**Http请求处理函数**

**请求处理函数**

请求处理函数用于处理http请求，其原型如下：

typedef ngx\_int\_t (\*ngx\_http\_handler\_pt)(ngx\_http\_request\_t \*r);

返回值：nginx错误码

**请求结构体**

struct ngx\_http\_request\_s {

    // 环境数据

    void                            \*\*ctx;

    // main，server，location 域的配置

    void                            \*\*main\_conf;

    void                            \*\*srv\_conf;

    void                            \*\*loc\_conf;

    // 内存池

    ngx\_pool\_t                       \*pool;

    // 请求的开始时间

    time\_t                            start\_sec;

    // 当前处理阶段的索引号

    ngx\_int\_t                         phase\_handler;

    // 内容处理函数指针

    ngx\_http\_handler\_pt               content\_handler;

    ...

};

请求环境数据：

就像session一样，ctx的数据在整个请求生命周期都可以访问，其结构是void\*的数组，nginx提供了两个宏用于获取和设置模块的环境数据

// 获取模块的环境数据

#define ngx\_http\_get\_module\_ctx(r, module)  (r)->ctx[module.ctx\_index]

// 设置模块的环境数据

#define ngx\_http\_set\_ctx(r, c, module)      r->ctx[module.ctx\_index] = c;

**注册请求处理函数**

ngx\_http\_core\_module（http类型模块）的main域的配置结构包含了ngx\_http\_phase\_t数组，数组的下标是处理阶段的枚举，ngx\_http\_phase\_t则是该阶段注册的处理函数数组

typedef struct {

    // 处理函数数组

    ngx\_array\_t                handlers;

} ngx\_http\_phase\_t;

注册示例：

// 获取 ngx\_http\_core\_module 模块的 main 域的配置结构

auto cmcf = reinterpret\_cast<ngx\_http\_core\_main\_conf\_t\*>(

    ngx\_http\_conf\_get\_module\_main\_conf(cf, ngx\_http\_core\_module)

);

// 获取 NGX\_HTTP\_REWRITE\_PHASE 阶段的数组

auto &arr = cmcf->phases[NGX\_HTTP\_REWRITE\_PHASE].handlers;

// 添加一个处理函数到数组中

\*(reinterpret\_cast<ngx\_http\_handler\_pt\*>(ngx\_array\_push(&arr))) = handler;

**处理阶段**

Nginx定义了11个处理阶段，其定义如下：

typedef enum {

    // 读取 http 头后

    NGX\_HTTP\_POST\_READ\_PHASE = 0,

    // server 重新 url

    NGX\_HTTP\_SERVER\_REWRITE\_PHASE,

    NGX\_HTTP\_FIND\_CONFIG\_PHASE,

    // location 重新 url

    NGX\_HTTP\_REWRITE\_PHASE,

NGX\_HTTP\_POST\_REWRITE\_PHASE,

    // 检查访问权限前

    NGX\_HTTP\_PREACCESS\_PHASE,

    // 检查访问权限

    NGX\_HTTP\_ACCESS\_PHASE,

NGX\_HTTP\_POST\_ACCESS\_PHASE,

    // 访问静态文件，不能添加处理函数

    NGX\_HTTP\_TRY\_FILES\_PHASE,

    // 处理请求，产生内容

NGX\_HTTP\_CONTENT\_PHASE,

    // 记录日志

    NGX\_HTTP\_LOG\_PHASE

} ngx\_http\_phases;

**特殊的NGX\_HTTP\_CONTENT\_PHASE阶段**

NGX\_HTTP\_CONTENT\_PHASE阶段的模块比较多，如下指令

location /test {

    myndg\_test on;

    ...

}

当我们希望出现myndg\_test指令是就使用myndgmodule来处理该请求，而不用经过其他NGX\_HTTP\_CONTENT\_PHASE模块处理，怎么做到？如下：

static ngx\_command\_t ndg\_test\_cmds[] =

{

    {

        ngx\_string("myndg\_test"),

        NGX\_HTTP\_LOC\_CONF | NGX\_CONF\_FLAG,

        // 指定自定义的解析函数

        myfun,

        NGX\_HTTP\_LOC\_CONF\_OFFSET,

        offsetof(NdgTestConf, enabled),

        nullptr

    },

    // 配置指令必须以一个空指令结尾

    ngx\_null\_command

};

static char \* myfun(ngx\_conf\_t \*cf, ...){

    ...

    // 获取 ngx\_http\_core\_module 模块在当前 location 的配置

    auto coreConf = ngx\_http\_conf\_get\_module\_loc\_conf(cf, ngx\_http\_core\_module);

    // 指定当前 location 的请求处理函数为我们自定义的函数

    coreConf->handler = myhandler;

}

**Http过滤函数**

过滤函数用于处理响应数据，nginx提供了http header和http body过滤

过滤处理函数需要在过滤模块中实现

**过滤函数原型**

// 响应头数据过滤

typedef ngx\_int\_t (\*ngx\_http\_output\_header\_filter\_pt)(ngx\_http\_request\_t \*r);

// 响应 body 数据过滤

// chain 为 body 数据

typedef ngx\_int\_t (\*ngx\_http\_output\_body\_filter\_pt)(ngx\_http\_request\_t \*r, ngx\_chain\_t \*chain);

// 请求 body 数据过滤

typedef ngx\_int\_t (\*ngx\_http\_request\_body\_filter\_pt)(ngx\_http\_request\_t \*r, ngx\_chain\_t \*chain);

**过滤模块实现**

MyFilter.cpp文件

#include <iostream>

extern "C" {

#include <ngx\_http.h>

}

// 用于保存下一个过滤函数

static ngx\_http\_output\_header\_filter\_pt ngx\_http\_next\_header\_filter;

// 过滤处理函数

static ngx\_int\_t handler(ngx\_http\_request\_t \*r)

{

std::cout << "filter module" << std::endl;

    // 调用下一个过滤处理函数

    return ngx\_http\_next\_header\_filter(r);

}

// 初始化函数

static ngx\_int\_t init(ngx\_conf\_t\* cf)

{

    // 保存当前的过滤函数

    ngx\_http\_next\_header\_filter = ngx\_http\_top\_header\_filter;

    // 指定当前过滤函数为我们的函数

ngx\_http\_top\_header\_filter = handler;

    return NGX\_OK;

}

// 函数集合

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

    nullptr,

    init,                   // 指定模块初始化函数

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr

};

// 模块

ngx\_module\_t ndg\_test\_module = {

    NGX\_MODULE\_V1,

    &ndg\_test\_ctx,                      // 模块函数集合

    nullptr,

    NGX\_HTTP\_MODULE,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    NGX\_MODULE\_V1\_PADDING

};

**过滤函数链表头**

Nginx定义了过滤函数链表头，当我们添加过滤函数时，直接将链表头指向我们的过滤函数，然后在我们的过滤数据内调用之前的过滤函数即可

// 响应头

ngx\_http\_output\_header\_filter\_pt  ngx\_http\_top\_header\_filter;

// 响应体

ngx\_http\_output\_body\_filter\_pt    ngx\_http\_top\_body\_filter;

// 请求头

ngx\_http\_request\_body\_filter\_pt   ngx\_http\_top\_request\_body\_filter;

**config配置文件**

ngx\_addon\_name=ndg\_test\_module

# 这里使用的是 HTTP\_FILTER\_MODULES 而不是 HTTP\_MODULES

HTTP\_FILTER\_MODULES="$HTTP\_FILTER\_MODULES ndg\_test\_module"

NGX\_ADDON\_SRCS="$NGX\_ADDON\_SRCS $ngx\_addon\_dir/MyFilter.cpp"