请求转发由upstream框架实现，ngx\_http\_upstream\_module模块实现了该框架

**上游结构**

ngx\_http\_upstream\_module作为内容处理模块，其根据上游结构提供的数据进行转发，然后返回给客户

上游结构包含请求转发的关键数据，上游结构包含在请求结构中

struct ngx\_http\_request\_s {

    // 上游结构

    ngx\_http\_upstream\_t\* upstream;

    // 引用计数

    unsigned count:8;

    // 是否对收到的响应数据进行过滤处理

unsigned subrequest\_in\_memory:1;

    ...

}

**上游结构**

struct ngx\_http\_upstream\_s {

    // 连接结构体

    ngx\_peer\_connection\_t            peer;

    // 发送的请求数据

    ngx\_chain\_t                     \*request\_bufs;

    // 上游配置

    ngx\_http\_upstream\_conf\_t        \*conf;

    // 从上游接收到的数据

    ngx\_chain\_t                     \*out\_bufs;

    // 处理上游响应数据的过滤函数

    ngx\_int\_t                      (\*input\_filter\_init)(void \*data);

    ngx\_int\_t                      (\*input\_filter)(void \*data, ssize\_t bytes);

void                            \*input\_filter\_ctx;

    // 生成发送到上游的请求数据函数

    ngx\_int\_t                      (\*create\_request)(ngx\_http\_request\_t \*r);

    // 解析收到的响应头函数

    ngx\_int\_t                      (\*process\_header)(ngx\_http\_request\_t \*r);

    // 请求结束时的收尾工作

    void                           (\*finalize\_request)(ngx\_http\_request\_t \*r, ngx\_int\_t rc);

    ...

};

**上游配置**

typedef struct {

    // 上游服务器配置

    ngx\_http\_upstream\_srv\_conf\_t    \*upstream;

    // 连接超时时间

    ngx\_msec\_t                       connect\_timeout;

    // 发送超时时间

    ngx\_msec\_t                       send\_timeout;

    // 接收超时时间

ngx\_msec\_t                       read\_timeout;

    ...

} ngx\_http\_upstream\_conf\_t;

**开发upstream模块**

upstream模块主要在请求处理函数中配置上游结构即可

（完整示例代码查看下一章）

**必须配置的字段**

上游结构：

必须配置上游的三个函数

    // 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;

上游配置结构：

    // 连接超时时间

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

    // 发送超时时间

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

    // 接收超时时间

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

    // 缓存区大小

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

    // 指定上游服务器配置

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

**配置结构**

我们的配置结构直接保存上游配置结构，这样在请求处理函数中我们直接将该地址赋给上游结构的上游配置结果字段即可

// 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**

// 创建配置结构函数

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

{

    MyUpStreamConf  \*conf;

    // 从内存中分配一个结构

    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;

}

**指令数组**

// 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

};

**自定义的指令解析函数**

// 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;

}

**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)

{

    // 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;

}