06 如何从整体上把握Netty的架构

更新时间：2020-07-15 09:57:03

> 我们活着不能与草木同腐，不能醉生梦死，枉度人生，要有所作为。——方志敏

前言

你好，我是彤哥。

前面几节，我们一起学习了 Netty 的发迹史，以及 IO 的基础知识，从本节开始我们将正式进入 Netty 的学习。

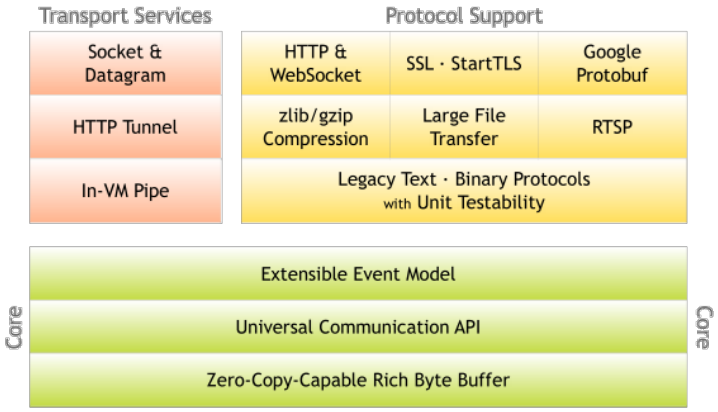
不过，很多同学可能会有所疑问，学习 Netty 应该从哪里切入呢？笔者认为，当先从 Netty 的整体结构入手，从整体到局部，从宏观到微观，是一种不错的学习方法。

所以，本节，我将从架构设计、模块设计两个维度来从宏观上分析 Netty 的整体结构。

好了，让我们进入今天的学习吧。

架构设计

关于架构设计，让我们看看 Netty 官方网站怎么说：



官网只给了这么一张图片，啥也没说 ^^。

从这张架构图上，我们可以看到 Netty 的分层特别清晰：

- Core，核心层，主要定义一些基础设施，比如事件模型、通信 API、缓冲区等。
- Transport Service，传输服务层，主要定义一些通信的底层能力，或者说是传输协议的支持，比如 TCP、UDP、HTTP 隧道、虚拟机管道等。
- Protocol Support，协议支持层，这里的协议比较广泛，不仅仅指编解码协议，还可以是应用层协议的编解码，比如 HTTP、WebSocket、SSL、Protobuf、文本协议、二进制协议、压缩协议、大文件传输等，基本上主流的协议都支持。

Netty 的核心在于其可扩展的事件模型、通用的通信 API、基于零拷贝的缓冲区等，它们就像厨房中的锅碗瓢盆和油盐酱醋，有了厨房的这些基础设施，才能做出美味的佳肴。然后，我们才能定义更上层的服务，比如传输协议、序列化协议、编解码协议等。传输协议就像是我们的点餐系统，是通过普通的菜单点餐，还是扫码点餐等。编解码协议更像是怎么盛放佳肴的问题，是通过好看的餐盘，还是通过纸质饭盒，当然，有了基础设施和传输服务，怎么编解码传输的内容就更简单了，甚至，我们可以定义自己的编解码方式，比如，JSON 序列化。

好的架构就像指南针，有了这份好的架构设计，才能指引我们更有效地 coding，当然，对于学习者也是一份很重要的参考指南。

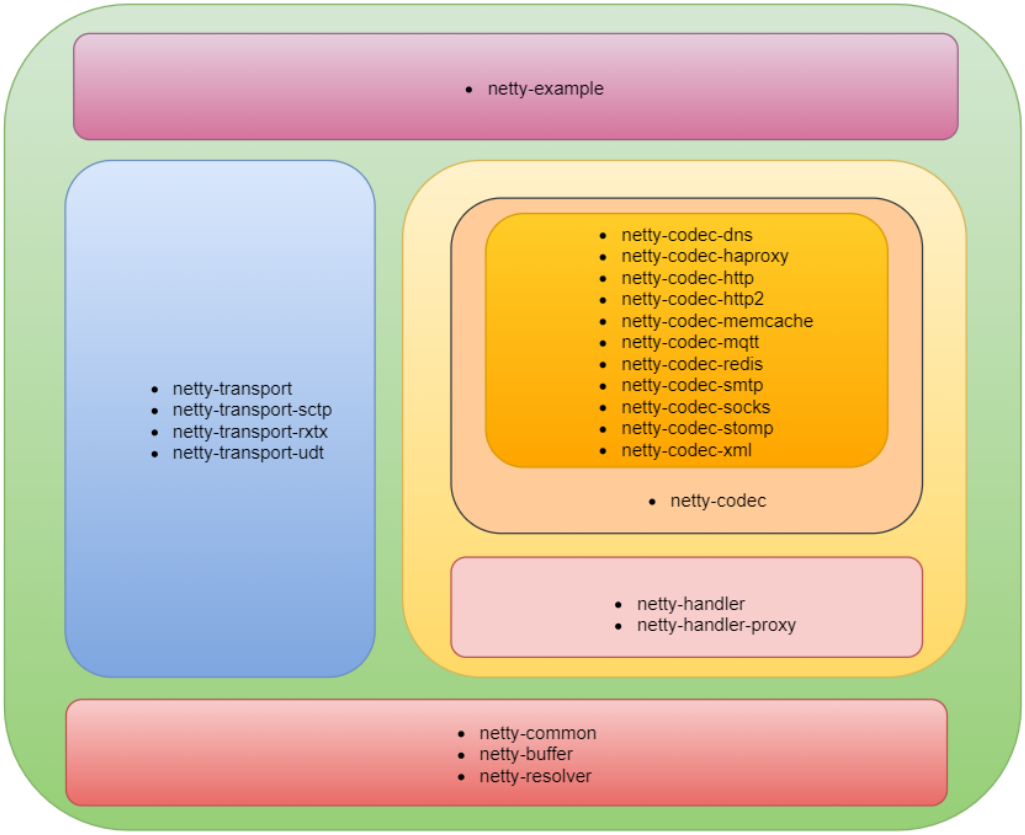
看了这份架构，相信你的心中已经有了很多疑惑，什么是事件模型？如何扩展？怎么做到通信 API 的通用性？零拷贝又是什么东西？如何自定义编解码？等等。对于这些问题，我们后面都会解答，这里先埋个伏笔哦。

好了，看完了架构设计，我们再来看看模块设计。

模块设计

Netty 是一个多模块的项目，它的模块设计得非常完美，使得对于 Netty 的扩展相当方便，至于怎么个方便法，请听我娓娓道来。

我大概数了下，一共有 40 个 jar 包，我们来归下类：



细心的同学有没有发现一个很有兴趣的点？哈哈，跟上面的架构图几乎一模一样？对的，就是一样一样的，让我们自底向上、自左向右来分析分析：

netty-common

`netty-common` 包主要定义了一些工具类，归纳下来大概如下：

- 通用的工具类，比如 `StringUtil` 等
- 对于 JDK 原生类的增强，比如 `Future`、`FastThreadLocal` 等
- Netty 自定义的并发包，比如 `EventExecutor` 等
- Netty 自定义的集合包，主要是对 HashMap 的增强

注意，其它所有模块都依赖于 common 包。

> 如果你是直接从 github 克隆下来的工程，会发现并没有集合包，那是怎么回事呢？
>
> 仔细观察一下 common 包，会发现 src/main 目录下还有两个目录：script 和 templates，一个是 groovy 脚本，一个是模板。粗略地看一下，会发现很像生成集合的代码模板，那么，怎么生成呢？无疑是在 maven 的 pom 文件中定义的，打开 pom.xml，搜索 groovy，果然能找到，而且有 `generate-sources` 这样的字眼，还配置了上面脚本的路径，所以，很简单，只要 compile 一下就可以了，当然，这里生成的不是源码，而是生成 class 文件到 target 下面。

netty-buffer

Netty 自己实现的 Buffer，不同于 JDK 原生的 ByteBuffer 那么难用，Netty 的 ByteBuf 要好用得多，等后面讲例子你就知道差距了。而且 Netty 做了很多优化，比如池化 Buffer、组合 Buffer 等都是在这个包里，Netty 把性能优化到了极致。

netty-resolver

主要是做地址解析用的。

其实，上面三个都可以看作是工具类，它们是构成整个 Netty 的基石，是底盘，很重要。

netty-transport

`netty-transport` 主要定义了一些服务于传输层的接口和类，比如 Channel、ChannelHandler、ChannelHandlerContext、EventLoop 等，这些接口和类都非常的酷，它们支撑起了 Netty 的半边天。

而且，`netty-transport` 还实现了对于 TCP、UDP 通信协议的支持，另外三个包 `netty-transport-sctp`、`netty-transport-rxtx`、`netty-transport-udt` 也是对不同协议的支持，不过后两个已经废弃了，为什么呢？原因很任性，写得不好，不好用，就像废弃 Netty5.0 一样，任性！

> TCP，传输控制协议，Java 中一般用 SocketXxx、ServerSocketXxx 表示基于 TCP 协议通信。
>
> UDP，用户数据报文协议，Java 中一般用 DatagramXxx 表示基于 UDP 协议通信，Datagram，数据报文的意思。
>
> SCTP，流控制传输协议。
>
> RXTX，串口通讯协议。
>
> UDT，基于 UDP 的数据传输协议。

netty-handler

`netty-handler` 中定义了各种不同的 Handler，满足不同的业务需求，这些 Handler 都是 Netty 中非常棒的功能，比如，IP 过滤、日志、SSL、空闲检测、流量整形等，有了这些 Handler，我们不仅能让我们的程序可运行，更能使我们的程序安全地运行，非常棒。

如果说 `netty-transport` 让你觉得 Netty 很酷，那 `netty-handler` 绝对会让你爱上了 Netty。

netty-codec

`netty-codec` 中定义了一系列编码解码器，比如，base64、json、marshalling、protobuf、serializaion、string、xml 等，几乎市面上能想到的编码、解码、序列化、反序列化方式，Netty 中都可以支持，它们是一类特殊的 ChannelHandler，专门负责编解码的工作。

> 编码和解码实际并没有明确的界限，是指把内容从一种形式转换成另一种形式，就像把水变成冰，把冰变成水一样，我们需要施加一些手段来达到这个目的，这种施加手段的过程就是编解码。

> 不过，一般来说，编码是把对象逐步转换成字节序列的过程，解码是把字节序列逐步转换成对象的过程。其中，序列化和反序列化是特殊的编码和解码过程。在 Netty 中，我们并不严格区分序列化反序列化与编解码的界限，统一叫作编解码。
>
> 为什么说是逐步转换的过程呢？因为我们可以把多个编解码器串在一起实现最终的转换，比如，冰先转成水，再转成水气，相当于是两个编解码器。

另外，Netty 还帮我们实现了很多主流协议的编解码器，比如，http、http2、mqtt、redis、stomp 等等，可以说覆盖了我们能遇到的 99% 的场景，另外 1% 我们也可以基于优秀的 ChannelHandler 接口自定义编解码器来解决。

> `netty-codec` 与 `netty-handler` 是两个平齐的模块，并不互相依赖，没有包含和被包含的关系，ChannelHandler 接口位于 `netty-transport` 模块中，两者都依赖于 `netty-transport` 模块。

如果说 `netty-handler` 让我们爱上了 Netty，那么，`netty-codec` 可以说让我们爱死了 Netty。

netty-example

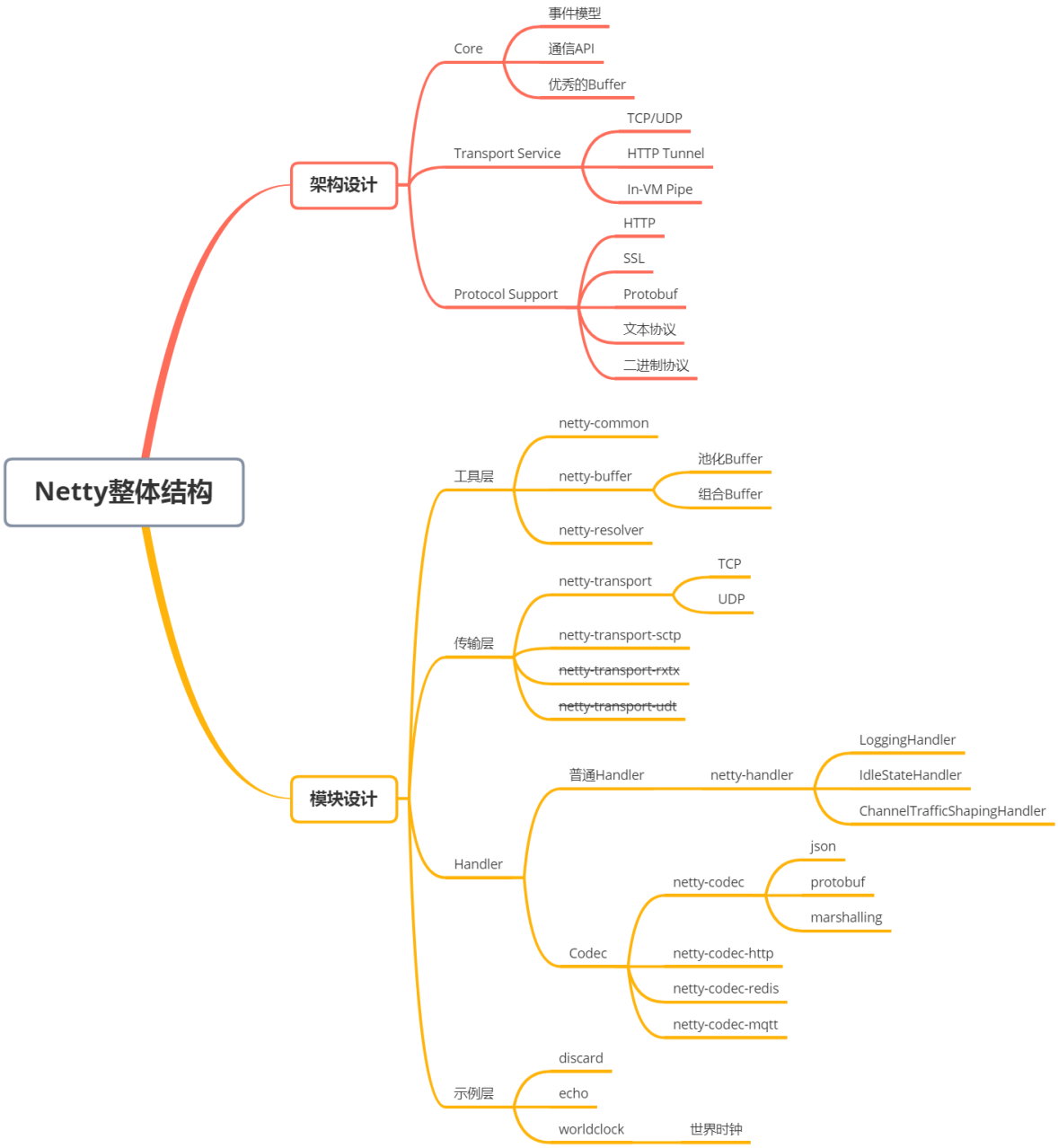
`netty-example` 包含了各种各样的案例，比如，我们经常拿来举例的 `echo` 和 `worldclock` 等。

如果说你已经爱死了 Netty，那么，`netty-example` 绝对会让你欲罢不能，因为，这里是专门提供给你抄代码的地方 ^^。

后记

本节，我们一起学习了 Netty 的架构设计以及模块设计，通过本文的介绍，不知道你有没有爱上 Netty 呢？

思维导图



}

05 Java NIO的核心组件有哪些

07 如何优雅地编写Netty应用程序