课前准备

- 准备redis安装包

课堂主题

redis事务分析、Redis持久化、Redis主从复制、Redis哨兵机制、Redis集群原理

课堂目标

- 理解redis事务原理
- 掌握redis的RDB和AOF的原理和选型
- 理解Redis主从复制原理和哨兵原理
- 能够配置Redis主从+哨兵
- 理解RedisCluster的原理和容错机制
- 能够配置RedisCluster并使用

知识要点

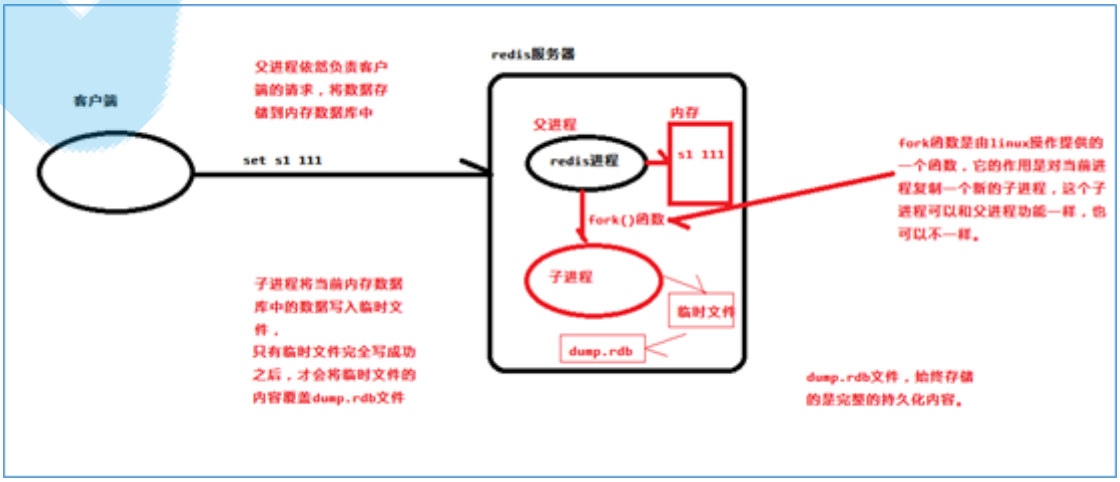
Redis事务

- `Redis`的事务是通过`MULTI`、`EXEC`、`DISCARD`和`WATCH`这四个命令来完成的。
- `Redis`的单个命令都是原子性的，所以这里需要确保事务性的对象是**命令集合。**
- `Redis`将命令集合序列化并确保处于同一事务的**命令集合连续且不被打断**的执行
- `Redis`不支持回滚操作。

Redis持久化

`Redis`是一个**内存**数据库，为了保证数据的持久性，它提供了两种持久化方案：

- `RDB`方式（默认）



- `AOF`方式

默认情况下`Redis`没有开启`AOF`（`append only file`）方式的持久化。

开启`AOF`持久化后，每执行一条会**更改`Redis`中的数据的命令**，`Redis`就会将该命令写入硬盘中的`AOF`文件，这一过程显然**会降低`Redis`的性能**，但大部分情况下这个影响是能够接受的，另外使**用较快的硬盘可以提高`AOF`的性能。**

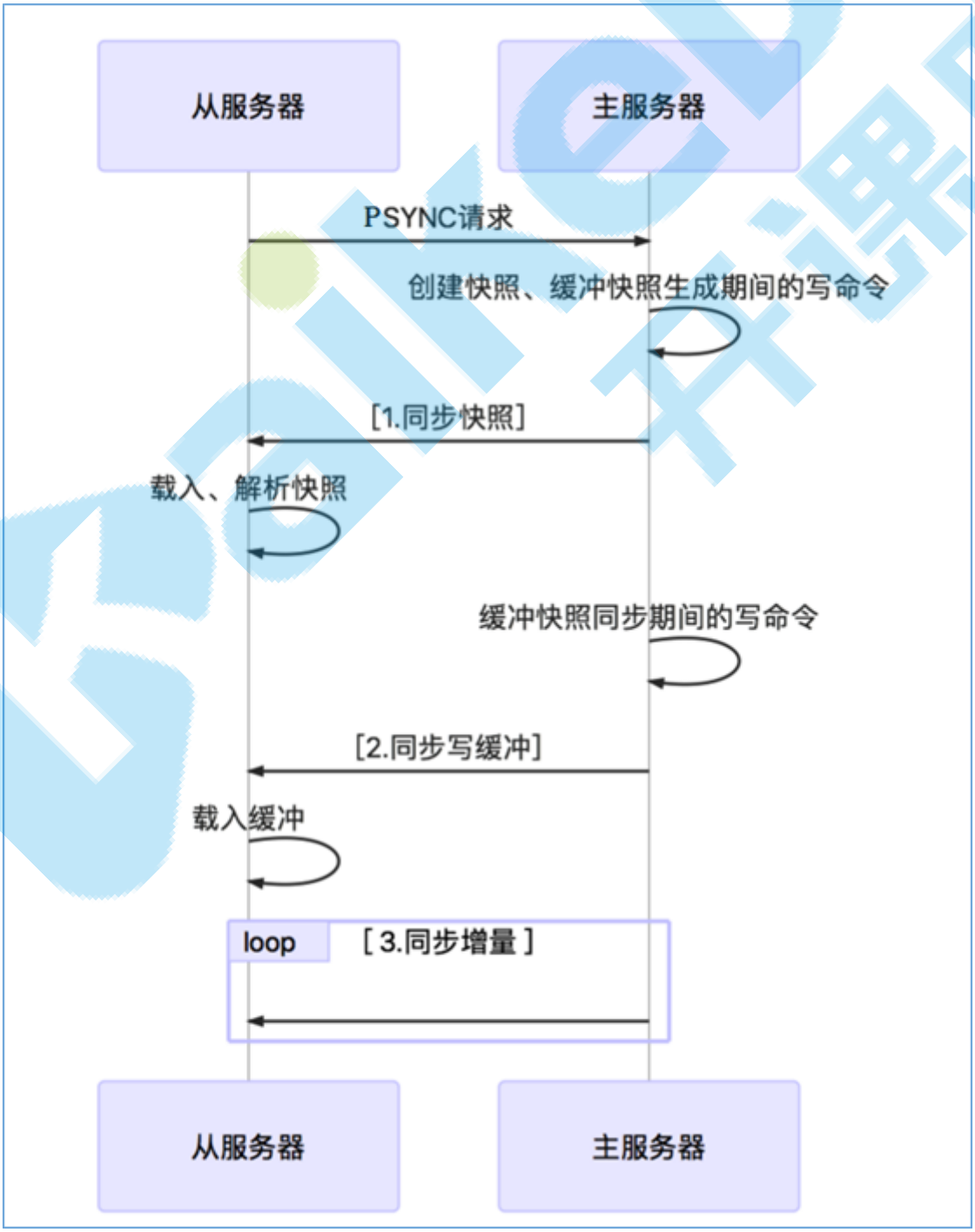
Redis主从复制

什么是主从复制

主从配置

实现原理

全量同步



增量同步

Redis哨兵机制

简介

哨兵进程的作用

故障判定原理分析

自动故障迁移

案例演示

- 修改从机的`sentinel.conf`:

```
# 哨兵sentinel监控的redis主节点的 ip port
# master-name  可以自己命名的主节点名字 只能由字母A-z、数字0-9 、这三个字符".-_"组成。
# quorum 当这些quorum个数sentinel哨兵认为master主节点失联 那么这时 客观上认为主节点失联了
# sentinel monitor <master-name> <master ip> <master port> <quorum>
sentinel monitor mymaster 192.168.10.133 6379 1
```

- **其他配置项说明**

`sentinel.conf`

```
# 哨兵sentinel实例运行的端口 默认26379
port 26379

# 哨兵sentinel的工作目录
dir /tmp

# 哨兵sentinel监控的redis主节点的 ip port
# master-name  可以自己命名的主节点名字 只能由字母A-z、数字0-9 、这三个字符".-_"组成。
# quorum 当这些quorum个数sentinel哨兵认为master主节点失联 那么这时 客观上认为主节点失联了
# sentinel monitor <master-name> <ip> <redis-port> <quorum>
sentinel monitor mymaster 127.0.0.1 6379 2

# 当在Redis实例中开启了requirepass foobared 授权密码 这样所有连接Redis实例的客户端都要提供密码
# 设置哨兵sentinel 连接主从的密码 注意必须为主从设置一样的验证密码
# sentinel auth-pass <master-name> <password>
sentinel auth-pass mymaster MySUPER--secret-0123passw0rd


# 指定多少毫秒之后 主节点没有应答哨兵sentinel 此时 哨兵主观上认为主节点下线 默认30秒
# sentinel down-after-milliseconds <master-name> <milliseconds>
sentinel down-after-milliseconds mymaster 30000

# 这个配置项指定了在发生failover主备切换时最多可以有多少个slave同时对新的master进行 同步，
```

```
这个数字越小，完成failover所需的时间就越长，
但是如果这个数字越大，就意味着越 多的slave因为replication而不可用。
可以通过将这个值设为 1 来保证每次只有一个slave 处于不能处理命令请求的状态。
# sentinel parallel-syncs <master-name> <numslaves>
sentinel parallel-syncs mymaster 1



# 故障转移的超时时间 failover-timeout 可以用在以下这些方面：
#1. 同一个sentinel对同一个master两次failover之间的间隔时间。
#2. 当一个slave从一个错误的master那里同步数据开始计算时间。直到slave被纠正为向正确的master那里同步数据时。
#3.当想要取消一个正在进行的failover所需要的时间。
#4.当进行failover时，配置所有slaves指向新的master所需的最大时间。不过，即使过了这个超时，slaves依然会被正确配置为指向master，但是就不按parallel-syncs所配置的规则来了
# 默认三分钟
# sentinel failover-timeout <master-name> <milliseconds>
sentinel failover-timeout mymaster 180000

# SCRIPTS EXECUTION

#配置当某一事件发生时所需要执行的脚本，可以通过脚本来通知管理员，例如当系统运行不正常时发邮件通知相关人员。
#对于脚本的运行结果有以下规则：
#若脚本执行后返回1，那么该脚本稍后将会被再次执行，重复次数目前默认为10
#若脚本执行后返回2，或者比2更高的一个返回值，脚本将不会重复执行。
#如果脚本在执行过程中由于收到系统中断信号被终止了，则同返回值为1时的行为相同。
#一个脚本的最大执行时间为60s，如果超过这个时间，脚本将会被一个SIGKILL信号终止，之后重新执行。

#通知型脚本:当sentinel有任何警告级别的事件发生时（比如说redis实例的主观失效和客观失效等等），将会去调用这个脚本，这时这个脚本应该通过邮件，SMS等方式去通知系统管理员关于系统不正常运行的信息。调用该脚本时，将传给脚本两个参数，一个是事件的类型，一个是事件的描述。
#如果sentinel.conf配置文件中配置了这个脚本路径，那么必须保证这个脚本存在于这个路径，并且是可执行的，否则sentinel无法正常启动成功。
#通知脚本
# sentinel notification-script <master-name> <script-path>
sentinel notification-script mymaster /var/redis/notify.sh

# 客户端重新配置主节点参数脚本
# 当一个master由于failover而发生改变时，这个脚本将会被调用，通知相关的客户端关于master地址已经发生改变的信息。
# 以下参数将会在调用脚本时传给脚本:
# <master-name> <role> <state> <from-ip> <from-port> <to-ip> <to-port>
# 目前<state>总是“failover”,
# <role>是“leader”或者“observer”中的一个。
# 参数 from-ip, from-port, to-ip, to-port是用来和旧的master和新的master(即旧的slave)通信的
# 这个脚本应该是通用的，能被多次调用，不是针对性的。
# sentinel client-reconfig-script <master-name> <script-path>
 sentinel client-reconfig-script mymaster /var/redis/reconfig.sh
```

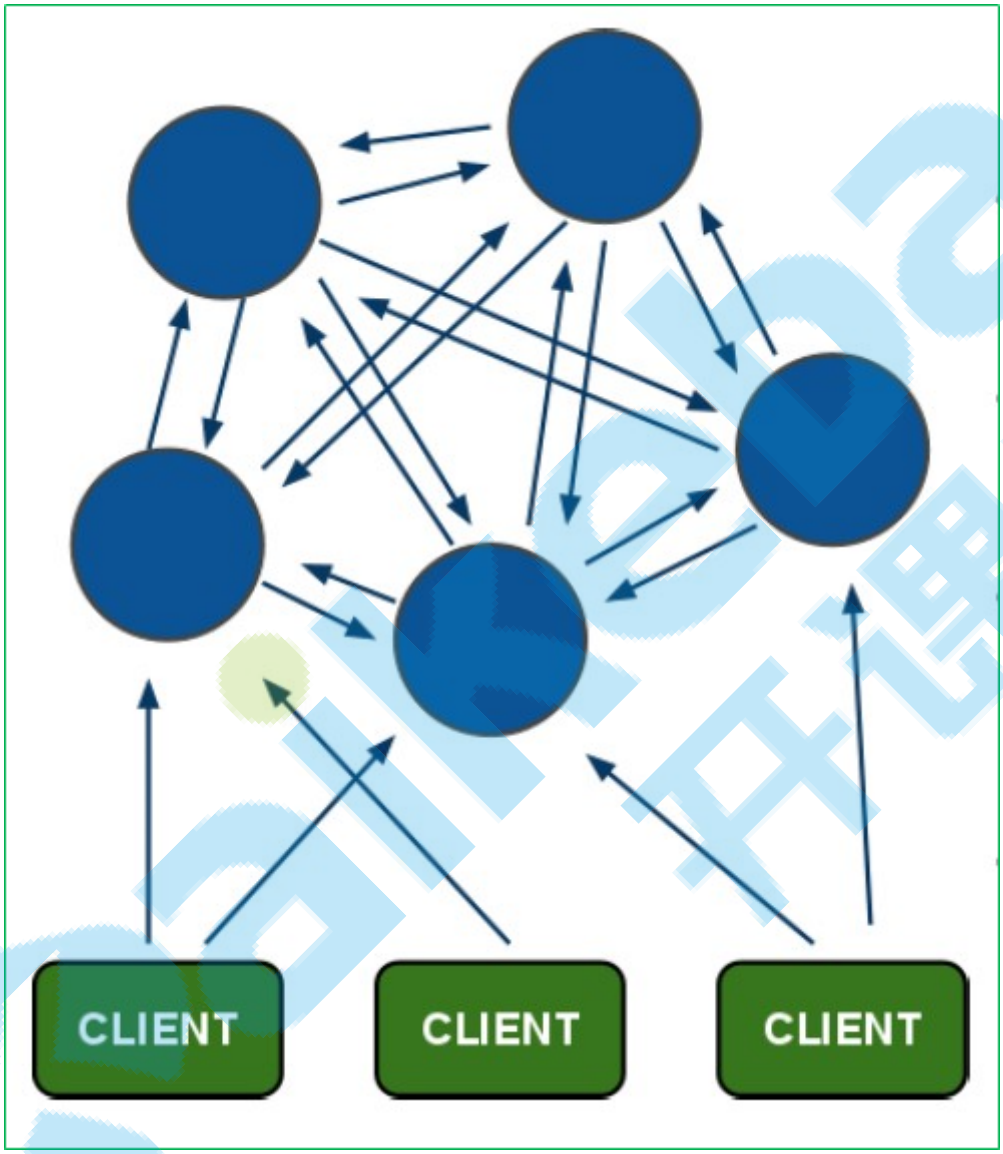
- 通过 `redis-sentinel` 启动哨兵服务

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

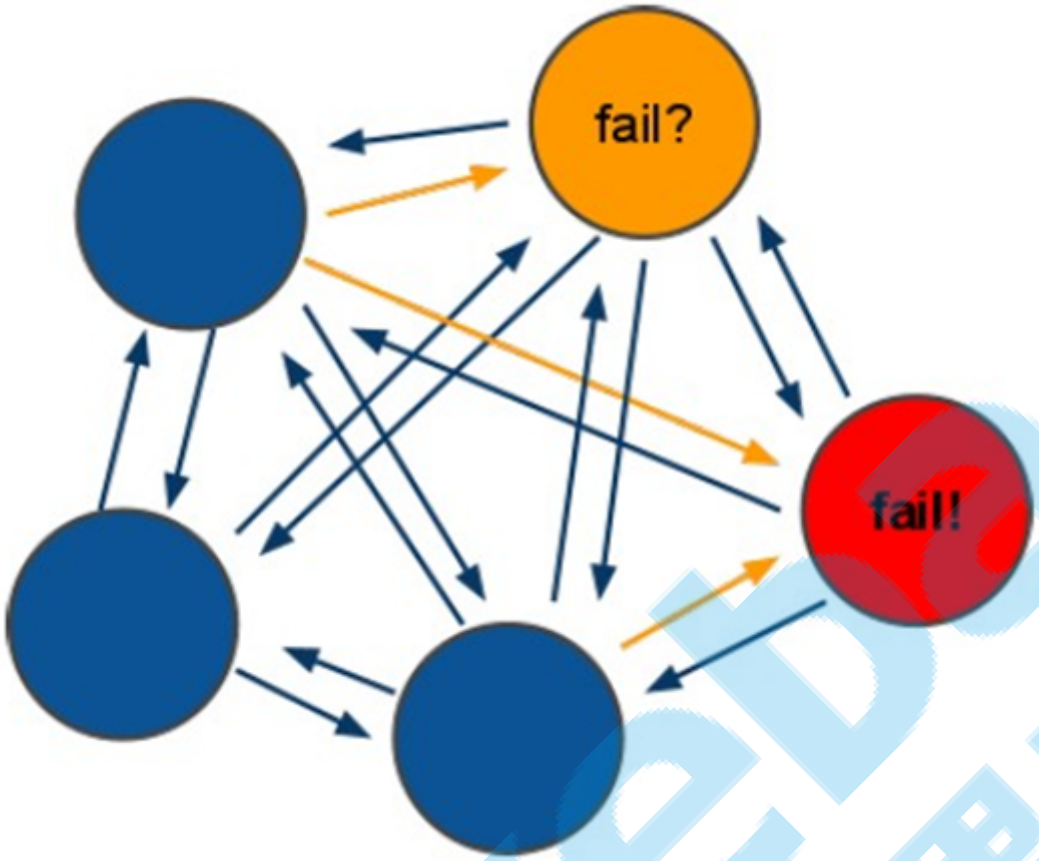
Redis集群

Redis的集群策略

Redis-cluster架构图



Redis-cluster投票:容错



安装RedisCluster

安装RedisCluster

> Redis集群最少需要三台主服务器，三台从服务器。
>
> 端口号分别为：7001~7006

- 第一步：创建7001实例，并编辑redis.conf文件，修改port为7001。

  注意：创建实例，即拷贝单机版安装时，生成的bin目录，为7001目录。

```
# Accept connections on the specified port, default is 6379.
# If port 0 is specified Redis will not listen on a TCP socket.
port 7001
```

- 第二步：修改redis.conf配置文件，打开Cluster-enable yes

```
################################ REDIS CLUSTER  ###############################
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# WARNING EXPERIMENTAL: Redis Cluster is considered to be stable code, however
# in order to mark it as "mature" we need to wait for a non trivial percentage
# of users to deploy it in production.
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Normal Redis instances can't be part of a Redis Cluster; only nodes that are
# started as cluster nodes can. In order to start a Redis instance as a
# cluster node enable the cluster support uncommenting the following:
#
cluster-enabled yes

# Every cluster node has a cluster configuration file. This file is not
# intended to be edited by hand. It is created and updated by Redis nodes.
# Every Redis Cluster node requires a different cluster configuration file.
# Make sure that instances running in the same system do not have
# overlapping cluster configuration file names.
#
```

- 第三步：复制7001，创建7002~7006实例，**注意端口修改。**
- 第四步：启动所有的实例
- 第五步：创建Redis集群

```
./redis-cli --cluster create 192.168.10.135:7001 192.168.10.135:7002
192.168.10.135:7003 192.168.10.135:7004 192.168.10.135:7005
192.168.10.135:7006 --cluster-replicas 1
>>> Creating cluster
Connecting to node 192.168.10.133:7001: OK
Connecting to node 192.168.10.133:7002: OK
Connecting to node 192.168.10.133:7003: OK
Connecting to node 192.168.10.133:7004: OK
Connecting to node 192.168.10.133:7005: OK
Connecting to node 192.168.10.133:7006: OK
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
192.168.10.133:7001
192.168.10.133:7002
192.168.10.133:7003
Adding replica 192.168.10.133:7004 to 192.168.10.133:7001
Adding replica 192.168.10.133:7005 to 192.168.10.133:7002
Adding replica 192.168.10.133:7006 to 192.168.10.133:7003
M: d8f6a0e3192c905f0aad411946f3ef9305350420 192.168.10.133:7001
   slots:0-5460 (5461 slots) master
M: 7a12bc730ddc939c84a156f276c446c28acf798c 192.168.10.133:7002
   slots:5461-10922 (5462 slots) master
M: 93f73d2424a796657948c660928b71edd3db881f 192.168.10.133:7003
   slots:10923-16383 (5461 slots) master
S: f79802d3da6b58ef6f9f30c903db7b2f79664e61 192.168.10.133:7004
   replicates d8f6a0e3192c905f0aad411946f3ef9305350420
S: 0bc78702413eb88eb6d7982833a6e040c6af05be 192.168.10.133:7005
   replicates 7a12bc730ddc939c84a156f276c446c28acf798c
S: 4170a68ba6b7757e914056e2857bb84c5e10950e 192.168.10.133:7006
   replicates 93f73d2424a796657948c660928b71edd3db881f
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 192.168.10.133:7001)
M: d8f6a0e3192c905f0aad411946f3ef9305350420 192.168.10.133:7001
   slots:0-5460 (5461 slots) master
M: 7a12bc730ddc939c84a156f276c446c28acf798c 192.168.10.133:7002
   slots:5461-10922 (5462 slots) master
M: 93f73d2424a796657948c660928b71edd3db881f 192.168.10.133:7003
   slots:10923-16383 (5461 slots) master
M: f79802d3da6b58ef6f9f30c903db7b2f79664e61 192.168.10.133:7004
   slots: (0 slots) master
   replicates d8f6a0e3192c905f0aad411946f3ef9305350420
M: 0bc78702413eb88eb6d7982833a6e040c6af05be 192.168.10.133:7005
   slots: (0 slots) master
   replicates 7a12bc730ddc939c84a156f276c446c28acf798c
M: 4170a68ba6b7757e914056e2857bb84c5e10950e 192.168.10.133:7006
   slots: (0 slots) master
   replicates 93f73d2424a796657948c660928b71edd3db881f
[OK] All nodes agree about slots configuration.
```

```
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@localhost-0723 redis]#
```

命令客户端连接集群

命令：

```
./redis-cli -h 127.0.0.1 -p 7001 -c
```

注意：**-c** 表示是以redis集群方式进行连接

```
./redis-cli -p 7006 -c
127.0.0.1:7006> set key1 123
-> Redirected to slot [9189] located at 127.0.0.1:7002
OK
127.0.0.1:7002>
```

查看集群的命令

- 查看集群状态

```
127.0.0.1:7003> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:6
cluster_my_epoch:3
cluster_stats_messages_sent:926
cluster_stats_messages_received:926
```

- 查看集群中的节点：

```
127.0.0.1:7003> cluster nodes
7a12bc730ddc939c84a156f276c446c28acf798c 127.0.0.1:7002 master - 0 1443601739754
2 connected 5461-10922
93f73d2424a796657948c660928b71edd3db881f 127.0.0.1:7003 myself,master - 0 0 3
connected 10923-16383
d8f6a0e3192c905f0aad411946f3ef9305350420 127.0.0.1:7001 master - 0 1443601741267
1 connected 0-5460
4170a68ba6b7757e914056e2857bb84c5e10950e 127.0.0.1:7006 slave
93f73d2424a796657948c660928b71edd3db881f 0 1443601739250 6 connected
f79802d3da6b58ef6f9f30c903db7b2f79664e61 127.0.0.1:7004 slave
d8f6a0e3192c905f0aad411946f3ef9305350420 0 1443601742277 4 connected
0bc78702413eb88eb6d7982833a6e040c6af05be 127.0.0.1:7005 slave
7a12bc730ddc939c84a156f276c446c28acf798c 0 1443601740259 5 connected
127.0.0.1:7003>
```

维护节点

集群创建成功后可以继续向集群中添加节点

添加主节点

- 先创建7007节点
- 添加7007结点作为新节点

  执行命令：

```
./redis-cli --cluster add-node 127.0.0.1:7007 127.0.0.1:7001
```

```
[root@server01 7007]# ./redis-trib.rb add-node  192.168.101.3:7007 192.168.101.3:7001
>>> Adding node 192.168.101.3:7007 to cluster 192.168.101.3:7001
Connecting to node 192.168.101.3:7001: OK
Connecting to node 192.168.101.3:7003: OK
Connecting to node 192.168.101.3:7006: OK
Connecting to node 192.168.101.3:7002: OK
Connecting to node 192.168.101.3:7005: OK
Connecting to node 192.168.101.3:7004: OK
>>> Performing Cluster Check (using node 192.168.101.3:7001)
M: cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 192.168.101.3:7001
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
M: 1a8420896c3ff60b70c716e8480de8e50749ee65 192.168.101.3:7003
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
S: 444e7bedbdfa40714ee55cd3086b8f0d5511fe54 192.168.101.3:7006
   slots: (0 slots) slave
   replicates 1a8420896c3ff60b70c716e8480de8e50749ee65
M: 4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841 192.168.101.3:7002
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: d2421a820cc23e17a01b597866fd0f750b698ac5 192.168.101.3:7005
   slots: (0 slots) slave
   replicates 4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841
S: 69d94b4963fd94f315fba2b9f12fae1278184fe8 192.168.101.3:7004
   slots: (0 slots) slave
   replicates cad9f7413ec6842c971dbcc2c48b4ca959eb5db4
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
Connecting to node 192.168.101.3:7007: OK
>>> Send CLUSTER MEET to node 192.168.101.3:7007 to make it join the cluster.
[OK] New node added correctly.
```

- 查看集群结点发现7007已添加到集群中

```
192.168.101.3:7005> cluster nodes
69d94b4963fd94f315fba2b9f12fae1278184fe8 192.168.101.3:7004 slave cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 0 1430155626174 4 connected
444e7bedbdfa40714ee55cd3086b8f0d5511fe54 192.168.101.3:7006 slave 1a8420896c3ff60b70c716e8480de8e50749ee65 0 1430155621629 9 connected
4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841 192.168.101.3:7002 master - 0 1430155627185 2 connected 5461-10922
d2421a820cc23e17a01b597866fd0f750b698ac5 192.168.101.3:7005 myself,slave 4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841 0 0 5 connected
1a8420896c3ff60b70c716e8480de8e50749ee65 192.168.101.3:7003 master - 0 1430155625164 9 connected 10923-16383
15b809eadae88955e36bcdbb8144f61bbbaf38fb 192.168.101.3:7007 master - 0 1430155628700 0 connected
cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 192.168.101.3:7001 master - 0 1430155628197 1 connected 0-5460
```

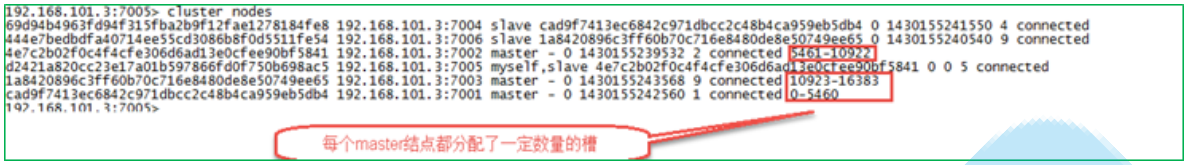
hash槽重新分配（数据迁移）

添加完主节点需要对主节点进行hash槽分配，这样该主节才可以存储数据。

- **查看集群中槽占用情况**

```
cluster nodes
```

redis集群有16384个槽，集群中的每个结点分配自已槽，通过查看集群结点可以看到槽占用情况。

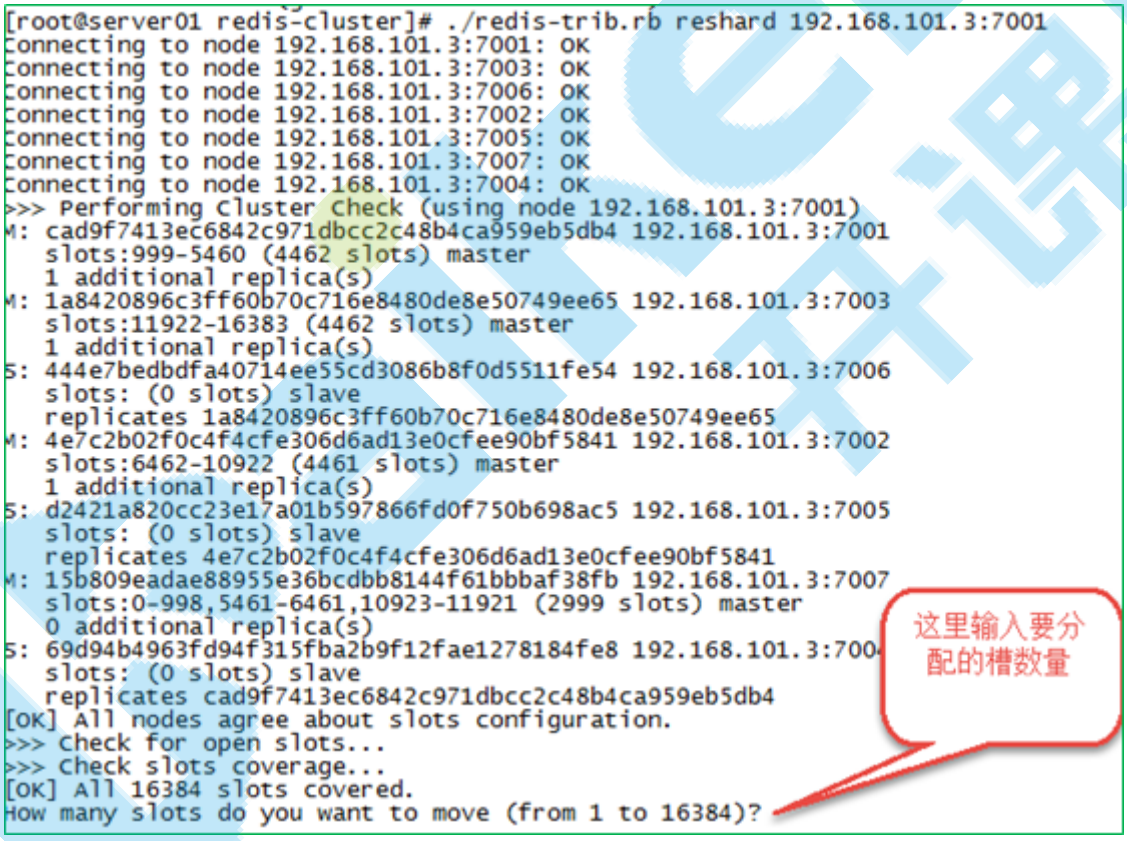


给刚添加的7007结点分配槽

- **第一步：连接上集群（连接集群中任意一个可用结点都行）**

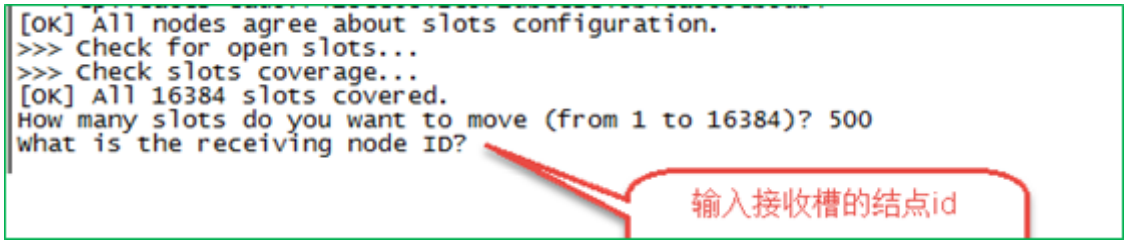
```
./redis-cli --cluster reshard 127.0.0.1:7007
```

- **第二步：输入要分配的槽数量**



输入：**3000**，表示要给目标节点分配3000个槽

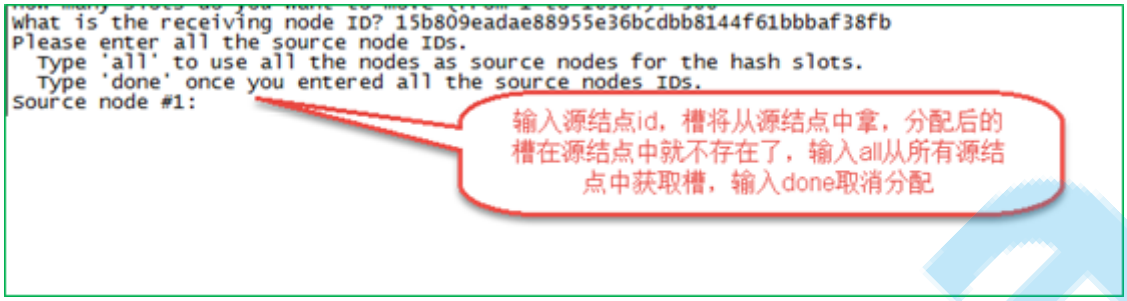
- **第三步：输入接收槽的结点id**



输入：**15b809eadae88955e36bcdbb8144f61bbbaf38fb**

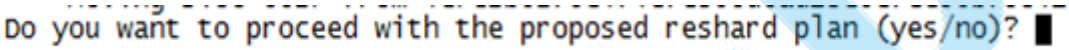
```
PS：这里准备给7007分配槽，通过cluster nodes查看7007结点id为：

15b809eadae88955e36bcdbb8144f61bbbaf38fb
```

- **第四步：输入源结点id**



输入：**all**

- **第五步：输入yes开始移动槽到目标结点id**



输入：**yes**

添加从节点

- **添加7008从结点，将7008作为7007的从结点**

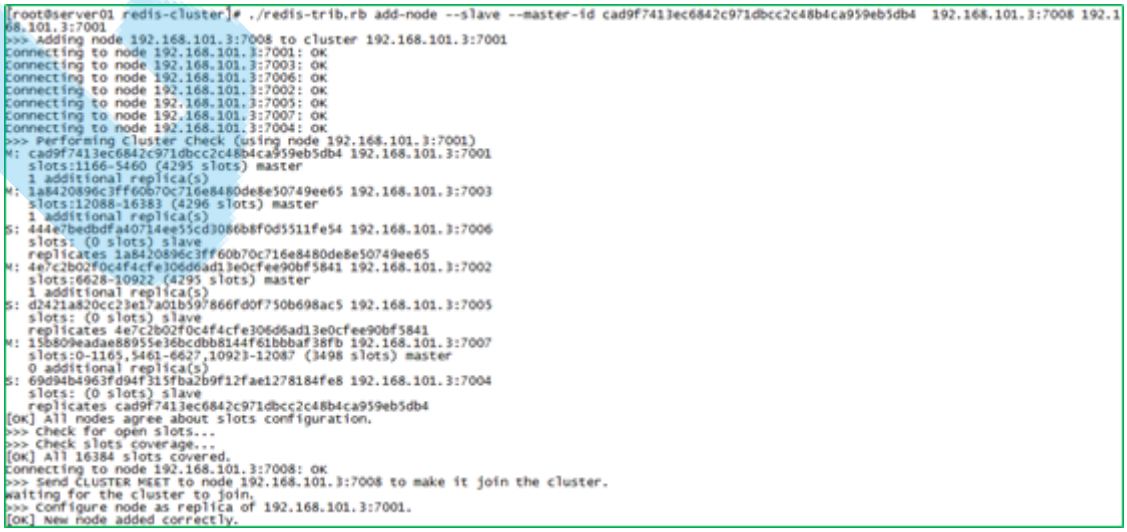
命令：

```
./redis-cli --cluster add-node 新节点的ip和端口    旧节点ip和端口 --cluster-slave --cluster-master-id 主节点id
```

例如：

```
./redis-cli --cluster add-node 127.0.0.1:7008 127.0.0.1:7007 --cluster-slave --cluster-master-id d1ba0092526cdfe66878e8879d446acfdcde25d8
```

d1ba0092526cdfe66878e8879d446acfdcde25d8是7007结点的id，可通过cluster nodes查看。



注意：如果原来该结点在集群中的配置信息已经生成到cluster-config-file指定的配置文件中（如果cluster-config-file没有指定则默认为**nodes.conf**），这时可能会报错：

```
[ERR] Node XXXXXX is not empty. Either the node already knows other nodes (check with CLUSTER NODES) or contains some key in database 0
```

解决方法是删除生成的配置文件nodes.conf，删除后再执行./redis-trib.rb add-node指令

- **查看集群中的结点，刚添加的7008为7007的从节点：**

```
192.168.101.3:7005> cluster nodes
05dbaf8059630157c245dfe5441dbe9c26b9016d 192.168.101.3:7008 slave cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 0 1430157051979 1 connected
69d94b4963fd94f315fba2b9f12fae1278184fe8 192.168.101.3:7004 slave cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 0 1430157046926 4 connected
444e7bedbdfa40714ee55cd3086b8f0d5511fe54 192.168.101.3:7006 slave 1a8420896c3ff60b70c716e8480de8e50749ee65 0 1430157051878 9 connected
4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841 192.168.101.3:7002 master - 0 1430157049956 2 connected 6628-10922
d2421a820cc23e17a01b597866fd0f750b698ac5 192.168.101.3:7005 myself,slave 4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841 0 0 5 connected
1a8420896c3ff60b70c716e8480de8e50749ee65 192.168.101.3:7003 master - 0 1430157050968 9 connected 12088-16383
15b809eadae88955e36bcdbb8144f61bbbaf38fb 192.168.101.3:7007 master - 0 1430157052990 10 connected 0-1165 5461-6627 10923-12087
cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 192.168.101.3:7001 master - 0 1430157048946 1 connected 1166-5460
```

删除结点

命令：

```
./redis-cli --cluster del-node 127.0.0.1:7008
41592e62b83a8455f07f7797f1d5c071cffedb50
```

删除已经占有hash槽的结点会失败，报错如下：

```
[ERR] Node 127.0.0.1:7005 is not empty! Reshard data away and try again.
```

需要将该结点占用的hash槽分配出去（参考hash槽重新分配章节）。

Jedis连接集群

需要开启防火墙，或者直接关闭防火墙。

```
service iptables stop
```

代码实现

创建JedisCluster类连接redis集群。

```
@Test
public void testJedisCluster() throws Exception {
    //创建一连接，JedisCluster对象,在系统中是单例存在
    Set<HostAndPort> nodes = new HashSet<>();
    nodes.add(new HostAndPort("192.168.10.133", 7001));
    nodes.add(new HostAndPort("192.168.10.133", 7002));
    nodes.add(new HostAndPort("192.168.10.133", 7003));
    nodes.add(new HostAndPort("192.168.10.133", 7004));
    nodes.add(new HostAndPort("192.168.10.133", 7005));
    nodes.add(new HostAndPort("192.168.10.133", 7006));
    JedisCluster cluster = new JedisCluster(nodes);
    //执行JedisCluster对象中的方法，方法和redis一一对应。
    cluster.set("cluster-test", "my jedis cluster test");
    String result = cluster.get("cluster-test");
    System.out.println(result);
    //程序结束时需要关闭JedisCluster对象
    cluster.close();
}
```

使用spring

Ø 配置applicationContext.xml

```
<!-- 连接池配置 -->
<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
    <!-- 最大连接数 -->
    <property name="maxTotal" value="30" />
    <!-- 最大空闲连接数 -->
    <property name="maxIdle" value="10" />
    <!-- 每次释放连接的最大数目 -->
    <property name="numTestsPerEvictionRun" value="1024" />
    <!-- 释放连接的扫描间隔（毫秒） -->
    <property name="timeBetweenEvictionRunsMillis" value="30000" />
    <!-- 连接最小空闲时间 -->
    <property name="minEvictableIdleTimeMillis" value="1800000" />
    <!-- 连接空闲多久后释放, 当空闲时间>该值 且 空闲连接>最大空闲连接数 时直接释放 -->
    <property name="softMinEvictableIdleTimeMillis" value="10000" />
    <!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -->
    <property name="maxWaitMillis" value="1500" />
    <!-- 在获取连接的时候检查有效性, 默认false -->
    <property name="testOnBorrow" value="true" />
    <!-- 在空闲时检查有效性, 默认false -->
    <property name="testWhileIdle" value="true" />
    <!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true -->
    <property name="blockWhenExhausted" value="false" />
</bean>
<!-- redis集群 -->
<bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">
    <constructor-arg index="0">
        <set>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.101.3"></constructor-
arg>
                <constructor-arg index="1" value="7001"></constructor-arg>
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.101.3"></constructor-
arg>
                <constructor-arg index="1" value="7002"></constructor-arg>
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.101.3"></constructor-
arg>
                <constructor-arg index="1" value="7003"></constructor-arg>
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.101.3"></constructor-
arg>
                <constructor-arg index="1" value="7004"></constructor-arg>
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.101.3"></constructor-
arg>
                <constructor-arg index="1" value="7005"></constructor-arg>
```

```
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.101.3"></constructor-
arg>
                <constructor-arg index="1" value="7006"></constructor-arg>
            </bean>
        </set>
    </constructor-arg>
    <constructor-arg index="1" ref="jedisPoolConfig"></constructor-arg>
</bean>
```

Ø 测试代码

```
private ApplicationContext applicationContext;
    @Before
    public void init() {
        applicationContext = new ClassPathXmlApplicationContext(
                "classpath:applicationContext.xml");
    }

    // redis集群
    @Test
    public void testJedisCluster() {
        JedisCluster jedisCluster = (JedisCluster) applicationContext
                .getBean("jedisCluster");

        jedisCluster.set("name", "zhangsan");
        String value = jedisCluster.get("name");
        System.out.println(value);
    }
```