1. nginx配置文件路径：

不同安装方式，nginx的文件存放路径也有所不同。

1.1 源码安装配置文件路径：

在安装目录下的conf目录下，比如我的安装目录是/usr/local/nginx，那么他的配置文件就在/usr/local/nginx/conf目录下。

1.2 yum安装配置文件路径：

在/etc/nginx/目录（主配置文件）与/etc/nginx/conf.d目录下。

2. nginx配置文件的结构：

通常源码安装的nginx的配置文件，会是下面这种结构，yum安装的有细微差异（大致是一样的，只是server是通过include引用的独立配置文件）。

```
...
events {
        ...
}

http {
    ...
    server {
        ....
        location / {
            root html;
            ...
        }
    }
}
```

nginx的配置指令可以分为两大类：指令块与单个指令。

指令块就是像events，http，server等；

单独指令就是像root html;这样的。

nginx规定指令块可以嵌套，如http块中可以嵌套server指令，server块中可以嵌套location指令，指令可以同时出现在不同的指令块，如root指令可以同时出现在http、server和location指令块，需要注意的是在location中定义的指令会覆盖server，http的指令。

3. 解析配置文件：

3.1 全局配置：

```
user  nobody;
worker_processes  1;
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
#pid        logs/nginx.pid;

events {
    use epoll;
    worker_connections  1024;
}
```

```
user  :指定nginx的工作进程的用户及用户组，默认是nobody用户。

worker_processes    :指定工作进程的个数，默认是1个。具体可以根据服务器cpu数量进行设置，
                     比如cpu有4个，可以设置为4。如果不知道cpu的数量，可以设置为auto。
                     nginx会自动判断服务器的cpu个数，并设置相应的进程数。

error_log    :设置nginx的错误日志路径，并设置相应的输出级别。
              如果编译时没有指定编译调试模块，那么 info就是最详细的输出模式了。
              如果有编译debug模块，那么debug时最为详细的输出模式。这里设置为默认就好了。

pid      :指定nginx进程pid的文件路径。

events   :这个指令块用来设置工作进程的工作模式以及每个进程的连接上限。
```

use :用来指定nginx的工作模式，通常选择epoll，除了epoll，还有select,poll。
worker_connections　　:定义每个工作进程的最大连接数，默认是1024。

ps:进程的最大连接数受Linux系统进程的最大打开文件数限制。

修改文件描述符方式：
临时生效： ulimit -n 65535

在压测的时候，如果遇到报错 apr_socket_recv: Connection reset by peer (104)：
解决办法：
临时解决：
加一个-r参数，避免因为套接字错误退出，但是影响测试结果。

# 根本解决：
# vim /etc/sysctl.conf
net.ipv4.tcp_syncookies = 0

然后执行：sysctl -p

3.2 http指令块：

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    charset utf-8;
    #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;
    tcp_nodelay on;
    #keepalive_timeout  0;
    keepalive_timeout  65;
    keepalive_requests 100;
```

```
    #gzip  on;
    server {
        ...
        location {
            root html;
            ...
        }
    }

}
```

include   mime.types; 定义数据类型
    如果用户请求lutixia.png，服务器上有lutixia.png这个文件，后缀名是png；
    根据mime.types，这个文件的数据类型应该是image/png；
    将Content-Type的值设置为image/png，然后发送给客户端

default_type   :设定默认类型为二进制流，也就是当文件类型未定义时使用这种方式，
                      例如在没有配置PHP环境时，Nginx是不予解析的，
                      此时，用浏览器访问PHP文件就会出现下载窗口。
charset utf-8;   解决中文字体乱码
log_format   :定义日志文件格式，并默认取名为main，可以自定义该名字。
                   也可以通过添加，删除变量来自定义日志文件的格式。

access_log   :定义访问日志的存放路径，并且通过引用log_format所定义的main名称设置其输出格式。
sendfile   on :用于开启高效文件传输模式。直接将数据包封装在内核缓冲区，然后返给客户，将tcp_nopush和tcp_nodelay两个指令设置为on用于防止网络阻塞；
keepalive_timeout   65   :设置客户端连接保持活动的超时时间。在超过这个时间之后，服务器会关闭该连接。
keepalive_requests 100   :设置nginx在保持连接状态最多能处理的请求数，到达请求数，即使还在保持连接状态时间内，也需要重新连接。

提示：可以用netstat -ntlpa |grep 80 查看链接状态
gzip on   :开启压缩功能，减少文件传输大小，节省带宽。
gzip_min_length 1k; 最小文件压缩，1k起压。
gzip_types   text/plain text/xml;   压缩文件类型

```
gzip_comp_level    3;    压缩级别，默认是1。
```

3.3 server指令块：

```
server {
        listen       80;
        server_name  localhost;
        #charset koi8-r;
        #access_log  logs/host.access.log  main;
         index  index.html index.htm;
        location /
         {
            root   html;
            ...
        }
        #error_page  404              /404.html;
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
        #location ~ \.php$ {
                ...
        #}
        #location ~ /\.ht {
        #    deny  all;
        #}
    }
```

```
server       :用来定义虚拟主机。
listen       :设置监听端口，默认为80端口
server_name      :域名，多个域名通过逗号隔开
Charset      :设置网页的默认编码格式
access_log       :指定该虚拟主机的独立访问日志，会覆盖前面的全局配置。
index         :设置默认的索引文件
location        :定义请求匹配规则。
error_page       :定义访问错误返回的页面，凡是状态码是500 502 503 504 都会返回这个页面。
```

3.4 location指令块：

```
 #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;
        #}
```

```
location ~ \.php$      :凡是以php结尾文件，都会匹配到这条规则。
root         :php文件存放的目录
fastcgi_pass         :指定php-fpm进程管理的ip端口或者unix套接字
fastcgi_index    :指定php脚本目录下的索引文件
fastcgi_param        :指定传递给FastCGI服务器的参数
location ~ /\.ht         :凡是请求类似.ht资源，都拒绝。
```

4. 热部署(方案一):

4.1. 查看原编译参数:

```
# 升级一般是添加新的模块，或者升级版本，所以要参考以前编译的模块，如果不添加，那么以前的模块就不能使用了
[root@node3 ~]# /usr/local/nginx/sbin/nginx -V
```

4.2. 预编译/编译/安装:

```
./configure --prefix=/usr/local/nginx
make && make install
```

4.3. 直接升级:

```
make upgrade
```

5. 热部署(方案二):

5.1. 查看原编译参数:

```
# 升级一般是添加新的模块，或者升级版本，所以要参考以前编译的模块，如果不添加，那么以前的模块就不能使用了
[root@node3 ~]# /usr/local/nginx/sbin/nginx -V
```

5.2. 预编译/编译/安装:

```
./configure --prefix=/usr/local/nginx
make && make install
```

5.3. 生成新的master进程:

```
kill -USR2 `cat /usr/local/nginx/logs/nginx.pid`

[root@node3 nginx-1.16.0]# ps -ef |grep "[n]ginx"
root      8054     1  0 21:07 ?        00:00:00 nginx: master process /usr/local/nginx/sbin/nginx
nginx     8097  8054  0 21:09 ?        00:00:00 nginx: worker process
nginx     8098  8054  0 21:09 ?        00:00:00 nginx: worker process
root      8134  8054  0 21:13 ?        00:00:00 nginx: master process /usr/local/nginx/sbin/nginx
nginx     8135  8134  0 21:13 ?        00:00:00 nginx: worker process
nginx     8136  8134  0 21:13 ?        00:00:00 nginx: worker process
```

5.4. 优雅退出老worker进程:

```
# 向老的master进程发信号：
[root@node3 nginx-1.16.0]# kill -WINCH 8054
[root@node3 nginx-1.16.0]# ps -ef |grep "[n]ginx"
root      8054     1  0 21:07 ?        00:00:00 nginx:
master process /usr/local/nginx/sbin/nginx
root      8134  8054  0 21:13 ?        00:00:00 nginx:
master process /usr/local/nginx/sbin/nginx
nginx     8135  8134  0 21:16 ?        00:00:00 nginx:
worker process
nginx     8136  8134  0 21:16 ?        00:00:00 nginx:
worker process
```

5.5. 抉择：

5.5.1 回滚：

重新拉起老的worker进程：

```
[root@node3 nginx-1.16.0]# kill -HUP 8054
[root@node3 nginx-1.16.0]# ps -ef |grep "[n]ginx"
root      8054     1  0 21:07 ?        00:00:00 nginx:
master process /usr/local/nginx/sbin/nginx
root      8134  8054  0 21:13 ?        00:00:00 nginx:
master process /usr/local/nginx/sbin/nginx
nginx     8135  8134  0 21:16 ?        00:00:00 nginx:
worker process
nginx     8136  8134  0 21:16 ?        00:00:00 nginx:
worker process
nginx     8154  8054  1 21:19 ?        00:00:00 nginx:
worker process
nginx     8155  8054  1 21:19 ?        00:00:00 nginx:
worker process
```

退出新的master进程：

```
[root@node3 nginx-1.16.0]# kill -QUIT `cat
/usr/local/nginx/logs/nginx.pid`
[root@node3 nginx-1.16.0]# ps -ef |grep "[n]ginx"
root      8054     1  0 21:07 ?        00:00:00 nginx:
master process /usr/local/nginx/sbin/nginx
nginx     8154  8054  0 21:19 ?        00:00:00 nginx:
worker process
nginx     8155  8054  0 21:19 ?        00:00:00 nginx:
worker process
```

** 换回nginx文件：**

```
# 先删除新的nginx二进制文件：
rm -rf /usr/local/nginx/sbin/nginx
# 还原老的nginx 二进制文件：
mv /usr/local/nginx/sbin/nginx.old
/usr/local/nginx/sbin/nginx
```

5.5.2 新生：

```
# 经过一段时间测试，服务器没问题，退出老的master:
[root@node3 nginx-1.16.0]# kill -QUIT 8054
[root@node3 nginx-1.16.0]# ps -ef |grep "[n]ginx"
root      8134     1  0 21:24 ?        00:00:00 nginx:
master process /usr/local/nginx/sbin/nginx
nginx     8135  8134  0 21:24 ?        00:00:00 nginx:
worker process
nginx     8136  8134  0 21:24 ?        00:00:00 nginx:
worker process
```

常用模块：

1. access模块：

访问控制模块，该模块可以实现简单的防火墙功能，过滤特定的主机。这个模块在我们编译nginx时会默认编译进nginx的二进制文件中。

语法:

```
Syntax: allow address | CIDR | unix: | all;
Default:    —
Context:    http, server, location, limit_except
```

```
Syntax: deny address | CIDR | unix: | all;
Default:    —
Context:    http, server, location, limit_except
```

allow: 允许访问的IP或者网段。

deny: 拒绝访问的ip或者网段。

从语法上看，它允许配置在http指令块中，server指令块中还有locatio指令块中，这三者的作用域有所不同。

如果配置在http指令段中，将对所有server(虚拟主机)生效；

配置在server指令段中，对当前虚拟主机生效；

配置在location指令块中，对匹配到的目录生效。
ps: 如果server指令块，location指令块没有配置限制指令，那么将会继承http的限制指令，但是一旦配置了会覆盖http的限制指令。

或者说作用域小的配置会覆盖作用域大的配置。

1.2 http指令块配置：

对单ip进行限制：

在http指令块下配置单ip限制；

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    # 限制192.168.75.135这个ip访问
    deny 192.168.75.135;
    ...
}
```

ps: 配置完记得重载配置文件。

```
/usr/local/nginx/sbin/nginx -s reload
```

- 在自己服务上访问：

```
[root@localhost ~]# curl -I 192.168.75.130
HTTP/1.1 200 OK
Server: nginx/1.16.0
Date: Mon, 22 Jul 2019 02:24:19 GMT
Content-Type: text/html
```

- 在192.168.75.135机器上访问：

```
[root@localhost ~]# curl -I 192.168.75.130
HTTP/1.1 403 Forbidden
Server: nginx/1.16.0
```

可以看到只对135这个ip生效了，如果有配置虚拟主机，那么这个ip将都不能访问。

对网段进行限制：

如果想让整个网段都不能访问，只需要将这个**ip改为网段**即可。

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    # 将ip改为网段
    deny 192.168.75.0/24;
    ...
}
```

- 在自己服务器上访问；

```
[root@localhost ~]# curl -I 192.168.75.130
HTTP/1.1 403 Forbidden
Server: nginx/1.16.0
```

- 在192.168.75.135机器上访问：

可以看到都不能访问了，对整个网段的限制已生效。

1.3 server指令块配置：

配置方法都是一样的，只是作用范围不同。

对单ip进行限制：

```
 server {
        listen       80;
        server_name  localhost;
        deny 192.168.75.135;
        ...
    }
```

- 在自己服务器上访问：

```
[root@localhost ~]# curl -I 192.168.75.130
HTTP/1.1 200 OK
Server: nginx/1.16.0
Date: Mon, 22 Jul 2019 02:45:06 GMT
```

- 在192.168.75.135机器上访问：

```
[root@localhost ~]# curl -I 192.168.75.130
HTTP/1.1 403 Forbidden
Server: nginx/1.16.0
```

限制网段同上。

1.4 location指令块配置：

在location指令块配置访问控制。

这种配置是最多的，因为有时候我们要限制用户对某些文件或者目录的访问，这些文件通常是比较重要的或者私密的。

```
location /secret {
                deny 192.168.75.135;
        }

创建目录以及测试文件：
[root@localhost ~]# mkdir -p /usr/local/nginx/html/secret
[root@localhost ~]# echo "this is secret" >
/usr/local/nginx/html/secret/index.html
```

- 在本机访问：

```
[root@localhost ~]# curl 192.168.75.130/secret/index.html
this is secret
```

- 在192.168.75.135上访问：

1.5 白名单设置：

通过指定限制某个IP或者网段，这些形式是黑名单式的。

但如果想某些服务或者文件只针对于某些IP或者网段（通常是内网）开放，那么可以使用白名单，默认拒绝，指定的放行。

实例：

```
location /secret {
                allow 192.168.75.130;
                deny all;
        }
```

- 在本机访问：

```
[root@localhost ~]# /usr/local/nginx/sbin/nginx -s reload
[root@localhost ~]# curl 192.168.75.130/secret/index.html
this is secret
```

- 在192.168.75.135上访问：

可以看到，现在只有自己可以访问了，其他都被拒绝了。

2. auth_basic模块：

用户认证模块，对访问目录进行加密，需要用户权限认证：

这个功能特性是由ngx_http_auth_basic_module提供的，默认编译nginx时会编译好，主要有以下两个指令。

语法：

```
Syntax: auth_basic string | off;
Default:    auth_basic off;
Context:    http, server, location, limit_except
```

```
Syntax: auth_basic_user_file file;
Default:    –
Context:    http, server, location, limit_except
```

认证的配置可以在http指令块，server指令块，location指令块配置。

auth_basic string ：定义认证的字符串，会通过响应报文返给客户端，也可以通过这个指令关闭认证。
auth_basic_user_file file：定义认证文件。

2.1 对指定目录加密：

```
# 对匹配目录加密：
location /img {
          auth_basic "User Auth";
          auth_basic_user_file
/usr/local/nginx/conf/auth.passwd;
        }
```

```
auth_basic  设置名字
auth_basic_user_file  用户认证文件放置路径
```

生成认证文件：

```
yum  install httpd-tools -y
htpasswd  -c /usr/local/nginx/conf/auth.passwd admin
输入两次密码确定。
```

重启服务后，访问验证：

现在当我们访问img目录时，需要输入用户名及密码。

它不仅可以设置**访问指定目录时认证**，还可以直接在访问首页时认证。

如果需要关闭可以使用auth_basic off;或者直接将这两段注释掉。

htpasswd命令工具：

```
htpasswd(选项)(参数)
选项
-c：创建一个加密文件；
-m：默认采用MD5算法对密码进行加密；
```

```
-d：采用CRYPT算法对密码进行加密；
-p：不对密码进行进行加密，即明文密码；
-s：采用SHA算法对密码进行加密；
-b：在命令行中一并输入用户名和密码而不是根据提示输入密码；
-D：删除指定的用户。


添加用户名及密码：
htpasswd -b /usr/local/nginx/conf/auth.passwd  lutixia
lutixia666

更新密码：
htpasswd  -D  /usr/local/nginx/conf/auth.passwd  lutixia
htpasswd -b  /usr/local/nginx/conf/auth.passwd  lutixia
lutixia777
```

2.2 对首页加密：

```
# 对匹配目录加密：
location / {
         auth_basic "User Auth";
         auth_basic_user_file
/usr/local/nginx/conf/auth.passwd;
        }
```

3. stub_status 模块：

状态查看模块， 该模块可以 输出nginx的基本状态信息 。

语法：

```
Syntax: stub_status;
Default:    —
Context:    server, location
```

3.1 配置：

```
location = /status {
    stub_status;
    allow   192.168.75.130;
    deny    all;
}
```

```
Active connections:当前状态，活动状态的连接数
accepts：统计总值，已经接受的客户端请求的总数
handled：统计总值，已经处理完成的客户端请求的总数
requests：统计总值，客户端发来的总的请求数
Reading：当前状态，正在读取客户端请求报文首部的连接的连接数
Writing：当前状态，正在向客户端发送响应报文过程中的连接数
Waiting：当前状态，正在等待客户端发出请求的空闲连接数
```

4. referer 模块：

该模块可以进行防盗链设置。

盗链的含义是网站内容本身不在自己公司的服务器上，而通过技术手段，直接调用其他公司的服务器网站数据，而向最终用户提供此内容。

语法：

```
Syntax: valid_referers none | blocked | server_names |
string ...;
Default:    —
Context:    server, location
```

4.1 配置防盗链：

在130服务器上配置nginx防盗链：

```
location ~* \.(gif|jpg|png|swf|flv)$ {
                valid_referers none blocked  lutixia.net
*.jfedu.net;
                root   /usr/share/nginx/html;
                if ($invalid_referer) {
                        return 403;
                        }
                }

valid_referers 表示合法的referers设置
none： 表示没有referers，直接通过浏览器或者其他工具访问。
blocked： 表示有referers，但是被代理服务器或者防火墙隐藏；
lutixia.net： 表示通过lutixia.net访问的referers；
*.jfedu.net： 表示通过*.jfedu.net访问的referers，*表示任意host
主机。
```

防盗链测试:

找另外一台测试服务器，基于Nginx发布如下test.html页面，代码如下，去调用130官网的test.png图片，由于130官网设置了防盗链，所以无法访问该图片。

```
<html>
<h1>welcome to nginx</h1>
<img src="http://192.168.75.130/test.png">
</html>
```