高可用架构篇

Redis 集群的高可用测试（含 Jedis 客户端的使用）

注意：本节教程内容紧接上一节教程《Dubbo 视频教程--高可用架构篇--第 05 节--Redis 集群的安装（Redis3+CentOS）》的内容

Redis 集群的使用测试（Jedis 客户端的使用）

1、 Jedis 客户端建议升级到最新版（当前为 2.7.3），这样对 3.0.x 集群有比较好的支持。

https://github.com/xetorthio/jedis

http://mvnrepository.com/artifact/redis.clients/jedis

2、 直接在 Java 代码中链接 Redis 集群：

```
// 数据库链接池配置
JedisPoolConfig config = new JedisPoolConfig();
config.setMaxTotal(100);
config.setMaxIdle(50);
config.setMinIdle(20);
config.setMaxWaitMillis(6 * 1000);
config.setTestOnBorrow(true);
// Redis集群的节点集合
Set<HostAndPort> jedisClusterNodes = new HashSet<HostAndPort>();
jedisClusterNodes.add(new HostAndPort("192.168.1.111", 7111));
jedisClusterNodes.add(new HostAndPort("192.168.1.112", 7112));
jedisClusterNodes.add(new HostAndPort("192.168.1.113", 7113));
jedisClusterNodes.add(new HostAndPort("192.168.1.114", 7114));
jedisClusterNodes.add(new HostAndPort("192.168.1.115", 7115));
jedisClusterNodes.add(new HostAndPort("192.168.1.116", 7116));
// 根据节点集创集群链接对象
//JedisCluster jedisCluster = new JedisCluster(jedisClusterNodes);
// 节点，超时时间，最多重定向次数，链接池
JedisCluster jedisCluster = new JedisCluster(jedisClusterNodes, 2000, 100, config);
int num = 1000;
String key = "wusc";
String value = "";
for (int i=1; i <= num; i++){
      // 存数据
      jedisCluster.set(key+i, "WuShuicheng"+i);
      // 取数据
      value = jedisCluster.get(key+i);
      log.info(key+i + "=" + value);
      // 删除数据
      //jedisCluster.del(key+i);
      //value = jedisCluster.get(key+i);
      //log.info(key+i + "=" + value);
```

```
}
```

3、Spring 配置 Jedis 链接 Redis3.0 集群的配置：

```
<!-- Jedis链接池配置，注意：Jedis版本建议升级到最新（当前最新版为2.7.2） -->
<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
    <property name="maxTotal" value="100" />
    <property name="maxIdle" value="20" />
    <property name="minIdle" value="10" />
    <property name="blockWhenExhausted" value="true"></property>
    <property name="maxWaitMillis" value="3000" />
    <property name="testOnBorrow" value="false" />
    <property name="testOnReturn" value="false" />
    <property name="testWhileIdle" value="true" />
    <property name="minEvictableIdleTimeMillis" value="60000" />
    <property name="timeBetweenEvictionRunsMillis" value="30000" />
    <property name="numTestsPerEvictionRun" value="-1" />
</bean>
<!-- JedisCluster -->
<bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">
    <constructor-arg index="0">
        <set>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.1.111" />
                <constructor-arg index="1" value="7111" type="int" />
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.1.112" />
                <constructor-arg index="1" value="7112" type="int" />
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.1.113" />
                <constructor-arg index="1" value="7113" type="int" />
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.1.114" />
                <constructor-arg index="1" value="7114" type="int" />
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.1.115" />
                <constructor-arg index="1" value="7115" type="int" />
            </bean>
            <bean class="redis.clients.jedis.HostAndPort">
                <constructor-arg index="0" value="192.168.1.116" />
                <constructor-arg index="1" value="7116" type="int" />
            </bean>
```

```
            </set>
        </constructor-arg>
        <constructor-arg index="1" value="2000" type="int"></constructor-arg>
        <constructor-arg index="2" value="100" type="int"></constructor-arg>
        <constructor-arg index="3" ref="jedisPoolConfig"></constructor-arg>
    </bean>
```

对应的 Java 调用代码样例（详细代码请看视频教程提供的 demo 源码）：

```
JedisCluster jedisCluster = (JedisCluster) context.getBean("jedisCluster");
int num = 1000;
String key = "wusc";
String value = "";
for (int i=1; i <= num; i++){
    // 存数据
    jedisCluster.set(key+i, "WuShuicheng"+i);
    // 取数据
    value = jedisCluster.get(key+i);
    log.info(key+i + "=" + value);
    // 删除数据
    //jedisCluster.del(key+i);
}
```

4、测试操作，请看视频教程。

Redis 集群的高可用性测试

一、Redis 集群特点

1、集群架构特点:

(1)所有的 redis 节点彼此互联(PING-PONG 机制)，内部使用二进制协议优化传输速度和带宽；

(2)节点的 fail 是通过集群中超过半数的节点检测失效时才生效；

(3)客户端与 redis 节点直连，不需要中间 proxy 层。客户端不需要连接集群所有节点，连接集群中任何一个可用节点即可；

(4)redis-cluster 把所有的物理节点映射到[0-16383]个 slot(哈希槽)上，cluster 负责维护 node<->slot<->value 。

2、集群选举容错:

(1)节点失效选举过程是集群中所有 master 参与,如果半数以上 master 节点与当前被检测 master 节点通信检测超时(cluster-node-timeout)，就认为当前 master 节点挂掉；

(2):什么时候整个集群不可用(cluster_state:fail)?

A:如果集群任意 master 挂掉，且当前 master 没有 slave。集群进入 fail 状态,也可以理解成集群的 slot 映射[0-16383]不完整时进入 fail 状态。 ps : redis-3.0.0.rc1 加入 cluster-require-full-coverage 参数,默认关闭,打开集群兼容部分失败；

B:如果集群超过半数以上 master 挂掉，无论是否有 slave 集群进入 fail 状态。ps:当集群不可用时,所有对集群的操作做都不可用，收到((error) CLUSTERDOWN The cluster is down)错误。

二、客户端集群命令

集群

cluster info ：打印集群的信息

cluster nodes ：列出集群当前已知的所有节点（node），以及这些节点的相关信息。

节点

cluster meet <ip> <port> ：将 ip 和 port 所指定的节点添加到集群当中，让它成为集群的一份子。

cluster forget <node_id> ：从集群中移除 node_id 指定的节点。

cluster replicate <node_id> ：将当前节点设置为 node_id 指定的节点的从节点。

cluster saveconfig ：将节点的配置文件保存到硬盘里面。

槽(slot)

cluster addslots <slot> [slot ...] ：将一个或多个槽（slot）指派（assign）给当前节点。

cluster delslots <slot> [slot ...] ：移除一个或多个槽对当前节点的指派。

cluster flushslots ：移除指派给当前节点的所有槽，让当前节点变成一个没有指派任何槽的节点。

cluster setslot <slot> node <node_id> ：将槽 slot 指派给 node_id 指定的节点，如果槽已经指派给另一个节点，那么先让另一个节点删除该槽>，然后再进行指派。

cluster setslot <slot> migrating <node_id> ：将本节点的槽 slot 迁移到 node_id 指定的节点中。

cluster setslot <slot> importing <node_id> ：从 node_id 指定的节点中导入槽 slot 到本节点。

cluster setslot <slot> stable ：取消对槽 slot 的导入（import）或者迁移（migrate）。

键

cluster keyslot <key> ：计算键 key 应该被放置在哪个槽上。

cluster countkeysinslot <slot> ：返回槽 slot 目前包含的键值对数量。

cluster getkeysinslot <slot> <count> ：返回 count 个 slot 槽中的键。

三、集群高可用测试（主要看视频的操作与解说）

1、重建集群，步骤：

(1)关闭集群的各节点；

(2)删除各节点数据目录下的 nodes.conf、appendonly.aof、dump.rdb；

```
# rm -rf appendonly.aof | rm -rf dump.rdb | rm -rf nodes.conf
```

(3)重新启用所有的节点

192.168.1.111

```
# /usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7111/redis-7111.conf
```

192.168.1.112

```
# /usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7112/redis-7112.conf
```

192.168.1.113

```
# /usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7113/redis-7113.conf
```

192.168.1.114

```
# /usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7114/redis-7114.conf
```

192.168.1.115

```
# /usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7115/redis-7115.conf
```

192.168.1.116

```
# /usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7116/redis-7116.conf
```

(4)执行集群创建命令（只需要在其中一个节点上执行一次则可）

```
# cd /usr/local/src/redis-3.0.3/src/
```

```
# cp redis-trib.rb /usr/local/bin/redis-trib
# redis-trib create --replicas 1 192.168.1.114:7114 192.168.1.115:7115 192.168.1.116:7116
192.168.1.111:7111 192.168.1.112:7112 192.168.1.113:7113
```

2、查看当前集群各节点的状态

```
[root@edu-redis-01 7111]# /usr/local/redis3/bin/redis-cli -c -p 7111
127.0.0.1:7111> cluster nodes
```

3、使用 demo 应用向集群写入 1000 个键值数据

使用 /usr/local/redis3/bin/redis-cli -c -p 711X 命令登录各节点，使用 keys * 查看各节点的所有 key

4、运行 demo 应用，获取所有的键值数据

如果有空值则停止

5、模拟集群节点宕机(实现故障转移，可重点看视频解说)

(1)Jedis 客户端循环操作集群数据（模拟用户持续使用系统）

(2)查看 Redis 集群当前状态（用于接下来做节点状态变化对比）

```
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439134613117 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 slave 8dd55e9b4da9f62b9b15232e86553f1337864179 0 1439134614122 4 connected
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 master - 0 1439134616123 4 connected 10923-16383
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439134611112 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439134615120 6 connected 0-5460
127.0.0.1:7116>
```

(3)关闭其中一个 master 节点（7111）

(4)观察该 master 节点和对应的 slave 节点的状态变化（请看视频解说）

```
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439134754409 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 slave 8dd55e9b4da9f62b9b15232e86553f1337864179 0 1439134753407 4 connected
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 master - 1439134742175 1439134741373 4 disconnected 10923-16383
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439134755412 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439134756412 6 connected 0-5460
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439134754409 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 slave 8dd55e9b4da9f62b9b15232e86553f1337864179 0 1439134753407 4 connected
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 master,fail? - 1439134742175 1439134741373 4 disconnected 10923-16383
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439134757416 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439134756412 6 connected 0-5460
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439134754409 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 slave 8dd55e9b4da9f62b9b15232e86553f1337864179 0 1439134753407 4 connected
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 master,fail? - 1439134742175 1439134741373 4 disconnected 10923-16383
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439134757416 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439134756412 6 connected 0-5460
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439134758418 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 slave 8dd55e9b4da9f62b9b15232e86553f1337864179 0 1439134753407 4 connected
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 master,fail - 1439134742175 1439134741373 4 disconnected 10923-16383
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439134757416 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439134756412 6 connected 0-5460
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439134758418 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 master - 0 1439134753407 7 connected 10923-16383
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 master,fail - 1439134742175 1439134741373 4 disconnected
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439134757416 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439134759421 6 connected 0-5460
```

节点状态 fail? 表示正在判断是否失败

节点状态 fail 表示节点失败，对应的 slave 节点提升为 master

(5)再查看集群状态变化# /usr/local/src/redis-3.0.3/src/redis-trib.rb check 192.168.1.116:7116

```
[root@edu-redis-06 7116]# /usr/local/src/redis-3.0.3/src/redis-trib.rb check 192.168.1.116:7116
Connecting to node 192.168.1.116:7116: OK
Connecting to node 192.168.1.115:7115: OK
Connecting to node 192.168.1.114:7114: OK
Connecting to node 192.168.1.112:7112: OK
Connecting to node 192.168.1.113:7113: OK
>>> Performing Cluster Check (using node 192.168.1.116:7116)
S: 1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116
   slots: (0 slots) slave
   replicates 4e46bd06654e8660e617f7249fa22f6fa1fdff0d
S: f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115
   slots: (0 slots) slave
   replicates d2c6c159b07e8197e2c8d2eae8c847050159f602
M: 48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114
   slots:10923-16383 (5461 slots) master
   0 additional replica(s)
M: d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
M: 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113
   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.
[root@edu-redis-06 7116]#
```

由上可见，7114 节点替换 7111，由 slave 变成了 master

此时再执行 demo 应用获取所有的键值数据，依然正常，说明 slave 替换 master 成功，集群正常。

6、恢复 fail 节点

(1)启动 7111

/usr/local/redis3/bin/redis-server /usr/local/redis3/cluster/7111/redis-7111.conf

(2)查看集群状态

```
127.0.0.1:7116> cluster nodes
f34b28f1483f0c0d9543e93938fc12b8818050cb 192.168.1.115:7115 slave d2c6c159b07e8197e2c8d2eae8c847050159f602 0 1439135456556 5 connected
1fd90d54090925afb4087d4ef94a1710a25160d6 192.168.1.116:7116 myself,slave 4e46bd06654e8660e617f7249fa22f6fa1fdff0d 0 0 3 connected
48db78bcc55c4c3a3788940a6458b921ccf95d44 192.168.1.114:7114 master - 0 1439135459563 7 connected 10923-16383
8dd55e9b4da9f62b9b15232e86553f1337864179 192.168.1.111:7111 slave 48db78bcc55c4c3a3788940a6458b921ccf95d44 0 1439135458561 7 connected
d2c6c159b07e8197e2c8d2eae8c847050159f602 192.168.1.112:7112 master - 0 1439135461567 5 connected 5461-10922
4e46bd06654e8660e617f7249fa22f6fa1fdff0d 192.168.1.113:7113 master - 0 1439135460564 6 connected 0-5460
127.0.0.1:7116>
```

其中 7111 变成 7114 的 slave

7、观察集群节点切换过程中，对客户端的影响

JedisCluster 链接 Redis 集群操作时遇到的几个常见异常：

(1)重定向次数过多

redis.clients.jedis.exceptions.JedisClusterMaxRedirectionsException: Too many Cluster redirections?

解决方法： 初始化 JedisCluster 时，设定 JedisCluster 的 maxRedirections

```
//集群各节点集合，超时时间（默认 2 秒），最多重定向次数(默认 5)，链接池
new JedisCluster(jedisClusterNodes, 2000, 100, config);
```

(2)集群不可以用

redis.clients.jedis.exceptions.JedisClusterException: CLUSTERDOWN The cluster is down

原因：集群节点状态切换过程中会出现临时闪断，客户端重试操作则可。

(3)链接超时

redis.clients.jedis.exceptions.JedisConnectionException: java.net.SocketTimeoutException: Read timed out

解决方法： 初始化 JedisCluster 时，设定 JedisCluster 的 timeout（默认为两秒）；也可以修改源码中的默认时间。

7、总结:

优点:

在 master 节点下线后，slave 节点会自动提升为 master 节点，保存集群持续提供服务；

fail 节点恢复后，会自动添加到集群中，变成 slave 节点；

缺点:

由于 redis 的复制使用异步机制，在自动故障转移的过程中，集群可能会丢失写命令。然而 redis 几乎是同时执行(将命令恢复发送给客户端，以及将命令复制到 slave 节点)这两个操作，所以实际中，命令丢失的窗口非常小。