1. redis主从同步：

Redis主从复制，当用户往Master端写入数据时，通过Redis Sync机制将数据文件发送至Slave，Slave也会执行相同的操作确保数据一致。同时slave上还可以开启二级slave，三级slave从库。 Redis主从配置非常简单，只需要在Redis从库配置中指定slaveof ip port 即可，IP表示指定主库的ip，port表示redis监听端口。

```
master: 192.168.75.136
slave:  192.168.75.130
```

1.1 主库配置：

不设置密码，从库可以直接使用slaveof ip port连接主库。

```
# 修改配置文件：
以守护进程在后台运行：
daemonize yes
bind 0.0.0.0
```

设置认证密码，从库需要指定master得密码才能完成同步，否则日志会有报错提示：

```
# 修改配置文件：
以守护进程在后台运行：
daemonize yes
# 可选设置认证密码
requirepass "123456"
bind 0.0.0.0
```

1.2 从库配置：

```
# 修改配置文件设置（重启服务器生效）
bind 0.0.0.0
# 指定隶属于谁：
slaveof 192.168.75.135 6379
或者
REPLICAOF 192.168.75.135 6379
```

```
#指定master的认证密码（如果远程服务器设置了密码，则需要认证密码）
masterauth "123456"

# 直接在命令行设置（直接生效，重启服务失效）：
# 指定谁是主库：
slaveof 192.168.75.135 6379
或者
REPLICAOF 192.168.75.135 6379
# 设置master得认证密码：
CONFIG set masterauth 123456
# 取消主从关系，恢复master身份：
SLAVEOF no one
```

1.3 从库查看状态：

```
[root@localhost ~]# redis-cli  -h 192.168.75.130 -a 123456
192.168.75.130:6379> info replication
# Replication
# 角色
role:slave
master_host:192.168.75.135
master_port:6379
# 主库连接状态必须为up，表示成功
master_link_status:up
master_last_io_seconds_ago:1
master_sync_in_progress:0
slave_repl_offset:2354
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
192.168.75.130:6379>
```

1.4 主库查看状态：

```
[root@localhost ~]# redis-cli -h 192.168.75.135 -a 123456
192.168.75.135:6379> info replication
```

```
# Replication
# 角色为master(主库)
role:master
# 当前有一个从库
connected_slaves:1
#能看到从库的ip地址信息
slave0:ip=192.168.75.130,port=6379,state=online,offset=179
63,lag=0
master_repl_offset:18106
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:18105
```

1.5 日志分析:

1.5.1 redis同步过程:

```
Connecting to MASTER 192.168.75.136:6379
MASTER <-> REPLICA sync started
# 开始非阻塞同步
Non blocking connect for SYNC fired the event.
Master replied to PING, replication can continue...
Partial resynchronization not possible (no cached master)
# 全量同步
Full resync from master:
74fb29643971834fdf701beb1708ce9ddc23bdee:0
# 接受数据
MASTER <-> REPLICA sync: receiving 193 bytes from master
# 清空以前缓存得数据
MASTER <-> REPLICA sync: Flushing old data
# 加载数据库到内存
MASTER <-> REPLICA sync: Loading DB in memory
# 完成同步
MASTER <-> REPLICA sync: Finished with success
```

1.5.2 redis主库设置了同步密码，从库没有指定，则报如下错:

```
Unexpected reply to PSYNC from master: -NOAUTH
Authentication required.
Retrying with SYNC...
MASTER aborted replication with an error: NOAUTH
Authentication required.
```

1.6 主从复制相关参数解读：

```
# 指定主库IP和端口：
replicaof  192.168.75.136  6379
# 指定主库得认证密码：
masterauth 123456
# 从库正在复制时，从库可以响应用户读请求，如果设置为no,则返回报错信息。
replica-serve-stale-data yes
# 设置从库为只读
replica-read-only yes
# 启动socket方式复制数据库，master生成rdb文件，不在是先保存到磁盘，然后发给从库，而是直接把rdb发送给从库，减少了磁盘IO
repl-diskless-sync yes
# 配置延时时间，让更多slave加入传输队列，如果复制已经开始，则5秒内，不接受新的slave同步请求
repl-diskless-sync-delay 5
# 指定从库定期检查主库状态，默认10秒
repl-ping-replica-period 10
# 同步超时时间
repl-timeout 60
# 是否禁用tcp-nodelay，yes表示禁用，redis会在写缓存积累到一定量之后一起发送，节省带宽，但是会导致master和slave数据延迟，no，表示启用，redis会立即发送数据包，即使是很小数据，数据同步会比较快，但是消耗更多带宽
repl-disable-tcp-nodelay no
# 设置在同步过程中，写缓冲区得大小，需要考虑到同步的时间和数据的写入速度
repl-backlog-size 1mb
# 设置从库的优先级，在主库宕机后，根据优先级选择slave,值越小，则优先级越高，0,表示不参与竞选，故永远不会被选中。
replica-priority 100
```

1.7 同步策略：

1.7.1 全量同步：

Redis全量复制一般发生在Slave初始化阶段，这时Slave需要将Master上的所有数据都复制一份。具体步骤如下：

> 1）从服务器连接主服务器，发送SYNC命令；
> 2）主服务器接收到SYNC命名后，开始执行BGSAVE命令生成RDB文件并使用缓冲区记录此后执行的所有写命令；
> 3）主服务器BGSAVE执行完后，向所有从服务器发送快照文件，并在发送期间继续记录被执行的写命令；
> 4）从服务器收到快照文件后丢弃所有旧数据，载入收到的快照；
> 5）主服务器快照发送完毕后开始向从服务器发送缓冲区中的写命令；
> 6）从服务器完成对快照的载入，开始接收命令请求，并执行来自主服务器缓冲区的写命令；

完成上面几个步骤后就完成了从服务器数据初始化的所有操作，从服务器此时可以接收来自用户的读请求。

1.7.2 增量同步：

Redis增量复制是指Slave初始化后开始正常工作时主服务器发生的写操作同步到从服务器的过程。 增量复制的过程主要是主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务器接收并执行收到的写命令。

1.7.3 Redis主从同步策略：

主从刚刚连接的时候，进行全量同步；全同步结束后，进行增量同步。当然，如果有需要，slave 在任何时候都可以发起全量同步。redis 策略是，**无论如何，首先会尝试进行增量同步，如不成功，要求重新进行全量同步。**

1.7.4 注意点：

如果master重启，会因为master_replid不一致导致全量同步，当有多个slave时，可能会导致Master IO剧增宕机。

2. redis哨兵模式：

redis主从虽然解决了单点导致的数据丢失问题，但是还是没有解决无缝的故障转移，也就是说在主库宕机后，从库无法自动切换为主库，需要手工去切换，在这一瞬间会对后端数据库造成极大的负载，可能直接导致后端数据宕机。

2.1 哨兵主要作用：

- **监控**：监控redis主库及从库运行状态；
- **通知**：如果redis发生故障转移，可以通过邮件通知管理员；
- **自动故障转移**：一旦发现主库宕机，则在从库中通过选举新的master进行故障转移。

2.2 工作原理：

哨兵(sentinel) 是一个分布式系统，你可以在一个架构中运行多个哨兵(sentinel) 进程，这些进程使用流言协议(gossipprotocols)来接收关于Master是否下线的信息，并使用投票协议(agreement protocols)来决定是否执行自动故障迁移，以及选择哪个Slave作为新的Master。

每个哨兵(sentinel) 会向其它哨兵(sentinel)、master、slave定时发送消息，以确认对方是否”活”着，如果发现对方在指定时间(可配置)内未回应，则暂时认为对方宕机了，这种宕机称为”**主观认为宕机**” Subjective Down,简称sdown)。

若”哨兵群”中的多数sentinel，都报告某一master没响应，系统才认为该master真正宕机，即**客观上认为宕机**，Objective Down,简称odown)，通过一定的vote算法，从剩下的slave节点中，选一台提升为master，**然后自动修改相关配置。**

2.3 哨兵模式配置：

```
master: 192.168.75.135
slave1: 192.168.75.136
slave2: 192.168.75.137
```

2.3.1 配置redis主从：

```
# 配置master，开启网卡监听，(可选设置密码)：
vim /usr/local/redis/6379/6379.conf
bind 0.0.0.0
```

```
# 配置slave1，指定master:
bind 0.0.0.0
replicaof  192.168.75.135 6379

# 配置slave2，指定master:
bind 0.0.0.0
replicaof  192.168.75.135 6379

# 启动master和slave得redis服务:
/etc/init.d/redis_6379 restart
```

2.3.2 查看主从状态:

```
# 在master执行以下指令，均可查看同步信息:
127.0.0.1:6379> role
1) "master"
2) (integer) 322
3) 1) 1) "192.168.75.136"
      2) "6379"
      3) "322"
   2) 1) "192.168.75.137"
      2) "6379"
      3) "322"
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.75.136,port=6379,state=online,offset=784
,lag=1
slave1:ip=192.168.75.137,port=6379,state=online,offset=784
,lag=1
...
```

2.3.3 配置哨兵:

```
# master端配置sentinel:
[root@node5 ~]# cd /usr/src/redis-5.0.5
# 复制哨兵得配置文件:
[root@node5 redis-5.0.5]# cp sentinel.conf
/usr/local/redis/
```

```
# 修改配置文件：
vim /usr/local/redis/sentinel.conf
# 绑定监听IP
bind 0.0.0.0
# 监听端口
port 26379
# 后台运行
daemonize yes
pidfile /var/run/redis-sentinel.pid
# 哨兵得日志文件
logfile "sentinel.log"
# 日志文件存放路径
dir /usr/local/redis/
# 设置初始master以及法定认为下线人数：
sentinel monitor mymaster 192.168.75.135 6379 2
# master主观下线时间,默认30秒，30秒内没有回复pong，则认为下线了
sentinel down-after-milliseconds mymaster 30000
# 指定在故障转移期间，多少个slave向新的master同步得数量，如果slave是提供查询服务，则应该设置小一点更好
sentinel parallel-syncs mymaster 1
# 指定故障转移超时时间，默认为3分钟
sentinel failover-timeout mymaster 180000
# 设置通知脚本，发生故障转移可以向管理员发送通知（可选）
sentinel notification-script mymaster
/usr/local/redis/notify.sh
# 禁止修改脚本，避免脚本重置
sentinel deny-scripts-reconfig yes
---------------
# notify.sh脚本：

#!/bin/bash
# lutixia
######################

TO="1550684538@qq.com"
SUBJECT="redis 发生故障转移"
CONTEXT="redis 发生故障转移"
echo -e $CONTEXT | mailx -s $SUBJECT $TO

# slave端配置sentinel，把配置好的sentinel配置文件直接复制过去，不要把启动后的配置文件发过去了：
```

```
scp /usr/local/redis/sentinel.conf
192.168.75.136:/usr/local/redis/
scp /usr/local/redis/sentinel.conf
192.168.75.137:/usr/local/redis/
```

3. Redis集群介绍：

在前面章节学习中，我们部署了redis主从，解决了redis单点问题，但是没有实现redis状态监控及故障自动切换，于是后来又引入了sentinel（哨兵）解决此问题。但是依然没能解决数据的一个并发读写的问题，那么Redis 集群就是来解决此问题的，它是一个提供在多个Redis节点间共享数据的程序集。

Redis 集群通过分区来提供一定程度的可用性，在实际环境中当某个节点宕机或者不可达的情况下可以继续处理命令。

3.1 Redis 集群的优势:

- 自动分割数据到不同的节点上。
- 整个集群的部分节点失败或者不可达的情况下能够继续处理命令。

要让集群正常运作至少需要三个主节点，为了**实现主节点的高可用**， 强烈建议使用**六个节点**： 其中三个为主节点， 而其余三个则是各个主节点的从节点。

3.2 Redis集群的工作原理：

Redis 集群没有使用一致性hash, 而是引入了 哈希槽的概念。

Redis 集群有16384个哈希槽,每个key通过CRC16校验后对16384取模来决定放置哪个槽。集群的每个节点负责一部分hash槽,举个例子,比如当前集群有3个节点,那么:

- 节点 A 包含 0 到 5460号哈希槽.
- 节点 B 包含5461到10922号哈希槽.
- 节点 C 包含10923到16383号哈希槽.

这种结构很容易添加或者删除节点，比如如果我想新添加个节点D，我需要从节点 A, B, C中得部分槽到D上。如果我想移除节点A，需要将A中的槽移到B和C节点上，然后将没有任何槽的A节点从集群中移除即可。由于从一个节点将哈希槽移动到另一个节点并不会停止服务，所以无论添加删除

或者改变某个节点的哈希槽的数量都不会造成集群不可用的状态。

Redis 集群的主从复制模型：

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有N-1个复制品。

在上面具有A，B，C三个节点的集群，在没有复制模型的情况下，如果节点B失败了，那么整个集群就会以为缺少5461到10922这个范围的槽而不可用。

然而如果在集群创建的时候（或者过一段时间）我们为每个节点添加一个从节点A1，B1，C1,那么整个集群便有三个master节点和三个slave节点组成，这样在节点B失败后，集群便会选举B1为新的主节点继续服务，整个集群便不会因为槽找不到而不可用了，不过当B和B1 都失败后，集群是不可用的。

4. redis cluster部署：

4.1 项目环境：

```
centos 7.4
redis 5.0.5
```

4.2 创建redis实例：

首先可以选择创建一个新的`redis-cluster`目录，并在此目录中创建六个以端口号为名字的子目录，稍后我们在将每个目录中运行一个 Redis 实例,命令如下:

```
# 创建redis-cluster目录：
mkdir /usr/local/redis-cluster/
cd /usr/local/redis-cluster/
# 创建六个实例目录：
mkdir 7000 7001 7002 7003 7004 7005
# 用初始化脚本，快速创建实例配置文件，日志目录：
/usr/src/redis-stable/utils/install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis
server
```

```
Selected config:
Port            : 7000
Config file     : /usr/local/redis-cluster/7000/7000.conf
Log file        : /usr/local/redis-cluster/7000/7000.log
Data dir        : /usr/local/redis-cluster/7000/
Executable      : /usr/local/redis-cluster/redis-server
Cli Executable : /usr/local/redis/bin/redis-cli
Copied /tmp/7000.conf => /etc/init.d/redis_7000
Installing service...
Successfully added to chkconfig!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!

# 用相同方式连接创建余下5个redis实例。
```

4.3 开启redis集群模式：

```
# 在每个redis实例的配置文件中，开启如下参数：
cluster-enabled yes
cluster-config-file nodes-7000.conf
cluster-node-timeout 5000
appendonly yes
```

4.4 启动redis实例：

```
for i in `seq 0 5`;do /etc/init.d/redis_700$i start;sleep
2;done
```

4.5 搭建集群:

现在已经有了六个正在运行中的 Redis 实例， 接下来就可以直接使用这些实例来创建集群， 并为每个节点编写配置文件。

如果使用的是Redis 5的版本，可以直接使用redis-cli命令创建新集群，检查或重新硬化现有集群等等。

对于Redis版本3或4，使用redis-trib.rb创建集群，它是一个Ruby程序。可以在Redis源代码的src目录下找到它。

```
# 创建集群：
```

```
/usr/local/redis/bin/redis-cli --cluster create
127.0.0.1:7000 127.0.0.1:7001 \
127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004
127.0.0.1:7005 \
--cluster-replicas 1

--cluster-replicas 1 表示为集群中的每个主节点创建一个从节点。

其中指定了6个redis的ip：port,3个master3个slave。
以上命令回车后，当我们输入yes，redis-cli就会将这份配置应用到集群当中,让各个节点开始互相通讯,最后可以得到如下信息：
...
[OK] All 16384 slots covered

# 关闭集群：
```

4.6 查看集群信息或状态：

```
redis-cli -p 7000 cluster info
```

4.7 查看集群节点：

```
redis-cli -p 7000 cluster nodes
```

4.8 添加新的节点：

```
# 重新初始化一个节点：
[root@node5 ~]# /usr/src/redis-
5.0.5/utils/install_server.sh
Welcome to the redis service installer
This script will help you easily set up a running redis
server

Please select the redis port for this instance: [6379]
7006
Please select the redis config file name
[/etc/redis/7006.conf] /usr/local/redis-
cluster/7006/7006.conf
```

```
Please select the redis log file name
[/var/log/redis_7006.log] /usr/local/redis-
cluster/7006/7006.log
Please select the data directory for this instance
[/var/lib/redis/7006] /usr/local/redis-cluster/7006/
Please select the redis executable path
[/usr/local/redis/bin/redis-server]
Selected config:
Port           : 7006
Config file    : /usr/local/redis-cluster/7006/7006.conf
Log file       : /usr/local/redis-cluster/7006/7006.log
Data dir       : /usr/local/redis-cluster/7006/
Executable     : /usr/local/redis/bin/redis-server
Cli Executable : /usr/local/redis/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/7006.conf => /etc/init.d/redis_7006
Installing service...
Successfully added to chkconfig!
Successfully added to runlevels 345!
Starting Redis server...
Installation successful!
# 添加节点
redis-cli  --cluster add-node 127.0.0.1:7006
127.0.0.1:7000
# 看到一下信息，表示节点添加成功：
...
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
>>> Send CLUSTER MEET to node 127.0.0.1:7006 to make it
join the cluster.
[OK] New node added correctly.
```

4.9 重新分片：

虽然节点添加成功，但是还没有分配散列槽，需要重新分片，就是将其他节点上的部分散列槽移动到该节点上。

```
# 可以看到虽然7006新节点添加成功，但是没有分配散列槽：
[root@node5 ~]# redis-cli --cluster check 127.0.0.1:7000
127.0.0.1:7001 (aa04c256...) -> 1 keys | 5462 slots | 1
slaves.
127.0.0.1:7002 (f6838a0e...) -> 0 keys | 5461 slots | 1
slaves.
127.0.0.1:7006 (44124bd9...) -> 0 keys | 0 slots | 0
slaves.
127.0.0.1:7004 (e4f85202...) -> 0 keys | 5461 slots | 1
slaves.
...
# 重新分配，只需指定一个节点，redis-cli将自动找到其他节点：
redis-cli --cluster reshard 127.0.0.1:7000
```

- 首先指定分配多少散列槽：

```
# 重新分配多少个散列槽，这里尝试重新设置2048个散列槽：
How many slots do you want to move (from 1 to 16384)? 2048
```

- 指定接收哈希槽的节点ID：

```
# 指定接受的redis节点ID，注意不是指定ip地址：
How many slots do you want to move (from 1 to 16384)? 2048
What is the receiving node ID?
44124bd95e8e54aedde9ab29ee420d11387000fa
```

- 指定从哪些节点获取散列槽：

```
How many slots do you want to move (from 1 to 16384)? 2048
What is the receiving node ID?
44124bd95e8e54aedde9ab29ee420d11387000fa
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

# all:    表示自动在所有节点进行分配。
# done:   表示指定节点结束，一般用在移除节点时，指定完Source node
#1: 节点ID，再输入done，表示输入结束。
```

- 是否确定分配：

```
    Moving slot 678 from
e4f852026e0f7fa6d6f452de98b73dd97577a999
    Moving slot 679 from
e4f852026e0f7fa6d6f452de98b73dd97577a999
    Moving slot 680 from
e4f852026e0f7fa6d6f452de98b73dd97577a999
    Moving slot 681 from
e4f852026e0f7fa6d6f452de98b73dd97577a999
Do you want to proceed with the proposed reshard plan
(yes/no)? yes
```

在最终确认之后，redis-cli开始执行重新分配：

```
# 下面这个片段可以看到redis正在从7004上移动散列槽到7006上：
...
Moving slot 674 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 675 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 676 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 677 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 678 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 679 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 680 from 127.0.0.1:7004 to 127.0.0.1:7006:
Moving slot 681 from 127.0.0.1:7004 to 127.0.0.1:7006:
```

重新分片正在进行中时，程序不受影响地运行。

检查集群的运行状态：

```
# 可以看到散列槽已经分配了
[root@node5 ~]# redis-cli --cluster check 127.0.0.1:7000
127.0.0.1:7001 (aa04c256...) -> 0 keys | 4779 slots | 1
slaves.
127.0.0.1:7002 (f6838a0e...) -> 0 keys | 4779 slots | 1
slaves.
127.0.0.1:7006 (44124bd9...) -> 1 keys | 2047 slots | 0
slaves.
127.0.0.1:7004 (e4f85202...) -> 0 keys | 4779 slots | 1
slaves.
```

4.10 免交互重新分片脚本：

可以自动执行重新分片，而无需以交互方式手动输入参数。这可以使用如下命令行：

```
redis-cli reshard <host>:<port> --cluster-from <node-id> --cluster-to <node-id> --cluster-slots <number of slots> --cluster-yes
```

4.11 删除节点：

删除节点与添加节点类似，就是一定要注意，删除节点之前一定要先将数据移到其他节点，不能直接删除，不过也不用担心，即使有数据，不小心执行了删除节点指令，也会报有数据存在，不可以删除的错误。

```
# 通过重新分片，将散列槽移到其他节点：
redis-cli  --cluster reshard 127.0.0.1:7000

How many slots do you want to move (from 1 to 16384)? 2047
# 用来接受散列槽的节点，可以任意指定：
What is the receiving node ID?
e4f852026e0f7fa6d6f452de98b73dd97577a999
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.
  # 7000节点的id(意思是把该节点的散列槽分配给其他节点)：
Source node #1: 44124bd95e8e54aedde9ab29ee420d11387000fa
Source node #2: done

# 查看状态：
[root@node5 ~]# redis-cli --cluster check 127.0.0.1:7000
127.0.0.1:7001 (aa04c256...) -> 0 keys | 4779 slots | 1 slaves.
127.0.0.1:7002 (f6838a0e...) -> 0 keys | 4779 slots | 1 slaves.
# 已经没有了散列槽：
127.0.0.1:7006 (44124bd9...) -> 0 keys | 0 slots | 0 slaves.
127.0.0.1:7004 (e4f85202...) -> 1 keys | 6826 slots | 1 slaves.
```

```
# 确定没问题了，执行下面命令，可以实现删除节点：
[root@node5 ~]# redis-cli --cluster del-node
127.0.0.1:7006 44124bd95e8e54aedde9ab29ee420d11387000fa
>>> Removing node 44124bd95e8e54aedde9ab29ee420d11387000fa
from cluster 127.0.0.1:7006
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.
```

4.11 报错解决：

```
# 在创建集群时，报错：
[ERR] Node 127.0.0.1:7000 is not empty. Either the node
already knows other nodes
(check with CLUSTER NODES) or contains some key in
database 0.
```

可能是有以前的数据存在，可以登录到redis服务器端，执行flushdb刷新。重新创建即可。
如果重新创建时，依然报错，可以修复下集群：

```
redis-cli --cluster fix  127.0.0.1:7000
```

```
# 查询或者创建键值报错：
127.0.0.1:7001> get hello
(error) MOVED 866 127.0.0.1:7003
```

解决办法：
启动时使用-c参数来启动集群模式，命令如下：

```
 /usr/local/cluster-test/redis-cli -c -p 7001
 127.0.0.1:7001> get hello
-> Redirected to slot [866] located at 127.0.0.1:7003
"world"
```

###