Nginx 在启动后，在 unix 系统中会以 daemon 的方式在后台运行，后台进程包含一个 master 进程和多个 worker 进程。

在 Nginx 中 connection 就是对 tcp 连接的封装，其中包括连接的 socket，读事件，写事件。利用 Nginx 封装的 connection，我们可以很方便的使用 Nginx 来处理与连接相关的事情，比如，建立连接，发送与接受数据等。而 Nginx 中的 http 请求的处理就是建立在 connection之上的，所以 Nginx 不仅可以作为一个web服务器，也可以作为邮件服务器。当然，利用 Nginx 提供的 connection，我们可以与任何后端服务打交道

首先，Nginx 在启动时，会解析配置文件，得到需要监听的端口与 ip 地址，然后在 Nginx 的 master 进程里面，先初始化好这个监控的 socket(创建 socket，设置 addrreuse 等选项，绑定到指定的 ip 地址端口，再 listen)，然后再 fork 出多个子进程出来，然后子进程会竞争 accept 新的连接。此时，客户端就可以向 Nginx 发起连接了。当客户端与服务端通过三次握手建立好一个连接后，Nginx 的某一个子进程会 accept 成功，得到这个建立好的连接的 socket，然后创建 Nginx 对连接的封装，即 ngx\_connection\_t 结构体。接着，设置读写事件处理函数并添加读写事件来与客户端进行数据的交换。最后，Nginx 或客户端来主动关掉连接，到此，一个连接就寿终正寝了。

当然，Nginx 也是可以作为客户端来请求其它 server 的数据的（如 upstream 模块），此时，与其它 server 创建的连接，也封装在 ngx\_connection\_t 中。作为客户端，Nginx 先获取一个 ngx\_connection\_t 结构体，然后创建 socket，并设置 socket 的属性（ 比如非阻塞）。然后再通过添加读写事件，调用 connect/read/write 来调用连接，最后关掉连接，并释放 ngx\_connection\_t。

在 Nginx 中，每个进程会有一个连接数的最大上限，这个上限与系统对 fd 的限制不一样。在操作系统中，通过 ulimit -n，我们可以得到一个进程所能够打开的 fd 的最大数，即 nofile，因为每个 socket 连接会占用掉一个 fd，所以这也会限制我们进程的最大连接数，当然也会直接影响到我们程序所能支持的最大并发数，当 fd 用完后，再创建 socket 时，就会失败。Nginx 通过设置 worker\_connectons 来设置每个进程支持的最大连接数。如果该值大于 nofile，那么实际的最大连接数是 nofile，Nginx 会有警告。Nginx 在实现时，是通过一个连接池来管理的，每个 worker 进程都有一个独立的连接池，连接池的大小是 worker\_connections。这里的连接池里面保存的其实不是真实的连接，它只是一个 worker\_connections 大小的一个 ngx\_connection\_t 结构的数组。并且，Nginx 会通过一个链表 free\_connections 来保存所有的空闲 ngx\_connection\_t，每次获取一个连接时，就从空闲连接链表中获取一个，用完后，再放回空闲连接链表里面。

在这里，很多人会误解 worker\_connections 这个参数的意思，认为这个值就是 Nginx 所能建立连接的最大值。其实不然，这个值是表示每个 worker 进程所能建立连接的最大值，所以，一个 Nginx 能建立的最大连接数，应该是worker\_connections \* worker\_processes。当然，这里说的是最大连接数，对于 HTTP 请求本地资源来说，能够支持的最大并发数量是worker\_connections \* worker\_processes，而如果是 HTTP 作为反向代理来说，最大并发数量应该是worker\_connections \* worker\_processes/2。因为作为反向代理服务器，每个并发会建立与客户端的连接和与后端服务的连接，会占用两个连接。

**二、常用命令**

       nginx 的使用比较简单，就是几条命令。

[复制代码](javascript:void(0);)

nginx -s stop # 快速关闭Nginx，可能不保存相关信息，并迅速终止web服务。

nginx -s quit #平稳关闭Nginx，保存相关信息，有安排的结束web服务。

nginx -s reload #因改变了Nginx相关配置，需要重新加载配置而重载。

nginx -s reopen #重新打开日志文件。

nginx -c filename #为 Nginx 指定一个配置文件，来代替缺省的。

nginx -t #不运行，而仅仅测试配置文件。nginx 将检查配置文件的语法的正确性，并尝试打开配置文件中所引用到的文件。

nginx -v #显示 nginx 的版本。

nginx -V #显示 nginx 的版本，编译器版本和配置参数。

**三、http反向代理配置**

   我们先实现一个小目标：不考虑复杂的配置，仅仅是完成一个 http 反向代理。

  nginx.conf 配置文件如下

[复制代码](javascript:void(0);)

#运行用户

#user somebody;

#启动进程,通常设置成和cpu的数量相等

worker\_processes 1;

#全局错误日志

error\_log D:/Tools/nginx-1.10.1/logs/error.log;

error\_log D:/Tools/nginx-1.10.1/logs/notice.log notice;

error\_log D:/Tools/nginx-1.10.1/logs/info.log info;

#PID文件，记录当前启动的nginx的进程ID

pid D:/Tools/nginx-1.10.1/logs/nginx.pid;

#工作模式及连接数上限

events {

worker\_connections 1024; #单个后台worker process进程的最大并发链接数

}

#设定http服务器，利用它的反向代理功能提供负载均衡支持

http {

#设定mime类型(邮件支持类型),类型由mime.types文件定义

include D:/Tools/nginx-1.10.1/conf/mime.types;

default\_type application/octet-stream;

#设定日志

log\_format main '[$remote\_addr] - [$remote\_user] [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log D:/Tools/nginx-1.10.1/logs/access.log main;

rewrite\_log on;

#sendfile 指令指定 nginx 是否调用 sendfile 函数（zero copy 方式）来输出文件，对于普通应用，

#必须设为 on,如果用来进行下载等应用磁盘IO重负载应用，可设置为 off，以平衡磁盘与网络I/O处理速度，降低系统的uptime.

sendfile on;

#tcp\_nopush on;

#连接超时时间

keepalive\_timeout 120;

tcp\_nodelay on;

#gzip压缩开关

#gzip on;

#设定实际的服务器列表

upstream zp\_server1{

server 127.0.0.1:8089;

}

#HTTP服务器

server {

#监听80端口，80端口是知名端口号，用于HTTP协议

listen 80;

#定义使用www.xx.com访问

server\_name www.helloworld.com;

#首页

index index.html

#指向webapp的目录

root D:\01\_Workspace\Project\github\zp\SpringNotes\spring-security\spring-shiro\src\main\webapp;

#编码格式

charset utf-8;

#代理配置参数

proxy\_connect\_timeout 180;

proxy\_send\_timeout 180;

proxy\_read\_timeout 180;

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarder-For $remote\_addr;

#反向代理的路径（和upstream绑定），location 后面设置映射的路径

location / {

proxy\_pass http://zp\_server1;

}

#静态文件，nginx自己处理

location ~ ^/(images|javascript|js|css|flash|media|static)/ {

root D:\01\_Workspace\Project\github\zp\SpringNotes\spring-security\spring-shiro\src\main\webapp\views;

#过期30天，静态文件不怎么更新，过期可以设大一点，如果频繁更新，则可以设置得小一点。

expires 30d;

}

#设定查看Nginx状态的地址

location /NginxStatus {

stub\_status on;

access\_log on;

auth\_basic "NginxStatus";

auth\_basic\_user\_file conf/htpasswd;

}

#禁止访问 .htxxx 文件

location ~ /\.ht {

deny all;

}

#错误处理页面（可选择性配置）

#error\_page 404 /404.html;

#error\_page 500 502 503 504 /50x.html;

#location = /50x.html {

# root html;

#}

}

}

**四、负载均衡配置**

上一个例子中，代理仅仅指向一个服务器。

但是，网站在实际运营过程中，多半都是有多台服务器运行着同样的app，这时需要使用负载均衡来分流。

nginx也可以实现简单的负载均衡功能。

假设这样一个应用场景：将应用部署在 192.168.1.11:80、192.168.1.12:80、192.168.1.13:80 三台linux环境的服务器上。网站域名叫 www.helloworld.com，公网IP为 192.168.1.11。在公网IP所在的服务器上部署 nginx，对所有请求做负载均衡处理。

**nginx.conf** 配置如下：

[复制代码](javascript:void(0);)

http {

#设定mime类型,类型由mime.type文件定义

include /etc/nginx/mime.types;

default\_type application/octet-stream;

#设定日志格式

access\_log /var/log/nginx/access.log;

#设定负载均衡的服务器列表

upstream load\_balance\_server {

#weigth参数表示权值，权值越高被分配到的几率越大

server 192.168.1.11:80 weight=5;

server 192.168.1.12:80 weight=1;

server 192.168.1.13:80 weight=6;

}

#HTTP服务器

server {

#侦听80端口

listen 80;

#定义使用www.xx.com访问

server\_name www.helloworld.com;

#对所有请求进行负载均衡请求

location / {

root /root; #定义服务器的默认网站根目录位置

index index.html index.htm; #定义首页索引文件的名称

proxy\_pass http://load\_balance\_server ;#请求转向load\_balance\_server 定义的服务器列表

#以下是一些反向代理的配置(可选择性配置)

#proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

#后端的Web服务器可以通过X-Forwarded-For获取用户真实IP

proxy\_set\_header X-Forwarded-For $remote\_addr;

proxy\_connect\_timeout 90; #nginx跟后端服务器连接超时时间(代理连接超时)

proxy\_send\_timeout 90; #后端服务器数据回传时间(代理发送超时)

proxy\_read\_timeout 90; #连接成功后，后端服务器响应时间(代理接收超时)

proxy\_buffer\_size 4k; #设置代理服务器（nginx）保存用户头信息的缓冲区大小

proxy\_buffers 4 32k; #proxy\_buffers缓冲区，网页平均在32k以下的话，这样设置

proxy\_busy\_buffers\_size 64k; #高负荷下缓冲大小（proxy\_buffers\*2）

proxy\_temp\_file\_write\_size 64k; #设定缓存文件夹大小，大于这个值，将从upstream服务器传

client\_max\_body\_size 10m; #允许客户端请求的最大单文件字节数

client\_body\_buffer\_size 128k; #缓冲区代理缓冲用户端请求的最大字节数

}

}

}

## 六、https反向代理配置

一些对安全性要求比较高的站点，可能会使用 HTTPS（一种使用ssl通信标准的安全HTTP协议）。

这里不科普 HTTP 协议和 SSL 标准。但是，使用 nginx 配置 https 需要知道几点：

* HTTPS 的固定端口号是 443，不同于 HTTP 的 80 端口
* SSL 标准需要引入安全证书，所以在 nginx.conf 中你需要指定证书和它对应的 key

其他和 http 反向代理基本一样，只是在 Server 部分配置有些不同。

[复制代码](javascript:void(0);)

#HTTP服务器

server {

#监听443端口。443为知名端口号，主要用于HTTPS协议

listen 443 ssl;

#定义使用www.xx.com访问

server\_name www.helloworld.com;

#ssl证书文件位置(常见证书文件格式为：crt/pem)

ssl\_certificate cert.pem;

#ssl证书key位置

ssl\_certificate\_key cert.key;

#ssl配置参数（选择性配置）

ssl\_session\_cache shared:SSL:1m;

ssl\_session\_timeout 5m;

#数字签名，此处使用MD5

ssl\_ciphers HIGH:!aNULL:!MD5;

ssl\_prefer\_server\_ciphers on;

location / {

root /root;

index index.html index.htm;

}

}