Nginx 400 Bad Request

 我说什么你都别信 于 2021-04-08 15:31:55 发布 6404 收藏 4

400 Bad Request是一种HTTP错误状态码。HTTP/1.1对400 Bad Request的定义主要是：1、语义有误，当前请求无法被服务器理解。除非进行修改，否则客户端不应该重复提交这个请求。2、请求参数有误。
在这段时间笔者遇到了好几次生产问题Nginx报400异常，且原因细究下来各不相同，有些甚至在网上没有搜到类似案例。遂产生了兴趣，做了本次梳理，希望会对大家有一定帮助！

1. 一般导致400异常的场景

一般使用Nginx在以下场景会报400 Bad Request：

1.1 请求头过大

nginx 400 Bad request是request header过大所引起，request过大，通常是由于cookie中写入了较大的值所引起。在nginx.conf中，调整client_header_buffer_size和large_client_header_buffer参数大小可以解决问题（ps，网上很多博客只片面强调调大两个参数的值，并未研究参数区别和用法，真正大规模应用明显是不合适的）。
那么这两个参数是如何定义的呢?

```
Syntax: client_header_buffer_size size;
Default: client_header_buffer_size 1k;
Context: http, server
```

设置用于读取客户端请求头的缓冲区大小。对于多数请求缓冲1K字节是足够的。然而，如果一个请求包括Cookie，或来自一个WAP客户端，它可能远不止1K。如果request line或者request header超过1K，则由large_client_header_buffers指令配置分配。

```
Syntax: large_client_header_buffers number size;
Default: large_client_header_buffers 4 8k;
```

```
Context: http, server
```

设置用于读取大型客户端请求头的缓冲区的最大数量和大小。request line不能超过一个缓冲区的大小，否则将返回414（请求URI太大）错误给客户端。request header不能超过一个缓冲区的大小，否则将返回400（错误请求）错误给客户端。缓冲区只能按需分配。默认情况下，缓冲区的大小为8K字节。如果一个连接请求处理结束后转变为保持状态，这些缓冲区被释放。
所以nginx处理header的方法是：

先处理请求的request_line，之后才是request_header。
这两者的buffer分配策略相同。
先根据client_header_buffer_size配置的值分配一个buffer，如果分配的buffer无法容纳 request_line/request_header，那么就会再次根据large_client_header_buffers配置的参数分配large_buffer，如果large_buffer还是无法容纳，那么就会返回414（处理request_line）/400（处理request_header）错误。
综上所述，网上的很多调整是不合适的。要按具体业务需求调整参数大小。

如果你的请求中的header都很大，那么应该使用client_header_buffer_size，这样能减少一次内存分配。
如果你的请求中只有少量请求header很大，那么应该使用large_client_header_buffers，因为这样就仅需在处理大header时才会分配更多的空间，从而减少无谓的内存空间浪费。
1.1 空请求
0.7.12以前版本的nginx收到一个空请求，nginx不会去与任何虚拟主机匹配，直接返回400错误，之后的新版本nginx可以用server_name _;匹配空请求头来处理。

```
server {
  listen 80 default_server;
  server_name _;
  return 404;
  access_log off;
}
```

当然以上情况在网上都很普遍，下面是本文重点想说的两种特殊场景。

2. 特殊问题场景一：URLConnection发起HTTPS请求经过代理400异常
我们有某个系统A，在和系统B之间通信时使用的是HTTPS协议，在系统B的nginx代理层有大量400错误抛出。经过我们排查，不存在上述请求头部过大或者空请求的情况。那么400问题因何而起呢？我们根据问题症状浏览了大量国外技术网站，终于定位到这是JDK的一个bug：

```
JDK-6687282 : URLConnection for HTTPS connection through Proxy w/ Digest Authentication gives 400 Bad Request
```

A系统的JDK版本比较老java version "1.6.0_05"，同时使用的是JDK原生的 URLConnection，在通信的过程中我们是这样做的：

1.- Create an HTTPS URL.
2.- Obtain an URLConnection with url.openConnection() providing a Proxy
3.- Setup the SSLSocketFactory.
4.- Setup the default Authenticator.
5.- Read result from connection

问题出现在身份验证上，身份验证在生成响应hash时使用request-URI作为其算法的一部分。request-URI通常取的是uri的绝对路径（abs_path ）。但也不全是，比如，在建立隧道请求（tunneling）时，需要用主机+端口号作为request-URI，例如。

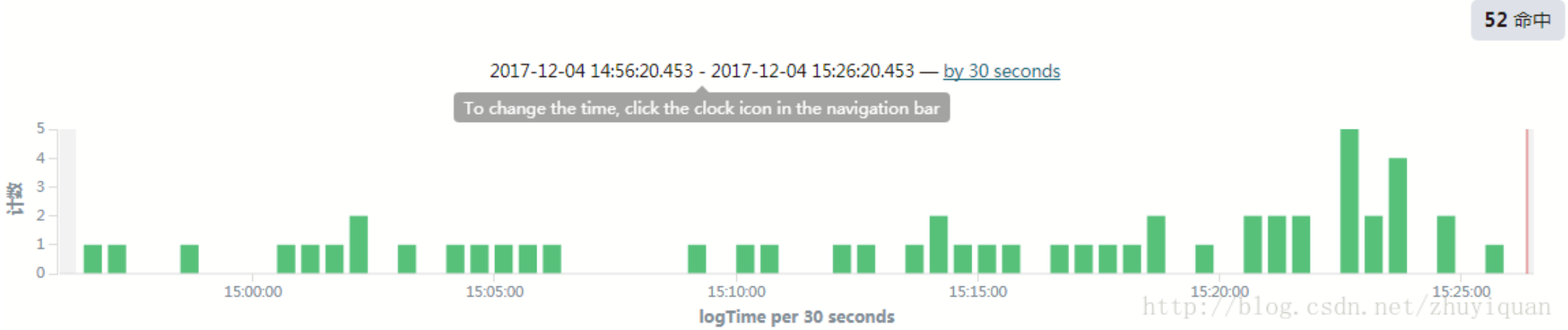
```
"CONNECT verisign.com:443 HTTP/1.1"
```

而sun.net.www.protocol.http.digestauthentication 只考虑了通过绝对路径作为request-URI的场景（abs_path ）。这显然是有问题的，在建立隧道时，需要使用主机+端口号方式。所以导致在身份验证时产生400 Bad Request：

1.- Send CONNECT HTTP Request to the Proxy
2.- Receive a 407 Proxy Authentication Required
3.- Send new CONNECT with authentication credentials.
4.- Receive 400 Bad Request

解决问题的方案有多种：
1.升级JDK版本，如下图

2.使用Apache HttpClient代替URLConnection



3. 特殊问题场景二：网络传输丢包导致的400异常
这个场景困扰我们时间比较长，某个和交易相关的核心系统每天有0.01~0.03%的400错误，全部都是用户端过来的post请求。

如上图，nginx有零星的400错误，然而tcp dump抓包发现并没有对应的400包。潜意识里我们认为所有的400 Bad Request不可能不被tcp dump所记录，所以我们纠结了很久。最终发现，是因为客户端Post请求Packet在网络传输过程中部分丢失导致到服务端无法正常响应，客户端30s超时断开连接，这时候nginx记录了400。这种情况下，nginx实际未反馈400的response，只是在连接断开时记录了400的日志。
如下图所示，

	Time	Source	Destination	Protocol	Length	Info	
1	0.000000000	45.115.164.210	[redacted]	TCP	118	33140→80 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=256	三次握手
2	0.000003870	[redacted]	45.115.164.210	TCP	112	80→33140 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460	
3	0.003818960	45.115.164.210	[redacted]	TCP	106	33140→80 [ACK] Seq=1 Ack=1 Win=14600 Len=0	
4	0.003896425	45.115.164.210	[redacted]	TCP	1503	[TCP segment of a reassembled PDU]	客户端POST请求载荷，缺失一个Packet
5	0.003948741	45.115.164.210	[redacted]	TCP	1406	[TCP segment of a reassembled PDU]	
6	0.004012738	[redacted]	45.115.164.210	TCP	106	80→33140 [ACK] Seq=1 Ack=1398 Win=33580 Len=0	服务端Ack，但因为请求报文不全，无法正常响应
7	0.004167848	[redacted]	45.115.164.210	TCP	106	80→33140 [ACK] Seq=1 Ack=2698 Win=36500 Len=0	
8	29.985857855	45.115.164.210	[redacted]	TCP	106	33140→80 [FIN, ACK] Seq=2698 Ack=1 Win=14600 Len=0	
9	29.985969013	[redacted]	45.115.164.210	TCP	112	80→33140 [RST, ACK] Seq=1 Ack=2699 Win=9203 Len=0	客户端30s超时，连接断开

data=%7B%22cityId%22%3A%22631%22%2C%22channelId%22%3A%2232%22%2C%22mainProductList%22%3A%5B%7B%22itemId
%22%3A%2257322517745485668 2%22%2C%22commdtyCode%22%3A%2200000000001427685 19%22%2C%22businessSign%22%3A
%220%22%2C%22commdtyType%22%3A%22%22%2C%22activeType%22%3A%2201%22%2C%22subActiveType%22%3A%22%22%2C
%22productListPrice%22%3A%2231.90%22%2C%22storeId%22%3A%220070062522%22%2C%22productQty%22%3A%221%22%7D%2C
%7B%22itemId%22%3A%2257322517745185665 0%22%2C%22commdtyCode%22%3A%2200000000001835849 94%22%2C
%22businessSign%22%3A%220%22%2C%22commdtyType%22%3A%22%22%2C%22activeType%22%3A%2201%22%2C%22subActiveType
%22%3A%22%22%2C%22productListPrice%22%3A%2216.90%22%2C%22storeId%22%3A%220070062522%22%2C%22productQty
%22%3A%221%22%7D%2C%7B%22itemId%22%3A%2257322517745185663 8%22%2C%22commdtyCode%22%3A
%2200000000001877854 35%22%2C%22businessSign%22%3A%220%22%2C%22commdtyType%22%3A%22%22%2C%22activeType%22%3A
%2201%22%2C%22subActiveType%22%3A%22%22%2C%22productListPrice%22%3A%2222.90%22%2C%22storeId%22%3A
%220070062522%22%2C%22productQty%22%3A%221%22%7D%2C%7B%22itemId%22%3A%2257322517745085662 8%22%2C
%22commdtyCode%22%3A%2200000000001387581 56%22%2C%22businessSign%22%3A%220%22%2C%22commdtyType%22%3A
%22%22%2C%22activeType%22%3A%2201%22%2C%22subActiveType%22%3A%22%22%2C%22productListPrice%22%3A
%2217.90%22%2C

POST报文请求缺失一部分

将上述报文转码后可以更直观发现，POST请求报文不全：


```
data={"cityId":"631","channelId":"32","mainProductList":
[{"itemId":"573225177454856682","commdtyCode":"000000000142768519","b
usinessSign":"0","commdtyType":"","activeType":"01","subActiveType":"
","productListPrice":"31.90","storeId":"0070062522","productQty":"1"}
,
{"itemId":"573225177451856650","commdtyCode":"000000000183584994","bu
sinessSign":"0","commdtyType":"","activeType":"01","subActiveType":""
,"productListPrice":"16.90","storeId":"0070062522","productQty":"1"},
{"itemId":"573225177451856638","commdtyCode":"000000000187785435","bu
sinessSign":"0","commdtyType":"","activeType":"01","subActiveType":""
,"productListPrice":"22.90","storeId":"0070062522","productQty":"1"},
{"itemId":"573225177450856628","commdtyCode":"000000000138758156","bu
sinessSign":"0","commdtyType":"","activeType":"01","subActiveType":""
,"productListPrice":"17.90",
```


综上所述，这种400错误实际上是因为网络传输过程中POST请求部分Packet丢失，导致服务端无法正常响应，超时后RST导致的。

4. 参考文献

http://www.jianshu.com/p/d028a37890b7 nginx的client_header_buffer_size和large_client_header_buffers学习 ligang1109

http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6687282 JDK-6687282 : URLConnection for HTTPS connection through Proxy w/ Digest Authentication gives 400 Bad Request

---