讲师介绍

Hash　　QQ：805921455

从事Java软件研发近十年。

前新浪支付核心成员、

咪咕视讯(中国移动)项目经理、

对分布式架构、高性能编程有深入的研究。

明天，你一定会感谢今天奋力拼搏的自己

你不知道的千万并发场景中
Nginx实用插件
分布式高并发—负载均衡

目录

课程安排

01

代理服务故障处理

后端服务出现故障如何处理？

02

高并发限流

保护能力有限的后端服务、针对爬虫限流

03

黑白名单插件

来者不善，给它一个黑名单

04

更多常用插件

玩转Nginx第三方插件

目录

代理服务故障处理

被动健康检查

ngx_http_upstream_module，可以做到基本的健康检查。

upstream指令块中的server子指令参数：max_fails、fail_timeout

```
upstream backend {
    server 127.0.0.1:8080 weight=1 max_fails=2 fail_timeout=10s;
    server 127.0.0.1:8081 weight=1 max_fails=2 fail_timeout=10s;
}
```

max_fails=2的意思是设定Nginx与服务器通信的尝试失败的次数。在fail_timeout参数定义的时间段内，如果失败的次数达到此值，Nginx就认为服务器不可用。在下一个fail_timeout时间段，服务器不会再被尝试。失败的尝试次数默认是1。设为0就会停止统计尝试次数，认为服务器是一直可用的。

fail_timeout=10s是设定服务器被认为不可用的时间段以及统计失败尝试次数的时间段。在这段时间中，服务器失败次数达到指定的尝试次数，服务器就被认为不可用。默认情况下，该超时时间是10秒。

Nginx能不能主动感知服务不可用呢？

主动健康检查

服务主动健康检查的Nginx插件，nginx_upstream_check_module

nginx 需要添加nginx_upstream_check_module 模块，用于对后端服务器的健康情况检测，如果后端服务器不可用，则把这台服务器移除负载均衡轮循集群，所有的请求不往这台服务器上转发，待这台服务器恢复正常后，再把这台加入到负载均衡集群。

```
upstream cluster {
    # simple round-robin
    server 127.0.0.1:8080;
    server 127.0.0.1:8081;
    check interval=5000 rise=1 fall=3 timeout=4000;
    #check_http_send "HEAD / HTTP/1.0\r\n\r\n";
    #check_http_expect_alive http_2xx http_3xx;
}
```

参考《Nginx实用插件》操作手册

故障转移

ngx_http_proxy_module模块对故障转移的支持

proxy_next_upstream

在尚未向客户端发送任何内容的情况下才能将请求传递给下一个服务器。也就是说，如果在传输响应的过程中发生错误或超时，则无法进行故障转移。一个请求传递给下一个服务时，还可以通过限制重试次数、等待时间。

proxy_next_upstream_timeout

限制请求可以传递到下一个服务器的时间。默认为0，表示关闭限制。

proxy_next_upstream_tries

限制将请求传递到下一个服务器的可能尝试次数 。默认为0，表示关闭限制。

如何应对服务雪崩

Nginx后面有2台上游服务，每台服务负载5000并发，假如第一台挂掉，Nginx将第一台的请求转发给第二台，将第二台打挂，整个服务都不可用了。如果是多台服务，就这样多米诺骨牌效应产生，将全部服务打挂？怎么处理？

proxy_read_timeout

定义从代理服务器读取响应的超时。仅在两个连续的读操作之间设置超时，而不是为整个响应的传输。如果代理服务器在此时间内未传输任何内容，则关闭连接。默认60s。

max_conns参数

设置upstream中server指令的max_conns参数来进行控制每台服务器能够处理的最大的连接数

动态更新上游服务

后端服务发生变化，需要调整nginx的upstream列表，而且需要重启nginx，有一种插件可以无需重动nginx，动态更新后端服务地址列表。这个插件就是，ngx_http_dyups_module模块。

安装演示操作，参考《Nginx实用插件》操作手册

这个插件有个问题比较明显，因为数据存放在内存中，nginx 重启之后，配置内容就会清空。

Nginx可以通过Consul+Consul-template来解决（需要Nginx reload）

OpenResty可以通过balancer_by_lua+Consul来解决（无需Nginx reload）

目录

高并发限流

限制请求

Nginx中可以通过请求限制模块的指令来控制客户端的请求速率。使用“漏桶”算法，限制指定Key的请求处理速率，特别是从一个单一的IP地址的请求的处理速率。

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
    limit_req_zone $server_name zone=preserver:10m rate=10r/s;
    server {
        location /search/ {
            limit_req zone=one burst=5;
            limit_req zone=perserver burst=10;
        }
     }
}
```

示例，通过定义内存空间块：

限制单个IP平均每秒允许不超过1个请求，突发不超过5个请求

整个虚拟服务，平均每秒不允许超过10个，突发不超过10个请求

限制请求-相关指令

指令	作用	默认值
limit_req	设置共享内存区域和请求的最大突发大小。如果请求速率超过为区域配置的速率，则延迟其处理，以便以定义的速率处理请求。过多的请求被延迟，直到它们的数量超过最大突发大小，在这种情况下请求以错误终止 。	-
limit_req_dry_run	启用空运行模式。这个模式下，请求处理速率不受限制，但是，在共享存储区中，过多请求的数量照常计算。	on
limit_req_log_level	为服务器因速率超过或延迟请求处理而拒绝处理请求的情况设置所需的日志记录级别。延迟的记录水平比拒绝的记录水平低一个点;	error
limit_req_status	设置要响应拒绝的请求而返回的状态代码	503
limit_req_zone	设置共享内存区域的参数，该区域将保留各种键的状态。特别是，状态存储当前的过多请求数。该key可以包含文本，变量，他们的组合，空键值的请求不计算在内。	-

限制连接

Nginx中通过ngx_http_limit_conn_module模块，来限制连接到nginx服务的数量。用于限制每个定义密钥的连接数，特别是来自单个IP地址的连接数。并非所有连接都被计算在内 只有当服务器正在处理请求并且已经读取了整个请求标头时，才会计算连接。

示例，通过定义内存空间块：

限制单个IP建立不超过1个连接

整个虚拟服务，不允许超过10个连接

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_conn_zone $server_name zone = perserver: 10m;
    server {
        location /download/ {
            limit_conn addr 1;
            limit_conn perserver 100;
        }
     }
}
```

限制连接-相关指令

指令	作用	默认值
limit_conn	设置共享内存区域和给定键值的最大允许连接数。超过此限制时，服务器将返回 错误 以回复请求。	–
limit_conn_log_level	为服务器限制连接数的情况设置所需的日志记录级别	error
limit_conn_status	设置要响应拒绝的请求而返回的状态代码	503
limit_conn_zone	设置共享内存区域的参数，该区域将保留各种键的状态。特别是，状态包括当前的连接数。该key可以包含文本，变量，他们的组合。具有空键值的请求不计算在内。	–

限制给客户端的响应速率

http核心模块中对响应速率限制提供了支持，视频网站，不可能将所有带宽都给一个请求，我们需要限制每个请求的速率

```
http {
    server {
        location /download/ {
            limit_rate_after 500k;
            limit_rate       50k;
        }
     }
}
```

示例，限制响应初始速率50k，后续响应速率限制在500k

针对爬虫限速

网络爬虫对网站的作用

一方面可以给网站带来一定的流量，便于搜索引擎收录，利于用户搜素。

另一方面会给服务器带来一定的压力，在网络爬虫对网站内容进行收录时，会引起服务器负载高涨。

有没有什么方法既不阻止网络爬虫对网站内容进行收录，同时对其连接数和请求数进行一定的限制呢？

robots.txt 协议

robots.txt文件内容称为爬虫协议，搜索引擎会根据这个文件的内容，来确定它访问权限的范围。它不是命令要求，需要搜索引擎自觉遵守。

robots.txt 必须放置在一个站点的根目录下，而且文件名必须全部小写,一词不差。

```
robots.txt  写法：
User-agent: * 这里的*代表的所有的搜索引擎种类，*是一个通配符
Allow: /cgi-bin/  这里定义是允许爬寻cgi-bin目录下面的目录
Disallow: /admin/ 这里定义是禁止爬寻 admin 目录下面的内容
Disallow: /require/ 这里定义是禁止爬寻 require 目录下面的内容
```

针对爬虫限速-Nginx配置

Map是nginx模块中用来映射变量的

根据每次请求$http_user_agent的内容进行子指令进行匹配，如果匹配成功，则将子指令右侧的值赋值给$agent变量。匹配不到则给默认值。

示例中表示，根据客户端的操作系统浏览器工具等信息，匹配是否为curl、apachebench、spider、bot、slurp也就是爬虫程序相关的名称，这里采用了原有内容作为$agent的值。

Limit_conn_zone、limit_req_zone则采用$agent做为key来存储限制速率。

```
map $http_user_agent $agent {
    default "";
    ~curl $http_user_agent;
    ~*apachebench $http_user_agent;
    ~*spider $http_user_agent;
    ~*bot $http_user_agent;
    ~*slurp $http_user_agent;
}
limit_conn_zone $agent zone=conn_ttlsa_com:10m;
limit_req_zone $agent zone=req_ttlsa_com:10m rate=1r/s;
    location / {
        limit_req zone=conn_ttlsa_com burst=5;
        limit_conn req_ttlsa_com 1;
        limit_rate 500k;
    }
```

限流插件-限制客户端上传速率

ngx_limit_upload_module是一个tengine模块，可以限制客户端发送请求主体时的上传速率。与标准的nginx limit_rate功能用法相同

速率限制请求主体从客户端的传输。速率以每秒字节数指定。值0禁用速率限制。根据请求设置限制，因此如果客户端同时打开两个连接，则总速率将是指定限制的两倍。

```
Syntax:    limit_upload_rate rate;
Default:   limit_upload_rate 0;
Context:   http, server, location, if in location
```

```
Syntax:     limit_upload_rate_after size;
Default:    limit_upload_rate_after 0;
Context:    http, server, location, if in location
```

设置初始金额，在此之后，来自客户端的请求正文的进一步传输将受到速率限制。

目录

黑白名单插件

Nginx的访问模块

ngx_http_access_module模块，提供允许、拒绝指令来控制客户端能否访问服务。

```
location / {
    deny 192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny all;
}
```

allow，允许某个ip或者一个 ip 段访问

deny，禁止某个ip或者一个 ip 段访问

黑白名单插件

nginx_white_black_list可以根据定义的黑白名单配置文件来进行控制客户端能否访问服务。

处在黑名单中的 ip 与网络，将无法访问 web 服务。

处在白名单中的 ip，访问 web 服务时，将不受 nginx 所有安全模块的限制。

支持动态黑名单（需要与 ngx_http_limit_req 配合）

目录

更多常用插件

其他插件

参考官网wiki：https://www.nginx.com/resources/wiki/modules/

谢谢观看