二

40 实战：Redis 集群模式（下）

上篇文章我们讲了 Redis 集群的搭建与节点的动态添加和删除，我们这里再来简单的复习一下，其中 30001~30006 是我们最初搭建的集群，而 30007 和 30008 是后面动态添加的主从节点，我们使用 `--cluster info` 命令来看一下主节点和槽位的分配情况，执行代码如下：

```
$ redis-cli --cluster info 127.0.0.1:30001
127.0.0.1:30001 (887397e6...) -> 0 keys | 5461 slots | 1 slaves.
127.0.0.1:30007 (df019085...) -> 0 keys | 0 slots | 1 slaves.
127.0.0.1:30003 (f5958382...) -> 0 keys | 5461 slots | 1 slaves.
127.0.0.1:30002 (3da35c40...) -> 0 keys | 5462 slots | 1 slaves.
[OK] 0 keys in 4 masters.
0.00 keys per slot on average.
```

可以看出动态添加的主节点 30007 有一个从节点，但并没有分配任何槽位，这显然是不能满足我们的需求的，只添加了节点，但不处理任何数据，所以我们需要重新分片，让数据存储在所有的主节点上，这样才能发挥集群的最大作用。

重新分片

我们可以使用 reshard 命令，对槽位（slots）进行重新分配，执行命令如下：

```
$ redis-cli --cluster reshard 127.0.0.1:30007
>>> Performing Cluster Check (using node 127.0.0.1:30007)
M: df0190853a53d8e078205d0e2fa56046f20362a7 127.0.0.1:30007
   slots:[0-1332],[5461-6794],[10923-12255] (4000 slots) master
   1 additional replica(s)
S: dc0702625743c48c75ea935c87813c4060547cef 127.0.0.1:30006
   slots: (0 slots) slave
   replicates 3da35c40c43b457a113b539259f17e7ed616d13d
M: 3da35c40c43b457a113b539259f17e7ed616d13d 127.0.0.1:30002
   slots:[6795-10922] (4128 slots) master
   1 additional replica(s)
S: 1a324d828430f61be6eaca7eb2a90728dd5049de 127.0.0.1:30004
   slots: (0 slots) slave
   replicates f5958382af41d4e1f5b0217c1413fe19f390b55f
S: 1d09d26fd755298709efe60278457eaa09cefc26 127.0.0.1:30008
```

```
   slots: (0 slots) slave
   replicates df0190853a53d8e078205d0e2fa56046f20362a7
S: abec9f98f9c01208ba77346959bc35e8e274b6a3 127.0.0.1:30005
   slots: (0 slots) slave
   replicates 887397e6fefe8ad19ea7569e99f5eb8a803e3785
M: f5958382af41d4e1f5b0217c1413fe19f390b55f 127.0.0.1:30003
   slots:[12256-16383] (4128 slots) master
   1 additional replica(s)
M: 887397e6fefe8ad19ea7569e99f5eb8a803e3785 127.0.0.1:30001
   slots:[1333-5460] (4128 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)?
```

在执行的过程中，它会询问你打算移动多少个节点，取值范围是 1 到 16384，我们这里输入 4000，意思是移动 4000 个槽位到某个主节点，输入命令之后，执行效果如下：

```
How many slots do you want to move (from 1 to 16384)? 4000
What is the receiving node ID?
```

接着它会询问你需要把这些槽位分配到哪个节点上，请输入节点 Id，我们把上面 30007 端口的 Id 输入进去之后，敲击回车，执行效果如下：

```
How many slots do you want to move (from 1 to 16384)? 4000
What is the receiving node ID? df0190853a53d8e078205d0e2fa56046f20362a7
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:
```

此时它会询问你要从那个源节点中进行转移，输入 `all` 命令表示从所有节点中随机抽取，执行效果如下：

```
# ......忽略其他
Moving slot 2656 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2657 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2658 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2659 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2660 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2661 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2662 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2663 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Moving slot 2664 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
```

```
Moving slot 2665 from 887397e6fefe8ad19ea7569e99f5eb8a803e3785
Do you want to proceed with the proposed reshard plan (yes/no)?
```

此时它会把所有要转移的节点信息列举出来，让你确认，你只需要输入 `yes` 就开始执行转移操作了。

在执行完转移之后，我们使用 `cluster slots` 命令来查看一下槽位的相关信息，结果如下：

```
$ redis-cli -c -p 30001
127.0.0.1:30001> cluster slots # 查看集群槽位信息
1) 1) (integer) 0
   2) (integer) 1332
   3) 1) "127.0.0.1"
      2) (integer) 30007
      3) "df0190853a53d8e078205d0e2fa56046f20362a7"
   4) 1) "127.0.0.1"
      2) (integer) 30008
      3) "1d09d26fd755298709efe60278457eaa09cefc26"
2) 1) (integer) 5461
   2) (integer) 6794
   3) 1) "127.0.0.1"
      2) (integer) 30007
      3) "df0190853a53d8e078205d0e2fa56046f20362a7"
   4) 1) "127.0.0.1"
      2) (integer) 30008
      3) "1d09d26fd755298709efe60278457eaa09cefc26"
3) 1) (integer) 10923
   2) (integer) 12255
   3) 1) "127.0.0.1"
      2) (integer) 30007
      3) "df0190853a53d8e078205d0e2fa56046f20362a7"
   4) 1) "127.0.0.1"
      2) (integer) 30008
      3) "1d09d26fd755298709efe60278457eaa09cefc26"
4) 1) (integer) 12256
   2) (integer) 16383
   3) 1) "127.0.0.1"
      2) (integer) 30003
      3) "f5958382af41d4e1f5b0217c1413fe19f390b55f"
   4) 1) "127.0.0.1"
      2) (integer) 30004
      3) "1a324d828430f61be6eaca7eb2a90728dd5049de"
5) 1) (integer) 6795
   2) (integer) 10922
   3) 1) "127.0.0.1"
      2) (integer) 30002
      3) "3da35c40c43b457a113b539259f17e7ed616d13d"
   4) 1) "127.0.0.1"
      2) (integer) 30006
      3) "dc0702625743c48c75ea935c87813c4060547cef"
6) 1) (integer) 1333
```

```
2) (integer) 5460
3) 1) "127.0.0.1"
   2) (integer) 30001
   3) "887397e6fefe8ad19ea7569e99f5eb8a803e3785"
4) 1) "127.0.0.1"
   2) (integer) 30005
   3) "abec9f98f9c01208ba77346959bc35e8e274b6a3"
```

从结果可以看出 30007 分别从其他三个主节点中抽取了一部分槽位，作为了自己的槽位。

> 注意，执行此过程中如果出现 `/usr/bin/env: ruby: No such file or directory` 错误，表明工具在执行的时候需要依赖 Ruby 环境，可使用命令 `yum install ruby` 安装 Ruby 环境即可。

槽位定位算法

Redis 集群总共的槽位数是 16384 个，每一个主节点负责维护一部分槽以及槽所映射的键值数据，Redis 集群默认会对要存储的 key 值使用 CRC16 算法进行 hash 得到一个整数值，然后用这个整数值对 16384 进行取模来得到具体槽位，公式为：

> slot = CRC16(key) % 16383

负载均衡

在 Redis 集群负载不均衡的情况下，我们可以使用 rebalance 命令重新分配各个节点负责的槽数量，从而使得各个节点的负载压力趋于平衡，从而提高 Redis 集群的整体运行效率。

rebalance 命令如下：

```
$ redis-cli --cluster rebalance 127.0.0.1:30007
```

需要注意的是，即使输入 rebalance 命令，但它可能不会执行，当它认为没有必要进行分配时会直接退出，如下所示：

```
$ redis-cli --cluster rebalance 127.0.0.1:30007
>>> Performing Cluster Check (using node 127.0.0.1:30007)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
```

```
[OK] All 16384 slots covered.
*** No rebalancing needed! All nodes are within the 2.00% threshold.
```

代码实战

前面我们讲了 Redis 集群搭建的相关功能，接下来我们使用 Java 代码来操作一下 Redis 集群，本文依然使用 Jedis 来作为客户端进行相关的操作，核心代码如下：

```
import redis.clients.jedis.HostAndPort;
import redis.clients.jedis.JedisCluster;
import java.util.HashSet;
import java.util.Set;

public class ClusterExample {
    public static void main(String[] args) {
        // 集群节点信息
        Set<HostAndPort> nodes = new HashSet<>();
        nodes.add(new HostAndPort("127.0.0.1", 30001));
        nodes.add(new HostAndPort("127.0.0.1", 30002));
        nodes.add(new HostAndPort("127.0.0.1", 30003));
        nodes.add(new HostAndPort("127.0.0.1", 30004));
        nodes.add(new HostAndPort("127.0.0.1", 30005));
        nodes.add(new HostAndPort("127.0.0.1", 30006));
        // 创建集群连接
        JedisCluster jedisCluster = new JedisCluster(nodes,
                10000,  // 超时时间
                10);    // 最大尝试重连次数
        // 添加数据
        String setResult = jedisCluster.set("lang", "redis");
        // 输出结果
        System.out.println("添加：" + setResult);
        // 查询结果
        String getResult = jedisCluster.get("lang");
        // 输出结果
        System.out.println("查询：" + getResult);
    }
}
```

以上程序的执行结果如下：

```
添加：OK
查询：redis
```

此结果表明 Redis 集群操作正常，除了使用的操作对象不同之外，操作的方法名称都是相同的，所以对程序员来说比较友好，你可以根据自己的业务场景去写相应的代码了。

故障

在文章的最后部分，我们来看一下 Redis 集群故障相关的知识点，这样在我们遇到一些故障问题时就不会那么慌张了，并且能为我们处理故障时提供一些帮助。

故障发现

故障发现里面有两个重要的概念：疑似下线（PFAIL-Possibly Fail）和确定下线（Fail）。

集群中的健康监测是通过定期向集群中的其他节点发送 PING 信息来确认的，如果发送 PING 消息的节点在规定时间内，没有收到返回的 PONG 消息，那么对方节点就会被标记为疑似下线。

一个节点发现某个节点疑似下线，它会将这条信息向整个集群广播，其它节点就会收到这个消息，并且通过 PING 的方式监测某节点是否真的下线了。如果一个节点收到某个节点疑似下线的数量超过集群数量的一半以上，就可以标记该节点为确定下线状态，然后向整个集群广播，强迫其它节点也接收该节点已经下线的事实，并立即对该失联节点进行主从切换。

这就是疑似下线和确认下线的概念，这个概念和哨兵模式里面的主观下线和客观下线的概念比较类似。

故障转移

当一个节点被集群标识为确认下线之后就可以执行故障转移了，故障转移的执行流程如下：

1. 从下线的主节点的所有从节点中，选择一个从节点（选择的方法详见下面“新主节点选举原则”部分）；
2. 从节点会执行 SLAVEOF NO ONE 命令，关闭这个从节点的复制功能，并从从节点转变回主节点，原来同步所得的数据集不会被丢弃；
3. 新的主节点会撤销所有对已下线主节点的槽指派，并将这些槽全部指派给自己；
4. 新的主节点向集群广播一条 PONG 消息，这条 PONG 消息是让集群中的其他节点知道此节点已经由从节点变成了主节点，并且这个主节点已经接管了原本由已下线节点负责处理的槽位信息；
5. 新的主节点开始处理相关的命令请求，此故障转移过程完成。

新主节点选举原则

新主节点选举的方法是这样的：

1. 集群的纪元（epoch）是一个自增计数器，初始值为0；
2. 而每个主节点都有一次投票的机会，主节点会把这一票投给第一个要求投票的从节点；
3. 当从节点发现自己正在复制的主节点确认下线之后，就会向集群广播一条消息，要求所有有投票权的主节点给此从节点投票；
4. 如果有投票权的主节点还没有给其他人投票的情况下，它会向第一个要求投票的从节点发送一条消息，表示把这一票投给这个从节点；
5. 当从节点收到投票数量大于集群数量的半数以上时，这个从节点就会当选为新的主节点。

到这里整个新主节点的选择就完成了。

小结

本文从动态新增的主节点通过 reshard 命令重新分配槽位开始，讲了槽位定位的算法以及负载均衡的实现方法，还使用代码的方式演示了如何在程序中操作 Redis 集群，最后讲了 Redis 集群故障发现以及故障转移、新主节点选举的整个流程，希望对你理解 Redis 的集群有帮助。

上一页 下一页