Redis集群的原理和搭建

阅读 9024 收藏 128 2017-07-10
原文链接：www.jianshu.com

前言

Redis 是我们目前大规模使用的缓存中间件，由于它强大高效而又便捷的功能，得到了广泛的使用。单节点的Redis已经就达到了很高的性能，为了提高可用性我们可以使用Redis集群。本文参考了Rdis的官方文档和使用Redis官方提供的Redis Cluster工具搭建Rdis集群。

注意： Redis的版本要在3.0以上,截止今天，Redis的版本是3.2.9，本教程也使用3.2.9作为教程
<!--more-->

Redis集群的概念

介绍

Redis 集群是一个可以在多个 Redis 节点之间进行数据共享的设施（installation）。

的错误。

Redis 集群通过分区（partition）来提供一定程度的可用性（availability）： 即使集群中有一部分节点失效或者无法进行通讯， 集群也可以继续处理命令请求。

Redis 集群提供了以下两个好处：

- 将数据自动切分（split）到多个节点的能力。
- 当集群中的一部分节点失效或者无法进行通讯时， 仍然可以继续处理命令请求的能力。

数据分片

Redis 集群使用数据分片（sharding）而非一致性哈希（consistency hashing）来实现： 一个 Redis 集群包含 16384 个哈希槽（hash slot）， 数据库中的每个键都属于这 16384 个哈希槽的其中一个， 集群使用公式 CRC16(key) % 16384 来计算键 key 属于哪个槽， 其中 CRC16(key) 语句用于计算键 key 的 CRC16 校验和 。

集群中的每个节点负责处理一部分哈希槽。 举个例子， 一个集群可以有三个哈希槽， 其中：

- 节点 A 负责处理 0 号至 5500 号哈希槽。
- 节点 B 负责处理 5501 号至 11000 号哈希槽。
- 节点 C 负责处理 11001 号至 16384 号哈希槽。

这种将哈希槽分布到不同节点的做法使得用户可以很容易地向集群中添加或者删除节点。 比如说：我现在想设置一个key,叫 `my_name`：

按照Redis Cluster的哈希槽算法， `CRC16('my_name')%16384 = 2412` 那么这个key就被分配到了节点A上

同样的，当我连接(A,B,C)的任意一个节点想获取 `my_name` 这个key,都会转到节点A上

再比如

如果用户将新节点 D 添加到集群中， 那么集群只需要将节点 A 、B 、 C 中的某些槽移动到节点 D 就可以了。

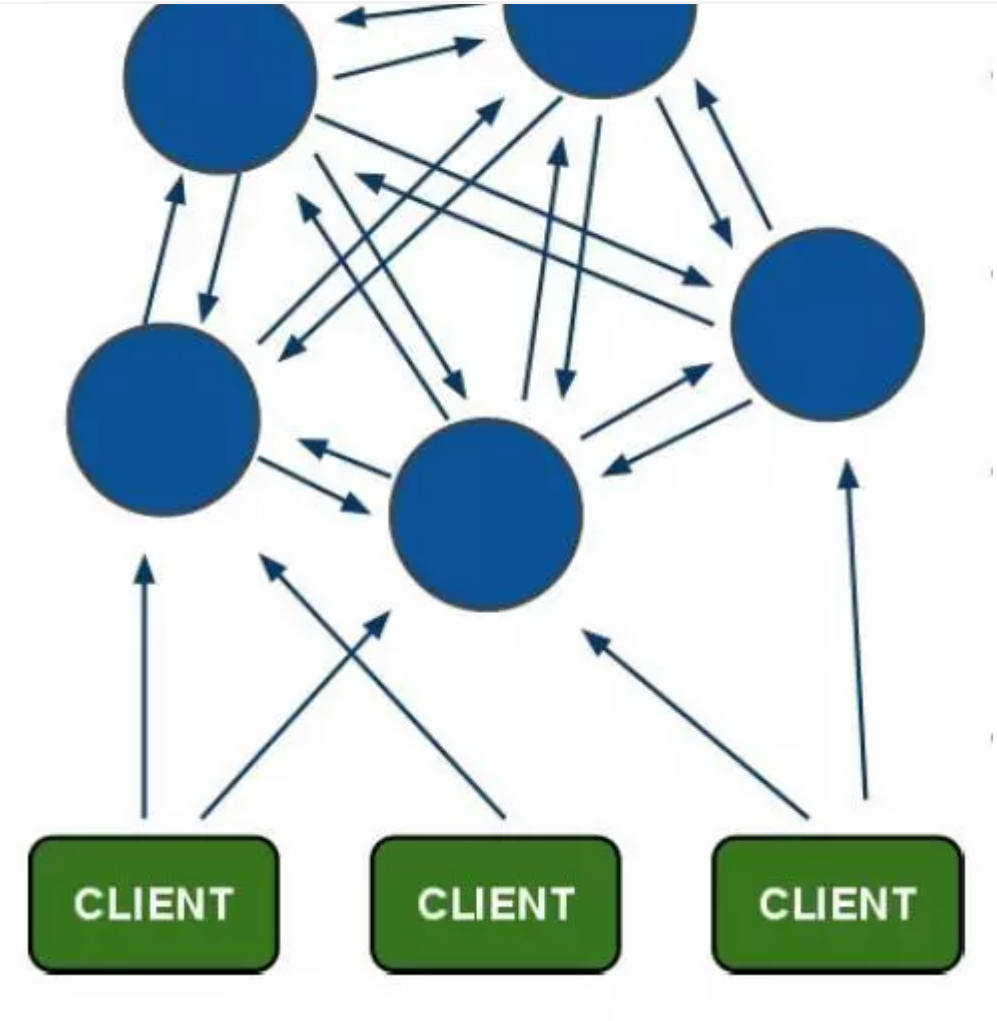
增加一个D节点的结果可能如下：

- 节点A覆盖1365-5460
- 节点B覆盖6827-10922
- 节点C覆盖12288-16383
- 节点D覆盖0-1364,5461-6826,10923-1228

与此类似， 如果用户要从集群中移除节点 A ， 那么集群只需要将节点 A 中的所有哈希槽移动到节点 B 和节点 C ， 然后再移除空白（不包含任何哈希槽）的节点 A 就可以了。

因为将一个哈希槽从一个节点移动到另一个节点不会造成节点阻塞， 所以无论是添加新节点还是移除已存在节点， 又或者改变某个节点包含的哈希槽数量， 都不会造成集群下线。

所以,Redis Cluster的模型大概是这样的形状



082248582749859.jpg

主从复制模型

制品（replica）， 其中一个复制品为主节点（master）， 而其余的 N-1 个复制品为从节点（slave）。

在之前列举的节点 A 、B 、C 的例子中， 如果节点 B 下线了， 那么集群将无法正常运行， 因为集群找不到节点来处理 5501 号至 11000号的哈希槽。

另一方面， 假如在创建集群的时候（或者至少在节点 B 下线之前）， 我们为主节点 B 添加了从节点 B1， 那么当主节点 B 下线的时候， 集群就会将 B1 设置为新的主节点， 并让它代替下线的主节点 B， 继续处理 5501 号至 11000 号的哈希槽， 这样集群就不会因为主节点 B 的下线而无法正常运作了。

不过如果节点 B 和 B1 都下线的话， Redis 集群还是会停止运作。

Redis一致性保证

Redis 并不能保证数据的强一致性. 这意味这在实际中集群在特定的条件下可能会丢失写操作：第一个原因是因为集群是用了异步复制. 写操作过程:

1. 客户端向主节点B写入一条命令.
2. 主节点B向客户端回复命令状态.
3. 主节点将写操作复制给他得从节点 B1, B2 和 B3

主节点对命令的复制工作发生在返回命令回复之后， 因为如果每次处理命令请求都需要等待复制操作完成的话， 那么主节点处理命令请求的速度将极大地降低 —— 我们必须在性能和一致性之间做出权衡。 注意：Redis 集群可能会在将来提供同步写的方法。 Redis 集群另外一种可能会丢失命令

、B1 、C1 为A，B，C的从节点， 还有一个客户端 Z1 假设集群中发生网络分区，那么集群可能会分为两方，大部分的一方包含节点 A 、C 、A1 、B1 和 C1 ，小部分的一方则包含节点 B 和客户端 Z1 .

Z1仍然能够向主节点B中写入, 如果网络分区发生时间较短,那么集群将会继续正常运作,如果分区的时间足够让大部分的一方将B1选举为新的master，那么Z1写入B中得数据便丢失了.

注意， 在网络分裂出现期间， 客户端 Z1 可以向主节点 B 发送写命令的最大时间是有限制的， 这一时间限制称为节点超时时间（node timeout）， 是 Redis 集群的一个重要的配置选项

搭建Redis集群

要让集群正常工作至少需要3个主节点，在这里我们要创建6个redis节点，其中三个为主节点，三个为从节点，对应的redis节点的ip和端口对应关系如下（为了简单演示都在同一台机器上面）

```
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
```

下载安装包

```
wget http://download.redis.io/releases/redis-3.2.9.tar.gz
```

解压安装

```
tar zxvf redis-3.2.9.tar.gz
cd redis-3.2.9
make && make PREFIX=/usr/local/redis install
```

这里如果失败的自行yum安装gcc和tcl

```
  yum install gcc
  yum install tcl
```

创建目录

```
cd /usr/local/redis
mkdir cluster
cd cluster
mkdir 7000 7001 7002 7003 7004 7005
```

将redis目录下的配置文件复制到对应端口文件夹下,6个文件夹都要复制一份

```
cp redis-3.2.9/redis.conf /usr/local/redis/cluster/7000
```

修改配置文件`redis.conf`，将下面的选项修改

```
# 端口号
port 7000
# 后台启动
daemonize yes
# 开启集群
cluster-enabled yes
#集群节点配置文件
cluster-config-file nodes-7000.conf
# 集群连接超时时间
cluster-node-timeout 5000
# 进程pid的文件位置
pidfile /var/run/redis-7000.pid
# 开启aof
appendonly yes
# aof文件路径
appendfilename "appendonly-7005.aof"
# rdb文件路径
dbfilename dump-7000.rdb
```

6个配置文件安装对应的端口分别修改配置文件

在 `/usr/local/redis` 目录下创建一个start.sh

```
#!/bin/bash
bin/redis-server cluster/7000/redis.conf
bin/redis-server cluster/7001/redis.conf
bin/redis-server cluster/7002/redis.conf
bin/redis-server cluster/7003/redis.conf
bin/redis-server cluster/7004/redis.conf
bin/redis-server cluster/7005/redis.conf
```

这个时候我们查看一下进程看启动情况

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

进程状态如下：

```
root      1731     1  1 18:21 ?        00:00:49 bin/redis-server *:7000 [cluster]
root      1733     1  0 18:21 ?        00:00:29 bin/redis-server *:7001 [cluster]
root      1735     1  0 18:21 ?        00:00:08 bin/redis-server *:7002 [cluster]
root      1743     1  0 18:21 ?        00:00:26 bin/redis-server *:7003 [cluster]
root      1745     1  0 18:21 ?        00:00:13 bin/redis-server *:7004 [cluster]
root      1749     1  0 18:21 ?        00:00:08 bin/redis-server *:7005 [cluster]
```

有6个redis进程在开启，说明我们的redis就启动成功了

这里我们只是开启了6个redis进程而已，它们都还只是独立的状态，还没有组成集群这里我们使用官方提供的工具redis-trib，不过这个工具是用ruby写的，要先安装ruby的环境

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

执行，报错

```
[root@centos]# redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:
/usr/lib/ruby/site_ruby/1.8/rubygems/custom_require.rb:31:in `gem_original_require': no s
    from /usr/lib/ruby/site_ruby/1.8/rubygems/custom_require.rb:31:in `require'
    from /usr/local/bin/redis-trib.rb:25
[root@centos]#
```

原来是ruby和redis的连接没安装好安装gem-redis

```
gem install redis
```

安装到这里的时候突然卡住很久不动，网上搜了下，这里需要翻墙或者换镜像

```
gem source -a https://gems.ruby-china.org
```

这里可以将镜像换成ruby-china的镜像，不过我好像更换失败，最终还是翻墙下载了

```
1 gem installed
Installing ri documentation for redis-3.2.1...
Installing RDoc documentation for redis-3.2.1...
```

等下载好后我们就可以使用了

```
[root@centos]# gem install redis
Successfully installed redis-3.2.1
1 gem installed
Installing ri documentation for redis-3.2.1...
Installing RDoc documentation for redis-3.2.1...
```

将redis-3.2.9的src目录下的trib复制到相应文件夹

```
cp redis-3.2.9/src/redis-trib.rb /usr/local/redis/bin/redis-trib
```

创建集群：

```
redis-trib create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:700
```

命令的意义如下：

- 给定 redis-trib.rb 程序的命令是 create ， 这表示我们希望创建一个新的集群。

集群。

简单来说，以上的命令的意思就是让redis-trib程序帮我们创建三个主节点和三个从节点的集群

接着， redis-trib 会打印出一份预想中的配置给你看， 如果你觉得没问题的话， 就可以输入 yes ， redis-trib 就会将这份配置应用到集群当中：

```
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
127.0.0.1:7000
127.0.0.1:7001
127.0.0.1:7002
Adding replica 127.0.0.1:7003 to 127.0.0.1:7000
Adding replica 127.0.0.1:7004 to 127.0.0.1:7001
Adding replica 127.0.0.1:7005 to 127.0.0.1:7002
M: bdcddddd3d78a866b44b68c7ae0e5ccf875c446a 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
S: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   replicates bdcddddd3d78a866b44b68c7ae0e5ccf875c446a
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
Can I set the above configuration? (type 'yes' to accept):
```

```
>>> 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 127.0.0.1:7000)
M: bdcddddd3d78a866b44b68c7ae0e5ccf875c446a 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
S: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   slots: (0 slots) slave
   replicates bdcddddd3d78a866b44b68c7ae0e5ccf875c446a
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   slots: (0 slots) slave
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   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.
```

连接集群

这里我们使用reids-cli连接集群，使用时加上 `-c` 参数，就可以连接到集群
连接7000端口的节点

```
[root@centos1 redis]# ./redis-cli -c -p 7000
127.0.0.1:7000> set name zgj
-> Redirected to slot [5798] located at 127.0.0.1:7001
OK
127.0.0.1:7001> get name
"zgj"
```

前面的理论知识我们知道了，分配key的时候，它会使用 `CRC16` 算法，这里将key `name` 分配到了7001节点上

```
Redirected to slot [5798] located at 127.0.0.1:7001
```

redis cluster 采用的方式很直接，它直接跳转到7001 节点了，而不是还在自身的7000节点。

好，现在我们连接7003这个从节点进入

```
[root@centos1 redis]# ./redis-cli -c -p 7003
127.0.0.1:7003> get name
```

这里获取 `name` 的值，也同样跳转到了7001上
我们再测试一下其他数据

```
127.0.0.1:7001> set age 20
-> Redirected to slot [741] located at 127.0.0.1:7000
OK
127.0.0.1:7000> set message helloworld
-> Redirected to slot [11537] located at 127.0.0.1:7002
OK
127.0.0.1:7002> set height 175
-> Redirected to slot [8223] located at 127.0.0.1:7001
OK
```

我们发现数据会在7000-7002这3个节点之间来回跳转

测试集群中的节点挂掉

上面我们建立了一个集群，3个主节点和3个从节点，7000-7002负责存取数据，7003-7005负责把7000-7005的数据同步到自己的节点上来。我们现在来模拟一下一台matser服务器宕机的情况

```
[root@centos1 redis]# ps -ef | grep redis
root      1731     1  0 18:21 ?        00:01:02 bin/redis-server *:7000 [cluster]
root      1733     1  0 18:21 ?        00:00:43 bin/redis-server *:7001 [cluster]
root      1735     1  0 18:21 ?        00:00:22 bin/redis-server *:7002 [cluster]
root      1743     1  0 18:21 ?        00:00:40 bin/redis-server *:7003 [cluster]
root      1745     1  0 18:21 ?        00:00:27 bin/redis-server *:7004 [cluster]
```

```
[root@centos1 redis]# kill 1731
[root@centos1 redis]# bin/redis-trib check 127.0.0.1:7001
>>> Performing Cluster Check (using node 127.0.0.1:7001)
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   slots: (0 slots) slave
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
M: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   slots:0-5460 (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.
```

这里看得出来，现在已经有了3个节点了，7003被选取成了替代7000成为主节点了。我们再来模拟7000节点重新启动了的情况，那么它还会自动加入到集群中吗？那么，7000这个节点上充当什么角色呢？ 我们试一下：

```
[root@centos1 redis]# bin/redis-server cluster/7000/redis.conf
[root@centos1 redis]# bin/redis-trib check 127.0.0.1:7000
```

```
   replicates d403713ab9db48aeac5b5393b69e1201026ef479
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   slots: (0 slots) slave
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
M: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   slots:0-5460 (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.
```

这里我们可以看到7000节点变成了7003节点的从节点我们试着将7000和7003两个节点都关掉

```
[root@centos1 redis]# ps -ef | grep redis
root      1733     1  0 18:21 ?        00:00:45 bin/redis-server *:7001 [cluster]
root      1735     1  0 18:21 ?        00:00:24 bin/redis-server *:7002 [cluster]
root      1743     1  0 18:21 ?        00:00:42 bin/redis-server *:7003 [cluster]
root      1745     1  0 18:21 ?        00:00:29 bin/redis-server *:7004 [cluster]
root      1749     1  0 18:21 ?        00:00:24 bin/redis-server *:7005 [cluster]
```

```
[root@centos1 redis] kill 1743
[root@centos1 redis] kill 24527

[root@centos1 redis]# bin/redis-trib check 127.0.0.1:7001
>>> Performing Cluster Check (using node 127.0.0.1:7001)
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   slots: (0 slots) slave
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[ERR] Not all 16384 slots are covered by nodes.
```

这里我们的集群就不能工作了，因为两个节点主节点和从节点都挂掉了，原来7001分配的slot现在无节点接管，需要人工介入重新分配slots。

集群中加入新的主节点

```
bin/redis-trib add-node 127.0.0.1:7006 127.0.0.1:7000
```

这里前面的节点表示要加入的节点，第二个节点表示要加入的集群中的任意一个节点，用来标识这个集群

```
[root@centos1 redis]# bin/redis-trib add-node 127.0.0.1:7006 127.0.0.1:7000
>>> Adding node 127.0.0.1:7006 to cluster 127.0.0.1:7000
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: bdcddddd3d78a866b44b68c7ae0e5ccf875c446a 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
S: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   slots: (0 slots) slave
   replicates bdcddddd3d78a866b44b68c7ae0e5ccf875c446a
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   slots: (0 slots) slave
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
```

```
[OK] New node added correctly.
[root@centos1 redis]# bin/redis-trib check 127.0.0.1:7006
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: bdcddddd3d78a866b44b68c7ae0e5ccf875c446a 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
S: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   slots: (0 slots) slave
   replicates bdcddddd3d78a866b44b68c7ae0e5ccf875c446a
S: b7ec92919e5bcffa76c8eee338f8ca5155293c64 127.0.0.1:7004
   slots: (0 slots) slave
   replicates b85519795fa42aa33d4e88d25104cbae895933a6
M: e55599320dabfb31bd22a01407e66121f075e7d3 127.0.0.1:7006
   slots: (0 slots) master
   0 additional replica(s)
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   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.
```

这里我们可以看到7006节点已经变成了一个主节点，然鹅，等等，好像发现了有什么地方不对

里面 `0 slots`,也就是说节点6没有分配哈希槽，即不能进行数据的存取，拿我加上去干嘛。。
原来redis cluster 不是在新加节点的时候帮我们做好了迁移工作，需要我们手动对集群进行重新分片迁移，也是这个命令：

```
/bin/redis-trib reshard 127.0.0.1:7000
```

这个命令是用来迁移slot节点的，后面的127.0.0.1:7000是表示哪个集群的，7000-7006都是可以的

```
[root@centos1]# redis-trib.rb reshard 127.0.0.1:7000
Connecting to node 127.0.0.1:7006: OK
Connecting to node 127.0.0.1:7001: OK
Connecting to node 127.0.0.1:7004: OK
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7002: OK
Connecting to node 127.0.0.1:7005: OK
Connecting to node 127.0.0.1:7003: OK
>>> Performing Cluster Check (using node 127.0.0.1:7006)
M: efc3131fbdc6cf929720e0e0f7136cae85657481 127.0.0.1:7006
   slots: (0 slots) master
   0 additional replica(s)
M: cb5c04b6160c3b7e18cad5d49d8e2987b27e0d6c 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: 4b4aef8b48c427a3c903518339d53b6447c58b93 127.0.0.1:7004
   slots: (0 slots) slave
   replicates cb5c04b6160c3b7e18cad5d49d8e2987b27e0d6c
```

```
M: dfa0754c7854a874a6ebd2613b86140ad97701fc 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
S: 30858dbf483b61b9838d5c1f853a60beaa4e7afd 127.0.0.1:7005
   slots: (0 slots) slave
   replicates dfa0754c7854a874a6ebd2613b86140ad97701fc
M: d2237fdcfbba672de766b913d1186cebcb6e1761 127.0.0.1:7003
   slots:0-5460 (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.
How many slots do you want to move (from 1 to 16384)?
```

它提示我们需要迁移多少slot到7006上，我们可以算一下：16384/4 = 4096，也就是说，为了平衡分配起见，我们需要移动4096个槽点到7006上。

好，那输入4096:
它又提示我们，接受的node ID是多少，7006的id 我们通过上面就可以看到是`efc3131fbdc6cf929720e0e0f7136cae85657481`：

```
What is the receiving node ID? efc3131fbdc6cf929720e0e0f7136cae85657481
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:
```

如果我们不打算从特定的节点上取出指定数量的哈希槽， 那么可以向 redis-trib 输入 all， 这样的话， 集群中的所有主节点都会成为源节点， redis-trib 将从各个源节点中各取出一部分哈希槽， 凑够 4096 个， 然后移动到7006节点上：

```
Source node #1:all
```

接下来就开始迁移了，并且会询问你是否确认：

```
Moving slot 1359 from d2237fdcfbba672de766b913d1186cebcb6e1761
    Moving slot 1360 from d2237fdcfbba672de766b913d1186cebcb6e1761
    Moving slot 1361 from d2237fdcfbba672de766b913d1186cebcb6e1761
    Moving slot 1362 from d2237fdcfbba672de766b913d1186cebcb6e1761
    Moving slot 1363 from d2237fdcfbba672de766b913d1186cebcb6e1761
    Moving slot 1364 from d2237fdcfbba672de766b913d1186cebcb6e1761
Do you want to proceed with the proposed reshard plan (yes/no)?
```

输入yes并回车后，redis-trib就会正式执行重新分片操作，将制定的哈希槽从源节点一个个移动到7006节点上迁移结束之后，我们来检查一下

```
M: bdcddddd3d78a866b44b68c7ae0e5ccf875c446a 127.0.0.1:7000
   slots:1365-5460 (4096 slots) master
   1 additional replica(s)
S: d403713ab9db48aeac5b5393b69e1201026ef479 127.0.0.1:7003
   slots: (0 slots) slave
   replicates bdcddddd3d78a866b44b68c7ae0e5ccf875c446a
```

```
M: e55599320dabfb31bd22a01407e66121f075e7d3 127.0.0.1:7006
   slots:0-1364,5461-6826,10923-12287 (4096 slots) master
   0 additional replica(s)
M: b85519795fa42aa33d4e88d25104cbae895933a6 127.0.0.1:7001
   slots:6827-10922 (4096 slots) master
   1 additional replica(s)
S: 8a0d2a3f271b349744a971e1b0a545405de2742e 127.0.0.1:7005
   slots: (0 slots) slave
   replicates b681e1a151890cbf957d1ff08352ee48f6ae39e6
M: b681e1a151890cbf957d1ff08352ee48f6ae39e6 127.0.0.1:7002
   slots:12288-16383 (4096 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.
```

我们可以看到

`slots:0-1364,5461-6826,10923-12287 (4096 slots)`

这些原来在其他节点上的哈希槽都迁移到了7006上

增加一个从节点

新建一个 7007从节点，作为7006的从节点

我们再新建一个节点7007，步骤类似，就先省略了。建好后，启动起来，我们看如何把它加入到集群中的从节点中：

add-node的时候加上`--slave`表示是加入到从节点中，但是这样加，是随机的。这里的命令行完全像我们在添加一个新主服务器时使用的一样，所以我们没有指定要给哪个主服 务器添加副本。这种情况下，redis-trib会将7007作为一个具有较少副本的随机的主服务器的副本。

那么，你猜，它会作为谁的从节点，应该是7006，因为7006还没有从节点。我们运行下。

```
[root@web3 7007]# redis-trib.rb add-node --slave 127.0.0.1:7007 127.0.0.1:7000
...
...
[OK] All 16384 slots covered.
Automatically selected master 127.0.0.1:7006
Connecting to node 127.0.0.1:7007: OK
>>> Send CLUSTER MEET to node 127.0.0.1:7007 to make it join the cluster.
Waiting for the cluster to join.
>>> Configure node as replica of 127.0.0.1:7006.
[OK] New node added correctly.
```

上面提示说，自动选择了7006作为master节点。并且成功了。我们检查下：

```
[root@centos1]# redis-trib.rb check 127.0.0.1:7000
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7006: OK
Connecting to node 127.0.0.1:7004: OK
Connecting to node 127.0.0.1:7005: OK
Connecting to node 127.0.0.1:7003: OK
Connecting to node 127.0.0.1:7001: OK
Connecting to node 127.0.0.1:7007: OK
```

```
   slots: (0 slots) slave
   replicates d2237fdcfbba672de766b913d1186cebcb6e1761
M: efc3131fbdc6cf929720e0e0f7136cae85657481 127.0.0.1:7006
   slots:0-1364,5461-6826,10923-12287 (4096 slots) master
   1 additional replica(s)
S: 4b4aef8b48c427a3c903518339d53b6447c58b93 127.0.0.1:7004
   slots: (0 slots) slave
   replicates cb5c04b6160c3b7e18cad5d49d8e2987b27e0d6c
S: 30858dbf483b61b9838d5c1f853a60beaa4e7afd 127.0.0.1:7005
   slots: (0 slots) slave
   replicates dfa0754c7854a874a6ebd2613b86140ad97701fc
M: d2237fdcfbba672de766b913d1186cebcb6e1761 127.0.0.1:7003
   slots:1365-5460 (4096 slots) master
   1 additional replica(s)
M: cb5c04b6160c3b7e18cad5d49d8e2987b27e0d6c 127.0.0.1:7001
   slots:6827-10922 (4096 slots) master
   1 additional replica(s)
S: 86d05e7c2b197dc182b5e71069e791d033cf899e 127.0.0.1:7007
   slots: (0 slots) slave
   replicates efc3131fbdc6cf929720e0e0f7136cae85657481
M: dfa0754c7854a874a6ebd2613b86140ad97701fc 127.0.0.1:7002
   slots:12288-16383 (4096 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.
```

果然，7007加入到了7006的从节点当中。

```
bin/redis-trib.rb add-node --slave --master-id efc3131fbdc6cf929720e0e0f7136cae85657481 1
```

`--master-id` 表示指定的主节点node id。这里指定的是 7006 这个主节点。

```
Waiting for the cluster to join.
>>> Configure node as replica of 127.0.0.1:7006.
[OK] New node added correctly.
```

提示我们已经作为7006的从节点了，也就是加入到7006的从节点来了，照这么说，7006就有2个从节点了，我们看一下：

```
bin/redis-cli -c -p 7008 cluster nodes |grep efc3131fbdc6cf929720e0e0f7136cae85657481
86d05e7c2b197dc182b5e71069e791d033cf899e 127.0.0.1:7007 slave efc3131fbdc6cf929720e0e0f71
efc3131fbdc6cf929720e0e0f7136cae85657481 127.0.0.1:7006 master - 0 1445089508289 8 connec
44321e7d619410dc4e0a8745366610a0d06d2395 127.0.0.1:7008 myself,slave efc3131fbdc6cf929720
```

我们过滤了下看结果，果真，7007和7008是7006的从节点了。

刚好，我们再做一个实验，我把7006的进程杀掉，看7007和7008谁会变成主节点：

```
[root@centos1]# ps -ef|grep redis
root     11384     1  0 09:56 ?        00:00:16 redis-server *:7001 [cluster]
root     11388     1  0 09:56 ?        00:00:16 redis-server *:7002 [cluster]
root     11392     1  0 09:56 ?        00:00:16 redis-server *:7003 [cluster]
root     11396     1  0 09:56 ?        00:00:15 redis-server *:7004 [cluster]
```

```
root     12202     1  0 13:14 ?        00:00:02 redis-server *:7007 [cluster]
root     12219     1  0 13:39 ?        00:00:00 redis-server *:7008 [cluster]
root     12239  8259  0 13:49 pts/0    00:00:00 grep redis
[root@centos1]# kill 12132
[root@centos1]# redis-cli -c -p 7008
127.0.0.1:7008> get ss5rtr
-> Redirected to slot [1188] located at 127.0.0.1:7007
"66"
127.0.0.1:7007> cluster nodes
efc3131fbdc6cf929720e0e0f7136cae85657481 127.0.0.1:7006 master,fail - 1445089780668 14450
d2237fdcfbba672de766b913d1186cebcb6e1761 127.0.0.1:7003 master - 0 1445089812195 7 connec
30858dbf483b61b9838d5c1f853a60beaa4e7afd 127.0.0.1:7005 slave dfa0754c7854a874a6ebd2613b8
86d05e7c2b197dc182b5e71069e791d033cf899e 127.0.0.1:7007 myself,master - 0 0 10 connected
cb5c04b6160c3b7e18cad5d49d8e2987b27e0d6c 127.0.0.1:7001 master - 0 1445089814214 2 connec
4b4aef8b48c427a3c903518339d53b6447c58b93 127.0.0.1:7004 slave cb5c04b6160c3b7e18cad5d49d8
44321e7d619410dc4e0a8745366610a0d06d2395 127.0.0.1:7008 slave 86d05e7c2b197dc182b5e71069e
3707debcbe7be66d4a1968eaf3a5ffaf4308efa4 127.0.0.1:7000 slave d2237fdcfbba672de766b913d11
dfa0754c7854a874a6ebd2613b86140ad97701fc 127.0.0.1:7002 master - 0 1445089813204 3 connec
127.0.0.1:7007>
```

这里7007获得了成为主节点的机会，7008就变成了7007的从节点。

那么这个时候，重启7006节点，那么他就会变成了一个7007的从节点了。

移除一个节点

上面是增加一个节点，接下来就是移除一个节点了，移除节点的命令是

没我们尝试下输入以下命令

```
[root@centos]# bin/redis-trib.rb del-node 127.0.0.1:7000 86d05e7c2b197dc182b5e71069e791d0
>>> Removing node 86d05e7c2b197dc182b5e71069e791d033cf899e from cluster 127.0.0.1:7000
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7006: OK
Connecting to node 127.0.0.1:7004: OK
Connecting to node 127.0.0.1:7001: OK
Connecting to node 127.0.0.1:7003: OK
Connecting to node 127.0.0.1:7007: OK
Connecting to node 127.0.0.1:7008: OK
Connecting to node 127.0.0.1:7005: OK
Connecting to node 127.0.0.1:7002: OK
[ERR] Node 127.0.0.1:7007 is not empty! Reshard data away and try again.
```

这里报错了，提示我们7007节点里面有数据，让我们把7007节点里的数据移除出去，也就是说需要重新分片，这个和上面增加节点的方式一样，我们再来一遍

```
bin/redis-trib.rb reshard 127.0.0.1:7000
```

省去中间内容，原来7007节点上已经有了4096个哈希槽，这里我们也移动4096个哈希槽然后将这些哈希槽移动到7001节点上

```
Source node #1:86d05e7c2b197dc182b5e71069e791d033cf899e
Source node #2:done
```

然后我们再继续执行移除命令，结果如下

```
[root@centos1]# redis-trib.rb del-node 127.0.0.1:7000 86d05e7c2b197dc182b5e71069e791d033c
>>> Removing node 86d05e7c2b197dc182b5e71069e791d033cf899e from cluster 127.0.0.1:7000
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7006: OK
Connecting to node 127.0.0.1:7004: OK
Connecting to node 127.0.0.1:7001: OK
Connecting to node 127.0.0.1:7003: OK
Connecting to node 127.0.0.1:7007: OK
Connecting to node 127.0.0.1:7008: OK
Connecting to node 127.0.0.1:7005: OK
Connecting to node 127.0.0.1:7002: OK
>>> Sending CLUSTER FORGET messages to the cluster...
>>> 127.0.0.1:7006 as replica of 127.0.0.1:7001
>>> 127.0.0.1:7008 as replica of 127.0.0.1:7001
>>> SHUTDOWN the node.
```

删除成功，而且还很人性化的将7006和7008这2个原来7007的附属节点送给了7001。考虑的真周到~

移除一个从节点

移除一个从节点就比较简单了，因为从节点没有哈希槽，也不需要考虑数据迁移，直接移除就行

```
[root@centos1]# redis-trib.rb del-node 127.0.0.1:7005 44321e7d619410dc4e0a8745366610a0d06
>>> Removing node 44321e7d619410dc4e0a8745366610a0d06d2395 from cluster 127.0.0.1:7005
```

```
Connecting to node 127.0.0.1:7004: OK
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7006: OK
Connecting to node 127.0.0.1:7008: OK
Connecting to node 127.0.0.1:7003: OK
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node.
[root@centos1]# redis-trib.rb check 127.0.0.1:7008
Connecting to node 127.0.0.1:7008: [ERR] Sorry, can't connect to node 127.0.0.1:7008
```

表示移除成功

Redis性能测试

Redis自带了性能测试工具 `redis-benchmark`
使用说明如下：

```
Usage: redis-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests]> [-k <boolea

 -h <hostname>      Server hostname (default 127.0.0.1)
 -p <port>          Server port (default 6379)
 -s <socket>        Server socket (overrides host and port)
 -c <clients>       Number of parallel connections (default 50)
 -n <requests>      Total number of requests (default 10000)
 -d <size>          Data size of SET/GET value in bytes (default 2)
 -k <boolean>       1=keep alive 0=reconnect (default 1)
 -r <keyspacelen>   Use random keys for SET/GET/INCR, random values for SADD
```

```
 number of values for the random number. For instance
 if set to 10 only rand:000000000000 - rand:000000000009
 range will be allowed.
-P <numreq>        Pipeline <numreq> requests. Default 1 (no pipeline).
-q                 Quiet. Just show query/sec values
--csv              Output in CSV format
-l                 Loop. Run the tests forever
-t <tests>         Only run the comma-separated list of tests. The test
                   names are the same as the ones produced as output.
-I                 Idle mode. Just open N idle connections and wait.
```

基准测试

基准的测试命令：

`redis-benchmark -q -n 100000`

结果入下：

```
root@centos1 bin]# redis-benchmark -q -n 100000
-bash: redis-benchmark: command not found
[root@centos1 bin]# ./redis-benchmark -q -n 100000
PING_INLINE: 61576.36 requests per second
PING_BULK: 60277.28 requests per second
SET: 61349.69 requests per second
GET: 60459.49 requests per second
INCR: 58858.15 requests per second
LPUSH: 59066.75 requests per second
RPUSH: 57339.45 requests per second
LPOP: 55586.44 requests per second
```

```
LPUSH (needed to benchmark LRANGE): 57012.54 requests per second
LRANGE_100 (first 100 elements): 55803.57 requests per second
LRANGE_300 (first 300 elements): 54914.88 requests per second
LRANGE_500 (first 450 elements): 53333.33 requests per second
LRANGE_600 (first 600 elements): 56529.11 requests per second
MSET (10 keys): 59276.82 requests per second
```

这里可以看出，单机版的redis每秒可以处理6万个请求，这已经是一个非常厉害的数据了，不得不佩服我们再来看下集群情况下是是什么情况

```
[root@centos1 bin]# ./redis-benchmark -q -n 100000 -p 7000
PING_INLINE: 64599.48 requests per second
PING_BULK: 64184.85 requests per second
SET: 66800.27 requests per second
GET: 65616.80 requests per second
INCR: 66269.05 requests per second
LPUSH: 40273.86 requests per second
RPUSH: 40355.12 requests per second
LPOP: 43421.62 requests per second
RPOP: 45187.53 requests per second
SADD: 62539.09 requests per second
SPOP: 61538.46 requests per second
LPUSH (needed to benchmark LRANGE): 38182.51 requests per second
LRANGE_100 (first 100 elements): 25555.84 requests per second
LRANGE_300 (first 300 elements): 9571.21 requests per second
LRANGE_500 (first 450 elements): 7214.49 requests per second
LRANGE_600 (first 600 elements): 5478.85 requests per second
MSET (10 keys): 41893.59 requests per second
```

流水线测试

使用流水线
默认情况下，每个客户端都是在一个请求完成之后才发送下一个请求（基准会模拟50个客户端除非使用-c指定特别的数量），这意味着服务器几乎是按顺序读取每个客户端的命令。RTT也加入了其中。
真实世界会更复杂，Redis支持/topics/pipelining，使得可以一次性执行多条命令成为可能。Redis流水线可以提高服务器的TPS
`redis-benchmark -n 1000000 -t set,get -P 16 -q` 加入-P选项使用管道技术，一次执行多条命令

```
./redis-benchmark -n 1000000 -t set,get -P 16 -q
SET: 515198.34 requests per second
GET: 613873.56 requests per second
```

每秒处理get/sret请求达到了60/50W,真的厉害！

遇到的问题

1. 安装redis集群的时候遇到了挺多问题，踩了很多坑，单单是修改配置文件就出了不少问题，那些配置文件的内容都要一一修改，有些配置不修改就会出现无法创建进程的错误
2. 注意配置集群的时候不要加密码，否则会出现无法连接的情况
3. `gem install` 的时候需要修改镜像或者翻墙
4. 昨天启动成功，今天启动的时候报错

解决方法：

1). 将需要新增的节点下aof、rdb等本地备份文件删除；

2). 同时将新Node的集群配置文件删除,即：删除你 `redis.conf` 里面 `cluster-config-file` 所在的文件；

3). 再次添加新节点如果还是报错，则登录新Node,执行 `bin/redis-cli-h x -p` 对数据库进行清除：

```
127.0.0.1:7001>  flushdb      #清空当前数据库
```

总结

之间对了Redis的了解并不是说非常多，只是简单的会用，因为现在企业里也很多都在用，刚好老大说接下来的项目可能会用到Redis集群，让我先去了解下，所以最近就在回头看，一边看文档，博客，一边实践，踩了很多的坑，出问题的时候的确是让人感到很痛苦很郁闷的，可是当运行成功的那一刻心情却是无比激动和开心的，可能这就是编程的魅力吧。

> 个人博客： blog.zgj12138.cn/)
> CSDN: blog.csdn.net/zgj12138

Redis

相关热门文章

你需要知道的那些 redis 数据结构（前篇）

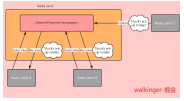
饿了么物流技术团队 1

后端开发应该掌握的Redis基础

张君鸿 31

Redis 高负载下的中断优化

我是月亮呀 10



你真的了解Redis的发布订阅?

walkinger 20

Redis集群容器化安装

MadPecker 10

评论

仲夏夜空星星多

新版的使用redis-cli --cluster来开启集群的，还有怎么关闭集群这是一个问题，这里没提到

2月前 回复

M莫 Lv2 JAVA工程师

赞

3月前 回复