

HttpClient偶尔报NoHttpResponseException: xxx failed to respond 问题分析

漫步无法人生 [关注](#)

 0.454 2019.11.26 09:25:03 字数 1,730 阅读 863

HttpClient偶尔报NoHttpResponseException: xxx failed to respond

背景描述

调用底层服务偶尔会报以下错误

```
1 org.apache.http.NoHttpResponseException: submit.10690221.com:9012 failed to respond
2
3     at org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpResponseParser.java:141)
4     ....
```

第一次碰到，先google一下，发现不少相同的情况，讲的也很不错，但是呢，我想自己复现一下，并且自己去分析并解决，这样能更好的去理解 网络 这东西

复现方法

这个怎么复现呢，通过google得知，这个只会在服务器端keep-alive刚好过期的时间我们进行访问才能大概率复现，方法如下：


wireshark进行抓包得出底层服务器的keep-alive时间

写一段程序，用于探测底层服务器的keep-alive，代码如下：

```
1 @Test
2 public void test121() throws Exception {
3     String url = "http://xxxxxxx:9012/hy/json";
4     CloseableHttpClient httpClient = HttpClients.createDefault();
5     HttpPost request = new HttpPost(url);
6
7     httpClient.execute(request, response -> {
8         String content = EntityUtils.toString(response.getEntity());
9         System.out.println(content);
10        return content;
11    });
12
13    Thread.sleep(1000000);
14 }
15 }
```

开启wireshark进行抓包，执行程序直到下图出现即可停止



漫步无法人生 [关注](#)

总资产1 (约0.15元)

6个重点帮你提升技术水平
阅读 65

全脑快速阅读训练21天
阅读 1

如何让大脑得到高效休息-正念
阅读 64

推荐阅读

三面字节跳动被虐得“体无完肤”，15天读完这份pdf，终拿下美团研发岗...
阅读 114,520

对于二本渣渣来说，面试阿里P6也太难了！（两年crud经验，已拿offer）
阅读 66,055

离开菜鸟&新的面试体验
阅读 13,204

Spring Cloud Stream 进阶配置——使用延迟队列实现“定时关闭超时未...
阅读 3,182

SpringCloud Gateway-过滤器执行逻辑链分析
阅读 104



秒，也就是当一个连接socket 65秒内没有数据交互，底层服务器就会认为这个连接可以关闭了，因此才会在3分36秒进行挥手操作发送一个FIN包,这时我们稍微改造一下这个程序，如下：

```
1  @Test
2  public void test121() throws Exception {
3      String url = "http://xxxxxxx:9012/hy/json";
4      CloseableHttpClient httpClient = HttpClients.createDefault();
5      HttpPost request = new HttpPost(url);
6      while (true) { //加了一个死循环 ^_^
7          httpClient.execute(request, response -> {
8              String content = EntityUtils.toString(response.getEntity());
9              System.out.println(content);
10             return content;
11         });
12
13         Thread.sleep(65000); //关键在这里，设置和底层服务器keep-alive相同
14     }
15 }
```

相比第一个，有两个改动

- 1. 加了一个循环
- 2. 每次调用的间隔改成和底层服务器相同的65秒

我们清空wireshark，运行该程序抓包，结果如下：

Time	Source	Destination	Protocol	Length	Info
26	2019-11-26 07:22:42.530009	192.168.0.102	47.102.208.224	TCP	78 59233 → 9012 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=0
27	2019-11-26 07:22:42.557935	47.102.208.224	192.168.0.102	TCP	74 9012 → 59233 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
28	2019-11-26 07:22:42.558062	192.168.0.102	47.102.208.224	TCP	66 59233 → 9012 [ACK] Seq=1 Ack=1 Win=131584 Len=0 TSval=131392
29	2019-11-26 07:22:42.577787	192.168.0.102	47.102.208.224	HTTP	252 POST /hy/json HTTP/1.1
30	2019-11-26 07:22:42.606241	47.102.208.224	192.168.0.102	TCP	66 9012 → 59233 [ACK] Seq=1 Ack=187 Win=30080 Len=0 TSval=131392
31	2019-11-26 07:22:42.607456	47.102.208.224	192.168.0.102	HTTP	253 HTTP/1.1 200 (application/json)
32	2019-11-26 07:22:42.607519	192.168.0.102	47.102.208.224	TCP	66 59233 → 9012 [ACK] Seq=187 Ack=188 Win=131392 Len=0 TSval=131392
2358	2019-11-26 07:23:47.609808	47.102.208.224	192.168.0.102	TCP	66 9012 → 59233 [FIN, ACK] Seq=188 Ack=187 Win=30080 Len=0 TSval=131392
2359	2019-11-26 07:23:47.609873	192.168.0.102	47.102.208.224	TCP	66 59233 → 9012 [ACK] Seq=187 Ack=189 Win=131392 Len=0 TSval=131392
2360	2019-11-26 07:23:47.654480	192.168.0.102	47.102.208.224	TCP	66 59233 → 9012 [FIN, ACK] Seq=187 Ack=189 Win=131392 Len=0 TSval=131392
2364	2019-11-26 07:23:47.683086	47.102.208.224	192.168.0.102	TCP	54 9012 → 59233 [RST] Seq=189 Win=0 Len=0
2366	2019-11-26 07:23:47.705827	192.168.0.102	47.102.208.224	TCP	78 59429 → 9012 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=0
2367	2019-11-26 07:23:47.732983	47.102.208.224	192.168.0.102	TCP	74 9012 → 59429 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
2368	2019-11-26 07:23:47.733075	192.168.0.102	47.102.208.224	TCP	66 59429 → 9012 [ACK] Seq=1 Ack=1 Win=131584 Len=0 TSval=131392
2369	2019-11-26 07:23:47.738206	192.168.0.102	47.102.208.224	HTTP	252 POST /hy/json HTTP/1.1
2370	2019-11-26 07:23:47.765364	47.102.208.224	192.168.0.102	TCP	66 9012 → 59429 [ACK] Seq=1 Ack=187 Win=30080 Len=0 TSval=131392
2371	2019-11-26 07:23:47.765791	47.102.208.224	192.168.0.102	HTTP	253 HTTP/1.1 200 (application/json)
2372	2019-11-26 07:23:47.765833	192.168.0.102	47.102.208.224	TCP	66 59429 → 9012 [ACK] Seq=187 Ack=188 Win=131392 Len=0 TSval=131392
7576	2019-11-26 07:24:52.768422	47.102.208.224	192.168.0.102	TCP	66 9012 → 59429 [FIN, ACK] Seq=188 Ack=187 Win=30080 Len=0 TSval=131392
7577	2019-11-26 07:24:52.768542	192.168.0.102	47.102.208.224	TCP	66 59429 → 9012 [ACK] Seq=187 Ack=189 Win=131392 Len=0 TSval=131392
7578	2019-11-26 07:24:52.781424	192.168.0.102	47.102.208.224	TCP	66 59429 → 9012 [FIN, ACK] Seq=187 Ack=189 Win=131392 Len=0 TSval=131392
7582	2019-11-26 07:24:52.808248	47.102.208.224	192.168.0.102	TCP	54 9012 → 59429 [RST] Seq=189 Win=0 Len=0
7584	2019-11-26 07:24:52.832269	192.168.0.102	47.102.208.224	TCP	78 59705 → 9012 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=0
7585	2019-11-26 07:24:52.862679	47.102.208.224	192.168.0.102	TCP	74 9012 → 59705 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
7586	2019-11-26 07:24:52.862798	192.168.0.102	47.102.208.224	TCP	66 59705 → 9012 [ACK] Seq=1 Ack=1 Win=131584 Len=0 TSval=131392
7598	2019-11-26 07:24:52.871941	192.168.0.102	47.102.208.224	HTTP	252 POST /hy/json HTTP/1.1
7612	2019-11-26 07:24:52.901135	47.102.208.224	192.168.0.102	TCP	66 9012 → 59705 [ACK] Seq=1 Ack=187 Win=30080 Len=0 TSval=131392
7613	2019-11-26 07:24:52.901425	47.102.208.224	192.168.0.102	HTTP	253 HTTP/1.1 200 (application/json)
7614	2019-11-26 07:24:52.901457	192.168.0.102	47.102.208.224	TCP	66 59705 → 9012 [ACK] Seq=187 Ack=188 Win=131392 Len=0 TSval=131392
9900	2019-11-26 07:25:57.923660	192.168.0.102	47.102.208.224	HTTP	252 POST /hy/json HTTP/1.1
9901	2019-11-26 07:25:57.925363	47.102.208.224	192.168.0.102	TCP	66 9012 → 59705 [FIN, ACK] Seq=188 Ack=187 Win=30080 Len=0 TSval=131392
9902	2019-11-26 07:25:57.925412	192.168.0.102	47.102.208.224	TCP	66 59705 → 9012 [ACK] Seq=373 Ack=189 Win=131392 Len=0 TSval=131392
9903	2019-11-26 07:25:57.926573	192.168.0.102	47.102.208.224	TCP	66 59705 → 9012 [FIN, ACK] Seq=373 Ack=189 Win=131392 Len=0 TSval=131392
9904	2019-11-26 07:25:57.933115	47.102.208.224	192.168.0.102	TCP	54 9012 → 59705 [RST] Seq=189 Win=0 Len=0
9905	2019-11-26 07:25:57.933179	47.102.208.224	192.168.0.102	TCP	54 9012 → 59705 [RST] Seq=189 Win=0 Len=0
9906	2019-11-26 07:25:57.955436	47.102.208.224	192.168.0.102	TCP	54 9012 → 59705 [RST] Seq=189 Win=0 Len=0

问题分析

首先我们分析一下抓包结果

31	2019-11-26 07:22:42.607456	47.102.208.224	192.168.0.102	HTTP	253	HTTP/1.1 200 (application/json)	
32	2019-11-26 07:22:42.607519	192.168.0.102	47.102.208.224	TCP	66	59233 → 9012 [ACK] Seq=187 Ack=188 Win=131392 Len=0 TSval=	
2358	2019-11-26 07:23:47.609808	47.102.208.224	192.168.0.102	TCP	66	9012 → 59233 [FIN, ACK] Seq=188 Ack=187 Win=0 Len=0 TSval=	
2359	2019-11-26 07:23:47.609873	192.168.0.102	47.102.208.224	TCP	66	59233 → 9012 [ACK] Seq=187 Ack=189 Win=131392 Len=0 TSval=	

1. 红色框1：前3个请求是建立连接的过程，三次握手，接着4个请求就是client和server的数据交互，着重看最后四个请求
1. 9012 -> 59233 [FIN, ACK]：服务器主动进行关闭，给client发送了FIN包
2. 59233 -> 9012 [ACK]：client进行回应ACK包
3. 69233 -> 9012 [FIN, ACK]：按照四次挥手原则，client发现目前数据已经发送完毕了，因此也发出FIN包
4. 9012 -> 59233 [RST]：服务器直接返回一个RST
2. 红色框2：同2
3. 红色框3：前面的7个步骤都是相同的，建立连接，数据交互，区别唯独在于绿色框
1. 9012 -> 59233 POST /hy/json: client认为服务器端可用，因此给服务器发送数据
2. 9012 -> 59233 [FIN, ACK]：服务器认为此连接已经失效，因为超过了65s的keep-alive时间，主动进行关闭，给client发送了FIN包
3. 59233 -> 9012 [ACK]：client进行回应ACK包
4. 69233 -> 9012 [FIN, ACK]：按照四次挥手原则，client发现目前数据已经发送完毕了，因此也发出FIN包
5. 9012 -> 59233 [RST]：服务器直接返回一个RST 通过Seq=188，可判断这条是给【9012 -> 59233 POST /hy/json】这个请求回的
6. 9012 -> 59233 [RST]：服务器直接返回一个RST 通过Seq=189，可判断这条是给【69233 -> 9012 [FIN, ACK]】回的
7. 9012 -> 59233 [RST]：服务器直接返回一个RST 通过Seq=189，同6

通过分析抓包数据，得出结果是，当client客户端认为这条Socket连接有用，这时服务器端却认为该Socket连接无用，并主动关闭，就会报错,属于临界值没有处理好的

这时有人就说了，为什么前两次就没有问题呢，原因是HttpClient会进行连接过期是否可用的检查，那么也就能理解这是httpclient的一个bug，即使httpclient有做这么一件事情，但是由于网络I/O原因，导致httpclient认为一个关闭了的连接是有效的，才报了这个错误

接下来我们看看HttpClient为什么会复用一个已经被关闭的连接

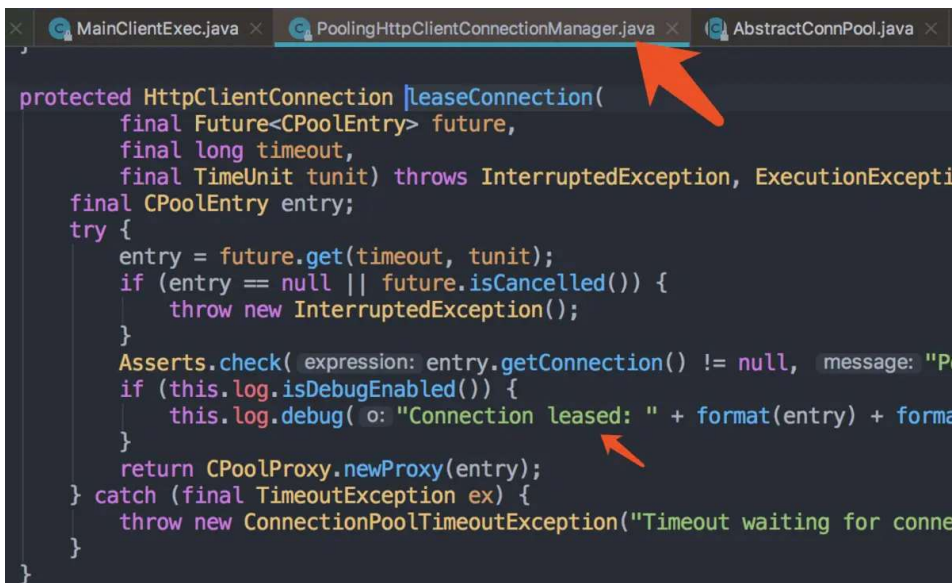
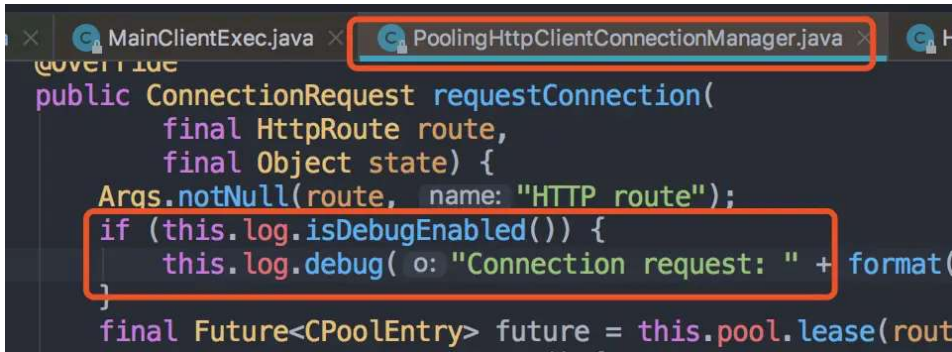
由于HttpClient代码有点多，为了方便快速定位缩小范围，我这边开启了debug，并对两者的日志进行了分析

左边日志是正常交互、右边是报错了

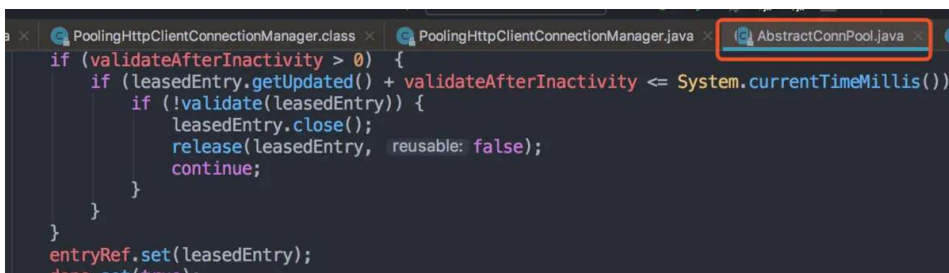
CookieSpec selected: default Auth cache not set in the context Connection request: [route: {}->http://submit.10690221.com:9012][total kept alive: 1; route allocated: 1 of 2; total allocated: 1 of 20] http-outgoing-1 << "end of stream" http-outgoing-1: Close connection Connection leased: [id: 2][route: {}->http://submit.10690221.com:9012][total kept alive: 0; route allocated: 1 of 2; total allocated: 1 of 20] Opening connection [{}->http://submit.10690221.com:9012] Connecting to submit.10690221.com/47.102.208.224:9012 Connection established 192.168.0.102:59705<->47.102.208.224:9012 Executing request POST /hy/json HTTP/1.1 Target auth state: UNCHALLENGED Proxy auth state: UNCHALLENGED http-outgoing-2 >> POST /hy/json HTTP/1.1 http-outgoing-2 >> Content-Length: 0 http-outgoing-2 >> Host: submit.10690221.com:9012 http-outgoing-2 >> Connection: Keep-Alive http-outgoing-2 >> User-Agent: Apache-HttpClient/4.5.6 (Java/1.8.0_201) http-outgoing-2 >> Accept-Encoding: gzip,deflate http-outgoing-2 >> "POST /hy/json HTTP/1.1[\r]\n" http-outgoing-2 >> "Content-Length: 0[\r]\n" http-outgoing-2 >> "Host: submit.10690221.com:9012[\r]\n" http-outgoing-2 >> "Connection: Keep-Alive[\r]\n" http-outgoing-2 >> "User-Agent: Apache-HttpClient/4.5.6 (Java/1.8.0_201)[\r]\n" http-outgoing-2 >> "Accept-Encoding: gzip,deflate[\r]\n" http-outgoing-2 >> "[\r]\n" http-outgoing-2 << "HTTP/1.1 200 [\r]\n" http-outgoing-2 << "Server: nginx/1.4.2[\r]\n" http-outgoing-2 << "Date: Mon, 25 Nov 2019 23:24:54 GMT[\r]\n" http-outgoing-2 << "Content-Type: application/json; charset=UTF-8[\r]\n" http-outgoing-2 << "Transfer-Encoding: chunked[\r]\n"	1 CookieSpec selected: default 2 Auth cache not set in the context 3 Connection request: [route: {}->http://submit.10690221.com:9012][total kept alive: 1; route allocated: 1 of 2; total allocated: 1 of 20] 4 http-outgoing-2 << "[read] I/O error: Read timed out" 5 Connection leased: [id: 2][route: {}->http://submit.10690221.com:9012][total kept alive: 0; route allocated: 1 of 2; total allocated: 1 of 20] 6 http-outgoing-2: set socket timeout to 0 7 Executing request POST /hy/json HTTP/1.1 8 Target auth state: UNCHALLENGED 9 Proxy auth state: UNCHALLENGED 10 http-outgoing-2 >> POST /hy/json HTTP/1.1 11 http-outgoing-2 >> Content-Length: 0 12 http-outgoing-2 >> Host: submit.10690221.com:9012 13 http-outgoing-2 >> Connection: Keep-Alive 14 http-outgoing-2 >> User-Agent: Apache-HttpClient/4.5.6 (Java/1.8.0_201) 15 http-outgoing-2 >> Accept-Encoding: gzip,deflate 16 http-outgoing-2 >> "POST /hy/json HTTP/1.1[\r]\n" 17 http-outgoing-2 >> "Content-Length: 0[\r]\n" 18 http-outgoing-2 >> "Host: submit.10690221.com:9012[\r]\n" 19 http-outgoing-2 >> "Connection: Keep-Alive[\r]\n" 20 http-outgoing-2 >> "User-Agent: Apache-HttpClient/4.5.6 (Java/1.8.0_201)[\r]\n" 21 http-outgoing-2 >> "Accept-Encoding: gzip,deflate[\r]\n" 22 http-outgoing-2 >> "[\r]\n" 23 http-outgoing-2 >> "end of stream" 24 http-outgoing-2: Close connection 25 http-outgoing-2: Shutdown connection 26 Connection discarded 27 Connection released: [id: 2][route: {}->http://submit.10690221.com:9012][total kept alive: 0; route allocated: 0 of 2; total allocated: 0 of 20] 28 org.apache.http.NoHttpResponseException: submit.10690221.com:9012 failed
--	--

打印了一串 "[read] I/O error: Read timed out" 后没有进行连接的重新建立，因此就报错了

那么可以通过打印 "[read] I/O error: Read timed out" 日志的上下文日志缩小 排查代码的范围，上文日志 Connection request，下文日志 Connection leased，进行代码定位



基本上定位到了PoolingHttpClientConnectionManager.java这个类，那么进行代码跟踪吧



追踪到了 AbstractConnPool.java类，那么这段代码什么意思呢，这个就是进行连接是否能够复用的检查代码

对validateAfterInactivity进行判断，这个是服务器keep-alive的值

1. leasedEntry.getUpdated() + validateAfterInactivity <= System.currentTimeMillis(): 如果连接的最后一次使用时间 + 服务器keep-alive的时间 小于等于当前时间，那么就认为该连接可能已经失效了

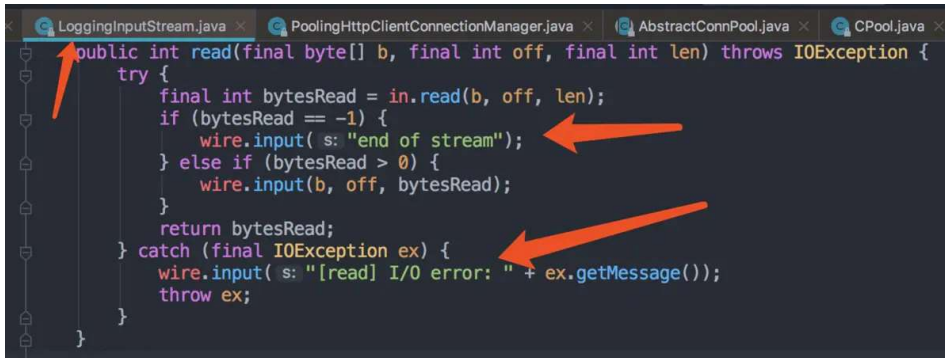
2. validate(leasedEntry): 用此会进行连接是否不生效的检查

写下你的评论...

评论2

赞5

...



最终找到"end of stream" and "[read] I/O error: Read timed out" 打印的地方
然后回到如下图代码:



可以看到

- 当bytesRead 值为 -1 时, 返回true,那么HttpClient就会认为该连接失效了, 不能够复用, 并进行清理操作,
- 当抛出异常是SocketTimeoutException时会返回false, 那么HttpClient就会认为该连接可复用

分析到这, 相信大部分人都已经知道为什么会报错了, 不过还是强烈建议自己动手分析一下, 另外大家可去了解一下, 为什么会输出"end of stream" and "[read] I/O error: Read timed out"两种不同的结果, 快去畅游底层Socket编程相关的原理吧, 这有助于你更加理解

解决方案

其实当你知道原因后, 也能想出对应的解决方案, 不过我这边还是收集列出来了一些

1. 禁用HttpClient的连接复用 (有点扯淡)
2. 重试方案:http请求使用重发机制, 捕获NoHttpResponseException的异常, 重新发送请求, 重发3次后还是失败才停止
3. 根据keep Alive时间, 调整validateAfterInactivity小于keepAlive Time,但这种方法依旧不能避免同时关闭
4. 系统主动检查每个连接的空闲时间, 并提前自动关闭连接, 避免服务端主动断开

性若枯中香过古安

写下你的评论...

评论2

赞5

...

"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



漫步无法人生 今天做了别人不愿意做的事情，明天就能做别人不能做的事情。小...
总资产1 (约0.15元) 共写了3.1W字 获得27个赞 共25个粉丝

关注

香港服务器

免备案 · 低延时 · 免费换IP · CN2带宽 · 高速回国

广告

写下你的评论...

全部评论 2

只看作者

按时间倒序

按时间正序

nicklause

2楼 2019.11.26 10:21

分析得非常细致，非常有帮助

赞 回复

漫步无法人生 作者

2019.11.26 11:20

感谢您的反馈

回复

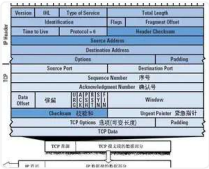
添加新评论

推荐阅读

更多精彩内容

在深谈TCP/IP三步握手&四步挥手原理及衍生问题—长文解剖IP
如果对网络工程基础不牢，建议通读《细说OSI七层协议模型及OSI参考模型中的数据封装过程？》下面就是TCP/IP...

zhoulujun 阅读 1,527 评论 0 赞 9



TCP连接的状态详解以及故障排查

1、TCP状态linux查看tcp的状态命令：1）、netstat -nat 查看TCP各个状态的数量2）、Iso...

北辰青 阅读 4,678 评论 0 赞 9

面试题

OSI（开放系统互联参考模型）标准模型 物理层负责为数据端设备透明地传输原始比特流，并且定义了数据终

写下你的评论...

评论2

赞5

...

《适合》

》合者适之 适者合也 之所以有臭味相投 也就有英雄相惜 是相互吸引 错过了 也许不可能再有 有的是默默地相守 高...



是小寒 阅读 63 评论 0 赞 4



《曲品》评汤显祖

汤奉常绝代奇才，冠世博学，周旋狂社，坎坷宦途。当阳之谪初还，彭泽之腰乍折。情痴一种，固属天生；才思万端，似挟灵气。...



呵手围炉 阅读 154 评论 0 赞 0