图文 59 探秘黑科技：RocketMQ 是如何基于Netty扩展出高性能网络的?

545 人次阅读 2019-12-20 09:26:53

详情 评论

探秘黑科技：RocketMQ是如何基于Netty扩展出高性能网络通信架构的?

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：

全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、额外的加餐：探秘RocketMQ的一些黑科技

目前咱们的专栏行进到这里为止，已经借着小猛的视角给大家分了一下RocketMQ运行的底层原理，本来应该到上一讲就结束这个阶段了

但是后来我们考虑了一下，还希望给大家再额外做一点加餐，单独加出来3讲RocketMQ黑科技探秘的内容，让大家能够对RocketMQ一些底层的比较关键的实现原理有一定的了解。

我们将会额外给大家讲解以下RocketMQ的核心底层原理：

RocketMQ是如何基于Netty扩展出高性能网络通信架构的？
基于mmap内存映射实现CommitLog磁盘文件的高性能读写

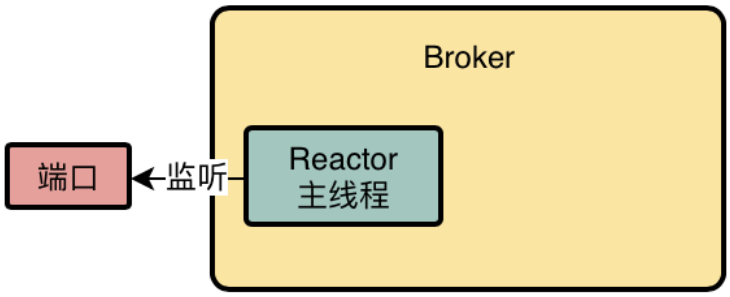
今天这一讲，我们会给大家介绍一下RocketMQ中的Broker的网络通信的架构是如何基于Netty进行扩展的。

即使有些朋友对Netty没什么了解也是不要紧的，因为我们会站在线程和网络等较为基础的角度，给大家一步一图去分析。

2、Reactor主线程与长短连接

首先，作为Broker而言，他会有一个Reactor主线程。大家是不是对这个“Reactor主线程”这个称呼感觉有点恐惧？感觉出来一些特别厉害的不明所以的名词，让人有点胆怯，不敢继续往下看

没关系，你先别管“Reactor”是个什么东西，总之就先知道有这么个名字就行了，我们看下面的图里，你会发现Broker里有这么一个名字的线程，而且这个线程是负责监听一个网络端口的，比如监听个2888，39150这样的端口。



接着假设我们有一个Producer他现在想要跟Broker建立一个TCP长连接，可能有的朋友对这个长连接、短连接，也有点不明所以

没关系，一句话解释一下短连接：如果你要给别人发送一个请求，必须要建立连接 -> 发送请求 -> 接收响应 -> 断开连接，下一次你要发送请求的时候，这个过程得重新来一遍

每次建立一个连接之后，使用这个连接发送请求的时间是很短的，很快就会断开这个连接，所以他存在时间太短了，就是短连接。

长连接的话，就是反过来的意思，你建立一个连接 -> 发送请求 -> 接收响应 -> 发送请求 -> 接收响应 -> 发送请求 -> 接收响应

大家会发现，当你建立好一个长连接之后，可以不停的发送请求和接收响应，连接不会断开，等你不需要的时候再断开就行了，这个连接会存在很长时间，所以是长连接。

那么TCP长连接是什么意思呢？

如果你对网络没太多的了解，简单理解为TCP就是一个协议，所谓协议的意思就是，按照TCP这个协议规定好的步骤建立连接，按照他规定好的步骤发送请求。

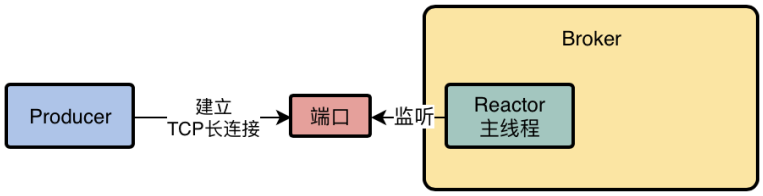
比如你要建立一个TCP连接，必须先给对方发送他规定好的几个数据，然后人家按照规定返回给你几个数据，你再给人家发送几个数据，一切都按TCP的规定来。按照规定来，大家就可以建立一个TCP连接。

所以TCP长连接，就是按照这个TCP协议建立的长连接。

3、Producer和Broker建立一个长连接

接着比如有一个Producer他就要跟Broker建立一个TCP长连接了，此时Broker上的这个Reactor主线程，他会在端口上监听到这个Producer建立连接的请求

我们看下面的图，我示意了这个过程。



接着这个Reactor主线程就专门会负责跟这个Producer按照TCP协议规定的一系列步骤和规范，建立好一个长连接。

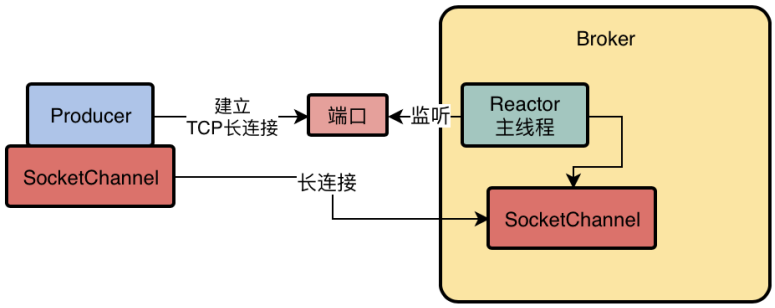
但是现在问题来了，在Broker里用什么东西代表跟Producer之间建立的这个长连接呢?

答案是：SocketChannel

Producer里面会有一个SocketChannel，Broker里也会有一个SocketChannel，这两个SocketChannel就代表了他们俩建立好的这个长连接。

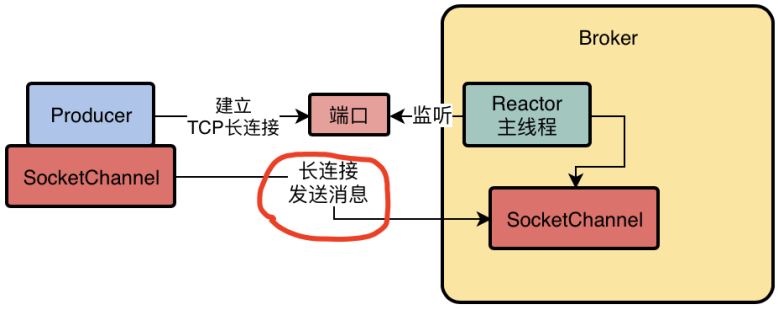
可能有人又会对这个SocketChannel名词感到很恐惧了，但是大家先别管这个SocketChannel，你只要知道他们俩建立好连接之后，就各自会有一个SocketChannel，俩SocketChannel配对起来就代表了一个连接

我们看下面的图



接着下一个问题来了，既然Producer和Broker之间已经通过SocketChannel维持了一个长连接了，接着Producer是不是应该会通过这个SocketChannel去发送消息给Broker?

没错，就是这么做的，我们看下面的图。



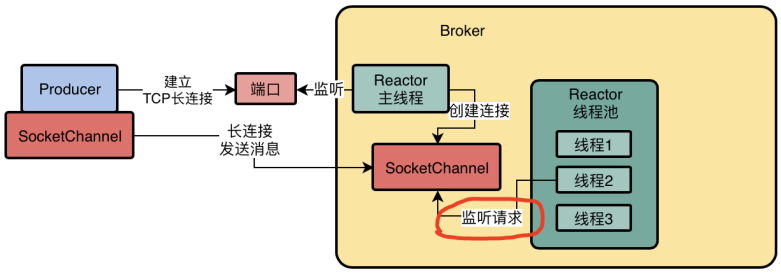
4、基于Reactor线程池监听连接中的请求

但是这个时候别急！我们还不能让Producer发送消息给Broker，因为虽然我们有一个SocketChannel组成的长连接，但是他仅仅是一个长连接而已!

假设Producer此时通过SocketChannel发送消息给到Broker那边的SocketChannel了，但是Broker里是哪个线程来负责从SocketChannel里获取这个消息呢？这是一个很大的问题!

所以我们接着要引入一个概念，就是Reactor线程池，你也先别管这里的“Reactor”是什么意思，你只要知道有一个这个名字的线程池就可以了，这个线程池里默认是3个线程!

然后Reactor主线程建立好的每个连接SocketChannel，都会交给这个Reactor线程池里的其中一个线程去监听请求。



好，现在有了Reactor线程池这个概念，我们总算是可以让Producer发送请求过来了，他发送一个消息过来到达Broker里的SocketChannel，此时Reactor线程池里的一个线程会监听到这个SocketChannel中有请求到达了！

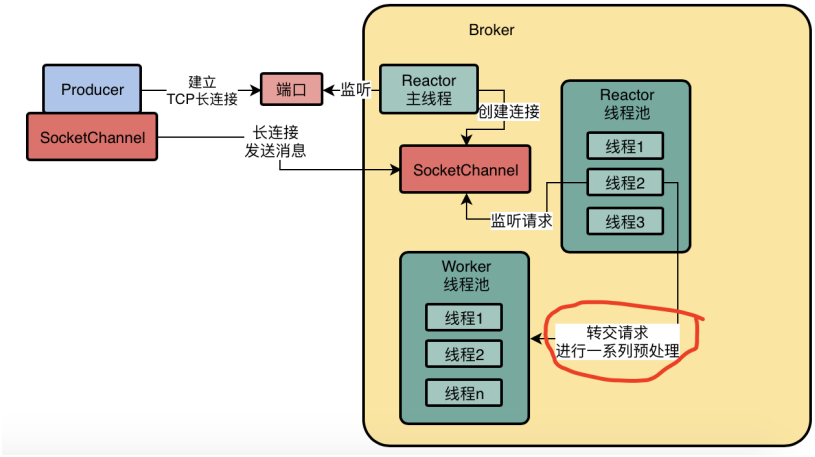
5、基于Worker线程池完成一系列准备工作

接着Reactor线程从SocketChannel中读取出来一个请求，这个请求在正式进行处理之前，必须就先要进行一些准备工作和预处理，比如SSL加密验证、编码解码、连接空闲检查、网络连接管理，诸如此类的一些事

那么问题又来了，这些事让谁来干呢?

这个时候需要引入一个新的概念，叫做Worker线程池，他默认有8个线程，此时Reactor线程收到的这个请求会交给Worker线程池中的一个线程进行处理，会完成上述一系列的准备工作

我们看下面的图



6、基于业务线程池完成请求的处理

那么现在如果Worker线程完成了一系列的预处理之后，比如SSL加密验证、编码解码、连接空闲检查、网络连接管理，等等，接着就需要对这个请求进行正式的业务处理了！我们来给大家举个例子。

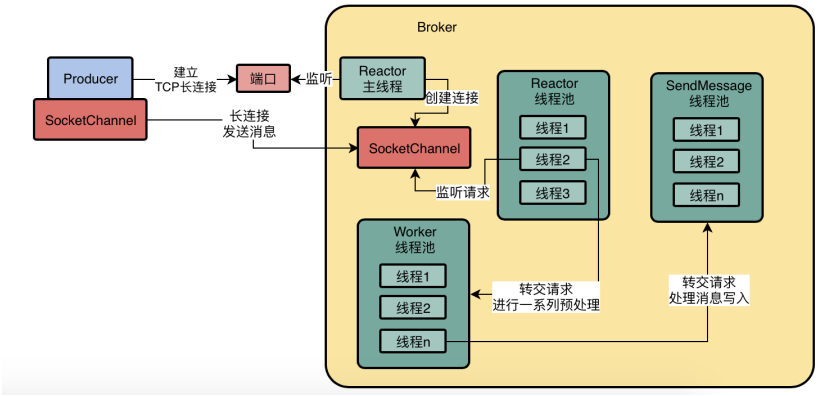
比如对于你发送过来的消息，大家还记得我们之前讲解的Broker数据存储机制吗?

你接收到了消息，肯定是要写入CommitLog文件的，后续还有一些ConsumeQueue之类的事情需要处理，类似这种操作，就是业务处理逻辑。

这个时候，就得继续把经过一系列预处理之后的请求转交给业务线程池

比如对于处理发送消息请求而言，就会把请求转交给SendMessage线程池，而且如果大家还有一点点印象的话，其实在之前讲集群部署的时候，我们讲到过这个SendMessage线程是可以配置的，你配置的越多，自然处理消息的吞吐量越高。

我们看下面的图，就是引入了一个业务线程池的概念。



7、为什么这套网络通信框架会是高性能以及高并发的?

最后我们来思考一下，为什么这套网络通信框架会是高性能以及高并发的呢?

原因很简单，假设我们只有一个线程来处理所有的网络连接的请求，包括读写磁盘文件之类的业务操作，那么会导致我们的并发能力必然很低。

一个线程每秒能处理多少个请求啊？是不是。

所以必须专门分配一个Reactor主线程出来，就是专门负责跟各种Producer、Consumer之类的建立长连接。

一旦连接建立好之后，大量的长连接均匀的分配给Reactor线程池里的多个线程。

每个Reactor线程负责监听一部分连接的请求，这个也是一个优化点，通过多线程并发的监听不同连接的请求，可以有效的提升大量并发请求过来时候的处理能力，可以提升网络框架的并发能力。

接着后续对大量并发过来的请求都是基于Worker线程池进行预处理的，当Worker线程池预处理多个请求的时候，Reactor线程还是可以有条不紊的继续监听和接收大量连接的请求是否到达。

而且最终的读写磁盘文件之类的操作都是交给业务线程池来处理的，当他并发执行多个请求的磁盘读写操作的时候，不影响其他线程池同时接收请求、预处理请求，没任何的影响。

所以最终的效果就是:

Reactor主线程在端口上监听Producer建立连接的请求，建立长连接
Reactor线程池并发的监听多个连接的请求是否到达
Worker请求并发的对多个请求进行预处理
业务线程池并发的对多个请求进行磁盘读写业务操作

这些事情全部是利用不同的线程池并发执行的！任何一个环节在执行的时候，都不会影响其他线程池在其他环节进行请求的处理!

这样的一套网络通信架构，最终实现的效果就是可以高并发、高吞吐的对大量网络连接发送过来的大量请求进行处理，这是保证Broker实现高吞吐的一个非常关键的环节，就是这套网络通信架构。

因此对于这类中间件，如果你给他部署在高配置的物理机上，有几十个CPU核，那么此时你可以增加他的各种线程池的线程数量，这样就可以让各个环节同时高并发的处理大量的请求，由大量的CPU核来支持大量线程的并发工作。

8、对今日内容做一点小的总结

今天我们给大家讲解了一下RocketMQ中的网络通信架构，我们考虑很多朋友不一定会对TCP、NIO、TCP、BIO、Netty这些技术有了解，所以本文采取的是基于线程和网络这两个最基本的概念进行讲解的思路。

因为只要你是懂Java语言的，一定知道最基本的线程和网络这两个概念有一定的了解，顺着本文的思路看下来，相信大家一定可以对Broker的网络通信架构有一定的理解，核心的思路和架构应该是明白了。

End

狸猫技术窝其他精品专栏推荐：

《从零开始带你成为JVM实战高手》

《21天Java 面试突击训练营》（分布式篇）（现更名为：**互联网Java工程师面试突击第2季**）

互联网Java工程师面试突击（第1季）

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

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）**