

图文 98 NameServer最终是如何启动Netty网络通信服务器的？

283 人次阅读 2020-02-17 09:04:39

详情 评论

NameServer最终是如何启动Netty网络通信服务器的？



狸猫技术

进店逛

相关频道



从 0 开
间件实站
已更新1



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：

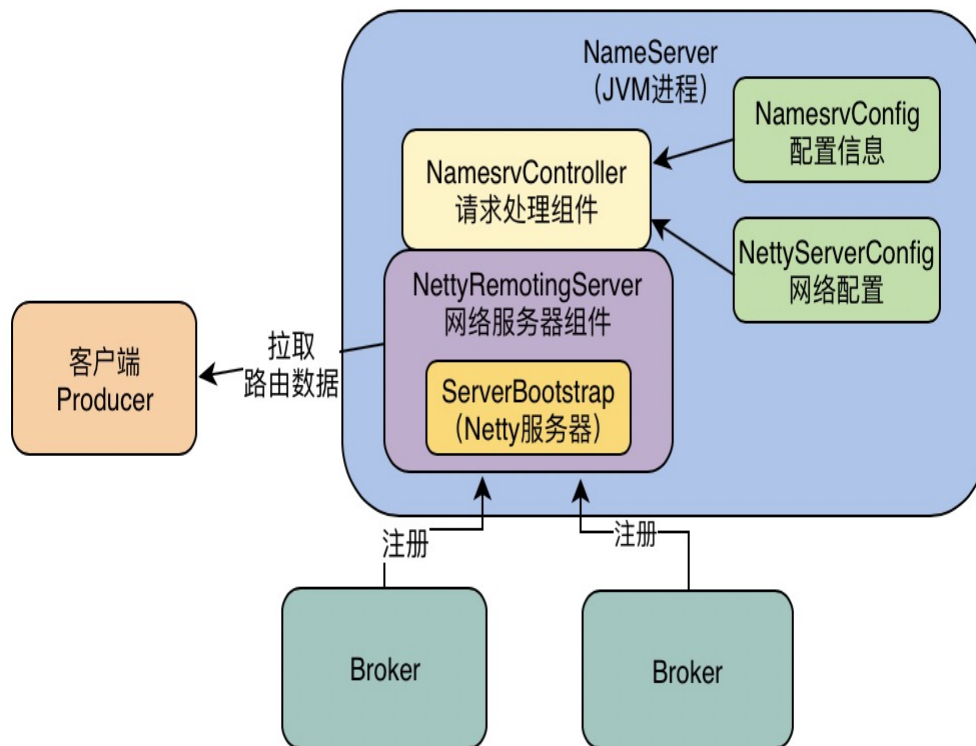
(点击下方蓝字试听)

[《从零开始带你成为MySQL实战优化高手》](#)

1、NamesrvController初始化过程的一些遗留代码

上次我们其实整体源码是分析到NamesrvController的initialize()方法，他在进行初始化，然后讲到他初始化了NettyRemotingServer，其中包含了一个Netty API开发的ServerBootstrap，说白了，就是一个网络服务器

我们看下下面的图，已经讲解的很明白了。



接着我们简单看下NamesrvController.initialize()方法遗留下来的还没讲的一些源码，那些源码暂时其实对我们来说都不是太重要，我稍微给大家讲一下就行了。

```

1 public boolean initialize() {
2
3     // 这个就不用多说了，一看就是在加载kv配置之类的东西，暂时不用关注
4     this.kvConfigManager.load();
5
6     // 这个是上一次分析的最重要的一行代码，初始化了Netty服务器
7     this.remotingServer = new NettyRemotingServer(
8         this.nettyServerConfig,
9         this.brokerHousekeepingService);
10
11     // 这个其实还是有一点重要的，他其实就是Netty服务器的工作线程池
12     this.remotingExecutor = Executors.newFixedThreadPool(
13         nettyServerConfig.getServerWorkerThreads(),
14         new ThreadFactoryImpl("RemotingExecutorThread_"));
15
16     // 下面这个，其实就是把工作线程池给了Netty服务器
17     this.registerProcessor();
18
19     // 下面这个就是启动了一个后台线程，执行定时任务的
20     // 其实你看下面的那行代码，scanNotActiveBroker()，就知道他是干嘛的
21     // 他就是定时扫描哪些Broker没发送心跳，就知道他挂了
22     // 暂时知道有这么个后台线程就行了，后续等我们需要的时候回来看
23     this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
24
25         @Override
26         public void run() {
27             NamesrvController.this.routeInfoManager.scanNotActiveBroker();
28         }
29     }, 5, 10, TimeUnit.SECONDS);
30
31     // 这个也是启动一个后台线程执行定时任务
32     // 不过他就很简单了，他就是定时打印kv配置信息，可以忽略他了
33     this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

```

```

34
35     @Override
36     public void run() {
37         NamesrvController.this.kvConfigManager.printAllPeriodically();
38     }
39 }, 1, 10, TimeUnit.MINUTES);
40
41 // 下面这个其实是FileWatch相关的，很多人肯定看不懂他干嘛的
42 // 其实看源码的时候，碰到这种代码是很正常的，看不懂就看不懂了，不用深究他
43 // 你就直接忽略和跳过下面的代码就行了
44 if (TlsSystemConfig.tlsMode != TlsMode.DISABLED) {
45     // Register a listener to reload SslContext
46     try {
47         fileWatchService = new FileWatchService(
48             new String[] {
49                 TlsSystemConfig.tlsServerCertPath,
50                 TlsSystemConfig.tlsServerKeyPath,
51                 TlsSystemConfig.tlsServerTrustCertPath
52             },
53             new FileWatchService.Listener() {
54                 boolean certChanged, keyChanged = false;
55                 @Override
56                 public void onChanged(String path) {
57                     if (path.equals(TlsSystemConfig.tlsServerTrustCertPath)) {
58                         log.info("...");
59                         reloadServerSslContext();
60                     }
61
62                     if (path.equals(TlsSystemConfig.tlsServerCertPath)) {
63                         certChanged = true;
64                     }
65
66                     if (path.equals(TlsSystemConfig.tlsServerKeyPath)) {
67                         keyChanged = true;
68                     }
69
70                     if (certChanged && keyChanged) {
71                         log.info("...");
72                         certChanged = keyChanged = false;
73                         reloadServerSslContext();
74                     }
75                 }
76             }
77         );
78     } catch (Exception e) {
79         log.warn("...");
80     }
81 }
82
83 private void reloadServerSslContext() {
84     ((NettyRemotingServer) remotingServer).loadSslContext();
85 }
86
87 } catch (Exception e) {
88     log.warn("...");
89 }
90
91 return true;
92 }

```

通过上面的源码分析，你会发现NamesrvController.initialize()方法，最核心的还是初始化Netty网络服务器，其他的就是启动了后台线程执行定时任务还重要一些，但是暂时我们还不用关注他，其他的代码你给忽略了也是可以的。

希望大家在跟着我逐步的分析RocketMQ核心源码的时候，也能够慢慢的掌握我分析源码的思路，你要明白，哪些是需要你重点关注的，哪些你可以暂时放着后续再来看，哪些是你干脆给忽略掉了。

2、回到start(controller)方法里看看

接着我们回到start(controller)方法里看看，大家看一下。

```
1 public static NamesrvController start(  
2     final NamesrvController controller) throws Exception {  
3  
4     if (null == controller) {  
5         throw new IllegalArgumentException(  
6             "NamesrvController is null");  
7     }  
8     boolean initResult = controller.initialize();  
9     if (!initResult) {  
10         controller.shutdown();  
11         System.exit(-3);  
12     }  
13     Runtime.getRuntime().addShutdownHook(  
14         new ShutdownHookThread(log, new Callable<Void>() {  
15  
16         @Override  
17         public Void call() throws Exception {  
18             controller.shutdown();  
19             return null;  
20         }  
21     }));  
22     controller.start();  
23     return controller;  
24 }
```

上面的controller.initialize()初始化这块代码我们实际上已经看完了，知道他已经初始化了Netty服务器出来，然后接着我们往下看他通过Runtime类注册了一个JVM关闭时候的shutdown钩子，就是JVM关闭的时候会执行上述注册的回调函数。

那个回调函数里执行了NamesrvController.shutdown()方法，其实我们都不用看里面的代码，都会知道，这里无非都是一些关闭Netty服务器的释放网络资源和线程资源的一些代码，如果大家一定要看，那我们看一下下面的代码。

```
1 public void shutdown() {  
2  
3     this.remotingServer.shutdown();  
4     this.remotingExecutor.shutdown();  
5     this.scheduledExecutorService.shutdown();  
6  
7     if (this.fileWatchService != null) {  
8         this.fileWatchService.shutdown();  
9     }  
10 }
```

感觉如何？是不是发现他就是在关闭NettyRemotingServer释放网络资源，然后关闭RemotingExecutor就是释放Netty服务器的工作线程池的资源，还有关闭ScheduledExecutorService就是释放执行定时任务的后台线程资源。

其实这里最关键的一行代码是：controller.start()。说白了，他已经初始化了Netty服务器了，但是现在还没启动，没启动的话，Netty服务器就不会监听9876这个默认的端口号，那么NameServer就什么也干不了。

所以此时，他必须要对NamesrvController组件做一个启动操作，这样的话，就可以把他内部的Netty服务器给启动了。

3、Netty服务器是如何启动的？

接着我们进入controller.start()方法内部看看，如下。

```

1 public void start() throws Exception {
2
3     this.remotingServer.start();
4
5     if (this.fileWatchService != null) {
6         this.fileWatchService.start();
7     }
8 }

```

其实这里就很清晰了，这个NamesrvContorller启动，核心就是在启动NettyRemotingServer，也就是Netty服务器。在remotingServer.start()方法里，有很多的代码，我给大家逐步的分析各个片段。

```

1 // 很多朋友可能对Netty开发不太熟悉，但是其实下面这里
2 // 主要核心就是基于Netty的API去配置和启动一个Netty网络服务器
3 ServerBootstrap childHandler =
4
5     // 下面这个地方，基于ServerBootstrap的group方法
6     // 对Netty服务器进行各种网络上的配置
7     // 这些配置就不去细讲了，因为都是跟Netty相关的
8     this.serverBootstrap.group(
9         this.eventLoopGroupBoss,
10        this.eventLoopGroupSelector).channel(useEpoll() ?
11                                            EpollServerSocketChannel.class :
12                                            NioServerSocketChannel.class)
13
14        .option(ChannelOption.SO_BACKLOG, 1024)
15        .option(ChannelOption.SO_REUSEADDR, true)
16        .option(ChannelOption.SO_KEEPALIVE, false)
17        .childOption(ChannelOption.TCP_NODELAY, true)
18        .childOption(ChannelOption.SO_SNDBUF,
19                    nettyServerConfig.getServerSocketSndBufSize())
20        .childOption(ChannelOption.SO_RCVBUF,
21                    nettyServerConfig.getServerSocketRcvBufSize())
22
23        .localAddress(new InetSocketAddress(this.nettyServerConfig.getListenPort()))

```

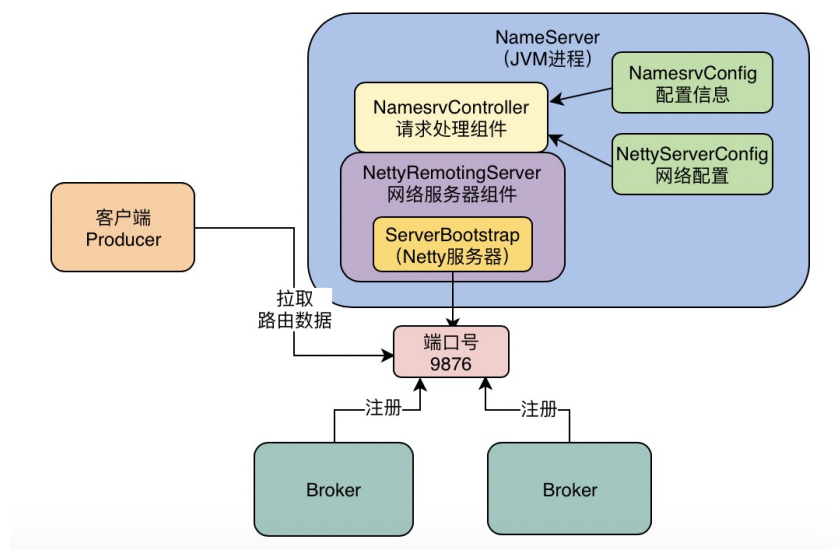
```

24 // 下面这个地方，说白了，就是设置了一大堆的网络请求处理器
25 // 只要Netty服务器收到一个请求，那么就会依次使用下面的处理器来处理请求
26 // 比如说handShakeHandler可能就是负责连接握手
27
28 // NettyDecoder是负责编码解码的，IdleStateHandler是负责连接空闲管理的
29 // connectionManageHandler是负责网络连接管理的
30 // serverHandler是负责最关键的网路请求的处理的
31 .childHandler(new ChannelInitializer<SocketChannel>() {
32     @Override
33     public void initChannel(SocketChannel ch) throws Exception {
34         ch.pipeline()
35             .addLast(defaultEventExecutorGroup,
36                 HANDSHAKE_HANDLER_NAME,
37                 handshakeHandler)
38             .addLast(defaultEventExecutorGroup,
39                 encoder,
40                 new NettyDecoder(),
41                 new IdleStateHandler(0, 0, nettyServerConfig.
42                     getServerChannelMaxIdleTimeSeconds()),
43                 connectionManageHandler,
44                 serverHandler
45             );
46     }
47 });
48 try {
49     // 下面的代码，其实就是启动Netty服务器了
50     // 那个bind方法，说白了，其实就是绑定和监听一个端口号
51     ChannelFuture sync = this.serverBootstrap.bind().sync();
52     InetSocketAddress addr = (InetSocketAddress) sync.channel().localAddress();
53     this.port = addr.getPort();
54 } catch (InterruptedException e1) {
55     ...
56 }

```

接着回看上面的一行代码：.localAddress(new InetSocketAddress(this.nettyServerConfig.getListenPort()))。这行代码，其实就是设置了Netty服务器要监听的端口号，默认就是9876

因此到此为止，你可以理解为Netty服务器启动了，开始监听端口号9876了，此时我们看下面的图，图里就展示出了Netty服务器监听端口号的这个示意。



4、总结

到此为止，我们已经初步了解了NameServer是如何启动的了，了解到他最核心的就是基于Netty实现了一个网络服务器，然后监听默认的9876端口号，可以接收Broker和客户端的网络请求。

接着明天开始我们就要研究一下NameServer启动好之后，Broker是如何启动的，如何向NameServer进行注册，如何进行心跳，NameServer是如何管理Broker的。

5、今天作业

今天给大家留的小作业，就是仔细看看NettyRemotingServer的start()方法，仔细看一下里面是如何基于Netty API实现一个网络服务器的配置和启动的。请大家认真完成作业，有什么问题或者心得，欢迎在评论区留言

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