NGINX从入门到精通进阶系列培训

高级篇：变量、API

林静　Software Solution Architect

https://myf5.net

变量

- 内置变量及其利用
- set变量及其作用域
- 变量mapping及其利用
- Keyval变量

NGINX变量概述

```
$variable_name
```

NGINX中以$开头的字符串为变量对象

变量形态

1.内置变量

2.Set自定义变量

3.Keyval变量

常见用途

1.条件代理

2.条件日志

3.HTTP报文头映射

4.采集和收集统计信息

5.选择性认证

6.动态黑名单

7.A/B测试

8.Request ID追踪

内置变量及其利用

- 内置变量一般用于获取请求或响应的各种信息，由NGINX各模块自动产生
- 常见的如$args, $uri, $request_uri,$geoip_latitude等

- 单个变量，例如$args
- 变量群 变量， 例如$arg_parameter1, $arg_parameter2 ..
  类似的变量群: $cookie_*, $http_* ,$upstream_http_*, $upstream_cookie_*,upstream_trailer_*
  $sent_http_*, $sent_trailer_*

变量群变量名的获取

```
[root@plus2 conf.d]# cat var.train.conf
    server {
        server_name 172.16.100.252;
        listen 172.16.100.252:8080;
        location / {
                return 200 "uri: $uri  request_uri: $request_uri <br> xff: $http_x_forwarded_for <br> arg: $args <br> arg_test: $arg_test <br> cookie_testcookie: $cookie_testcookie";
                #proxy_pass http://60.205.226.20:80;
        }
    }
```

```
λ curl -H "Cookie: testCOOkie=1111" -H "X-Forwarded-For: 8.8.8.8" "http://172.16.100.252:8080/test?test=aaa%26bb=111"
uri: /test  request_uri: /test?test=aaa%26bb=111 <br> xff: 8.8.8.8 <br> arg: test=aaa%26bb=111 <br> arg_test: aaa%26bb=111 <br> cookie_testcookie: 1111
```

- header，转化为小写匹配，中杠变下杠
- parameter， 转化为小写

内置变量是否可以重新赋值?

- 已索引类的变量，诸如$uri 等此类变量一般不能被重新赋值
- nginx: [emerg] the duplicate "uri" variable in /etc/nginx/./conf.d/var.train.conf:83
- （已索引的的变量，在NGINX内部有存放该变量值的容器）

内置变量是否可以重新赋值?

- 未索引类变量，诸如$args, $arg_*, $cookie_*, $http_*等可被重新赋值
- 未索引指的是当用户调用时候去启动一个内置程序去获取当前的值，并不提前预先索引好

```
location /internal {
            set $orig_args $args;
            set $args "changed=1&result=1";
            set $orig_testheader $http_testheader;
            set $http_testheader "had-changed-header-value";
            return 200 "Original args: $orig_args , But last nginx internal args is: $args |||| Original testheader: $orig_testheader , But last nginx internal header: $http_testheader";
        }
```

```
λ curl -H "testheader: 11111" "http://172.16.100.252:8080/internal?a=1&b=1"
Original args: a=1&b=1 , But last nginx internal args is: changed=1&result=1 |||| Original testheader: 11111 , But last nginx internal header: had-changed-header-value
```

内置变量与子请求

- 一般来说可以将子请求理解为一个独立的请求，因此其内置变量的取值就是该子请求相关的值，比如 $uri, $args 等等

- 根据子请求实现模块的不同，有些变量在子请求中可能不是期望的,例如$request_method, $request_uri， 在子请求上下文中其值依然来自主请求

使用变量

- 强制80端口跳转到443端口

```
server {
   listen 80;
   return https://$host$request_uri
}
```

- 在代理http请求中保留uri

```
server {
   listen 443 ssl;
   proxy_pass http://example.com:8080$request_uri;
}
```

所有内置变量

https://nginx.org/en/docs/varindex.html

set设置自定义变量

SET指令用来自定义变量

- set 有两个含义： 创建变量，给变量赋值
- 创建变量是在加载配置时候
- 赋值变量是当请求处理上下文需要时触发
- 直接调用一个未被创建的变量，会导致配置无法启动
- 变量名是整个配置文件可见，但变量的值是基于每个独立请求的上下文

```
set $variable_name
value;

set $foo “hello world”;
set $bar 101;
set $combo $foo
```

set自定义变量作用域

- 变量名是整个配置文件可见，但变量的值是基于每个独立请求的上下文
- Rewrite导致NGINX内部location跳转情形下，此时请求还属于同一个请求，因此变量值是不变的
- 同一个上下文中，多次set同一个变量，使用配置中最后一个set值 （rewrite模块中return有额外影响）
- set命令可以用在 server，location，if 配置上下文中

以下配置是否可以启动?

```
#set $f5 "is the best Loadbalancer";

 location /noset {
            return 200 "F5: $f5";

      }

 location /set {
            set $f5 "nginx is part of F5";
            return 200 "F5: $f5";
      }
```

```
λ curl http://172.16.100.252:8080/noset
F5:
C:\Users\jlin
λ curl http://172.16.100.252:8080/set
F5: nginx is part of F5
```

- 直接调用一个未被创建的变量，会导致配置无法启动
- 变量名是整个配置文件可见，但变量的值是基于每个独立请求的上下文
- 赋值变量是当请求处理上下文需要时触发

问题：

```
location /set_after_return {
        set $var "before return";
        #return 200 "var: $var";
        proxy_pass http://60.205.226.20:80$var;
        set $var "/";
}
```

$var 值是什么?

/

```
location /set_after_return {
        set $var "before return";
        return 200 "var: $var";
        #proxy_pass http://60.205.226.20:80$var;
        set $var "/";
}
```

$var 值是什么?

before return

set变量与子请求

- 正常情况下，主 子 请求都应维护其自己的独立变量值，就像两个完全不同的请求一样
- 但是有可能存在父请求使用子请求的变量值情况，比如auth_request指令

```
location /main_auth_request {
        set $var "/";
        auth_request /sub_auth;
        #return 200 "The var value is $var";
        proxy_pass http://60.205.226.20:80$var;
}

location /sub_auth {
        set $var "/get404fromcnadn";
        return 200 "sub request";
}
```

右图中，最后服务器收到的请求uri是

http://60.205.226.20:80/get404fromcnadn

如果将/sub_auth下的 $var设置为 / 则不会返回404错误

变量mapping

基于源变量值的匹配来赋值结果变量（$result-variable）

```
map $source-variable $result-variable {
        default    foo
        f5         ltm
        f4         shancai
}
```

源变量的值匹配：

- 字符串匹配，无大小写之分
- 正则表达式匹配，正则表达式可以采取“命名/位置”捕捉方式产生一个新的变量以便其它地方使用，正则表达式应以 ~或~*开头来表示是否大小写敏感匹配
- 如果多个源变量可匹配，顺序为：
  - 精确最长匹配
  - 最长的前缀匹配
  - 最长的后缀匹配
  - 配置中的第一个正则被匹配
  - 缺省 default (如果没有default配置，则返回空字符串)

结果变量赋值：

- 可以是字符串赋值，也可以用其它变量为其赋值，或者混合（带插值）
- 设置map映射，并不表示立即执行，只有当访问触发“结果变量”取值时，才会实际到map中去匹配查找

一个精致的map使用案例

```
map $cookie_route $route_from_cookie {
    ~.(?P<route>w+)$ $route;
}
```

正则查找route cookie的值，并形成一个内部命名捕捉变量$route
将命名变量$route作为结果变量（$route_from_cookie）的输入值

ROUTE=iDmDe26BdBDS28FuVJlWc1FH4b13x4fn**.a**

匹配该cookie，并将取出的结果“a” 赋值给$route_from_cookie

```
http {
    # ...
    map $cookie_route $route_from_cookie {
        ~.(?P<route>w+)$ $route;
    }

    map $arg_route $route_from_uri {
        ~.(?P<route>w+)$ $route;
    }

    upstream backend {
        zone backend 64k;
        server 10.0.0.200:8098 route=a;
        server 10.0.0.201:8099 route=b;

        sticky route $route_from_cookie $route_from_uri;
    }

    server {
        listen 80;

        location / {
            # ...
            proxy_pass http://backend;
        }
    }
}
```

完整示例

Map中让结果变量取值本身具备动态性

问题： 如果一个请求的session cookie不存在，则取样1%的这样请求记录到access log里

```
map $cookie_SESSION   $logme {
           ""                 $perhaps;
          default             0;
}

split_clients $request_id  $perhaps {
                      1%     1;  # $perhaps is true 1% of the time
                       *     0;
}

server {
    listen 80;
    access_log  /var/log/nginx/secure.log notes if=$logme;
    ...
}
```

map缓存

map与geo指令类似， 在一次上请求上下文中会被缓存

```
map $args $test {
        default 0;
        debug 1;
}
```

```
location /maptest {
        set $orig_test $test;
        set $args debug;
        return 200 "original test mapping value: $orig_test ||| After changing args, the test mapping value is: $test";
}
```

① ②

curl http://172.16.100.252:8080/maptest

original test mapping value: **0** ||| After changing args, the test mapping value is： **0**

map缓存-强制刷新参数

1.11.7版本后增加了缓存控制参数 volatile

```
map $args $test {
        volatile;
        default 0;
        debug 1;
}
```

```
location /maptest {
        set $orig_test $test;
        set $args debug;
        return 200 "original test mapping value: $orig_test ||| After changing args, the test mapping value is: $test";
}
```

① ②

curl http://172.16.100.252:8080/maptest

original test mapping value: **0** ||| After changing args, the test mapping value is: **1**

Map缓存在子请求下的行为

```
map $args $test {
        #volatile;
        default 0;
        debug 1;
}
```

```
location /map_main {
        ① auth_request /map_sub?debug;
          proxy_pass http://60.205.226.20:80$test;
                                                  ④
}

location /map_sub {
          set $args "debug"; ②
          return 200 "sub request $test";
                                        ③
}
```

当存在缓存行为时候：

Proxy到800端口还是801端口？

172.16.100.252.47818 > 60.205.226.20.801: Flags [S],

Map缓存在子请求下的行为

```
map $args $test {
        volatile;
        default 0;
        debug 1;
}
```

```
location /map_main {
        ① auth_request /map_sub?debug;
          proxy_pass http://60.205.226.20:80$test;
}

location /map_sub {
          set $args "debug"; ②
          return 200 "sub request $test";
}
```

④ ③

当**不**存在缓存行为时候：

Proxy到800端口还是801端口？

172.16.100.252.37818 > 60.205.226.20.800: Flags [S]

其它重要的变量举例

Variable	Definition
$upstream_connect_time	Connection to upstream/backend
$upstream_header_time	First byte response header
$upstream_response_time	Last byte response body
$request_time	Total time of request

上述变量没有默认的加入到日志中，需要在log_format指令中手工加入

keyval （商业模块）

- 通过API操纵一个KV共享内存数据库，该库存储在一个共享内存zone中

- keyval_zone命令用于设置KV数据库

  keyval_zone zone=*name*:*size* [state=*file*] [timeout=*time*] [type=string|ip|prefix] [sync];

  - State：可用于持久化
  - Timeout： 控制kv多久从库中自动被删除
  - Type:定义索引，
  - String 默认不索引，采取精确匹配
  - Ip: ipv4/ipv6/CIDR地址索引，检索的IP精确匹配或属于某个子网
  - Prefix： 前缀匹配，KV中的K必须是待检索key的前缀
  - Sync:用于控制是否在在cluster内同步

- 通过keyval命令查找KV中的记录，获得对应的V值并存到指定的变量里

  keyval *key* *$new-variable* zone=*name;*

keyval

- 由于需要通过API控制KV数据库内容，因此首先要将NGINX的 API打开，且要可写

```
location /api {
  api write=on;
}
```

* 最小示意配置，实际还应该考虑/api的访问控制，确保API安全.

```
keyval_zone zone=cmb:64k state=cmb.keyval;
keyval abswitch $abswitchvalue zone=cmb;

if ($abswitchvalue = 0){
        set $backend A;
}
if ($abswitchvalue = 1){
        set $backend B;
}
 location / {
                return 200 "$backend";
        }
```

Keyval的API操纵

```
curl -X POST "http://172.16.100.253:8080/api/5/http/keyvals/cmb" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"abswitch\": \"0\"}"

curl -X GET  "http://172.16.100.253:8080/api/5/http/keyvals/cmb"

curl http://172.16.100.253/
A

curl -X PATCH "http://172.16.100.253:8080/api/5/http/keyvals/cmb" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"abswitch\": \"1\"}"

curl http://172.16.100.253/
B
```

NGINX API

- API介绍
- API Endpoints与Versioning
- API配置持久化
- 利用API与F5结合实现更好的LB体验

NGINX API

前提：在配置中启用 api location

API作用：

- On the fly 改变配置，免reload配置
- Devops融入
- 状态监控、统计信息、配置管理

http://demo.nginx.com/

http://demo.nginx.com/swagger-ui/

```
server {
        listen 8080;
        set $apimgmt_entry_point -;
        location /api {
            api write=on;
        }

        location = /dashboard.html {
            root /usr/share/nginx/html;
        }

        location  /swagger-ui {
           root /usr/share/nginx/html;
        }

    }
```

ngx_http_api_module

Endpoints

- /
- /nginx
- /processes
- /connections
- /slabs/
- /slabs/{slabZoneName}
- /http/
- /http/requests
- /http/server_zones/
- /http/server_zones/{httpServerZoneName}
- /http/location_zones/
- /http/location_zones/{httpLocationZoneName}
- /http/caches/
- /http/caches/{httpCacheZoneName}
- /http/limit_conns/
- /http/limit_conns/{httpLimitConnZoneName}
- /http/limit_reqs/
- /http/limit_reqs/{httpLimitReqZoneName}
- /http/upstreams/
- /http/upstreams/{httpUpstreamName}/
- /http/upstreams/{httpUpstreamName}/servers/
- /http/upstreams/{httpUpstreamName}/servers/{httpUpstreamServerId}
- /http/keyvals/
- /http/keyvals/{httpKeyvalZoneName}
- /stream/
- /stream/server_zones/
- /stream/server_zones/{streamServerZoneName}
- /stream/limit_conns/
- /stream/limit_conns/{streamLimitConnZoneName}
- /stream/upstreams/
- /stream/upstreams/{streamUpstreamName}/
- /stream/upstreams/{streamUpstreamName}/servers/
- /stream/upstreams/{streamUpstreamName}/servers/{streamUpstreamServerId}
- /stream/keyvals/
- /stream/keyvals/{streamKeyvalZoneName}
- /stream/zone_sync/
- /resolvers/
- /resolvers/{resolverZoneName}

API Versioning

API接口采用版本化，新的功能会加入到新的版本下，保持向后的兼容性

调用接口时需指明版本　curl http://172.16.100.253:8080/api
[1,2,3,4,5,6]

http://172.16.100.253:8080/api/6/slabs

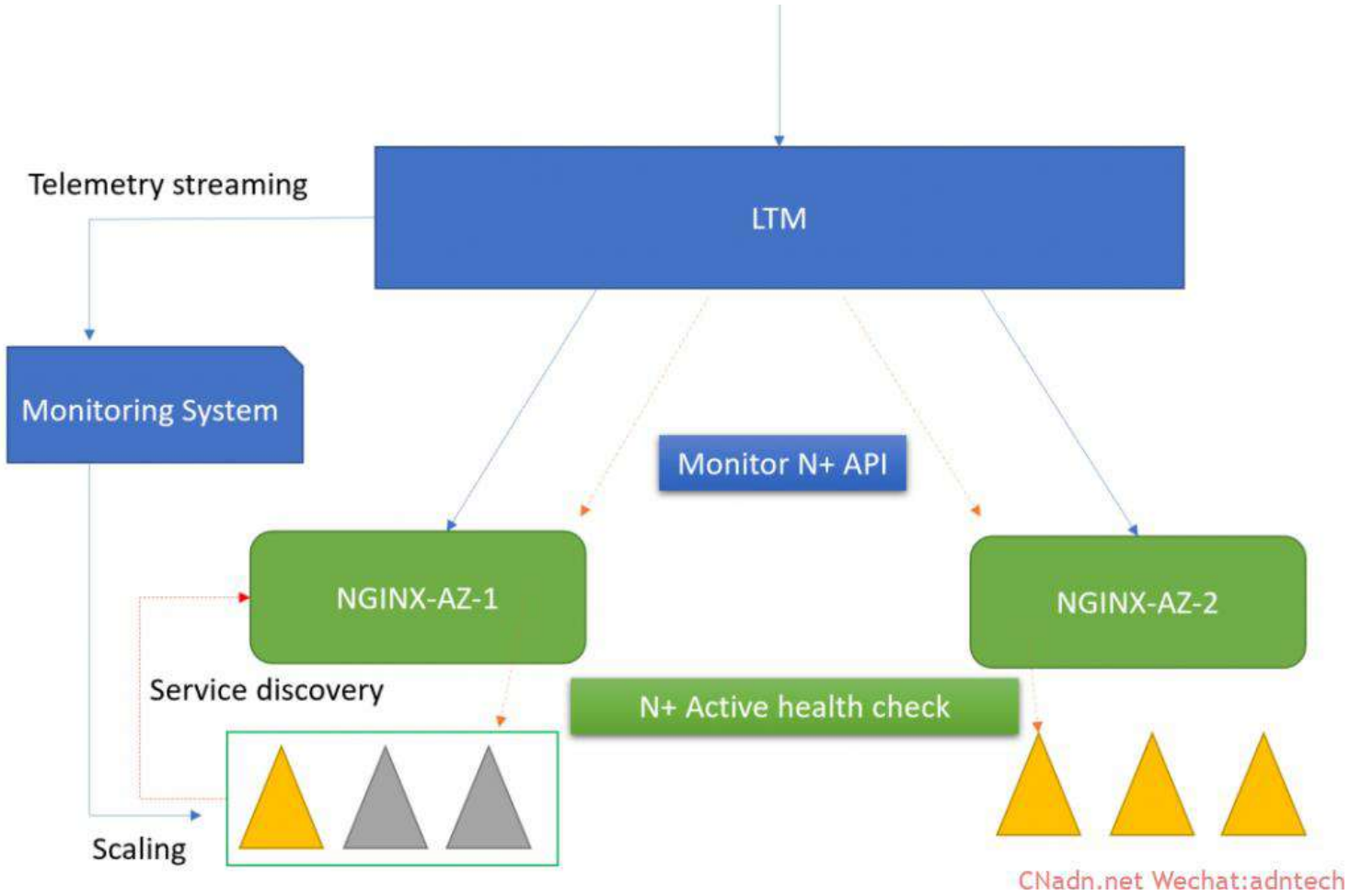
- The /stream/limit_conns/ data were added in version 6.
- The /http/limit_conns/ data were added in version 6.
- The /http/limit_reqs/ data were added in version 6.
- The "expire" parameter of a key-value pair can be set or changed since version 5.
- The /resolvers/ data were added in version 5.
- The /http/location_zones/ data were added in version 5.
- The path and method fields of nginx error object were removed in version 4. These fields continue to exist in earlier api versions, but show an empty value.
- The /stream/zone_sync/ data were added in version 3.
- The drain parameter was added in version 2.
- The /stream/keyvals/ data were added in version 2.

API配置持久化

```
stream {
    upstream dns {
        zone dns_zone 64k;
        #server 172.16.10.203:53              max_fails=3 fail_timeout=30s;
        state /var/lib/nginx/state/stream_upstream_dns.conf;
    }
```

```
[root@plus1 state]# pwd
/var/lib/nginx/state
[root@plus1 state]# ll
total 4
-rw-r--r--. 1 nginx nginx 70 Feb 14 10:15 stream_upstream_dns.conf
[root@plus1 state]# cat stream_upstream_dns.conf
server 172.16.10.203:53 max_conns=100 fail_timeout=15s slow_start=5s;
```

避免NGINX后server过载




```python
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import sys
import urllib2
import json

def get_nginxapi(url):
    ct_headers = {'Content-type':'application/json'}
    request = urllib2.Request(url,headers=ct_headers)
    response = urllib2.urlopen(request)
    html = response.read()
    return html

api = sys.argv[3]

try:
    data = get_nginxapi(api)
    data = json.loads(data)
except:
    data = ''
m = 0
lowwater = int(sys.argv[4])
try:
    for peer in data['peers']:
        state = peer['state']
        if state == 'up':
            m = m + 1
except:
    m = 0
#print data['peers'][]['state']
#print m
if m >= lowwater:
    print 'UP'
```

NGINX


f5
N
Thank You

关注我们

F5 官方微信
(新闻，技术文章)

F5社区
(答疑，吐槽，分享，互动)

NGINX技术群

加入F5社区：关注“F5社区”微信公众号，注册成为社员，随时参加meet up活动，代码共享，讨论，答疑等。只要你有想法，有创造，那么就快来大展身手吧，让我们在社区里尽情分享，交流，吐槽和互动，在这个自由的国度里，发现闪亮的自己。让我们一起来见证“一群有才能的人在一起做有梦想的事！

操作步骤：

1. 扫描二维码并在“入群信息”栏填写姓名
2. 点击下方“我要入群”
3. 长按识别二维码进入群聊