

图文 104 深入探索BrokerOuter API是如何发送注册请求的?

171 人次阅读 2020-02-25 07:00:00

详情 评论



狸猫技术

进店逛

相关频道



从 0 开
间件实站
已更新1

深入探索BrokerOuter API是如何发送注册请求的?



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、进入真正的注册请求方法去看看

现在我们进入到真正的注册Broker的网络请求方法里去看看，其实入口就是下面这行代码：

```
RegisterBrokerResult result = registerBroker(  
    namesrvAddr, oneway, timeoutMills, requestHeader, body);
```

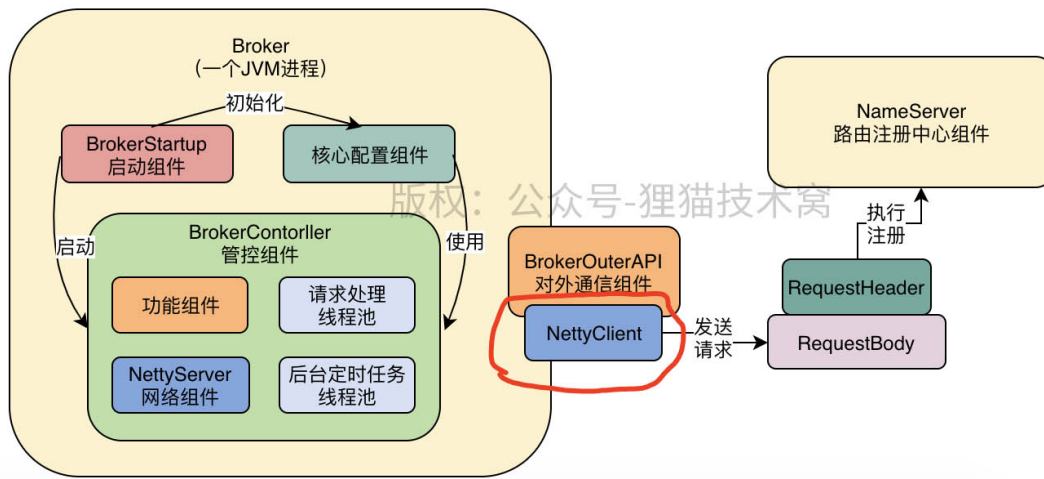
进入这个方法之后，会看到下面的一段代码，我们可以看看，我写了详细的注释，大家仔细看看我写的注释。

```

1 private RegisterBrokerResult registerBroker(final String namesrvAddr,
2                                             final boolean oneway,
3                                             final int timeoutMills,
4                                             final RegisterBrokerRequestHeader requestHeader,
5                                             final byte[] body
6 ) throws Exception {
7
8 // 两行代码大家看着有什么感触？是不是发现他搞了一个RemotingCommand
9 // 然后他把请求头和请求体，都给弄到里面去了，这就封装了一个完整请求出来？
10 RemotingCommand request =
11     RemotingCommand.createRequestCommand(
12         RequestCode.REGISTER_BROKER,
13         requestHeader);
14
15 request.setBody(body);
16
17 // 这个oneway是说不用等待注册结果的意思，这里先不看了
18 // 这属于特殊请求
19 if (oneway) {
20
21     try {
22         this.remotingClient.
23             invokeOneway(namesrvAddr, request, timeoutMills);
24     } catch (RemotingTooMuchRequestException e) {
25         // Ignore
26     }
27     return null;
28 }
29
30 // 真正的发送网络请求的逻辑是在这里
31 // 大家注意一下这里面的RemotingClient是什么？不就是NettyClient吗！
32 // 这个NettyClient就是用来发送网络请求出去的！
33 RemotingCommand response = this.remotingClient.invokeSync(namesrvAddr, request, timeoutMills);
34
35 // 下面就是在处理网络请求的返回结果了
36 // 其实就是把注册请求的结果封装成了Result，保存了起来，并且返回了Result
37 assert response != null;
38
39 switch (response.getCode()) {
40
41     case ResponseCode.SUCCESS: {
42
43         RegisterBrokerResponseHeader responseHeader =
44             (RegisterBrokerResponseHeader) response.decodeCommandCustomHeader(RegisterBrokerResponseHeader.class);
45
46         RegisterBrokerResult result = new RegisterBrokerResult();
47         result.setMasterAddr(responseHeader.getMasterAddr());
48         result.setHaServerAddr(responseHeader.getHaServerAddr());
49
50         if (response.getBody() != null) {
51             result.setKvTable(KVTable.decode(response.getBody(), KVTable.class));
52         }
53
54         return result;
55     }
56     default:
57         break;
58 }
59 throw new MQBrokerException(response.getCode(), response.getRemark());
60 }

```

看到这里，大家先看看下面的图，有没有发现最终的请求是基于NettyClient这个组件给发送出去的？大家看下面的红圈处。



2、进入到NettyClient的网络请求方法中去看看

接着我们进入到NettyClient的网络请求方法中去看看，大家仔细看下面的代码，我都写了详细的注释了。

```

1 @Override
2 public RemotingCommand invokeSync(String addr,
3                                   final RemotingCommand request,
4                                   long timeoutMillis) {
5
6     // 下面的代码就是获取一个当前时间
7     long beginStartTime = System.currentTimeMillis();
8
9     // 这行代码是很关键的!!! 大家可以看到，在这里，他获取了一个Channel!
10    // 大家知道这个Channel是什么吗?
11    // 说白了，就是你这台Broker机器跟NameServer机器之间的一个连接
12    // 所以你看下面传递进去了NameServer的地址，就是跟他建立了一个网络连接!
13    // 而连接建立之后，就是用Channel来代表他的!
14    final Channel channel = this.getAndCreateChannel(addr);
15
16    // 下面的意思就是说，如果你跟NameServer之间的网络连接是ok的
17    // 那么就可以发送请求了
18    if (channel != null && channel.isActive()) {
19        try {
20            // 下面一大坨代码不用管，计算一些时间的
21            doBeforeRpcHooks(addr, request);
22            long costTime = System.currentTimeMillis() - beginStartTime;
23            if (timeoutMillis < costTime) {
24                throw new RemotingTimeoutException("invokeSync call timeout");
25            }
26            // 这里是真正发送网络请求出去的地方
27            RemotingCommand response = this.invokeSyncImpl(channel, request, timeoutMillis - costTime);
28

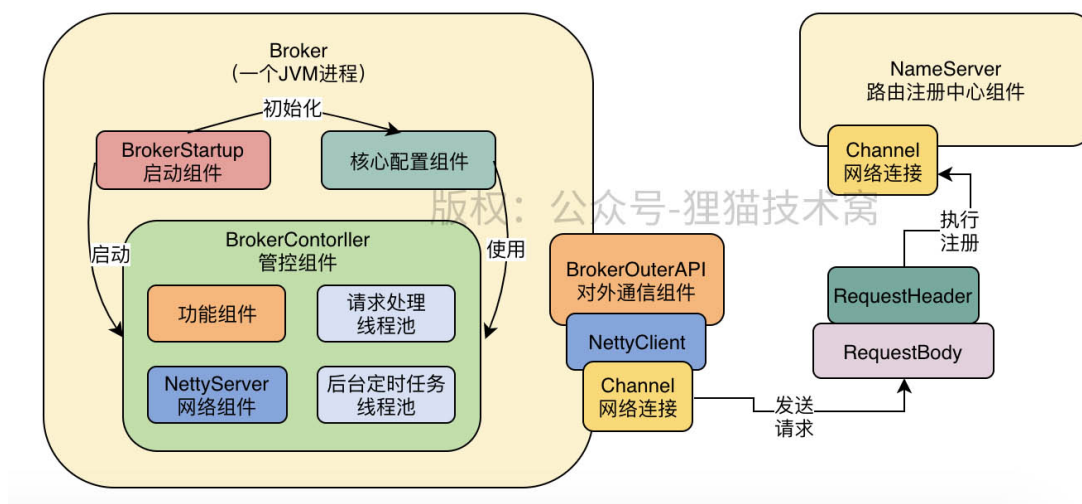
```

```

29 // 下面的也不用管，就是发送请求之后干点别的什么，最后返回结果了
30 doAfterRpcHooks(RemotingHelper.parseChannelRemoteAddr(channel), request, response);
31 return response;
32
33 } catch (RemotingSendRequestException e) {
34
35     log.warn("invokeSync: send request exception, so close the channel[{}]", addr);
36     this.closeChannel(addr, channel);
37     throw e;
38 } catch (RemotingTimeoutException e) {
39     if (nettyClientConfig.isClientCloseSocketIfTimeout()) {
40         this.closeChannel(addr, channel);
41         log.warn("invokeSync: close socket because of timeout, {}ms, {}", timeoutMillis, addr);
42     }
43     log.warn("invokeSync: wait response timeout exception, the channel[{}]", addr);
44     throw e;
45 }
46 } else {
47     this.closeChannel(addr, channel);
48     throw new RemotingConnectException(addr);
49 }
50 }

```

通过上面代码的分析，我现在在下面的图里，给大家再次加入一些东西，我通过Channel这个概念，表示出了Broker和NameServer之间的一个网络连接的概念，然后通过这个Channel就可以发送实际的网络请求出去！



3、如何跟NameServer建立网络连接？

接着我们进入上面的this.getAndCreateChannel(addr)这行代码看看，他是如何跟NameServer之间建立实际的网络连接的？

大家看下面的代码，下面的代码就是先从缓存里尝试获取连接，如果没有缓存的话，就创建一个连接。

```

1 private Channel getAndCreateChannel(final String addr) {
2
3     if (null == addr) {
4         return getAndCreateNameserverChannel();
5     }
6     ChannelWrapper cw = this.channelTables.get(addr);
7
8     if (cw != null && cw.isOK()) {
9         return cw.getChannel();
10    }
11    return this.createChannel(addr);
12 }

```

那我们接着看下面的this.createChannel(addr)方法是如何实际通过一个NameServer的地址创建出来一个网络连接的吧。

我们看下面的代码，我写了详细的注释，大家仔细看里面的注释。

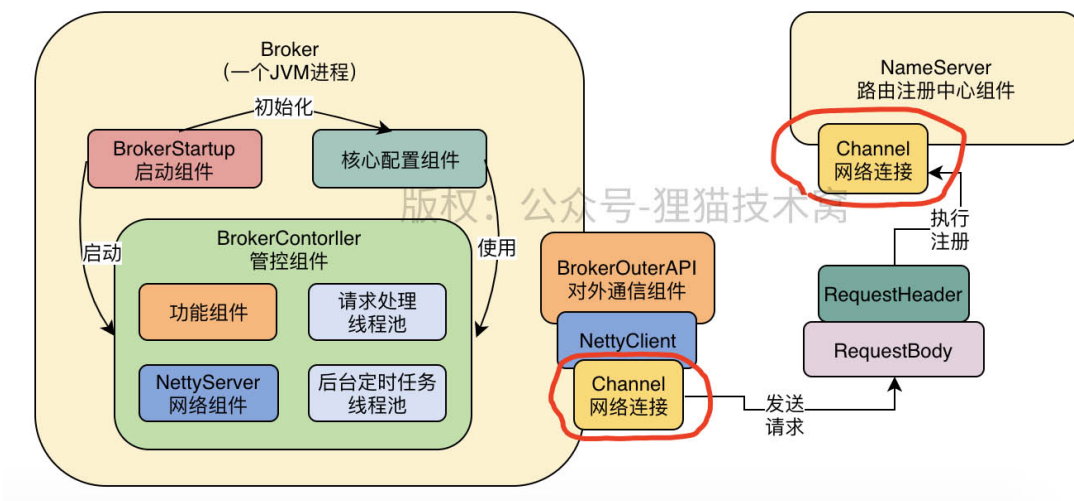
```
1 private Channel createChannel(final String addr) throws InterruptedException {
2
3     // 这个地方就是在尝试获取缓存里的连接，如果有缓存就返回连接了
4     ChannelWrapper cw = this.channelTables.get(addr);
5
6     if (cw != null && cw.isOK()) {
7         return cw.getChannel();
8     }
9
10    if (this.lockChannelTables.tryLock(LOCK_TIMEOUT_MILLIS, TimeUnit.MILLISECONDS)) {
11        try {
12            // 下面一大坨代码也是在获取缓存里的连接
13            boolean createNewConnection;
14            cw = this.channelTables.get(addr);
15            if (cw != null) {
16
17                if (cw.isOK()) {
18                    return cw.getChannel();
19                } else if (!cw.getChannelFuture().isDone()) {
20                    createNewConnection = false;
21                } else {
22                    this.channelTables.remove(addr);
23                    createNewConnection = true;
24                }
25            } else {
26                createNewConnection = true;
27            }
28
29            if (createNewConnection) {
30                // 其实这里才是真正创建连接的地方
31                // 大家可以看到，这里基于Netty的Bootstrap这个类的connect()方法
32                // 就构建出来了一个真正的Channel网络连接！
33                ChannelFuture channelFuture = this.bootstrap.connect(RemotingHelper.string2SocketAddress(addr));
34                log.info("createChannel: begin to connect remote host[{}] asynchronously", addr);
35                cw = new ChannelWrapper(channelFuture);
36                this.channelTables.put(addr, cw);
37            }
38            } catch (Exception e) {
39                log.error("createChannel: create channel exception", e);
40            } finally {
41                this.lockChannelTables.unlock();
42            }
43        } else {
44            log.warn("createChannel: try to lock channel table, but timeout, {}ms", LOCK_TIMEOUT_MILLIS);
45        }
46    }
```

```

47 // 下面一大坨代码，无非就是把Channel连接返回回去了
48 if (cw != null) {
49
50     ChannelFuture channelFuture = cw.getChannelFuture();
51     if (channelFuture.awaitUninterruptibly(this.nettyClientConfig.getConnectTimeoutMillis())) {
52         if (cw.isOK()) {
53             log.info("createChannel: connect remote host[{}] success, {}", addr, channelFuture.toString());
54             return cw.getChannel();
55         } else {
56             log.warn("createChannel: connect remote host[" + addr + "] failed, " + channelFuture.toString(), channelFuture.cause());
57         }
58     } else {
59         log.warn("createChannel: connect remote host[{}] timeout {}ms, {}", addr, this.nettyClientConfig.getConnectTimeoutMillis(),
60             channelFuture.toString());
61     }
62 }
63 }
64 return null;
65 }

```

大家看下图，只要上面的Channel网络连接建立起来之后，我下面画红圈的地方，其实Broker和NameServer都会有一个Channel用来进行网络通信。



4、如何通过Channel网络连接发送请求？

接着我们看看，如何通过Channel网络连接发送请求出去？

其实核心入口就是下面的方法，之前讲过了。

```
RemotingCommand response = this.invokeSyncImpl(channel, request, timeoutMillis - costTime);
```

我们进入这个方法去看看，他是如何发送网络请求出去的？我同样写了详细的注释，大家注意看我的注释就行，一些乱七八糟的代码如果暂时看不明白也没关系的，关键是抓住重点的逻辑。

```

1 public RemotingCommand invokeSyncImpl(final Channel channel,
2                                     final RemotingCommand request,
3                                     final long timeoutMillis) {
4
5     final int opaque = request.getOpaque();
6     try {
7         final ResponseFuture responseFuture =
8             new ResponseFuture(channel, opaque, timeoutMillis, null, null);
9
10        this.responseTable.put(opaque, responseFuture);
11        final SocketAddress addr = channel.remoteAddress();
12
13        // 前面的代码你不用过多的关注，重点是下面的代码
14        // 你会发现，基于Netty来开发，核心就是基于Channel把你的请求写出去
15        channel.writeAndFlush(request).addListener(new ChannelFutureListener() {
16
17            @Override
18            public void operationComplete(ChannelFuture f) throws Exception {
19
20                if (f.isSuccess()) {
21                    responseFuture.setSendRequestOK(true);
22                    return;
23                } else {
24                    responseFuture.setSendRequestOK(false);
25                }
26                responseTable.remove(opaque);
27                responseFuture.setCause(f.cause());
28                responseFuture.putResponse(null);
29                log.warn("send a request command to channel <" + addr + "> failed.");
30            }
31        });
32

```

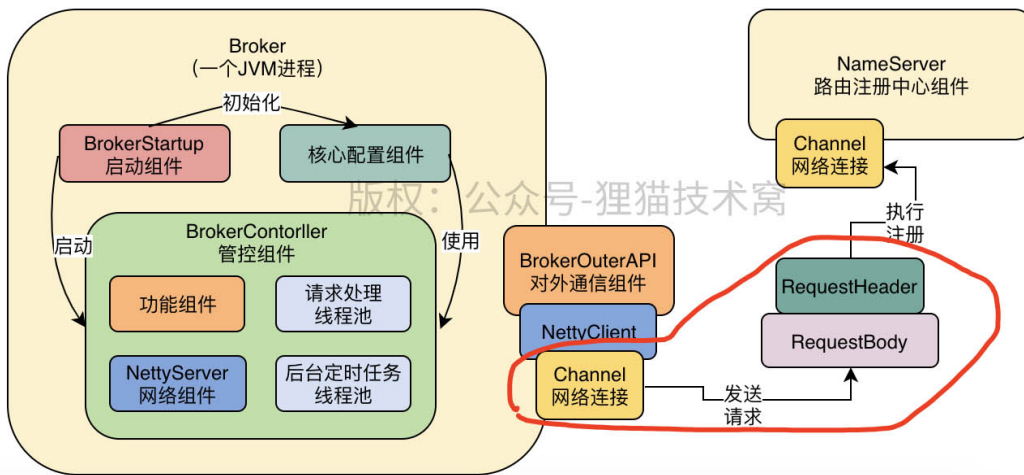
```

33        // 在下面这个地方，其实核心就是在等待请求的响应结果回来
34        RemotingCommand responseCommand =
35            responseFuture.waitResponse(timeoutMillis);
36
37        if (null == responseCommand) {
38            if (responseFuture.isSendRequestOK()) {
39                ...
40            } else {
41                ...
42            }
43        }
44        return responseCommand;
45    } finally {
46        this.responseTable.remove(opaque);
47    }
48
49 }

```

其实上面的代码只要关注我写的几行注释就可以了，抓住重点，就知道，最终底层其实就是基于Netty的Channel API，把注册请求发送给发送到了NameServer就可以了。

我们看下面的图，里面的红圈就展示了通过Channel发送网络请求出去的示意。



5、今日源码分析作业

希望大家参考今天的文章，把Broker注册的时候，在NettyClient底层进行Channel网络连接建立，以及通过Channel连接把注册请求发送出去的这些逻辑，都自己看一遍，同时好好理解我文章里画出来的图。

大家一定要注意，看源码的每一行细节是一个过程，加深你的理解，但是最终记在你脑子里的，一定是一幅一幅的图，这才是最终你自己沉淀下来的东西

如果大家有什么源码分析心得，可以发在评论区进行交流。

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）](#)
- [《互联网Java工程师面试突击》（第1季）](#)
- [《互联网Java工程师面试突击》（第3季）](#)

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）

Copyright © 2015-2020 深圳小鹅网络技术有限公司 All Rights Reserved. [粤ICP备15020529号](#)

 小鹅通提供技术支持