/tos-cn-i-k3u1fbpfcp/b8b613ffd77246d1af06193487333816~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1960&h=778&s=587276&e=png&b=feffff)

那么，我们是否也可以将多个set、get操作打包成batch，一次性提交多条操作命令?

pipeline

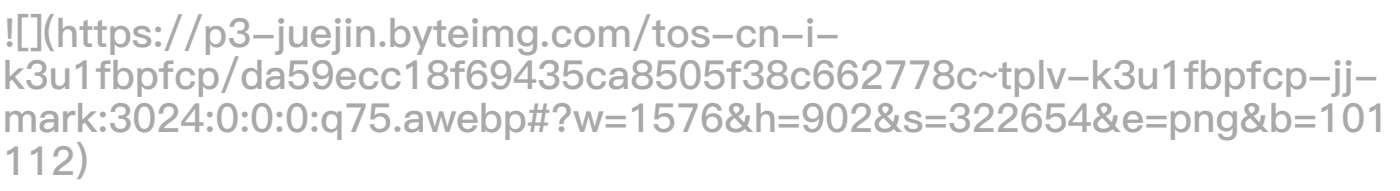
Jedis提供了pipeline的操作模式，允许一次性执行多个操作命令，然后通过手动执行*sync*来发送到Redis，返回请求结果。接着上面连接池实现pipeline代码。

```
Jedis jedis = jedisPool.getResource()
Pipeline pipeline = jedis.pipelined();
Response<String> res1 = pipeline.get(”aa”);
Response<String> res2 = pipeline.get(”bb”);
// 这里也可以进行set、hmget等操作
pipeline.sync();
String s1 = res1.get();
String s2 = res2.get();
```

从*jedis.pipelined()* 创建pipeline到执行sync发送redis之间，你可以执行多个不同redis命令。通过Jedis执行get操作，返回的是String。而使用pipeline执行，返回的是泛型为String的*Response*。

Response数据回填

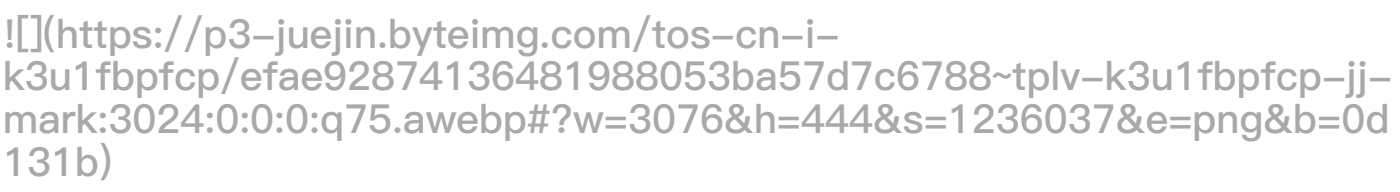
这个和NIO里面的Future差不多，通过上面代码也知道，pipeline执行完get(key)之后，不是立即发送到redis的，所以这时候只能返回一个Response对象，预留出一个字段来接收返回值，当执行sync()发送请求到redis之后，再将返回的结果一一回填到对应Response的字段上。



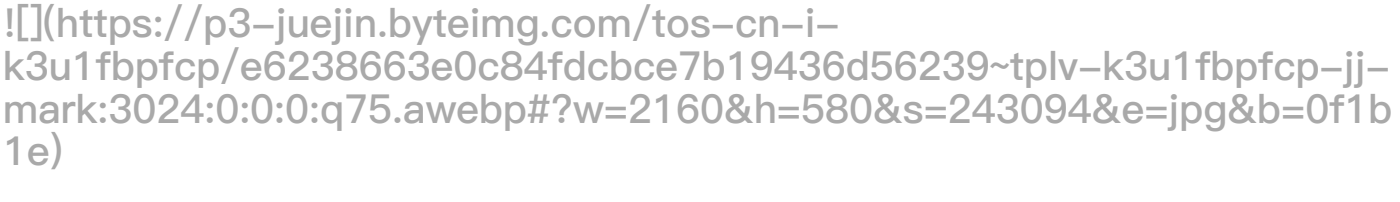
如图，只有sync之后，Response才会有数据，可以debug看一下。

Response debug

我使用docker，启动一个四主四从的redis cluster，对应端口从10001 – 10008。



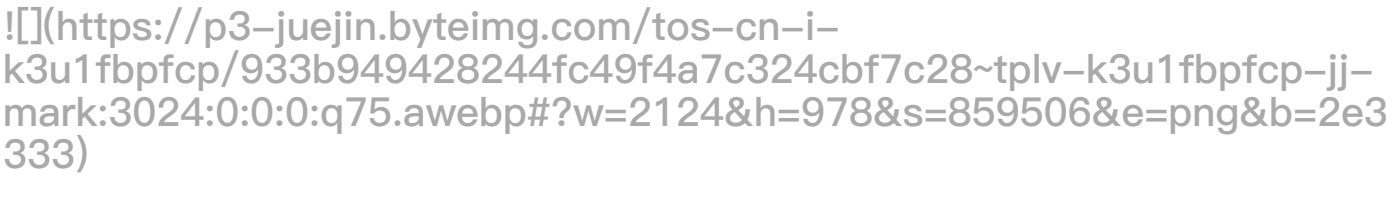
使用redis-cli set两个key的数据。在cluster模式下，一定要加-c，否则就不会重定向到key所在slot的节点上，现在不懂slot无妨，因为接下来我会讲slot。



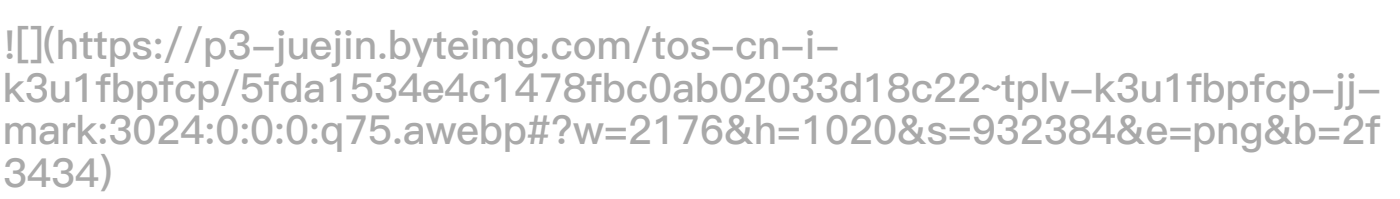
代码中使用Jedis连接其中的10001节点（不能使用JedisCluter，后面会讲）.

```
 Jedis jedis = new Jedis(”121.91.xxx.xx”, 10001);
jedis.auth(”1qaz@WSX”);
Pipeline pipeline = jedis.pipelined();
Response<String> res1 = pipeline.get(”aa”);
pipeline.sync();
String s1 = res1.get();
```

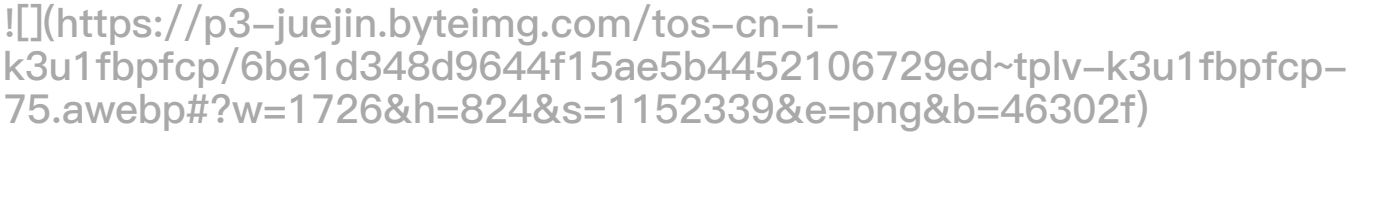
然后使用pipeline获取key为aa的value值。在sync()处打上断点，debug启动。



可以看到，response的data属性此时为null。执行下一步，也就是执行完sync()之后，data就有值了，返回的应该是我们之前set的1，这里49是字符串1在ACSII码表中的编号。



所以，通常pipeline中执行一批(batch)的命令之后，再执行sync()将所有命令发送到redis，我在SparkStreaming的开发中，通常将batch设置为256或者512，也就是一次执行256或512个命令。



JedisCluster底层为集群中所有redis，都创建了一个JedisPool。但纵观整个JedisCluster，没有给用户暴露出来Jedis对象，也就是说JedisCluster面对的是cluster中所有的节点，而非其中某一个节点。所以*JedisCluster*是无法使用pipeline的。

redis cluster的pipeline

我在SparkStreaming中，处理1亿/min数据使用redis cluster的时候，会因为与redis cluster交互过多而导致数据延迟，为此在刚从codis切换到redis cluster的时候，尝试了很多办法。

lettuce

redis cluster除了JedisCluster，也有一个开源的*lettuce*，其中的*async异步*类pipeline操作。

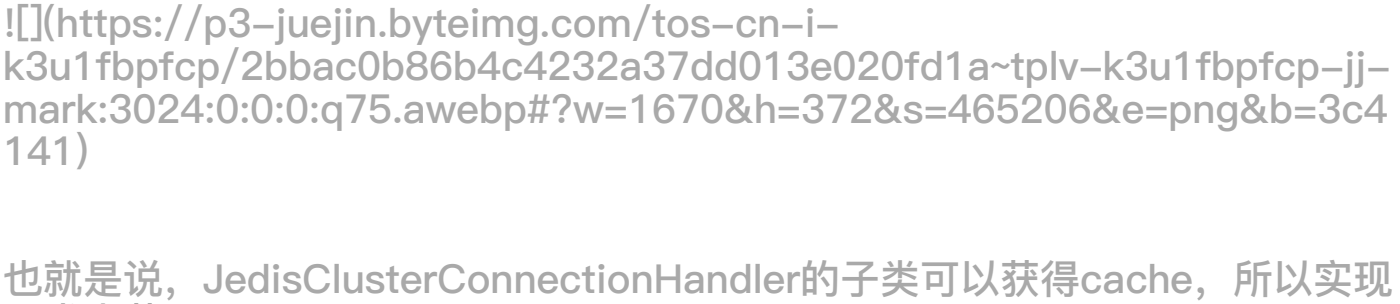
```
RedisURI uri = RedisURI.builder()
        .withHost(”47.102.xxx.xxx”)
        .withPassword(”1qaz@WSX”.toCharArray())
        .withPort(10001)
        .build();
RedisClusterClient client = RedisClusterClient.create(uri);
StatefulRedisClusterConnection<String, String> connect =
client.connect();
RedisAdvancedClusterAsyncCommands<String, String> async =
connect.async();
async.set(”key1”, ”v1”);
async.set(”key2”, ”v2”);
async.flushCommands();
Thread.sleep(1000 * 3);
connect.close();
client.shutdown();
```

但是实测之后，效率还是提不上去，有兴趣的朋友可以自己尝试一下。还有一个*Redisson*客户端，测试效果也不太理想。后来我还是自己实现了一个基于JedisCluster的pipeline客户端，现在生产在用，效率还是很快的。

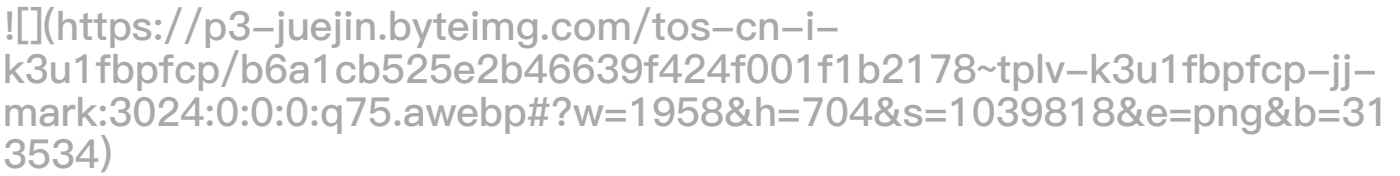
自定义pipeline

上面讲了，JedisCluster的*connectionHandler*属性的*cache*属性的*slots*属性，建立了slot与JedisPool的关系，但是JedisCluster中没有对外暴露cache。

万幸的是，cache是*protected*，而不是private。



也就是说，JedisClusterConnectionHandler的子类可以获得cache，所以实现子类来获取slots。



利用slots的映射关系，我们也建立几个map映射，*来实现针对于每个redis节点，我们都能获取到一个Jedis，来开启pipeline*。

1. <JedisPool, Jedis>： 在打入一个key时，计算出key对应的slot，然后从slots中获取到对应的JedisPool，从JedisPool中取出一个Jedis，建立一个<JedisPool, Jedis>的映射，如果下次的key也在这个slot，就直接使用这个Jedis。保证一个redis实例只有一个Jedis，这样pipeline才有意义。
2. <Jedis, Pipeline>：通过这个映射，保证一个Jedis只开启一次Pipeline，而且相同redis上的key都可以通过这个pipeline来操作。
3. <Pipeline, num>：这个是用来设置sync的阈值，num是记录使用这个pipeline操作的次数，遍历，到达设定阈值的时候就执行sync，将pipeline的命令发送到redis中。

具体实现思路可以参考早期的文章：[JedisCluster Pipeline的实现思路](http://cxyroad.com/”https://cloud.tencent.com/developer/article/1997885”)

在这个理论基础上，后来根据自己的实际开发需求，使用java + scala开发除了适配SparkStreaming版本的JedisCluster pipeline。

结语
--

这就是我个人在大数据开发中，对redis使用的一些经验之谈。感觉redis有好多要写的，除了上面的这些优化手段之外，例如hash结构中K/V的合理设计等等一系列手段。

最近，陪伴了七年之久的codis，在无法通过升级来修复漏洞的情况下，宣布下线。虽然很早就开始从codis向redis cluster靠拢，但是此刻也不免感叹一声”自古美人叹迟暮，不许英雄见白头”。

原文链接: https://juejin.cn/post/7386958406134874162