浅谈HTTP Keep-Alive

原创：宋士杰 大转转FE 1周前

背景是一次线上故障

项目类型Vue SSR

与server的数据交互用的http内网域名方案

在5月发生了一次线上CPU100%的问题，直接导致了NodeServer 500。

最终解决办法是：

1、请求超时时间timeout 5s->1s

2、开启Keep-Alive。

问题排查

首先我们重启服务短暂解决了不可访问的问题，既然是线上问题，一定不能忽视，我们开始通过一些手段复现问题。

查看nginx log

通过log分析我们发现，有些接口耗时是比较长的，而node ssr server是需要等待这些接口返回数据后才能把计算好的渲染结果返回给用户，跟相关后端同学商量，提升接口质量，同时我们把timeout从5s改为1s

qps

简单的qps测试，我们发现随着qps上升服务器cpu上涨特别快，相比较timeout修改之前已有了明显提升，但是明显感觉服务还是不够用，具体qps数值暂不透露。

分析服务器请求连接数

通过一些命令我们发现请求连接数竟然有6k+，很可怕的一个数字，而我们的qps是远远小于6k的，猜测很可能是每次请求都建立了新的链接，那就开始着手优化吧，就想到了Keep-Alive。

Keep-Alive模式

HTTP协议采用“请求-应答”模式，不开启KeepAlive模式时，每个req/res客户端和服务端都要新建一个连接，完成之后立即断开连接（HTTP协议为无连接的协议）；当开启Keep-Alive模式（又称持久连接、连接重用）时，Keep-Alive功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接。

Keep-Alive优点

从上面的分析来看，启用Keep-Alive模式肯定更高效，性能更高。因为避免了建立/释放连接的开销。下面是RFC 2616上的总结：

1. By opening and closing fewer TCP connections, CPU time is saved in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches), and memory used for TCP protocol control blocks can be saved in hosts.
2. HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much lower elapsed time. Network congestion is reduced by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.
3. Latency on subsequent requests is reduced since there is no time spent in TCP's connection opening handshake.
4. HTTP can evolve more gracefully, since errors can be reported without the penalty of closing the TCP connection. Clients using future versions of HTTP might optimistically try a new feature, but if communicating with an older server, retry with old semantics after an error is reported.

RFC 2616（P47）还指出：单用户客户端与任何服务器或代理之间的连接数不应该超过2个。一个代理与其它服务器或代码之间应该使用超过2 * N的活跃并发连接。这是为了提高HTTP响应时间，避免拥塞（冗余的连接并不能代码执行性能的提升）。

在项目中开启

上面大概介绍了Http Keep-Alive模式和优点，接下来就是怎么在项目里引用了，因为我们用axios作为请求库，它支持httpAgent,只需设置这个参数即可，Keep-Alive选用的agentkeepalive，接入项目非常简单。部分事例代码如下：

```
const Agent = require('agentkeepalive')

const keepaliveAgent = new Agent({
    maxSockets: 100,
    maxFreeSockets: 10,
    timeout: 60000,
    freeSocketKeepAliveTimeout: 30000 // free socket keepalive for 30 seconds
})
...
$axios.defaults.httpAgent = keepaliveAgent
```

结论

在调整请求超时时间和开启Keep-Alive后，cpu100%的问题没有再次出现，qps压测的数值也让人相当放心。

关于KeepAlive还有TCP和HTTP的区别，具体不在本文详述，感兴趣的同学可以自行网上搜索学习。

阅读原文