tcp短连接TIME_WAIT问题解决方法大全（2）——SO_LINGER

2012年11月04日 21:48:23 阅读数：11186

SO_LINGER是一个socket选项，通过setsockopt API进行设置，使用起来比较简单，但其实现机制比较复杂，且字面意思上比较难理解。
解释最清楚的当属《Unix网络编程卷1》中的说明（7.5章节），这里简单摘录：
SO_LINGER的值用如下数据结构表示：

```
struct linger {
    int l_onoff; /* 0 = off, nozero = on */
    int l_linger; /* linger time */
};
```

其取值和处理如下：
1、设置 l_onoff为0，则该选项关闭，l_linger的值被忽略，等于内核缺省情况，close调用会立即返回给调用者，如果可能将会传输任何未发送的数据；
2、设置 l_onoff为非0，l_linger为0，则套接口关闭时TCP夭折连接，TCP将丢弃保留在套接口发送缓冲区中的任何数据并发送一个RST给对方，
　而不是通常的四分组终止序列，这避免了TIME_WAIT状态；
3、设置 l_onoff 为非0，l_linger为非0，当套接口关闭时内核将拖延一段时间（由l_linger决定）。
　如果套接口缓冲区中仍残留数据，进程将处于睡眠状态，直 到（a）所有数据发送完且被对方确认，之后进行正常的终止序列（描述字访问计数为0）
　或（b）延迟时间到。此种情况下，应用程序检查close的返回值是非常重要的，如果在数据发送完并被确认前时间到，close将返回EWOULDBLOCK错误且套接口发送缓冲区中的任何数据都丢失。
　close的成功返回仅告诉我们发送的数据（和FIN）已由对方TCP确认，它并不能告诉我们对方应用进程是否已读了数据。如果套接口设为非阻塞的，它将不等待close完成。

第一种情况其实和不设置没有区别，第二种情况可以用于避免TIME_WAIT状态，但在Linux上测试的时候，并未发现发送了RST选项，而是正常进行了四步关闭流程，
初步推断是“只有在丢弃数据的时候才发送RST”，如果没有丢弃数据，则走正常的关闭流程。
查看Linux源码，确实有这么一段注释和源码：

```
=====linux-2.6.37 net/ipv4/tcp.c 1915=====
/* As outlined in RFC 2525, section 2.17, we send a RST here because
* data was lost. To witness the awful effects of the old behavior of
* always doing a FIN, run an older 2.1.x kernel or 2.0.x, start a bulk
* GET in an FTP client, suspend the process, wait for the client to
* advertise a zero window, then kill -9 the FTP client, wheee...
* Note: timeout is always zero in such a case.
```

```
*/
if (data_was_unread) {
/* Unread data was tossed, zap the connection. */
NET_INC_STATS_USER(sock_net(sk), LINUX_MIB_TCPABORTONCLOSE);
tcp_set_state(sk, TCP_CLOSE);
tcp_send_active_reset(sk, sk->sk_allocation);
}
```

另外，从原理上来说，这个选项有一定的危险性，可能导致丢数据，使用的时候要小心一些，但我们在实测libmemcached的过程中，没有发现此类现象，
应该是和libmemcached的通讯协议设置有关，也可能是我们的压力不够大，不会出现这种情况。

第三种情况其实就是第一种和第二种的折中处理，且当socket为非阻塞的场景下是没有作用的。
对于应对短连接导致的大量TIME_WAIT连接问题，个人认为第二种处理是最优的选择，libmemcached就是采用这种方式，
从实测情况来看，打开这个选项后，TIME_WAIT连接数为0，且不受网络组网（例如是否虚拟机等）的影响。