又见KeepAlive

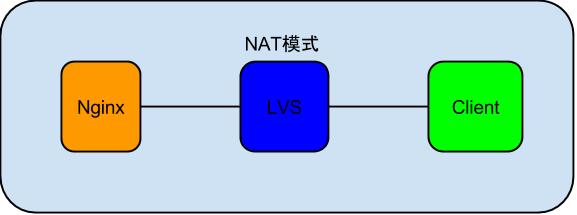
http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif (2014-07-08 11:24:53)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

|  |  |
| --- | --- |
| 标签：  [keepalive](http://search.sina.com.cn/?c=blog&q=keepalive&by=tag)    [tcp](http://search.sina.com.cn/?c=blog&q=tcp&by=tag)    [阿里技术保障](http://search.sina.com.cn/?c=blog&q=%B0%A2%C0%EF%BC%BC%CA%F5%B1%A3%D5%CF&by=tag)    [linux](http://search.sina.com.cn/?c=blog&q=linux&by=tag) |  |

**我为什么要谈KeepAlive**

最近工作中遇到一个问题，想把它记录下来，场景是这样的：

[](http://img2.tbcdn.cn/L1/461/1/4319379ee686ea33f3c637c962917949ded85667)

从上图可以看出，用户通过Client访问的是LVS的VIP， VIP后端挂载的RealServer是Nginx服务器。 Client可以是浏览器也可以是一个客户端程序。一般情况下，这种架构不会出现问题，但是如果Client端把请求发送给Nginx，Nginx的后端需要一段时间才能返回结果，超过1分30秒就会有问题，使用LVS作为负载均衡设备看到的现象就是1分30秒之后， Client和Nginx链接被断开，没有数据返回。原因是LVS默认保持TCP的Session为90s，超过90s没有TCP报文在链接上传输，LVS就会给两端发送RESET报文断开链接。LVS这么做的原因相信大家都知道一二，我所知道的原因主要有两点：

1.节省负载均衡设备资源，每一个TCP/UDP的链接都会在负载均衡设备上创建一个Session的结构， 链接如果一直不断开，这种Session结构信息最终会消耗掉所有的资源，所以必须释放掉。 2.另外释放掉能保护后端的资源，如果攻击者通过空链接，链接到Nginx上，如果Nginx没有做合适 的保护，Nginx会因为链接数过多而无法提供服务。

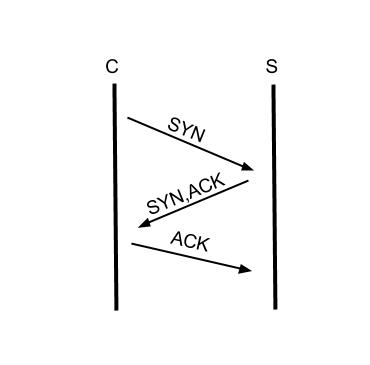
这种问题不只是在LVS上有，之前在商用负载均衡设备F5上遇到过同样的问题，F5的Session断开方式和LVS有点区别，F5不会主动发送RESET给链接的两端，Session消失之后，当链接中一方再次发送报文时会接收到F5的RESET, 之后的现象是再次发送报文的一端TCP链接状态已经断开，而另外一端却还是ESTABLISH状态。

知道是负载均衡设备原因之后，第一反应就是通过开启KeepAlive来解决。到此这个问题应该是结束了，但是我发现过一段时间总又有人提起KeepAlive的问题，甚至发现由于KeepAlive的理解不正确浪费了很多资源，原本能使用LVS的应用放在了公网下沉区，或者换成了商用F5设备(F5设备的Session断开时间要长一点，默认应该是5分钟)。所以我决定把我知道的KeepAlive知识点写篇博客分享出来。

**为什么要有KeepAlive？**

在谈KeepAlive之前，我们先来了解下简单TCP知识(知识很简单，高手直接忽略)。首先要明确的是在TCP层是没有“请求”一说的，经常听到在TCP层发送一个请求，这种说法是错误的。TCP是一种通信的方式，“请求”一词是事务上的概念，HTTP协议是一种事务协议，如果说发送一个HTTP请求，这种说法就没有问题。也经常听到面试官反馈有些面试运维的同学，基本的TCP三次握手的概念不清楚，面试官问TCP是如何建立链接，面试者上来就说，假如我是客户端我发送一个请求给服务端，服务端发送一个请求给我。。。这种一听就知道对TCP基本概念不清楚。下面是我通过wireshark抓取的一个TCP建立握手的过程。（命令行基本上用TCPdump,后面我们还会用这张图说明问题）:

[](http://img4.tbcdn.cn/L1/461/1/56c37463b083017ca6d1054965f76d13097427bf)  
现在我看只要看前3行，这就是TCP三次握手的完整建立过程，第一个报文SYN从发起方发出，第二个报文SYN,ACK是从被连接方发出，第三个报文ACK确认对方的SYN，ACK已经收到，如下图：

[](http://img2.tbcdn.cn/L1/461/1/82dae438b29274d623a495ca8395d5aa769cc9c0)

但是数据实际上并没有传输，请求是有数据的，第四个报文才是数据传输开始的过程，细心的读者应该能够发现wireshark把第四个报文解析成HTTP协议，HTTP协议的GET方法和URI也解析出来，所以说TCP层是没有请求的概念，HTTP协议是事务性协议才有请求的概念，TCP报文承载HTTP协议的请求(Request)和响应(Response)。

现在才是开始说明为什么要有KeepAlive。链接建立之后，如果应用程序或者上层协议一直不发送数据，或者隔很长时间才发送一次数据，当链接很久没有数据报文传输时如何去确定对方还在线，到底是掉线了还是确实没有数据传输，链接还需不需要保持，这种情况在TCP协议设计中是需要考虑到的。TCP协议通过一种巧妙的方式去解决这个问题，当超过一段时间之后，TCP自动发送一个数据为空的报文给对方，如果对方回应了这个报文，说明对方还在线，链接可以继续保持，如果对方没有报文返回，并且重试了多次之后则认为链接丢失，没有必要保持链接。

**如何开启KeepAlive**

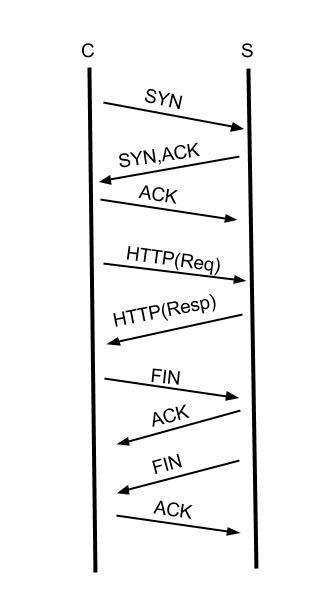
KeepAlive并不是默认开启的，在Linux系统上没有一个全局的选项去开启TCP的KeepAlive。需要开启KeepAlive的应用必须在TCP的socket中单独开启。Linux Kernel有三个选项影响到KeepAlive的行为：  
1.net.ipv4.tcp*keepalive*intvl = 75  
2.net.ipv4.tcp*keepalive*probes = 9  
3.net.ipv4.tcp*keepalive*time = 7200  
tcp*keepalive*time的单位是秒，表示TCP链接在多少秒之后没有数据报文传输启动探测报文; tcp*keepalive*intvl单位是也秒,表示前一个探测报文和后一个探测报文之间的时间间隔，tcp*keepalive*probes表示探测的次数。

TCP socket也有三个选项和内核对应，通过setsockopt系统调用针对单独的socket进行设置：  
TCP*KEEPCNT: 覆盖 tcp*keepalive*probes  
TCP*KEEPIDLE: 覆盖 tcp*keepalive*time  
TCP*KEEPINTVL: 覆盖 tcp*keepalive\_intvl

举个例子，以我的系统默认设置为例，kernel默认设置的tcp*keepalive*time是7200s, 如果我在应用程序中针对socket开启了KeepAlive,然后设置的TCP\_KEEPIDLE为60，那么TCP协议栈在发现TCP链接空闲了60s没有数据传输的时候就会发送第一个探测报文。

**TCP KeepAlive和HTTP的Keep-Alive是一样的吗？**

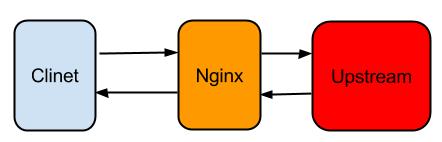
估计很多人乍看下这个问题才发现其实经常说的KeepAlive不是这么回事，实际上在没有特指是TCP还是HTTP层的KeepAlive，不能混为一谈。TCP的KeepAlive和HTTP的Keep-Alive是完全不同的概念。TCP层的KeepAlive上面已经解释过了。 HTTP层的Keep-Alive是什么概念呢？ 在讲述TCP链接建立的时候，我画了一张三次握手的示意图，TCP在建立链接之后， HTTP协议使用TCP传输HTTP协议的请求(Request)和响应(Response)数据，一次完整的HTTP事务如下图：

[](http://img3.tbcdn.cn/L1/461/1/1c4bbc41628604483760e2f8a89d38cfaf0dbe9d)

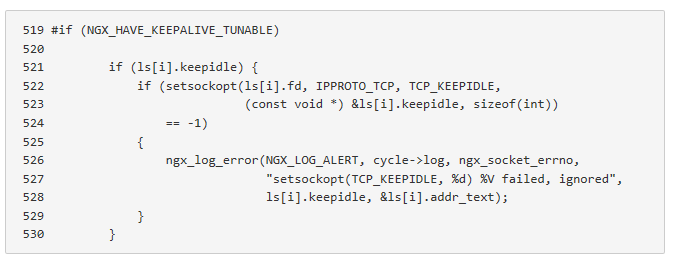
各位看官请注意，这张图我简化了HTTP(Req)和HTTP(Resp)，实际上的请求和响应需要多个TCP报文。从图中可以发现一个完整的HTTP事务，有链接的建立，请求的发送，响应接收，断开链接这四个过程,早期通过HTTP协议传输的数据以文本为主，一个请求可能就把所有要返回的数据取到，但是，现在要展现一张完整的页面需要很多个请求才能完成，如图片,JS,CSS等，如果每一个HTTP请求都需要新建并断开一个TCP，这个开销是完全没有必要的，开启HTTP Keep-Alive之后，能复用已有的TCP链接，当前一个请求已经响应完毕，服务器端没有立即关闭TCP链接，而是等待一段时间接收浏览器端可能发送过来的第二个请求，通常浏览器在第一个请求返回之后会立即发送第二个请求，如果某一时刻只能有一个链接，同一个TCP链接处理的请求越多，开启KeepAlive能节省的TCP建立和关闭的消耗就越多。当然通常会启用多个链接去从服务器器上请求资源，但是开启了Keep-Alive之后，仍然能加快资源的加载速度。HTTP/1.1之后默认开启Keep-Alive, 在HTTP的头域中增加Connection选项。当设置为Connection:keep-alive表示开启，设置为Connection:close表示关闭。实际上HTTP的KeepAlive写法是Keep-Alive，跟TCP的KeepAlive写法上也有不同。所以TCP KeepAlive和HTTP的Keep-Alive不是同一回事情。

**Nginx的TCP KeepAlive如何设置**

开篇提到我最近遇到的问题，Client发送一个请求到Nginx服务端，服务端需要经过一段时间的计算才会返回， 时间超过了LVS Session保持的90s,在服务端使用Tcpdump抓包,本地通过wireshark分析显示的结果如第二副图所示，第5条报文和最后一条报文之间的时间戳大概差了90s。在确定是LVS的Session保持时间到期的问题之后，我开始在寻找Nginx的TCP KeepAlive如何设置，最先找到的选项是keepalive*timeout,从同事那里得知keepalive*timeout的用法是当keepalive*timeout的值为0时表示关闭keepalive,当keepalive*timeout的值为一个正整数值时表示链接保持多少秒，于是把keepalive*timeout设置成75s,但是实际的测试结果表明并不生效。显然keepalive*timeout不能解决TCP层面的KeepAlive问题，实际上Nginx涉及到keepalive的选项还不少，Nginx通常的使用方式如下：

[](http://img2.tbcdn.cn/L1/461/1/41348f9b06861246b08800932aa5513cf3b59217)

从TCP层面Nginx不仅要和Client关心KeepAlive,而且还要和Upstream关心KeepAlive, 同时从HTTP协议层面，Nginx需要和Client关心Keep-Alive,如果Upstream使用的HTTP协议，还要关心和Upstream的Keep-Alive，总而言之，还比较复杂。所以搞清楚TCP层的KeepAlive和HTTP的Keep-Alive之后，就不会对于Nginx的KeepAlive设置错。我当时解决这个问题时候不确定Nginx有配置TCP keepAlive的选项，于是我打开Ngnix的源代码，在源代码里面搜索TCP\_KEEPIDLE,相关的代码如下：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=e59371cc0102ux5w&url=http://album.sina.com.cn/pic/004cF6UIgy6KhIb8WuZ99)

从代码的上下文我发现TCP KeepAlive可以配置，所以我接着查找通过哪个选项配置，最后发现listen指令的so\_keepalive选项能对TCP socket进行KeepAlive的配置。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=e59371cc0102ux5w&url=http://album.sina.com.cn/pic/004cF6UIgy6KhIdwEJY52)

以上三个参数只能使用一个，不能同时使用， 比如so*keepalive=on, so*keepalive=off或者so*keepalive=30s::(表示等待30s没有数据报文发送探测报文)。通过设置listen 80,so*keepalive=60s::之后成功解决Nginx在LVS保持长链接的问题，避免了使用其他高成本的方案。在商用负载设备上如果遇到类似的问题同样也可以通过这种方式解决。

**参考资料**

《TCP/IP协议详解VOL1》--强烈建议对于网络基本知识不清楚同学有空去看下。

<http://tldp.org/HOWTO/html_single/TCP-Keepalive-HOWTO/#overview>

<http://nginx.org/en/docs/http/ngx_http_core_module.html>

Nginx Source code: <https://github.com/alibaba/tengine>

作者：李成栋阿里技术保障 架构工具资深工程师

          个人微博 Bryton的岁月<http://weibo.com/u/2474674454>