# 配置解析

nginx不同模块保存配置的数据结构不同，因此不同模块在配置解析过程中的处理方式也不同。本文以http模块为例，分析配置解析的流程和http模块存储配置的数据结构。

## 核心数据结构

### ngx\_conf\_s

struct ngx\_conf\_s {

ngx\_array\_t \*args;

ngx\_cycle\_t \*cycle;

ngx\_conf\_file\_t \*conf\_file;

void \*ctx;

ngx\_uint\_t module\_type;

ngx\_uint\_t cmd\_type;

…

};

**args**：保存配置项数据的数组，包括配置项名称，配置项参数1，参数2等等。

**cycle**：指向cycle的指针。

**conf\_file**：配置文件的信息，比如配置文件的名称、配置文件大小等等。

**ctx**：指向cycle->conf\_ctx，4个\*的指针。

**module\_type**：当前需要解析哪一类模块的配置。

**cmd\_type**：当前需要解析哪一类的配置项，比如main、serv、loc。

## 关键函数

### ngx\_conf\_param

入参cf：一个ngx\_conf\_t的数据结构，这个函数唯一调用点就是在ngx\_init\_cycle。传入的是一个conf的局部变量。

出参char \*： 执行ngx\_conf\_parse返回的结果。

这个函数是对cycle->conf\_param的处理，其值来源于ngx\_conf\_param的全局变量。而这个全局变量的赋值在ngx\_get\_option。

在启动nginx设置了-g参数的时候，ngx\_conf\_param变量就会赋值，内容就是-g后面设定的nginx全局配置。函数里面用一个局部变量conf\_file来保存param里面的数据，相当于是把命令行设置的配置当作是一个临时的配置文件。然后传入ngx\_conf\_parse进行解析。

### ngx\_conf\_parse

入参cf：一个ngx\_conf\_t的数据结构。这个函数有很多的调用点，这里先分析从ngx\_init\_cycle调用的情况。

conf.ctx和cycle->conf\_ctx指向同一块内存。在这个阶段conf\_ctx仅仅是刚分配内存，然后调用定义了create\_conf方法的模块进行初始化。但大部分模块是没有定义create\_conf的，比如http。因此http模块在conf\_ctx中对应的内存空间是空的。

conf.cycle就是ngx\_init\_cycle初始化的cycle。

conf.module\_type为NGX\_CORE\_MODULE。

conf.cmd\_type为NGX\_MAIN\_CONF。

在ngx\_init\_cycle中conf所指向的内存在函数结束的时候是被释放了，每个模块配置解析得到的数据最终都存放在了cycle->conf\_ctx中。

入参filename：配置文件的名称。

出参char \*：

函数具体流程如下：

1. 如果传入了file\_name，则会读取file\_name对应的文件内容。从ngx\_init\_cycle调用ngx\_conf\_parse，传入的就是配置文件的路径。type字段用来标识解析的配置数据是来源于配置文件还是启动时设置的配置参数。
2. for循环调用ngx\_conf\_read\_token读取配置文件中的内容和ngx\_conf\_handler解析配置文件的内容。直到配置文件解析完成或者出现错误。

### ngx\_conf\_read\_token

入参cf：函数只有在ngx\_conf\_parse内部调用，cf就是ngx\_conf\_parse入参的cf。如果是解析配置文件的话，在ngx\_conf\_parse函数中对cf->conf\_file中fd、name、offset等字段进行赋值。

出参：NGX\_CONF\_OK表示配置数据读取成功。

NGX\_CONF\_ERROR表示配置数据读取出错。

NGX\_CONF\_FILE\_DONE表示配置文件读取完成。

NGX\_CONF\_BLOCK\_START表示一个配置块的开始，配置块就是用{ }包裹的配置内容。

NGX\_CONF\_BLOCK\_END表示一个配置块的结束。

函数的作用就是将配置文件中的内容读取到cf->conf\_file->buffer字段中。先看buffer的起始状态。在ngx\_conf\_parse函数中，申请了一块大小为4K的buffer用于配置解析需要。起始状态下：

buf.start = buf.pos = buf.last，buf.end = buf.last + 4K；

cf->conf\_file->file.offset = 0;

cf->conf\_file->line = 1；

根据上面描述，**第一次**执行for循环进入if (b->pos >= b->last)分支，各变量值情况如下：

len = b->pos – start = 0;表示buffer中已经保存的数据长度。

size = file\_size – cf->conf\_file->file.offset;这时候size就是配置文件的大小。

修正size的大小，如果size大小比buf剩余的空间要大，那么修正size大小为buf剩余空间。

调用ngx\_read\_file从配置文件中读取数据，读取大小为size。传入的offset则指定从文件的什么位置开始读取。数据读取完成之后，重新设置b->pos和b->last。数据读取完成之后，跳出if (b->pos >= b->last)分支，进入后续流程。

接下来就对读取到的数据进行格式分析。读取出来的4K数据中，可能包含一个或者多个完整的配置项，以及不完整的配置项。数据格式的分析需要执行多次for循环，每次for循环执行一次ch = \*b->>pos++;。每次遇到LF，说明以读取了完整的一行，因此cf->conf\_file->line++，同时还需要清空sharp\_comment标记（#注释符）。

先看if (last\_space)流程，因为need\_space初始值为0。当碰到**\**、**”**、**’**这三个符号的时候，last\_space会变为0，接下来的字符会进入else的流程。当出现另一半的**”**、**‘**的时候need\_space会被设置为1，表示一段完整的配置数据已经被读取，接下来要出现的必须是特定的几个字符，同时重置last\_space的状态。

found标记表示读取到一段完整的配置数据，注意这里并不一定是读取到一个完整的配置项，也有可能是某一个配置项中的某一个配置参数。因为在碰到‘ ‘和’/t’的时候，found都会被设置为1，只有当碰到’;’的时候才表示读取了一个完整的配置项内容。

每当读取都一段完整的配置数据，都会保存到cf->args数组中。当函数返回的时候数组中已经包含了一个完整配置项的所有配置数据。

ngx\_conf\_read\_token退出条件：

1. 碰到**;**，表示读取到一组完整的配置内容，返回NGX\_OK。
2. 碰到**{**，表示遇到配置块的开始，返回NGX\_CONF\_BLOCK\_START。
3. 碰到**}**，表示遇到配置块的结束，返回NGX\_CONF\_BLOCK\_END。
4. 异常情况，返回NGX\_ERROR。

其实当ngx\_conf\_read\_token正常返回的时候，表示已经读取到了一个可以调用解析函数解析的配置内容。

### ngx\_conf\_handler

入参cf：ngx\_conf\_parse传入的cf结构。

入参last：ngx\_conf\_read\_token函数的返回值。

出参：NGX\_OK表示某个配置项解析成功，NGX\_ERROR表示配置项解析失败。

这个函数的执行效率其实比较低下，每次都要遍历所有的模块，取出每个模块的配置项与cf->args数组中的配置项名称进行比较。注意几个判断条件：

1. 模块的配置项名称要匹配args中保存的配置项名称。
2. 配置项对应模块的类型要匹配cf->module\_type。cf->module\_type会随着ngx\_conf\_parse调用点不同而不同。在ngx\_init\_cycle流程中，cf->module\_type为NGX\_CORE\_MODULE。
3. 配置项的类型要匹配cf->cmd\_type。
4. 根据不同的cmd->type，对配置项的参数个数进行校验。

通过了上述各种校验之后，才调用每个配置项自定义的解析函数对配置进行解析。这里要注意局部变量conf的值，对于NGX\_MAIN\_CONF类型的配置项，conf实际就是cycle->conf\_ctx中指向某个模块保存配置数据的指针。

比如当读取到配置文件中”http”这个配置项，就找到了指向ngx\_http\_module配置数据结构的指针，但在ngx\_init\_cycle的流程里，这个指针目前还是空的，要在ngx\_http\_block函数中才会分配内存。

接下来就是调用每个配置项自定义的解析函数。根据自定义解析函数的返回值判断配置解析是成功还是失败。

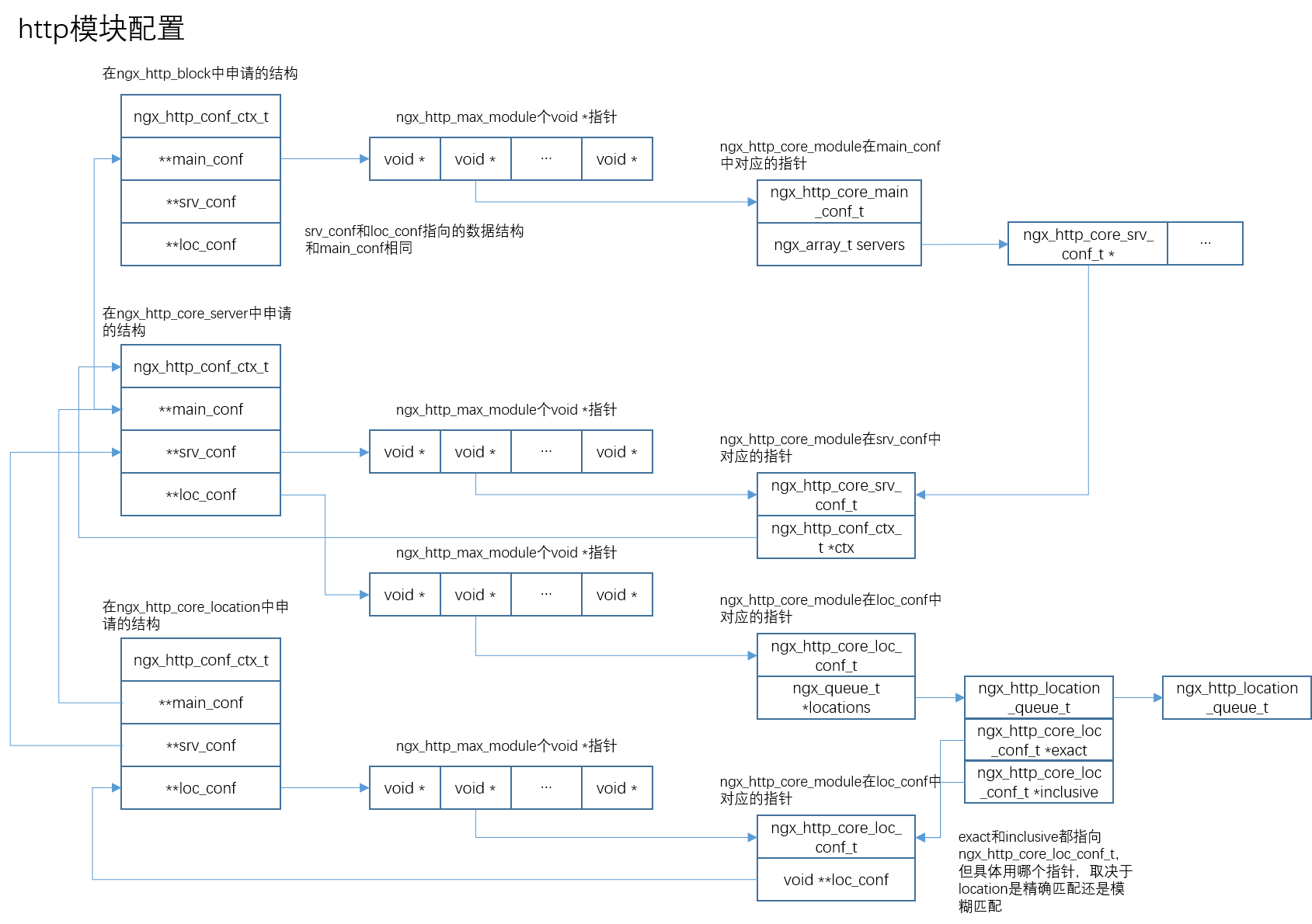
# http模块配置解析

http模块配置解析的入口函数是ngx\_http\_block，即在配置文件中读到http这个配置之后调用的函数。其余的http模块配置及各http子模块的配置都是由该函数触发解析的。

http模块的配置分为:NGX\_HTTP\_MAIN\_CONF、NGX\_HTTP\_SRV\_CONF、NGX\_HTTP\_LOC\_CONF三种类型。各个http子模块中command数组里定义的各个配置项都有设置类型。很多配置项会同时设置多种类型，比如同时设置三种类型。这种写法表示该配置项既可以配置在http { … }配置块中，也可以配置在http配置块里面某个server的配置块，或者某个server里的某个location配置块中。如果配置在http配置块，则该配置项对于http配置块中所有的server和location都生效。如果配置在server配置块，则对于该server及其内部所有的location配置块生效。

解析http模块配置首先是解析NGX\_HTTP\_MAIN\_CONF，当从配置文件中读取到server、location配置项的时候则会去触发NGX\_HTTP\_SRV\_CONF和NGX\_HTTP\_LOC\_CONF。

## 数据结构关系



## 核心数据结构

### ngx\_http\_conf\_ctx\_t

typedef struct {

void \*\*main\_conf;

void \*\*srv\_conf;

void \*\*loc\_conf;

} ngx\_http\_conf\_ctx\_t;

**main\_conf**：指针数组，数组成员是http每个子模块对应的main\_conf数据结构。

**srv\_conf**：指针数组，数组成员是http每个子模块对应的srv\_conf数据结构。

**loc\_conf**：指针数组，数组成员是http每个子模块对应的loc\_conf数据结构。

### ngx\_command\_s

struct ngx\_command\_s {

ngx\_str\_t name;

ngx\_uint\_t type;

char \*(\*set)(ngx\_conf\_t \*cf, ngx\_command\_t \*cmd, void \*conf);

ngx\_uint\_t conf;

ngx\_uint\_t offset;

void \*post;

};

**name**：配置项名称。

**type**：配置项类型或者叫配置项属性，NGX\_MAIN\_CONF、NGX\_CONF\_BLOCK、NGX\_CONF\_NOARGS等。

**set**：配置项对应的解析函数。在ngx\_conf\_handler中调用。

**conf**：表示配置项存储在哪个数据结构中。对于http配置，配置是存在main\_conf、srv\_conf还是loc\_conf中。

**offset**：配置项在数据结构中的偏移量。

**post**：配置解析之后需要进行的额外的自定义操作。

### ngx\_http\_core\_main\_conf\_t

typedef struct {

ngx\_array\_t servers;

…

} ngx\_http\_core\_main\_conf\_t;

## 关键函数

### ngx\_http\_block

入参cf：ngx\_conf\_handler的入参cf，实际就是ngx\_init\_cycle中局部变量conf。

入参cmd：本次要解析的配置项的结构ngx\_command\_t。

入参conf：实际就是cycle->conf\_ctx中ngx\_http\_module对应的数组成员的指针。该指针最终指向ngx\_http\_conf\_ctx\_t数据结构。

出参：异常返回NGX\_CONF\_ERROR，正常返回ngx\_conf\_parse的返回值。

函数入口实际就是给cycle->conf\_ctx中ngx\_http\_module所对应的成员指针赋值。然后调用所有NGX\_HTTP\_MODULE定义的create\_main\_conf、create\_srv\_conf、create\_loc\_conf给ngx\_http\_conf\_ctx\_t中main\_conf、srv\_conf、loc\_conf对应的成员赋值。

替换cf->ctx为ngx\_http\_conf\_ctx\_t，cf->ctx决定了接下来调用ngx\_conf\_parse解析的配置数据存放在哪里。同时修改cf->module\_type和cf->cmd\_type，这样接下来的ngx\_conf\_parse就只会解析http各个子模块中满足条件的配置项。这里cmd\_type是设置为NGX\_HTTP\_MAIN\_CONF。在ngx\_http\_core\_server函数中会被设置为NGX\_HTTP\_SRV\_CONF。

ngx\_conf\_parse的流程上面已经讲过，过程还是从配置文件中读取数据，然后分析数据，每得到一个完整的配置项，就调用ngx\_conf\_handler进行解析。需要注意的是，在这个流程进入ngx\_conf\_handler中，cmd->type已经被设置为NGX\_HTTP\_MAIN\_CONF，因此传入cmd->set的第三个参数conf赋值如下：

confp = \*(void \*\*) ((char \*) cf->ctx + cmd->conf);

if (confp) {

conf = confp[cf->cycle->modules[i]->ctx\_index];

}

这里的cf->ctx已经指向了ngx\_http\_conf\_ctx\_t，cmd->conf对应到ngx\_http\_commands里面的值是0。因此confp实际指向ngx\_http\_conf\_ctx\_t中的main\_conf。conf则是配置项对应的http子模块在main\_conf数组中的成员。

ngx\_conf\_parse一旦执行完成，则表示所有http模块相关的配置项都解析完成。接下来要做的就是配置项合并的操作，详见配置项合并相关函数的解析。

配置解析处理完成之后就是各类数据结构的初始化操作，其中需要特别关注的是ngx\_http\_init\_phase\_handlers，这个函数处理了http的各个模块在各个不同的处理阶段定义自定义的处理函数，是开发http模块的核心之一，具体内容将在《Nginx HTTP处理流程》文档中说明。

### ngx\_http\_core\_server

入参cf：ngx\_conf\_handler的入参cf，实际就是ngx\_init\_cycle中局部变量conf。

入参cmd：本次要解析的配置项的结构ngx\_command\_t。

入参dummy：无用参数。

出参：异常返回NGX\_CONF\_ERROR，正常返回ngx\_conf\_parse的返回值。

函数会重新创建一个ngx\_http\_conf\_ctx\_t的数据结构，这个数据结构和ngx\_http\_block的数据结构是分开的，只是通过字段指针进行关联。这个server配置块中所有的配置内容都会保存到新创建的数据结构中。数据结构字段之间的关系可以参考上面的图。

注意一下这段代码：

cscf = ctx->srv\_conf[ngx\_http\_core\_module.ctx\_index];

cscf->ctx = ctx;

cmcf = ctx->main\_conf[ngx\_http\_core\_module.ctx\_index];

cscfp = ngx\_array\_push(&cmcf->servers);

if (cscfp == NULL) {

return NGX\_CONF\_ERROR;

}

\*cscfp = cscf;

cscf是本次创建的ctx中ngx\_http\_core\_module在srv\_conf中对应的成员。cmcf是本次创建的ctx中ngx\_http\_core\_module在main\_conf中对应的成员。但本次创建的ctx的main\_conf实际上是指向ngx\_http\_block中创建的ctx的main\_conf。

下面的数组操作实际上就是将本次创建的ctx中ngx\_http\_core\_module在srv\_conf中对应的成员加入了ngx\_http\_block创建的ctx中ngx\_http\_core\_module在main\_conf中对应的成员的server数组中，这样就把server和main配置对应关系建立了。

在通过最上面两行代码，可以通过ngx\_http\_core\_module的成员找到ctx，实际上就是找到当前server配置块，这就形成了完整的从main\_conf到srv\_conf的路径。

配置解析完成之后，会对server的监听端口进行处理（待补充）。

### ngx\_http\_core\_location

入参cf：ngx\_conf\_handler的入参cf，实际就是ngx\_init\_cycle中局部变量conf。

入参cmd：本次要解析的配置项的结构ngx\_command\_t。

入参dummy：无用参数。

函数前半部分的处理和ngx\_http\_core\_server类似。后半部分则是对于location名称的处理，是精确匹配还是模糊匹配还是正则表达式匹配等。最后调用ngx\_conf\_parse解析location中的配置。

需要注意的就是ngx\_http\_add\_location函数，这个函数建立了ngx\_http\_core\_server函数中创建的ngx\_http\_conf\_ctx\_t中loc\_conf与ngx\_http\_core\_location函数中创建的ngx\_http\_conf\_ctx\_t中loc\_conf之间的关系。之所以这样做是因为一个server配置块里面可以由多个location的配置块，而根据ngx\_http\_core\_location的逻辑，每个location配置块之间的内存都是独立的。就需要通过server配置块中的location链表将其关联起来。

### ngx\_http\_merge\_servers

入参cf：ngx\_http\_block入参的cf。只是此时cf->ctx指向http\_conf\_ctx\_t。

入参cmcf：ctx->main\_conf中ngx\_http\_core\_module对应的成员。

入参module：所有http子模块的数据结构。

入参ctx\_index：http子模块的index。

出参：虽然函数定义的返回值类型是char\*，但该函数实际返回的是ngx\_http\_conf\_ctx\_t\*结构。

该函数的调用是在ngx\_http\_block，只有这一处调用点。先看调用点位置的处理逻辑，遍历找到所有NGX\_HTTP\_MODULE，对定义了init\_main\_conf方法的http模块调用该方法。以ngx\_http\_core\_module为例，ngx\_http\_core\_init\_main\_conf函数是对该模块部分没有配置值的main\_conf配置项设置默认值。

函数内部遍历server数组，调用入参module定义的merge\_srv\_conf的方法，合并每个server配置块中的配置，以及合并每个server配置块中所有location配置块中的配置。还是以ngx\_http\_core\_module为例，参见ngx\_http\_core\_merge\_srv\_conf和ngx\_http\_core\_merge\_loc\_conf函数。

### ngx\_http\_core\_merge\_srv\_conf

入参cf：ngx\_http\_merge\_serves的入参cf。

入参parent：各个模块在ngx\_http\_block中创建的ctx->svr\_conf里对应的成员。

入参child：各个模块在之前解析各个server配置块所创建的ctx->svr\_conf里对应的成员。

函数中调用ngx\_conf\_merge\_size\_value作用是，如果child中对应的配置项没有配置值，那么就看parent是否有配置值，如果有就用parent的值，没有就用默认值。

ngx\_conf\_merge\_value也是这个作用。

### ngx\_http\_merge\_locations