5460
[root@taotao2 6480]# █
```

测试6480中的数据：

```
[root@taotao2 6480]# redis-cli -c -p 6480
127.0.0.1:6480> keys *
1) "abc"
127.0.0.1:6480> get abc
-> Redirected to slot [7638] located at 192.168.56.102:6380
"456"
192.168.56.102:6380> █
```

看到已经更新成最新数据。

4.9.6. 测试集群中master宕机

假设6381宕机：

```
[root@taotao2 6480]# ps -ef|grep redis
root      1858      1  0 15:50 ?        00:00:01 redis-server *:6379 [cluster]
root      1862      1  0 15:50 ?        00:00:01 redis-server *:6380 [cluster]
root      1864      1  0 15:50 ?        00:00:01 redis-server *:6381 [cluster]
root      1910      1  0 15:57 ?        00:00:01 redis-server *:6479 [cluster]
root      1943      1  0 15:59 ?        00:00:01 redis-server *:6481 [cluster]
root      2061      1  0 16:17 ?        00:00:00 redis-server *:6480 [cluster]
root      2077    1430  0 16:21 pts/0    00:00:00 grep redis
[root@taotao2 6480]# kill -9 1864
```

查看集群情况：

```
[root@taotao2 6480]# redis-cli cluster nodes
4f4ffa2d77f27a94e830eaede0601cd207462a08 192.168.56.102:6380 master - 0 1433578935733 2 connected 5461-10922
3fc265c57ba04c45203e538adf04234548d28884 192.168.56.102:6480 slave 4f4ffa2d77f27a94e830eaede0601cd207462a08 0 1433578935733 5 connected
6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479 slave e901103802fe41256217971350d56ea9f831e222 0 1433578936238 4 connected
dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 192.168.56.102:6481 master - 0 1433578936740 7 connected 10923-16383
615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381 master, fail - 1433578901177 1433578899466 3 disconnected
e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379 myself,master - 0 0 1 connected 0-5460
[root@taotao2 6480]#
```

发现：

- 1、6381节点失效不可用
- 2、6481节点从slave转换为master

测试集群是否可用：

```
[root@taotao2 6480]# redis-cli -c
127.0.0.1:6379> get abc
-> Redirected to slot [7638] located at 192.168.56.102:6380
"456"
192.168.56.102:6380> set taotao 123
OK
192.168.56.102:6380> get tatao
(nil)
192.168.56.102:6380> get taotao
"123"
192.168.56.102:6380>
```

集群可用。

恢复6381：

```
[root@taotao2 6381]# redis-cli cluster nodes
4f4ffa2d77f27a94e830eaede0601cd207462a08 192.168.56.102:6380 master - 0 1433579135234 2 connected 5461-10922
3fc265c57ba04c45203e538adf04234548d28884 192.168.56.102:6480 slave 4f4ffa2d77f27a94e830eaede0601cd207462a08 0 1433579135737 5 connected
6fd1dd8d6ea33bdee5513b5127b8bbe6a5cc7bf8 192.168.56.102:6479 slave e901103802fe41256217971350d56ea9f831e222 0 1433579134730 4 connected
dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 192.168.56.102:6481 master - 0 1433579136239 7 connected 10923-16383
615bd451b7ec656e5088ece832eae87faac3f97a 192.168.56.102:6381 slave dbfabf223345d7cbd68ba9a829fedf8f6a5aec2a 0 1433579136239 7 connected
e901103802fe41256217971350d56ea9f831e222 192.168.56.102:6379 myself,master - 0 0 1 connected 0-5460
```

发现：

- 1、6381节点可用
- 2、6481依然是主节点
- 3、6381成为6481的从数据库

4.10. 使用集群需要注意的事项

- 1、多键的命令操作（如MGET、MSET），如果每个键都位于同一个节点，则可以正常支持，否则会提示错误。
- 2、集群中的节点只能使用0号数据库，如果执行SELECT切换数据库会提示错误。