**模块**

**结构**

ngx\_module\_t代表一个模块，nginx使用一个数组ngx\_modules保存所有模块，这个数组是在执行shell命令 ./configure 时生成的

struct ngx\_module\_s {

    ngx\_uint\_t            ctx\_index;        // 当前模块在当前相同类型的模块中的位置

    ngx\_uint\_t            index;            // 当前模块在模块数组的位置

    void                 \*ctx;              // 特定类型模块的专用数据

    ngx\_command\_t        \*commands;         // 模块的指令数组

ngx\_uint\_t            type;             // 模块类型

    ...

};

**模块的类型**

Nginx目前的模块类型有5种

#define NGX\_CORE\_MODULE ...     // 核心模块

#define NGX\_CONF\_MODULE ...

#define NGX\_EVENT\_MODULE ...

#define NGX\_HTTP\_MODULE ...     // http 模块，本教程所开发的模块

#define NGX\_MAIL\_MODULE ...

**指令**

一个指令对应一个ngx\_command\_s结构，如（test off指令）

struct ngx\_command\_s {

    // 指令的名称

    ngx\_str\_t             name;

    // 指令的解析函数

    // cf:解析的环境上下文

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

    ...

};

**特定类型模块的专用数据**

对于NGX\_CORE\_MODULE类型，ctx的结构是ngx\_core\_module\_t

typedef struct {

    ngx\_str\_t             name;

    void               \*(\*create\_conf)(ngx\_cycle\_t \*cycle);     // 创建配置结构函数

    char               \*(\*init\_conf)(ngx\_cycle\_t \*cycle, void \*conf);   // 初始化配置结构函数

} ngx\_core\_module\_t;

对于NGX\_HTTP\_MODULE类型，ctx的结构是ngx\_http\_module\_t

static ngx\_http\_module\_t ndg\_test\_ctx = {

    nullptr,                // 解析配置文件前被调用

    init,                   // 解析配置文件后被调用

    nullptr,                // 创建 http main 域的配置结构函数

    nullptr,                // 初始化 http main 域的配置结构函数

    nullptr,                // 创建 server 域的配置结构函数

    nullptr,                // 合并 server 域的配置结构函数

    create,                 // 创建 location 域的配置结构函数

    nullptr                 // 合并 location 域的配置结果函数

};

**nginx的配置**

Nginx只认得core模块，所以要解析配置首先的实现一个core模块作为入口

Nginx的http的指令允许低层次的指令配置在高层次中，如location的指令可以配置在server中，server的指令可以配置在http中

Nginx使用配置结构保存指令

**ngx\_cycle\_t**

ngx\_cycle\_t.conf\_ctx保存了core类型模块的配置结构

// 定义在 core/ngx\_core.h

typedef struct ngx\_cycle\_s ngx\_cycle\_t;

struct ngx\_cycle\_s {

    void\*\*\*\* conf\_ctx;      // 配置数据的起始存储位置

    ...

}

conf\_ctx是一个 void\* 的2维数组（但源码似乎只保存一维数据）

**ngx\_http\_conf\_ctx\_t**

http系列模块的入口是ngx\_http\_module（这是一个core类型的模块）

ngx\_http\_conf\_ctx\_t是ngx\_http\_module模块和http类型模块使用的配置结构，其保存了http，server，location的配置

注：在ngx\_http\_module模块中srv\_conf，loc\_conf并不是指server的配置结构和location的配置结构，而是指配置在http的main域下的server和location的配置结构

typedef struct {

    void        \*\*main\_conf;    // http 类型模块所创建的 main 域的配置结构

    void        \*\*srv\_conf;     // http 类型模块所创建的 server 域的配置结构

    void        \*\*loc\_conf;     // http 类型模块所创建的 loc 域的配置结构

} ngx\_http\_conf\_ctx\_t;

**ngx\_conf\_t**

ngx\_conf\_t是解析配置时的运行环境

struct ngx\_conf\_s {

    ngx\_array\_t          \*args;     // 指令字符串数组

    ngx\_pool\_t           \*pool;     // 内存池

    ngx\_log\_t            \*log;      // 日志

    void                 \*ctx;      // 当前上下文

    ...

};

在解析core类型的配置时ctx是什么结构我也不清楚

在解析http类型的配置时，ctx是ngx\_http\_conf\_ctx\_t结构

**core类型配置存储**

for (i = 0; ngx\_modules[i]; i++) {

    // 如果模块不是 core 类型

    if (ngx\_modules[i]->type != NGX\_CORE\_MODULE) {

        continue;

    }

    module = ngx\_modules[i]->ctx;

    if (module->create\_conf) {

        // 调用 ctx->create\_conf 生成配置结构

        rv = module->create\_conf(cycle);

        ...

        // 配置保存到 conf\_ctx 中

        cycle->conf\_ctx[ngx\_modules[i]->index] = rv;

    }

}

**http指令配置存储**

ngx\_http\_module的commands字段（指令数组）包含了name为“http”的指令，对应的解析函数set为ngx\_http\_block()

ctx = ngx\_pcalloc(cf->pool, sizeof(ngx\_http\_conf\_ctx\_t));

...

ctx->main\_conf = ngx\_pcalloc(cf->pool, sizeof(void \*) \* ngx\_http\_max\_module);

ctx->srv\_conf = ngx\_pcalloc(cf->pool, sizeof(void \*) \* ngx\_http\_max\_module);

ctx->loc\_conf = ngx\_pcalloc(cf->pool, sizeof(void \*) \* ngx\_http\_max\_module);

...

// 遍历所有 http 模块

for (m = 0; ngx\_modules[m]; m++) {

    if (ngx\_modules[m]->type != NGX\_HTTP\_MODULE) {

        continue;

}

    module = ngx\_modules[m]->ctx;

    mi = ngx\_modules[m]->ctx\_index;

// 创造 main 域的配置结构

    if (module->create\_main\_conf) {

        ctx->main\_conf[mi] = module->create\_main\_conf(cf);

    }

// 创造 server 域的配置结构，因为server域的指令允许配置在main域中

    if (module->create\_srv\_conf) {

        ctx->srv\_conf[mi] = module->create\_srv\_conf(cf);

    }

// 创造 location 域的配置结构，因为location域的指令允许配置在main域中

    if (module->create\_loc\_conf) {

        ctx->loc\_conf[mi] = module->create\_loc\_conf(cf);

    }

}

...

// 在http内执行配置解析

rv = ngx\_conf\_parse(cf, NULL);

**server指令配置存储**

ngx\_http\_core\_module（http类型模块）的commands字段（指令数组）包含了name为“server”的指令，对应的解析函数set为ngx\_http\_core\_server()

static char \* ngx\_http\_core\_server(ngx\_conf\_t \*cf, ngx\_command\_t \*cmd, void \*dummy)

{

    // 创建 ngx\_http\_conf\_ctx\_t 结构

    ctx = ngx\_pcalloc(cf->pool, sizeof(ngx\_http\_conf\_ctx\_t));

    // 指定模块的 main 域配置

    // 因为模块的 main 与当前环境的 main 域是相同的，所以直接指向当前环境的 main 域

    http\_ctx = cf->ctx;

ctx->main\_conf = http\_ctx->main\_conf;

    // 创建 server 域配置

    ctx->srv\_conf = ngx\_pcalloc(cf->pool, sizeof(void \*) \* ngx\_http\_max\_module);

    if (ctx->srv\_conf == NULL) {

        return NGX\_CONF\_ERROR;

}

    // 创建 location 域配置

    // 因为 server 中允许配置 location 域

    ctx->loc\_conf = ngx\_pcalloc(cf->pool, sizeof(void \*) \* ngx\_http\_max\_module);

    if (ctx->loc\_conf == NULL) {

        return NGX\_CONF\_ERROR;

}

    // 遍历所有模块

    for (i = 0; ngx\_modules[i]; i++) {

        if (ngx\_modules[i]->type != NGX\_HTTP\_MODULE) {

            continue;

        }

        module = ngx\_modules[i]->ctx;

        if (module->create\_srv\_conf) {

            mconf = module->create\_srv\_conf(cf);

            ctx->srv\_conf[ngx\_modules[i]->ctx\_index] = mconf;

        }

        if (module->create\_loc\_conf) {

            mconf = module->create\_loc\_conf(cf);

            ctx->loc\_conf[ngx\_modules[i]->ctx\_index] = mconf;

        }

    }

...

    // 在server内解析配置

rv = ngx\_conf\_parse(cf, NULL);

    ...

    return rv;

}

**location指令配置存储**

ngx\_http\_core\_module（http类型模块）的commands字段（指令数组）包含了name为“location”的指令，对应的解析函数set为ngx\_http\_core\_location()

同理location指令的存储结构为ngx\_http\_conf\_ctx\_t，其main域和server域的配置为指向其所在的main域和server域

**访问配置数据**

// cf 为 ngx\_conf\_t

// module 为 ngx\_module\_t

// 获取模块在 main 域中的配置

#define ngx\_http\_conf\_get\_module\_main\_conf(cf, module) ...

// 获取模块在 server 域中的配置

#define ngx\_http\_conf\_get\_module\_srv\_conf(cf, module) ...

// 获取模块在 location 域中的配置

#define ngx\_http\_conf\_get\_module\_loc\_conf(cf, module) ...

**指令**

**结构**

struct ngx\_command\_s {

    // 指令名

    ngx\_str\_t             name;

    // 指令的作用域和类型

    ngx\_uint\_t            type;

    // 解析指令的函数

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

    // 指定该指令保存位置

    ngx\_uint\_t            conf;

    // 指定该指令所对应的配置结构变量的地址

    ngx\_uint\_t            offset;

    void                 \*post;

};

**指令类型和作用域**

类型：

// 取值 on | off

#define NGX\_CONF\_FLAG        0x00000200

// 不限制参数数量

#define NGX\_CONF\_ANY         0x00000400

// 参数必须超过1个

#define NGX\_CONF\_1MORE       0x00000800

// 参数必须超过2个

#define NGX\_CONF\_2MORE       0x00001000

...

作用域：

// 可以出现在 main 域中

#define NGX\_HTTP\_MAIN\_CONF        0x02000000

// 可以出现在 server 域中

#define NGX\_HTTP\_SRV\_CONF         0x04000000

// 可以出现在 location 域中

#define NGX\_HTTP\_LOC\_CONF         0x08000000

// 可以出现在 upstream 域中

#define NGX\_HTTP\_UPS\_CONF         0x10000000

...

**解析指令函数**

预设的指令解析函数

// 处理 on | off 值

ngx\_conf\_set\_flag\_slot()

// 处理一个参数，转为 ngx\_str\_t

ngx\_conf\_set\_str\_slot()

// 处理多个参数，转为 ngx\_str\_t 数组

ngx\_conf\_set\_str\_array\_slot()

// 处理一个参数，转为整数

ngx\_conf\_set\_num\_slot()

...

**指令保存位置**

每个配置结构都可以在main，server，location 中有对应的元素，本字段说明该指令要保存在那个位置

// 保存在 main\_conf 中（main 域的配置）

#define NGX\_HTTP\_MAIN\_CONF\_OFFSET  offsetof(ngx\_http\_conf\_ctx\_t, main\_conf)

// 保存在 srv\_conf 中（server 域的配置）

#define NGX\_HTTP\_SRV\_CONF\_OFFSET   offsetof(ngx\_http\_conf\_ctx\_t, srv\_conf)

// 保存在 loc\_conf 中（location 域的配置）

#define NGX\_HTTP\_LOC\_CONF\_OFFSET   offsetof(ngx\_http\_conf\_ctx\_t, loc\_conf)

**配置变量地址**

// 一般使用 offsetof 宏指定变量地址，如：

offsetof(mysettings, setting1)