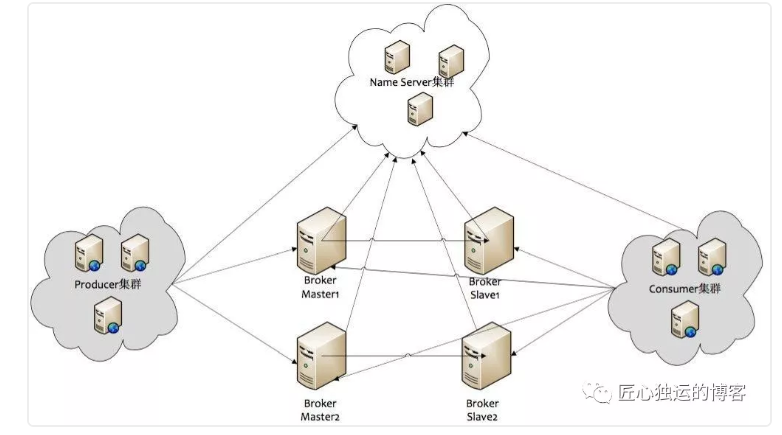
消息中间件-RocketMQ的RPC通信

RocketMQ使用Netty进行网络通信，Netty是一个封装了JDK的NIO库的高性能网络通信开源框架，提供异步、事件驱动的网络应用程序框架和工具，使用Netty作为RocketMQ底层通信库的优点：

* Netty的编程API使用简单，开发门槛低，无序编程者去关注和了解NIO编程模型和概念
* 对于编程者来说，可根据业务的要求进行定制化地开发，通过Netty的ChannelHandler对通信框架进行灵活的定制化扩展
* Netty框架本身拆分/解包，异常检测等机制，让编程者可以从Java NIO的繁琐细节中解脱，而只需要关注业务处理逻辑
* Netty对JDK NIO进行了优化
* Netty框架内部对线程，Selector做了细节优化，精心设计Reactor多线程模型，可以实现非常高效的并发处理
* Netty在多个开源项目中都得到充分验证

其整体部署架构如下图所示：



在RocketMQ中，Broker启动后注册到NameServer，随后每隔30s定期向NameServer上报Topic路由信息。Producer作为客户端发送消息，会从NameServer上拉取Topic的路由信息，然后选择一个MessageQueue向Broker消息发送。

RocketMQ的RPC通信部分采用了1+N+M1+M2的Reactor多线程模型，对网络通信部分进行了一定的扩展和优化，其设计思想：

1. 分而治之

网络请求连接的完整处理可以分为接受（accept）、数据读取（read）、解码/编码（decode/

encode）、业务处理（process）、发送响应（send）这几个步骤。Reactor模型将每个步骤都映射为一个任务，服务端线程执行的最小逻辑单元不再是一个完整的而网络请求，而是这个任务，且采用以非阻塞方法执行。

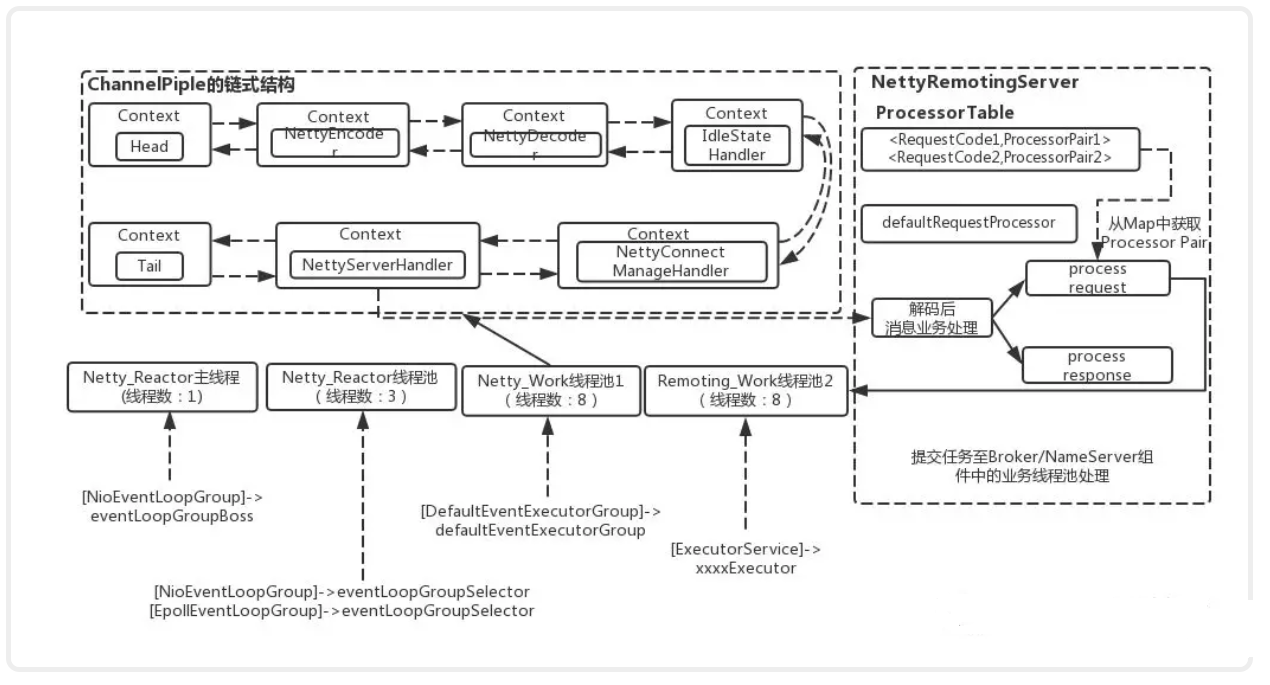
1. 事件驱动

每个任务对应特定网络事件，当任务准备就绪时，Reactor收到对应的网络事件通知，并将任务分发给绑定了对应网络事件的Handler执行

1. **RocketMQ模型框架**

## 1.1 多线程模型框架

RocketMQ的RPC通信层的Netty多线程模型框架如下图：



上面框图描述了RocketMQ的NettyRemotingServer的Reactor多线程模型

* Reactor主线程（EventLoopGroupBoss ->1），负责监听TCP网络连接请求，建立好连接后丢给Reactor线程池（eventLoopGroupSelector）
* EventLoopGroupSelector(N)，负责将建立好连接的Socket注册到selector上去，然后监听真正的网络数据
* EventLoopGroupSelector获取网络数据后，丢给Worker线程池(defaultEventExecutorGr

Oup,M1)，这里的线程池专门用来处理网络通信相关的业务，包括编码/解码、空闲链接管理、网络连接管理以及网络请求处理

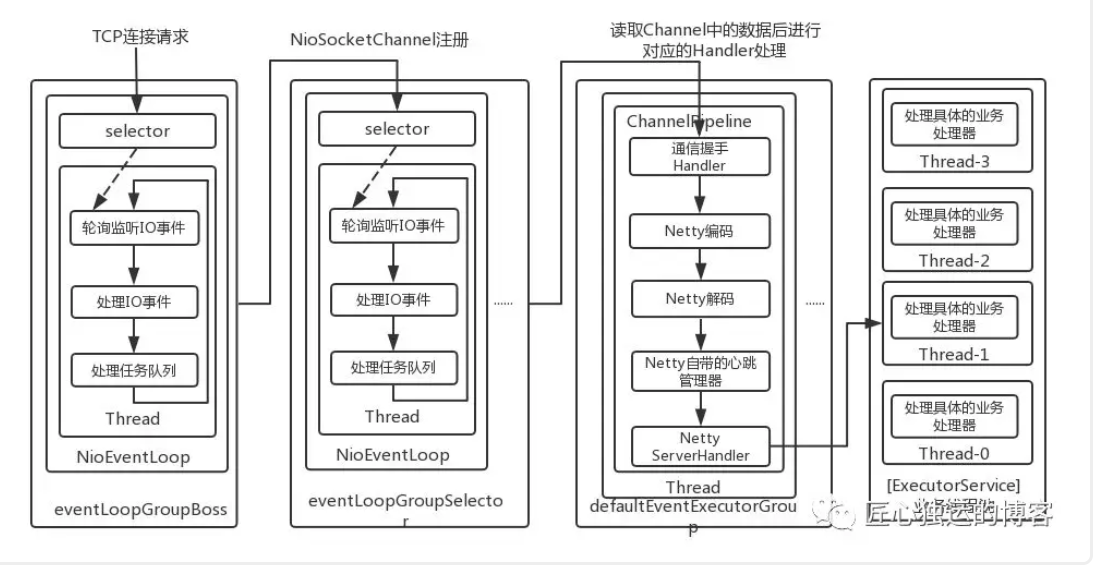
* 处理业务的操作放在业务线程池中执行，根据RemotingCommand的业务请求码在本地缓存比那里processTable中找到对应的Processor，然后封装成Task，提交到对应的业务processor处理线程池来执行（sendMessageExecutor， M2）

## **1.2 各线程及执行序列**

各线程及处理的请求如下表所示：

|  |  |  |
| --- | --- | --- |
| 线程数 | 线程名 | 线程具体说明 |
| 1 | NettyBoss\_%d | Reactor主线程 |
| N | NettyServerEPOLLSelector%d%d | Reactor线程池 |
| M1 | NettyServerCodecThread\_%d | Worker线程池 |
| M2 | RemotingExecutorThread\_%d | 业务Processor处理线程池 |

其执行序列如下所示：

****

## **1.3 Reactor多线程的代码实现**

以上是RocketMQ的Reactor多线程整体的设计与流程，在NettyRemotingServer实例初始化时，会初始化各个相关变量：

*serverBootstrap*

*nettServerConfig参数*

*channelEventListener监听器*

*eventLoopGroupBoss*

*eventLoopGroupSelector*

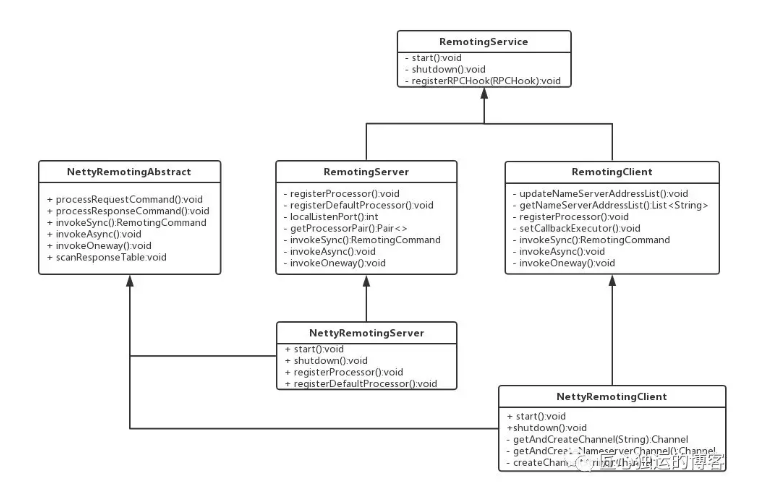
其初始化源码如下：



整体可以看出RocketMQ的RPC通信借助Netty的多线程模型，其服务端监听线程和IO线程分离，同时将RPC通信层的业务逻辑和处理具体业务的线程进一步分离。

1. **RocketMQ 通信模块**

Rocketmq-remoting模块是RocketMQ消息队列中负责网络通信的模块，其类结构如下所示：



1. RemotingService，最上层的接口，提供三个方法：

*void start();*

*void shutdown();*

*void registerRPCHook(RPCHook rpcHook);*

1. RemotingClient/RemotingServer，两个接口继承最上层接口RemotingService，分别为Client和Server提供必需的方法
2. NettyRemotingAbstract，Netty通信处理的抽象类，定义并封装了Netty处理的公共处理方法
3. NettyRemotingClient以及NettyRemotingServer

## **2.1 消息的编解码**

为了实现Client和Server之间的消息发送和收到消息的处理，需要对消息进行编码，在RocketMQ中，RemotingCommand这个类在消息传输过程中对所有数据内容进行封装，不但保护所有数据结构，还包含编解码的操作，下面是Broker向NameServer发送心跳注册的报文：



RocketMQ通信协议的格式：



1. 消息长度，4个字节存储
2. 序列化类型&消息头长度，4个字节，第一个字节序列化类型，后面三个字节消息头长度
3. 消息头数据，经过序列化的消息头数据
4. 消息主题数据，消息主题的二进制字节数据内容

消息的编解码在RemotingCommand类的encode和decode方法中完成，在进行传输过程中具体的调用由NettyEncoder和NettyDeCoder来完成

*public void initChannel(SocketChannel ch) throws Exception {  
 ch.pipeline()  
 .addLast(defaultEventExecutorGroup, HANDSHAKE\_HANDLER\_NAME,  
 new HandshakeHandler(TlsSystemConfig.tlsMode))  
 .addLast(defaultEventExecutorGroup,  
 new NettyEncoder(),  
 new NettyDecoder()*

*......*

*}*

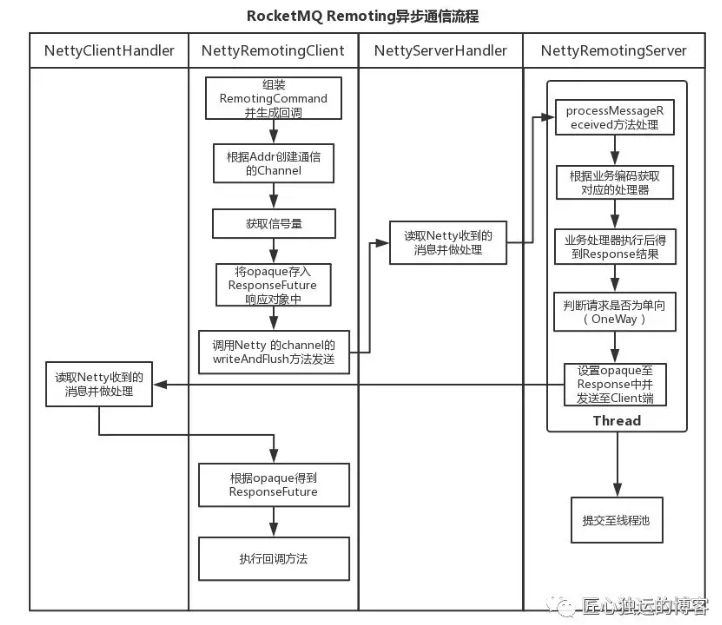
以NettyEncoder为例，编码执行过程：

*public void encode(ChannelHandlerContext ctx, RemotingCommand remotingCommand, ByteBuf out) throws Exception {  
 ByteBuffer header = remotingCommand.encodeHeader();  
 out.writeBytes(header);  
 byte[] body = remotingCommand.getBody();  
 if (body != null) {  
 out.writeBytes(body);  
 } .....*

*......*

## **2.2 消息的通信方式和流程**

在RocketMQ消息队列中支持的通信方法包括：同步，异步和单向。其中同步相对简单，下图是异步通信的整体流程：



### 2.2.1 客户端消息发送

客户端异步通信调用接口invokeAsync，NettyRemotingClient其执行如下： *public void invokeAsync(String addr, RemotingCommand request, long timeoutMillis, InvokeCallback invokeCallback) ....{*

*//根据Addr获取相应的channel，本地缓存中没有则创建  
 final Channel channel = this.getAndCreateChannel(addr);  
 if (channel != null && channel.isActive()) {  
 try {  
 if (this.rpcHook != null) {  
 this.rpcHook.doBeforeRequest(addr, request);  
 }*

*//调用channel.writeAndFlush(request)来发送  
 this.invokeAsyncImpl(channel, request, timeoutMillis, invokeCallback);  
 } ....*

*}*

**2.2.2 Server端消息接收及处理**

Server端接收消息的处理入口在NettyServerHandler类的channelRead0方法中，其中调用了processMessageReceived方法，其中服务端最为重要的处理请求方法实现如下：

*public void processMessageReceived(ChannelHandlerContext ctx, RemotingCommand msg) throws Exception {  
 final RemotingCommand cmd = msg;  
 if (cmd != null) {  
 switch (cmd.getType()) {  
 case REQUEST\_COMMAND:  
 processRequestCommand(ctx, cmd);  
 break;  
 case RESPONSE\_COMMAND:  
 processResponseCommand(ctx, cmd);  
 break;  
 default:  
 break;  
 }  
 }  
}*

请求处理方法中根据RemotingCommand的请求业务代码来匹配相应的业务处理器。将结果返回给客户端，在通信模块中给异步回调处理提供了接口，如下：

*public void invokeAsync(String addr*

*, RemotingCommand request,*

*long timeoutMillis,*

*InvokeCallback invokeCallback)*

其中invokeCallback处理返回结果。

https://mp.weixin.qq.com/s?\_\_biz=MzU5OTUyMTAwNQ==&mid=2247483750&idx=1&sn=ff95737cce00b7bcfe0d5cfb28b8a3c5&chksm=feb2e2e4c9c56bf2f174aa94d1df489c036cf6ed3cdb38fe1d54d327bfbdda26ff60f6a07589&scene=21#wechat\_redirect