dubbo源码解析——拆包和粘包

关于拆包和粘包

关于拆包和粘包，其实是出现在TCP链接发送数据时的现象，这篇文章不做详解，主要是来看dubbo如何处理这种现象（dubbo并没有使用原生的netty中的Decoder），而是基于自己的协议栈实现了自己的`ENCODER`/`DECODER`，

如果有兴趣可以查看这里Netty解决拆包粘包。

dubbo拆/粘包入口

入口是Dubbo中的`NettyCodecAdapter`，这里包含了`encoder`和`decoder`的相关定义，我们看下拆包的相关部分，分段来看，先看第一部分：

```
com.alibaba.dubbo.remoting.buffer.ChannelBuffer message;
if (buffer.readable()) {
    if (buffer instanceof DynamicChannelBuffer) {
        buffer.writeBytes(input.toByteBuffer());
        message = buffer;
    } else {
        int size = buffer.readableBytes() + input.readableBytes();
        message = com.alibaba.dubbo.remoting.buffer.ChannelBuffers
                size > bufferSize ? size : bufferSize);
        message.writeBytes(buffer, buffer.readableBytes());
        message.writeBytes(input.toByteBuffer());
    }
} else {
    message = com.alibaba.dubbo.remoting.buffer.ChannelBuffers.wra
            input.toByteBuffer());
}
```

先判断buffer是否有可读的数据，如果有则把消息进行合并，得到message再

进行解析。

继续看一下解析部分：

```
do {
    saveReaderIndex = message.readerIndex();
    try {
        msg = codec.decode(channel, message);
    } catch (IOException e) {
        buffer = com.alibaba.dubbo.remoting.buffer.ChannelBuffers.
        throw e;
    }
    if (msg == Codec2.DecodeResult.NEED_MORE_INPUT) {
        message.readerIndex(saveReaderIndex);
        break;
    } else {
        if (saveReaderIndex == message.readerIndex()) {
            buffer = com.alibaba.dubbo.remoting.buffer.ChannelBuff
            throw new IOException("Decode without read data.");
        }
        if (msg != null) {
            Channels.fireMessageReceived(ctx, msg, event.getRemote
        }
    }
} while (message.readable());
```

可以看到这是一个循环，一直处理message直到消息不可读。这里可以看到，先是储存了reader索引，用以回滚，然后进入`decoder`，这里是`DubboCountCodec`，这里注意：`DubboCountCodec`每次只会解析出一个完整的dubbo协议栈（或者是`NEED_MORE_INPUT`）：

```
int save = buffer.readerIndex();
MultiMessage result = MultiMessage.create();
do {
    Object obj = codec.decode(channel, buffer);
    if (Codec2.DecodeResult.NEED_MORE_INPUT == obj) {
        buffer.readerIndex(save);
```

```
            buffer.readerIndex(save);
            break;
        } else {
            result.addMessage(obj);
            logMessageLength(obj, buffer.readerIndex() - save);
            save = buffer.readerIndex();
        }
    } while (true);
    if (result.isEmpty()) {
        return Codec2.DecodeResult.NEED_MORE_INPUT;
    }
    if (result.size() == 1) {
        return result.get(0);
    }
    return result;
```

这里暂存了当前`buffer`的读索引，同样也是为了后面的回滚（这里回滚的不是外层中的`buffer`，这里要区分）。可以看到当`decode`返回的是`NEED_MORE_INPUT`则表示当前的`buffer`中数据不足，不能完整解析出一个dubbo协议栈，同时将`buffer`的读索引回滚到之前暂存的索引并且退出循环，将结果返回。

这里继续看一下`codec.decode`，最终定位到`ExchangeCodec`的`decode`中，还是分段来看，最开始我们忽略，这是dubbo对`telnet`的支持：

```
    // check length.
    if (readable < HEADER_LENGTH) {
        return DecodeResult.NEED_MORE_INPUT;
    }

    // get data length.
    int len = Bytes.bytes2int(header, 12);
    checkPayload(channel, len);

    int tt = len + HEADER_LENGTH;
    if (readable < tt) {
        return DecodeResult.NEED_MORE_INPUT;
    }
```

这里有两种情况：
1. 可读长度小于协议头的长度，说明：可读字段都不够协议头的数据量，返回`DecodeResult.NEED_MORE_INPUT`。
2. 当前buffer包含了完整的消息头，得到payload的长度，发现它的可读的长度，并没有包含整个协议栈的数据，返回`DecodeResult.NEED_MORE_INPUT`。
3. 否则说明至少包含一个协议栈，那么解析出一个dubbo数据，当然可能读取完之后还有剩余数据。

我们看回到第二段代码，msg如果是`NEED_MORE_INPUT`，则把`readerIndex`回滚。循环结束：

```
if (message.readable()) {
    message.discardReadBytes();
    buffer = message;
} else {
    buffer = com.alibaba.dubbo.remoting.buffer.ChannelBuffers.EMPT
}
NettyChannel.removeChannelIfDisconnected(ctx.getChannel());
```

如果可读，丢弃读过的字节，然后把message剩下的信息缓存在buffer中，供下次读取时使用，至此我们就知道了，为什么dubbo需要实现自己的拆包粘包：

`Dubbo需要解析自己独有的协议栈，而默认的方式不符合dubbo的解析方式。`