TCP是个流协议，就是没有界限的一串数据。因此一个完整的包可能会被TCP拆分成多个包进行发送，也有可能把多个小的包封装成一个大的数据包发送，这就是所谓的TCP粘包和拆包问题。

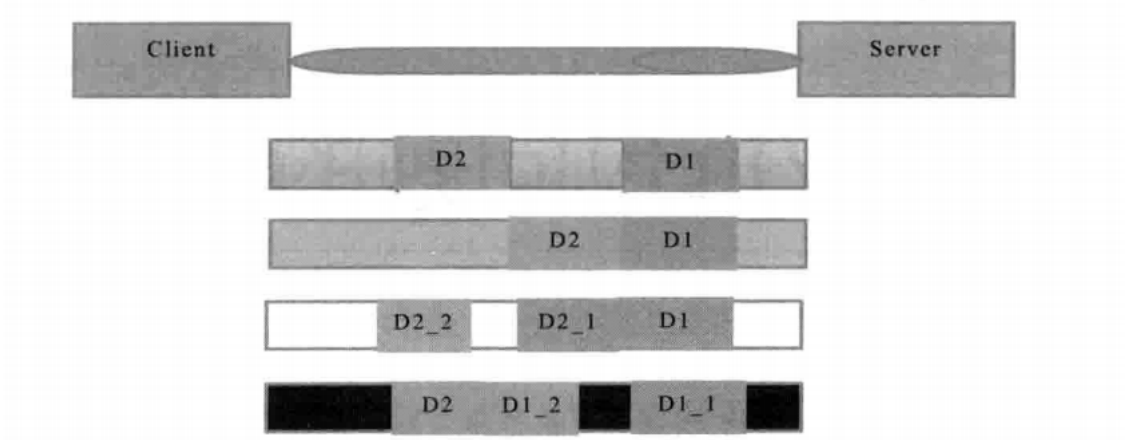

图 4-1　TCP 粘包/拆包问题

假设客户端分别发送了两个数据包D1和D2给服务端，由于服务端一次读取到的字节数是不确定的，故可能存在以下四种情况。

1.服务端分两次读取到了两个独立的数据包，分别是D1和D2，没有粘包和拆包。

2.服务端一次接收到了两个数据包，D1和D2粘合在一起，被称为TCP粘包。

3.服务端分两次读取到了两个数据包，第一次读取到了完整的D1包和D2包的部分内容，第二次读取到了D2包的剩余内容，被称为TCP粘包。

4.服务端分两次读取到了两个数据包，第一次读取到了D1包的一部分，第二次读取到了D1包的剩余内容和D2包的整包。

出现粘包和拆包的原因，tcp为了提高性能，发送端将需要发送的数据发送到缓冲区，等待缓冲区满了之后，再将缓冲区的数据发送到接收方。同理，接收方也有缓冲区这样的机制，来接收数据。

1.应用程序写入数据的字节大于套接字发送缓冲区的大小将发生拆包。

2.进行MSS大小的TCP分段，MSS是TCP报文段，当TCP报文长度-TCP头部长度>MSS的时候将发生拆包。

3.应用程序写入数据小于套接字缓冲区大小，网卡将应用多次写入的数据发送到网络上，将发生粘包。

解决策略，由于底层的TCP无法理解上层的业务数据，所以在底层是无法保证数据包不被拆分和充足的，这个问题只能通过上层的应用协议栈设计来解决，根据业界的主流协议的解决方案，可以归纳如下：

1.消息定长，例如每个报文的大小为固定长度200字节，如果不够，空位补空格。

2.在包尾增加回车换行符进行分割。

3.将消息分为消息头和消息体，消息头中包含表示消息总长度的字段。

Netty如何用半包解码器来解决TCP粘包/拆包问题。

LinBasedFrameDecoder+StringDecoder。

```
41.
42.       private class ChildChannelHandler extends ChannelIni
<SocketChannel> {
43. @Override
44. protected void initChannel(SocketChannel arg0) throws Except
45.     arg0.pipeline().addLast(new LineBasedFrameDecoder(1024));
46.     arg0.pipeline().addLast(new StringDecoder());
47.     arg0.pipeline().addLast(new TimeServerHandler());
48. }
49.     }
50.
51.     /**
```

4.3.4 LineBasedFrameDecoder 和 StringDecoder 的原理分析

LineBasedFrameDecoder 的工作原理是它依次遍历 ByteBuf 中的可读字节，否有“\n”或者“\r\n”，如果有，就以此位置为结束位置，从可读索引到结束位字节就组成了一行。它是以换行符为结束标志的解码器，支持携带结束符或者不符两种解码方式，同时支持配置单行的最大长度。如果连续读取到最大长度后仍现换行符，就会抛出异常，同时忽略掉之前读到的异常码流。

StringDecoder 的功能非常简单，就是将接收到的对象转换成字符串，然后继面的 handler。LineBasedFrameDecoder + StringDecoder 组合就是按行切换的文本它被设计用来支持 TCP 的粘包和拆包。

DelimiterBasedFrameDecoder解码器。本例程中以“$_”作为分隔符，然后把它加入到ChannelPipleline中，传递两个参数，第一个是1024表示单条信息的最大长度，第二个对象就是分割符缓冲对象。

```
@Override
public void initChannel(SocketChannel ch)
    throws Exception {
    ByteBuf delimiter = Unpooled.copiedBuffer("$_"
        .getBytes());
    ch.pipeline().addLast(
        new DelimiterBasedFrameDecoder(1024,
            delimiter));
    ch.pipeline().addLast(new StringDecoder());
    ch.pipeline().addLast(new EchoServerHandler());
}
});
```

2.FixedLengthFrameDecoder是固定长度解码器，它能够按照指定的长度对消息进行自动解码，开发者不需要考虑粘包和拆包问题。

5.2.1 FixedLengthFrameDecoder 服务端开发

在服务端的 ChannelPipeline 中新增 FixedLengthFrameDecoder，长度设置为 2
再依次增加字符串解码器和 EchoServerHandler，代码如下。

代码清单 5-5 EchoServer 服务端 EchoServer

```
20. public class EchoServer {
21.     public void bind(int port) throws Exception {
22.      // 配置服务端的 NIO 线程组
23.      EventLoopGroup bossGroup = new NioEventLoopGroup();
24.      EventLoopGroup workerGroup = new NioEventLoopGroup();
25.      try {
26.          ServerBootstrap b = new ServerBootstrap();
27.          b.group(bossGroup, workerGroup)
28.              .channel(NioServerSocketChannel.class)
29.              .option(ChannelOption.SO_BACKLOG, 100)
30.              .handler(new LoggingHandler(LogLevel.INFO))
31.              .childHandler(new ChannelInitializer<SocketChannel>
32.              @Override
33.              public void initChannel(SocketChannel ch)
34.                  throws Exception {
35.                  ch.pipeline().addLast(
36.                      new FixedLengthFrameDecoder(20));
37.                  ch.pipeline().addLast(new StringDecoder());
38.                  ch.pipeline().addLast(new EchoServerHandler());
39.              }
```

UDP是整包发、整包收，因此不会发生粘包的现象。UDP协议是无连接的，面向消息的，提供高效率服务。它为每个消息打包一个消息头，这样接收端就可以分辨出消息的边界。即面向消息的通信是有消息保护边界的。