java.net.SocketException：Connection reset 原因分析与故障重现

作者：believelelf
原文链接：https://ld246.com/article/1572277642298
来源网站：链滴

前两天同事问了一个java.net.SocketException: Connection reset的问题，场景是在一个Java应用使用httpClient请求远程一个RestfulApi接口，但返回报错Connection reset。由于没有到现场查看志，只能通过查看源码和资料进行故障重现，本文记录了问题的分析过程。

异常信息

[2019-10-24 14:55:34.390][HttpClientUtils.java:797]-doPost发送POST请求出现异常:请求参数:{aa:bbb,ccc:ddd},url:http://xxx.xxx.com/xxx/xxx, 时间秒:0.019秒，Connection reset java.net.SocketException: Connection reset

在源码中定位异常信息

查看jdk10源码，发现Connection reset信息出现于方法 java.net.SocketInputStream#read(byte[], nt, int, int) 。

- 此方法通过间接调用socketRead-->socketRead0-->NET_Read-->read读取内核态缓存区数据。
- 如果NET_Read函数(unix系统)检测Connection reset或Broken pipe，将抛出sun/net/ConnectionResetException，相应地在jvm中会抛出异常SocketException("Connection reset")。
- 详情请见下方两段源码中文注释部分。

```
/**
 * Reads into a byte array <i>b</i> at offset <i>off</i>,
 * <i>length</i> bytes of data.
 * @param b the buffer into which the data is read
 * @param off the start offset of the data
 * @param length the maximum number of bytes read
 * @return the actual number of bytes read, -1 is
 *         returned when the end of the stream is reached.
 * @exception IOException If an I/O error has occurred.
```

```
     */
    public int read(byte b[], int off, int length) throws IOException {
        return read(b, off, length, impl.getTimeout());
    }

    int read(byte b[], int off, int length, int timeout) throws IOException {
        int n;

        // EOF already encountered
        if (eof) {
            return -1;
        }

        // connection reset
        if (impl.isConnectionReset()) {
            throw new SocketException("Connection reset");
        }

        // bounds check
        if (length <= 0 || off < 0 || length > b.length - off) {
            if (length == 0) {
                return 0;
            }
            throw new ArrayIndexOutOfBoundsException("length == " + length
                    + " off == " + off + " buffer length == " + b.length);
        }

        boolean gotReset = false;

        // acquire file descriptor and do the read
        FileDescriptor fd = impl.acquireFD();
        try {
            // 间接调用socketRead0函数，从内核态缓存区读取数据
            n = socketRead(fd, b, off, length, timeout);
            if (n > 0) {
                return n;
            }
        } catch (ConnectionResetException rstExc) {
            // 如果socketRead0抛出ConnectionResetException，此处进行捕获。后续再重试一次读
操作。
            gotReset = true;
        } finally {
            impl.releaseFD();
        }

        /*
         * We receive a "connection reset" but there may be bytes still
         * buffered on the socket
         * 重试读取操作
         */
        if (gotReset) {
            impl.setConnectionResetPending();
            impl.acquireFD();
            try {
```

```
                n = socketRead(fd, b, off, length, timeout);
                if (n > 0) {
                    return n;
                }
            } catch (ConnectionResetException rstExc) {
            } finally {
                impl.releaseFD();
            }
        }

        /*
         * If we get here we are at EOF, the socket has been closed,
         * or the connection has been reset.
         */
        if (impl.isClosedOrPending()) {
            throw new SocketException("Socket closed");
        }
        if (impl.isConnectionResetPending()) {
            impl.setConnectionReset();
        }
        if (impl.isConnectionReset()) {
            // 最终抛出异常
            throw new SocketException("Connection reset");
        }
        eof = true;
        return -1;
    }
```

- 下方源码为unix系统的socketRead0函数，可以看见相关errno的处理逻辑。源码相关链接为：

https://github.com/openjdk/jdk/blob/master/src/java.base/unix/native/libnet/SocketInputStream.c

```
/*
 * Class:     java_net_SocketInputStream
 * Method:    socketRead0
 * Signature: (Ljava/io/FileDescriptor;[BIII)I
 */
JNIEXPORT jint JNICALL
Java_java_net_SocketInputStream_socketRead0(JNIEnv *env, jobject this,
                                            jobject fdObj, jbyteArray data,
                                            jint off, jint len, jint timeout)
{
    char BUF[MAX_BUFFER_LEN];
    char *bufP;
    jint fd, nread;

    if (IS_NULL(fdObj)) {
        JNU_ThrowByName(env, "java/net/SocketException",
                        "Socket closed");
        return -1;
    }
    fd = (*env)->GetIntField(env, fdObj, IO_fd_fdID);
    if (fd == -1) {
        JNU_ThrowByName(env, "java/net/SocketException", "Socket closed");
```

```
    return -1;
}

/*
 * If the read is greater than our stack allocated buffer then
 * we allocate from the heap (up to a limit)
 */
if (len > MAX_BUFFER_LEN) {
    if (len > MAX_HEAP_BUFFER_LEN) {
        len = MAX_HEAP_BUFFER_LEN;
    }
    bufP = (char *)malloc((size_t)len);
    if (bufP == NULL) {
        bufP = BUF;
        len = MAX_BUFFER_LEN;
    }
} else {
    bufP = BUF;
}
if (timeout) {
    nread = NET_ReadWithTimeout(env, fd, bufP, len, timeout);
    if ((*env)->ExceptionCheck(env)) {
        if (bufP != BUF) {
            free(bufP);
        }
        return nread;
    }
} else {
    nread = NET_Read(fd, bufP, len);
}

if (nread <= 0) {
    if (nread < 0) {

        switch (errno) {
            case ECONNRESET:
            case EPIPE:
                /* 抛出异常的情况对应为Connection reset和Broken pipe
                * ECONNRESET      Connection reset (POSIX.1-2001).
                * EPIPE           Broken pipe (POSIX.1-2001).
                * @see http://man7.org/linux/man-pages/man3/errno.3.html
                */
                JNU_ThrowByName(env, "sun/net/ConnectionResetException",
                    "Connection reset");
                break;

            case EBADF:
                JNU_ThrowByName(env, "java/net/SocketException",
                    "Socket closed");
                break;

            case EINTR:
                 JNU_ThrowByName(env, "java/io/InterruptedIOException",
                      "Operation interrupted");
```

```
                    break;
                default:
                    JNU_ThrowByNameWithMessageAndLastError
                        (env, "java/net/SocketException", "Read failed");
            }
        }
    } else {
        (*env)->SetByteArrayRegion(env, data, off, nread, (jbyte *)bufP);
    }

    if (bufP != BUF) {
        free(bufP);
    }
    return nread;
}
```

java.net.SoecketException：Connection reset 出现原因

根据IBM开发者社区博客Understanding some common SocketExceptions in JAVA描述，在Java出现Connnection reset的原因为本地socket在接收或发送数据时收到远端系统带RST位的数据包。

● This exception appears when the remote connection is unexpectedly and forcefully closed ue to various reasons like application crash, system reboot, hard close of remote host. Kernel f om the remote system sends out a packets with RST bit to the local system. The local socket n performing any SEND (could be a Keep-alive packet) or RECEIVE operations subsequently fa l with this error. Certain combinations of linger settings can also result in packets with RST bit et.

RST报文段出现的场景

根据《TCP/IP详解.卷一：协议》，复位报文段（RST）可能在以下三种情况下出现：

1. 到不存在的端口的连接请求

● 当连接请求到达时，目的端口没有进程正在监听。即TCP连接建立时，客户端发送SYN报文段，务端的TCP端软件回复RST报文段，此过程对应Socket Api的connect函数，抛出异常为：Connection refused

● 出现阶段：建立连接时

2. 异常终止一个连接

● 正常终止一个连接的方式是一方发送FIN。因为在所有排队数据都已经发送之后才发送FIN,正常况下没有任何数据丢失。

● 存在一种异常终止连接的方式：一方发送一个复位报文段（RST）而不是FIN来中途释放一个连。其行为表现为：一方丢弃任何待发数据并立即发送复位报文段，收到RST的一方不会再回ACK报文，而是终止该连接，并通知应用层连接复位，抛出异常Connection reset by peer。

● Socket Api通过“linger on close”选项（SO_LINGER）提供了异常终止连接的能力。

● 详情可以继续参考 TCP option SO_LINGER (zero) - when it's required

● 出现阶段：终止连接时

3. 检测半开连接

- 半开连接：如果一方已经关闭或异常终止连接而另一方却不知道，这样的TCP连接称为半开连接。异常的一方没有发出FIN报文段来通知对方开始终止连接。

- 现象： 只要不打算在半开连接上传输数据，仍处于连接状态的一方就不会检测另一方已经出现常。这个异常会在处于连接状态一方进行read或write时抛出，错误信息为Connection reset或Connection reset by peer。

- 原因：异常一方已经丢失了连接的所有信息，它不知道另一方发送数据报文段中提到的连接。TC的处理原理是接收方以复位（RST）作为回应。

- 什么时候有半开连接？更多情况可见 Detection of Half-Open (Dropped) Connections

  - application crash 应用崩溃
  - system reboot 系统重启（导致应用崩溃或导致应用崩溃后重启）
  - hard close of remote host (远程主机的硬关闭(掉电等))
  - Router crash/reboot 两方中间某跳路由器崩溃或重启

- 出现阶段：读取或写入数据时

重现故障的方案

根据RST报文段出现的三种场景，分三种情况对故障进行重现。

到不存在的端口的连接请求

重现步骤

1. 准备一个客户端httpClient请求示例，用于发起远程RestfulApi接口调用。示例程序链接见参考资中的connection-reset-demo。

```
/**
 * java.net.SocketException : Connection reset 测试客户端
 */
@SpringBootApplication
@RestController
public class ApplicationCenter {

    private static final Logger LOGGER = LoggerFactory.getLogger(ApplicationCenter.class);

    @Autowired
    private RestTemplate restTemplate;

    public static void main(String[] args) {
        SpringApplication.run(ApplicationCenter.class, args);
    }

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder){
        return builder.build();
    }

    @RequestMapping(path = "/hello", produces = MediaType.APPLICATION_JSON_VALUE)
```

```
    public String serverHello(@RequestParam("key")String key) {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        MultiValueMap<String, String> map= new LinkedMultiValueMap<>();
        map.add("key", key);
        map.add("requestId", UUID.randomUUID().toString());
        map.add("timestamp", String.valueOf(System.currentTimeMillis()));
        HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);

        String value = restTemplate.postForObject("http://www.weiquding.com:8083/hello", req
est, String.class);
        LOGGER.info("receive value:[{}]", value);
        return value;
    }
}
```

2. 准备一个服务端RestfulApi服务接口，并将服务端部署在远程主机上，此情况下不启动应用，另要保远端防火墙相应的端口8083已经打开。

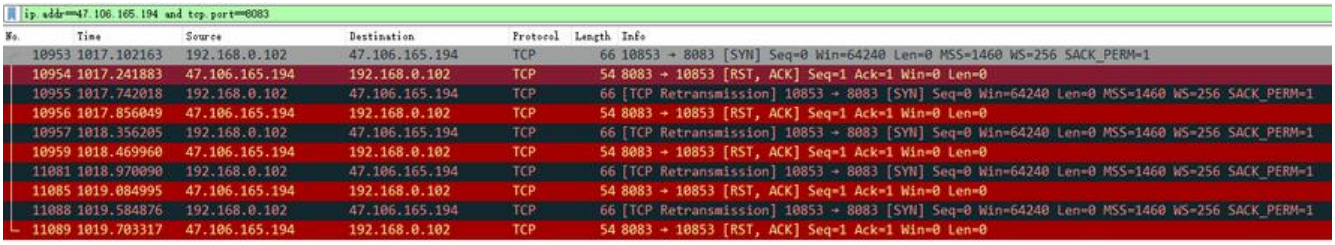
```
/**
 * java.net.SocketException : Connection reset 测试服务端
 */
@SpringBootApplication
@RestController
public class ApplicationCenter {

    private static final Logger LOGGER = LoggerFactory.getLogger(ApplicationCenter.class);

    public static void main(String[] args) {
        SpringApplication.run(ApplicationCenter.class, args);
    }

    @RequestMapping(path = "/hello", produces = MediaType.APPLICATION_JSON_VALUE)
    public String serverHello(HttpServletRequest request) {
        Enumeration<String> headerNames = request.getHeaderNames();
        while (headerNames.hasMoreElements()) {
            String headerName = headerNames.nextElement();
            LOGGER.info("headerName:[{}]==>{}", headerName, request.getHeader(headerName))

        }
        Enumeration<String> parameterNames = request.getParameterNames();
        while (parameterNames.hasMoreElements()){
            String parameterName = parameterNames.nextElement();
            LOGGER.info("parameterName:[{}]==>{}", parameterName, request.getParameter(par
meterName));
        }
        return "server hello";
    }

}
```

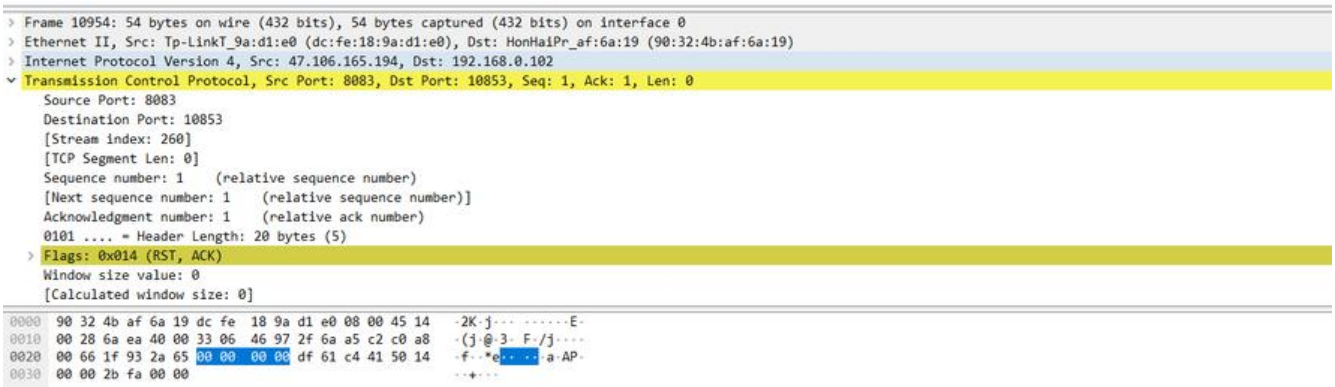
3. 在客户端打开wireshark软件,选择对应的数据交换的网络接口，在显示过滤器栏位输入规则"ip.addr ==

xxx.xxx.xxx.xxx and tcp.port==8083",
便只显示对socket上的流量。

4. 启动客户端程序，调用远端服务。产生服务调用的方式为在本地浏览器输入http://localhost:8082/ello?key=client

5. 在wireshark的Packet List面板，查看双方交互的数据包。





6. 打开客户端控制台，查看报错的堆栈信息。

```
java.net.ConnectException: Connection refused: connect
    at java.base/java.net.DualStackPlainSocketImpl.connect0(Native Method) ~[na:na]
    at java.base/java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.ja
a:79) ~[na:na]
    at java.base/java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:400)
~[na:na]
    at java.base/java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.j
va:243) ~[na:na]
    at java.base/java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:225) ~[
a:na]
    at java.base/java.net.PlainSocketImpl.connect(PlainSocketImpl.java:148) ~[na:na]
    at java.base/java.net.Socket.connect(Socket.java:591) ~[na:na]
    at java.base/java.net.Socket.connect(Socket.java:540) ~[na:na]
    at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:182) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:474) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:569) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.<init>(HttpClient.java:242) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.New(HttpClient.java:341) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.New(HttpClient.java:362) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.getNewHttpClient(HttpURLCo
nection.java:1242) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.plainConnect0(HttpURLConnec
ion.java:1181) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.plainConnect(HttpURLConnect
on.java:1075) ~[na:na]
```

```
    at java.base/sun.net.www.protocol.http.HttpURLConnection.connect(HttpURLConnection.ja
a:1009) ~[na:na]
    at org.springframework.http.client.SimpleBufferingClientHttpRequest.executeInternal(Simpl
BufferingClientHttpRequest.java:76) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.http.client.AbstractBufferingClientHttpRequest.executeInternal(Abs
ractBufferingClientHttpRequest.java:48) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.http.client.AbstractClientHttpRequest.execute(AbstractClientHttpR
quest.java:53) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:742) ~[sprin
-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:677) ~[spring-
eb-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.postForObject(RestTemplate.java:421) ~[s
ring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at com.weiquding.connection.reset.demo.client.ApplicationCenter.serverHello(ApplicationC
nter.java:53) ~[classes/:na]
```

分析

1. 由上节wireshark捕获的数据包可知，客户端与服务端只存在一次有效交互，即客户端发送SYN报段，试图建立连接，但服务端由于没有端口监听，回了RST报文段。

2. 由上节报错的堆栈信息可知，客户端建立连接的过程最终会调用到Native方程connect0,以window平台为例，方法为java.net.DualStackPlainSocketImpl#connect0，对应源码PlainSocketImpl#connect0如下：

```
/*
 * Class:     java_net_PlainSocketImpl
 * Method:    connect0
 * Signature: (ILjava/net/InetAddress;I)I
 */
JNIEXPORT jint JNICALL Java_java_net_PlainSocketImpl_connect0
  (JNIEnv *env, jclass clazz, jint fd, jobject iaObj, jint port) {
    SOCKETADDRESS sa;
    int rv, sa_len = 0;
    jboolean v4MappedAddress = ipv6_available() ? JNI_TRUE : JNI_FALSE;

    if (NET_InetAddressToSockaddr(env, iaObj, port, &sa,
                                  &sa_len, v4MappedAddress) != 0) {
        return -1;
    }

    rv = connect(fd, &sa.sa, sa_len);
    if (rv == SOCKET_ERROR) {
        int err = WSAGetLastError();
        if (err == WSAEWOULDBLOCK) {
            return java_net_PlainSocketImpl_WOULDBLOCK;
        } else if (err == WSAEADDRNOTAVAIL) {
            JNU_ThrowByName(env, JNU_JAVANETPKG "ConnectException",
                "connect: Address is invalid on local machine,"
                " or port is not valid on remote machine");
        } else {
            /*
            * 当errno为WSAECONNREFUSED时，走此分支，抛出异常：java/net/ConnectException
```

```
Connection refused: connect
            *{ WSAECONNREFUSED,         "ConnectException",     "Connection refused" },
            */
            NET_ThrowNew(env, err, "connect");
        }
        // return value not important.
    }
    return rv;
}
```

3. 此种情况下，异常在TCP建立过程中发生，抛出异常不是java.net.SocketException:Connection reset。

检测半开连接

重现步骤

1. 准备一个客户端httpClient请求示例。同上一情况。

2. 准备一个服务端RestfulApi服务接口，并将服务端部署在远程主机上，此情况下先启动应用，等客端执行read或write前再强制杀掉服务端应用，另要确保远端防火墙相应的端口8083已经打开。

3. 在客户端打开wireshark软件,选择对应的数据交换的网络接口（此时选择本地回环网络接口Loopback,因为因特网半开连接不好模拟），在显示过滤器栏位输入规则"ip.addr == xxx.xxx.xxx.xxx and tcp.port==8083",
便只显示对socket上的流量。

客户端write时遇到半开连接的情况

4. 首先看write时遇到半开连接的情况，在java.net.SocketOutputStream#socketWrite方法的socketWrite0方法调用处打一个断点。

```
/**
     * Writes to the socket with appropriate locking of the
     * FileDescriptor.
     * @param b the data to be written
     * @param off the start offset in the data
     * @param len the number of bytes that are written
     * @exception IOException If an I/O error has occurred.
     */
    private void socketWrite(byte b[], int off, int len) throws IOException {


        if (len <= 0 || off < 0 || len > b.length - off) {
            if (len == 0) {
                return;
            }
            throw new ArrayIndexOutOfBoundsException("len == " + len
                    + " off == " + off + " buffer length == " + b.length);
        }

        FileDescriptor fd = impl.acquireFD();
        try {
```

```
            // 以下行为断点位置
            socketWrite0(fd, b, off, len);
        } catch (SocketException se) {
            /* 此处会抛出异常：java.net.SocketException: Connection reset by peer: socket write er
or
             * 但由于整个out.flush过程会调用三次，且异常发生后存在半开连接的拆除和重连接。
             * 最终的异常信息为：java.net.ConnectException: Connection refused: connect
             *
             */
            if (se instanceof sun.net.ConnectionResetException) {
                impl.setConnectionResetPending();
                se = new SocketException("Connection reset");
            }
            if (impl.isClosedOrPending()) {
                throw new SocketException("Socket closed");
            } else {
                // 异常抛出之处
                throw se;
            }
        } finally {
            impl.releaseFD();
        }
    }
```

5. 启动客户端程序，调用远端服务。产生服务调用的方式为在本地浏览器输入http://localhost:8082/ello?key=client

6. 当客户端程序运行到socketWrite0时，强关服务端程序（idea中直接关停应用即可），然后再放断点让客户端程序执行完成。此时客户端应先已经产生异常：java.net.SocketException: Connection reset by peer: socket write error，但不抛出，这部分原因在分析章节指出。

7. . 在wireshark的Packet List面板，查看双方交互的数据包。

8. 打开客户端控制台，查看报错的堆栈信息。

```
java.net.ConnectException: Connection refused: connect
    at java.base/java.net.DualStackPlainSocketImpl.connect0(Native Method) ~[na:na]
    at java.base/java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.ja
a:79) ~[na:na]
    at java.base/java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:400)
~[na:na]
    at java.base/java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.j
va:243) ~[na:na]
    at java.base/java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:225) ~[
a:na]
    at java.base/java.net.PlainSocketImpl.connect(PlainSocketImpl.java:148) ~[na:na]
    at java.base/java.net.Socket.connect(Socket.java:591) ~[na:na]
    at java.base/java.net.Socket.connect(Socket.java:540) ~[na:na]
    at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:182) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:474) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:569) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.<init>(HttpClient.java:242) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.New(HttpClient.java:341) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.New(HttpClient.java:376) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.setNewClient(HttpURLConnect
on.java:793) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.setNewClient(HttpURLConnect
on.java:781) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.writeRequests(HttpURLConnect
on.java:723) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.getOutputStream0(HttpURLCo
nection.java:1366) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.getOutputStream(HttpURLCon
ection.java:1331) ~[na:na]
    at org.springframework.http.client.SimpleBufferingClientHttpRequest.executeInternal(Simpl
BufferingClientHttpRequest.java:78) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.http.client.AbstractBufferingClientHttpRequest.executeInternal(Abs
ractBufferingClientHttpRequest.java:48) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.http.client.AbstractClientHttpRequest.execute(AbstractClientHttpR
quest.java:53) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:742) ~[sprin
-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:677) ~[spring-
eb-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.postForObject(RestTemplate.java:421) ~[s
ring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at com.weiquding.connection.reset.demo.client.ApplicationCenter.serverHello(ApplicationC
nter.java:53) ~[classes/:na]
```

客户端write时遇到半开连接情况的分析

1. 客户端在java.net.SocketOutputStream#socketWrite方法中得到异常信息java.net.SocketExcept
on: Connection reset by peer: socket write error后，为什么没有抛出?

- 由于存在整个PrintStream-->BufferedOutputStream-->SocketOutputStream包装，out.flus
过程会调用三次socketWirte，且异常发生后异常信息会捕获，后续进行半开连接的拆除和重连接。
异常捕获PrintStream.flush()方法及三次socketWrite调用及连接重建的的栈信息。

```
/**
```

```
 * Flushes the stream.  This is done by writing any buffered output bytes to
 * the underlying output stream and then flushing that stream.
 *
 * @see        java.io.OutputStream#flush()
 */
public void flush() {
    synchronized (this) {
        try {
            ensureOpen();
            //此处调用BufferedOutputStream.flush();间接调用SocketOutputStream.socketWrite。
            out.flush();
        }
        catch (IOException x) {
            // 异常捕获，设置异常标志。
            trouble = true;
        }
    }
}
```

- 第一次调用socketWrite方法的栈信息。

```
socketWrite:111, SocketOutputStream (java.net)
write:155, SocketOutputStream (java.net)
flushBuffer:81, BufferedOutputStream (java.io)
flush:142, BufferedOutputStream (java.io)
flush:417, PrintStream (java.io)
print:301, MessageHeader (sun.net.www)
writeRequests:655, HttpClient (sun.net.www.http)
writeRequests:666, HttpClient (sun.net.www.http)
writeRequests:711, HttpURLConnection (sun.net.www.protocol.http)
getOutputStream0:1366, HttpURLConnection (sun.net.www.protocol.http)
getOutputStream:1331, HttpURLConnection (sun.net.www.protocol.http)
executeInternal:78, SimpleBufferingClientHttpRequest (org.springframework.http.client)
executeInternal:48, AbstractBufferingClientHttpRequest (org.springframework.http.client)
execute:53, AbstractClientHttpRequest (org.springframework.http.client)
doExecute:742, RestTemplate (org.springframework.web.client)
execute:677, RestTemplate (org.springframework.web.client)
postForObject:421, RestTemplate (org.springframework.web.client)
serverHello:53, ApplicationCenter (com.weiquding.connection.reset.demo.client)
```

- HttpClient#writeRequests(sun.net.www.MessageHeader, sun.net.www.http.PosterOutputtream)方法

```
public void writeRequests(MessageHeader head,
                          PosterOutputStream pos) throws IOException {
    requests = head;
    // 第一次调用SocketOutputStream.socketWrite的入口
    requests.print(serverOutput);
    poster = pos;
    if (poster != null)
        poster.writeTo(serverOutput);
    // PrintStream.flush, 第二次调用SocketOutputStream.socketWrite的入口
    serverOutput.flush();
}
```

- 第二调用SocketOutputStream.socketWrite的栈信息

```
socketWrite:111, SocketOutputStream (java.net)
write:155, SocketOutputStream (java.net)
flushBuffer:81, BufferedOutputStream (java.io)
flush:142, BufferedOutputStream (java.io)
flush:417, PrintStream (java.io)
writeRequests:659, HttpClient (sun.net.www.http)
writeRequests:666, HttpClient (sun.net.www.http)
writeRequests:711, HttpURLConnection (sun.net.www.protocol.http)
getOutputStream0:1366, HttpURLConnection (sun.net.www.protocol.http)
getOutputStream:1331, HttpURLConnection (sun.net.www.protocol.http)
executeInternal:78, SimpleBufferingClientHttpRequest (org.springframework.http.client)
executeInternal:48, AbstractBufferingClientHttpRequest (org.springframework.http.client)
execute:53, AbstractClientHttpRequest (org.springframework.http.client)
doExecute:742, RestTemplate (org.springframework.web.client)
execute:677, RestTemplate (org.springframework.web.client)
postForObject:421, RestTemplate (org.springframework.web.client)
serverHello:53, ApplicationCenter (com.weiquding.connection.reset.demo.client)
```

- java.io.PrintStream#checkError方法

```
/**
 * Flushes the stream and checks its error state. The internal error state
 * is set to {@code true} when the underlying output stream throws an
 * {@code IOException} other than {@code InterruptedIOException},
 * and when the {@code setError} method is invoked.  If an operation
 * on the underlying output stream throws an
 * {@code InterruptedIOException}, then the {@code PrintStream}
 * converts the exception back into an interrupt by doing:
 * <pre>{@code
 *     Thread.currentThread().interrupt();
 * }</pre>
 * or the equivalent.
 *
 * @return {@code true} if and only if this stream has encountered an
 *         {@code IOException} other than
 *         {@code InterruptedIOException}, or the
 *         {@code setError} method has been invoked
 */
public boolean checkError() {
    if (out != null)
        // PrintStream.flush, 第三次调用SocketOutputStream.socketWrite的入口
        flush();
    if (out instanceof java.io.PrintStream) {
        PrintStream ps = (PrintStream) out;
        return ps.checkError();
    }
    // 返回是否异常，true将进入连接重建逻辑。
    return trouble;
}
```

- 第三次调用SocketOutputStream.socketWrite的栈信息

```
socketWrite:111, SocketOutputStream (java.net)
write:155, SocketOutputStream (java.net)
flushBuffer:81, BufferedOutputStream (java.io)
```

```
flush:142, BufferedOutputStream (java.io)
flush:417, PrintStream (java.io)
checkError:471, PrintStream (java.io)
writeRequests:712, HttpURLConnection (sun.net.www.protocol.http)
getOutputStream0:1366, HttpURLConnection (sun.net.www.protocol.http)
getOutputStream:1331, HttpURLConnection (sun.net.www.protocol.http)
executeInternal:78, SimpleBufferingClientHttpRequest (org.springframework.http.client)
executeInternal:48, AbstractBufferingClientHttpRequest (org.springframework.http.client)
execute:53, AbstractClientHttpRequest (org.springframework.http.client)
doExecute:742, RestTemplate (org.springframework.web.client)
execute:677, RestTemplate (org.springframework.web.client)
postForObject:421, RestTemplate (org.springframework.web.client)
serverHello:53, ApplicationCenter (com.weiquding.connection.reset.demo.client)
```

- sun.net.www.protocol.http.HttpURLConnection#writeRequests重建TCP链接逻辑分析

```
/* adds the standard key/val pairs to reqests if necessary & write to
 * given PrintStream
 */
private void writeRequests() throws IOException {
    /*
    * 省略设计http请求首部的逻辑
    */

    // 前两次的SocketOutputStream.socketWrite调用入口
    http.writeRequests(requests, poster, streaming());
    // 第三次SocketOutputStream.socketWrite调用，并检查是否有异常
    if (ps.checkError()) {
        String proxyHost = http.getProxyHostUsed();
        int proxyPort = http.getProxyPortUsed();
        // 断开半开连接
        disconnectInternal();
        if (failedOnce) {
            throw new IOException("Error writing to server");
        } else { // try once more
            failedOnce=true;
            if (proxyHost != null) {
                setProxiedClient(url, proxyHost, proxyPort);
            } else {
                // 进行一次连接重建，此时会抛出java.net.ConnectException: Connection refused: con
ect
                setNewClient (url);
            }
            ps = (PrintStream) http.getOutputStream();
            connected=true;
            responses = new MessageHeader();
            setRequests=false;
            // 递归调用，以便重新进入socketWrite调用
            writeRequests();
        }
    }
}
```

2. 从上述源码和栈调用可以看出，HttpURLConnection在处理write时进行很好的容错处理，包括以两点：

- 如果缓冲区仍存在数据，多次尝试写入socket.
  - 多次写失败后，尝试重建连接，如果新的连接建立成功，再次开始写入流程。

客户端read时遇到半开连接情况

1. 前三步与wirte时一致，不再叙述。对于read时遇到半开连接的情况，在java.net.SocketInputStream#read(byte[], int, int, int)方法的socketRead方法调用处打一个断点。

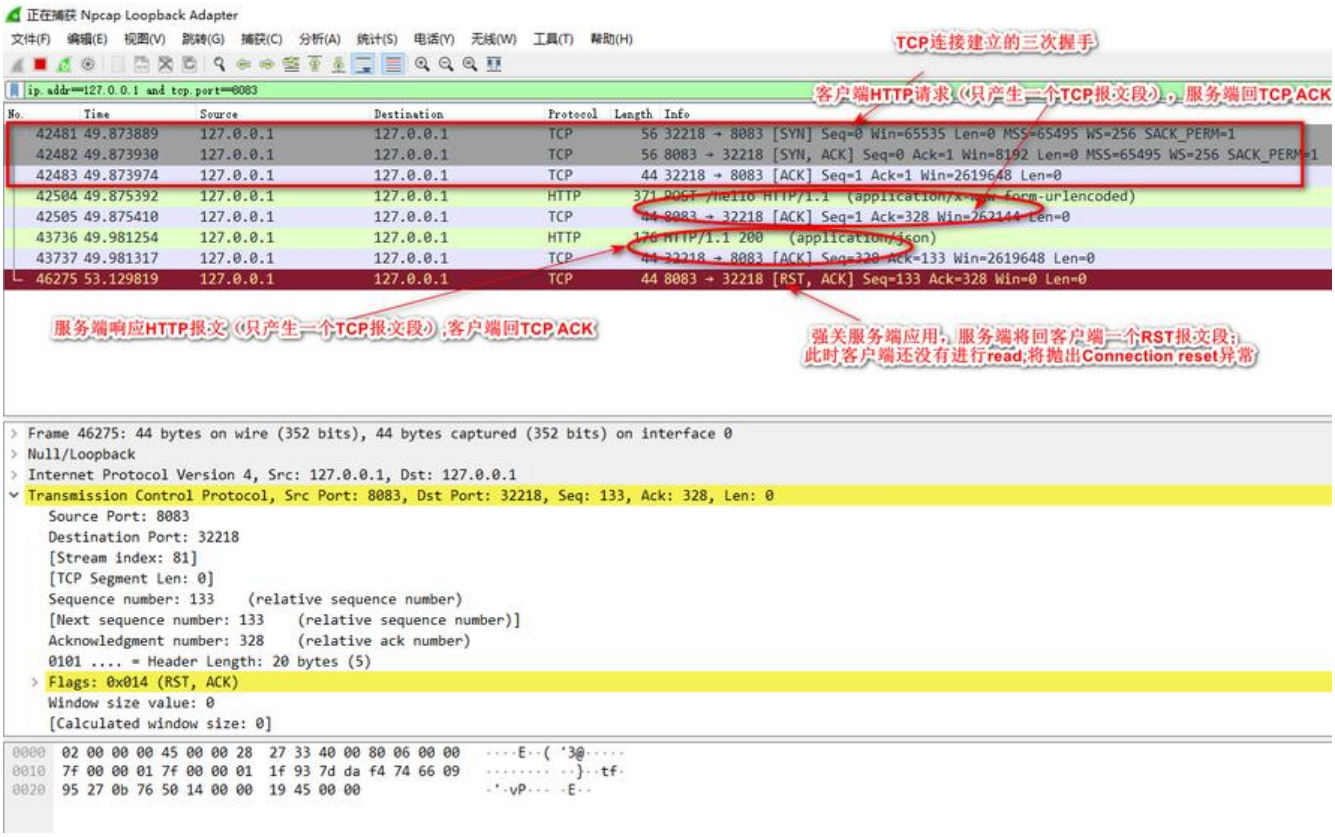
```
/**
     * Reads into a byte array <i>b</i> at offset <i>off</i>,
     * <i>length</i> bytes of data.
     * @param b the buffer into which the data is read
     * @param off the start offset of the data
     * @param length the maximum number of bytes read
     * @return the actual number of bytes read, -1 is
     *          returned when the end of the stream is reached.
     * @exception IOException If an I/O error has occurred.
     */
    public int read(byte b[], int off, int length) throws IOException {
        return read(b, off, length, impl.getTimeout());
    }

    int read(byte b[], int off, int length, int timeout) throws IOException {
        int n;

        // EOF already encountered
        if (eof) {
            return -1;
        }

        // connection reset
        if (impl.isConnectionReset()) {
            throw new SocketException("Connection reset");
        }

        // bounds check
        if (length <= 0 || off < 0 || length > b.length - off) {
            if (length == 0) {
                return 0;
            }
            throw new ArrayIndexOutOfBoundsException("length == " + length
                    + " off == " + off + " buffer length == " + b.length);
        }

        boolean gotReset = false;

        // acquire file descriptor and do the read
        FileDescriptor fd = impl.acquireFD();
        try {
            // 以下行为打断点处，此方法将调用Native方法socketRead0
            n = socketRead(fd, b, off, length, timeout);
            if (n > 0) {
                return n;
```

```
            }
        } catch (ConnectionResetException rstExc) {
            // 捕获sun.net.ConnectionResetException连接重置异常，改为设置标志以便再次尝试读取
存区数据
            gotReset = true;
        } finally {
            impl.releaseFD();
        }

        /*
         * We receive a "connection reset" but there may be bytes still
         * buffered on the socket
         * 再次尝试读取缓存区数据
         */
        if (gotReset) {
            impl.setConnectionResetPending();
            impl.acquireFD();
            try {
                n = socketRead(fd, b, off, length, timeout);
                if (n > 0) {
                    return n;
                }
            } catch (ConnectionResetException rstExc) {
            } finally {
                impl.releaseFD();
            }
        }

        /*
         * If we get here we are at EOF, the socket has been closed,
         * or the connection has been reset.
         */
        if (impl.isClosedOrPending()) {
            throw new SocketException("Socket closed");
        }
        if (impl.isConnectionResetPending()) {
            impl.setConnectionReset();
        }
        if (impl.isConnectionReset()) {
            // 如果两次都因为连接重置没有读取成功，转换异常，抛出java.net.SocketException: Conn
ction reset异常。
            throw new SocketException("Connection reset");
        }
        eof = true;
        return -1;
    }
```

2. 启动客户端程序，调用远端服务。产生服务调用的方式为在本地浏览器输入http://localhost:8082/ello?key=client

3. 当客户端程序运行到socketRead时，强关服务端程序（idea中直接关停应用即可），然后再放开点让客户端程序执行完成。此时客户端产出异常：sun.net.ConnectionResetException。

4. 在wireshark的Packet List面板，查看双方交互的数据包。



5. 打开客户端控制台，查看报错的堆栈信息。

```
java.net.SocketException: Connection reset
    at java.base/java.net.SocketInputStream.read(SocketInputStream.java:210) ~[na:na]
    at java.base/java.net.SocketInputStream.read(SocketInputStream.java:141) ~[na:na]
    at java.base/java.io.BufferedInputStream.fill(BufferedInputStream.java:252) ~[na:na]
    at java.base/java.io.BufferedInputStream.read1(BufferedInputStream.java:292) ~[na:na]
    at java.base/java.io.BufferedInputStream.read(BufferedInputStream.java:351) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.parseHTTPHeader(HttpClient.java:746) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.parseHTTP(HttpClient.java:689) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream0(HttpURLCon
ection.java:1604) ~[na:na]
    at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConne
tion.java:1509) ~[na:na]
    at java.base/java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:527) ~
na:na]
    at org.springframework.http.client.SimpleClientHttpResponse.getRawStatusCode(SimpleCli
ntHttpResponse.java:55) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.DefaultResponseErrorHandler.hasError(DefaultResponse
rrorHandler.java:55) ~[spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.handleResponse(RestTemplate.java:773) ~
spring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:743) ~[sprin
-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:677) ~[spring-
eb-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at org.springframework.web.client.RestTemplate.postForObject(RestTemplate.java:421) ~[s
ring-web-5.2.0.RELEASE.jar:5.2.0.RELEASE]
    at com.weiquding.connection.reset.demo.client.ApplicationCenter.serverHello(ApplicationC
```

nter.java:53) ~[classes/:na]

客户端read时遇到半开连接情况的分析

1. read产生Connection reset需要在第一次进行socketRead时服务端已经回了RST报文段。

2. 从SocketInputStream.read方法的源码中，可以在异常发生后还会尝试进行一次重试，考虑了容性。

异常终止一个连接

重现步骤

1. 由于没有找到tomcat直接没有设置Socket参数SO_LINGER的地方，写一对Socket客户端和服务端模拟此情形。

2. 写一个Socket客户端。示例代码见参考资料的connetion-reset-demo。

```
@RequestMapping(path = "/socket", produces = MediaType.APPLICATION_JSON_VALUE)
public String connectSocket(@RequestParam("key")String key) {
    Socket socket = null;
    InputStream inputStream = null;
    OutputStream outputStream = null;
    try {
        socket = new Socket();
        socket.connect(new InetSocketAddress("127.0.0.1", 8084));
        outputStream = socket.getOutputStream();
        outputStream.write(key.getBytes(StandardCharsets.UTF_8));
        inputStream = socket.getInputStream();
        byte[] rcvBytes = new byte[1024];
        inputStream.read(rcvBytes);
        return new String(rcvBytes, StandardCharsets.UTF_8);
    }catch (IOException e){
        LOGGER.error("socket通信异常", e);
    }finally {
        if(outputStream != null){
            try {
                outputStream.close();
            }catch (IOException e){
                LOGGER.error("关闭outputStream异常", e);
            }
        }
        if(inputStream != null){
            try {
                inputStream.close();
            }catch (IOException e){
                LOGGER.error("关闭inputStream异常", e);
            }
        }
        if(socket != null){
            try {
                socket.close();
            } catch (IOException e) {
```

```
                LOGGER.error("关闭socket异常", e);
            }
        }
    }
    return "";
  }
}
```

3. 写一个ServerSocket服务端,设置SO_LINGER参数

```
public class EchoServer {

    private static final Logger LOGGER = LoggerFactory.getLogger(EchoServer.class);


    public static void main(String[] args) {
        try(ServerSocket serverSocket = new ServerSocket(8084)) {
            while(true){
                Socket accept = serverSocket.accept();
                new Thread(new ThreadHandler(accept)).start();
            }
        } catch (IOException e) {
            LOGGER.error("serverSocket异常", e);
        }
    }

    static class ThreadHandler implements Runnable{

        private Socket socket;

        public ThreadHandler(Socket socket){
            this.socket = socket;
        }

        @Override
        public void run() {
            InputStream inputStream = null;
            OutputStream outputStream = null;
            try {
                // 设置SO_LINGER
                socket.setSoLinger(true, 0);
                inputStream = socket.getInputStream();
                byte[] rcvBytes = new byte[1024];
                inputStream.read(rcvBytes);
                LOGGER.info("receive data:[{}]", new String(rcvBytes, StandardCharsets.UTF_8));
                outputStream = socket.getOutputStream();
                outputStream.write("server socket".getBytes(StandardCharsets.UTF_8));
                socket.close();
            } catch (IOException e) {
                LOGGER.error("处理socket异常", e);
            }finally {
                if(inputStream != null){
                    try {
                        inputStream.close();
                    }catch (IOException e){
```

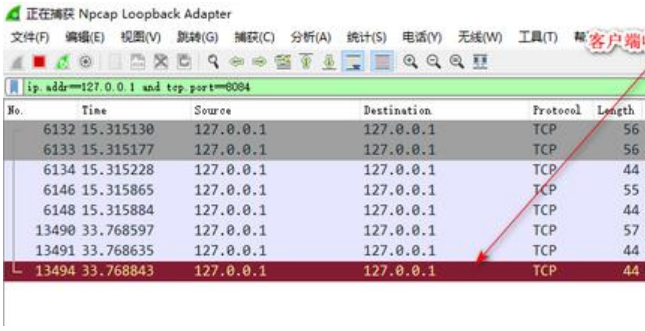
```
                    LOGGER.error("关闭inputStream异常", e);
                }
            }
            if(outputStream != null){
                try {
                    outputStream.close();
                }catch (IOException e){
                    LOGGER.error("关闭outputStream异常", e);
                }
            }
            try {
                socket.close();
            } catch (IOException e) {
                LOGGER.error("关闭socket异常", e);
            }
        }

    }
  }
}
```

4. 启动服务端进行监听

5. 在客户端打开wireshark软件,选择对应的数据交换的网络接口（此时选择本地回环网络接口Loopback,因为因特网半开连接不好模拟），在显示过滤器栏位输入规则"ip.addr == xxx.xxx.xxx.xxx and tcp.port==8084",
便只显示对应socket上的流量。

6. 启动客户端，通过调用http://localhost:8082/socket?key=client,产生服务端socket通信。

7. 在wireshark的Packet List面板，查看双方交互的数据包。



8. 打开客户端控制台，查看报错的堆栈信息。

```
java.net.SocketException: Connection reset
    at java.base/java.net.SocketInputStream.read(SocketInputStream.java:210) ~[na:na]
    at java.base/java.net.SocketInputStream.read(SocketInputStream.java:141) ~[na:na]
    at java.base/java.net.SocketInputStream.read(SocketInputStream.java:127) ~[na:na]
    at com.weiquding.connection.reset.demo.client.ApplicationCenter.connectSocket(ApplicationCenter.java:82) ~[classes/:na]
```

分析

1. 服务端设置了SO_LINGER,关闭链接不会发FIN报文段，而是发RST报文段。

2. 客户端收到此RST报文段，再进行SocketInputStream.read，将抛出java.net.SocketException: Connection reset。

解决故障的方案

1. 对于“到不存在的端口的连接请求”的情况。

- 如果是程序端口错误，无法解决。
- 其它情况可以在设置服务端多实例，客户端使用降级重试策略，提高可用性。

2. 对于“检测半开连接”的情况。

- 可以使用Http短连接，减少长连接使用过程服务端发生异常导致半开连接的机会，但这会降低能。
- 客户端提供Http连接池，单个连接使用长连接并配置较短的超时时间，对连接复用，能减少连创建的消耗；同时连接池提供连接有效性的检测，有无效连接进行拆除。
- 如果服务端只有单一实例，其反复出现应用崩溃等引起半开连接，则要检测应用程序是否有问。另外要考虑服务端多实例部署，并在客户端准备降级策略。

3. 对于“异常终止一个连接”的情况。

- 服务端不要设置Socket参数SO_LINGER，执行正常的连接终止流程。同时连接的终止应当考虑客户端发起。

参考资料

1. Setting the SO_LINGER Option
2. TCP option SO_LINGER (zero) - when it's required
3. What's causing my java.net.SocketException: Connection reset?
4. java.net.SocketException: Connection reset
5. Understanding some common SocketExceptions in JAVA
6. Tomcat配置api
7. TCP/IP详解.卷一 P187 异常中止一个连接
8. HTTP权威指南 P100 Keep-Alive和哑代理
9. 计算机网络（自顶向下方法）P170 TCP连接管理
10. Wireshark数据包分析实战 P117 TCP 重置
11. TIME_WAIT and its design implications for protocols and scalable client server systems
12. connection-reset-demo示例程序地址
13. Detection of Half-Open (Dropped) Connections