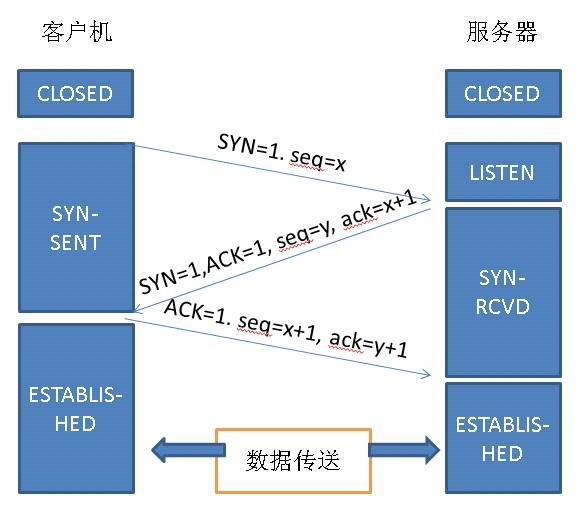
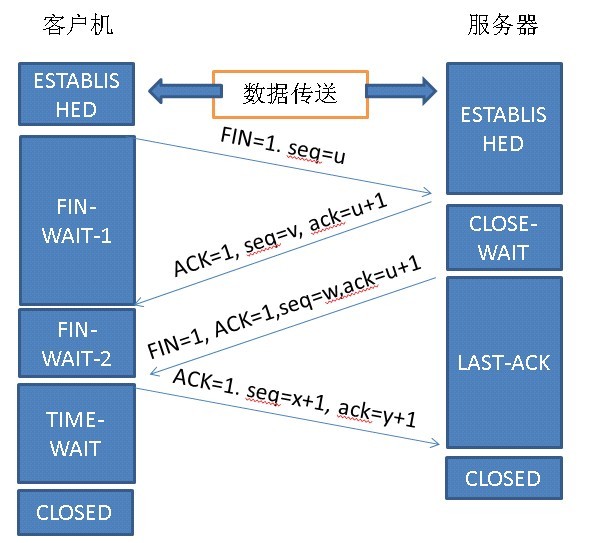
Nginx反向代理tomcat，两端TIME\_WAIT优化

发表于 2016-02-27   |   分类于 [nginx](https://lanjingling.github.io/categories/nginx/)  |

**一、tcp三次握手和time wait**

第一次握手：建立连接时，客户端发送syn包和一个随机序列号seq=x到服务器，并进入SYN\_SEND状态，等待服务器进行确认。（syn，同 步序列编号）。第二次握手，服务器收到syn包，必须确认客户的SYN，然后服务器发送一个ACK=1, SYN=1, seq=y的随机数和ack=x+1的确认数的包发送回去。第三次握手是客户端收到服务器端的SYN+ACK包，然后向服务器端发送确认包 ack=y+1, seq=x+1, ACK=1,客户端和服务器端进入ESTABLISHED状态，完成三次握手。具体图示如下：  
[](http://ww4.sinaimg.cn/mw690/72c913fbjw1f1dq3ga4b0j20g50en3zu.jpg)  
这里多说一点，既然提到了连接时的三次握手，就顺便把断开连接时的四次挥手也复习一下。首先客户端主动发送Fin=1，seq=u，它等于前面已传送过去的最后一个字节的序号加1，这是A进入FIN-WAIT-1状态，等待B的确认。B收到连接后立即发出确认，确认号是ack=u+1，而这个报文段 自己的序号是v，等于B前面已传送过的数据的最后一个字节的序号加1，然后B即进入CLOSE-WAIT状态。因而A到B的这个链接现在已经断开了，这时 的TCP连接处于半关闭状态，即A已经没有数据需要发送了。但B若发送数据，A还是要接受的。A收到来自B的确认之后就进入了FIN-WAIT-2状态等待B发出连接释放报文段。若B已经没有要向A发送数据，其应用进程就通知TCP释放连接。这时B发出的连接释放报文段必须使用FIN=1，现在假定B的序 号为w，B还必须重复上次已发送过的确认号ack=u+1，这时B就进入了LAST-ACK状态，等待A确认。A在收到B的连接释放之后必须对此发出确 认，在确认号中把ACK置1，确认号ack=w+1，而自己的序号是seq=u+1，接着A进入TIME-WAIT状态。为了保证B可以收到确认释放报文 段。如下图：  
[](http://ww4.sinaimg.cn/mw690/72c913fbjw1f1dq4482tvj20gi0f375y.jpg)

是不是所有执行主动关闭的socket都会进入TIME\_WAIT状态呢？有没有什么情况使主动关闭的socket直接进入CLOSED状态呢？

主动关闭的一方在发送最后一个 ack 后就会进入 TIME\_WAIT 状态 停留2MSL（max segment lifetime）时间，这个是TCP/IP必不可少的，也就是“解决”不了的。也就是TCP/IP设计者本来是这么设计的，主要有两个原因：  
• 防止上一次连接中的包，迷路后重新出现，影响新连接（经过2MSL，上一次连接中所有的重复包都会消失）  
• 可靠的关闭TCP连接，在主动关闭方发送的最后一个 ack(fin) ，有可能丢失，这时被动方会重新发fin, 如果这时主动方处于 CLOSED 状态 ，就会响应 rst 而不是 ack。所以主动方要处于 TIME\_WAIT 状态，而不能是 CLOSED 。  
  
*小结：*  
  
- 因为TCP连接是双向的，所以在关闭连接的时候，两个方向各自都需要关闭。先发FIN包的一方执行的是主动关闭；后发FIN包的一方执行的是被动关闭。主动关闭的一方会进入TIME\_WAIT状态，并且在此状态停留两倍的MSL时长（指的是报文段的最大生存时间）。  
- TIME\_WAIT产生原因：为什么主动关闭的一端不直接进入closed状态，而是要先进入time\_wait并且停留两倍的MSL时长呢？这是因为TCP是建立在不可靠网络上的可靠协议。如果主动关闭的一端收到被动关闭一端的发出的FIN包后，返回ACK包，同时进入TIME\_WAIT，但是由于网络的原因，主动关闭一端发送的ACK包可能会延迟，从而触发被动关闭一方重传FIN包，这样一来一回极端情况正好是2MSL。如果主动关闭的一端直接close或者不到两倍MSL时间就关闭，那么被动关闭发出重传FIN包到达，可能出现的问题是：旧的连接不存在，系统只能返回RST包；新的TCP连接已经建立，延迟包可能会干扰新连接。这都可能导致TCP不可靠。  
- 在生产过程中，如果服务器使用短连接，那么完成一次请求后会主动断开连接，就会造成大量time\_wait状态。因此我们常常在系统中会采用长连接，减少建立连接的消耗，同时也减少TIME\_WAIT的产生，但实际上即使使用长连接配置不当时，当TIME\_WAIT的生产速度远大于其消耗速度时，系统仍然会累计大量的TIME\_WAIT状态的连接。TIME\_WAIT状态连接过多就会造成一些问题。如果客户端的TIME\_WAIT连接过多，同时它还在不断产生，将会导致客户端端口耗尽，新的端口分配不出来，出现错误。如果服务器端的TIME\_WAIT连接过多，可能会导致客户端的请求连接失败。

**二、nginx反向代理应用服务器，导致后端应用服务器TIME\_WAIT过多问题：**

1、使用下面命令可以查看服务器的连接情况：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | shell> netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}' TIME\_WAIT 250263 CLOSE\_WAIT 57 FIN\_WAIT2 3 ESTABLISHED 2463 SYN\_RECV 8 或者 ss -ant | awk 'NR>1 {++s[$1]} END {for(k in s) print k,s[k]}' |

TIME\_WAIT危害：

* 如果客户端的TIME\_WAIT连接过多，同时它还在不断产生，将会导致客户端端口耗尽，新的端口分配不出来，出现错误；
* 如果服务器端的TIME\_WAIT连接过多，可能会导致客户端的请求连接失败。

2、案例一：nginx端出现大量TIME\_WAIT：

将nginx作为反向代理时，后连tomcat等服务器。测试中不同并发压力下多次、反复出现nginx服务器端口资源耗尽的问题。表现为nginx服务器出现大量time\_wait状态连接，端口资源耗尽（nginx报错：cannot assign requested address ）。首先检查nginx开启了长连接keepalive，但是系统仍然出现了大量的TIME\_WAIT，这就和之前提到的当系统产生TIME\_WAIT的速度大于其消耗速度时，就会累计TIME\_WAIT。原因是：keepalive取配置太小，将其增大后问题得以解决。(PS:nginx总的keepalive连接池大小 = keepalive取值 \* nginx worker数）

*注：*从nginx 1.1.4 开始有了原生的ngx\_http\_upstream\_keepalive 模块；

2、案例二：tomcat端出现大量TIME\_WAIT：  
Nginx作为反向代理，长连接配置主要有三项，upstream中的keepalive设置单个worker最大请求数，参数proxy\_http\_version 1.1强制转换为http1.1协议(默认支持长连接)，proxy\_set\_header Connection将请求头部connection为空(http1.0请求默认connection头部为close)。如下

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 | http {  include mime.types;  default\_type application/octet-stream;   sendfile on;  #tcp\_nopush on;  #gzip on;   keepalive\_timeout 65;#默认0，长连接的持续时间   include conf.d/\*.conf; }  upstream http\_backend {  server 127.0.0.1:8080;   keepalive 16;#连接池大小 }  server {  ...   location /http/ {  proxy\_pass http://http\_backend;  proxy\_http\_version 1.1;  proxy\_set\_header Connection "";  ...  } } |

但是我们不禁会尝试疑问TIME\_WAIT出现在Tomcat而不是在Nginx上？从抓包可以看出Nginx发送给Tomcat包头部Connection为close，所以Tomcat在处理完head请求后就主动关闭，所以TIME\_WAIT出现在Tomcat服务器。

3、可以修改系统的/etc/sysctl.conf配置来减少TIME\_WAIT的tcp连接：

|  |  |
| --- | --- |
| 1 2 3 4 5 | vi /etc/sysctl.conf net.ipv4.tcp\_syncookies = 1 （某些情况下该参数已启用） net.ipv4.tcp\_tw\_reuse = 1 net.ipv4.tcp\_tw\_recycle = 1 net.ipv4.tcp\_fin\_timeout = 30 |

然后执行/sbin/sysctl -p让参数生效。再用命令查看TIME\_WAIT连接数 netstat -ae | grep “TIME\_WAIT” |wc -l 发现大量的TIME\_WAIT 已不存在。

* net.ipv4.tcp\_syncookies = 1表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；
* net.ipv4.tcp\_tw\_reuse = 1表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；
* net.ipv4.tcp\_tw\_recycle = 1表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。
* net.ipv4.tcp\_fin\_timeout修改系統默认的TIMEOUT时间