Redis- 探究：集群扩容导致 Jedis 客户端报 JedisMovedDataException 异常的原因

作者：Lord-X

原文链接：https://ld246.com/article/1565927030823

来源网站：链滴

□rose□rose　　如果您觉得我的文章对您有帮助的话，记得在GitHub上star一波哈□
ose□rose

□rose□rose　　GitHub_awesome-it-blog □rose□rose

---

0 问题的产生

由于线上Redis集群内存使用量已经接近达到预警阈值，需要对Redis集群扩容。（使用的是Redis自的Redis-Cluster）

目前有6台主节点，6台从节点。

暂时称为：

- redis-master001 ~ redis-master006
- redis-slave001 ~ redis-slave006

需要增加3主3从。

- redis-master007 ~ redis-master009
- redis-slave007 ~ redis-master009

之前Redis集群的16384个槽均匀分配在6台主节点中，每个节点2730个槽。

为保证扩容后，槽依然均匀分布，需要将之前6台的每台机器上迁移出910个槽，方案如下：

- redis-master001的910个slot迁移到redis-master007
- redis-master002的910个slot迁移到redis-master007

- redis-master003的910个slot迁移到redis-master008
- redis-master004的910个slot迁移到redis-master008
- redis-master005的910个slot迁移到redis-master009
- redis-master006的910个slot迁移到redis-master009

分配完之后，每台节点1820个slot。

当将redis-master001的910个slot迁移到redis-master007后，业务上开始报下面的异常

在马赛克的上一行，可以看到是调Jedis的get方法出的问题。

1 原因及解决方案

问题的原因在于使用了Jedis客户端，改为使用JedisCluster客户端即可解决问题。

出问题的get方法是这样写的（在Jedis原生基础上包装了一层）

```
// 自己封装的get方法
public String get(String key) {
    String result =  "";
    // 为了打印获取连接耗时的日志，这里单独获取了一下Jedis连接
    try (Jedis jedis = this.getResourceLog(key)) {
        TimeCost timeCost = new TimeCost();
        result = jedis.get(key); // 这里报错
        debugLogger.debug("redis cluster get TimeCost={}", timeCost.getCostMillSeconds());
    }
    // 其实改为下面这样get就可以解决一直报JedisMovedDataException问题
    // return jedisCluster.get(key);
    return result;
}
```

getResourceLog方法的作用是根据key计算出这个key所在的slot，再通过slot获取Redis连接。代码下

```
private Jedis getResourceLog(String key) {
    TimeCost tc = new TimeCost();
    int slot = JedisClusterCRC16.getSlot(key); // CRC计算slot
    debugLogger.debug("calc slot TimeCost={}", tc.getCostMillSeconds());
    tc.reset();
    Jedis jedis = connectionHandler.getConnectionFromSlot(slot); // 通过slot获取连接
```

```
    debugLogger.debug("get connection TimeCost={}", tc.getCostMillSeconds());
    return jedis;
}
```

上面的get方法可以直接改为JedisCluster的get方法解决。

再考虑另外一种情况，如果必须通过Jedis操作呢？比如watch方法，JedisCluster是不提供watch的那么只能通过上述方法在Redis集群中根据key获取到slot，再通过slot获取到jedis链接，然后调用watch。这样一来，在调watch的地方也会报JedisMovedDataException。

例如下面的代码，在业务上需要保证事务的情况下（或乐观锁），可能会这样实现：

```
Jedis jedis = null;
String key = ...; // redis key
try {
    // 通过上面的getResource方法获取jedis链接
    jedis = getResource(userId);
    // 通过jedis watch key
    if (RedisConstants.SAVE_TO_REDIS_OK.equals(jedis.watch(key))) {

        // .... 业务逻辑 ....
        // ....
        // 通过jedis链接开始事务
        Transaction transaction = jedis.multi();
        // ...
        // ... 执行一些transaction操作...
        // ...
        // 提交事务
        List<Object> execResult = transaction.exec();

        return ...;
    }
} catch (Exception ex) {
    // do something ...
} finally {
    if (jedis != null) {
        try {
            if (!flag) {
                jedis.unwatch();
            }
        } finally {
            jedis.close();
        }
    }
}
```

此时如果发生slot迁移，就会报JedisMovedDataException。

那这种情况下的解决方案是什么呢？

其实，优先catch住JedisMovedDataException，然后通过JedisCluster.get(key);一下就行，如下：

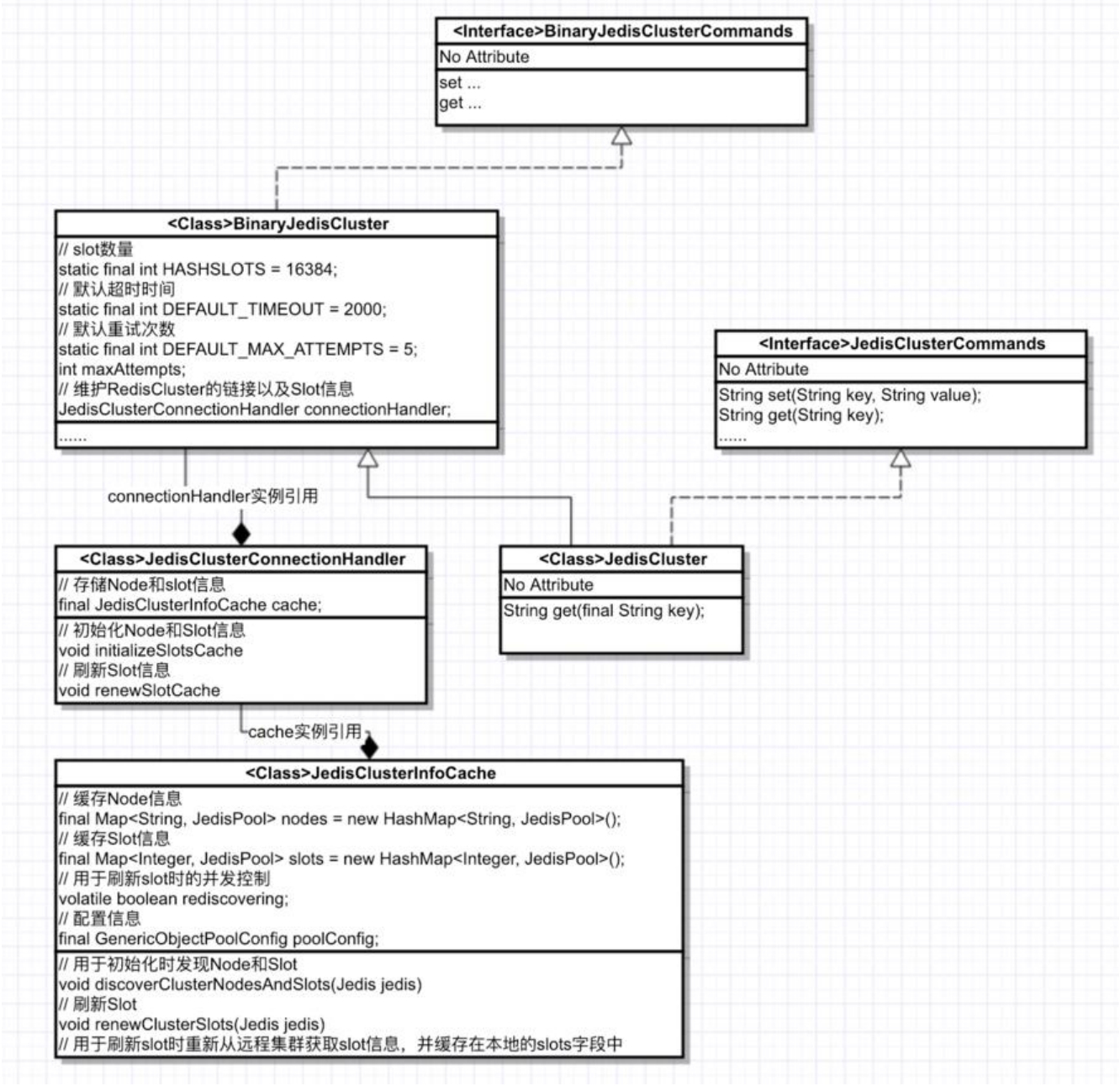
```
Jedis jedis = null;
String key = ...; // redis key
try {
```

```
    // 通过上面的getResource方法获取jedis链接
    jedis = getResource(userId);
    // 通过jedis watch key
    if (RedisConstants.SAVE_TO_REDIS_OK.equals(jedis.watch(key))) {

        // .... 业务逻辑 ....
        // ....
        // 通过jedis链接开始事务
        Transaction transaction = jedis.multi();
        // ...
        // ... 执行一些transaction操作...
        // ...
        // 提交事务
        List<Object> execResult = transaction.exec();

        return ...;
    }
} catch (JedisMovedDataException jmde) {
    jmde.printStackTrace();
    // redisClusterService中维护着jedisCluster实例，这个get实际上调用的是jedisCluster的get
    redisClusterService.get(key);
    return ...;
} catch (Exception ex) {
    // do something ...
} finally {
    if (jedis != null) {
        try {
            if (!flag) {
                jedis.unwatch();
            }
        } finally {
            jedis.close();
        }
    }
}
```

需要注意的是，用Jedis的get是不能解决的。

2 JedisCluster类图

JedisCluster整体的UML关系如下，先有个整体的印象，在后面的源码分析中，可以再回来看。



3 为什么通过RedisCluster.get一下可以解决?

下面通过JedisCluster源码解释为什么这么做可以解决问题，注释中会有详细说明。

JedisCluster.get源码如下：

```
@Override
public String get(final String key) {
  return new JedisClusterCommand<String>(connectionHandler, maxAttempts) {
    @Override
    public String execute(Jedis connection) {
      return connection.get(key);
    }
  }.run(key);
}
```

发现他是委托给JedisClusterCommand来完成get操作的，也可以发现execute方法实际上是使用Jedi来执行的get。这个Jedis实际上就是通过上述方法，先计算出slot，再通过slot获取到Jedis链接的。键在于最下面run方法的执行，下面具体看一下。

Run方法源码如下：

```
public T run(String key) {
  //  JedisClusterCRC16.getSlot(key) 计算出slot
  return runWithRetries(JedisClusterCRC16.getSlot(key), this.maxAttempts, false, null);
}
```

runWithRetries源码如下

```
private T runWithRetries(final int slot, int attempts, boolean tryRandomNode, JedisRedirectio
Exception redirect) {
  // 这里是一个重试机制，报异常时触发
  if (attempts <= 0) {
    throw new JedisClusterMaxAttemptsException("No more cluster attempts left.");
  }

  Jedis connection = null;
  try {

    if (redirect != null) {
      connection = this.connectionHandler.getConnectionFromNode(redirect.getTargetNode());
      if (redirect instanceof JedisAskDataException) {
        // TODO: Pipeline asking with the original command to make it faster....
        connection.asking();
      }
    } else {
      if (tryRandomNode) {
        connection = connectionHandler.getConnection();
      } else {
        // 执行到这里，通过slot获取到Jedis connection
        // 内部是通过一个map维护的slot到JedisPool的映射关系
        connection = connectionHandler.getConnectionFromSlot(slot);
      }
    }

    // 执行上面JedisClusterCommand定义的execute方法。
    return execute(connection);

  } catch (JedisNoReachableClusterNodeException jnrcne) {
    throw jnrcne;
  } catch (JedisConnectionException jce) {
    // release current connection before recursion
    releaseConnection(connection);
    connection = null;

    if (attempts <= 1) {
      //We need this because if node is not reachable anymore - we need to finally initiate slots
      //renewing, or we can stuck with cluster state without one node in opposite case.
      //But now if maxAttempts = [1 or 2] we will do it too often.
      //TODO make tracking of successful/unsuccessful operations for node - do renewing only
```

```
      //if there were no successful responses from this node last few seconds
      this.connectionHandler.renewSlotCache();
    }

    return runWithRetries(slot, attempts - 1, tryRandomNode, redirect);
  } catch (JedisRedirectionException are) { // *** 关键在这 ***
    // if MOVED redirection occurred,
    // JedisMovedDataException是JedisRedirectionException的子类，所以会执行下面if中的代码
    if (jre instanceof JedisMovedDataException) {
      // it rebuilds cluster's slot cache recommended by Redis cluster specification
      // 重新通过这个jedis链接获取RedisCluster中的Node信息以及slot信息
      this.connectionHandler.renewSlotCache(connection);
    }

    // release current connection before recursion
    releaseConnection(connection);
    connection = null;

    return runWithRetries(slot, attempts - 1, false, jre);
  } finally {
    releaseConnection(connection);
  }
}
```

注释中说到了最终会通过this.connectionHandler.renewSlotCache(connection);来重新获取slot信。下面来看下这个方法。

```
public void renewSlotCache(Jedis jedis) {
  cache.renewClusterSlots(jedis);
}
```

调用了cache的renewClusterSlots方法来重新获取slot信息，这个cache是JedisClusterInfoCache类实例，他里面维护这Node和Slot信息，如下：

```
public class JedisClusterInfoCache {
  private final Map<String, JedisPool> nodes = new HashMap<String, JedisPool>();
  private final Map<Integer, JedisPool> slots = new HashMap<Integer, JedisPool>();

  // ..
}
```

renewClusterSlots方法如下

```
public void renewClusterSlots(Jedis jedis) {
  //If rediscovering is already in process - no need to start one more same rediscovering, just
eturn
  if (!rediscovering) {
    try {
      w.lock();
      if (!rediscovering) {
        rediscovering = true;

        try {
          if (jedis != null) {
            try {
```

```
            // 关键在于这一步，这个方法会重新从远程集群中获取最新的slot信息
            discoverClusterSlots(jedis);
            return;
          } catch (JedisException e) {
            //try nodes from all pools
          }
        }

        for (JedisPool jp : getShuffledNodesPool()) {
          Jedis j = null;
          try {
            j = jp.getResource();
            discoverClusterSlots(j);
            return;
          } catch (JedisConnectionException e) {
            // try next nodes
          } finally {
            if (j != null) {
              j.close();
            }
          }
        }
      } finally {
        rediscovering = false;
      }
    }
  } finally {
    w.unlock();
  }
 }
}
```

关键在于discoverClusterSlots方法，这个方法的实现如下：

```
private void discoverClusterSlots(Jedis jedis) {
  // 通过slots命令从远程获取slot信息
  List<Object> slots = jedis.clusterSlots();
  this.slots.clear(); // 清除本地缓存slot信息

  // 每个slotInfoObj包含集群中某一节点的slot信息
  for (Object slotInfoObj : slots) {
    List<Object> slotInfo = (List<Object>) slotInfoObj;

    if (slotInfo.size() <= MASTER_NODE_INDEX) {
      continue;
    }
    // 计算当前节点的slot信息
    List<Integer> slotNums = getAssignedSlotArray(slotInfo);

    // hostInfos
    // 获取这组slot所在的节点信息
    List<Object> hostInfos = (List<Object>) slotInfo.get(MASTER_NODE_INDEX);
    if (hostInfos.isEmpty()) {
      continue;
```

```
    }

    // at this time, we just use master, discard slave information
    HostAndPort targetNode = generateHostAndPort(hostInfos);
    // 重新关联这组slot到远程节点的映射，至此，完成slot信息的刷新
    assignSlotsToNode(slotNums, targetNode);
  }
}
```

4 为什么Jedis的get不行?

首先我们来对比一下JedisCluster的get和Jedis的get

JedisCluster.get

```
@Override
public String get(final String key) {
  return new JedisClusterCommand<String>(connectionHandler, maxAttempts) {
    @Override
    public String execute(Jedis connection) {
      return connection.get(key); // 这里追踪进去，就是Jedis.get
    }
  }.run(key);
}
```

Jedis.get

```
@Override
public String get(final String key) {
  checkIsInMultiOrPipeline();
  client.get(key);
  return client.getBulkReply();
}
```

由此可知，Jedis.get没有了run方法中的异常重试和重新发现机制，所以Jedis.get不行。

5 总结

本文从一次线上扩容引发问题的讨论，由扩容引出了slot的迁移，由slot的迁移引出线上报错-JedisMvedDataException，然后说明了引发这个异常的原因，是因为我们使用了Jedis客户端，导致无法自发现远程集群slot的变化。

然后提出了解决方案，通过使用JedisCluster来解决无法自动发现slot变化的问题。并从源码的角度明了为什么JedisCluster的get方法可以自动发现远程slot的变化。