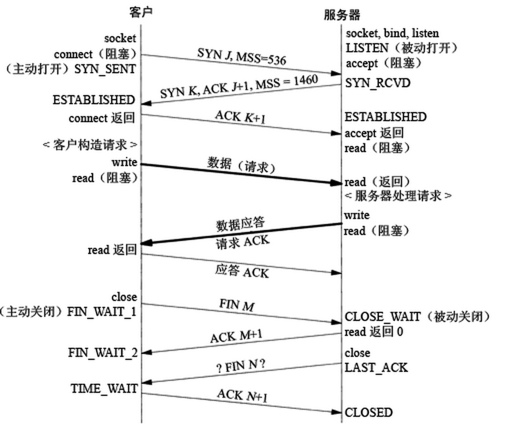
从tcp原理角度理解Broken pipe和Connection Reset by Peer的区别

  以前我们经常会碰到Broken pipe或者Connection reset by peer之类的异常，但是tcp实现里什么情况下会抛出这些异常呢，以前我给对方的回答都是模棱两可的，自己说实话都没把握，因为自己也没有验证过，对它们的认识都是从网上看来的，正确与否也不知道，昨天独明突然又问到这个问题，前段时间正好对tcp这块研究了一段时间，有了点理论知识之后再从实践角度对此问题进行一下分析，下面对我这次的调研过程进行下描述与大家分享，希望大家以后对此类问题都能很自信地应答。

**三次握手和四次挥手过程**

  在讲具体的原因之前，我们有必要补充下tcp这块的一些基础知识，我们都知道tcp通信有三次握手和四次挥手，网上介绍的文章也一大堆，图我也懒得画了，直接网上找一个图给大家

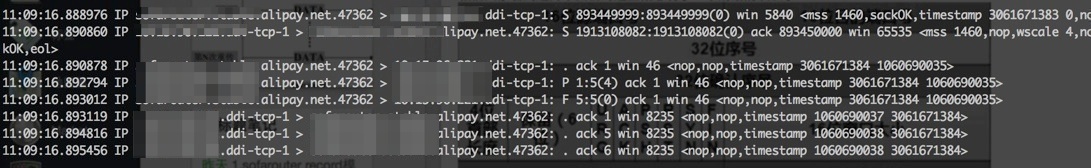


  三次握手是最前面的三条线表示的过程，四次挥手是最后面的四条线表示的过程，里面涉及到几个关键词，SYN，ACK，FIN，MSS，其中SYN是主要用在三次握手过程中的，FIN用在四次挥手过程中，ACK在三次握手和四次挥手过程中的作用就是对收到的SYN和FIN做一个确认，SYN，FIN等存在于TCP头里(tcp报文图也给大家弄了个图，不用再去找啦)，0/1表示有无此标记，在tcp实现里后面还会跟一个依次递增的数字，比如上面的J，K等，确认就是递增这些数字(真正的数据报文的ack除外)，MSS是表示每一个tcp报文里数据字段的最大长度，不包括tcp头的大小噢 相信大家看到这两个图会对这些概念有了一个清晰的认识了



**tcpdump抓包工具**

  介绍了基础原理之后，再介绍下抓包工具，tcpdump，这工具对你了解tcp的整个过程会非常有帮助，在你无法调试tcp实现的情况下这个工具自然也是必不可少的，具体用法网上有很多介绍，直接从man page上也可以看到详细的介绍，我也不多说啦，下面的截图就是tcpdump根据tcp通信过程获取到的



  这要稍微提下tcpdump的结果和上面的几个过程的对应关系 前面三条其实就是我们上面所说的三次握手，四次握手过程上面没有完全表现出来，只完成了一半的挥手过程（5，8两条表示的） 里面有几个标识S，F，ack，P，其实还有个R，如果有这些标识那么在tcp头里的SYN，FIN，ACK，PSH，RET分别为1，其中PSH表示要求tcp立即将数据传递给上层，不要做别的什么处理，RET这个表示重置连接，也是和我们今天讨论的问题有很大关系的FLAG，下面会详细介绍

**reset报文发送场景**

  RST的标志位，这个标识为在如下几种情况下会被设置，以下是我了解的情况，可能还有更多的场景，没有验证

* 当尝试和未开放的服务器端口建立tcp连接时，服务器tcp将会直接向客户端发送reset报文
* 双方之前已经正常建立了通信通道，也可能进行过了交互，当某一方在交互的过程中发生了异常，如崩溃等，异常的一方会向对端发送reset报文，通知对方将连接关闭
* 当收到TCP报文，但是发现该报文不是已建立的TCP连接列表可处理的，则其直接向对端发送reset报文
* ack报文丢失，并且超出一定的重传次数或时间后，会主动向对端发送reset报文释放该TCP连接

**Broken pipe以及Connection reset by peer**

  做了这么些铺垫之后下面进入正题，那么Broken pipe或者Connection reset by peer分别代表什么意思呢，下面从glibc的源码里有对此的介绍

#. TRANS Broken pipe; there is no process reading from the other end of a pipe.

#. TRANS Every library function that returns this error code also generates a

#. TRANS @code{SIGPIPE} signal; this signal terminates the program if not handled

#. TRANS or blocked. Thus, your program will never actually see @code{EPIPE}

#. TRANS unless it has handled or blocked @code{SIGPIPE}.

#: sysdeps/generic/siglist.h:39 sysdeps/gnu/errlist.c:359

#: sysdeps/unix/siglist.c:39

msgid "Broken pipe"

msgstr "断开的管道"

#. TRANS A network connection was closed for reasons outside the control of the

#. TRANS local host, such as by the remote machine rebooting or an unrecoverable

#. TRANS protocol violation.

#: sysdeps/gnu/errlist.c:614

msgid "Connection reset by peer"

msgstr ""

  其实我们java异常里看到的Broken pipe或者Connection reset by peer信息不是jdk或者jvm里定义的，我看到这些关键字往往会首先搜索下jdk或者hotspot源码找到位置进行上下文分析，但是这次没找到，后面才想到应该是linux或者glibc里定义的，果然在glibc离看到了如上的描述和定义

  对于Broken pipe在管道的另外一端没有进程再读的时候就会抛出此异常，Connection reset by peer的描述其实不是很正确，从我的实践来看只描述了一方面，其实在某一端正常close之后，也是可能会有此异常的。

**异常模拟**

  从我的测试场景是这样的， 共同的前提是客户端向服务端发了数据之后立马调用close关闭socket并进程退出，而服务端在收到客户端的数据之后sleep一会，保证对方的socket已经关闭，接着分别进行两种场景测试

  场景：

1. 服务端往socket里写一次数据，返回继续做select
2. 服务端连续写两次数据，必须保证两次的buffer都是有数据的，也就是保证ByteBuffer的pos和limit要不是一个值

  结果：

1. 会抛出Connection reset by peer   
   
2. 会抛出Broken pipe  
   

  分析：

1. 当我们往一个对端已经close的通道写数据的时候，对方的tcp会收到这个报文，并且反馈一个reset报文，tcpdump的结果如下所示,当收到reset报文的时候，继续做select读数据的时候就会抛出Connect reset by peer的异常，从堆栈可以看得出 http://lovestblog.cn/images/2014/07/tcpdump_2.jpg
2. 当第一次往一个对端已经close的通道写数据的时候会和上面的情况一样，会收到reset报文，当再次往这个socket写数据的时候，就会抛出Broken pipe了 ，根据tcp的约定，当收到reset包的时候，上层必须要做出处理，调用将socket文件描述符进行关闭，其实也意味着pipe会关闭，因此会抛出这个顾名思义的异常