

TCP 链路重置场景总结

秦新良

2014 年 7 月 19 日

目 录

1	TCP 链路重置场景	2
1.1	服务端端口未监听	2
1.2	SO_LINGER 选项	2
1.3	半打开连接	3
1.4	接收缓存不为空	3

1 TCP 链路重置场景

TCP 链路被重置导致业务失败的情况时有发生，一般我们都是通过抓包来看是客户端还是服务端发起的 RST，然后再进一步分析为什么会 RST。本章列出 TCP 复位的几种场景，可作为定位该类问题的一个参考。

1.1 服务端端口未监听

如果在建立连接时链路就被重置了，表示服务端的端口未启动监听，可通过 netstat 命令查看服务端的端口是否监听。如图1所示，表示在服务端有三个端口 53、631、8080 启动监听。

```
q00148943@Inspiron-3421:~$ netstat -an | grep LISTEN | grep tcp
tcp        0      0 127.0.1.1:53          0.0.0.0:*             LISTEN
tcp        0      0 127.0.0.1:631         0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:8080          0.0.0.0:*             LISTEN
tcp6       0      0 :::1:631              :::*                   LISTEN
q00148943@Inspiron-3421:~$
```

图 1: netstat

该场景下使用 Wireshark 抓包结果如图3所示。

No.	Time	Source	Destination	Protocol	Length	Info
1	2014-07-14 07:36:01.410071000	192.168.1.103	128.199.194.158	TCP	74	50031 > http-alt [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=408887
2	2014-07-14 07:36:01.499817000	128.199.194.158	192.168.1.103	TCP	54	http-alt > 50031 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

图 2: RST

从图3中可以看出，客户端发了一个 SYN 包后，服务端立刻回了一个 RST 将链路重置。

1.2 SO_LINGER 选项

实际的应用中，客户端一般来说是作为主动断链的一方，应用程序关闭 socket 后，TCP 链路会进入 TIME_WAIT 状态，在等待 2MSL 后，该链路才会彻底释放。在 2MSL 时间内，客户端是不能复用这个端口的。在大并发或客户端本地端口在小范围内随机的场景下，出现端口复用的概率非常大。为解决这一问题，有时候我们会采用代码1的方式来解决这一问题。这样应用程序关闭 socket 后，TCP 会直接将链路 RST，而不会进入 TIME_WAIT 状态。

```
1
2     struct linger so_linger;
3
4     so_linger.l_onoff  = 1;
5     so_linger.l_linger = 0;
6
7     setsockopt(sock, SOL_SOCKET, SO_LINGER, &so_linger, sizeof(
        so_linger));
```

代码 1: 设置 SO_LINGER

1.3 半打开连接

该场景一般是由于通信的一方异常终止，而另一方并没有感知到，这时候如果另一方继续在原有的连接上发送数据，对端就会直接将链路重置。该场景可通过down掉一台主机的网卡，然后再将该主机上的服务重启来构造，实际的网络环境中几乎不会出现。

1.4 接收缓存不为空

当应用程序关闭一个socket时，如果该连接对应的TCP缓存中还有未接收的数据，此时TCP也会直接将连接重置，而不会通过正常的四次握手来关闭连接。我们可以通过代码2中的程序来验证。

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <netdb.h>
6  #include <sys/types.h>
7  #include <sys/socket.h>
8  #include <netinet/in.h>
9  #include <arpa/inet.h>
10
11 int main(int argc, char *argv[])
12 {
13     struct sockaddr_in srv_addr;
14     struct sockaddr_in cli_addr;
15
16     if (argc != 2)
17     {
18         fprintf(stderr, "usage: %s <port>\n", argv[0]);
19         return -1;
20     }
```

```
20 }
21
22 int fd = socket(AF_INET, SOCK_STREAM, 0);
23 if (fd < 0)
24 {
25     perror("open socket failed.");
26     return -1;
27 }
28
29 bzero((char *) &srv_addr, sizeof(srv_addr));
30
31 srv_addr.sin_family = AF_INET;
32 srv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
33 srv_addr.sin_port = htons((unsigned short)atoi(argv[1]));
34
35 if (bind(fd, (struct sockaddr *)&srv_addr, sizeof(srv_addr))
36     < 0)
37 {
38     perror("bind error.");
39     return -1;
40 }
41
42 if (listen(fd, 5) < 0)
43 {
44     perror("listen error.");
45     return -1;
46 }
47
48 socklen_t cli_len = sizeof(cli_addr);
49
50 int accepted_fd = accept(fd, (struct sockaddr *)&cli_addr, &
51     cli_len);
52 if (accepted_fd < 0)
53 {
54     perror("accept error.");
55     return -1;
56 }
57
58 sleep(20);
59 close(accepted_fd);
60
61 sleep(5);
62 close(fd);
63 return 0;
64 }
```

代码 2: TCP Server

代码2中的服务端在启动监听后，等待一个连接请求的到来并接受该连接，但不会在该连接上接收任何数据，休眠10秒后直接将连接关闭。

服务端启动后，使用telnet连接服务端并发送”Hello world.”，使用Wireshark抓包的结果如图3所示。

No.	Time	Source	Destination	Protocol	Length	Info
1	2014-07-16 23:45:14.494100000	127.0.0.1	127.0.0.1	TCP	74	51696 > ospf-lite [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK PERM=1 TSval=18262728 TSecr=18262728
2	2014-07-16 23:45:14.494129000	127.0.0.1	127.0.0.1	TCP	74	ospf-lite > 51696 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK PERM=1 TSval=18262728 TSecr=18262728
3	2014-07-16 23:45:14.494182000	127.0.0.1	127.0.0.1	TCP	66	51696 > ospf-lite [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=1826273 TSecr=18262728
4	2014-07-16 23:45:18.311506000	127.0.0.1	127.0.0.1	TCP	79	51696 > ospf-lite [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=13 TSval=1827228 TSecr=18262728
5	2014-07-16 23:45:18.311576000	127.0.0.1	127.0.0.1	TCP	66	ospf-lite > 51696 [ACK] Seq=1 Ack=14 Win=43776 Len=0 TSval=1827228 TSecr=1827228
6	2014-07-16 23:45:34.494439000	127.0.0.1	127.0.0.1	TCP	66	ospf-lite > 51696 [RST, ACK] Seq=1 Ack=14 Win=43776 Len=0 TSval=1831273 TSecr=1827228

图 3: RST

如上四种场景中的前三种在TCP/IP详解中都有详细说明，有兴趣的读者可再详细阅读该书中的相关章节。