Nginx Module Configure

# 概述

无

# 模块Module

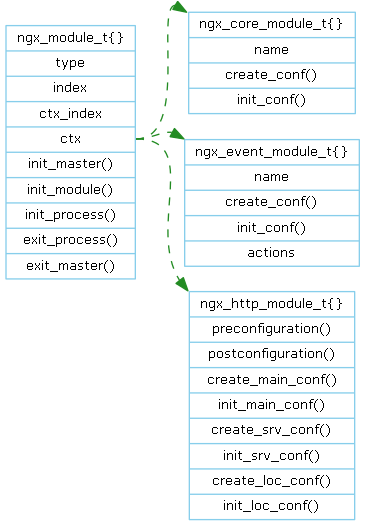
Nginx提供的功能都由模块来实现，和Apache不同Nginx的模块不能运行时安装，只能在编译时静态链接进入。

Nginx模块类型分为四类：

* CORE ：总管理模块，共有三个具体模块 core、events、http
* CONF ：配置模块，只有conf
* EVENT ：事件模块，有event\_core和select、epoll、poll、kqueue等
* HTTP ：HTTP处理模块，有http\_core和http\_upstream、http\_static等

## 模块定义

ngx\_module\_t 是一个公共的结构体，每个模块都有它的一个实例(全局变量)。但是四类模块又有自已特有的属性，



于是在ngx\_module\_t开发了一个ctx指针指向各类模块特有的数据结构。

CORE类为ngx\_core\_module\_t，CONF类没有，EVENT类为ngx\_event\_module\_t，HTTP类为ngx\_http\_module\_t。

### ngx\_module\_t

#### type

模块的类型（CORE、CONF、EVENT、HTTP）

#### index

所有模块间的索引

#### ctx\_index

同类模块间的索引

#### ctx

各类模块特有的数据结构

#### init\_master()

初始化Master时，回调该函数

#### init\_module()

初始化Module时，回调该函数

#### init\_process()

初始化Worker时，回调该函数

#### exit\_process()

退出Worker时，回调该函数

#### exit\_master()

退出Master时，回调该函数

### ngx\_core\_module\_t

略

### ngx\_event\_module\_t

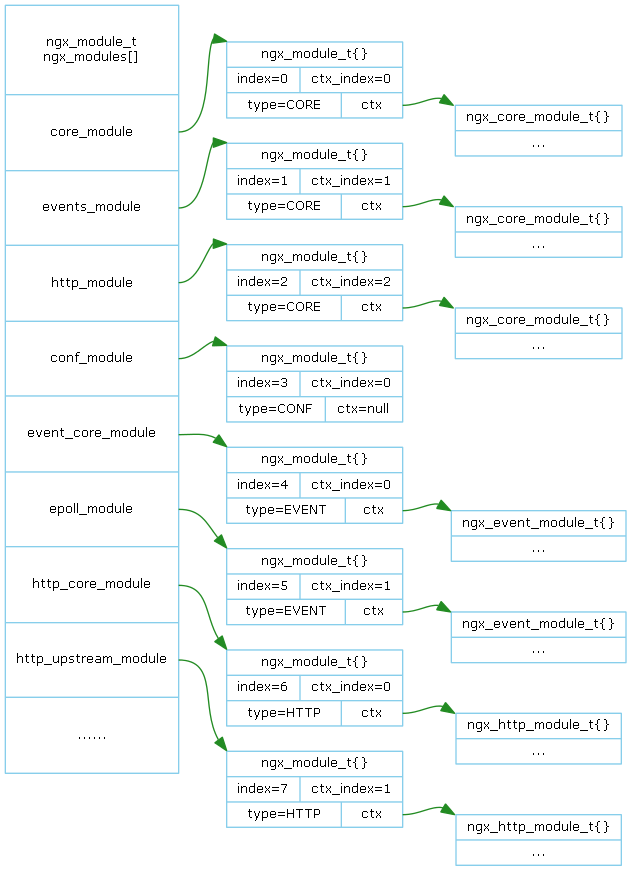
略

### ngx\_http\_module\_t

略

## 模块组织

在objs/ngx\_modules.c文件中定义了全局数组变量ngx\_modules[]，所有的模块都添加到了这个数组中。



## 模块运作

在init\_master()/init\_module()/init\_process()回调函数中，注册业务回调函数，在业务回调函数中处理业务。

# 配置文件

daemon off;

worker\_processes 1;

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

server {

listen 8000;

server\_name localhost;

location / {

root html;

index index.html index.htm;

}

}

}

types {

text/html html htm shtml;

text/css css;

image/jpeg jpeg jpg;

}

#file mine.types

指令名(Directive) 是上面绿色的内容。指令参数(Arguments)是上面红色的内容。

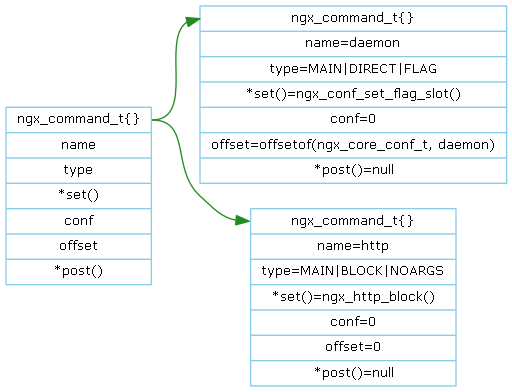
指令由一个指令名和若干个指令参数组成并以分号或大括号结束，例如daemon off 和 http。带有大括号的指令称为块(block)指令。

配置文件可以包含其它配置文件（include）。

# 配置数据结构

## 配置命令定义

Nginx支持的指令都会有一个对应的ngx\_command\_t，这里称之为配置命令。



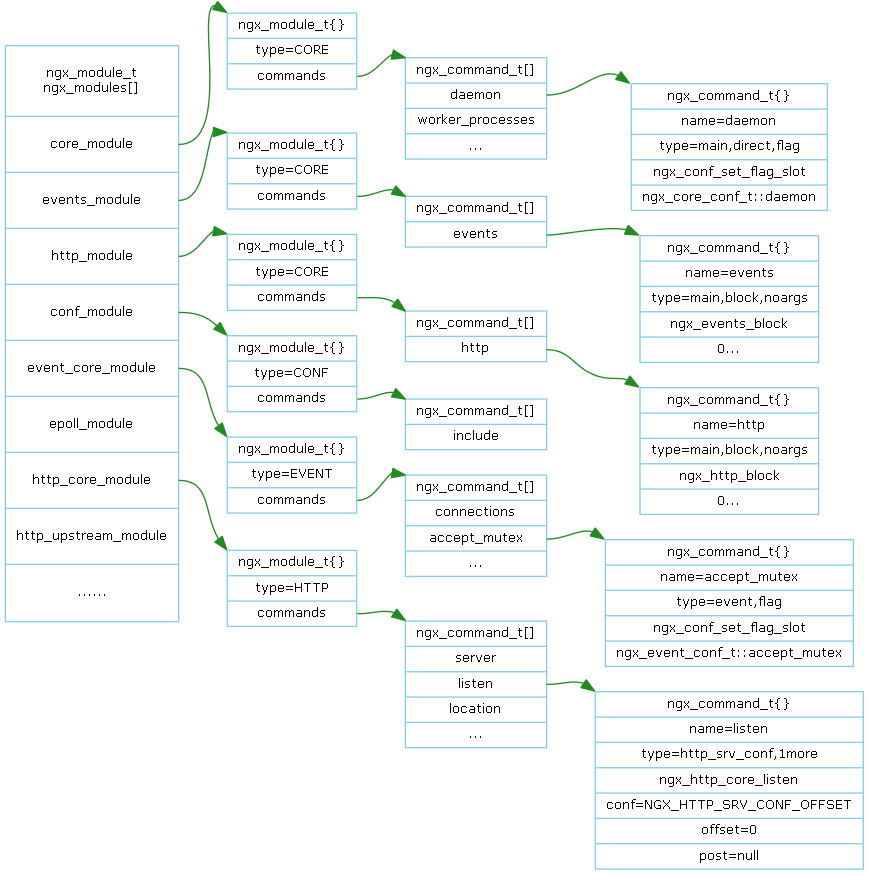
* name ： 配置命令对应的指令名
* type ： 配置命令指令参数个数、类型和上下文信息
* NGX \_ DIRECT\_CONF：见下文。
* NGX\_CONF\_BLOCK：是一个BLOCK(大括号)。
* NGX\_CONF\_NOARGS：对应的指令没有指令参数。

以下表示该命令有效范围：

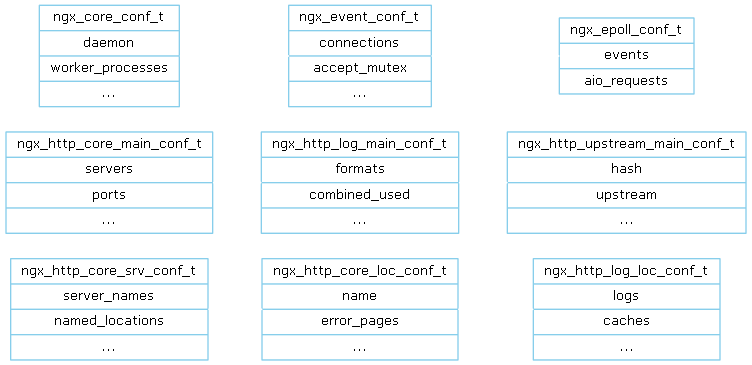
* NGX\_MAIN\_CONF：不在任何BLOCK内。
* NGX\_EVENT\_CONF：在events BLOCK内。
* NGX\_HTTP\_MAIN\_CONF：在http BLOCK内。
* NGX\_HTTP\_SRV\_CONF：在 http server BLOCK内。
* NGX\_HTTP\_LOC\_CONF：在http server location BLOCK内。
* NGX\_HTTP\_LMT\_CONF：在http limit except BLOCK内。
* NGX\_HTTP\_UPS\_CONF：在http upstream BLOCK内。
* set() ： 回调函数，用于设置配置域
* conf ： 用于找配置结构（第二层的偏移）
* offset ： 配置域在配置结构中的偏移。（第四层的偏移）
* post() ： 设置完配置域后的回调函数

## 配置命令组织

Nginx可以通过配置文件配置每个模块。模块数据结构ngx\_module\_t的commands是配置命令数组，记录了该模块支持的所有配置命令。



## 配置结构定义

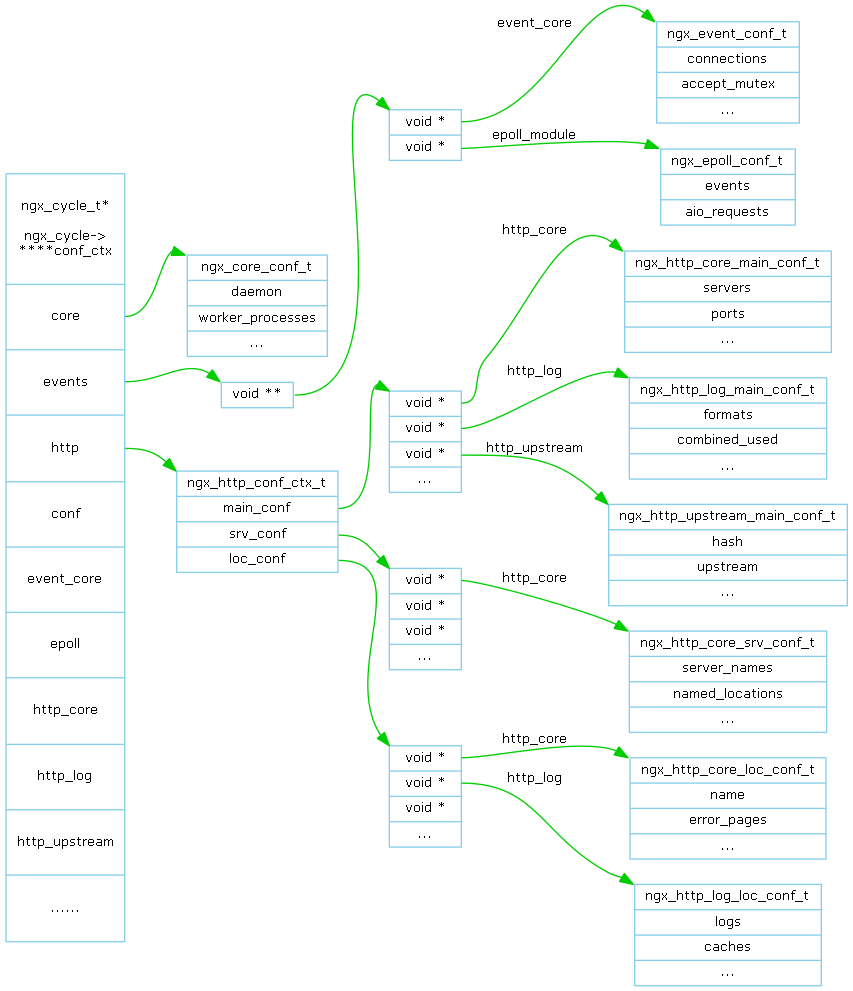


以上的结构体称之为配置结构。配置结构中的各个域称之为配置域。

ngx\_core\_conf\_t 是core\_module的配置结构，ngx\_event\_conf\_t是event\_core\_module的，ngx\_epoll\_conf\_t是epoll\_module的。

## 配置结构组织

配置结构由ngx\_cycle\_t的conf\_ctx所管理(如上图)。conf\_ctx指向一个void指针数组(下文简称V1数组)，该数组大小为module数量。



上图结构中共有四层，所以conf\_ctx被定义为一个四层指针(void \*\*\*\*conf\_ctx)。我个人认为此法费解，建议定义为void\*\*conf\_ctx表示conf\_ctx

指向一个void指针数组，其余层次用强制类型转换。

由conf\_ctx所管理的配置结构可分为三种管理方式：

* 空类型

conf\_module等模块没有配置结构，V1[ngx\_conf\_module.index]为NULL。

1. 直接类型(Direct)

core\_module的配置结构属于于这种管理方式。V1数组中直接保存配置结构结构体的指针。ngx\_core\_conf\_t的指针可以如下得到：

(ngx\_core\_conf\_t\*)ngx\_cycle->conf\_ctx[ngx\_core\_module.index];

1. 间接类型

用于管理EVENT和HTTP类型模块的配置结构。例如epoll\_module的配置结构并不是直接在V1[ngx\_epoll\_module.index]

而是在V1[ngx\_events\_module.index]。具体看上图更清楚。

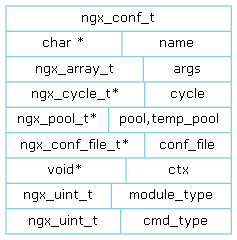
epoll模块的配置结构aio\_requests 可以由如下表达式得到：

ngx\_epoll\_conf\_t\* ec = (\*ngx\_cycle->conf\_ctx[ngx\_events\_module.index])[ngx\_epoll\_module.ctx\_index];

ec->aio\_requests;

## 配置上下文

* name 和 args



从配置文件读取的指令名和指令参数

* conf\_file

当前的配置文件句柄

* ctx

配置结构上下文

* module\_type 和 cmd\_type

当前配置的模块类型和命令类型

# 配置过程

配置过程如下(入口函数是ngx\_init\_cycle，关键函数是ngx\_conf\_parse)：

1. 初始化

设置上下文。

1. 读取指令

从配置文件(包括命令行参数)读取一个指令 (关键函数是ngx\_conf\_read\_token)。

1. 查找配置命令

根据指令名从各模块的配置命令数组中查找匹配的配置命令(关键函数是ngx\_conf\_handler)。

1. 查找配置结构

根据配置命令找到配置结构(关键函数是ngx\_conf\_handler)。

1. 执行配置

调用配置命令的set回调函数执行配置。

解析参数并设置到配置域 (daemon, accept\_mutex等)。

创建配置结构并切换上下文 (events, http等)。

打开配置文件并切换上下文 (include)。

## 读取指令

先从文件读取字符填满缓冲区，再从缓冲区解析所有的token，无法解析为token的剩余字符移到缓冲区的头部，再从文件读字符填满缓冲区。

如此循环直到遇到分号大括号或者结尾。第一个token是指令名，存入ngx\_conf\_t的name域，其余的token是指令参数，存入args域。

## 包含文件

遇到include指令，打开新配置文件，保存老配置文件句柄(ngx\_conf\_t::conf\_file)，解析新配置文件，恢复老配置文件句柄。

## 切换上下文

配置过程是从ngx\_init\_cycle函数开始的，上下文ngx\_conf\_t是该函数的一个局部变量conf，它的指针被传入ngx\_conf\_parse。

conf被初始为：

ctx指向ngx\_cycle->conf\_ctx

module type 为CORE

cmd type 为 MAIN

下面分别分析读取不同的指令后，上下文切换情况：

* daemon

不切换上下文

* events

在set回调函数ngx\_events\_block中切换上下文，切换为：

ctx指向ngx\_cycle->conf\_ctx[ngx\_events\_module.index]所指向的void\*\*

module type 为 EVENT

cmd type 为 NGX\_EVENT\_CONF

ngx\_events\_block返回之前上下文恢复

* http

在set回调函数ngx\_http\_block中切换上下文，切换为：

ctx指向ngx\_cycle->conf\_ctx[ngx\_http\_module.index]所指向的ngx\_http\_conf\_ctx\_t

module type 为 HTTP

cmd type 为 NGX\_HTTP\_MAIN\_CONF

ngx\_http\_block返回之前上下文恢复

切换上下文实际上是利用运行时栈空间，切换ngx\_conf\_t。下面的代码片段说明如何切换：

static char \* ngx\_events\_block(ngx\_conf\_t \*cf, ngx\_command\_t \*cmd, void \*conf)

{

ngx\_conf\_t **saved\_conf**;

**saved\_conf = \*cf;**

**cf**->ctx = …

**cf**->module\_type = NGX\_EVENT\_MODULE;

**cf**->cmd\_type = NGX\_EVENT\_CONF;

rv = ngx\_conf\_parse(**cf**, NULL);

**\*cf = saved\_conf;**

return rv;

}

## 设置配置结构

通过ngx\_conf\_t::ctx、ngx\_command\_t::type、ngx\_command\_t::conf、ngx\_command\_t::offset这四个信息找出配置域的地址。

解析参数后设入即可。

* 直接类型

ngx\_command\_t::type 为DIRECT

ngx\_conf\_t::ctx指向第一层(void\*指针数组V1)，ngx\_module\_t::index是第一层的index。

由第一层指针和ngx\_module\_t::index，便可定位出第二层（配置结构指针），ngx\_command\_t::offset是第二层的偏移。

由第二层指针和ngx\_command\_t::offset，便可定位出配置域。

* 间接类型

ngx\_conf\_t::ctx指向第二层（void\*指针数组），ngx\_command\_t::conf是第二层的index。

由第二层指针和ngx\_command\_t::conf，便可定位出第三层（void\*指针数组），ngx\_module\_t::ctx\_index是第三层的index。

由第三层指针和ngx\_module\_t:ctx\_index，便可定位出第四层（配置结构指针），ngx\_command\_t::offset是第四层的偏移。

由第四层指针和ngx\_command\_t::offset便可定位出配置域。

# Prototype

<https://github.com/lingjf/nginx_configure.git>

# Latest revision

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