Nginx 四层代理功能主要部分文档翻译和注释

在 Nginx 中，四层的数据被称为 stream，和四层代理有关的模块主要有：

- `ngx_stream_core_module`：四层代理的基本功能模块
- `ngx_stream_upstream_module`：四层代理转发到上游的模块
- `ngx_stream_proxy_module`：四层代理相关配置

其他 stream 相关模块用于如 SSL 支持、geoip 支持、简单访问控制支持等，本次测试并没有使用到。

使用的注意事项

Nginx 的四层反代功能较为简单，其访问控制模块因为源站 IP 可以进行伪造，基本不可用于 UDP Flood 的防护。

使用健康检测功能的前提是他们在一个共享内存的 zone 里，注意各个区块的层次关系即可，zone 是配置上游服务器组共享内存的功能，因此 zone 要放在 upstream 区块。status 命令即监控 dashboard 是 ngx_http_status_module 的内容，严格来说不是四层代理的部分。

我使用的 UDP 反代配置：

```
stream {

    upstream dns_server {
        zone stream_dns 10m;
        server 127.0.0.1:53;
    }

    match dns {
        send \x00\x2a\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x06\x73\x65\x72\x76\
x65\x72\x0a\x73\x74\x61\x72\x64\x75\x73\x74\x65\x72\x02\x6d\x65\x00\x00\x01\x00\x0
1;
        expect ~* \x6a;
    }


    server {
        listen 10086 udp;
        proxy_pass dns_server;
        error_log  /home/nginx/dns-error.log debug;
        health_check udp match=dns interval=1s;
        status_zone stream_dns;
   }
}
```

健康检测页面配置：

```
server {
    listen 8080;

    root   /usr/share/nginx/html;

    location /status {
        status;
    }
    location = /status.html {
    }
}
```

ngx_stream_core_module

`ngx_stream_core_module` 模块从1.9.0版本开始提供，默认编译不包含此模块，需要在编译参数中加上 `--with-stream` 。

> 语法：listen address:port [ssl] [udp] [backlog=number] [bind] [ipv6only=onloff] [reuseport] [so_keepalive=onloffl [keepidle]:[keepintvl]:[keepcnt]];
> 环境：server

设置接受连接的端口和地址，可以只有端口号，也可以包含IP或主机名。

```
listen 127.0.0.1:12345;
listen *:12345;
listen 12345;     # same as *:12345
listen localhost:12345;
```

IPV6 地址需要使用中括号标注：

```
listen [::1]:12345;
listen [::]:12345;
```

Unix socket 需要使用前缀“Unix:”

`listen unix:/var/run/nginx.sock;`

参数表：

- `ssl` 参数所有连接都需要使用 SSL 加密；
- `udp` 参数指定监听 UDP 端口（从1.9.13版本开始支持）；

其他相关参数：

- `backlog=number` 参数设置 listen() 调用的最大等待连接队列数，默认情况下，backlog 在 BSD 上被设置为-1，在其他平台被设置为511；
- `bind` 参数使用所给的 address:port 产生一个独立的 bind() 调用，用于使用多个 listen 命令监听不同地址上的相同端口号；
- `ipv6only=on|off` 参数，配置 [::] 监听地址是否接受 IPV4 请求，此选项默认为 on；
- `reuseport` 参数（从1.9.1开始支持）为每个 worker 进程独立地创建一个监听的 socket（使用 SO_REUSEPORT），允许内核将入站连接分发到不同 worker 进程，仅支持 Linux3.9+ 和 DragonFly BSD；
- `so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]` 参数配置 TCP keepalive 特性；

不同的 Server 区块必须 listen 不同的地址+端口组合。

> 语法：resolver address ... [valid=time] [ipv6=on|off];
> 环境：server

配置解析 upstream 域名所使用的 DNS 服务器：

```
resolver 127.0.0.1 [::1]:5353;
```

地址可以以 IP 或者域名的形式给出，也可以附带端口号。默认情况下，nginx 会解析 V4 和 V6 地址，并会在 TTL 时间内缓存解析结果。

> 语法：resolver_timeout time;
> 默认：30秒
> 环境：stream，server
> （从1.11.3版本开始提供）

设置 DNS 解析超时时间。

> 语法：server {...}
> 环境：stream

设置一个独立的 Server。

> 语法：stream {...}
> 环境：main

提供四层代理的 Server 的配置环境，和 http {...} 类似。

> 语法：tcp_nodelay on | off;
> 默认：on
> 环境：stream，server
> （从1.9.4版本开始提供）

开启或关闭 TCP_NODELAY 选项，这个选项将会同时应用于到客户端和到上游服务器的连接。

> 语法：variables_hash_bucket_size size;
> 默认：64
> 环境：stream
> （从1.11.2版本开始提供）

设置变量 hash 表的桶大小，详细设置参见独立文档

> 语法：variables_hash_max_size size;
> 默认：1024
> 环境：stream （从1.11.2版本开始提供）

设置 hash 表的最大大小，详细设置参见独立文档

内置变量，（ngx_stream_core_module 模块从1.11.2开始支持变量）：

- $binary_remote_addr：二进制形式的客户端地址
- $bytes_sent：向客户端发送的字节数量
- $connection：连接序号
- $hostname：主机名
- $msec：以毫秒为分辨率的当前时间
- $nginx_version：nginx 的版本

- $pid：worker 的 PID
- $remote_addr：客户端地址
- $remote_port：客户端端口
- $server_addr：接受连接的服务端地址，获取此变量需要一次 system call，为避免产生 system call 在 listen 命令后指定监听地址并增加 bind 参数
- $server_port：接受连接的服务端端口
- $time_iso8601：ISO 8601 格式的本地时间
- $time_local：通常日志格式中的本地时间

ngx_stream_proxy_module

`ngx_stream_proxy_module` 允许被代理的数据流通过 TCP、UDP、Unix socket，从1.9.0版本开始提供。

> 语法：proxy_bind address [transparent] | off;
> 环境：stream, server
> （从1.9.2版本开始提供）

指定向外的连接通过指定的 IP，参数的值可以包含变量（1.11.2）。参数 off 可以取消从上一级继承的 proxy_bind 命令的效果，让系统自动选择出口 IP。

transparent（透明代理）参数可以发往源站（proxied server）的连接的源 IP 为非本地的 IP，如：客户端的真实 IP。 `proxy_bind $remote_addr transparent;` 为达到这个目的，nginx worker 进程需要以超级用户权限运行并配置内核路由表以拦截被源站发回的网络流量。

> 语法：proxy_buffer_size size;
> 默认：16k
> 环境：stream, server
> （从1.9.14版本开始提供）

设置从源站读取数据时候缓冲区的大小，也将同时设置从客户端读取数据的缓冲区大小。

> 语法：proxy_connect_timeout time;
> 默认：60s;
> 环境：stream, server

定义和源站已经建立的连接超时时间。

> 语法：proxy_download_rate rate;
> 默认：0;
> 环境：stream, server

从源站接受数据的速率限制，rate 的单位是 byte，默认数值为0即没有限制。这个限制是针对每个连接的，因此在打开两个到源站的连接时，总速率将会是 rate 值的两倍。

> 语法：proxy_next_upstream on | off;
> 默认：on;
> 环境：stream, server

当往源站的连接无法建立时，是否尝试将客户端的连接传往下一个源站。

尝试连接下一个源站的次数和时间都可被限制。

> 语法：proxy_next_upstream_timeout time;
> 默认：0;
> 环境：stream, server

限制连接到下一个源站时最大的尝试时间，默认数值为0 ，即没有限制。

> 语法：proxy_next_upstream_tires number;
> 默认：0;
> 环境：stream, server

限制连接到下一个源站时最大的尝试次数，默认数值为0 ，即没有限制。

> 语法：proxy_pass address;
> 环境：server

设置源站地址，地址可以以域名或 IP 和端口的形式出现，如 `proxy_pass localhost:12345;` ，也可以是 Unix socket 如 `proxy_pass unix:/tmp/stream.socket;` 。

当域名被解析为多个 IP 时，将使用 round-robin 方式进行轮询，也可以使用服务器组（ngx_stream_upstream_module 提供），如 `proxy_pass $upstream;`

在使用服务器组的情况下，server name 将首先在服务器组中搜索，如无结果则使用 resolver 进行查询。

> 语法：proxy_protocol;
> 默认：off;
> 环境：stream, server （从1.9.2版本开始提供）

启用PROXY proxy_protocol 连接到源站。

> 语法：proxy_response number;
> 环境：stream, server
> （从1.9.3版本开始提供）

在 UDP 代理启用的情况下，设置从源站期望接收的数据报数量，默认情况下接收报文的数量没有限制，将持续接收响应直到 proxy_timeout 超时。

> 语法：proxu_ssl on l off;
> 默认：off;
> 环境：stream, server

对和源站之间的连接启用 SSL。

> 语法：proxy_ssl_certificate file;
> 环境：stream, server

指定一个 PEM 格式的证书用于进行和源站的 SSL 连接认证。

> 语法：proxy_ssl_certificate_key file;
> 环境：stream, server

指定一个 PEM 格式的私钥用于进行和源站的 SSL 连接认证。

> 语法：proxy_ssl_ciphers ciphers;
> 默认：DEAFAULT;
> 环境：stream, server

指定和源站进行握手所使用的加密套件，以 OpenSSL 库可读的格式书写，使用命令 `openssl ciphers` 可以查看完整的列表。

> 语法：proxy_ssl_crl file;
> 环境：stream, server

指定吊销证书列表。

> 语法：proxy_ssl_name name; 默认：host from proxy_pass; 环境：stream, server

用于覆盖发往源站的 SNI 请求中的 server name，该 server name 可以包含变量（自1.11.3版本开始），默认情况下，proxy_pass 参数中的主机名将被使用。

> 语法：proxy_ssl_password_file file;
> 环境：stream, server

指定私钥的加密口令，每行一个，口令将在加载私钥文件的时候被依次尝试。

> 语法：proxy_ssl_session_reuse on | off;
> Default:on;
> Context:stream, server

指定和源站连接时，SSL session 是否被复用，如果出现“SSL3_GET_FINISHED”错误，请尝试关闭 SSL 会话复用。

> 语法：proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];
> 默认：TLSv1 TLSv1.1 TLSv1.2;
> 环境：stream, server

和源站连接时使用指定的 SSL 协议版本。

> 语法：proxy_ssl_trusted_certificate file;
> 环境：stream, server

指定 PEM 格式的可信的 CA 列表用于验证源站的认证有效性。

> 语法：proxy_ssl_verify on | off;
> 默认：off;
> 环境：stream, server

是否启用和源站之间的认证。

> 语法：proxy_ssl_verify_depth number;
> 默认：1;
> 环境：stream, server

指定验证源站认证的证书链层数。

> 语法：proxy_timeout time;
> 默认：10m;
> 环境：stream, server

指定两次读和写操作的超时间隔，此设置将同时应用在与客户端的连接和与源站的连接。若在这个时间内没有数据传输，连接将被关闭。

> 语法：proxy_upload_rate rate;
> 默认：0;
> 环境：stream, server （从1.9.3开始提供）

限制从客户端读取数据的速率，数字单位是 byte/s，默认值0表示没有限制。这个数值是针对每个连接的，因此打开两个连接时，数据传输速率将是这个值的两倍。

ngx_stream_upstream_module

`ngx_stream_upstream_module` 模块出现于1.9.0版本，用于定义可以被 `proxy_pass` 所调用的源站服务器组和相关的配置，收费订阅版本还支持动态配置组和主动健康检测。

> 语法：upstream name { ... }
> 环境：stream

定义一组服务器，服务器可以监听不同的端口，也可以监听 TCP 端口和 Unix Socket 的混合搭配：

```
upstream backend {
    server backend1.example.com:12345 weight=5;
    server 127.0.0.1:12345            max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend2;
    server backend3.example.com:12345 resolve;

    server backup1.example.com:12345  backup;
}
```

默认情况下，入站连接将被 round-robin 算法均分到每个后端，在上面这个配置了权重的示例中，平均每7个入站连接有5个将被转到 backend1，第二和第三个 backend 各接到一个连接；连接如果失败，将会尝试下一个 server，全部失败时连接将被关闭。

> 语法：server address [parameters];
> 环境：upstream

定义 server 的地址和参数，地址可以以 IP 或者域名的形式给出，此时端口号必须指明；也可以使用 `unix:` 前缀指明地址是个 Unix socket，如果域名被解析为多个 IP，则被认为是定义了多个服务器。

参数表：

- `weight=number` 参数指定了加权 round-robin 算法的权重值。
- `max_fails=number` 参数指定了最大失败重试次数。
- `fail_timeout=number` 参数同时指定了一次到后端 server 的连接的超时时间，和一个 server 被认为不可用的时间。
- `backup` 参数将该 server 标注为备用，在主要服务器全部不可用的时候连接将被传到 backup 服务器。
- do``wn 参数将手动将一台 server 标注为不可用。
- `max_conns=number` 限制向源站连接的最大连接数，默认为0，即没有限制。
- `resolve` 参数监视 server 域名对应的 IP 变化，而不需要在 upstream 中修改服务器配置也不需要重启 nginx，本功能要求服务器组在一个共享内存的 zone 中。使用这个参数必须在 stream 区块里有至少一个 resolver 命令：

  ```
  stream {
      resolver 10.0.0.1;

      upstream u {
          zone ...;
          ...
          server example.com:12345 resolve;
      }
  }
  ```

- `service=name` 参数将解析 DNS SRV 记录，要使用这个命令，需要 resolve 参数且 hostname 后不能含有端口号。
- `slow_start=time` 参数设定了一个被认为是不可用的 Server 恢复的时间，默认为0，即 slow_start 被关闭

注意，当一个服务器组里只有一个服务器的时候，max_fails fail_timeout slow_start 参数将会被忽略。

> 语法：zone name [size];
> 环境：upstream

一个 zone 使用 name 和 size 定义了一个共享的内存区域，workers 之间可以共享配置文件和实时状态，多个服务器组可以共享一个 zone，因此只需要声明一个 zone。

另外，作为收费订阅的一部分，动态配置组允许组内服务器在不完整重读配置文件的前提下更换或者修改组内成员，相关配置方式参见upstream_conf in ngx_http_stream_module。

> 语法：state file;
> 环境：upstream
> （从1.9.7版本开始提供）

指定一个文件路径，保存动态配置组当前配置状态，当前版本中这个状态包括配置组中的服务器列表和各个服务器的参数。这个文件在每次配置文件解析时被载入、upstream 配置变动的时候被更新，注意不应直接对这个文件进行编辑。

> 语法：hash key [consistent];
> 环境：upstream

负载均衡所使用的客户端-服务端映射的哈希算法是基于 key 的，hash key 可以包含文字和变量，
如 `hash $remote_addr` 。

> 语法：least_conn;
> 环境：upstream

使用“最少连接”负载均衡算法，nginx 将自动选择连接最少的后端，如果存在多个后端连接数相等，将使用加权的 round-robin 进行选择。

> 语法：least_time connect | first_byte | last_byte;
> 环境：upstream

使用“最快连接”负载均衡算法，nginx 将选择速度最快的后端，可以通过参数选择“最快”的定义是首包时间、传输完成时间还是连接建立时间。

> 语法：health_check [parameters];
> 环境：server

开启服务器组的周期性主动健康检测功能。

参数表：

- `interval=time` 参数设置了两次健康检查的间隔，单位是秒，默认值为5。
- `fails=number` 参数设置了健康检查连续失败达到指定次数后，将 Server 标记为不健康。
- `pass=number` 参数设置了健康检查连续通过达到指定次数后，将 Server 标记为健康。
- `match=name` 参数通过名称指定某个 match 区块为健康检测通过的条件，默认情况下，只会检测是否可以成功建立 TCP 连接。
- `port=number` 参数定义了进行健康检测时连接的端口号，默认情况等于 server 区块的端口号。
- `udp` 参数指定了进行 UDP 健康检测，此时必须指定 match 参数，并提供 send 和 expect 内容。

```
server {
    proxy_pass backend;
    health_check;
}
```

这个配置将会每5秒检测配置组中的每台服务器是否可以成功建立 TCP 连接，当连接建立失败，健康检测将不会通过并将服务器标记为不健康的。此时客户端将不会连接到不健康的服务器。

健康检测也可以指定发送的数据内容和期待接收的内容，这部分配置独立地在 match 命令中设置，并在 health_check 命令的 match 参数中进行引用。

服务器组必须在同一个共享内存区域里。

如果一个服务器组设置了多项健康检查，一项未能通过的检查就会将整个服务器组标记为不健康的。

> 语法：health_check_timeout timeout;
> 默认：health_check_timeout 5s;
> 环境：stream, server

对健康检测操作，以 health_check_timeout 的值覆盖 proxy_timeout 的值。

> 语法：match name { ... }
> 环境：stream

以名称定义一组验证健康检测返回值的测试集。

参数表：

- `send string` 参数定义发送给 server 的字符串。
- `expect string | ~ regex` 参数定义了服务器返回的字符串，可以使用正则进行匹配，如“~*”进行大小写不敏感匹配，使用“~”进行大小写敏感匹配.
    - send 和 expect 参数都可以使用“\x”前缀表示16进制字符，如“\x80\x1a”。
- 当满足下面条件时，健康检测通过：
    - TCP 连接建立
    - send 里的字符串成功发送
    - 服务器响应符合 expect
    - 未超出 health_check_timeout 指定的值

样例：

```
upstream backend {
    zone     upstream_backend 10m;
    server   127.0.0.1:12345;
}

match http {
    send     "GET / HTTP/1.0\r\nHost: localhost\r\n\r\n";
    expect ~ "200 OK";
}

server {
    listen       12346;
    proxy_pass   backend;
    health_check match=http;
}
```