Nginx HTTP Limit

# 概述

Nginx提供了三个方面的限制。利用它们可以提高安全性。

# Limit Rate

限制连接的带宽。单位是Bytes每秒。该功能由http\_core、write\_filter模块提供。

## 配置

location /flv/ {

limit\_rate\_after 500k;

limit\_rate 50k;

}

这个配置例子的功能是发送Response时，前500Kbytes不限速，之后限速每秒50Kbytes。

## 算法

1. 计算下一秒可以发送的字节数。limit = limit\_rate \* S(now – start + 1) – (already\_sent – limit\_rate\_after)
2. 如果limit < 0 （即already\_sent太大导致）表示发送太快，需要延迟发送。
3. 计算需要延迟时间以补偿超额发送的 – limit 个字节。timeout = – limit \* 1000 / limit\_rate + 1
4. 设置写事件超时时间为timeout。
5. 如果limit > 0 表示发送较慢，可以继续发送limit个字节。
6. 调用API发送limit个字节。假设发送成功result个字节。
7. 再计算需要暂停的时间抵消发送result个字节所需的时间。timeout = result \* 1000 / limit\_rate
8. 设置写事件超时时间为timeout。
9. 写事件超时触发，回1)执行。

## 实现

ngx\_http\_write\_filter()

# Limit Request

限制请求率。单位是Requests每秒。该功能由limit\_req模块提供。

可以配置成不同的限制策略，例如：

* 来自同一个IP地址的请求限速为每秒5个。
* 访问某个虚拟服务器的请求限速为每秒1000个。

## 配置

http {

limit\_req\_zone $binary\_remote\_addr zone=one:10m rate=5r/s;

...

server {

...

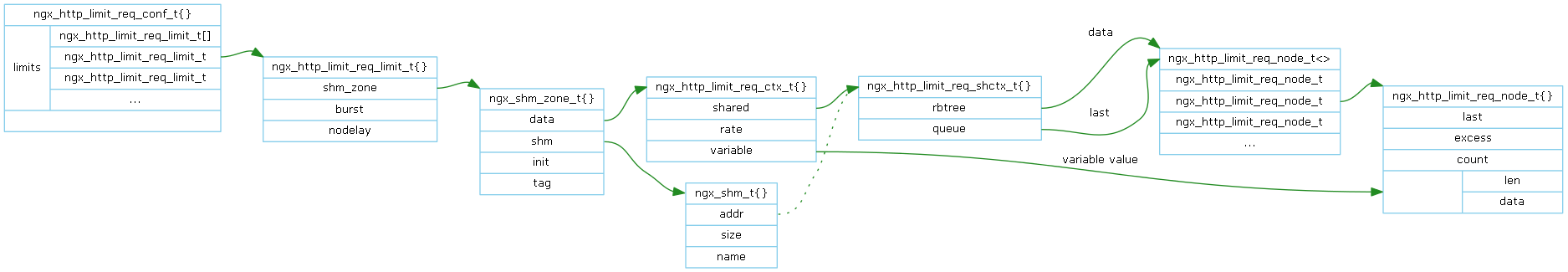
location /search/ {

limit\_req zone=one burst=9;

}

这个配置例子的功能是限制来自同一个Client IP的平均请求速率为每秒5个请求，突发量为9个。

## 数据结构

* ngx\_http\_limit\_req\_conf\_t

limit\_req模块的location conf数据结构，在limits数组保存了该Location下的所有limit\_req。

* ngx\_http\_limit\_req\_limit\_t

表示一个limit\_req。因为各个Worker并发在处理请求，需要共享请求数的信息。于是需要创建一块共享内存用于存放这些共享数据。

1. shm\_zone：管理共享内存
2. burst：最大突发量
3. nodelay：不能延迟处理请求

* ngx\_shm\_zone\_t

共享内存

* ngx\_http\_limit\_req\_ctx\_t

limit\_req 上下文，但它不在共享内存。

1. shared：指向共享的limit\_req上下文
2. rate：配置的请求率（0.001Requests / Second）
3. variable：被限制的对象的变量，比如$binary\_remote\_addr、$server\_name。用它来实现不同的限制策略。

* ngx\_http\_limit\_req\_shctx\_t

limit\_req上下文，它在共享内存。被各个Worker所访问。

它维护各个ngx\_http\_limit\_req\_node\_t。

* ngx\_http\_limit\_req\_node\_t

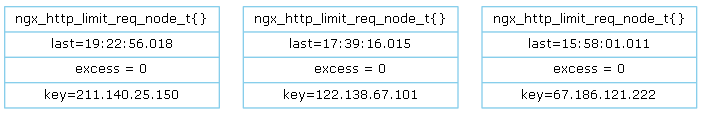
例如：表示一个Client IP的结节，用于记录请求情况。

1. data/len：变量值
2. last：前一个请求时间
3. excess：超额量

## 算法

现在以“来自同一个IP地址的请求限速为每秒5个”为例，讲一下限速算法原理。

来自同一个IP地址的请求信息记录在同一个ngx\_http\_limit\_req\_node\_t，IP地址是这个结点的key。



Client第一次请求时，显然无法找到key为Client IP的结点，于是创建并初始化一个。

计算从上次请求时刻到当前时刻这段时间内，允许的请求个数：allow\_count = rate \* S(now – last)

计算处理这个请求后的超额量：excess = last\_excess – allow\_count + 1

如果超额量 <= 0，即不超额，则直接继续处理请求。

如果超额量 > burst，即超过突发量，则直接503响应。

如果超额量 [0 , burst]，即超额但不超过突发量，则需要延迟请求的后续处理。

计算延迟时间： delay = ( excess / rate ) \* 1000 = excess \* 1000 / rate

## 实现

### 结点管理

结点(ngx\_http\_limit\_req\_node\_t)，是由各个Worker所共享的， 所以它们在共享内存中分配。

为了根据变量值快速查找结点，将结点组织成一棵红黑树。

另一方面由于共享内存大小是配置固定的，当有很多来自不同的IP地址的请求时，需要分配很多结点，导致共享内存不足，

这时需要释放0 rate的结点，或者释放末尾结点。为了高效实现这点，结点又同时组织成LRU队列。

### 算法优化

rate的单位是 1R/S，算出来的excess通常会是小数（float或double）。浮点数的算术计算比整数的算术计算要慢很多。

将rate的单位设为 0.001R/S，可将excess从浮点数转化为整数。

excess = last\_excess – rate \* MS(now – last) / 1000 + 1000

# Limit Connection

限制Client的并发连接数。该功能由limit\_req模块提供。

## 配置

http {

limit\_conn\_zone $binary\_remote\_addr zone=addr:10m;

...

server {

...

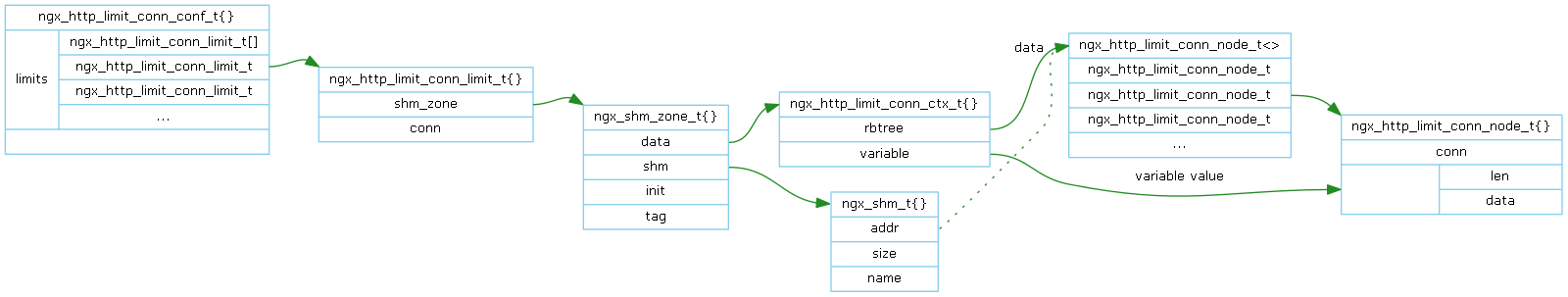
location /download/ {

limit\_conn addr 2;

}

这个配置例子的功能是限制来自同一个Client IP的连接个数为2个。

## 数据结构



* ngx\_http\_limit\_conn\_conf\_t

limit\_conn模块的location conf数据结构，在limits数组保存了该Location下的所有limit\_conn。

* ngx\_http\_limit\_conn\_limit\_t

表示一个limit\_conn。

1. shm\_zone：管理共享内存
2. conn：最大并发连接数

* ngx\_http\_limit\_conn\_ctx\_t

limit\_conn 上下文，但它不在共享内存。

1. rbtree：管理ngx\_http\_limit\_conn\_node\_t的红黑树
2. variable：被限制的对象的变量，比如$binary\_remote\_addr、$server\_name。用它来实现不同的限制策略

* ngx\_http\_limit\_conn\_node\_t

例如：表示一个Client IP的结节，用于记录连接情况。

1. data/len：变量值
2. conn：当前并发连接数

## 算法

现在以“来自同一个IP地址的并发连接数为2”为例，讲一下限制并发算法原理。

来自同一个IP地址的并发连接数记录在同一个ngx\_http\_limit\_conn\_node\_t，IP地址是这个结点的key。

Client第一次请求时，显然无法找到key为Client IP的结点，于是创建并初始化一个。

如果当前并发连接数 >= 最在并发连接数，则直接503响应。

如果当前并发连接数 < 最在并发连接数，则当前并发连接数增加1。注册pool的cleanup回调函数。继续处理请求。

当请求结束，pool被释放时，在cleanup的回调函数中，将当前并发连接数减少1。如果减为0，则释放结点。

？Question：

Request结束，不等于Connection结束。为何在Request而不是在Connection的内存池释放时机，将当前并发连接数减少1？

# Prototype

暂无

# Latest revision

https://github.com/lingjf/nginx\_analyse/blob/master/doc/