链滴

Netty 之 TCP 黏包与拆包

作者：xjtushilei

原文链接：https://ld246.com/article/1527735363985

来源网站：链滴

昨天在进行 netty NIO 学习时发现的，如果 client 连续不断的向 server 发送数据包时， server 接收的数据会出现两个数据包粘在一起的情况，这就是 TCP 协议中经常会遇到的粘包以及拆包的问 。
接下来讲的主要包括：

tcp 黏包拆包的概念
netty 中如何解决
顺带提一下类似 dubbo，http 之类的网络协议怎么设计

<h2 id="TCP黏包拆包">TCP 黏包拆包</h2>
TCP 的基础概念就不讲了，但是提一下几个对理解黏包拆包有关的概念。
<h3 id="1-长连接和短连接-">1. 长连接和短连接。</h3>

长连接

Client 方与 Server 方先建立通讯连接，连接建立后 不断开， 然后再进行报文发送和接收。

短连接

Client 方与 Server 每进行一次报文收发交易时才进行通讯连接，交易完毕后立即断开连接。此方式常用于一点对多点通讯，比如多个 Client 连接一个 Server.
<h3 id="2-面向消息和面向流">2. 面向消息和面向流</h3>
面向消息就是指存在保护消息边界，就是指传输协议把数据当作一条独立的消息在网上传输，接端只能接收独立的消息，接收端一次只能接收发送端发出的一个数据包。
而面向流则是指无保护消息保护边界的，如果发送端连续发送数据，接收端有可能在一次接收动中，会接收两个或者更多的数据包。
<h2 id="黏包拆包的表现形式">黏包拆包的表现形式</h2>
现在假设客户端向服务端连续发送了两个数据包，用 packet1 和 packet2 来表示，那么服务端到的数据可以分为三种，现列举如下：
第一种情况，接收端正常收到两个数据包，即没有发生拆包和粘包的现象，此种情况不在本文的论范围内。

第二种情况，接收端只收到一个数据包，由于 TCP 是不会出现丢包的，所以这一个数据包中包了发送端发送的两个数据包的信息，这种现象即为粘包。这种情况由于接收端不知道这两个数据包的限，所以对于接收端来说很难处理。

第三种情况，这种情况有两种表现形式，如下图。接收端收到了两个数据包，但是这两个数据包么是不完整的，要么就是多出来一块，这种情况即发生了拆包和粘包。这两种情况如果不加特殊处理对于接收端同样是不好处理的。

<h2 id="什么时候发生黏包拆包">什么时候发生黏包拆包</h2>

如果利用 tcp 每次发送数据，就与对方建立连接，然后双方发送完一段数据后，就关闭连接，这就不会出现粘包问题（因为只有一种包结构,类似于 http 协议）。就是 TCP 的短连接，长链接的话需解决黏包问题。
关闭连接主要是要双方都发送 close 连接（参考 tcp 关闭协议）。如：A 需要发送一段字符串给，那么 A 与 B 建立连接，然后发送双方都默认好的协议字符如”hello give me sth abour yourself

，然后 B 收到报文后，就将缓冲区数据接收，然后关闭连接，这样粘包问题不用考虑到，因为大家都道是发送一段字符。

如果双方建立连接，需要在连接后一段时间内发送不同结构数据，如连接后发送：
1)<code>"hello give me sth abour yourself"</code>
2)<code>"Don'tgive me sth abour yourself"</code>
那这样的话，发送方连续发送这个两个包出去，接收方一次接收可能会是：
<code>"hello give me sth abour yourselfDon't give me sth abour yourself"</code>
这样接收方就傻了，到底是要干嘛？不知道，因为协议没有规定这么诡异的字符串，所以要处理它分包，怎么分也需要双方组织一个比较好的包结构，所以一般可能会在头加一个数据长度之类的包以确保接收能够拆分。

总结原因：

发送和接受方不及时发送接收，等缓冲区满了才发送:

发送端需要等缓冲区满才发送出去，造成粘包.

接收方不及时接收缓冲区的包，造成多个包接收.

数据包大小问题

要发送的数据大于 TCP 发送缓冲区剩余空间大小，将会发生拆包成两个进行发送。

待发送数据大于 MSS（最大报文长度），TCP 在传输前将进行拆包发送。

要发送的数据小于 TCP 发送缓冲区的大小，TCP 将多次写入缓冲区的数据一次发送出去，将会生粘包。

<h2 id="如何解决黏包拆包">如何解决黏包拆包</h2>
主要有三种策略，其利弊也有简单说明：

（1）发送固定长度的消息
我们首先假设总长度 1000，如果我发送一个”hi”，长度不足 1000,其他地方补全即可。但是浪费资源，同时不能传输长度超过 1000 的报文。

（2）使用特殊标记来区分消息间隔
回车换行符就是一种特殊的结束分隔符

这种方法虽然解决了长度固定的问题，但是有分隔符被使用了怎么办的问题。当然了你的系统如不存在 <code>"#@!"</code> 三个字符转化为字节的这样的顺序，那么你就可以使用这些字节作分隔符。但是像微信就不可以用这种方式，因为用户可能什么都会发，万一发送了个跟 <code>"#@!</code> 字节一样的消息，就会发生混乱。

（3）把消息的尺寸与消息一块发送
这个就是最常规和正常的方式。dubbo 等协议都是基于这些的，因为 dubbo 也是基于 netty 实现底层通讯的，所以速度可能会比 spring 等速度好点。
举个简单的实现例子，我们把报文的前 4 个字节永远固定，就是一个长整数，用来声明这条报文总长度，后面的字节当成 body。每次一方接收到报文，取出前 4 个字节进行解码，得知总长度，然进行这个包的处理就可以了。
如果把前面的 4 个字节，扩展到 16 字节，那就是 dubbo 协议的长度了。

<h2 id="Netty中的解决方案">Netty 中的解决方案</h2>
netty 中的 decode 和 encode 已经有了很多接口，想个性化自己实现可以，也可以直接采用他提供的。

<code>FixedLengthFrameDecoder</code>。主要对应上一节中的策略 1，使用很简单。</li

<code>DelimiterBasedFrameDecoder</code>。主要对应上一节中的策略 2，可以自定义输自己的分隔符。
编写自己的协议。

编写自己的协议的话，主要就是约定好报头就好啦。具体可以参考 dubbo 的协议：

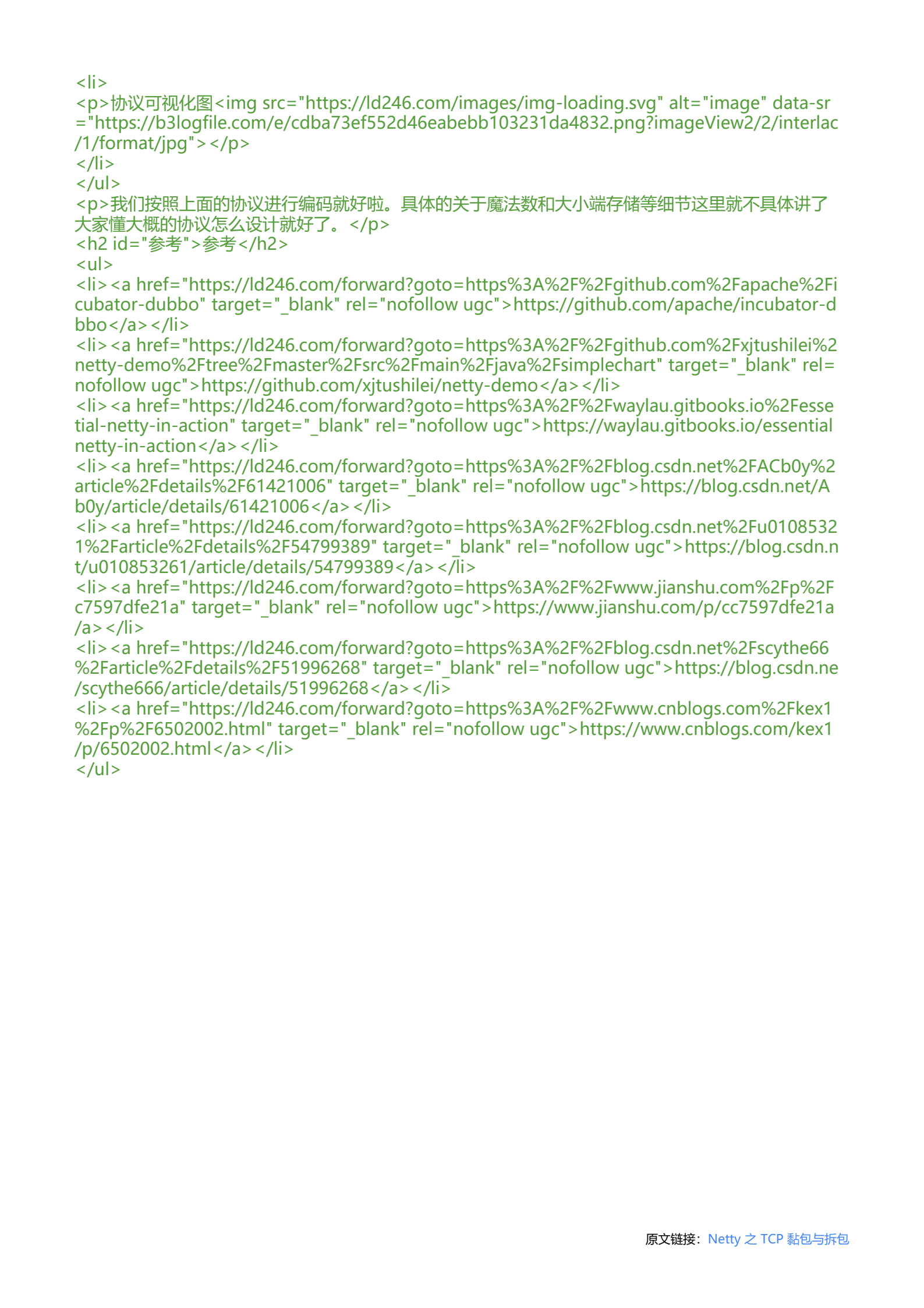
下面是 github 源码

ExchangeCodec
DubboCodec

<h2 id="如何自定义应用层网络协议">如何自定义应用层网络协议</h2>
简单的说，就是之前讲的，主要就是我们规定好我们协议的报头就好。下面举例一个很简单的协。

协议头
8 字节的定长协议头。支持版本号，基于魔数的快速校验，不同服务的复用。定长协议头使协议于解析且高效。

协议体
变长 json 作为协议体。json 使用明文文本编码，可读性强、易于扩展、前后兼容、通用的编解算法。json 协议体为协议提供了良好的扩展性和兼容性。

协议可视化图 

我们按照上面的协议进行编码就好啦。具体的关于魔法数和大小端存储等细节这里就不具体讲了大家懂大概的协议怎么设计就好了。
<h2 id="参考">参考</h2>

https://github.com/apache/incubator-dubbo
https://github.com/xjtushilei/netty-demo
https://waylau.gitbooks.io/essential netty-in-action
https://blog.csdn.net/Ab0y/article/details/61421006
https://blog.csdn.net/u010853261/article/details/54799389
https://www.jianshu.com/p/cc7597dfe21a
https://blog.csdn.net/scythe666/article/details/51996268
https://www.cnblogs.com/kex1/p/6502002.html
