**目录结构**

├── MyNginx

│   ├── config

│   ├── MyNginx.conf

│   └── MyNginx.cpp

**模块**

Nginx使用ngx\_module\_t结构表示一个模块，因此我们编写模块的入口是ngx\_module\_t

// 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,                      // 模块所需函数集合

    ndg\_test\_cmds,                     // 指令

    NGX\_HTTP\_MODULE,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    nullptr,

    NGX\_MODULE\_V1\_PADDING

};

**函数集合**

// 函数集合

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

};

**解析配置文件后调用函数**

我们一般在解析配置文件后我们注册一个http请求处理函数

// 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));

ngx\_log\_error(NGX\_LOG\_ERR, r->pool->log, 0, "hello c++");

    // 根据配置决定输出

    if(cf->enabled){

        std::cout << "hello nginx" << std::endl;

    }

    else{

        std::cout << "hello disabled" << std::endl;

}

    // 执行成功，但未作处理，这样 nginx 会调用后续模块对该请求进行处理

    return NGX\_DECLINED;

}

// 在解析配置文件后被调用

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)

);

    // 注册请求处理函数

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

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

    return NGX\_OK;

}

**创建 location 域的配置结构函数**

如果我们想在配置文件location域里配置指令，那么必须使用一个结构来保存配置的指令，

Nginx希望由我们来创建这个配置结构

// 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;

}

**指令**

我想希望在location中配置这样一个指令

location /test {

myndg\_test off;

...

}

我们只需描述这个指令，nginx会将这个指令保存到我们的location 域的配置结构中

// 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

};

**编译脚本**

我们需要新建一个名为config文件（shell脚本），将模块添加到nginx中

# 模块的名称

ngx\_addon\_name=ndg\_test\_module

# 我们源码中模块的变量名

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

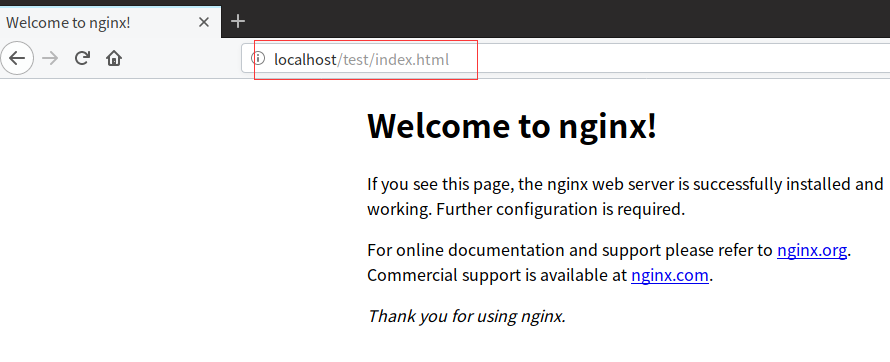
# 编译模块所需要的源文件

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

在nginx编译配置时指定模块

./configure --add-module=$HOME/Project/MyNginx

**验证**



浏览器访问：<http://localhost/test/index.html>

控制台输出：hello disabled

日志记录：hello c++

**以下为本篇文章的源码**

**MyNginx.cpp**

#include <iostream>

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));

ngx\_log\_error(NGX\_LOG\_ERR, r->pool->log, 0, "hello c++");

    std::cout << cf->enabled << std::endl;

    // 根据配置决定输出

    if(cf->enabled){

        std::cout << "hello nginx" << std::endl;

    }

    else{

        std::cout << "hello disabled" << std::endl;

}

    // 执行成功，但未作处理，这样 nginx 会调用后续模块对该请求进行处理

    return NGX\_DECLINED;

}

// 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)

);

    // 注册请求处理函数

    auto &arr = cmcf->phases[NGX\_HTTP\_REWRITE\_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

};

**MyNginx.conf**

master\_process off;

daemon off;

events {

    worker\_connections  1024;  ## Default: 1024

}

http

{

default\_type application/octet-stream; # 默认文件类型，默认为text/plain

    keepalive\_timeout 65;  # 连接超时时间，默认为75s，可以在http，server，location块。

    server {

        keepalive\_requests 120; # 单连接请求上限次数

        listen 80;            # 监听端口

        server\_name localhost; # 监听地址

        # 将所有请求转发至mysvr提供的服务器

        location /test {

            myndg\_test off;

            alias /usr/local/nginx/html;

            index index.html index.htm;

        }

    }

}