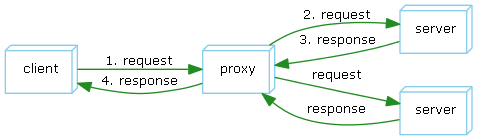
Nginx HTTP Proxy

# 概述

Nginx一个主要的用途是用作反向代理(Reverse Proxy)。



# 配置

## Upstream

上游服务器群的管理与组织。

upstream **news\_backend** {

server 192.168.1.100;

server 192.168.1.102;

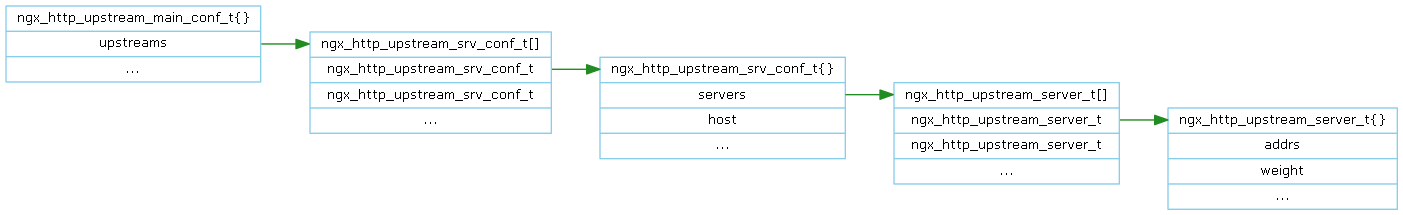
}

upstream **blog\_backend** {

server 192.168.1.200;

server 192.168.1.202:8080;

}

* ngx\_http\_upstream\_main\_conf\_t

参见<Nginx Module Configure>

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

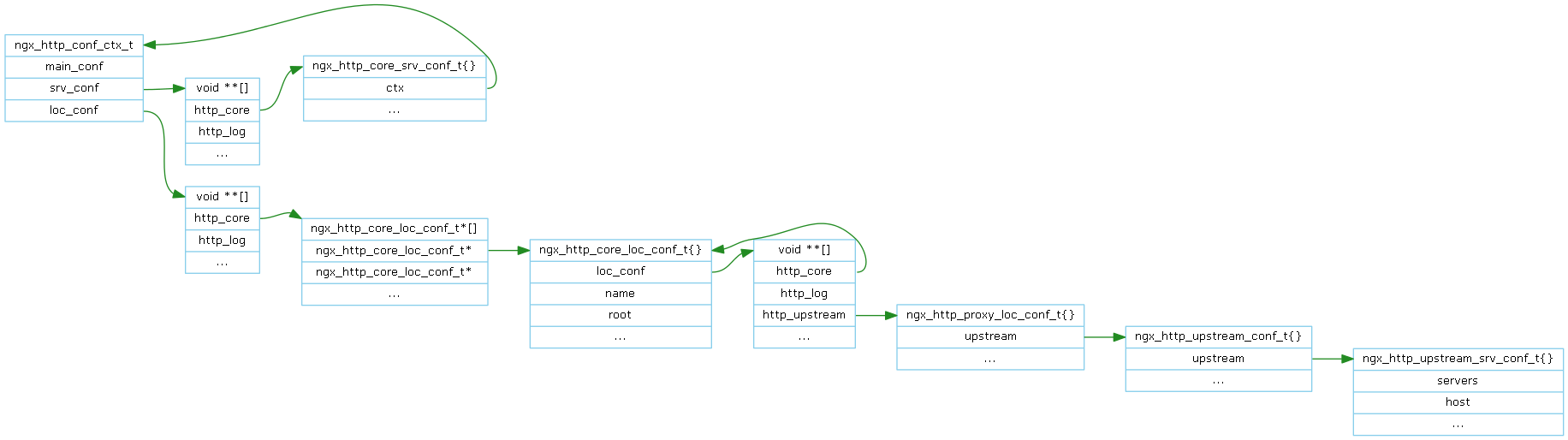
表示配置文件中的upstream news\_backend、upstream blog\_backend，上游服务器群。

* ngx\_http\_upstream\_server\_t

表示配置文件中的server 192.168.1.100、server 192.168.1.102等，一台上游服务器。关键数据是address。

## Location

管理Location与上游服务器群的关系。

* ngx\_http\_core\_loc\_conf\_t

参见<Nginx HTTP framework>

* ngx\_http\_proxy\_loc\_conf\_t

参见<Nginx HTTP framework>

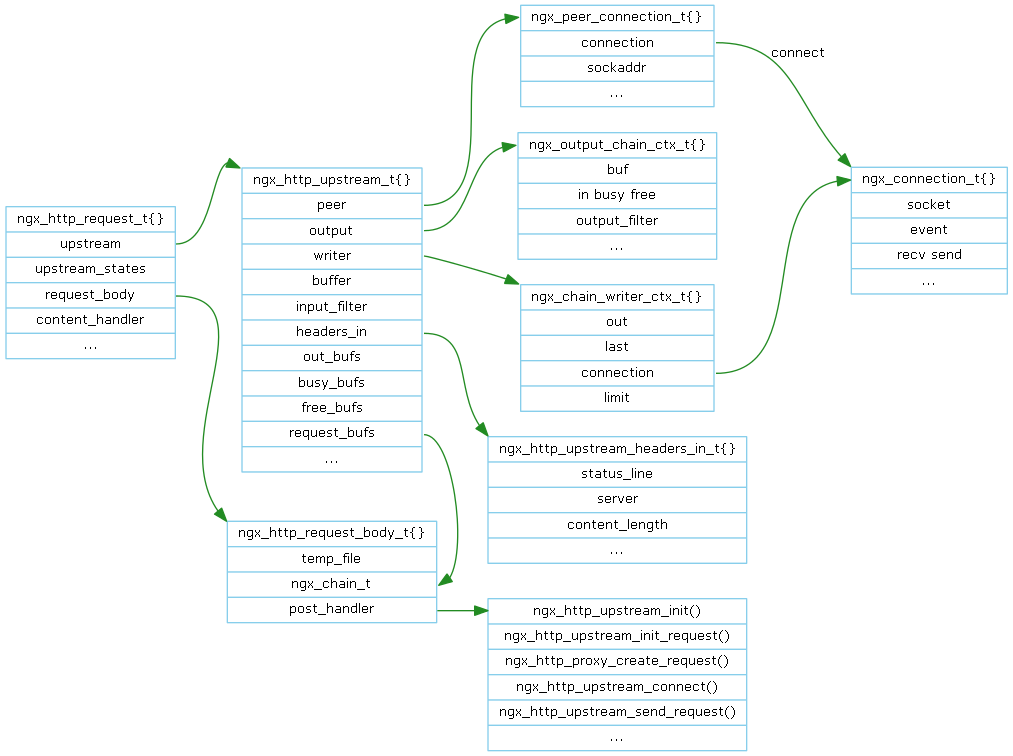
* ngx\_http\_upstream\_conf\_t

TODO

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

将Location与Upstream联系起来

# Request与Upstream

ngx\_upstream\_t是核心数据结构。

## ngx\_request\_t

代表Client的一个HTTP请求。参见<Nginx HTTP framework>。

## ngx\_http\_request\_body\_t

管理Request Body，包括数据以及回调函数。

## ngx\_upstream\_t

* request\_bufs

Client的请求数据。

* buffer

ngx\_buf\_t接收上游服务器发过来的数据。

* input\_filter

过滤上游服务器发过来的数据，并将数据持入out\_bufs。

* out\_bufs

需发送给Client的数据。

## ngx\_peer\_connection\_t

代表与上游服务器的连接。

## ngx\_output\_chain\_ctx\_t

管理发送给上游服务器的数据。

## ngx\_chain\_writer\_ctx\_t

向上游服务器发送数据。

## ngx\_http\_upstream\_headers\_in\_t

保存上游服务器发送过来的Header，它们将被发送给Client。

# 阶段处理

配置时Proxy\_module将Location的handler设为ngx\_http\_proxy\_handler()。

Find\_config阶段时，ngx\_request\_t::content\_handler被设为Location的handler即ngx\_http\_proxy\_handler()。

在Content阶段时，ngx\_http\_proxy\_handler()被调用，而忽略其它阶段回调函数。

ngx\_http\_proxy\_handler()依次做如下事情：

* 创建ngx\_upstream\_t
* 读取Client Request Body

# 读取Client Request Body

# 连接服务器

读取完Client Request Body，需要将Request Body转发到上游服务器，而这之前需要连接上游服务器。

连接由函数ngx\_http\_upstream\_connect()->ngx\_event\_connect\_peer()、ngx\_http\_upstream\_test\_connect()完成。

1. ngx\_event\_connect\_peer()

创建ngx\_connection\_t对象。调用非堵塞的系统调用connect()，connect()不一定立即连接成功（错误码EINPROGRESS）。

如何知道连接成功？ 当写事件触发，并且用getsockopt所得到的SO\_ERROR选项为0，则说明连接成功了。参见connect manual(man connect)。

1. ngx\_http\_upstream\_connect()

设置新连接读写事件的回调处理函数。

* ngx\_http\_upstream\_send\_request\_handler()
* ngx\_http\_upstream\_process\_header()

1. ngx\_http\_upstream\_test\_connect()

写事件触发，调用回调函数ngx\_http\_upstream\_send\_request\_handler()，利用ngx\_http\_upstream\_test\_connect()判断连接是否成功。

调用ngx\_http\_upstream\_send\_request()发送数据到上游服务器。

# 转发请求

# 读取并转发响应Header

上游Connection读事件触发，调用回调ngx\_http\_upstream\_process\_header()，将数据读入ngx\_upstream\_t::buffer。

调用ngx\_upstream\_t::process\_header解析数据中的Headers并写入ngx\_upstream\_t::headers\_in。

重复上述过程，直到Headers都读入。

调用ngx\_http\_upstream\_process\_headers()，将ngx\_upstream\_t::headers\_in加入ngx\_request\_t::headers\_out。

调用ngx\_http\_send\_header()，将headers加入ngx\_request\_t::out准备发给Client。

# 读取并转发响应Body

设置上游连接读事件的回调处理函数为ngx\_http\_upstream\_process\_non\_buffered\_upstream()。

设置下游连接写事件的回调处理函数为ngx\_http\_upstream\_process\_non\_buffered\_downstream()。

将数据读入ngx\_upstream\_t::buffer。

调用ngx\_upstream\_t::input\_filter将数据加入ngx\_upstream\_t::out\_bufs。

调用ngx\_http\_output\_filter()将out\_bufs挂入ngx\_request\_t::out，并发送。

重复上述过程，直到Body都读入并发完。

# Prototype

暂无

# Latest revision

https://github.com/lingjf/nginx\_analyse/blob/master/doc/