

图文 103 第三个场景驱动：Broker是如何把自己注册到NameServer去的？

191 人次阅读 2020-02-24 18:00:00

详情 评论

返回
前进
重新加载
打印



狸猫技术

进店逛

相关频道



从 0 开
间件实站
已更新1

第三个场景驱动：Broker是如何把自己注册到NameServer去的？



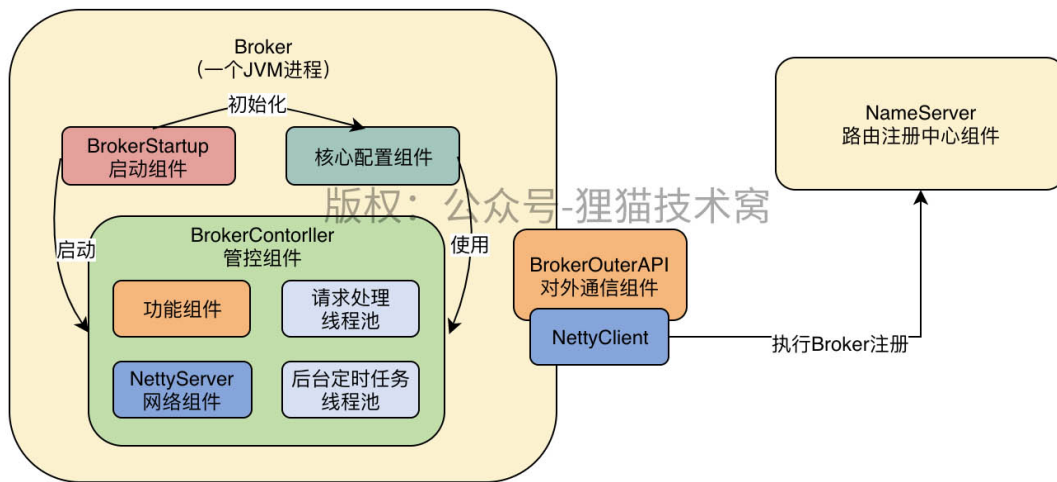
继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

(点击下方蓝字试听)

[《从零开始带你成为MySQL实战优化高手》](#)

1、Broker将自己注册到NameServer的入口

上回我们讲到了BrokerController启动的过程，其实他本质就是启动了Netty服务器去接收网络请求，然后启动了一堆核心功能组件，启动了一些处理请求的线程池，启动了一些执行定时调度任务的后台线程，如下图所示，我们回顾一下。



返回
前进
重新加载
打印

当然，最为关键的一点，就是他执行了将自己注册到NameServer的一个过程，我们看一下这个注册自己到NameServer的源码入口，下面这行代码就是在BrokerController.start()方法中

```
BrokerController.this.registerBrokerAll(true, false, brokerConfig.isForceRegister());
```

因此如果我们要继续研究RocketMQ源码的话，当然应该场景驱动来研究，之前已经研究完了NameServer和Broker两个核心系统的启动场景，现在来研究第三个场景，就是Broker往NameServer进行注册的场景。

因为只有完成了注册，NameServer才能知道集群里有哪些Broker，然后Producer和Consumer才能找NameServer去拉取路由数据，他们才知道集群里有哪些Broker，才能去跟Broker进行通信！

2、进入registerBrokerAll()方法去初步看一看

接着我们就进入到registerBrokerAll()方法中初步的去看一看，大家看下面的源码片段，就是registerBrokerAll()方法的源码，我都写了详细的注释了，大家仔细看一下。

```

1 public synchronized void registerBrokerAll(
2
3     final boolean checkOrderConfig,
4     boolean oneway,
5     boolean forceRegister) {
6
7     // 下面这行代码估计很多人看了会一脸懵逼
8     // 其实这个没关系，你大致就知道，他就是Topic配置信息相关的东西就行
9     // 看源码的时候如果老是深究类似下面这种代码是干什么的，那你绝对会放弃的
10    TopicConfigSerializeWrapper topicConfigWrapper =
11        this.getTopicConfigManager().
12        buildTopicConfigSerializeWrapper();
13
14    // 下面一大段代码，其实你会发现，都是在处理TopicConfig的一些东西
15    // 而且里面的一些类名看着稀奇古怪的，可能很多人根本不知道怎么理解他们
16    // 其实这个真没关系，你只要脑子里有个印象，这里在搞一些TopicConfig的东西就好
17    if (!PermName.isWritable(this.getBrokerConfig().getBrokerPermission()))
18
19        || !PermName.isReadable(this.getBrokerConfig().getBrokerPermission())) {
20
21        ConcurrentHashMap<String, TopicConfig> topicConfigTable =
22            new ConcurrentHashMap<String, TopicConfig>();
23
24        for (TopicConfig topicConfig : topicConfigWrapper.getTopicConfigTable().values()) {
25
26            TopicConfig tmp =
27                new TopicConfig(topicConfig.getTopicName(),
28                                topicConfig.getReadQueueNums(),
29                                topicConfig.getWriteQueueNums(),
30                                this.brokerConfig.getBrokerPermission());
31
32            topicConfigTable.put(topicConfig.getTopicName(), tmp);
33        }
34        topicConfigWrapper.setTopicConfigTable(topicConfigTable);
35    }
36
37    // 下面这个才比较关键一些，他就是判断了一下，是否要进行注册
38    // 如果要进行注册的话，就调用了doRegisterBrokerAll()这个方法，真正去注册
39    if (forceRegister || needRegister(this.brokerConfig.getBrokerClusterName(),
40                                     this.getBrokerAddr(),
41                                     this.brokerConfig.getBrokerName(),
42                                     this.brokerConfig.getBrokerId(),
43                                     this.brokerConfig.getRegisterBrokerTimeoutMills())) {
44
45        doRegisterBrokerAll(checkOrderConfig, oneway, topicConfigWrapper);
46    }
47 }

```

3、继续探索真正的进行Broker注册的方法

接着我们继续探索真正进行Broker注册的方法，也就是下面的doRegisterBrokerAll()方法，我们进去可以先初步看一下方法的整体情况，我都写了详细的注释，大家也仔细看一看就行。

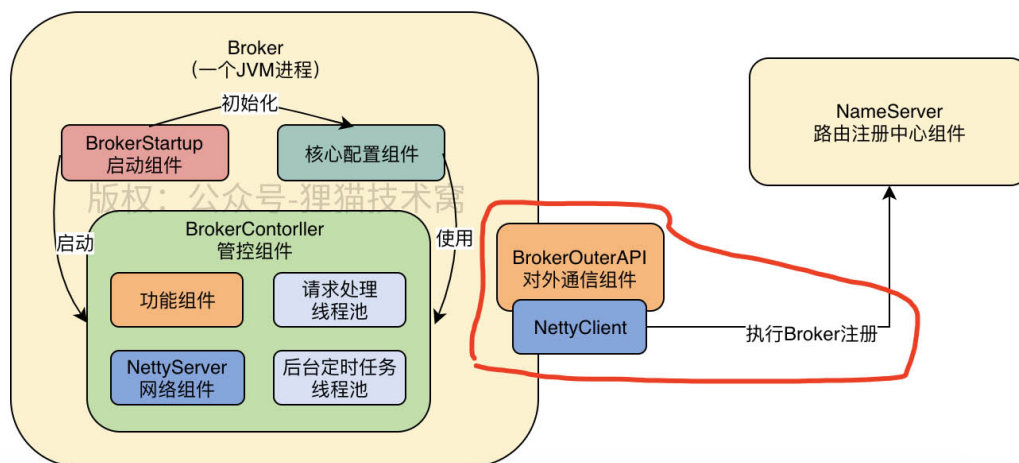
```

1 private void doRegisterBrokerAll(boolean checkOrderConfig,
2                                 boolean oneway,
3                                 TopicConfigSerializeWrapper topicConfigWrapper) {
4
5     // 说白了，最核心的进行注册的代码就在这里
6     // 大家可以看到，这里调用了BrokerOuterAPI去发送请求给NameServer
7     // 在这里就完成了Broker的注册，然后获取到了注册的结果
8     // 为什么注册结果是个List呢？因为Broker会把自己注册给所有的NameServer！
9     List<RegisterBrokerResult> registerBrokerResultList =
10
11         this.brokerOuterAPI.registerBrokerAll(this.brokerConfig.getBrokerClusterName(),
12                                             this.getBrokerAddr(),
13                                             this.brokerConfig.getBrokerName(),
14                                             this.brokerConfig.getBrokerId(),
15                                             this.getHAServerAddr(),
16                                             topicConfigWrapper,
17                                             this.filterServerManager.buildNewFilterServerList(),
18                                             oneway,
19                                             this.brokerConfig.getRegisterBrokerTimeoutMills(),
20                                             this.brokerConfig.isCompressedRegister());
21
22     // 如果说注册结果的数量大于0，那么就在这里对注册结果进行处理
23     // 处理的逻辑涉及到了MasterHAServer之类的东西，这里我们就先不用深究了
24     if (registerBrokerResultList.size() > 0) {
25
26         RegisterBrokerResult registerBrokerResult = registerBrokerResultList.get(0);
27
28         if (registerBrokerResult != null) {
29             if (this.updateMasterHAServerAddrPeriodically && registerBrokerResult.getHaServerAddr() != null) {
30                 this.messageStore.
31                     updateHaMasterAddress(registerBrokerResult.getHaServerAddr());
32             }
33             this.slaveSynchronize.
34                 setMasterAddr(registerBrokerResult.getMasterAddr());
35
36             if (checkOrderConfig) {
37                 this.getTopicConfigManager().
38                     updateOrderTopicConfig(registerBrokerResult.getKvTable());
39             }
40         }
41     }
42 }

```

返回
前进
重新加载
打印

其实大家看完上面的代码，再看一下下面的图中，我用红圈圈出来的部分，你就会发现，在这里实际上就是通过BrokerOuterAPI去发送网络请求给所有的NameServer，把这个Broker注册了上去。



4、深入到网络请求级别的Broker注册逻辑

接着我们继续去看BrokerOuterAPI中的registerBrokerAll()方法，就是深入到了网络请求级别的Broker注册了，我给代码写了详细的注释，大家也是仔细看一看。

```
1 public List<RegisterBrokerResult> registerBrokerAll(  
2  
3     final String clusterName,  
4     final String brokerAddr,  
5     final String brokerName,  
6     final long brokerId,  
7     final String haServerAddr,  
8     final TopicConfigSerializeWrapper topicConfigWrapper,  
9     final List<String> filterServerList,  
10    final boolean oneway,  
11    final int timeoutMills,  
12    final boolean compressed) {  
13  
14    // 下面其实就是初始化一个list，用来放向每个NameServer注册的结果的  
15    final List<RegisterBrokerResult> registerBrokerResultList = Lists.newArrayList();  
16  
17    // 下面这个list就不用多说了，其实就是NameServer的地址列表  
18    List<String> nameServerAddressList = this.remotingClient.getNameServerAddressList();  
19  
20    if (nameServerAddressList != null && nameServerAddressList.size() > 0) {  
21  
22        // 下面这个很关键，他其实就是在构建注册的网络请求了  
23        // 首先他有一个请求头，在请求头里加入了很多的信息，比如broker的id和名称  
24        final RegisterBrokerRequestHeader requestHeader = new RegisterBrokerRequestHeader();  
25  
26        requestHeader.setBrokerAddr(brokerAddr);  
27        requestHeader.setBrokerId(brokerId);  
28        requestHeader.setBrokerName(brokerName);  
29        requestHeader.setClusterName(clusterName);  
30        requestHeader.setHaServerAddr(haServerAddr);  
31        requestHeader.setCompressed(compressed);  
32  
33        // 下面这个就是请求体了，请求体里就会包含一些配置  
34        RegisterBrokerBody requestBody = new RegisterBrokerBody();  
35        requestBody.setTopicConfigSerializeWrapper(topicConfigWrapper);  
36        requestBody.setFilterServerList(filterServerList);  
37        final byte[] body = requestBody.encode(compressed);  
38        final int bodyCrc32 = UtilAll.crc32(body);  
39  
40        requestHeader.setBodyCrc32(bodyCrc32);  
41  
42        // 然后在这个地方，他搞了一个CountDownLatch  
43        // 这个意思就是要注册完全部的NameServer之后才能往下走  
44        final CountDownLatch countDownLatch =  
45            new CountDownLatch(nameServerAddressList.size());  
46  
47        // 在这个地方，就是遍历NameServer地址列表  
48        // 因为每个NameServer都要发送请求过去进行注册的  
49        for (final String namesrvAddr : nameServerAddressList) {  
50  
51            brokerOuterExecutor.execute(new Runnable() {  
52                @Override  
53                public void run() {  
54                    try {  
55                        // 真正执行注册的地方在这里  
56                        RegisterBrokerResult result =  
57                            registerBroker(namesrvAddr, oneway, timeoutMills, requestHeader, body);  
58  
59                        // 注册完了，注册结果就放到一个list里去  
60                        if (result != null) {  
61                            registerBrokerResultList.add(result);  
62                        }  
59  
60  
61  
62
```

返回
前进
重新加载
打印

```

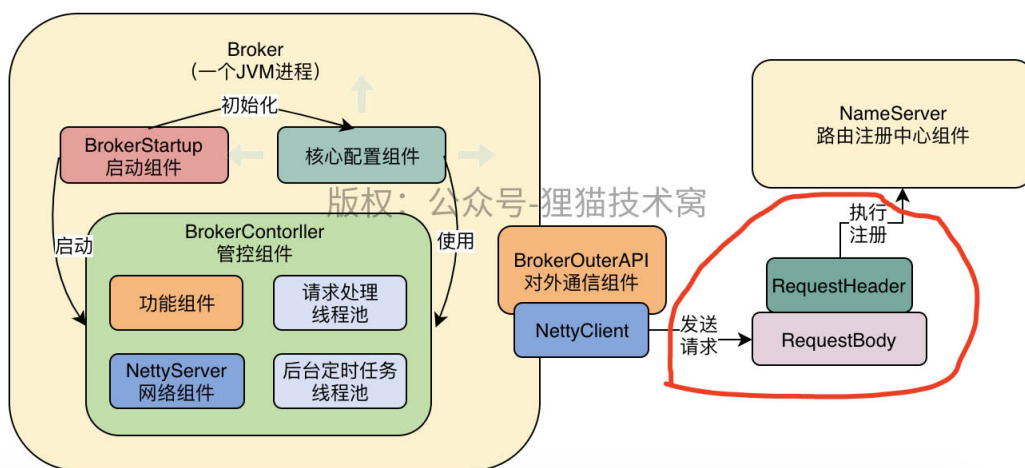
63
64         log.info("register broker[{}]to name server {} OK", brokerId, namesrvAddr);
65     } catch (Exception e) {
66         log.warn("registerBroker Exception, {}", namesrvAddr, e);
67     } finally {
68         // 注册完了, 就会执行CountDownLatch的countDown
69         countDownLatch.countDown();
70     }
71 }
72 });
73 }
74 // 如果有人对CountDownLatch是什么都不知道? 那建议你上网搜一下资料
75 // 简单来说, 他在这里会等待所有的NameServer都注册完毕了, 才会继续往下走
76 try {
77     countDownLatch.await(timeoutMills, TimeUnit.MILLISECONDS);
78 } catch (InterruptedException e) {
79 }
80 }
81 return registerBrokerResultList;
82 }

```

返回
前进
重新加载
打印

上面这段代码里, 大家最主要的, 是要提取出来RequestHeader和RequestBody两个概念, 就是通过请求头和请求体构成了一个请求, 然后会通过底层的NettyClient把这个请求发送到NameServer去进行注册

我们看下图, 我加入了这个概念。



5、今日源码作业

今天给大家布置一个源码小作业, 就是希望大家能够自己在IntelliJ IDEA里, 把今天给大家分析的Broker注册的初步的一些流程都自己看一下, 尝试跟我一样, 去从乱七八糟的源码里提取出来最重要和关键的一些概念。

比如你应该注意到的是Broker注册的时候, 最为关键的BrokerOuterAPI这个组件, 然后注意到他里面是对每个NameServer都执行了注册, 包括他还构造了RequestHeader和RequestBody组成的请求去进行注册。

如果大家有什么分析源码的心得, 可以在评论区里发出来。

End

专栏版权归公众号狸猫技术窝所有

未经许可不得传播, 如有侵权将追究法律责任

狸猫技术窝精品专栏及课程推荐:

[《从零开始带你成为JVM实战高手》](#)

[《21天互联网Java进阶面试训练营》\(分布式篇\)](#)

重要说明:

如何提问: 每篇文章都有评论区, 大家可以尽情留言提问, 我会逐一答疑

如何加群: 购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群, 一个非常纯粹的技术交流的地方

具体加群方式, 请参见目录菜单下的文档: 《付费用户如何加群》(**购买后可见**)

返回
前进
重新加载
打印

