```
# main 全局配置
#user username[groupname]
user username_xyz gropname_x ;

#Nginx worker进程个数：其数量直接影响性能
#每个worker进程都是单线程的进程，他们会调用各个模块以实现多种多样的功能。
#如果这些模块不会出现阻塞式的调用，那么，有多少CPU内核就应该配置多少个进程，
#反之，有可能出现阻塞式调用，那么，需要配置稍多一些的worker进程。

worker_processes 1 ;

##ssl硬件加速。
#用户可以用OpneSSL提供的命令来查看是否有ssl硬件加速设备：openssl engine -t
#ssl_engine device;

##守护进程(daemon)。是脱离终端在后台允许的进程。
#它脱离终端是为了避免进程执行过程中的信息在任何终端上显示。
#这样一来，进程也不会被任何终端所产生的信息所打断。
##关闭守护进程的模式，之所以提供这种模式，是为了放便跟踪调试nginx，
#毕竟用gdb调试进程时最繁琐的就是如何继续跟进fork出的子进程了。
##如果用off关闭了master_proccess方式，就不会fork出worker子进程来处理请求，
#而是用master进程自身来处理请求
#daemon off;    #查看是否以守护进程的方式运行Nginx 默认是on
#master_process off; #是否以master/worker方式工作 默认是on


##error日志的设置#
#语法：  error_log /path/file level;
#默认：  error_log / log/error.log error;
#当path/file 的值为 /dev/null时，这样就不会输出任何日志了，这也是关闭error日志的唯一手段;
#leve的取值范围是debug、info、notice、warn、error、crit、alert、emerg从左至右级别依次增大。
#当level的级别为error时，error、crit、alert、emerg级别的日志就都会输出。
#大于等于该级别会输出，小于该级别的不会输出。
#如果设定的日志级别是debug，则会输出所有的日志，这一数据量会很大，
#需要预先确保/path/file所在的磁盘有足够的磁盘空间。
#级别设定到debug，必须在configure时加入 --with-debug配置项。
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;


##pid文件（master进程ID的pid文件存放路径）的路径
#pid        logs/nginx.pid;

#事件类配置
events {
 #仅对指定的客户端输出debug级别的日志：  语法：debug_connection[IP|CIDR]
 #这个设置项实际上属于事件类配置，因此必须放在events{……}中才会生效。它的值可以是IP地址或者是CIRD地址。
        #debug_connection 10.224.66.14;  #或是debug_connection 10.224.57.0/24
 #这样，仅仅以上IP地址的请求才会输出debug级别的日志，其他请求仍然沿用error_log中配置的日志级别。
```

```
 #注意：在使用debug_connection前，需确保在执行configure时已经加入了--with-debug参数，否则不会生效。
        worker_connections  1024;
}

##核心转储(coredump):在Linux系统中，当进程发生错误或收到信号而终止时，
#系统会将进程执行时的内存内容(核心映像)写入一个文件(core文件)，
#以作为调试只用，这就是所谓的核心转储(coredump).

http {
##嵌入其他配置文件 语法：include /path/file
#参数既可以是绝对路径也可以是相对路径（相对于Nginx的配置目录，即nginx.conf所在的目录）
    include       mime.types;
    default_type  application/octet-stream;

    #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  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    #gzip  on;

    server {
##listen监听的端口
#语法：listen address:port [ default(deprecated in 0.8.21) | default_server | [ backlog=num | rc
# sndbuf=size | accept_filter=filter | deferred | bind | ssl ] ]
#default_server：如果没有设置这个参数，那么将会以在nginx.conf中找到的第一个server块作为默认server块
        listen       8080;

#主机名称：其后可以跟多个主机名称，开始处理一个HTTP请求时，nginx会取出header头中的Host，与每个server中的
         server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        #location / {
        #    root   html;
        #    index  index.html index.htm;
        #}

##location 语法： location [=|~|~*|^~] /uri/ { ... }
# location的使用实例见文末。
#注意：location时有顺序的，当一个请求有可能匹配多个location时，实际上这个请求会被第一个location处理。
        location / {
```

```
        proxy_pass http://192.168.1.60; # 反向代理
}

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}



# another virtual host using mix of IP-, name-, and port-based configuration
#
#server {
#    listen       8000;
#    listen       somename:8080;
#    server_name  somename  alias  another.alias;

#    location / {
#        root   html;
#        index  index.html index.htm;
#    }
#}
```

```
    # HTTPS server
    #
    #server {
    #    listen       443 ssl;
    #    server_name  localhost;

    #    ssl_certificate      cert.pem;
    #    ssl_certificate_key  cert.key;

    #    ssl_session_cache    shared:SSL:1m;
    #    ssl_session_timeout  5m;

    #    ssl_ciphers  HIGH:!aNULL:!MD5;
    #    ssl_prefer_server_ciphers  on;

    #    location / {
    #        root   html;
    #        index  index.html index.htm;
    #    }
    #}

}
```

server_name与host的匹配优先级

1. 首先选择字符串完全匹配的server_name
2. 选择通配符在前面的server_name
3. 选择通配符在后面的server_name
4. 选择正则表达式才能够匹配的server_name

location 匹配示例

> **1. 只有当用户请求是/时，才会使用该location下的配置**

```
location = {
    [configuration A]
}
```

> **2. 匹配所有的请求**

```
location / {
    [configuration A]
}
```

3. 匹配以/images/开头的任何查询并停止搜索（忽略字母大小写问题）

```
location ^~ /images/ {
    [configuration C]
}
```

4. 匹配任何以gif,jpg或jpeg结尾的请求

```
location ~* \.(gif|jpg|jpeg)$ {
    [ configuration D ]
}
```

5. 以root方式设置资源路径

```
location /download/ {
    root /opt/wab/
}
```

假如有一个请求URL为 /download/index/test.html， 那么Web服务器就会返回服务器上的/opt/wab/download/index/test.html文件的内容。

6. 以alias方式设置资源路径

alias也是用来设置文件资源路径的，和root的不同点在于，alias主要是解读loacation后面的uri参数。例如如果一个请求是/conf/nginx.conf ，而用户实际想要访问的是/usr/local/nginx/conf/nginx.conf。通过root只能在请求路径前面加上路径，通过alias可以解读路径。alias会把解析的字符串丢弃，而root不会。

```
    #alias
    location /conf {
        alias /usr/local/nginx/conf
    }
    #root
    location /conf{
        root /usr/local/nginx
    }
```

7. 以index方式访问首页

```
location / {
    root path;
    index /index.html /html/index.php /index.php
}
```

index后面可以跟多个文件参数，Nginx会按照顺序（从右到左的顺序）来访问文件。接收到请求后Nginx会首先尝试访问path/index.php , 如果可以访问，则结束请求。否则再尝试返回path/html/index.pho文件内容，以此类推。

通用模板

```
# 用户设置任何人
user nobody ;

# 设置nginx的worker数量
worker_processes 1 ;

# 设置错误日志
error_log logs/error.log


pid logs/nginx.pid ;

events {
    use epoll ;
    worker_connections 2048 ;
}

http{

    include       mime.types;
    default_type  application/octet-stream;
    sendfile on ;
    keepalive_timeout 65 ;

    # gzip压缩功能设置
    gzip on ;
    gzip_min_length 1k ;
    gzip_buffers 4 16K ;
    gzip_http_version 1.0;
    gzip_comp_level 6 ;
    gzip_types text/html text/plain text/css text/javascript application/json application/javasc
    gzip_vary on  ;

    # http_proxy设置
    client_max_body_size 10m ;
    client_body_buffer_size 128k;
    proxy_connect_timeout 75 ;
    proxy_send_timeout 75 ;
    proxy_read_timeout 75 ;
    proxy_buffer_size 4k ;
    proxy_buffers 4 32k ;
    proxy_busy_buffers_size 64k;
    proxy_temp_file_write_size 64k ;
    proxy_temp_path /usr/local/nginx/proxy_temp 1 2 ;

    # 设置负载均衡后台服务器列表
    upstream backend {
        # ip_hash
        server 192.168.10.100:8080 max_fails=2 fail_timeout=30s ;
        server 192.168.101:8080 max_fails=2 fail_timeout=30s;
```

```
    # 轮询
    #server 127.0.0.1:8080 weight=1 ;
        #server 127.0.0.1:8081 weight=1 ;
}

# 虚拟主机配置
server{
    listen 80 ;
    # 主机名称， 一般是域名
    server_name localhost ;
    root /apps/oaapp

    charset utf-8 ;
    access_log logs/host.access.log main;

    # 对 / 所有做负载均衡和反向代理
    location / {
        root /apps/oaapp
        index index.jsp index.html index.htm
    }

    proxy_pass http://backend ;  # 负载均衡
    proxy_redirect off ;

    # 后端的web服务器可以通过X-Forwarded-For获取用户的真实IP
    proxy_set_header Host $host ;
    proxy_set_header X-Real-IP $remote_addr ;
    proxy_set_header X-Forwarded-For $proxy_add_x_forward_for
    proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504 ;
}

# 静态文件， nginx服务器处理，不需要通过backend请求tomcat
location ~* /download/{
    root /apps/oa/fs ;
}

location ~.*\.(gif|jpg|jpeg|bmp|png|ico|txt|js|css)${
    root /apps/oaapp
    expires 7d;
}

location /nginx_status {
    stub_status on ;
    access_log off ;
    allow 192.168.10.0/24 ;
    deny all ;
}

location ~ ^/(WEB-INF)/{
    deny all ;
}
```

```
    # 设置error_page
    # redirect server error pages to the static page /50x.html
    error_page 500 502 503 504 /50x.html
    location = /50x.html{
        root html ;
    }

}
```

main全局配置

nginx在运行时与具体业务功能（比如http服务或者email服务代理）无关的一些参数，比如工作进程数，运行的身份等。

> woker_processes 2

在配置文件的顶级main部分，worker角色的工作进程的个数，master进程是接收并分配请求给worker处理。这个数值简单一点可以设置为cpu的核数grep ^processor /proc/cpuinfo | wc -l，也是 auto 值，如果开启了ssl和gzip更应该设置成与逻辑CPU数量一样甚至为2倍，可以减少I/O操作。如果nginx服务器还有其它服务，可以考虑适当减少。

> worker_cpu_affinity

也是写在main部分。在高并发情况下，通过设置cpu粘性来降低由于多CPU核切换造成的寄存器等现场重建带来的性能损耗。如worker_cpu_affinity 0001 0010 0100 1000; （四核）。

> worker_connections 2048

写在events部分。每一个worker进程能并发处理（发起）的最大连接数（包含与客户端或后端被代理服务器间等所有连接数）。nginx作为反向代理服务器，计算公式 最大连接数 = worker_processes * worker_connections/4，所以这里客户端最大连接数是1024，这个可以增到到8192都没关系，看情况而定，但不能超过后面的worker_rlimit_nofile。当nginx作为http服务器时，计算公式里面是除以2。

> worker_rlimit_nofile 10240

写在main部分。默认是没有设置，可以限制为操作系统最大的限制65535

> use epoll

写在events部分。在Linux操作系统下，nginx默认使用epoll事件模型，得益于此，nginx在Linux操作系统下效率相当高。同时Nginx在OpenBSD或FreeBSD操作系统上采用类似于epoll的高效事件模型kqueue。在操作系统不支持这些高效模型时才使用select。

http服务器

与提供http服务相关的一些配置参数。例如：是否使用keepalive啊，是否使用gzip进行压缩等。

> sendfile on

开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，减少用户空间到内核空间的上下文切换。对于普通应用设为 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，降低系统的负载。

> keepalive_timeout 65

长连接超时时间，单位是秒，这个参数很敏感，涉及浏览器的种类、后端服务器的超时设置、操作系统的设置，可以另外起一片文章了。长连接请求大量小文件的时候，可以减少重建连接的开销，但假如有大文件上传，65s内没上传完成会导致失败。如果设置时间过长，用户又多，长时间保持连接会占用大量资源。

> send_timeout

用于指定响应客户端的超时时间。这个超时仅限于两个连接活动之间的时间，如果超过这个时间，客户端没有任何活动，Nginx将会关闭连接。

> client_max_body_size 10m

允许客户端请求的最大单文件字节数。如果有上传较大文件，请设置它的限制值

> client_body_buffer_size 128k

缓冲区代理缓冲用户端请求的最大字节数

http_proxy

> proxy_connect_timeout 60

nginx跟后端服务器连接超时时间(代理连接超时)

> proxy_read_timeout 60

连接成功后，与后端服务器两个成功的响应操作之间超时时间(代理接收超时)

> proxy_buffer_size 4k

设置代理服务器（nginx）从后端realserver读取并保存用户头信息的缓冲区大小，默认与proxy_buffers大小相同，其实可以将这个指令值设的小一点

> proxy_buffers 4 32k

proxy_buffers缓冲区，nginx针对单个连接缓存来自后端realserver的响应，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_max_temp_file_size

当 proxy_buffers 放不下后端服务器的响应内容时，会将一部分保存到硬盘的临时文件中，这个值用来设置最大临时文件大小，默认1024M，它与 proxy_cache 没有关系。大于这个值，将从upstream服务器传回。设置为0禁用。

> proxy_temp_file_write_size 64k

当缓存被代理的服务器响应到临时文件时，这个选项限制每次写临时文件的大小。

> proxy_temp_path
> （可以在编译的时候）指定写到哪那个目录。

http_gzip

> gzip on

开启gzip压缩输出，减少网络传输。

> gzip_min_length 1k

设置允许压缩的页面最小字节数，页面字节数从header头得content-length中进行获取。默认值是20。建议设置成大于1k的字节数，小于1k可能会越压越大。

> gzip_buffers 4 16k

设置系统获取几个单位的缓存用于存储gzip的压缩结果数据流。4 16k代表以16k为单位，安装原始数据大小以16k为单位的4倍申请内存。

> gzip_http_version 1.0

用于识别 http 协议的版本，早期的浏览器不支持 Gzip 压缩，用户就会看到乱码，所以为了支持前期版本加上了这个选项，如果你用了 Nginx 的反向代理并期望也启用 Gzip 压缩的话，由于末端通信是 http/1.0，故请设置为 1.0。

> gzip_comp_level 6

gzip压缩比，1压缩比最小处理速度最快，9压缩比最大但处理速度最慢(传输快但比较消耗cpu)

> gzip_types

匹配mime类型进行压缩，无论是否指定,”text/html”类型总是会被压缩的。

> gzip_proxied any

Nginx作为反向代理的时候启用，决定开启或者关闭后端服务器返回的结果是否压缩，匹配的前提是后端服务器必须要返回包含”Via”的 header头。

> gzip_vary on

和http头有关系，会在响应头加个 Vary: Accept-Encoding ，可以让前端的缓存服务器缓存经过gzip压缩的页面，例如，用Squid缓存经过Nginx压缩的数据。。

server 虚拟机

http服务上支持若干虚拟主机。每个虚拟主机一个对应的server配置项，配置项里面包含该虚拟主机相关的配置。在提供mail服务的代理时，也可以建立若干server。每个server通过监听地址或端口来区分。

> listen

监听端口，默认80，小于1024的要以root启动。可以为listen *:80、listen 127.0.0.1:80等形式。

> server_name

服务器名，如localhost、www.example.com，可以通过正则匹配。

http_stream

这个模块通过一个简单的调度算法来实现客户端IP到后端服务器的负载均衡，upstream后接负载均衡器的名字，后端realserver以 host:port options; 方式组织在 {} 中。如果后端被代理的只有一台，也可以直接写在 proxy_pass 。

loaction

> root /var/www/html

定义服务器的默认网站根目录位置。如果locationURL匹配的是子目录或文件，root没什么作用，一般放在server指令里面或/下。

> index index.jsp index.html index.htm

定义路径下默认访问的文件名，一般跟着root放

> proxy_pass http:/backend

请求转向backend定义的服务器列表，即反向代理，对应upstream负载均衡器。也可以proxy_pass http://ip:port。

proxy_redirect off;
proxy_set_header Host $host; $proxy_set_header X-Real-IP remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

访问控制

Nginx的访问控制模块，通过allow和deny来表示。
可以有多个allow，deny，允许或禁止某个ip或ip段访问。
依次从上到下匹配，满足任何一个规则就停止往下匹配。

```
location /nginx-status {
    stub_status on ;
    # auth_basic "NginxStatus";
    # auth_basic_user_file /usr/local/nginx-1.6/htpasswd;
    access_log off ;
    allow 192.168.10.0/24 ;
     deny all ;
}
```

我们也常用 httpd-devel 工具的 htpasswd 来为访问的路径设置登录密码：

```
# htpasswd -c htpasswd admin
New passwd:
Re-type new password:
Adding password for user admin

# htpasswd htpasswd admin //修改admin密码
# htpasswd htpasswd sean //多添加一个认证用户
```

这样就生成了默认使用CRYPT加密的密码文件。打开上面nginx-status的两行注释，重启nginx生效。

列出目录 autoindex

Nginx默认是不允许列出整个目录的。如需此功能，打开nginx.conf文件，在location，server 或 http段中加入autoindex on;，另外两个参数最好也加上去:

autoindex_exact_size off; 默认为on，显示出文件的确切大小，单位是bytes。改为off后，显示出文件的大概大小，单位是kB或者MB或者GB
autoindex_localtime on;
默认为off，显示的文件时间为GMT时间。改为on后，显示的文件时间为文件的服务器时间

```
location /images {
    root /var/www/nginx-default/images;
    autoindex on;
    autoindex_exact_size off;
    autoindex_localtime on;
}
```