cpp代码

#include <iostream>

#include <memory>

extern "C" {

#include <ngx\_http.h>

}

extern ngx\_module\_t ndg\_test\_module;

// 配置结构

struct NdgTestConf final

{

    // 是否启用

    ngx\_flag\_t enabled = NGX\_CONF\_UNSET;

};

// 创建配置结构函数

// nginx 需要我们自己创建配置结构

static void\* create(ngx\_conf\_t\* cf)

{

    // 内存池指针

auto pool = cf->pool;

    // 从内存池中分配内存

auto conf = reinterpret\_cast<NdgTestConf\*>(ngx\_pcalloc(pool, sizeof(NdgTestConf)));

    // 初始化该结构

\*conf = NdgTestConf();

    return conf;

}

// 配置指令，描述了配置使用的指令，用于配置解析

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

{

    {

        // 配置指令名

        ngx\_string("myndg\_test"),

        // 指令的作用域，和类型

        // NGX\_HTTP\_LOC\_CONF：只能出现在 locatiion 中

        // NGX\_CONF\_FLAG：可接受的参数 on | off

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

        // 解析指令的函数，这里使用标准函数

        ngx\_conf\_set\_flag\_slot,

        // 数据结构所在位置，这里指定位置为 http/location

        NGX\_HTTP\_LOC\_CONF\_OFFSET,

        // 指定该指令所对应的配置结构变量的地址

        offsetof(NdgTestConf, enabled),

        // 暂无需关心

        nullptr

    },

    // 配置指令必须以一个空指令结尾

    ngx\_null\_command

};

// 请求处理函数

// 当接收到 http 请求时，会调用该函数进行除了

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

{

    // 获取配置

auto cf = reinterpret\_cast<NdgTestConf\*>(ngx\_http\_get\_module\_loc\_conf(r, ndg\_test\_module));

    // 如果指令未开启，则不进行处理

    if(!cf->enabled){

        return NGX\_DECLINED;

}

    // 获取请求参数

auto data = r->args;

    // 填充响应头

    r->headers\_out.content\_length\_n = data.len;

    r->headers\_out.content\_type = ngx\_string("text/plain");

r->headers\_out.status = 200;

    // 填充响应体

    auto chain = ngx\_alloc\_chain\_link(r->pool);

    auto sendData = ngx\_create\_temp\_buf(r->pool,  data.len);

    chain->buf = sendData;

    std::uninitialized\_copy\_n(data.data, data.len, sendData->start);

sendData->last = sendData->start  + data.len;

    // 设置响应头和响应体

    ngx\_http\_send\_header(r);

ngx\_http\_output\_filter(r, chain);

    // 设置响应体数据结束

    return ngx\_http\_send\_special(r, NGX\_HTTP\_LAST);;

}

// 初始化函数

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

{

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

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

);

    // 在内容处理阶段注册函数

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

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

    create,             // 指定创建配置结构函数

    nullptr

};

// 模块

// ngx\_module\_t 用于描述我们模块

ngx\_module\_t ndg\_test\_module = {

    NGX\_MODULE\_V1,

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

    ndg\_test\_cmds,                      // 配置指令

    NGX\_HTTP\_MODULE,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    NGX\_MODULE\_V1\_PADDING

};