

[博客专区](#) > [无毁的湖光-AI的博客](#) > [博客详情](#)

原荐 解Bug之路-TCP粘包Bug

无毁的湖光-AI 发表于 7天前 阅读 2894 收藏 143 点赞 13 评论 87

已收藏



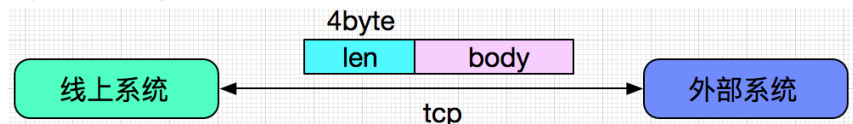
解Bug之路-TCP粘包Bug

笔者很热衷于解决Bug,同时比较擅长(网络/协议)部分,所以经常被唤去解决一些网络IO方面的Bug。现在就挑一个案例出来,写出分析思路,以飨读者,希望读者在以后的工作中能够少踩点坑。

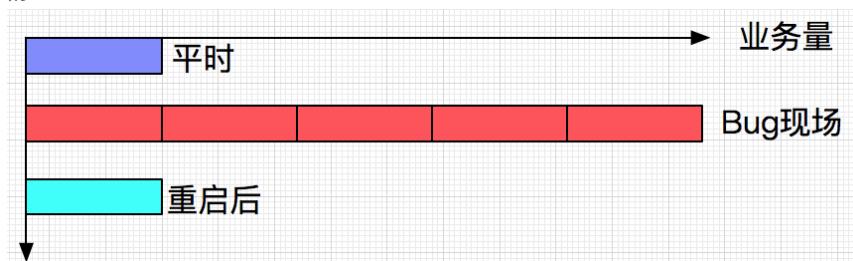
TCP粘包Bug

Bug现场

出Bug的系统是做与外部系统进行对接之用。这两者并不通过http协议进行交互，而是在通过TCP协议之上封装一层自己的报文进行通讯。如下图所示:



通过监控还发现，此系统的业务量出现了不正常的飙升，大概有4倍的增长。而且在监控看来，这些业务还是成功的。



第一反应，当然是祭出重启大法，第一时间重启了机器。此后一切正常，交易量也回归正常，仿佛刚才的Bug从来没有发生过。在此之前，此系统已经稳定运行了好几个月，从来没出现过错误。

但是，这事不能就这么过去了，下次又出这种Bug怎么办，继续重启么？由于笔者对分析这种网络协议比较在行，于是Bug就抛到了笔者这。

错误日志

线上系统用的框架为Mina,不停的Dump出其一堆以16进制表示的二进制字节流。

[illegible]

首先定位异常抛出点

以下代码仅为笔者描述Bug之用，和当时代码有较大差别。

```
private boolean handleMessage( IoBuffer in, ProtocolDecoderOutput out ){
    int lenDes = 4;
    byte[] data = new byte[ lenDes ];
    in.mark();
    in.get( data, 0, lenDes );
    int messageLen = decodeLength( data );
    if( in.remaining() < messageLen ){
        logger.warn( "未接收完毕" );
        in.reset();
        return false;
    } else {
        *****
    }
}
```

解Bug之路-TC

TCP粘包Bug

Bug现场

错误日志

首先定位异常

为何会抛出异常

为何报文会计

演绎

为何流量会飙升

完结了么？

NO,整个演绎

最后的高潮

丢失的两字

Bug的源头:

正确代码

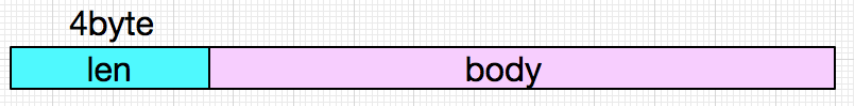
为什么线上一重

总结

原文链接

}

笔者本身经常写这种拆包代码，第一眼就发现问题。让我们再看一眼报文结构：



上面的代码首先从报文前4个字节中获取到报文长度，同时检测在buffer中的存留数据是否够报文长度。

```
if(in.remaining() < messageLen)
```

为何没有在一开始检测buffer中是否有足够的4byte字节呢。此处有蹊跷。直觉上就觉的是这导致了后来的种种现象。

事实上，在笔者解决各种Bug的过程中，经常通过猜想等手段定位出Bug的原因。但是从现场取证，通过证据去解释发生的现象，通过演绎去说服同事，并对同事提出的种种问题做出合理的解释才是最困难的。

猜想总归是猜想，必须要有实锤，没有证据也说服不了自己。

为何会抛出异常

这个异常由这句代码抛出：

```
int messageLen = decodeLength(data);
```

从上面的Mina框架Dump出的数据来看，是解析前四个字节出了问题，前4个字节为30,31,2E,01(16进制) 最前面的包长度是通过字符串来表示的，翻译成十进制就是48、49、46、1，再翻译为字符串就是{'0','1', 非数字, 非数字)

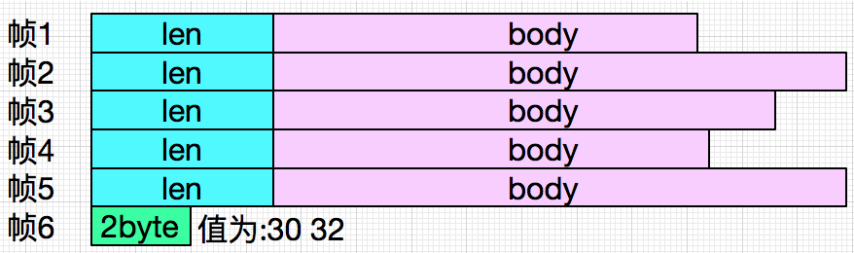
```
30, 31,    2E,    01  (16进制)
48, 49,    46,    1  (10进制)
'0', '1', 非数字, 非数字 (字符串)
```

很明显，解析字符串的时候遇到前两个byte.0和1可以解析出来，但是遇到后面两个byte就报错了。至于为什么是For input String.'01',而不是2E,是由于传输用的是小端序。

为何报文会出现非数字的字符串

鉴于上面的错误代码，笔者立马意识到，应该是粘包了。这时候就应该去找发生Bug的最初时间点的日志，去分析为何那个时间会粘包。

由于最初那个错误日志Dump出来的数据过于长，在此就不贴出来了，以下示意图是笔者当时人肉decode的结果：



抛出的异常为：

```
Caused by: java.nio.BufferUnderflowException: null
    at java.nio.HeapByteBuffer.get(HeapByteBuffer.java:127) ~[na:1.6.0_45]
    at org.apache.mina.core.buffer.AbstractIoBuffer.get(AbstractIoBuffer.java:615) ~[mina-core-2.0.7.jar:na]
```

这个异常抛出点恰恰就在笔者怀疑的

```
in.get(data,0,lenDes);
```

这里。至此，笔者就几乎已经确定是这个Bug导致的。

演绎

Mina框架在Buffer中解帧，前5帧正常。但是到第六帧的时候，只有两个字节，无法组成报文的4byte长度头,而代码没有针对此种情况做处理,于是报错。为何会出现这种情况：

TCP窗口的大小取决于当前的网络状况、对端的缓冲大小等等因素，TCP将这些都从底层屏蔽。开发者无法从应用层获取这些信息。这就意味着，当你在接收TCP数据流的时候无法知道当前接收了多少数据流，数据可能在任意一个比特位（seq）上。这就是所谓的“粘包”问题。
详情见笔者另一篇博客<https://my.oschina.net/alchemystar/blog/833937>

第六帧的头两个字节是30,32正好和后面dump出来的30 31 2e 01中的30、31组成报文长度

30,32,30,31（16进制）
48,50,48,49（10进制）
0, 2, 0, 1（字符串）
2, 0, 1, 0（整理成大端序）

这四个字节组合起来才是正常的报文头，再经过运算得到整个Body的长度。
第一次Mina解析的时候，后面的两个30,31尚未放到buffer中，于是出错：

```
public ByteBuffer get(byte[] dst, int offset, int length) {
    checkBounds(offset, length, dst.length);
    // 此处抛出异常
    if (length > remaining())
        throw new BufferUnderflowException();
    int end = offset + length;
    for (int i = offset; i < end; i++)
        dst[i] = get();
    return this;
}
```

为何流量会飙升

解释这个问题前，我们先看一段Mina源码：

```
// if there is any data left that cannot be decoded, we store
// it in a buffer in the session and next time this decoder is
// invoked the session buffer gets appended to
if (buf.hasRemaining()) {
    if (usingSessionBuffer && buf.isAutoExpand()) {
        buf.compact();
    } else {
        storeRemainingInSession(buf, session);
    }
} else {
    if (usingSessionBuffer) {
        removeSessionBuffer(session);
    }
}
```

Mina框架为了解决粘包问题，会将这种尚未接收完全的包放到sessionBuffer里面，待解析完毕后把这份Buffer删除。
如果代码正确，对报文头做了校验，那么前5个报文的buffer将经由这几句代码删除，只留下最后两个没有被decode的两字节。

```
if (usingSessionBuffer && buf.isAutoExpand()) {
    buf.compact();
} else {
    storeRemainingInSession(buf, session);
}
```

但是，由于decode的时候抛出了异常，没有走到这段逻辑，所以前5个包还留在sessionBuffer中，下一次解包的时候，又会把这5个包给解析出来，发送给后面的系统。如下图所示：



这也很好的解释了为什么业务量激增，因为系统不停的发相同的5帧给后面系统，导致监控认为业务量飙升。后查询另一个系统的日志，发现一直同样的5个序列号坐实了这个猜想。

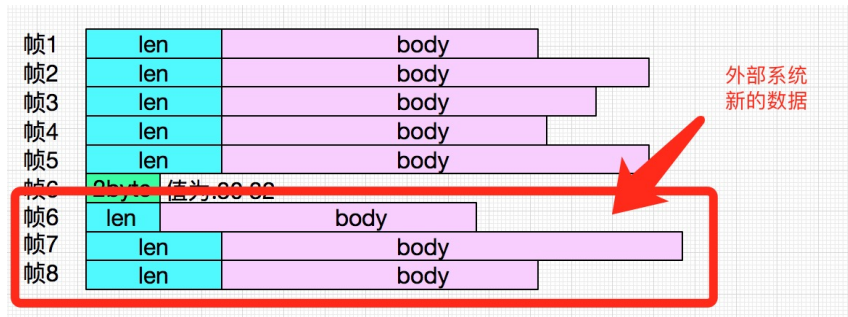
完结了么？

NO,整个演绎还有第二段日志的推演

就是系统后来不停dump出的日志，再贴一次：

[illegible]

这个buffer应该是Mina继续接收外部系统的数据到buffer中导致，

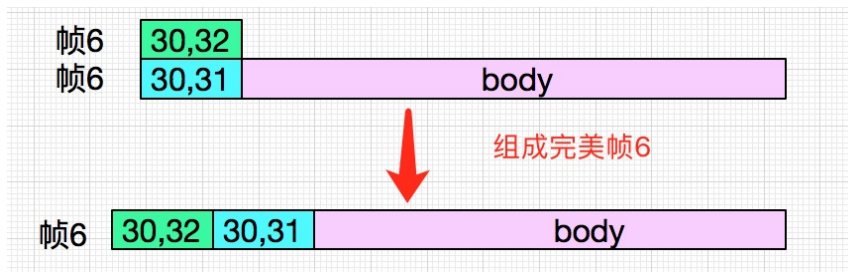


Mina框架不停的接收数据，直到buffer区满，然后整个框架不停的解析出前5帧，到第6帧的时候，出错，然后dump出其尚未被解帧的数据。这就是第二段日志。

最后的高潮

到现在推理似乎很完美了，但是我突然觉得不对(另一位同事也提出了相同的疑问):

如果说Mina接收到新的数据放到buffer中的话，第6帧的前两个字节和后来发过来的若干字节不是又拼成了完整的一帧了么，那么后来为什么会一直出错了呢。如下图所示：



丢失的两字节

按照前面的推理，帧6的前两个字节30、32肯定是丢了，那么怎么丢的呢？推理又陷入了困境，怎么办？日志已经帮不了笔者了，毕竟日志的表现都已解释清楚。翻源码吧：

Bug的源头:

如果有问题，肯定出在将数据放在Buffer中的环节，于是笔者找到了这段代码：

```

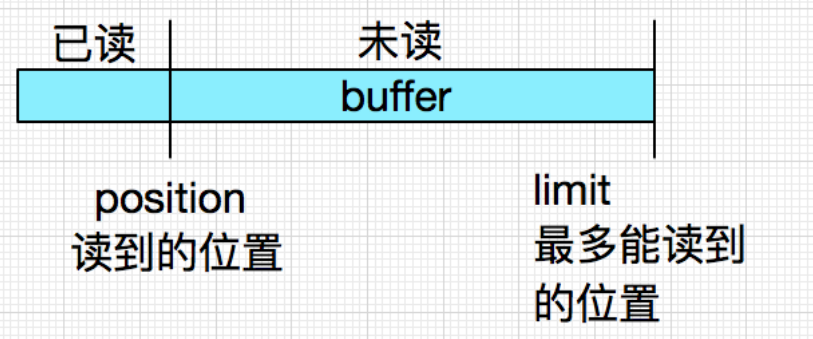
if (appended) {
    buf.flip();
} else {
    // Reallocate the buffer if append operation failed due to
    // derivation or disabled auto-expansion.
    buf.flip();
    *****
}

```

问题出在buf.flip()上面，这段代码最后调用的代码是Java的Nio的Buffer的flip,代码如下:

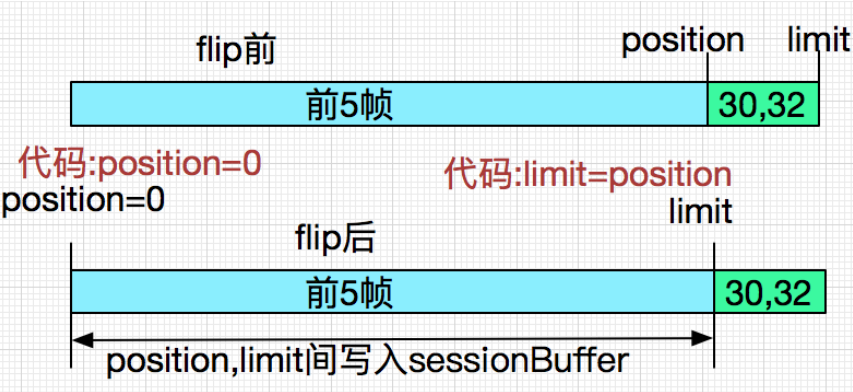
```
public final Buffer flip() {
    // 下面这一句导致了最终的Bug现象
    limit = position;
    position = 0;
    mark = -1;
    return this;
}
```

为什么呢？首先我们需要了解一下Nio Buffer的一些特点：

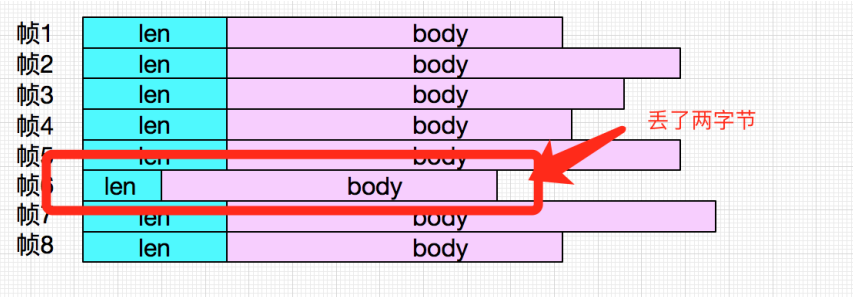


同时当Mina框架将数据(数据本身也是一个buffer)放到sessionBuffer的时候,也是将position到limit的数据放到新buffer中,

下面我们演绎一下第一次抛异常时候的flip前和flip后：



这样就清楚了,在buf.flip()后,由于limit变成了原position的位置，这样最后的两个字节30,32就被无情的丢弃了。这样整个sessionBuffer就变成：



为什么position在flip前没有指向limit的位置，是由于在每次读取前有一个checkBound的动作，在检查buffer数据不够后，不会推进position的位置，直接抛出异常：

```
static void checkBounds(int off, int len, int size) { // package-private
    if ((off | len | (off + len) | (size - (off + len))) < 0)
        throw new IndexOutOfBoundsException();
}
```

这样所有的都说的通了，也完美解释了所有的现象。

正确代码

```
private boolean handleMessage(ioBuffer in,ProtocolDecoderOutput out){
    int lenDes = 4;
    byte[] data = new byte[lenDes];
    in.mark();
    // 前4字节校验代码
    if(in.remaining() < lenDes){
        // 由于未消费字节，无需reset
        return false;
    }
    in.get(data,0,lenDes);
    int messageLen = decodeLength(data);
    if(in.remaining() < messageLen){
        logger.warn("未接收完毕");
        in.reset();
        return false;
    }else{
        .....
    }
}
```

```
}  
  
}
```

为什么线上一直稳定

随着网络不断发展的今天，一些短小的帧很难出现中间断开的粘包现象。而在一个好几百字节的包中，前4个字节正好出错的概率那更是微乎其微。这样就导致Bug难复现，很难抓住。即使猜到是这里，也没有足够的证据来证明。

总结

Mina/Netty等各种网络框架给我们解决粘包问题提供了非常好的解决方案。但是我们写代码的时候也不能掉以轻心，必须时刻以当前可能读不够字节的心态去读取buffer中的数据，不然就可能遭重。
在此感谢给力的各位同事们，是你们的各种反驳让我能够找到最终的源头，也让我对网络框架有了更加深刻的理解。

原文链接

<https://my.oschina.net/alchemystar/blog/880659>

© 著作权归作者所有
分类： 工作日志 字数： 2610
标签： Java MINA Netty TCPIP

打赏

点赞

已收藏

分享



无毁的湖光-AI

后端工程师 浦东

+ 关注 粉丝 73 | 博文 15 | 码字总数 14785

相关博客



netty vs mina

 zhanghua.1199 3375 9



NIO框架入门(三): iOS与MINA2、Netty4的跨平台UDP双向通信实战

 JackJiang- 108 0




【原创】NIO框架入门(四): Android与MINA2、Netty4的跨平台UDP双向通信实战

 JackJiang- 147 0

评论 (87)

Ctrl+Enter 发表评论



周胜

1楼 2017/04/17 19:33

楼主技术大拿，小弟佩服佩服



2楼 2017/04/17 19:34

引用来自“周胜”的评论

楼主技术大拿，小弟佩服佩服

@周胜：)



小斯

3楼 2017/04/17 20:06

佩服~~曾遇到类似bug，不过没怎么复杂



无毁的湖光-AI

4楼 2017/04/17 20:08

引用来自“小斯”的评论

佩服~~曾遇到类似bug，不过没怎么复杂

同事各种给力的反驳和质疑，让我把这个Bug查到了底。



小斯

5楼 2017/04/17 20:13

引用来自“无毁的湖光-AI”的评论

引用来自“小斯”的评论

佩服~~曾遇到类似bug，不过没怎么复杂

同事各种给力的反驳和质疑，让我把这个Bug查到了底。

回复@无毁的湖光-AI：不过我用了mina做了一个服务器接收tcp数据的，也没见过会粘包。这是因为你们流量比较原因吗？



无毁的湖光-AI

6楼 2017/04/17 20:14

引用来自“小斯”的评论

引用来自“无毁的湖光-AI”的评论

引用来自“小斯”的评论

佩服~~曾遇到类似bug，不过没怎么复杂

同事各种给力的反驳和质疑，让我把这个Bug查到了底。

回复@无毁的湖光-AI：不过我用了mina做了一个服务器接收tcp数据的，也没见过会粘包。这是因为你们流量比较原因吗？

随着网络不断发展的今天，一些短小的帧很难出现中间断开的粘包现象。而在一个好几百字节的包中，前4个字节正好出错的概率更是微乎其微。这样就导致Bug难复现，很难抓住 文中有写哦，如果是局域网的话，那概率更小了^_^



无毁的湖光-AI

7楼 2017/04/17 20:22

引用来自“小斯”的评论

引用来自“无毁的湖光-AI”的评论

引用来自“小斯”的评论

佩服~~曾遇到类似bug，不过没怎么复杂

同事各种给力的反驳和质疑，让我把这个Bug查到了底。

回复@无毁的湖光-AI：不过我用了mina做了一个服务器接收tcp数据的，也没见过会粘包。这是因为你们流量比较原因吗？

在服务器负载高的时候也会出现这种情况，如果你要模拟的话，可以在发送数据的时候，每隔几十个字节sleep一秒，模拟这种粘包Bug



styleman

8楼 2017/04/17 20:36

搞tcp必遇到这个坑，知道并解决即可。



无毁的湖光-AI

9楼 2017/04/17 20:39

引用来自“styleman”的评论

搞tcp必遇到这个坑，知道并解决即可。

https://my.oschina.net/alchemystar/blog/880659?nocache=1492957811374

7/9

代码上的错误一眼就发现了，难的是找到证据说服同事们 发生的所有现象都是由这个Bug引起的，没有其他更多的Bug



sunrise程序员

10楼 2017/04/18 08:36

这种BUG只会在大并发压测时才会发现到。



无毁的湖光-AI

11楼 2017/04/18 08:52

引用来自“sunrise程序员”的评论
这种BUG只会在大并发压测时才会发现到。

@sunrise程序员 嗯 线上遇到的概率很小 所以稳定运行了几个月才出现一次 但一出现 整个系统就不能接收新的请求了



talent-tan

12楼 2017/04/18 09:02

引用来自“sunrise程序员”的评论
这种BUG只会在大并发压测时才会发现到。

网络不好且消息体大时，容易出现



talent-tan

13楼 2017/04/18 09:02

引用来自“styleman”的评论
搞tcp必遇到这个坑，知道并解决即可。

一开始就做好包的分界点处理，就不会有这个坑。



无毁的湖光-AI

14楼 2017/04/18 09:07

引用来自“talent-tan”的评论
引用来自“sunrise程序员”的评论
这种BUG只会在大并发压测时才会发现到。

网络不好且消息体大时，容易出现

@talent-tan 出现的概率 取决于 收发双方的负载，双方socket的buffer剩余，网络的拥塞程度，包体的大小以及整个链路中路由器的最小MTU等等不一而足。所以统一抽象成粘包的概念。



szf

15楼 2017/04/18 09:26

写得挺好的。

不过我代表少数派，认为在tcp协议解析中使用“包”概念的思路出发点就有问题。

正确的方法是把它看成“流”，再考虑算法，一切都顺其自然了，包括理解mina/netty这些框架。



amita

16楼 2017/04/18 09:26

所以结论是，Mina框架有bug?

可以报bug了



无毁的湖光-AI

17楼 2017/04/18 09:29

引用来自“szf”的评论
写得挺好的。
不过我代表少数派，认为在tcp协议解析中使用“包”概念的思路出发点就有问题。
正确的方法是把它看成“流”，再考虑算法，一切都顺其自然了，包括理解mina/netty这些框架。

@szf 嗯 我一直主张流这个概念 可见我上一篇博客Tcp滑动窗口 不过貌似oschina把这篇博客的很多图给弄没了：) 其实我搞过一段协议栈 tcp流这个概念下的ip层也是由ip的包概念组成的 不过应用层看起来是流就够了



无毁的湖光-AI

18楼 2017/04/18 09:30

引用来自“amita”的评论

所以结论是，Mina框架有bug?
可以报bug了

@amita 是代码没有严格按照分包的标准方法处理 不算mina的bug



FPE
19楼 2017/04/18 09:36
大概看了下，感觉很明显是这个什么框架本身的bug。sessionBuffer难道他们没设计成环形缓冲区吗？再说，从环形缓冲区中读取数据时，不应该需要用户处理offset是否达到len。感觉纯粹是这框架的设计失败。



无毁的湖光-AI
20楼 2017/04/18 09:40
引用来自“FPE”的评论
大概看了下，感觉很明显是这个什么框架本身的bug。sessionBuffer难道他们没设计成环形缓冲区吗？再说，从环形缓冲区中读取数据时，不应该需要用户处理offset是否达到len。感觉纯粹是这框架的设计失败。

@FPE 个人感觉 框架本身不应该对错误的代码进行容错

社区

开源项目
技术问答
动弹
博客

众包

开源资讯
技术翻译
专题
招聘

码云

项目大厅
软件与服务
接活赚钱

活动

Git代码托管
Team
PaaS
在线工具

关注微信公众号

下载手机客户端

©开源中国(OSChina.NET)

关于我们 广告联系 @新浪微博 合作单位

开源中国社区是工信部 开源软件推进联盟 指定的官方社区 粤ICP备12009483号-3