课前准备

- 准备redis安装包

课堂主题

Redis集群原理、Redis和lua整合、Redis消息模式、Redis实现分布式锁、缓存穿透、缓存雪崩、缓存击穿、缓存双写一致性

课堂目标

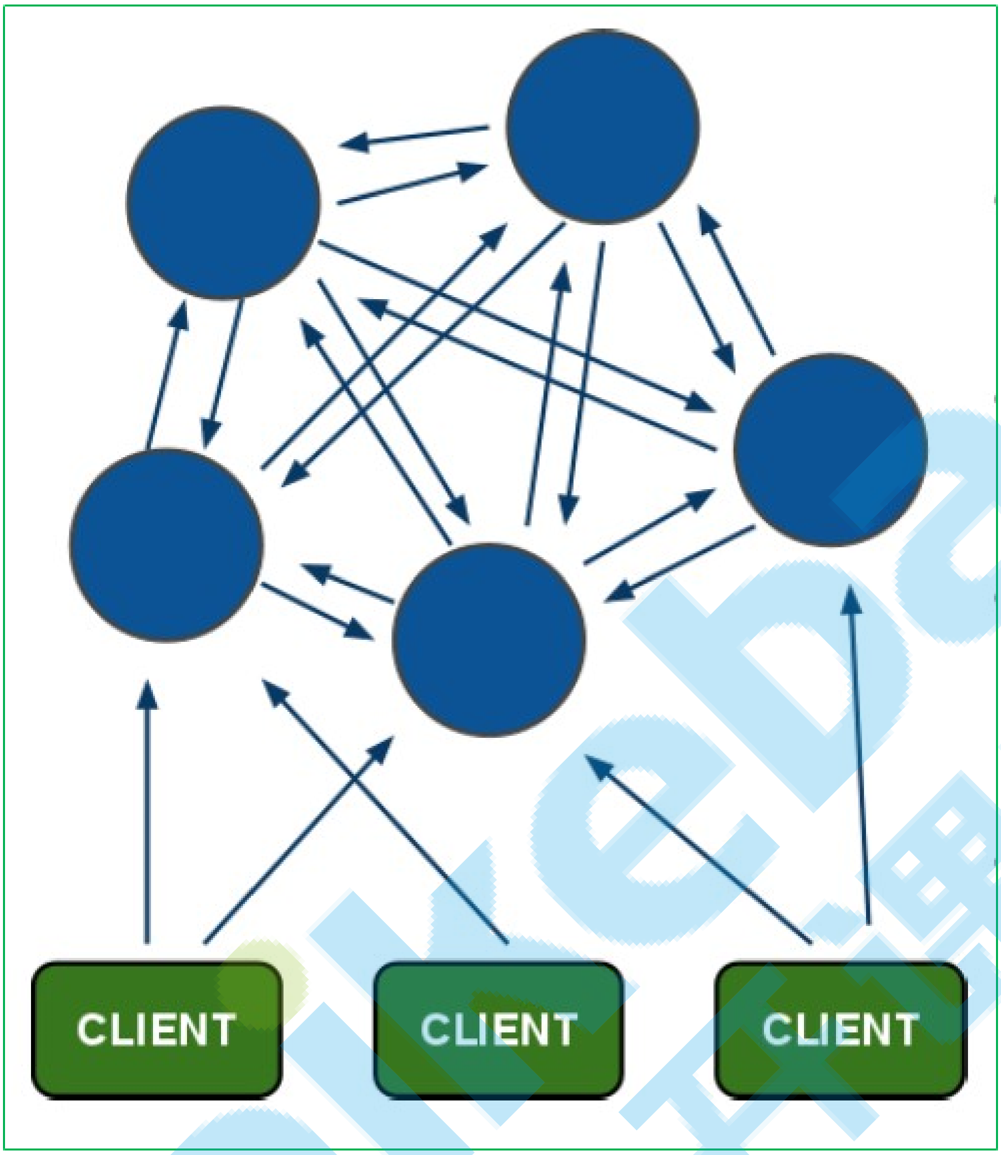
- 理解RedisCluster的原理和容错机制
- 能够配置RedisCluster并使用
- 理解lua概念，能够使用Redis和lua整合使用
- 理解redis消息原理
- 掌握redis分布式锁
- 理解缓存穿透、缓存雪崩、缓存击穿、缓存双写一致性并掌握解决方案

知识要点

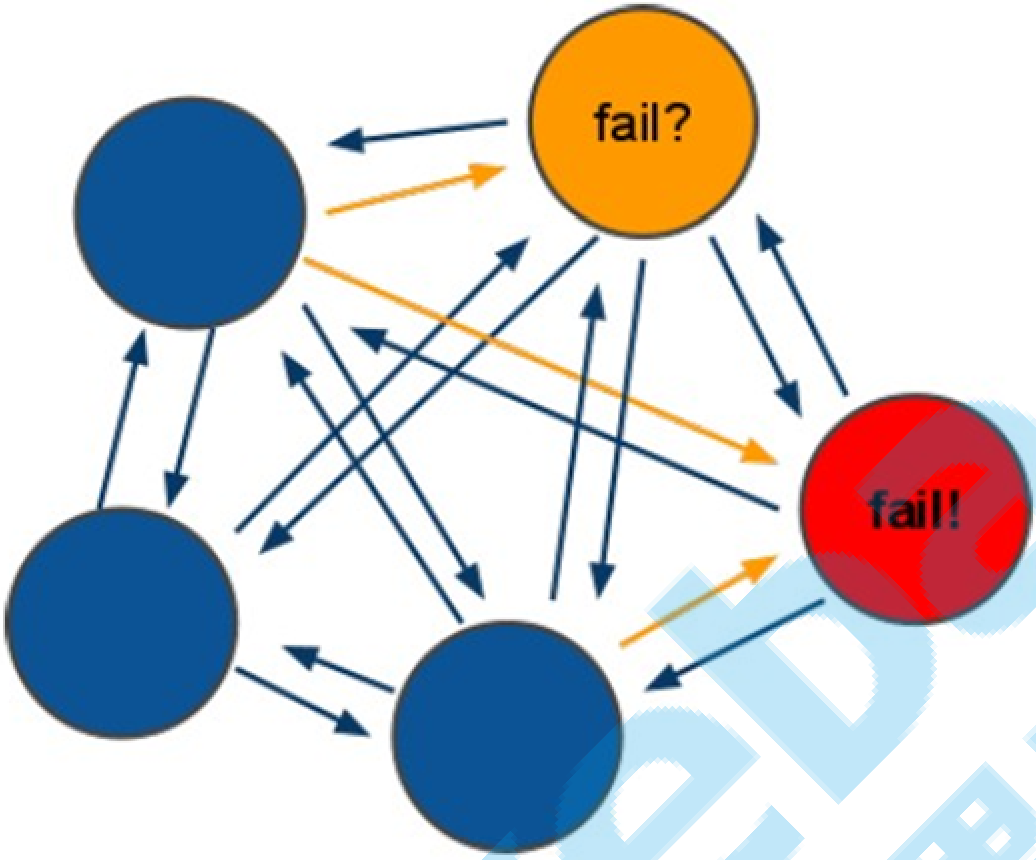
Redis集群

Redis的集群策略

Redis-cluster架构图



Redis-cluster投票:容错



安装RedisCluster

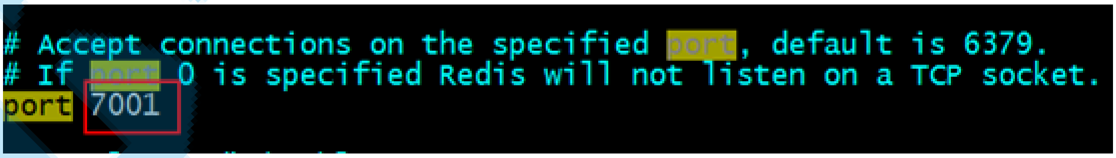
安装RedisCluster

```
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
```

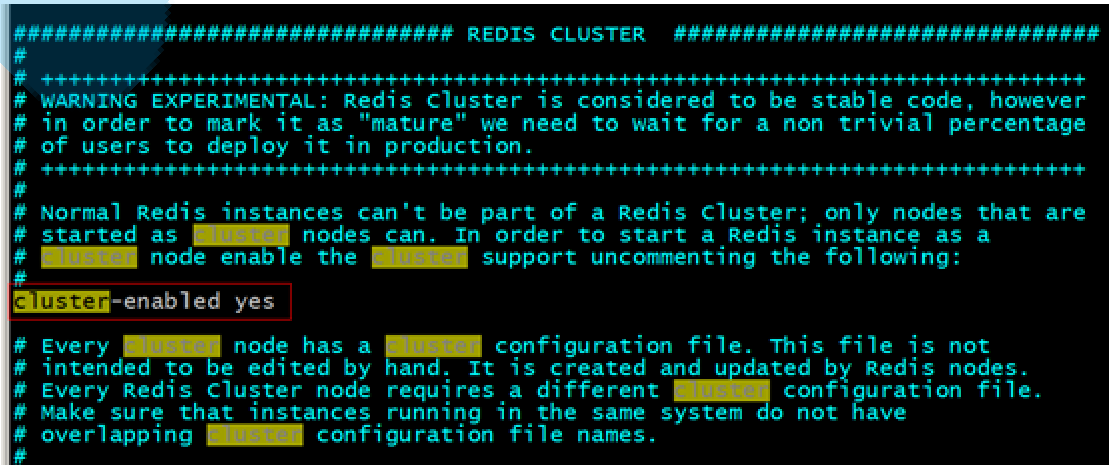
- 第一步：创建7001实例，并编辑redis.conf文件，修改port为7001。

  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

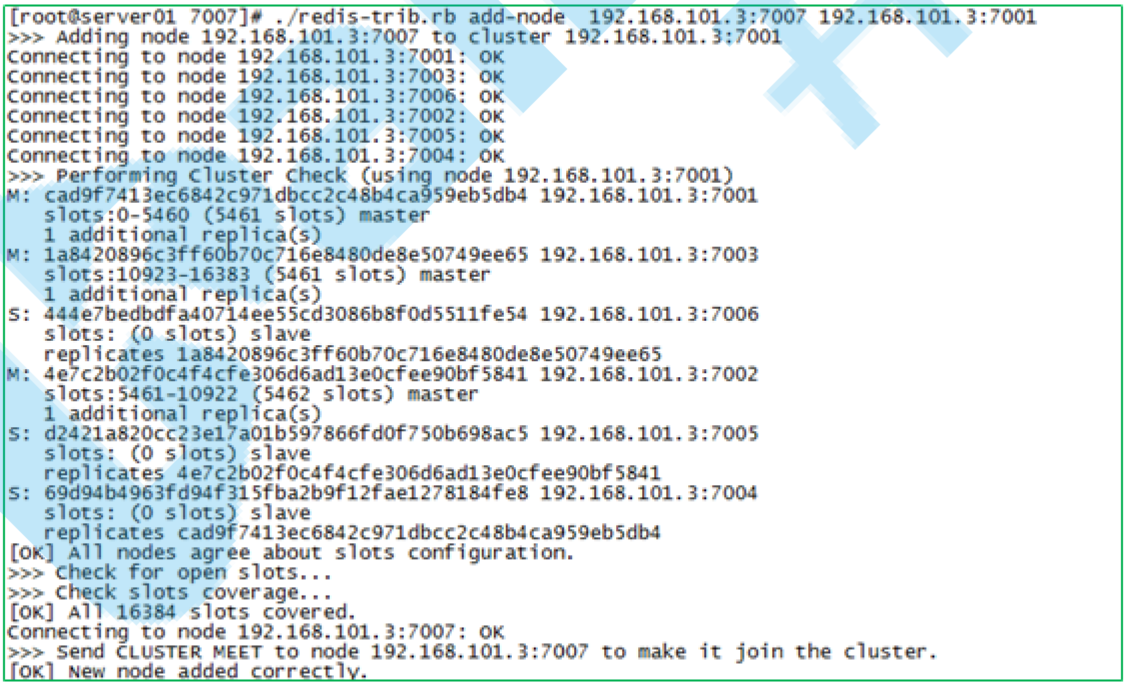
集群创建成功后可以继续向集群中添加节点

添加主节点

- 先创建7007节点
- 添加7007结点作为新节点

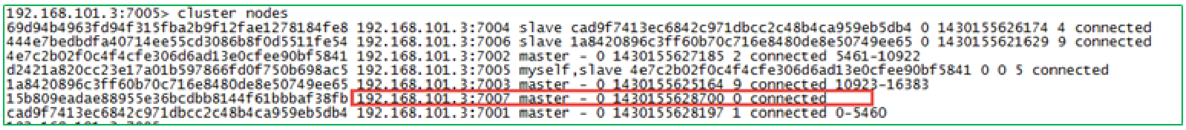
  执行命令：

```
./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
```

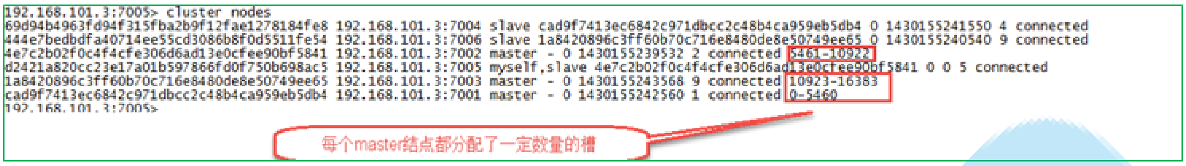
hash槽重新分配（数据迁移）

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。



```
192.168.101.3:7005> cluster nodes
69d94b4963fd94f315fba2b9f12fae1278184fe8 192.168.101.3:7004 slave cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 0 1430155241550 4 connected
444e7bedbdfa40714ee55cd3086b8f0d5511fe54 192.168.101.3:7006 slave 1a8420896c3ff60b70c716e8480de8e50749ee65 0 1430155240540 9 connected
4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841 192.168.101.3:7002 master - 0 1430155239532 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 1430155243568 9 connected 10923-16383
cad9f7413ec6842c971dbcc2c48b4ca959eb5db4 192.168.101.3:7001 master - 0 1430155242560 1 connected 0-5460
192.168.101.3:7005>
```

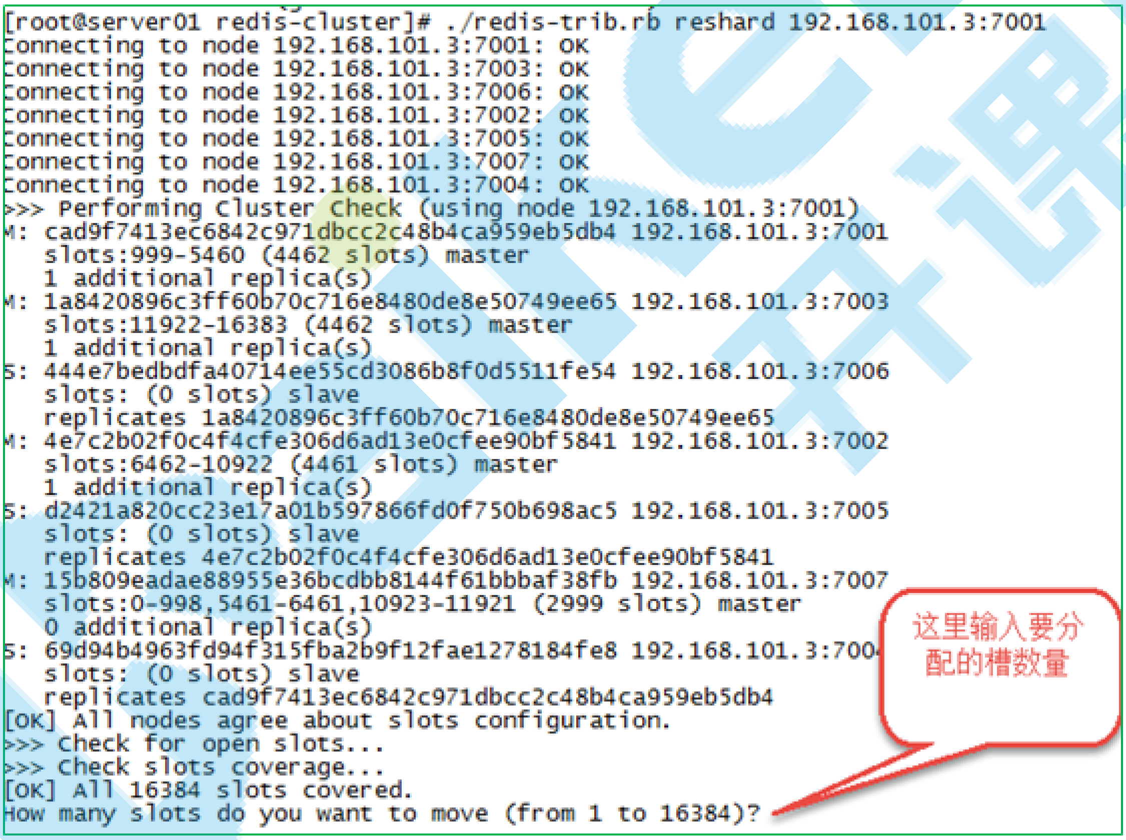
每个master结点都分配了一定数量的槽

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**

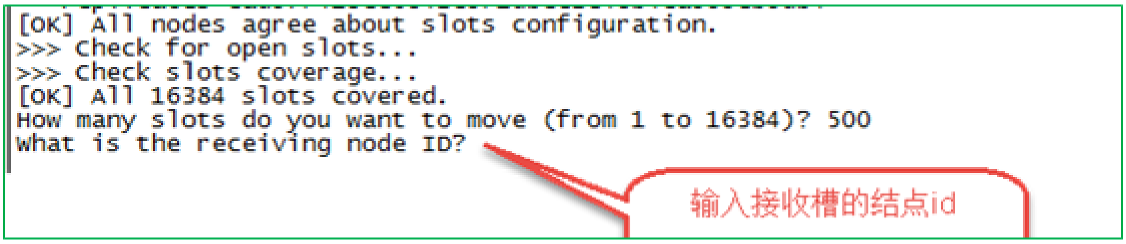


```
[root@server01 redis-cluster]# ./redis-trib.rb reshard 192.168.101.3:7001
Connecting to node 192.168.101.3:7001: OK
Connecting to node 192.168.101.3:7003: OK
Connecting to node 192.168.101.3:7006: OK
Connecting to node 192.168.101.3:7002: OK
Connecting to node 192.168.101.3:7005: OK
Connecting to node 192.168.101.3:7007: 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:999-5460 (4462 slots) master
   1 additional replica(s)
M: 1a8420896c3ff60b70c716e8480de8e50749ee65 192.168.101.3:7003
   slots:11922-16383 (4462 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:6462-10922 (4461 slots) master
   1 additional replica(s)
S: d2421a820cc23e17a01b597866fd0f750b698ac5 192.168.101.3:7005
   slots: (0 slots) slave
   replicates 4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841
M: 15b809eadae88955e36bcdbb8144f61bbbaf38fb 192.168.101.3:7007
   slots:0-998,5461-6461,10923-11921 (2999 slots) master
   0 additional replica(s)
S: 69d94b4963fd94f315fba2b9f12fae1278184fe8 192.168.101.3:700
   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.
How many slots do you want to move (from 1 to 16384)?
```

这里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**



```
[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)? 500
What is the receiving node ID?
```

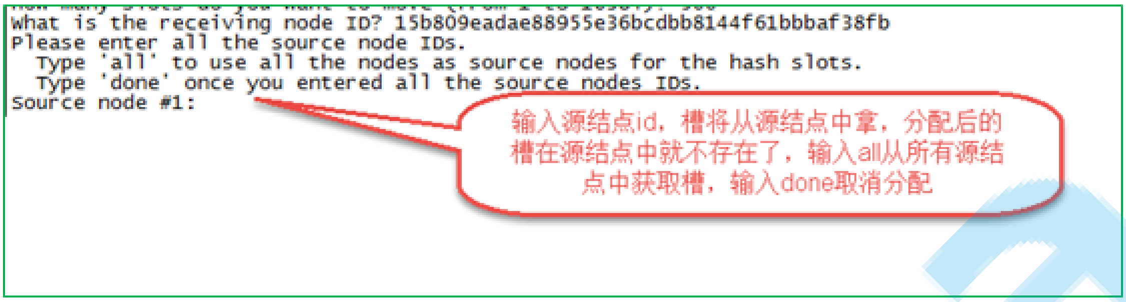
输入接收槽的结点id

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**

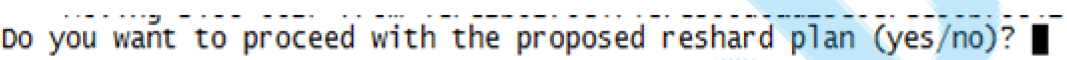


```
What is the receiving node ID? 15b809eadae88955e36bcdbb8144f61bbbaf38fb
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1:
```

输入源结点id，槽将从源结点中拿，分配后的槽在源结点中就不存在了，输入all从所有源结点中获取槽，输入done取消分配

输入：**all**

- **第五步：输入yes开始移动槽到目标结点id**



```
Do you want to proceed with the proposed reshard plan (yes/no)?
```

输入：**yes**

添加从节点

- **添加7008从结点，将7008作为7007的从结点**

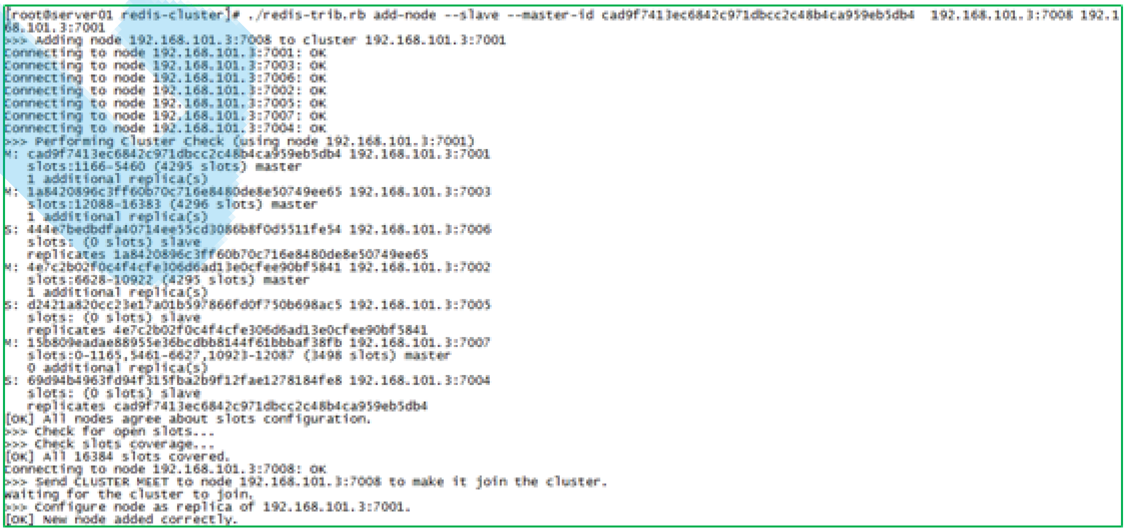
命令：

```
./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。



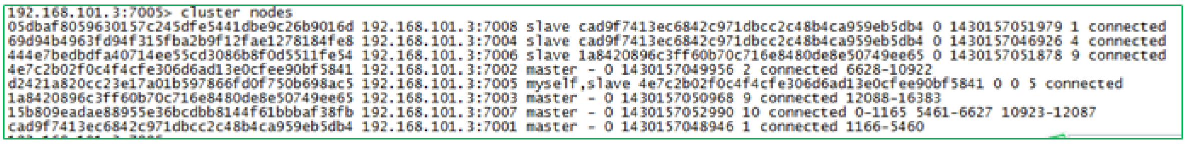
```
[root@server01 redis-cluster]# ./redis-trib.rb add-node --slave --master-id cad9f7413ec6842c971dbcc2c48b4ca959eb5db4  192.168.101.3:7008 192.1
68.101.3:7001
>>> Adding node 192.168.101.3:7008 to cluster 192.168.101.3:7001
Connecting to node 192.168.101.3:7001: OK
Connecting to node 192.168.101.3:7003: 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 192.168.101.3:7007: 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:1166-5460 (4295 slots) master
   1 additional replica(s)
M: 1a8420896c3ff60b70c716e8480de8e50749ee65 192.168.101.3:7003
   slots:12088-16383 (4296 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:6628-10922 (4295 slots) master
   1 additional replica(s)
S: d2421a820cc23e17a01b597866fd0f750b698ac5 192.168.101.3:7005
   slots: (0 slots) slave
   replicates 4e7c2b02f0c4f4cfe306d6ad13e0cfee90bf5841
M: 15b809eadae88955e36bcdbb8144f61bbbaf38fb 192.168.101.3:7007
   slots:0-1165,5461-6627,10923-12087 (3498 slots) master
   0 additional replica(s)
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 192.168.101.3:7008: OK
>>> Send CLUSTER MEET to node 192.168.101.3:7008 to make it join the cluster.
Waiting for the cluster to join.
>>> Configure node as replica of 192.168.101.3:7001.
[OK] New node added correctly.
```

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);
    }
```

Redis和lua整合

什么是lua

Redis中使用lua的好处

lua的安装（了解）

lua常见语法（了解）

Redis整合lua脚本

EVAL命令

lua脚本中调用Redis命令

EVALSHA

SCRIPT命令

redis-cli --eval

Redis消息模式

Redis实现分布式锁

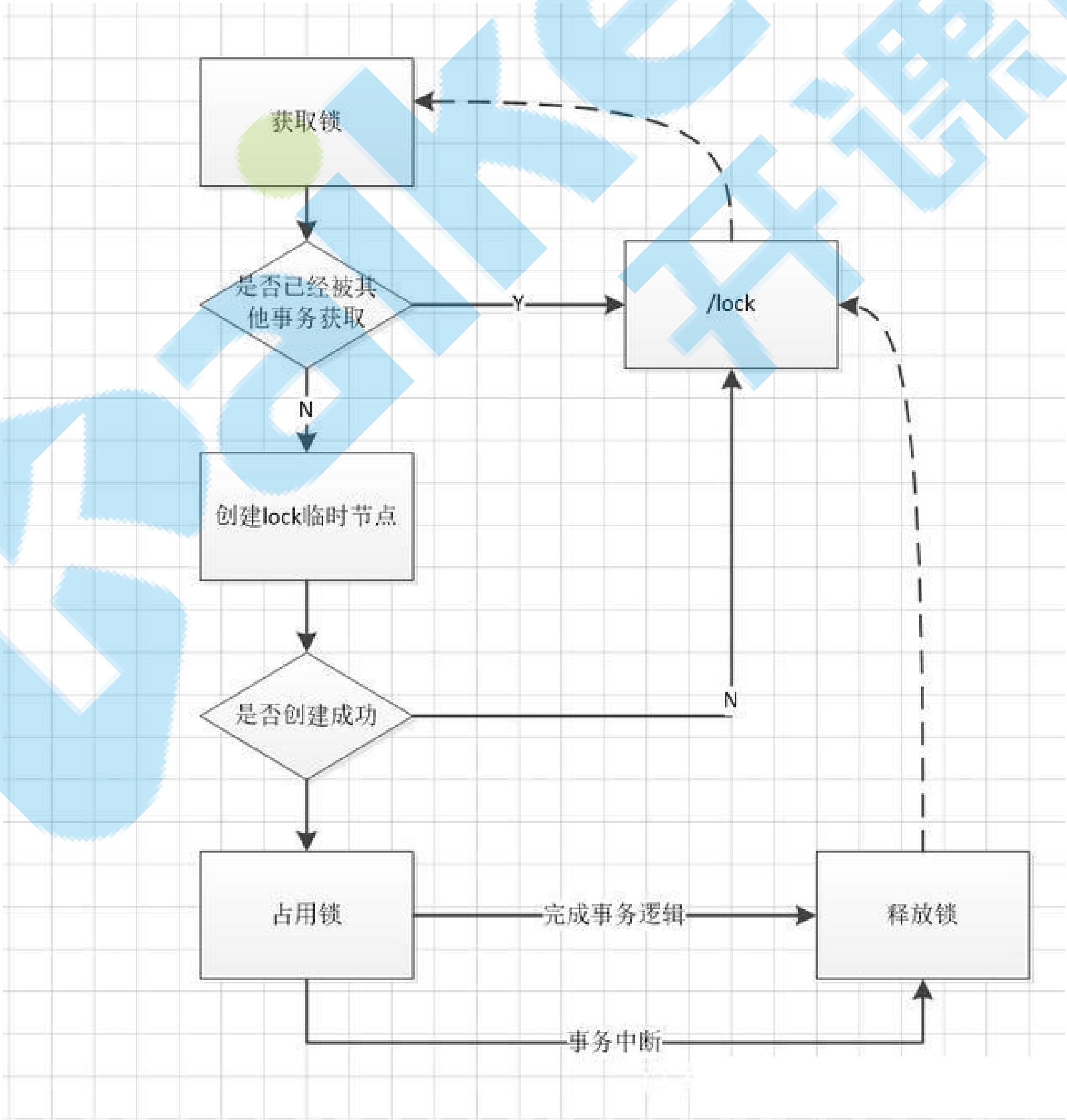
锁的处理

单应用中使用锁：（单进程多线程）

分布式应用中使用锁：（多进程多线程）

分布式锁的实现方式

- 基于数据库的乐观锁实现分布式锁
- 基于`zookeeper`临时节点的分布式锁



- 基于`Redis`的分布式锁

实现分布式锁

获取锁

方式1（**使用set命令实现**）--推荐

方式2（**使用setnx命令实现**） -- 并发会产生问题

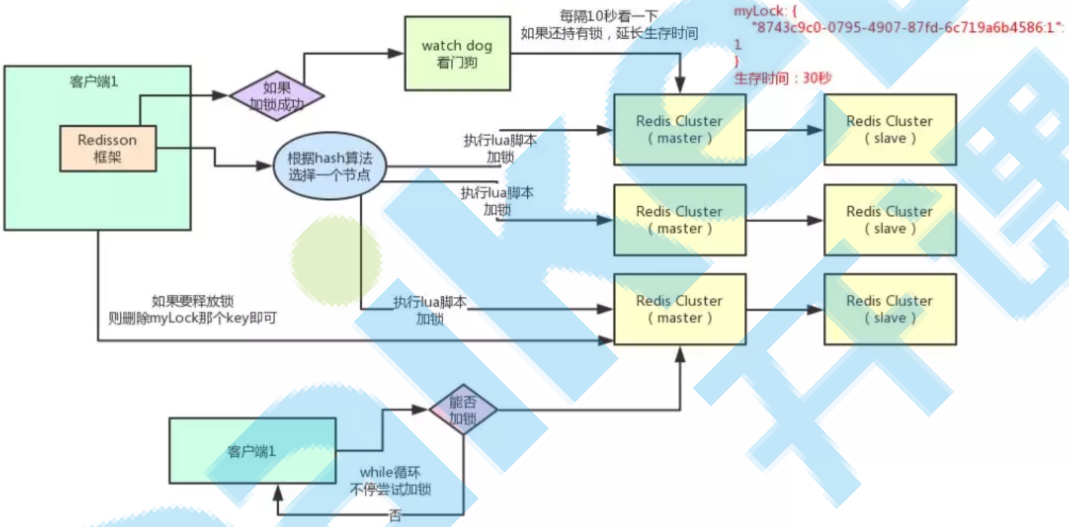
释放锁

方式1（del命令实现）

方式2（**redis+lua脚本实现**）--推荐

生产环境中的分布式锁

落地生产环境用分布式锁，一般采用开源框架，比如Redisson。下面来讲一下Redisson对Redis分布式锁的实现原理。



常见缓存问题

缓存穿透

一般的缓存系统，都是按照key去缓存查询，如果不存在对应的value，就应该去后端系统查找（比如DB）。如果key对应的value是一定不存在的，并且对该key并发请求量很大，就会对后端系统造成很大的压力。

也就是说，对不存在的key进行高并发访问，导致数据库压力瞬间增大，这就叫做【缓存穿透】。

缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，也会给后端系统(比如DB)带来很大压力。

缓存击穿

对于一些设置了过期时间的key，如果这些key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题，这个和缓存雪崩的区别在于这里针对某一key缓存，前者则是很多key。

缓存在某个时间点过期的时候，恰好在这个时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端DB压垮。

缓存双写一致性

先更新数据库再更新缓存(不建议使用)

先更新数据库再删除缓存

先删除缓存再更新数据库