1. timeout时间

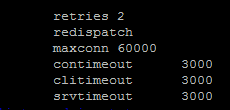
统一设为3s

retries 2

contimeout 3000

clitimeout 3000

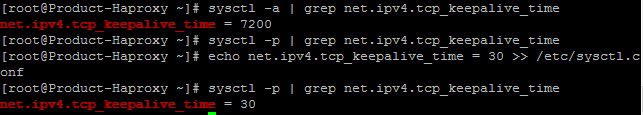
srvtimeout 3000



1. net.ipv4.tcp\_keepalive\_time

centos7默认是7200，改为30s

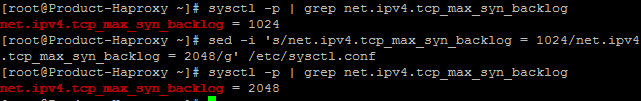
net.ipv4.tcp\_keepalive\_time = 30



1. net.ipv4.tcp\_max\_syn\_backlog

centos7默认是1024，改为2048

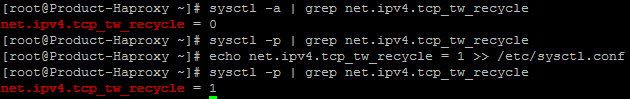
net.ipv4.tcp\_max\_syn\_backlog = 2048



1. net.ipv4.tcp\_tw\_recycle

centos7默认是0，关闭，要开启，开启TCP连接中TIME-WAIT sockets的快速回收

**net.ipv4.tcp\_tw\_recycle = 1**



tcp\_tw\_recycle：顾名思义就是回收TIME\_WAIT连接。可以说这个内核参数已经变成了大众处理TIME\_WAIT的万金油，如果你在网络上搜索TIME\_WAIT的解决方案，十有八九会推荐设置它，不过这里隐藏着一个不易察觉的陷阱：

当多个客户端通过NAT方式联网并与服务端交互时，服务端看到的是同一个IP，也就是说对服务端而言这些客户端实际上等同于一个，可惜由于这些客户端的时间戳可能存在差异，于是乎从服务端的视角看，便可能出现时间戳错乱的现象，进而直接导致时间戳小的数据包被丢弃。参考：tcp\_tw\_recycle和tcp\_timestamps导致connect失败问题。

tcp\_tw\_reuse：顾名思义就是复用TIME\_WAIT连接。当创建新连接的时候，如果可能的话会考虑复用相应的TIME\_WAIT连接。通常认为「tcp\_tw\_reuse」比「tcp\_tw\_recycle」安全一些，这是因为一来TIME\_WAIT创建时间必须超过一秒才可能会被复用；二来只有连接的时间戳是递增的时候才会被复用。官方文档里是这样说的：如果从协议视角看它是安全的，那么就可以使用。这简直就是外交辞令啊！按我的看法，如果网络比较稳定，比如都是内网连接，那么就可以尝试使用。

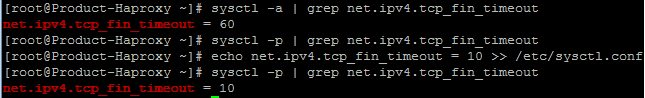
不过需要注意的是在哪里使用，既然我们要复用连接，那么当然应该在连接的发起方使用，而不能在被连接方使用。举例来说：客户端向服务端发起HTTP请求，服务端响应后主动关闭连接，于是TIME\_WAIT便留在了服务端，此类情况使用「tcp\_tw\_reuse」是无效的，因为服务端是被连接方，所以不存在复用连接一说。让我们延伸一点来看，比如说服务端是PHP，它查询另一个MySQL服务端，然后主动断开连接，于是TIME\_WAIT就落在了PHP一侧，此类情况下使用「tcp\_tw\_reuse」是有效的，因为此时PHP相对于MySQL而言是客户端，它是连接的发起方，所以可以复用连接。

说明：如果使用tcp\_tw\_reuse，请激活tcp\_timestamps，否则无效。

1. net.ipv4.tcp\_fin\_timeout

Centos7默认是60s，改为10s，表示如果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2状态的时间

net.ipv4.tcp\_fin\_timeout = 10



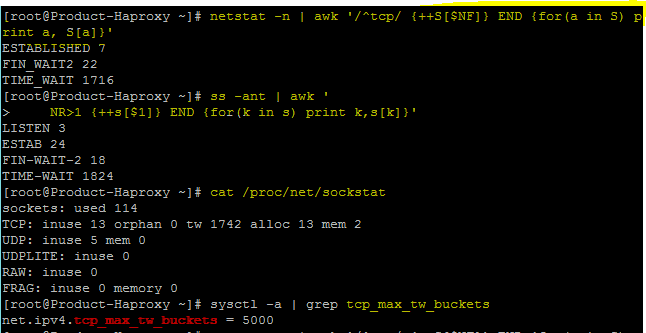
1. tcp\_max\_tw\_buckets

centos7默认是5000，

cat /proc/net/sockstat可以直接查看TCP连接状态

netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

ss -ant | awk 'NR>1 {++s[$1]} END {for(k in s) print k,s[k]}'



tcp\_max\_tw\_buckets：顾名思义就是控制TIME\_WAIT总数。官网文档说这个选项只是为了阻止一些简单的DoS攻击，平常不要人为的降低它。如果缩小了它，那么系统会将多余的TIME\_WAIT删除掉，日志里会显示：「TCP: time wait bucket table overflow」。

需要提醒大家的是物极必反，曾经看到有人把「tcp\_max\_tw\_buckets」设置成0，也就是说完全抛弃TIME\_WAIT，这就有些冒险了，用一句围棋谚语来说：入界宜缓。

…

有时候，如果我们换个角度去看问题，往往能得到四两拨千斤的效果。前面提到的例子：客户端向服务端发起HTTP请求，服务端响应后主动关闭连接，于是TIME\_WAIT便留在了服务端。这里的关键在于主动关闭连接的是服务端！在关闭TCP连接的时候，先出手的一方注定逃不开TIME\_WAIT的宿命，套用一句歌词：把我的悲伤留给自己，你的美丽让你带走。如果客户端可控的话，那么在服务端打开KeepAlive，尽可能不让服务端主动关闭连接，而让客户端主动关闭连接，如此一来问题便迎刃而解了。