

]

Introduction to nginx.conf scripting

Introduction to **nginx.conf** *scripting*

☺ *agentzh@gmail.com* ☺

章亦春 (*agentzh*)

2010.4

```
$ nginx -c /path/to/nginx.conf
```



```
$ ps aux | grep nginx
```

```
root  2003  0.0  0.0 25208  412 ? Ss 10:08 0:00 nginx: master process nginx
```

```
nobody 2004  0.0  0.0 25608 1044 ? S  10:08 0:00 nginx: worker process
```

```
nobody 2005  0.0  0.0 25608 1044 ? S  10:08 0:00 nginx: worker process
```

```
# nginx.conf
worker_processes 2;
events {
    worker_connections 1024;
}
http {
    ...
    server {
        listen 80;
        server_name localhost;
        ...

        location / {
            root /var/www;
            index index.html index.htm;
        }
    }
}
```

♡ *Hello World* on the *nginx* land

```
00100000000010100011011000000100101100011
110001011101000100011111111110100000100
00101001011000011010111011010110110010001
01101100000101011001000100001110001001111
0100110010110100110110100111101111011110
000110100010001110100011000101101000011010
0100100110100001010001010001110
10001001int main()
010101001{
1110011000    printf("Hello World");
001000001111111111111111111111111111111111
00011010001000111010010110001101000011010
01001001101111010111011110000001010001110
1000100100010101100100111011101000101111
01010100111001101010111000101010100011000
1110011000001101111110101001111110001100
00100000111111101010010010011010101110110
```

enable the ngx_echo module in your nginx build

**\$./configure --prefix=/opt/nginx **
--add-module=/path/to/echo-nginx-module

```
location = '/hello' {  
    echo "hello, world!";  
}
```

```
$ curl 'http://localhost/hello'  
hello, world!
```

♥ Introducing *parameterized* hello

```
location = '/hello' {  
    echo "hello, $arg_person!";  
}
```

```
$ curl 'http://localhost/hello?person=agentzh'  
hello, agentzh!
```

```
$ curl 'http://localhost/hello'  
hello, !
```

♥ Add a *default* value to the *person* parameter

```
location = '/hello' {  
    if ($arg_person = "") {  
        echo "hello, anonymous!";  
        break;  
    }  
    echo "hello, $arg_person";  
}
```

```
$ curl 'http://localhost/hello?person=agentzh'  
hello, agentzh!
```

```
$ curl 'http://localhost/hello'  
hello, anonymous!
```

♥ ...or *avoid* using the **if** statement

```
# enable the ngx_set_misc module and  
# Marcus Clyne's ngx_devel_kit in your nginx build  
$ ./configure --prefix=/opt/nginx \  
    --add-module=/path/to/echo-nginx-module \  
    --add-module=/path/to/ngx_devel_kit \  
    --add-module=/path/to/set-misc-nginx-module
```

```
location = '/hello' {  
    set $person $arg_person;  
    set_if_empty $person 'anonymous';  
    echo "hello, $person!";  
}
```

```
$ curl 'http://localhost/hello?person=agentzh'  
hello, agentzh!
```

```
$ curl 'http://localhost/hello'  
hello, anonymous!
```

♡ Some *UTF-8* love in the **person** parameter?



sigh...

\$ curl 'http://localhost/hello?person=%E7%AB%A0%E4%BA%A6%E6%98%A5'

hello, %E7%AB%A0%E4%BA%A6%E6%98%A5

♥ Let's *fix* it using the `set_unescape_uri` directive!

```
location = '/hello' {  
    set_unescape_uri $person $arg_person;  
    set_if_empty $person 'anonymous';  
    echo "hello, $person!";  
}
```

Yay!

\$ curl 'http://localhost/hello?person=%E7%AB%A0%E4%BA%A6%E6%98%A5'

hello, 章亦春

♡ Nginx *variables* are very powerful, but how about *arrays*?

enable the ngx_array_var module in your nginx build

```
$ ./configure --prefix=/opt/nginx \  
    --add-module=/path/to/echo-nginx-module \  
    --add-module=/path/to/ngx_devel_kit \  
    --add-module=/path/to/set-misc-nginx-module \  
    --add-module=/path/to/array-var-nginx-module
```

```
location ~ '^/foo/(.*)' {
```

```
    set $list $1;
```

```
    array_split ',' $list;
```

```
    array_map '[$array_it]' $list;
```

```
    array_join ' ' $list;
```

```
    echo $list;
```

```
}
```

```
$ curl 'http://localhost/foo/Bob,Marry,John'  
[Bob] [Marry] [John]
```

♥ Using *subrequests* to do mashup

```
location = '/merge' {  
    echo '[';  
    echo_location_async /moon;  
    echo ',';  
    echo_location_async /earth;  
    echo ']';  
}  
location /moon {  
    echo '"moon"';  
}  
location /earth {  
    echo '"earth"';  
}
```

```
$ curl 'http://localhost/merge'
```

```
[
```

```
"moon"
```

```
,
```

```
"earth"
```

```
]
```

♥ or even **dynamic** mashups...

```
location ~ '^/merge/(.*)' {  
    set $list $1;  
    echo '[';  
    echo_foreach_split ',' $list;  
        echo_location_async "$echo_it";  
        echo ",";  
    echo_end;  
    echo 'null';  
    echo ']';  
}
```

```
$ curl 'http://localhost/merge/earth,moon'
```

```
[  
  "earth"  
  ,  
  "moon"  
  ,  
  null  
]
```

♡ Some *non-blocking* memcached love

enable the ngx_memc module in your nginx build

**\$./configure --prefix=/opt/nginx \
--add-module=/path/to/echo-nginx-module \
--add-module=/path/to/memc-nginx-module**

(not quite) REST interface to our memcached server

at 127.0.0.1:11211

location = /memc {

set \$memc_cmd \$arg_cmd;

set \$memc_key \$arg_key;

set \$memc_value \$arg_val;

set \$memc_exptime \$arg_exptime;

memc_pass 127.0.0.1:11211;

}

```
$ curl 'http://localhost/memc?cmd=flush_all';
```

OK

```
$ curl 'http://localhost/memc?cmd=replace&key=foo&val=FOO';
```

NOT_STORED

```
$ curl 'http://localhost/memc?cmd=add&key=foo&val=Bar&exptime=60';  
STORED
```

```
$ curl 'http://localhost/memc?cmd=replace&key=foo&val=Foo';  
STORED
```

```
$ curl 'http://localhost/memc?cmd=set&key=foo&val=Hello';  
STORED
```

```
$ curl 'http://localhost/memc?cmd=get&key=foo';
```

Hello

```
$ curl 'http://localhost/memc?cmd=delete&key=foo';
```

DELETED

```
$ curl 'http://localhost/memc?cmd=flush_all';
```

OK

```
$ curl 'http://localhost/memc?cmd=incr&key=counter&val=1';
```

```
<html>
```

```
<head><title>404 Not Found</title></head>
```

```
<body bgcolor="white">
```

```
<center><h1>404 Not Found</h1></center>
```

```
<hr><center>nginx/0.8.35</center>
```

```
</body>
```

```
</html>
```

```
$ curl 'http://localhost/memc?cmd=add&key=counter&val=0';  
STORED
```

```
$ curl 'http://localhost/memc?cmd=incr&key=counter&val=1';  
STORED
```

♡ *Safe* memcached *incr* operation

```
location = /safe-incr {  
    if ($arg_key = "") {  
        return 400;  
        break;  
    }  
    if ($arg_val !~ '^\d+$') {  
        return 400;  
        break;  
    }  
    echo_exec /safe-memc?cmd=incr&key=$arg_key&val=$arg_val;  
}
```

```
location = /safe-memc {  
    internal;  
    set $memc_cmd $arg_cmd;  
    set $memc_key $arg_key;  
    set $memc_value $arg_val;  
    set $memc_exptime $arg_exptime;  
  
    memc_pass 127.0.0.1:11211;  
  
    error_page 404 = /add-and-retry;  
}
```

```
location = /add-and-retry {  
    internal;  
    echo_location /memc?cmd=add&key=$arg_key&val=0;  
    echo_location /memc?$query_string;  
}
```

```
$ curl 'http://localhost/memc?cmd=flush_all';
```

OK

```
$ curl 'http://localhost/safe-incr?key=counter&val=1';
```

STORED

STORED

♥ Memcached **connection pool** support



enable Maxim Dounin's ngx_http_upstream_keepalive module
in your nginx build

**\$./configure --prefix=/opt/nginx \
--add-module=/path/to/echo-nginx-module \
--add-module=/path/to/memc-nginx-module \
--add-module=/path/to/ngx_http_upstream_keepalive**

```
http {  
    ...  
    upstream my_memc_backend {  
        server 127.0.0.1:11211;  
  
        # a connection pool that can cache  
        #   up to 1024 connections  
        keepalive 1024 single;  
    }  
    ...  
}
```

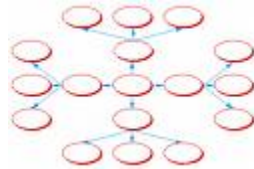
```
location = /memc {
```

```
...
```

```
memc_pass my_memc_backend;
```

```
}
```

♥ Memcached server *hashing* based on user keys
(Hey, memcached **cluster**!)



enable the ngx_set_misc module and Marcus Clyne's

ngx_devel_kit again in your nginx build

**\$./configure --prefix=/opt/nginx **

**--add-module=/path/to/memc-nginx-module **

**--add-module=/path/to/ngx_devel_kit **

--add-module=/path/to/set-misc-nginx-module

```
http {  
    upstream A {  
        server 10.32.110.5:11211;  
    }  
    upstream B {  
        server 10.32.110.16:11211;  
    }  
    upstream C {  
        server 10.32.110.27:11211;  
    }  
    upstream_list my_cluster A B C;  
    ...  
}
```

```
location = /memc {  
    set $memc_cmd $arg_cmd;  
    set $memc_key $arg_key;  
    set $memc_value $arg_val;  
    set $memc_exptime $arg_exptime;  
  
    # hashing the $arg_key to an upstream backend  
    # in the my_cluster upstream list, and set $backend:  
    set_hashed_upstream $backend my_cluster $arg_key;  
  
    # pass $backend to memc_pass:  
    memc_pass $backend;  
}
```

♡ *Capture* subrequests' responses into nginx **variables**



enable Valery Kholodkov's nginx_eval_module

in your nginx build

**\$./configure --prefix=/opt/nginx \
--add-module=/path/to/echo-nginx-module \
--add-module=/path/to/memc-nginx-module \
--add-module=/path/to/nginx_eval_module**

```
location = /save {  
    eval_override_content_type 'text/plain';  
    eval $res {  
        set $memc_cmd 'set';  
        set $memc_key $arg_id;  
        set $memc_val $arg_name;  
        memc_pass 127.0.0.1:11211;  
    }  
    if ($res !~ '^STORED$') {  
        return 500;  
        break;  
    }  
    echo 'Done!';  
}
```

♥ Use my *fork* of **ngx_eval** module to capture *arbitrary* location's response (with filters!)

<http://github.com/agentzh/nginx-eval-module>

```
location = /hello {  
    eval_override_content_type 'text/plain';  
    eval_subrequest_in_memory off;  
    eval_buffer_size 1k;  
    eval $out {  
        echo_before_body hello;  
        echo world;  
    }  
    echo "[$out]";  
}
```

```
$ curl 'http://localhost/hello'  
[hello  
world]
```

♡ Some *non-blocking* MySQL love



```
# install libdrizzle first and then
# enable the ngx_drizzle and ngx_rds_json
# modules in your nginx build
$ ./configure --prefix=/opt/nginx \
    --add-module=/path/to/drizzle-nginx-module \
    --add-module=/path/to/rds-json-nginx-module
```

```
http {  
    upstream my_mysql_backend {  
        drizzle_server 127.0.0.1:3306 dbname=test  
        password=some_pass user=monty  
        protocol=mysql;  
    }  
    ...  
}
```

```
location = /cats {  
    drizzle_query 'select * from cats';  
    drizzle_pass my_mysql_backend;  
    rds_json on;  
}
```

```
$ curl 'http://localhost/cats'
```

```
[{"name":"Jerry","age":1}, {"name":"Tom","age":3}]
```

♥ Database connection pool support



```
http {  
    upstream my_mysql_backend {  
        drizzle_server 127.0.0.1:3306 dbname=test  
            password=some_pass user=monty  
            protocol=mysql;  
  
        # a connection pool that can cache up to  
        # 200 mysql TCP connections  
        drizzle_keepalive max=200 overflow=reject;  
    }  
    ...  
}
```

♥ Mysql **cluster** *hashing* love



re-enable the ngx_set_misc module and Marcus Clyne's

ngx_devel_kit in your nginx build

**\$./configure --prefix=/opt/nginx **

**--add-module=/path/to/drizzle-nginx-module **

**--add-module=/path/to/rds-json-nginx-module **

**--add-module=/path/to/ngx_devel_kit **

--add-module=/path/to/set-misc-nginx-module

```
http {  
    upstream A {  
        drizzle_server ...;  
    }  
    upstream B {  
        drizzle_server ...;  
    }  
    upstream C {  
        drizzle_server ...;  
    }  
    upstream_list my_cluster A B C;  
    ...  
}
```

```
location ~ '^/cat/(.*)' {  
    set $name $1;  
    set_quote_sql_str $quoted_name $name;  
    drizzle_query "select *  
        from cats  
        where name=$quoted_name";  
  
    set_hashed_upstream $backend my_cluster $name;  
    drizzle_pass $backend;  
  
    rds_json on;  
}
```

♥ Highly *dynamic* SQL query **construction**

```
location ~ '^/cats/(.*)' {  
    set $list $1;  
    array_split ',' $list;  
    array_map_op set_quote_sql_str $list;  
    array_map 'name=$array_it' $list;  
    array_join ' or ' $list to=$cond;  
  
    drizzle_query "select *  
        from cats  
        where $cond";  
  
    drizzle_pass my_mysql_backend;  
    rds_json on;  
}
```

Let's do

select * from cats

where name='Jerry' or name='Tom'

\$ curl 'http://localhost/cats/Jerry,Tom'

[{"name":"Jerry","age":1},{"name":"Tom","age":3}]

♥ Piotr Sikora's *ngx_postgres* is drawing near :D



```
upstream my_pg_backend {  
    postgres_server 10.62.136.3:5432 dbname=test  
    user=someone password=123456;  
}
```

```
location /cats {  
    postgres_query 'select * from cats';  
    postgres_pass my_pg_backend;  
    rds_json on;  
}
```

♡ chaoslawful is already *working* on `ngx_lua` :D



♥ A quick *summary* of our existing modules

- ✓ **ngx_echo**: Brings "echo", "sleep", "time", "exec", *background job* and even more shell-style goodies to Nginx config file.
<http://wiki.nginx.org/NginxHttpEchoModule>
- ✓ **ngx_chunkin**: HTTP 1.1 *chunked*-encoding request body support for Nginx.
<http://wiki.nginx.org/NginxHttpChunkinModule>
- ✓ **ngx_headers_more**: Set and clear input and output *headers*...more than "add"!
<http://wiki.nginx.org/NginxHttpHeadersMoreModule>

and even more...

- ✓ **ngx_memc**: An extended version of the standard *memcached* module that supports set, add, delete, and many more memcached commands.
<http://wiki.nginx.org/NginxHttpMemcModule>
- ✓ **ngx_drizzle**: ngx_drizzlen nginx upstream module that talks to *mysql*, drizzle, and sqlite3 by libdrizzle.
<http://github.com/chaoslawful/drizzle-nginx-module>
- ✓ **ngx_rds_json**: An nginx output filter that formats Resty DBD Streams generated by ngx_drizzle and others to JSON.
<http://github.com/agentzh/rds-json-nginx-module>

Well, still continued...

- ✓ **ngx_xss**: Native support for *cross-site scripting* (XSS) in an nginx.
<http://github.com/agentzh/xss-nginx-module>
- ✓ **ngx_set_misc**: Various nginx.conf variable transformation utilities.
<http://github.com/agentzh/set-misc-nginx-module>
- ✓ **ngx_array_var**: Add support for array variables to nginx config files.
<http://github.com/agentzh/array-var-nginx-module>

♡ *New* nginx modules on our **TODO** list

- ✓ **ngx_form_input**
- ✓ **ngx_iconv**
- ✓ **ngx_srcache**
- ✓ **ngx_encrypted_session**
- ✓ **ngx_rds**
- ✓ **ngx_rds_tt2**

♡ Update: recent developments

<http://agentzh.org/misc/slides/recent-dev-nginx-conf/>

😊 *Any questions?* 😊

