Nginx HTTP Variable

# 概述

配置文件中的脚本功能，以简洁的方法增强了Nginx的功能。

# 变量数据结构

## http_variable.pngngx\_http\_variable\_t

表示一个变量，但不包含变量的值。

* name：变量名。
* set\_handler：设置变量值的函数。
* get\_handler：获取变量值的函数。
* data：设置（获取）变量值的函数的参数。
* index：在变量数组的index。
* flags：
  + changeable：变量值不是一个常量，可以改变。
  + nocacheable：不可缓存，每次都要通过get\_handler()获取变量值。
  + indexed：变量在索引数组。
  + nohash：变量在哈希表。

## ngx\_http\_variable\_value\_t

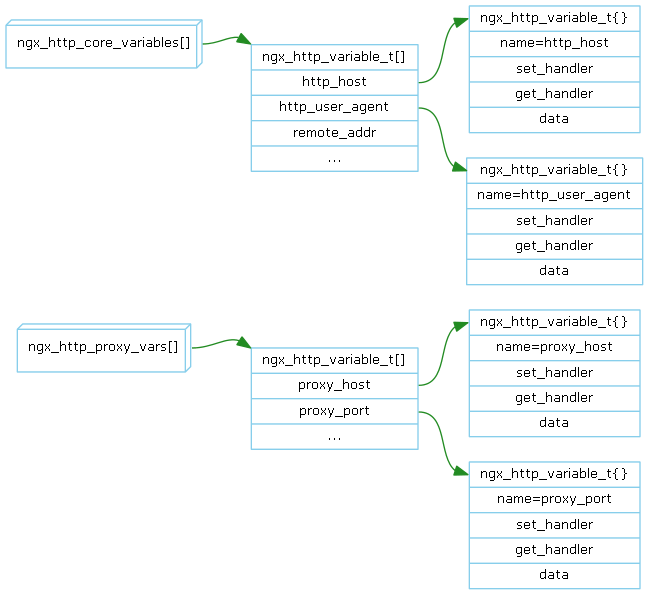
表示一个变量的值，它可以通过变量get\_handler()获取，或者通过set\_handler()设入变量。

它实际上是一个ngx\_str\_t{len, data}，只是多了4个属性。

* valid：变量值有效。
* no\_cacheable：不可缓存。
* not\_found：TODO
* escape：TODO

# 变量定义

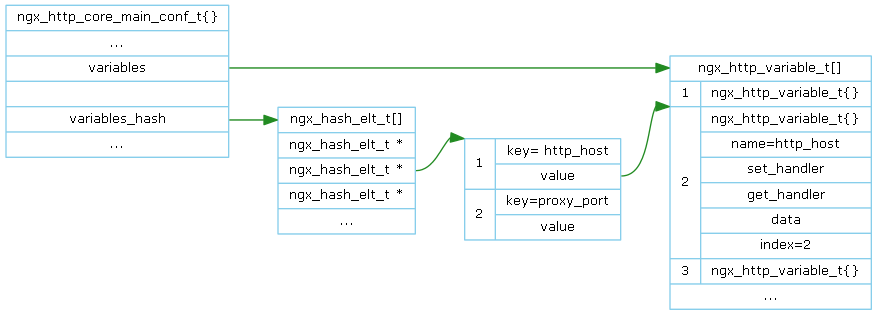
## 内建变量

* ngx\_http\_core\_variables：http\_core模块所定义的变量。
* ngx\_http\_proxy\_vars：http\_proxy模块所定义的变量。
* ......

## 外部变量

set配置指令或者map配置指令。

## 变量组织

* variables\_hash：为了快速由变量名查找到ngx\_http\_variable\_t，把已定义的变量建立哈希表。
* variables：将已定义变量加入该索引数组。

变量有三类组织方式：

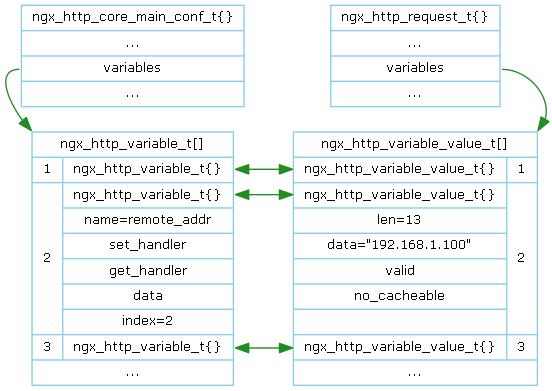
1. HASH-ED INDEX-ED ：既在哈希表，也在索引数组。
2. HASH-ED NO-INDEX ：在哈希表，但不在索引数组。
3. NO-HASH INDEX-ED ：不在哈希表，但在索引数组。

# 变量访问

变量是和Request相关联的。Request是变量的宿主。比如说$remote\_addr变量是某个Request的$remote\_addr。

## HASH-ED INDEX-ED

HASH-ED INDEX-ED变量由ngx\_http\_core\_main\_conf\_t的variables\_hash和variables管理，而变量值则由ngx\_request\_t的variables管理。



ngx\_http\_variable\_value\_t数组和ngx\_http\_variable\_t数组是一对并列数组，创建ngx\_http\_request\_t时同时也被创建。

ngx\_http\_variable\_value\_t数组的第N个是ngx\_http\_variable\_t数组第N个变量的值。

ngx\_http\_variable\_value\_t \* ngx\_http\_get\_variable(ngx\_http\_request\_t \*r, name, key)函数是获取某个变量的变量值函数。

下面以读变量$remote\_addr值为例，作分析：

1. 根据变量名“remote\_addr”在variables\_hash哈希表中找到ngx\_http\_variable\_t。
2. 根据ngx\_http\_variable\_t的index，调用ngx\_http\_get\_indexed\_variable()得到ngx\_http\_variable\_value\_t。
   1. 此时ngx\_http\_variable\_value\_t的valid为0，即无效数据，调用ngx\_http\_variable\_t的get\_handler()
   2. ngx\_http\_variable\_remote\_addr()从ngx\_request\_t取得ngx\_connection\_t对象从而取得remote address。
   3. 将remote address写入ngx\_http\_variable\_value\_t并设valid为1。
   4. 返回ngx\_http\_variable\_value\_t。

此后如果再读该变量的值，因为ngx\_http\_variable\_value\_t的valid为1，即有效，则无需get\_handler()，而直接返回。

## HASH-ED NO-INDEX

HASH-ED NO-INDEX变量只由ngx\_http\_core\_main\_conf\_t的variables\_hash管理，变量值不由ngx\_request\_t的variables管理。

TODO：举例。

下面以读变量$xxxxxxx值为例，作分析：

1. 根据变量名“xxxxxxx”在variables\_hash哈希表中找到ngx\_http\_variable\_t。
2. 分配ngx\_http\_variable\_value\_t。
3. 调用ngx\_http\_variable\_t的get\_handler取得ngx\_http\_variable\_value\_t。

## NO-HASH INDEX-ED

NO-HASH INDEX-ED变量由ngx\_http\_core\_main\_conf\_t的variables管理，变量值则由ngx\_request\_t的variables管理。

没有被查找需求的变量（调用者持有ngx\_http\_variable\_t指针），可以使用此类组织方式，以减小哈希表规模，提高查找效率。

例如：$invalid\_referer

下面以读变量$invalid\_referer值为例，作分析：

1. 已知ngx\_http\_variable\_t，即已知索引数组的index。
2. 调用ngx\_http\_get\_indexed\_variable(index)得到ngx\_http\_variable\_value\_t。
3. 根据index从ngx\_request\_t::variables得到ngx\_http\_variable\_value\_t。
4. 此时ngx\_http\_variable\_value\_t的valid为0，即无效数据，调用ngx\_http\_variable\_t的get\_handler()。
5. ngx\_http\_referer\_value()根据Referer头和配置数据计算结果。
6. 将结果写入ngx\_http\_variable\_t并设valid为1。
7. 返回ngx\_http\_variable\_value\_t。

# 变量表达式

index\_module模块可以指定默认访问的文件名。参见<Nginx HTTP Index>。

location / {

index home.$http\_host.html index.html;

}

文件名中可以包含变量。home.$http\_host.html 就是一个变量表达式（Complex Value）。现在开始讨论如何计算这个变量表达式。

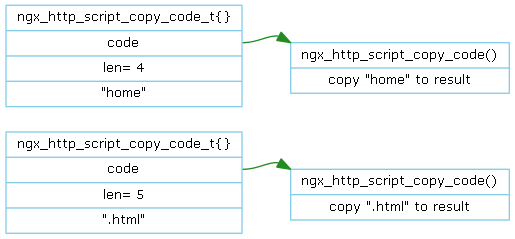
## Script Code

home.$http\_host.html可以分为三部分：

1. home
2. $http\_host
3. .html

这个变量表达式最终值由这三部分的值组合而成。第1、3部分的值为原值。第二部分的值可通过前面讲的<变量访问>求得。

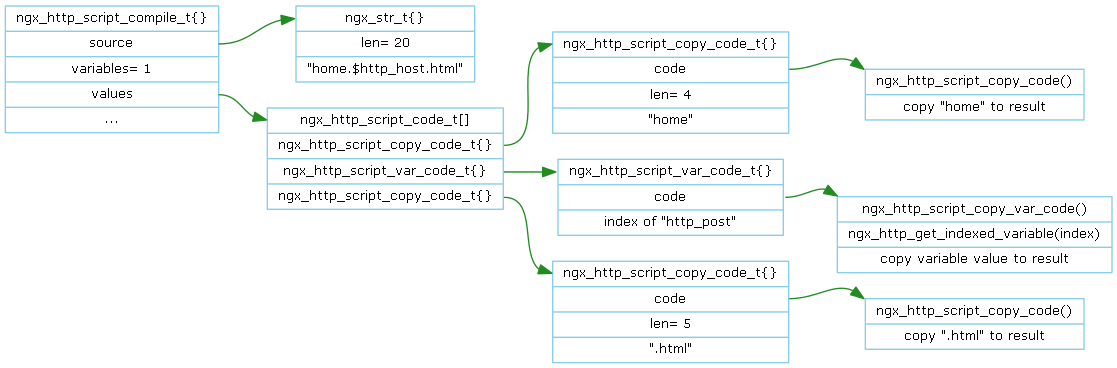
* ngx\_http\_script\_copy\_code\_t：这个数据结构用来描述第1、3部分。



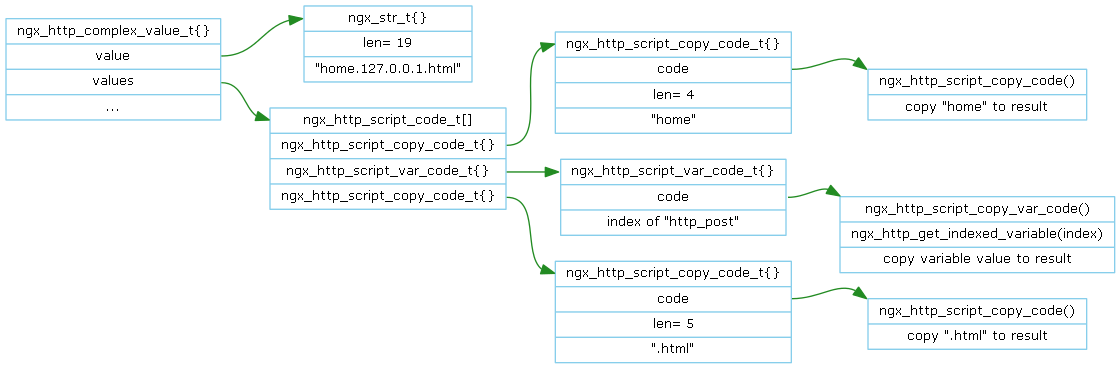
* ngx\_http\_script\_var\_code\_t：这个数据结构用来描述第2部分。

## http_script_code_var.pngCompile

编译的过程是将“home.$http\_host.html”解析出三个Script Code的过程(ngx\_http\_script\_compile()函数)。



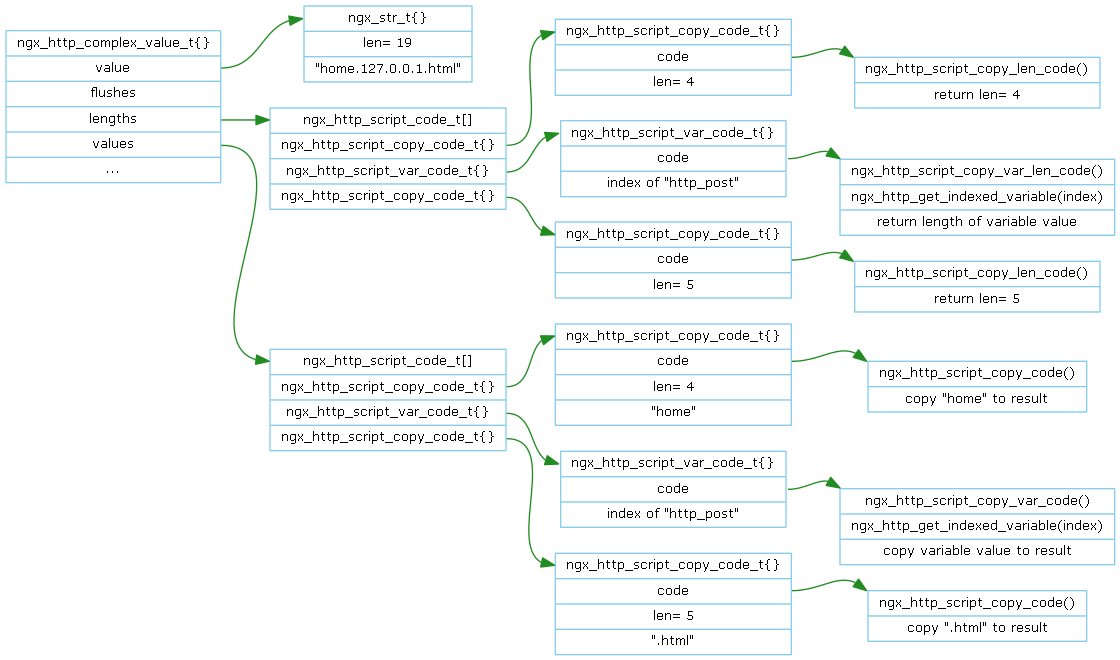
## Complex Value



ngx\_http\_complex\_value()函数求得complex\_value的最终值。它依次调用Script Code数组中的回调函数，将各个部分的字符串追加到value，value即为最终值。

不过还有一个问题：要给最终值value预先分配多少空间呢？

编译阶段是无法获知最终值的大少的，只能在运行时获知。于是Compile时要生成一个用于计算length的Script Code数组。



# 脚本引擎

if ($request\_method = POST) {

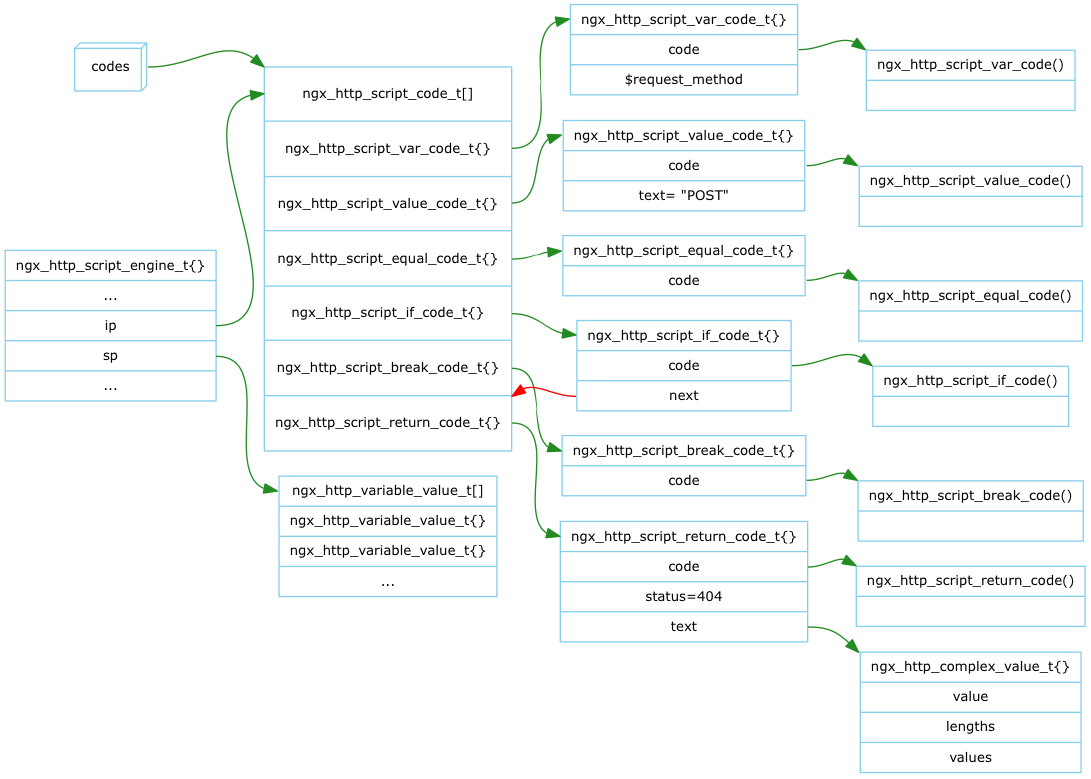
break;

}

return 404 ${http\_host}\_only\_support\_post;

## 数据结构

配置过程将上述脚本编译成下图codes所指的Script Code集合。



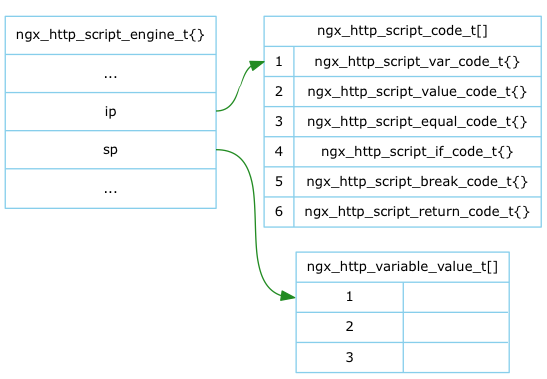
## 运行引擎

ngx\_http\_script\_engine\_t是脚本的一个引擎（解释器实例）。它采用Virtual Stack Machine方式实现。

ip指向下一条要执行的指令。sp指向数据Stack顶部。

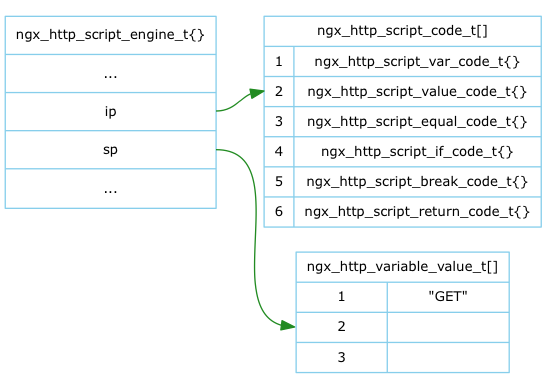
### Initial

初始情况下，ip/sp的状态：



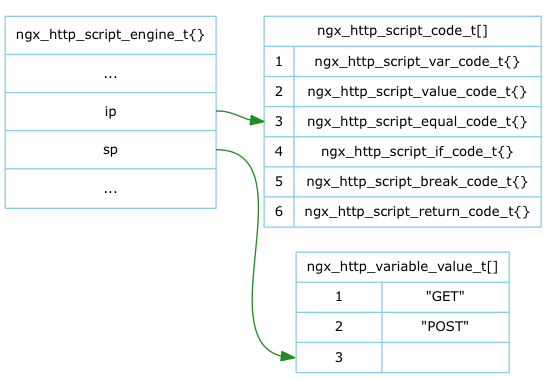
### ngx\_http\_script\_var\_code()

读取变量$request\_method的值，并压入Stack。执行完后ip/sp的状态：



### ngx\_http\_script\_value\_code()

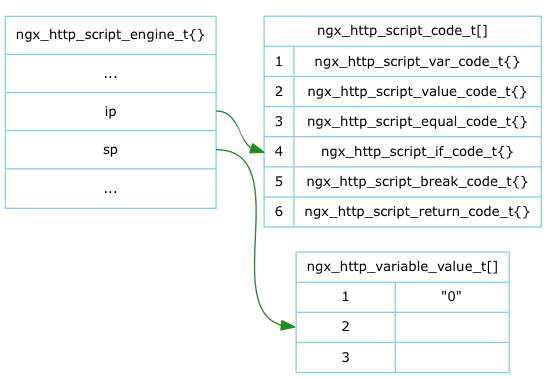
将“POST”压入Stack。执行完后ip/sp的状态：



### ngx\_http\_script\_equal\_code()

从Stack中弹出二个值并作比较，如果相等，则压“1”到Stack，否则压“0”到Stack。约定“1”表示true，“0”表示false。

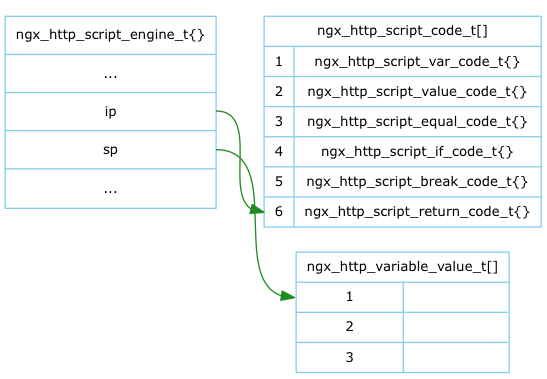
执行完后ip/sp的状态：



### ngx\_http\_script\_if\_code()

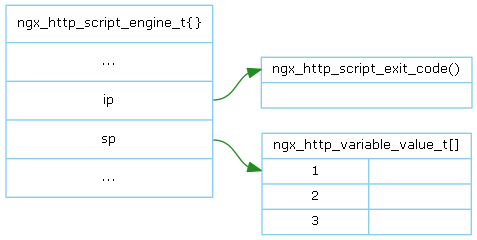
从Stack中弹出一个值，如果该值为真则ip指向连续的下一条指令（break），否则ip指向ngx\_http\_script\_if\_code\_t::next所指的指令（return）。

执行完后ip/sp的状态：



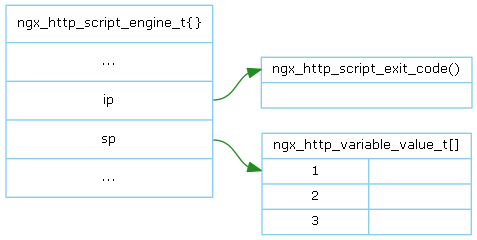
### ngx\_http\_script\_break\_code()

没有被执行。假设Request为POST，则执行完后ip/sp的状态：



### ngx\_http\_script\_return\_code()

发送404错误码的HTTP Response。执行完后ip/sp的状态：



### ngx\_http\_script\_exit\_code()

遍历Script Code数组结束，脚本引擎退出。

## 阶段处理

脚本代码位于Location之中，配置过程将脚本代码编译为Script Code数组并由Location管理。

阶段处理模块的回调处理函数中运行引擎。

# Prototype

暂无

# Latest revision

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