应用场景深度解析：Nignx性能优化指南

2017-11-22 11:23 | 阅读 5.2k | 评论 1

讲师介

> **李强**
>
> 天天拍车运维总监

- 网名：撒加，9年以上运维及管理经验。先后在AdMaster、饿了么担任运维经理，现任天天拍车运维总监，主要负责天天拍车运维架构的管理、持续优化记运维团队的建设、培养。
- 作为国内最早一批思科网络模拟器的推广者、虚拟者先锋论坛的创始人，一直致力于网络模拟器使用的推广，为国内培养网络工程师尽一份力。

主题简介：

> 1. 基础架构网络对Nignx性能的影响
> 2. Nignx安装优化
> 3. Nignx配置文件优化
> 4. Nignx与HAProxy、 Tomcat、 PHP-FOM搭配使用时的注意事项

> 5. TCP/IP协议栈优化对Nignx的影响

影响Nginx性能的因素

Nginx的优化不能单纯看Nginx本身，其实有很多方面会影响到Nginx的整体性能。

1、网络层面

带宽

带宽对Nginx性能的影响是最为直接的，就算如何独享10M的带宽也肯定不如100M带宽下Nginx的性能。另外，现在大部分公司的网站都拥有多个二级域名在提供服务，而这多个二级域名通常是共享一个出口带宽的，这样就造成Nginx来提供服务时，资源会受干扰。

网络质量

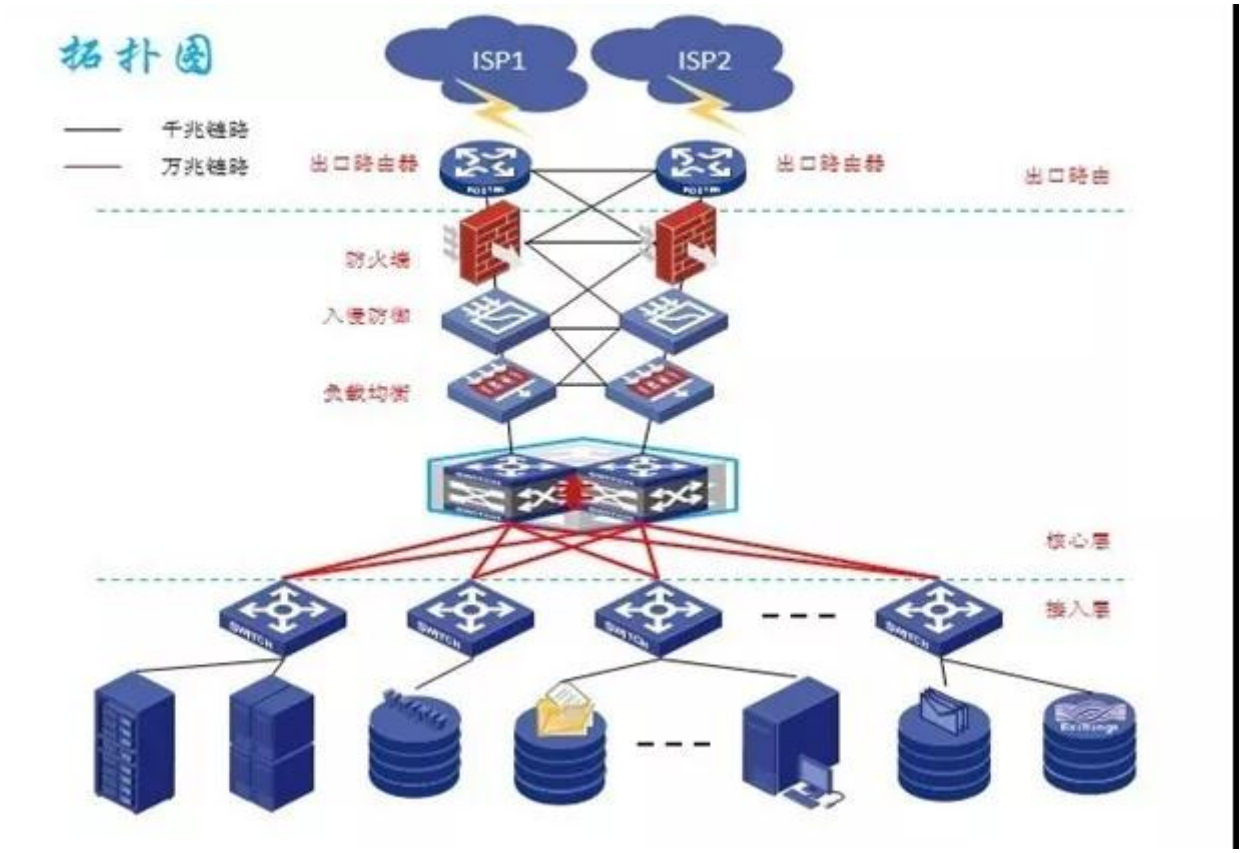
在中国的互联网中，网络质量跟国外是没法比的，经常出现的情况是客户端到Nginx服务端总是会经过好几个路由，但凡其中任何一个路由节点出现问题，会影响最终Nginx服务端的各种性能问题，例如数据重传、数据超时等。

Nginx直连交换机收敛比

交换机收敛比通常指交换机接服务器的下行流量跟交换机上行链路的比例，一般中型及中小型公司对IDC业务网络没有过多的规划，导致交换机收敛比通常是大于1的，也就意味着数据在交换机中是阻塞型转发，会影响Nginx的数据转发，尤其是作为反向代理时的影响更大。

Nginx架构中网络部署方式

大部分互联网公司通常会采用如下结构的网络：



这种结构对比互联网公司而言，请求需要经过路由器、防火墙、IDS/IPS等好几层的设备最终才能由业务服务器来响应请求，这期间请求没经过一层都需要消耗一定的时间，通过累计这个时间就会被放大，而且防火墙的性能直接决定了Nginx可以承载的请求数。所以迫不得已要用防火墙的话，请使用30万以上的防火墙来做，几万的防火墙性能其实达不到业务需求的。

2、服务器硬件层面

CPU

Nginx的工作模型是master-worker方式，简单来说，越多的worker也就意味着越多的承载力，CPU的核心数从某种程度来说决定了Nginx最佳的worker工作数量。

内存

内存的容量直接决定了Nginx可承载的最大连接数。

硬盘

Nginx的使用过程中，容易产生IO的地方还是挺多的，比如各种临时文件、错误日志、访问日志、缓存等。而硬盘的传输速度可以影响到这些产生的IO，比如硬盘的转速、硬盘的容量、硬盘的输出带宽等。

网卡

这里以戴尔服务器举例，戴尔服务器默认配置的网卡为Broadcom的，这种网卡一般使用没问题，但遇上Nginx在处理大量小包的情况下，默认配置的网卡就会发生严重丢包的事情，内核层面的丢包会造成大量的数据重传从而影响整个Nginx服务器的性能。所以对于Nginx而言需要选择合适的网卡。

3、操作系统层面

/etc/sysctl.conf的配置

很多时候大部分的运维都是从网上复制粘贴sysctl.conf的配置，对于其中一些value的配置都是没有任何依据，所以容易被出现的各种TCP状态而担心。而这些抄来的配置也从根本上影响着Nginx的运行。

系统资源限制

这里的资源限制主要指文件句柄数的限制，文件句柄数的多少限制了Nginx可以支持的最大连接数，不合理的配置会造成Nginx出现大量的500、502错误

IRQ Balanc

关于IRQ Balance服务，初衷是为了更好地利用CPU的资源来处理事务，但很多场景下，包括Nginx的应用，这个服务并不能起到利好的作用，反而会引起中断的不平衡造成Nginx性能下降。

系统开放多余的端口

严格来说，这种情况发生的几率还是比较低的。一般没有做运维标准化的公司，在服务器安装系统时，默认就会开启很多的服务，比如sendmail、postfix、ntpd、bind等等，这些服务一般监听在所有的IP上，也就意味着Nginx部署在这种环境下，一旦有恶意攻击者攻击非Nginx的端口，就会造成整个服务器资源被耗尽（DDoS的典型症状），从而让Nginx失去服务能力。

4、Nginx层面

编译进业务不需要的模块

有很多运维对于Nginx的安装大部分都是用yum或者apt-get进行安装，这种安装基本上都会把一些不需要的模块编译进去，在Nginx提供高并发时多少还是要消耗一定的内存资源，前面我们说了内存直接决定了Nginx能够承载多少连接的能力。

糟糕的配置

很多人发挥不了Nginx的性能，主要原因就是对Nginx的配置不熟悉，对于Nginx的参数指令不了解，所以各种各样的配置五花八门，这里实在没法一一列举，详情可以参看度娘，一搜一大片。

缓存使用不合理

Nginx的缓存，本意是为了提升Nginx的处理能力，降低上游服务器的压力，而大部分运维也是从网上抄，于是带来的结果就是缓存的不合理配置造成响应延时过高、缓存清理麻烦等问题。

5、上游服务器层面

Nginx在大部分的应用场景要么是通过http协议来反向代理上游服务器，要么是通过fastcgi协议来代理php应用，这两类应用也是会对Nginx自身的性能带来影响的。

PHP应用

当前一般都是使用PHP-FPM来提供fastcgi协议的接入，PHP的版本以及PHP-FPM的配置都会造成Nginx反代php-fpm时，容易发生502、504的错误。

Java应用

一般用于Java应用都逃不过Tomcat、jetty、resin，如果出现容器本身配置不合理、JVM不优化、Java容器选择不合理，将直接影响Nginx做反代时的并发能力。

Nginx优化项目

Nginx的优化根据上述的性能影响因素来一一说明。

1、网络层面

- 在满足业务以及性价比的前提下，越高的带宽意味着Nginx可以承载更多的连接和并发；
- 在条件允许的情况下使用多个ISP运营商线路来提供服务，并使用策略路由解决多线路之间的请求响应；
- 在网络的部署架构中，尽量避免防火墙作为HTTP请求第一个经过的设备，建议由Nginx或者第一级的4层负载均衡来转发请求；
- 对于整个网站而言，有条件要设计大二层的网络（胖树结构），这样有利于服务器的水平扩展；
- 对于使用IDC的用户，建议选机房时使用mtr工具对目标机房进行72小的测试，来检验机房的网络质量，这里主要包括请求到达机房时有没有出现某一跳出现多个路由的情况、某一跳是否延迟超级不稳定（mtr工具中最后一列表示标准方差，简单认为就是这个值越大这个节点越不稳定），通过这样选出网络质量相对较好的机房；
- 在设计网络时，要考虑到交换机的收敛比，尽可能做到<=1，才能做到数据的线速转发。

2、服务器硬件层面

CPU

不管使用云主机还是物理机，将Nginx独立并且把Nginx作为负载均衡服务器时，越多的CPU也就意味着Nginx可以启用更多的worker来处理请求，这里特别要说明下，CPU的主频不需要作为参考指标，主要是物理CPU的核心数以及是否支持HT技术。

内存

越大的内存也就意味着Nginx理论上拥有的最大连接承载能力，而且Nginx使用到缓存时，还可以将通过tmpfs将缓存内容直接放在内存中，从而提高Nginx的处理性能；也可以使用tmpfs作为Nginx各种临时文件的存放地，降低磁盘的IO。

硬盘

用SSD、PCIe-SSD来替换SAS机械硬盘，在Nginx并发较高的服务器上，可以将访问日志及错误日志放在拥有更高IOPS的磁盘上，从而保证Nginx的性能，实际上，我们大部分时候测试Nginx时，是在测试Nginx服务器上磁盘的性能。（大家好好回味下why）

网卡

上面网络层说要更高的带宽，这个带宽指运营商提供的服务带宽，光有服务带宽还不够，还需要服务器有对应的网卡来支持，比如用到了5G的带宽，那服务器网卡最少需要万兆的，当然网卡的品牌也很重要。对于千兆的环境，建议使用英特尔i350的网卡，对于需要使用万兆网卡的环境，建议优先选择Mellanox的万兆网卡（最便宜也要6000块钱一块，土豪用），资金有压力的公司可以选择英特尔x520的网卡，因为这些网卡除了小包的转发性能强劲外，还有需要offload特性可以降低系统kernel的消耗，提升Nginx的承载能力。

只不过在用这些好的网卡时，一定要做中断绑定（减少CPU0的中断请求，让充分发挥网络多队列的优势）。

3、操作系统层面

```
/etc/sysctl.conf

net.ipv4.tcp_max_tw_buckets = 1000

net.ipv4.tcp_sack = 1

net.ipv4.tcp_window_scaling = 1

net.ipv4.tcp_rmem = 4096 8388608 16777216

net.ipv4.tcp_wmem = 4096 8388608 16777216

net.core.wmem_default = 8388608

net.core.rmem_default = 8388608
```

> net.core.rmem_max = 16777216
>
> net.core.wmem_max = 16777216
>
> net.ipv4.tcp_orphan_retries = 1（TCP_FIN_WAIT1、TCP_FIN_WAIT2、TCP_LAST_ACK、TCP_CLOSING）
>
> net.core.somaxconn = 262144
>
> net.ipv4.tcp_max_orphans = 32768
>
> net.ipv4.tcp_max_syn_backlog = 262144
>
> net.ipv4.tcp_timestamps = 0
>
> net.ipv4.tcp_synack_retries = 1
>
> net.ipv4.tcp_syn_retries = 1
>
> net.ipv4.tcp_retries2 = 5
>
> net.ipv4.tcp_tw_recycle = 0
>
> net.ipv4.tcp_tw_reuse = 0
>
> net.ipv4.ip_local_port_range = 10000 65535
>
> net.ipv4.tcp_slow_start_after_idle = 0
>
> 其中net.core.somaxconn这个配置在CentOS 6系列中，这个参数的值可以大于65535，在CentOS 7中，这个参数的最大值为65535
>
> net.ipv4.tcp_timestamps
>
> net.ipv4.tcp_tw_recycle
>
> net.ipv4.tcp_tw_reuse

这三个参数一般建议都配置为0，高并发时tw连接想快速回收及复用还是呵呵吧。

> net.ipv4.tcp_slow_start_after_idle

这个参数是禁用TCP慢启动的，当然如果使用CentOS 6的用户建议将版本升级到6.5以及6.5以后的版本，这个参数配置才有意义，低于6.5的版本，需要做些特殊配置才可以。

其它注意事项

1、net.ipv4.tcp_mem没事别瞎改，系统自动计算，别吃饱撑着！

2、fs.file-max没事别瞎改，内核根据内存大小自动计算，表示kernel可分配的最大文件句柄数，这个参数的大小简单的来说约等于cat /proc/meminfo|grep MemTotal的大小乘以10%，精确的计算可以在我博客查看。

3、fs.nr_open可以修改，此为单个进程可分配的最大文件句柄数，默认是为1024*1024

4、以下三个参数在Nginx没有启用SO_TCPKEEPALIVE特性时，根本不会生效，且启用SO_TCPKEEPALIVE特性的参数，不是keepalive_timeout！

net.ipv4.tcp_keepalive_time

net.ipv4.tcp_keepalive_probes

net.ipv4.tcp_keepalive_intvl

HTTP Keep-Alive与TCP Keepalive是两个不同的概念，简单说前者是复用一个TCP连接，后者用于TCP连接的链路探活。

关于HTTP Keep-Alive与TCP Keepalive我博客近期会发布一片文章，通过wireshark抓包来详解。

5、net.ipv4.tcp_max_orphans不要设置过大，一个孤儿连接要占用64K的不可交换内存，网上动辄这个值配置个几百万，吼吼，为内存感到担忧。

6、net.ipv4.tcp_max_tw_buckets没事别设置几万几十万的，这个参数是TW的连接超过这个设置时在/var/log/messages中打印警告信息，同时释放TW连接，可以设置小一点，当系统中TW太多时，net.ipv4.tcp_tw_recycle和net.ipv4.tcp_tw_reuse两个参数并没有卵用，所以TW连接过大的可以直接暴力的将net.ipv4.tcp_max_tw_buckets设置为0，并没有什么问题，当然最终解决还是要依赖程序和对Nginx的配置以及重新编译内核，减少TW连接的产生。

7、rmem和wmem可根据netstat –s|grep socket的结果来判断是否足够，出现类似以下信息时就要考虑增加对应的大小了。

269651 packets pruned from receive queue because of socket buffer overrun

5346082 packets collapsed in receive queue due to low socket buffer

Nginx层面

1、安装优化

安装优化主要针对Nginx的编译安装，这里包含两个部分，一是针对编译器的编译参数优化，二是通过Nginx的应用场景来定制化Nginx的编译选项。以下通过Nginx最为静态资源服务器的案例来抛砖引

玉地说下Nginx安装的优化：

首先要确定Nginx是针对哪类静态资源做服务，例如提供js、css以及jpg、png等资源的访问，还是针对MP3、MP4、RMVB、zip、rar等大文件的资源访问。这两种静态资源的场景使用对Nginx的编译参数还是有很大影响的。js\css\jpg等这类静态资源一般文件都是比较小的，作为互联网公司一般文件的平均大小不会超过4M，所以一般编译时是不需要将–with-threads编译进去，而后者这种文件动辄都会超过10M的静态资源，一般就需要将–with-threads开启了，以便使用Nginx的线程池来提升文件传输的性能。

其次，作为静态资源服务器，大部分的模块是使用不到的，能使用到的模块有ssl模块、http v2模块，所以编译Nginx的参数就变成下面这样了：

```
./configure –prefix=/opt/websuite/nginx \
–conf-path=/opt/config/nginx/nginx.conf \
–modules-path=/opt/websuite/nginx/modules \
–error-log-path=/opt/logs/nginx/error.log \
–http-log-path=/opt/logs/nginx/access.log \
–pid-path=/opt/run/nginx –user=websuite \
–group=websuite \
–with-file-aio \
–with-http_ssl_module \
–with-http_v2_module \
–with-http_stub_status_module \
–without-http_ssi_module \
–without-http_charset_module \
–without-http_access_module \
–without-http_auth_basic_module \
–without-http_autoindex_module \
–without-http_geo_module \
–without-http_split_clients_module \
```

```
–without-http_proxy_module \

–without-http_fastcgi_module \

–without-http_uwsgi_module \

–without-http_scgi_module \

–without-http_memcached_module \

–without-http_empty_gif_module \

–without-http_browser_module \

–without-http_upstream_hash_module \

–without-http_upstream_ip_hash_module \

–without-http_upstream_least_conn_module \

–without-http_upstream_keepalive_module \

–without-http_upstream_zone_module \

–http-client-body-temp-path=/opt/websuite/nginx/temp/client \

–without-mail_pop3_module \

–without-mail_imap_module \

–without-mail_smtp_module \

–with-google_perftools_module \

–with-pcre=/tmp/nginx/pcre-8.41 \

–with-pcre-jit \

–with-openssl=/tmp/nginx/openssl-1.0.2j \

–with-openssl-opt="threads shared no-zlib no-comp no-ssl2 no-ssl3 no-ssl3-method"
```

通过以上的编译参数编译生成的Nginx可执行文件是很小的，启动时载入的动态链接库也最少，在并发较高时，单个worker占用的内存资源是很小的。

2、Nginx访问日志优化

访问日志的优化主要是3个方面：

- 增加buffer
- 使用map过滤不必要的日志
- 制定合适的log_format

例如：

```
log_format proxy        '<$remote_addr> <$time_local> <$http_user_agent> <$http_referer> <$server_protocol> '
                        '<$request_method> <$uri> <$request_uri> <$request_length byte> <$request_time> <$status> '
                        '<$upstream_addr> <$upstream_status> <$upstream_response_length> <$upstream_connect_time> '
                        '<$upstream_response_time> <$upstream_header_time> <$bytes_sent byte> <$realip_remote_addr>';
```

```
map $uri $expanded_name {

~^(.*\.(gif|jpg|jpeg|png|bmp|swf|js|css)$)  0;

default 1;

}

access_log /opt/logs/openresty/access.www.grapenvine.cn proxy buffer=1m if=$expanded_name
```

表示使用proxy作为www.grapenvine.cn的日志记录格式，并且设置buffer为1m，当buffer中的日志超过1m时再写入日志文件，同时请求中包含map中指定的文件将不记录到访问日志里。

Proxy相关参数优化

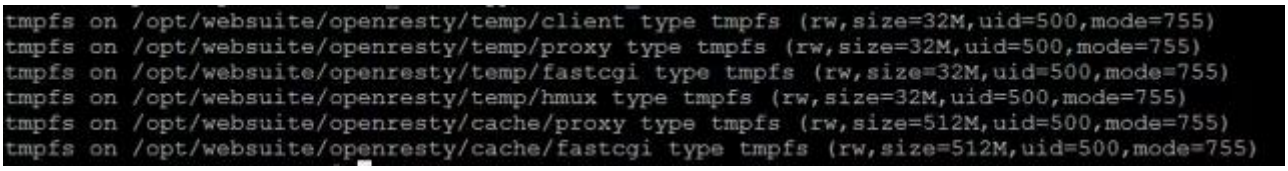
```
proxy_connect_timeout   30;

proxy_send_timeout      30;

proxy_read_timeout      60;

proxy_buffer_size       64k;

proxy_buffers           4 64k;

proxy_busy_buffers_size  128k;

proxy_next_upstream     invalid_header http_500 http_503 http_403 http_502 http_504;

proxy_next_upstream_timeout     1s;

proxy_next_upstream_tries    1;
```

其中proxy_buffer_size的值可通过统计一天accesslog中的$bytes_sent，$bytes_sent的平均值即为proxy_buffer_size的值，$bytes_sent的最大值则为proxy_buffers的大小。例如，平均值为50K，最大值为230k，则proxy_buffer_size就为64k，proxy_buffers就设置4 64k。

对于next_upstream的用法可参考 http://www.grapenvine.cn/post/95。

fastcgi模块的优化与proxy模块的优化方法类似。

3、合理利用tmpfs



通过使用tmpfs，可将Nginx的临时目录及缓存目录放到内存中，降低对磁盘IO的消耗。

4、Nginx主配置优化

```
worker_processes auto;

worker_cpu_affinity auto;

worker_rlimit_nofile 100000;

pcre_jit on;

events{

use epoll;

worker_connections 8192;

accept_mutex off;

}
```

一般worker_connections* worker_processes < worker_rlimit_nofile,其中worker_rlimit_nofile不受ulimit –n的限制，如果没有配置此参数，则以ulimit –n的值为主。

accept_mutex网站流量小设置为on也没问题，大流量的情况下要设置为off，在nginx 1.11.3版本中默认就关闭了。

5、Nginx默认服务器

建议配置一个default server，例如：

```
server
{
        listen 80 default_server backlog=2048 reuseport;
        rewrite_log off;

        location / {
                rewrite ^/(.*)$ http://www.ttpai.cn/$1 break;
        }
        access_log /opt/logs/openresty/access.default.log static;
        error_log /opt/logs/openresty/error.default.log;

}
```

当你的Nginx要提供的服务很多时，增加default server中的backlog。加入reuseport可以让所有的worker都监听在80端口，提高Nginx的并发性能，reuseport与accept_mutex是互斥的。另外如果要启用SO_TCPKEEPALIVE机制，需要在listen指令中配置so_keepalive=on

6、Nginx负载均衡

Nginx负载均衡环境中，在upstream启用keepalive指令，可用在proxy和fastcgi的场景里。

例如：

```
upstream netemu {

server unix:/opt/run/php/pool1.sock;

server unix:/opt/run/php/pool4.sock;

keepalive 4;

}
```

- 对于proxy模块需要增加指令

```
proxy_http_version 1.1;

proxy_set_header Connection “”;
```

另外上游服务器必须配置keepalive timetout的相关配置，并且将这个超时时间可以设置长一点，比如10分钟、半小时都可以，如果上游服务器关闭了keepalive，那么Nginx的配置指令里配置了keepalive NUM，是没什么效果的。

- 对于fastcgi模块需要增加指令

```
fastcgi_keep_conn on;
```

- 当在upstream中使用负载均衡算法时，算法指令要放在keepalive指令前；

通过这种方式，可以显著降低Nginx与upstream中server的timewait连接，keepalive的值不要设置过大，因为这是针对每个worker的。

Nginx优化的题外话

这个环节，主要是通过Nginx的第三方模块来扩展Nginx的功能。

- **Nginx状态输出**

> nginx-module-vts

这个模块可以详细地输出Nginx的运行情况，通过json格式的输出，还可以纳入Zabbix监控中，来监控不同的vhost以及不同的upstream的运行状态，便于详细掌握Nginx的运行情况。

- **动态更新Nginx Upstream**

> ngx_dynamic_upstream
>
> nginx-upsync-module
>
> lua-upstream-nginx-module

动态更新nginx upstream其实是为了实现上游服务器进行代码发布时，让上游服务器能够平滑的上下架，从而避免对用户抛出错误页面的。其中nginx-upsync-module这个模块是最好用的，配合良好的代码发布流程可以很好地实现上游服务器的上下架。

- **缓解robots压力**

> testcookie-nginx-module

相信很多做互联网公司的运维童鞋，尤其是那些互联网金融，还有提供各种优惠券的站点，最头大的就是被别人恶意的刷短息接口、刷优惠码等，这个模块对于那些没有有Lua开发能力的公司而言，是个福音。

- **分布式共享缓存**

> srcache-nginx-module+memc-nginx-module+memcached
>
> srcache-nginx-module+memc-nginx-module+couchbase
>
> srcache-nginx-module+redis2-nginx-module+redis

这三种方案都是用于将缓存放入memcache或者Redis的，从而让Nginx支持分布式缓存，而且对于缓存存储不受Nginx本地存储的影响，其中使用CouchBase的方案我正在测试中，有结果会发布在博客上。

Q&A

Q1：Nginx可以缓存动态内容吗？

A1：可以，需要做一些条件判断，简单说就是把get方法的缓存，post方法的bypass，后续大家可以参考我博客里的动态页面缓存。

Q2：新版本支持TCP吗？

A2：支持，但不建议使用Nginx来做四层负载均衡。

Q3：如果对数据的实时性要求不高，能用Nginx把jsp页面和后台数据库的交互数据缓存？

A3：是的，而且可以对缓存内容做控制，根据页面时间的需要做配置。

Q4：我们业务有websocket？连接传数据，这样哪个更合适？
Q4：如果这样，一般建议使用nginx的一个模块来实现，模块叫nchan来做websocket。

回放链接
直播：https://m.qlchat.com/topic/details?topicId=2000000180697770&nullPPT：
https://pan.baidu.com/s/1pLMGwVD

原文来自微信公众号：DBAplus社群