java.io.IOException Broken pipe solution ClientAbortException: java.io.IOException: Broken pipe

Today, the company's technical support reported that a customer's service did not work, and they urgently asked for help, so I remotely logged on to the server to check the problem.

Check the tomcat log of the collected data, and habitually turn to the end of the log to see if there is any exception printing. several kinds of exception information are found, but this is the most one

```
24-Nov-2016 09:54:21.116 SEVERE [http-nio-8081-Acceptor-0] org.apache.tomcat.util.net.NioEndp
 java.io.IOException: Too many open files
        at sun.nio.ch.ServerSocketChannelImpl.accept0(Native Method)
        at sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:241)
        at org.apache.tomcat.util.net.NioEndpoint$Acceptor.run(NioEndpoint.java:688)
        at java.lang.Thread.run(Thread.java:745)
```

The problem of "manage open files" is very common. If the file descriptor exceeds the limit, the file cannot be opened or the network connection cannot be created. This problem will lead to some other problems. It must be that ulimit is not optimized, so check the ulimit setting;

```
[root@sdfassd logs]# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 62819
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 65535
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 10240
cpu time               (seconds, -t) unlimited
max user processes              (-u) 62819
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

open files is 65535. It has been optimized. Is it tomcat and other services start first, and then optimize ulimit? It's possible, if werestart the service, it's ok. Then I restart all the services and run normally. After a while, the report will display the data. Then I tell the technical support that the problem has been solved, and then I go to deal with other cases;

As a result, within 20 minutes, the technical support said that there was no data in the report again, so I checked the tomcat log of the data collection application and found a bunch of exceptions, all of which were this error

```
24-Nov-2016 09:54:24.574 WARNING [http-nio-18088-exec-699] org.apache.catalina.core.StandardH
org.apache.catalina.connector.ClientAbortException: java.io.IOException: Broken pipe
        at org.apache.catalina.connector.OutputBuffer.realWriteBytes(OutputBuffer.java:393)
        at org.apache.tomcat.util.buf.ByteChunk.flushBuffer(ByteChunk.java:426)
        at org.apache.catalina.connector.OutputBuffer.doFlush(OutputBuffer.java:342)
        at org.apache.catalina.connector.OutputBuffer.close(OutputBuffer.java:295)
        at org.apache.catalina.connector.Response.finishResponse(Response.java:453)
        at org.apache.catalina.core.StandardHostValve.throwable(StandardHostValve.java:378)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:174)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:79)
        at org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.ja
        at org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.ja
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:88)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:537)
        at org.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.j
        at org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProto
        at org.apache.coyote.http11.Http11NioProtocol$Http11ConnectionHandler.process(Http11N
        at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1556
        at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.run(NioEndpoint.java:1513)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
        at java.lang.Thread.run(Thread.java:745)
```

There are a lot of exceptions. Read the error message. It is the Broken pipe exception of tomcat's connector when it performs the write operation. The connector is used by tomcat to process the network request. Is there something wrong with the network? But why is it that the exception is all written and the read is OK? In order to determine whether it is a network problem, I use wget command to access an interface of the server locally. The result shows that we have waited for a long time but have not responded. Normally, we should have responded immediately. This indicates that it is not the network problem, but the server problem. Then I use the command to check the current tcpip connection status

```
[root@sdfassd logs]# netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key
CLOSE_WAIT        3853
TIME_WAIT         40
ESTABLISHED       285
LAST_ACT          6
```

There are 3853 CLOSE_WAIT state connections. This is abnormal. It means that the client has closed the connection first, but the server has not closed the connection. As a result, the server is always CLOSE_WAIT state, If we do not optimize the keepalive of the operating system, this state will last for two hours by default. Check the settings of the following system:

```
[root@sdfassd logs]# sysctl -a |grep keepalive
net.ipv4.tcp_keepalive_time = 7200
net.ipv4.tcp_keepalive_probes = 9
net.ipv4.tcp_keepalive_intvl = 75
```

Sure enough, it's 7200 seconds, which explains why the last error of checking tomcat log for the first time is "Too manay open files". It must be within two hours, CLOSE_WAIT state increases sharply, the file descriptor exceeds the maximum limit of 65535;

This state should be caused by the broken pipe exception. What causes the broken pipe exception? Why does the probe close the connection, but the data acquisition server does not? The exception is the connector of tomcat. tomcat can't forget to call the close method to close the connection, which eliminates the problem of the program and can't figure out what caused it;

So I went to check the log of collection server, and there was a large number of exceptions:

```
2016-11-24 16:27:36,217 [TingYun Harvest Service 1] 166 WARN  - Error occurred sending metric
java.net.SocketTimeoutException: Read timed out
        at java.net.SocketInputStream.socketRead0(Native Method) ~[na:1.7.0_60]
        at java.net.SocketInputStream.read(SocketInputStream.java:152) ~[na:1.7.0_60]
        at java.net.SocketInputStream.read(SocketInputStream.java:122) ~[na:1.7.0_60]
        at com.tingyun.agent.libs.org.apache.http.impl.io.SessionInputBufferImpl.streamRead(S
        .................
```

If all of them are read time out exceptions, then the problem is clear. It is that the probe side reads overtime and disconnects. At this time, the data acquisition server is still processing the request. It does not know that the probe side has disconnected. After processing the request, it sends the processing result to the probe, and then it breaks the pipe;

The original exception is that the client has closed the connection due to reading timeout. At this time, when the server writes data to the disconnected connection of the client, the broken pipe exception occurs!

The probe read timeout time is 2 minutes. Why did the server not respond for such a long time? So we use jstack command to export the thread stack information of tomcat for analysis, and finally find that there are time-consuming operations in the code are locked, leading to thread blocking (for confidentiality reasons, we don't paste the code here);

To sum up, some of my friends who sent me private messages didn't get the key point of the broken pile problem. It's not only the timeout that causes this problem. As long as the connection is disconnected and the write operation is performed on the disconnected connection, this exception will appear. The client's overtime disconnection is just one of the cases

In addition, when you see the "Too manay open files" exception, in addition to checking the ulimit of system you should also check the number of file handles opened by the process. The **cat /proc/sys/fs/file-nr** command is used to check the total number of system handles, The number of file handles opened by the current application uses the **ls -l /proc/<pid>/fd | wc -l** command. Fortunately, this step is ignored here, otherwise it may take some time to find the real problems of the system;

From this case, we can see that in some cases, the exception information you see at the first sight may not be the root of the problem, but a chain reaction in the follow-up. Especially when a large number of the same exception occurs, instead of looking at the last exception log, you should first go to the log to find the location where the first exception occurs and see the reason why the exception occurs;

java tcp/ip exception

1 java.net.SocketTimeoutException .
This exception is quite common, socket timeout. Generally, there are two places to throw this. One is when connecting, the timeout parameter is determined by second parameter in connect(SocketAddress endpoint,int timeout), and the other is setSoTimeout(int timeout), which is to set the reading timeout. They are set to 0 to indicate infinity.

2 java.net.BindException:Address already in use: JVM_Bind
The exception occurs when a new ServerSocket(port) or socket.bind(SocketAddress bindpoint) operation.

Reason: the port has been started and monitored. At this point, with netstat –an command, you can see the port in the listening state. Just find a port that is not occupied to solve this problem.

3 java.net.ConnectException: Connection refused: connect

CNCF

The exception occurs when the client performs new Socket(ip, port) or socket.connect(address,timeout) operation, reason: the machine with the specified ip address cannot be found (that is, the ip route from the current machine does not exist to the specified ip route), or the ip exists, but the specified port cannot be found for listening. You should first check whether the ip and port of the client are wrongly written. If it is correct, ping the server from the client to see if it can be pinged. If it can be pinged (another method is needed if the server disables ping), check whether the program monitoring the specified port on the server is started.

4 java.net.SocketException: Socket is closed
This exception can occur on both the client and the server. The reason for the exception is that the network connection is read and written after the connection is closed (the close method of Socket is called).

5 java.net.SocketException : connection reset or Connect reset by peer:Socket write error
Connection reset by peer will appear when calling write or read. According to glibc, such as by the remote machine rebooting or an unrecoverable protocol violation. Literally, it means that the remote machine restarts or an unrecoverable error occurs. According to my tests, only the opposite end directly kills the process. What's the difference between the two? Compared with tcpdump's packet interception diagram, if the remote process is directly killed, the remote end does not send the FIN serial number to tell the other party that I have closed the pipeline, but directly sends the RST serial number. If the remote end calls close or shutdown, it will send the FIN serial number. According to the four waves of TCP, FIN is needed. I guess that if I receive the RST sequence number directly instead of receiving the FIN sequence number of the other party at the local end, it indicates that machine rebooting or an unrecoverable protocol violation at the opposite end. At this time, the IO operation of this pipeline will result in a connection reset by peer error.

This exception may occur on both the client and the server. There are two reasons for this exception. The first is that if the Socket at one end is closed (or actively closed or closed due to an exception exit), the other end still sends data, and the first packet sent raises this exception (Connect reset by peer). The other is that one end exits, but the connection is not closed when exiting. If the other end is reading data from the connection, the exception (Connection reset) will be thrown. In short, it is caused by the read and write operations after the connection is disconnected.

In another case, if one end sends an RST packet to interrupt the TCP connection, the other end will also have this exception. If it is tomcat, the exception is as follows:

```
org.apache.catalina.connector.ClientAbortException: java.io.IOException: Connection reset by peer
```

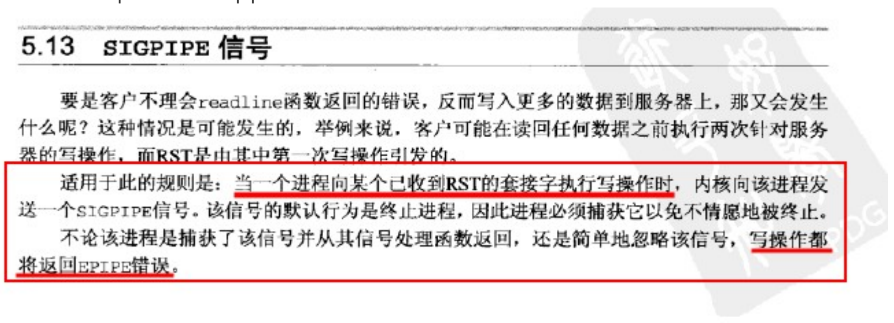
In order to improve the performance of Alibaba's tcp mode health check, the server will send an RST to terminate the connection, which will cause this exception;

For the server, the general reason can be considered as follows:

a) If the number of concurrent connections exceeds its capacity, the server will actively Down some of them

b) In the process of data transmission, the browser or the receiving client is closed, while the server is still sending data to the client.

6 java.net.SocketException: Broken pipe



5.13 SIGPIPE 信号

要是客户不理会readline函数返回的错误，反而写入更多的数据到服务器上，那又会发生什么呢？这种情况是可能发生的，举例来说，客户可能在读回任何数据之前执行两次针对服务器的写操作，而RST是由其中第一次写操作引发的。

适用于此的规则是：当一个进程向某个已收到RST的套接字执行写操作时，内核向该进程发送一个SIGPIPE信号。该信号的默认行为是终止进程，因此进程必须捕获它以免不情愿地被终止。

不论该进程是捕获了该信号并从其信号处理函数返回，还是简单地忽略该信号，写操作都将返回EPIPE错误。

This exception can occur on both the client and the server. after SocketExcepton:Connect reset by peer:Socket write error. If the data continues to be written, the exception will be thrown. The solution to the first two exceptions is to error," If the data continues to be written, the exception will be thrown. The solution to the first two exceptions is to make sure that all network connections are closed before the program exits. The second is to detect the closing operation of the other party. After the other party closes the connection, it should also close the connection itself.

Broken pipe only appears when write is called. Broken pipe means that the opposite end of the pipe has been disconnected, which often occurs If the read / write pipe is closed, you cannot read and write to the pipe. In terms of the four waves of tcp, the far end has sent the FIN serial number to tell you that the pipeline has been closed. At this time, if you continue to write data into the pipe, the first time you will receive an RST signal sent by the far end. If you continue to write data into the pipeline, the operating system will send you a SIGPIPE signal and set errno to broken Pipe (32). If your program does not process SIGPIPE by default, the program will interrupt and exit. In general, signal(SIGPIPE,SIG_IGN) can be used to ignores this signal, so the program will not exit, but write will return - 1 and set errno to broken pipe (32). Broken pipe only occurs when data is written to the closed pipeline of the opposite end (after receiving the rst serial number of the opposite end, the first write will not appear broken pipe, but write returns - 1. At this time, the correct way is to close the pipe of the local end. If you continue to write, this error will appear).

```
java.net.SocketException: Broken pipe (Write failed)        at java.net.SocketOutputStream.so
```

For the exception of 4 and 5, special attention should be paid to the maintenance of the connection. In the case of short connection, it's OK. In the case of long connection, if the maintenance of the connection state is improper, it's very easy to get exceptions. Basically, what we need to do for long connection is:

a) Detect the active disconnection of the other party (the other party calls the close method of Socket). Because the other party actively disconnects, if the other party is reading, the return value at this time is -1. Therefore, once the disconnection of the other party is detected, cloase connection in self end (call the close method of Socket).

b) To detect the other party's downtime, abnormal exit and network failure, the general method is heartbeat detection. Both sides send data to the other side periodically, and receive "heartbeat data" from the other side at the same time. If the other side's heartbeat is not received for several consecutive cycles, it can be judged that the other side is down, or exited abnormally, or the network is out of order. At this time, it also needs to actively close its own connection. If it is a client, it can restart the connection after a certain delay. Although Socket has a keep alive option to maintain the connection, it usually takes two hours to find the other party's downtime, abnormal exit and network failure.

7 java.net.SocketException: Too many open files
Reason: the maximum number of open file handles in the operating system is limited, which often occurs when many concurrent users access the server. Because in order to execute each user's application, the server has to load many files (a new socket needs a file handle), which leads to the lack of open file handles.

Solution:

a) Try to make the class a jar package, because a jar package consumes only one file handle. If it is not packaged, a class consumes one file handle.

b) java GC cannot close the file handle opened by network connection. If close() is not executed, the file handle will always exist and cannot be closed.

You can also consider setting the maximum number of open socket s to control this problem. Set the operating system to increase the maximum number of file handles.

ulimit -a can view the current resource limit of the system, ulimit -n 10240 can be modified, which is only valid for the current window.

8 Cannot assign requested address
1. The address cannot be bound because the port number is occupied:

java.net.BindException: Cannot assign requested address: bind: it is caused by the change of IP address;

2. server network configuration exception:

The address configured in /etc/hosts is wrong;

3. Another case is that there is no loop address when ipconfig is executed, because the loop address configuration file is lost;

Tag: Tomcat Posted by **fakedec** at Apr 02, 2021 - 11:59 AM