**Cpp**

#include <iostream>

#include <memory>

extern "C" {

#include <ngx\_http.h>

}

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

// 配置结构

struct MyUpStreamConf final

{

    // 上游配置结构

    ngx\_http\_upstream\_conf\_t upstream;

};

// 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)

{

MyUpStreamConf  \*conf;

    conf = reinterpret\_cast<MyUpStreamConf\*>(ngx\_pcalloc(cf->pool, sizeof(MyUpStreamConf)));

    if (conf == NULL) {

        return NULL;

}

\*conf = MyUpStreamConf();

    // 必须设置的四个参数

    conf->upstream.connect\_timeout = 30\*1000;

    conf->upstream.send\_timeout = 30\*1000;

    conf->upstream.read\_timeout = 30\*1000;

conf->upstream.buffer\_size = ngx\_pagesize;

    return conf;

}

void myFinalize(ngx\_http\_request\_t \*r, ngx\_int\_t rc){

    std::cout << "myFinalize" << std::endl;

}

ngx\_int\_t myCreateRequest(ngx\_http\_request\_t \*r){

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

    ngx\_str\_t rqStrs[] = {

        r->request\_line,

        ngx\_string("\r\n"),

        ngx\_string("Host: "),

        myconf->upstream.upstream->host,

        ngx\_string("\r\n"),

        ngx\_string("\r\n"),

        ngx\_null\_string

};

    int size = 0;

    for(int n = 0; rqStrs[n].len != NULL; n++ ){

        size = size + rqStrs[n].len;

}

auto buf = reinterpret\_cast<ngx\_buf\_t\*>(ngx\_create\_temp\_buf(r->pool, size));

    for(int n = 0; rqStrs[n].len != NULL; n++ ){

        std::uninitialized\_copy\_n(rqStrs[n].data, rqStrs[n].len, buf->last);

        buf->last = buf->last + rqStrs[n].len;

}

    auto chain = reinterpret\_cast<ngx\_chain\_t\*>(ngx\_pcalloc(r->pool, sizeof(ngx\_chain\_t)));

chain->buf = buf;

std::cout << buf->pos << std::endl;

r->upstream->request\_bufs = chain;

    return NGX\_OK;

}

ngx\_int\_t myProcessHeader(ngx\_http\_request\_t\* r){

std::cout << "myProcessHeader" << std::endl;

    return NGX\_OK;

}

// 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)

{

    // 启动upstream框架

    if(ngx\_http\_upstream\_create(r) != NGX\_OK){

        return NGX\_HTTP\_NOT\_ALLOWED;

}

    // 获取我们的配置结构

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

    // 将配置结构的上游配置结构赋给上游结构

r->upstream->conf = &myconf->upstream;

    // 生成请求头函数

    r->upstream->create\_request = myCreateRequest;

    // 处理相应头函数

    r->upstream->process\_header = myProcessHeader;

    // 转发完成函数

r->upstream->finalize\_request = myFinalize;

    // +1，防止请求结构在退出函数后销毁

r->main->count++;

    // 开始转发

ngx\_http\_upstream\_init(r);

    // 执行完成

    return NGX\_DONE;

}

// 字段解析函数

static char \* myupstream\_fun(ngx\_conf\_t \*cf, ngx\_command\_t \*cmd, void \*conf)

{

MyUpStreamConf  \*myconf = reinterpret\_cast<MyUpStreamConf \*>(conf);

    // 指令配置的参数

auto value = reinterpret\_cast<ngx\_str\_t\*>(cf->args->elts);

    // 分配一个 url 结构

    auto url = reinterpret\_cast<ngx\_url\_t\*>(ngx\_pcalloc(cf->pool, sizeof(ngx\_url\_t)));

    url->url = value[1];

url->no\_resolve = 1;

    // 指定上游配置结构的上游服务器配置

    myconf->upstream.upstream = ngx\_http\_upstream\_add(cf, url, 0);

    if(myconf->upstream.upstream == NULL){

        return reinterpret\_cast<char\*>(NGX\_CONF\_ERROR);

}

    // 指定当前location的处理函数为我们的处理函数

    auto httpCoreConf = reinterpret\_cast<ngx\_http\_core\_loc\_conf\_t\*>(ngx\_http\_conf\_get\_module\_loc\_conf(cf, ngx\_http\_core\_module));

httpCoreConf->handler = handler;

    return NGX\_CONF\_OK;

}

// 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[] =

{

    {

        ngx\_string("myupstream"),

        NGX\_HTTP\_LOC\_CONF  | NGX\_CONF\_TAKE1,

        // 指定自定义的解析函数

        myupstream\_fun,

        NGX\_HTTP\_LOC\_CONF\_OFFSET,

        0,

        nullptr

    },

    ngx\_null\_command

};

// 初始化函数

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

{

    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

        # 将所有请求转发至上游服务器

        location / {

            # 配置上游地址

            myupstream 192.168.31.169:8080;

        }

    }

}