实例分析 set 命令及相关脚本引擎

Nginx 的 ngx_http_rewrite_module 模块使用了 Nginx 的脚本引擎，提供了外部变量的功能，主要是通过 set 命令进行定义。在 Nginx 中，什么是外部变量，什么是内部变量呢？简单来说，内部变量就是在 Nginx 的代码中定义的，外部变量就是在 nginx.conf 配置文件中定义的，并且在文件中确定了变量的赋值。其实 Nginx 中对变量的定义的语义和 C 语言中的对变量的声明的语义是比较接近的，因为这个时候并没有为这个变量分配用于存储变量值的内存，只是说明了有这么个变量存在。

外部变量在 Nginx 启动的时候通过脚本引擎被编译为了 C 代码，但是它们是在请求处理过程中才被执行和生效的。因此，外部变量一般是通过如下步骤进行设计的：

1、 Nginx 启动时将配置文件中的 set 脚本式配置编译为相关的数据和执行方法。

2、 接收到 http 请求时，在 NGX_HTTP_SERVER_REWRITE_PHASE 和 NGX_HTTP_REWRITE_PHASE 阶段中查找匹配了的 location 下是否有待执行的脚本，如果有则依次执行。

下面先来看下变量及脚本引擎涉及到的一些相关的数据结构，理解这些数据结构的定义有助于代码的分析：

先来看下变量涉及到的相关数据结构。我们知道，变量通常由变量名和变量值组成，因此变量值和变量名都需要一个数据结构他对其进行描述。对于同一个变量名，随着场景的不同而具有不同的值，即变量值随着请求不同而不同（当然并不排除不同请求的部分或全部变量具有相同的值）。但是，对于不同的请求，变量名是一样的，只是变量值不同而已，因此，在 Nginx 内部全局只有一个用来保存所有变量名的地方，即 ngx_http_core_main_conf_t 结构体中。而对于变量值来说，因为其对于每个请求而不尽相同，因此存储变量值的地方应该在请求的结构中。对于上述描述中涉及到的结构体或相关成员分别介绍如下：

1、 变量定义结构体 ngx_http_variable_t

```
struct ngx_http_variable_s {
    ngx_str_t    name;    /*变量名，但不包括前置的$符号*/
    ngx_http_set_variable_pt    set_handler;    /*如果需要变量在最初赋值的
    时候进行变量值设置，需实现该方法*/
     ngx_http_get_variable_pt    get_handler;    /*每次获取变量值时会调用该
方法*/
     uintptr_t    data;    /*作为 get_handler 或者 set_handler 方法的参数*/
     ngx_uint_t    flags;    /*变量的特性，如值可变、索引化、不缓存、不 hash*/
     ngx_uint_t    index;    /*变量值在请求的缓存数组中的索引值*/
};
```

在上面这个结构体中，我们可以看到有两个用于变量解析的方法，分别为 set_handler 和 get_handler。其中的 data 成员即作为这两个解析方法的其中一个参数，用于传递必要的信息。如果一个变量被使用模块进行了索引化，则 index 成员描述的意思就是该变量在索引数组中的下标，通过该下标可以直接获得变量名结构体。另外，flags 成员描述的是这个变量相关的一些属性，如值可变、索引化、不缓存和不 hash 等，其属性定义如下：

```
#define NGX_HTTP_VAR_CHANGEALBE          1
#define NGX_HTTP_VAR_NOCACHAEABLE        2
#define NGX_HTTP_VAR_INDEXED             4
#define NGX_HTTP_VAR_NOHASE              8
```

2、 变量值结构体 ngx_http_variable_value_t

```
typedef struct {
    unsigned len:28; /*变量值必须是在一段连续内存中存放，len 表示长度*/
    unsigned valid:1; /*valid 为 1，表示该变量是被解析过的，且数据可用*/
    unsigned no_cacheable:1; /*为 1 表示该变量不可以被缓存*/
    unsigned not_found:1;/*表示该变量被解析过，但没有解析到相应的值*/
    unsigned escape:1;
    u_char *data; /*指向变量值所在内存起始地址，与 len 成员配合使用*/
} ngx_variable_value_t;
```

上面结构体中的 data 成员和 len 成员共同描述了变量值，其余成员主要是描述了变量值的一些属性。

3、 存储变量名的数据结构

在上文中我们说过，变量名存储在全局唯一的 ngx_http_core_main_conf_t 结构体对象中，具体说是存放在 ngx_http_core_main_conf_t 中的如下三个成员中：

```
typedef struct {
    ……
    ngx_hash_t    variables_hash; /*存储变量名的 hash 表*/
    ngx_array_t    variables;     /*存储索引过的变量的数组*/
    ngx_hash_keys_arrays_t    *variables_keys;    /*用于构造 variables_hash 散列表*/
    ……
} ngx_http_core_main_conf_t;
```

对于上面的三个结构体成员，分别介绍如下：

1) variable_keys 当 Nginx 解析配置正常结束时，所有的变量都被集中在 variables_keys 中。
2) variables_hash variables_keys 中的变量，除了显示说明变量不能进行 hash 的变量，其余变量都会进行 hash，以加快访问速度，variables_hash 就是最终构造出来的变量 hash 表。
3) variables variables 数组中存储的是被使用模块索引化了的变量。其中，被索引的变量必须是定义过的，其合法性校验就是利用其是否在 variables_keys 数组中来判断的。

4、 存储变量值的数据结构

```
struct ngx_http_request_s {
    ngx_http_variable_value_t *variables; //variables 存储的是索引化的变量值
}
```

变量值如果可以被缓存，那么它一定只能存放在每一个 http 请求内，因为我们知道变量值的生命周期和一个请求是一样的。所以变量值就存放在 ngx_http_request_s 内的 variables 数组内。可以简单认为，Nginx 在解析配置文件过程中遇到的所有变量都会加入到 r->variables 数组中。当 http 请求刚到达 Nginx 的时候，就会创建存放变量值的 variables 数组。这里的 r->variables 数组和存放

变量名的数组 cmcf->variables 是一一对应的，即两者对应元素一起构成键值对。

看完变量相关的数据结构后，再一起看下 set 命令涉及到的脚本引擎相关的数据结构。

1、脚本引擎上下文

```
typedef struct {
    u_char    *ip;   //指向待执行的脚本指令。
    u_char    *pos; //指向下面的 buf，即最后变量的值存放的地方
    ngx_http_variable_value_t    *sp;   //变量值构成的栈
    ngx_str_t    buf; //解析一个复杂值参数变量时，最后解析的结果存放在此
    unsigned    skip:1;   //置 1 表示不需要拷贝数据，直接跳过数据拷贝步骤
    ngx_int_t    status;    //脚本引擎执行状态
    ngx_http_request_t    *request;   //指向脚本引擎所属的 http 请求
    ……
} ngx_http_script_engine_t;
```

同一段脚本被编译进 Nginx 中，在不同的请求里执行的效果是完全不同的（因为变量值的生命周期和请求是一样的），所以每个请求都必须有其自己的脚本执行上下文。其中 ip 指向的是待执行的脚本指令，sp 是解析变量值的时候临时存放变量值的地方，待解析变量名的时候会从 sp 中获取对应变量值，存放在 r->variables 数组中。buf 成员是用来解析复杂值参数变量的时候用来存放变量值的地方，pos 成员指向其中。

3、 编译变量名的结构体

```
typedef struct {
    ngx_http_script_code_pt    code; //code 指向获取变量名的脚本指令方法
    uintptr_t    index;        //r-variables 数组中的索引号
} ngx_http_script_var_code_t;
```

4、 编译变量值的结构体

```
typedef struct {
    ngx_http_script_code_pt    code;    //code 指向获取变量值的脚本指令
    uintptr_t    value;   //外部变量值如果为整数，则转为整数后赋值给 value，
否则 value 为 0
    uintptr_t    text_len;   //外部变量值(set 的第二个参数)的长度
    uintptr_t    text_data; //外部变量值的起始地址
} ngx_http_script_value_code_t;
```

5、 编译复杂变量值的结构体

```
typedef struct {
  ngx_http_script_code_pt code; //code 指向编译复杂变量值的脚本指令方法
  ngx_array_t    *lengths; //lengths 存放的是复杂变量值中内嵌变量的值长度
} ngx_http_script_complex_value_code_t;
```

6、 存放编译所有编译脚本的结构体

我们知道 ngx_http_script_engine_t 是随着 http 请求到来时才创建的，所以它无法保存 Nginx 启动时就编译出来的脚本。保存编译出来的脚本这个工作实际上是由 ngx_http_rewrite_loc_conf_t 结构体来承担的，如下所示：

```
typedef struct {
    ngx_array_t    *codes;   //保存着所属 location 下的所有编译后脚本
```

```
    ngx_uint_t       stack_size;   //变量值栈 sp 的大小
    ……
} ngx_http_rewrite_loc_conf_t;
```

ngx_http_rewrite_loc_conf_t 其实就是 rewrite 模块在 location 级别下面的配置结构体。如果匹配的 location 下面没有脚本式配置，则 codes 成员是空的，否则 codes 成员就会放置承载着解析后的脚本指令的结构体。

到这里，set 命令涉及到的相关数据结构就简单介绍完了，下面将结合具体的例子对代码中的处理流程进行简单分析。下文将以一条具有代表性的 set 配置命令进行代码层面上的分析，配置如下：

```
location /image/ {
    set $key “${value}test”
}
```

当解析到 set 命令相关的配置项时，就会调用 set 命令实现的回调函数：ngx_http_rewrite_set。在这个函数中就会将 set 命令及其参数进行编译成脚本。待请求匹配到对应的 location 并执行到 NGX_HTTP_SERVER_REWRITE_PHASE 或者 NGX_HTTP_REWRITE_PHASE 阶段的时候，就会执行编译成的脚本。函数 ngx_http_rewrite_set 的执行流程如下：

1、 检查 set 命令配置的合法性。具体做法就是检查 set 命令的第一个参数是否以“$”符号开头。在 Nginx 配置文件中，变量都是以”$”字符开头的，还有可选的用于将变量名括起来的大括号--”{}”。
2、 向 Nginx 内核添加这个外部变量。具体做法就是调用 ngx_http_add_variable()这个函数，将变量名字加入到全局唯一的 ngx_http_core_mian_conf_t 结构体的 variables_keys 成员中。我们知道，ngx_http_core_mian_conf_t 结构体的 variables_keys 成员会收集 Nginx 中出现的所有变量，不管是内部还是外部变量。另外，外部变量是允许赋值的，因此在调用 ngx_http_add_variable()这个函数时，其变量属性必须是值可变的 NGX_HTTP_VAR_CHANGEABLE。
3、 将外部变量索引化。具体做法就是调用 ngx_http_get_variable_index()函数将变量加入到将变量名字加入到全局唯一的 ngx_http_core_mian_conf_t 结构体的 variables 成员中。因为从逻辑上来讲，既然定义了外部变量，那么一般来说都是会使用的，因此将其索引化，加快访问速度。
4、 对于不是 5 类特殊变量的外部变量，需要设置变量的 get_handler 回调函数和索引化下标。通常来说，内部变量的 get_handler 回调是必须要实现的，因为只有读取到这个变量的时候才会调用 get_handler 去获取内部变量值。但是外部变量是每次 set 都会立即赋值的，获取变量值的时候是直接从请求的 r->variables 数组中获取的，此时的 get_handler 用处不大，但是为什么又要设置外部变量的 get_handler 回调呢？因为可能有些模块会在 set 脚本执行前就使用到这个外部变量，此时外部变量值是不存在的，即缓存在 r->variables 中的值是空的，此时会调用 get_handler 求值。故外部变量的 get_handler 也不能为 NULL，将其设置为 ngx_http_variable_null_value，即设置为空字符串值。
5、 解析变量值。因为变量值可以是纯字符串的简单值参数，也可以是包含其他变量的复杂值参数。这两种值参数的编译过程是不同的，需要区别对

待。本例中的便是属于复杂值参数类型的处理。ngx_http_rewrite_value 函数的处理流程如下：

1） 判断值参数是复杂值参数还是简单值参数。具体做法就是遍历变量值参数字符串，看其中是否出现了”$”字符并统计其个数。

2） 如果值参数中”$”个数为 0，表明是简单值参数。那么在 lcf->codes 数组中申请存放编译简单值参数的指令性结构体 ngx_http_script_value_code_t 所需的内存，并设置结构体的值。

3） 如果值参数中”$”个数不为 0，，表明是复杂值参数。这个也是上述实例将会走的处理流程。首先向 lcf->codes 数组中申请用于存放编译复杂值参数的指令性结构体 ngx_http_script_complex_value_code_t 所需内存，并设置相应的成员。然后根据复杂值参数中具体内容进行分类编译，具体就是按照值参数内容对内含变量和普通字符串进行按序编译。此部分流程可以结合下面的实例布局图进行理解。

6、 编译变量名。此时有两种情况，一、编译一个想通过 set 命令对其值进行更改内部变量；二、编译一个外部变量。

1） 编译一个想通过 set 命令对其值进行更改内部变量。如果一个内部变量，在 Nginx 的处理过程中希望可以通过 set 命令改变其值，则在实现该内部变量的时候，会实现 set_handler 回调方法，当执行到 set 命令的时候，解析变量值的时候会调用 set_handler 回调将配置文件中设置的值赋给内部变量。set_handler 方法是有内部变量定义过的，因此其不需要索引下标也可以找到对应的变量。

2） 编译一个外部变量。向 lcf->codes 数组中申请用于存放编译变量名的指令性结构体所需要的空间。设置编译指令性结构体方法和索引化下标。

下面是上述配置项相关数据结构间的内存布局示意图：

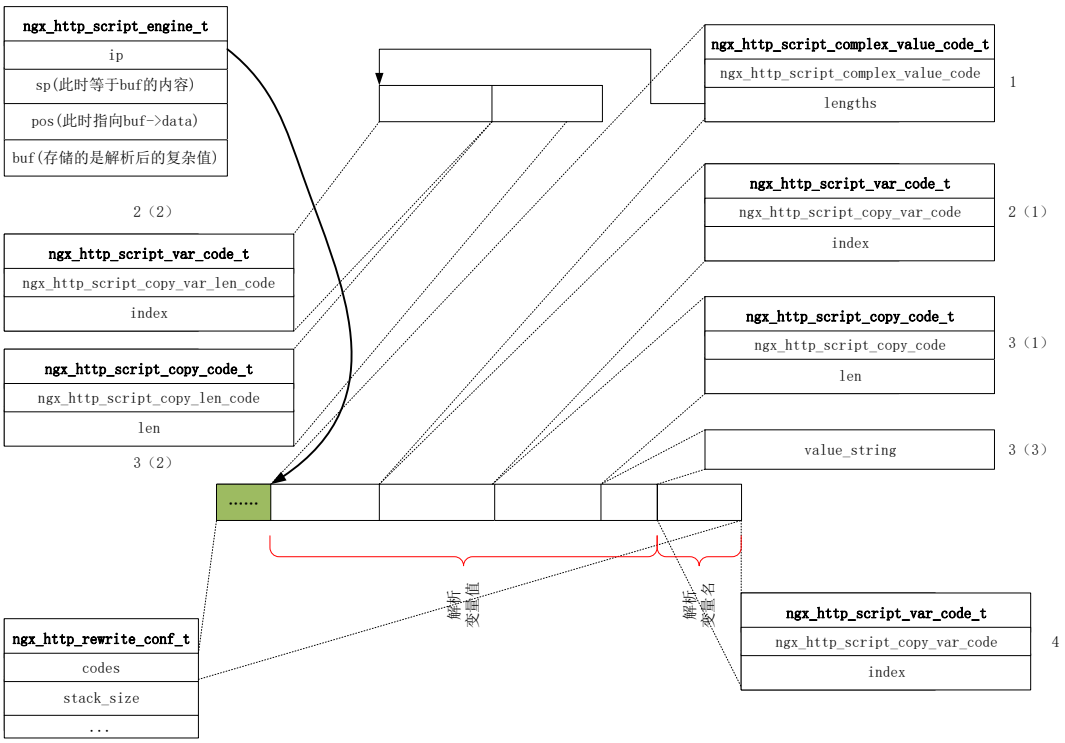

图 1 复杂值参数编译内存布局

在上面的内存布局图中我们可以得到一下几个信息：

1、如果配置文件中出现了一个带有复杂变量值参数的 set 命令，则结构体 ngx_http_script_complex_value_code_t 意味着编译该条 set 命令的开始。

2、在结构体 ngx_http_script_complex_value_code_t 中有两部分，第一部分是 lengths 数组，其中保存的是复杂值参数各个部分的值的长度，其中包含的编译值长度的结构体是在编译对应部分值的结构体时进行赋值，比如本例中的“${value}test”可以分为两部分，一个是内含变量”${value}”，一个是普通字符串“test”，则 lengths 数组中有两个元素，分别存储着用于获取内含变量”${value}”对应的值的长度，以及普通字符串”test”的长度的编译结构体。第二部分是函数指针 ngx_http_script_code_pt，此时对应的实现为 ngx_http_script_complex_value_code，这个函数的主要作用就是遍历 lengths 数组，计算实例中 set 赋值的变量的值的总长度，然后用这个长度为 e->buf 申请内存，保存解析后的变量值，并将 e->pos 指向 e->buf.data，e->sp 等于 e->buf。

3、在结构体 ngx_http_script_complex_value_code_t 之后，就是用于编译变量值的结构体，如上面所说，实例中的复杂值参数可以分为两部分，并按照先后顺序对这两部分进行分别编译，首先是编译获取内含变量”${value}”值的结构体--ngx_http_script_var_code_t，在这个结构体中包含了两部分，第一部分是 ngx_http_script_code_pt，此时实现是 ngx_http_script_copy_var_code，该函数用于结合第二部分的索引化下标获取变量值，并将此部分存储到 e->buf 中，如内存布局图中的 2（1）所示；然后是编译普通字符串“test”的部分，对于普通字符串，脚本引擎的处理比较特殊，在编译普通字符串的结构体（ngx_http_script_copy_code_t）之后紧跟着普通字符串值本身，然后 ngx_http_script_copy_code_t 结构体中的 ngx_http_script_code_pt 的实现 ngx_http_script_copy_code 会将紧跟在后面的普通字符串值拷贝到 e->buf 中，与内含变量值组成目标变量的值，如内存布局图中的标号 3（1）和 3（2）所示。

4、编译完变量值之后是编译变量名，编译变量名涉及到的结构体是 ngx_http_script_var_code_t，其中的 ngx_http_script_code_pt 对应的实现是 ngx_http_script_set_var_code，这个函数会将 e->sp（此时也为 e->buf）中解析后的变量值拷贝 r->variables 数组中对应的下标处。到此编译便完成了。

在 Nginx 解析完配置项之后，上面实例中涉及到的 set 命令也已经编译成脚本了，存放在 ngx_http_rewrite_loc_conf_t 结构体中的 codes 数组中。那何时会执行编译好的脚本呢？当请求匹配了上面的”/ image”的 location 并执行到 NGX_HTTP_SERVER_REWRITE_PHASE 或者 NGX_HTTP_REWRITE_PHASE 阶段时，就会执行存放在 codes 数组中的脚本。具体对应到代码就是 ngx_http_rewrite_handler()函数的处理，其执行流程如下：

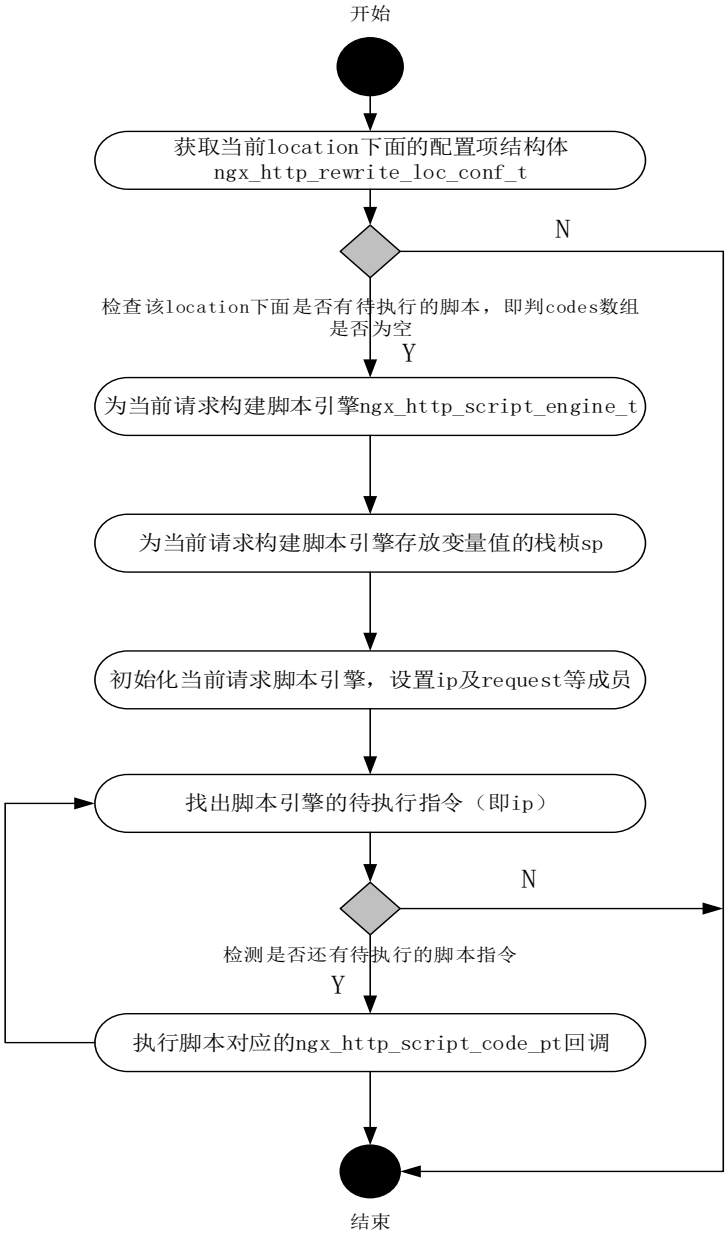

图 2 ngx_http_rewrite_handler 处理流程