# nginx

## 在线安装

安装nginx时需要openssl、pcre（为了重写）、zlib（gzip压缩）等的支持可能需要下载openssl、pcre、zlib等的安装包

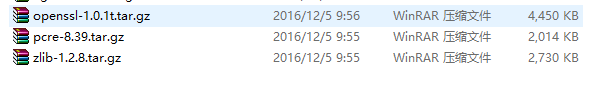
|  |
| --- |
| yum install -y gcc gcc-c++  yum install -y pcre pcre-devel  yum install -y zlib zlib-devel  yum install -y openssl openssl-devel |

## 离线安装

### 下载nginx安装包

C:\Users\RenKaiGe-Office\Desktop\html5图片\3.PNG

### 安装nginx时需要openssl、pcre（为了重写）、zlib（gzip压缩）等的支持可能需要下载openssl、pcre、zlib等的安装包,安装下面支持包时需要gcc编译环境



### 安装gcc编译环境

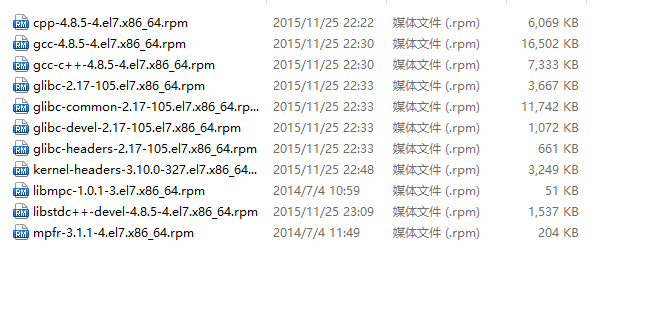
#### 查看linux版本号

cat /etc/redhat-release

#### 强制安装所有rpm并忽略警告

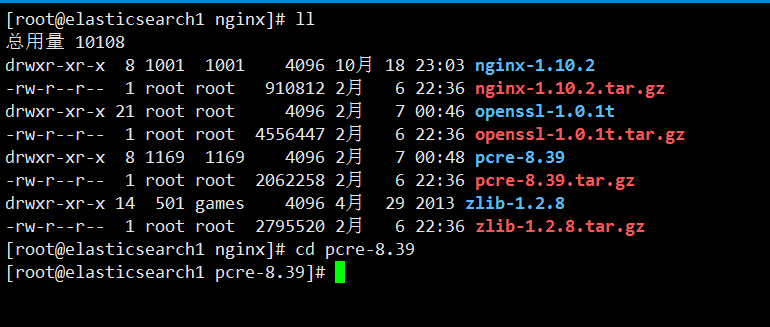
rpm -Uvh \*.rpm --nodeps --force

这些rpm可以去centos的安装包的packages文件夹中查找



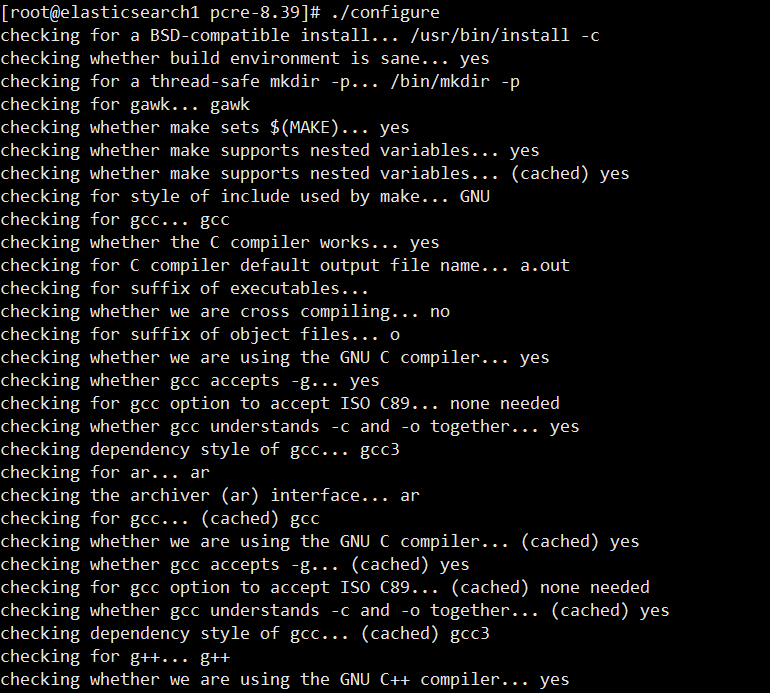
### 安装pcre(该包提供正则验证和http rewrite功能)

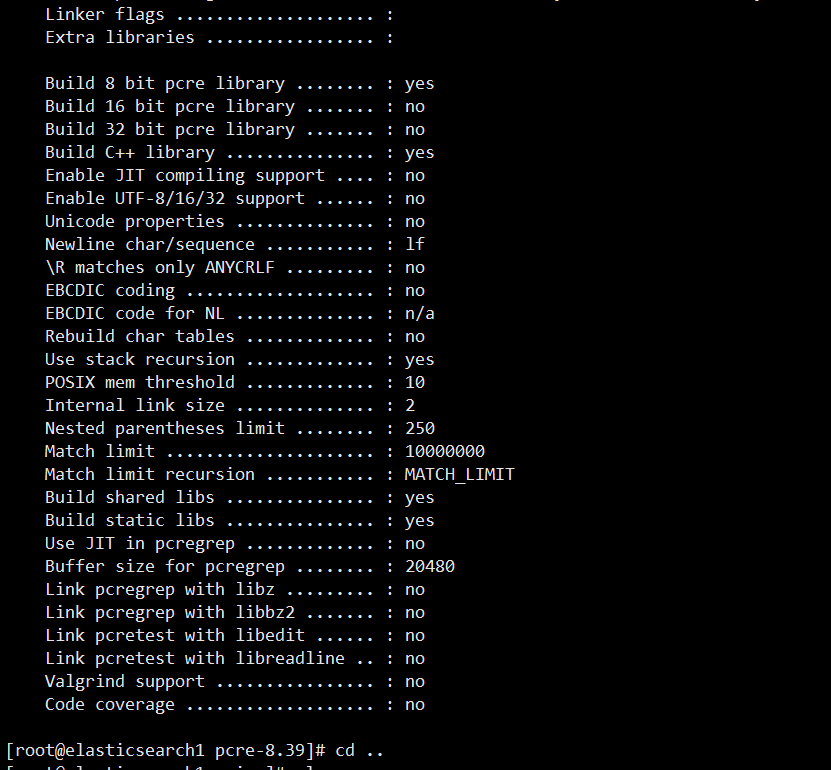
#### 解压压缩包,进入pcre-8.39目录



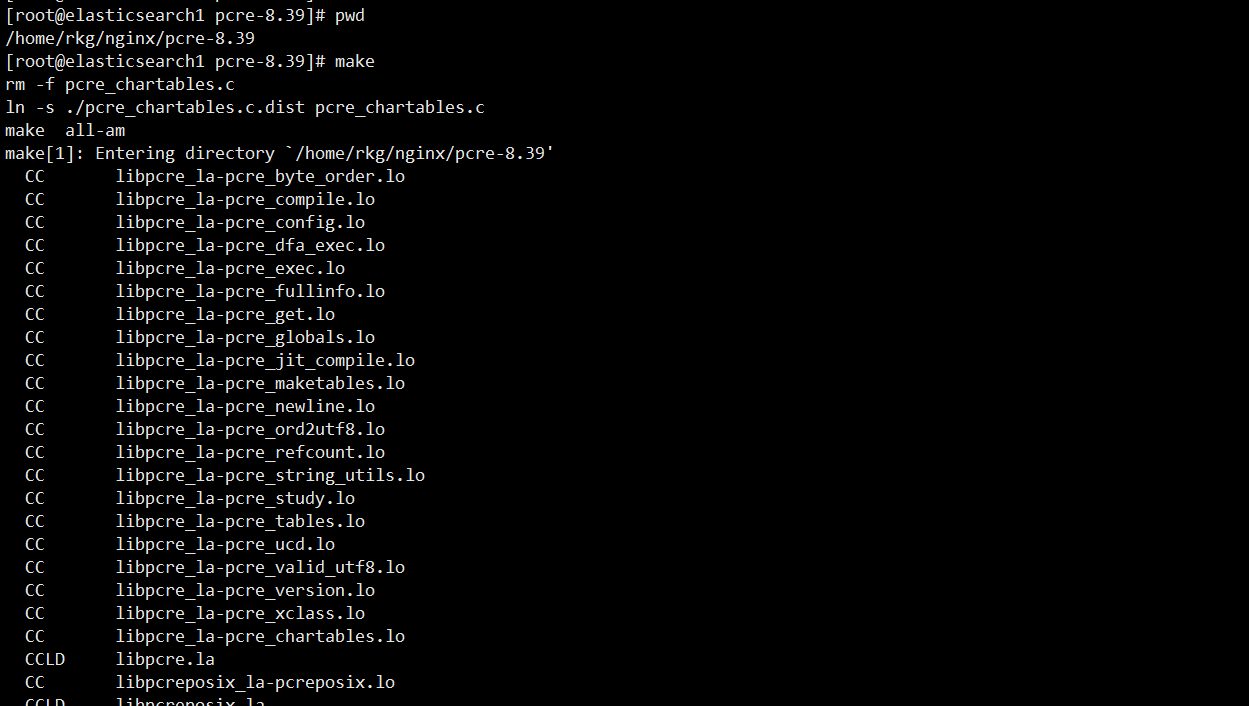
#### 执行configure命令(./configure)

./configure[--prefix=/home/rkg/install/pcre(指定pcre安装目录,如果此处指定的话,在安装nginx时也要指定pcre安装目录--with-pcre=/home/rkg/install/pcre),如果此处不指定的话在安装nginx时也无需指定] 不建议添加pcre路径

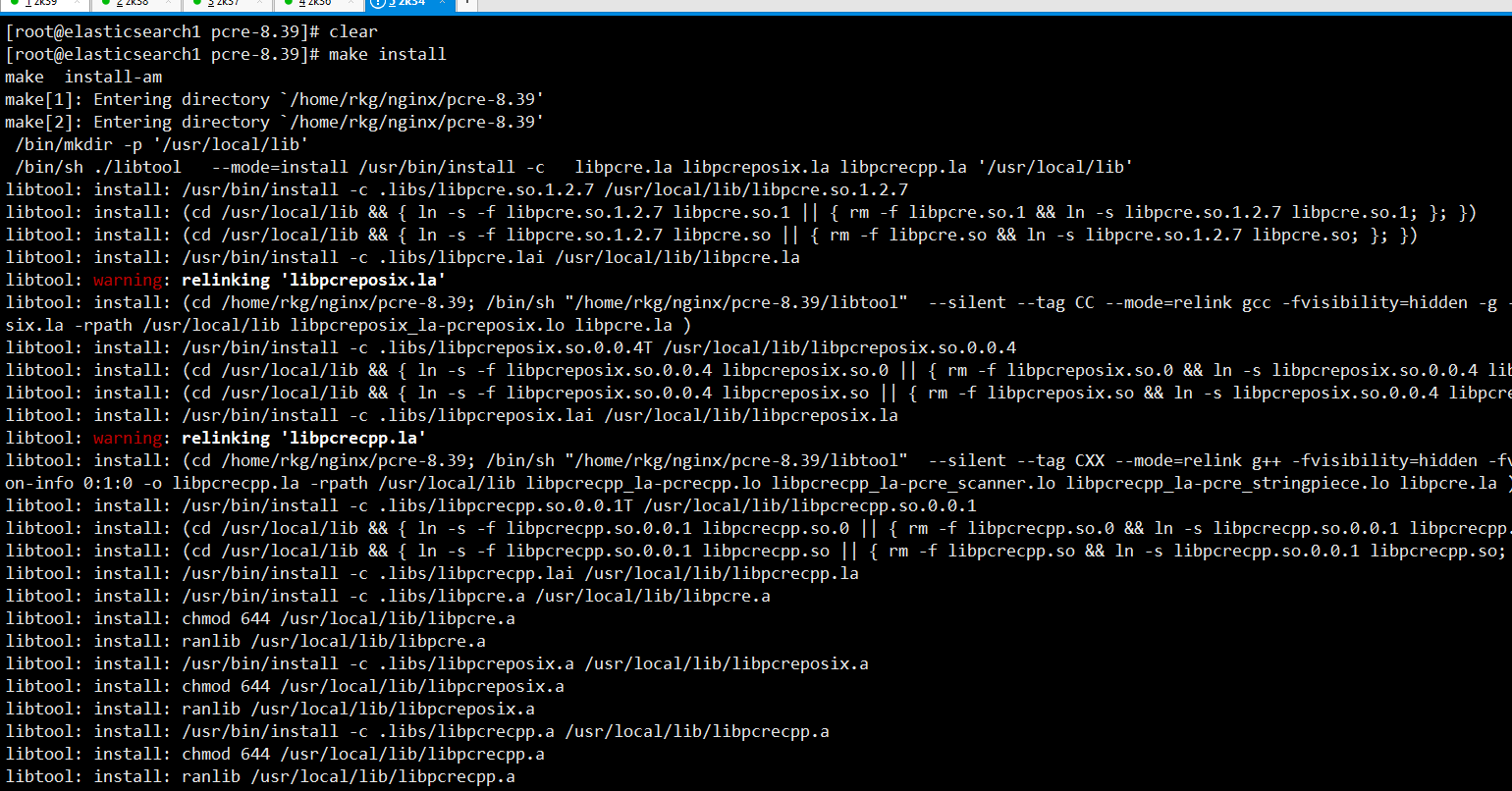




#### 执行make



#### 执行make install



### 安装zlib

同上安装pcre

### 安装openssl

这个需要执行./config不是./configure其他同安装pcre

## 安装nginx

同上安装pcre

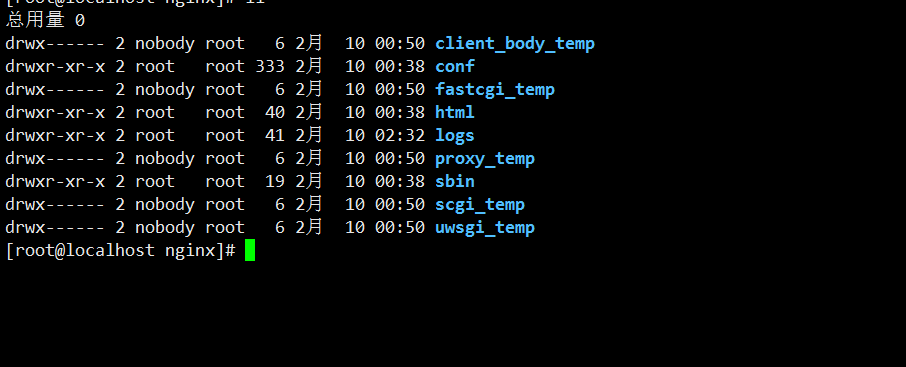
nginx默认安装目录

/usr/local/nginx

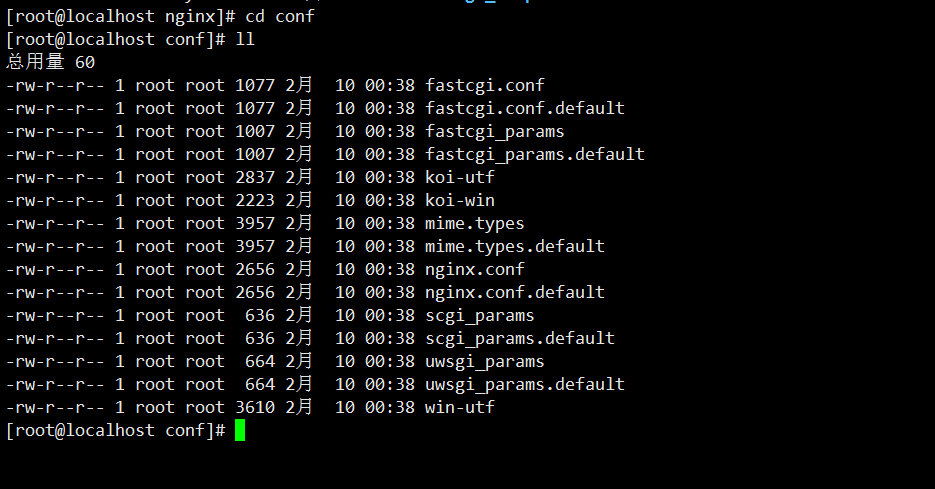
### nginx目录结构

安装后目录结构

<conf:配置文件目录>



nginx.conf,主配置文件

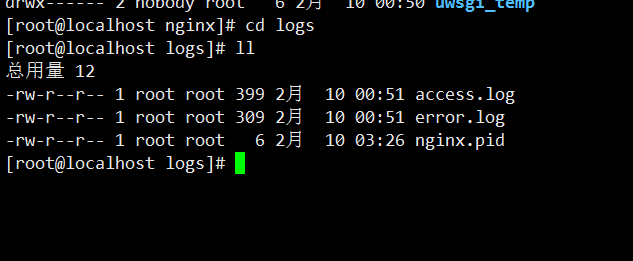


logs:日志文件目录

access.log:nginx的默认访问日志,记录所有的请求

error.log:记录错误的请求或启动错误日志

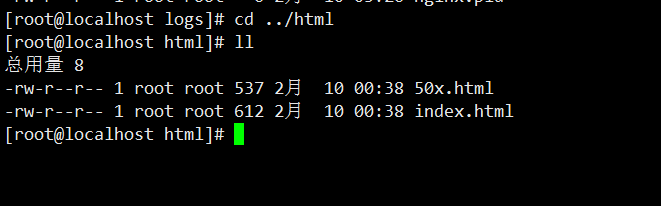
nginx.pid:记录所有启动的nginx进程的id号



html:访问时的默认欢迎界面根目录,类似于tomcat的/webapp/项目名称/

index.html:默认欢迎界面

50x.html:错误界面的替代页面,例如出现502错误时会调用此页面,可以通过error\_page参数来指定错误码对应的错误界面

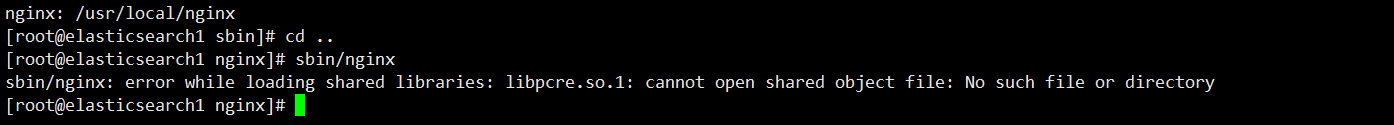


### 启动nginx

进入/usr/local/nginx/sbin目录下

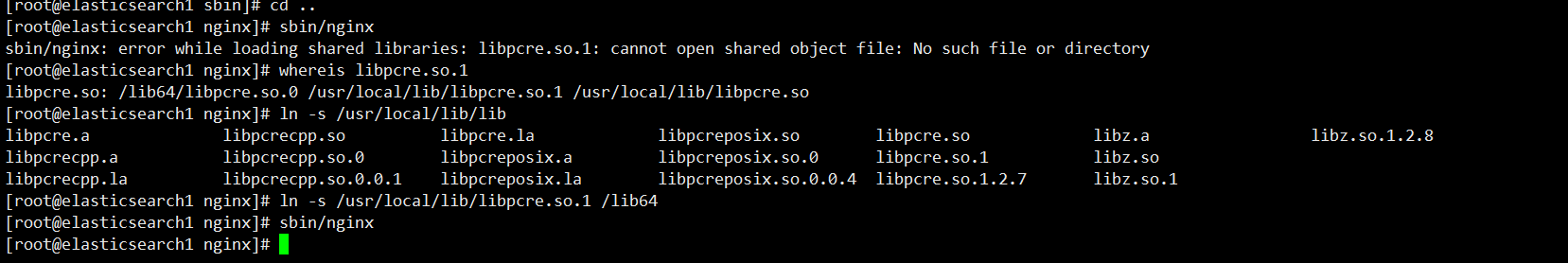
执行./nginx

如果出现如下错误时



可使用如下命令为libpcre.so.1创建软连接即可

ln -s /usr/local/lib/libpcre.so.1 /lib64



### 关闭/重启nginx等: ./nginx –s stop(quit,reload)

### 验证配置文件是否合法: ./nginx –t

### 命令帮助: ./nginx –h

### 在nginx.conf中添加引用其他配置文件

在nginx.conf最后一个大括号前面添加

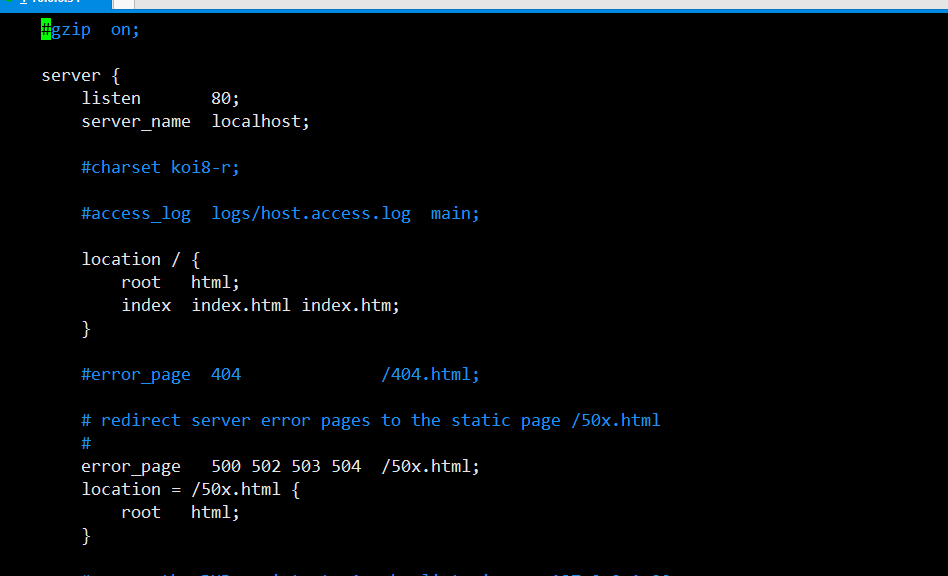
include /usr/local/nginx/extra/upstream01.conf;一定不要忘记分号

## nginx.conf属性讲解

### root,index

root:启动后通过ip地址访问nginx时的默认欢迎页面根目录(相对路径)

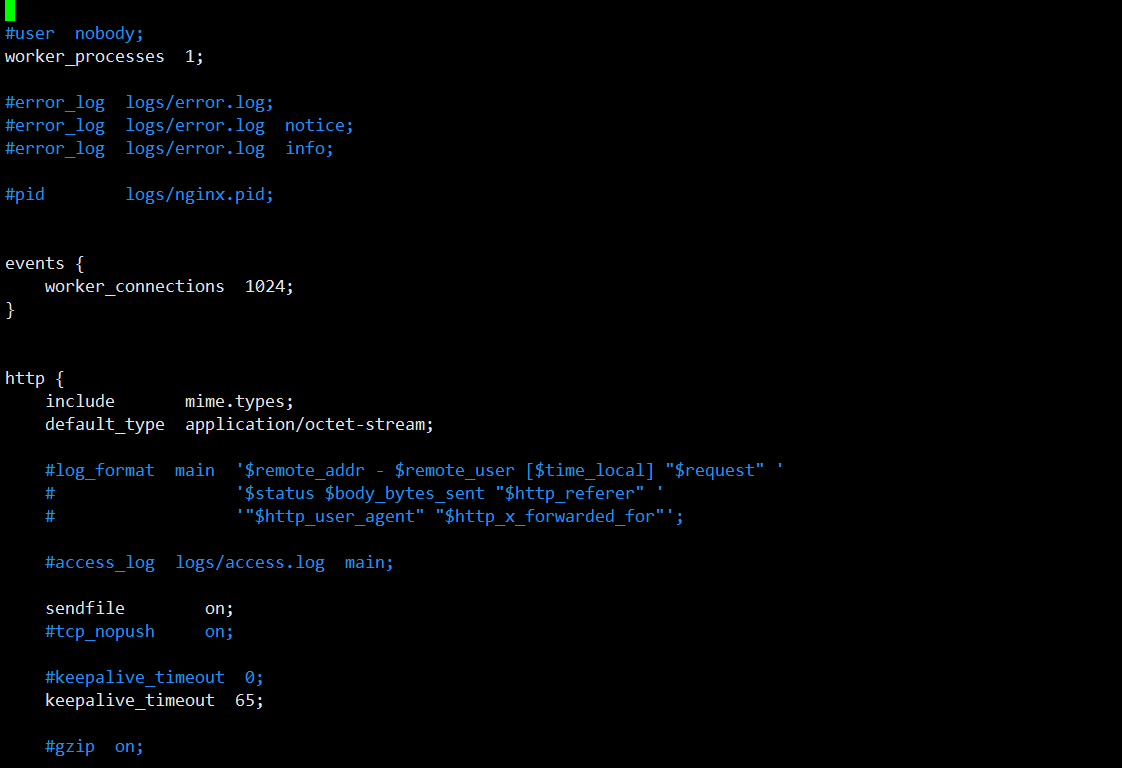
index:欢迎页面



### work\_processes:worker进程的个数(一般与cpu核心数相同)

### worker\_connections代表每个worker能够处理的请求数(与内存成正比)

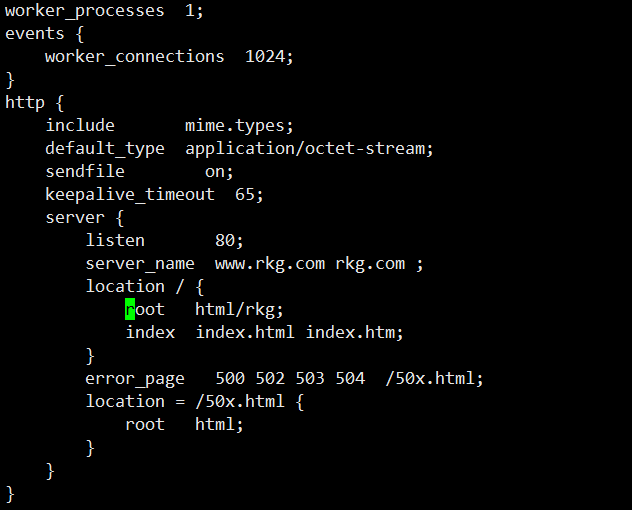
### include, default\_type保持默认属性值即可



## 基于域名的虚拟主机设置

## 设置访问192.168.11.161时可以通过[www.rkg.com或者rkg.com](http://www.rkg.com或者rkg.com)来访问

修改server如下所示,更改客户端hosts文件



## upstream模块介绍

定义一组服务器,这些服务器可以监听不同的端口,而且,监听在tcp和unix域套接字的服务器可以混用.例如

upstream backend{

server [www.baidu.com](http://www.baidu.com) weight=5;

server 127.0.0.1:8080 max\_fails=3 fail\_timeout=30s;

server unix:/home/rkg;

}

默认情况下,nginx按加权轮转的方式将请求分发到各服务器.在上面的例子中,每七个请求会通过以下方式分发:五个请求分发到[www.baidu.com](http://www.baidu.com),一个请求分发到第二个服务器,一个请求分到第三个服务器.与服务器通信的时候,如果出现错误,请求会被传给下一个服务器,直到所有可用的服务器都被尝试过.如果所有服务器都返回失败,客户端将会得到最后通信的那个服务器的(失败)响应结果.

### upstream语法

**upstream** rkg{

**server** [baidu.com](http://www.baidu.com) weight=5;

**server** 127.0.0.1:8080 max\_fails=3 fail\_timeout=30s weight=5;

**server** unix:/home/rkg weight=5 backup;

}

server {

location /{

proxy\_pass http://rkg;

}

}

#### 注意说明:

##### 加粗部分不能更改,是固定语法

##### 红色部分要保持一致

##### backup表示只有当前面的机器全部挂了之后才会启用(起热备功能)

##### upstream模块应该放置在nginx.conf的http标签内

##### upstream模块默认算法是wrr(权重轮询weighted round-robin)

### upstream模块内部部分参数介绍

|  |  |
| --- | --- |
| server 127.0.0.1:80 | 负载均衡后面的rs(真实服务器)配置,可以写ip或者ip+端口或者域名或者域名+端口或者服务器的路径地址等,端口不写默认是80,在高并发场景下建议尽量使用域名,内部通过dns映射解析到多台服务器,也起到负载均衡作用 |
| weight=1 | 权重,默认是1,两台机器1:1或者5:5分配的请求是一样的,只有权重大的才会分配的请求多 |
| max\_fails=1 | 失败后尝试再次连接的次数默认为1,0表示禁止失败后尝试,企业场景2-3,根据业务需求去配置 |
| fail\_timeout=20s | 失败超时时间(1:统计失败尝试次数的时间段,在这段时间中,服务器失败次数达到指定的尝试次数,服务器就被认为不可用.2:服务器被认为不可用的时间段),默认是10s,根据业务需求去配置,常规业务2-3s合理 |
| backup | 热备配置(rs节点的高可用),当前面激活的rs都失败后会自动启动热备rs |
| down | 设置服务器永远不可用,这个参数一致配置ip\_hash指令一起使用 |

### upstream算法介绍

#### rr轮询

按照客户端请求顺序把客户端请求逐一分配到不同的后端服务器,这相当于LVS中的rr算法,如果后端服务器宕机(默认情况下值检测80端口,如果后端报502,404,403,503等还是会直接返回给用户),宕机服务器会被自动剔除,使用户访问不受影响.请求会分配给正常的服务器

#### weight权重

在轮询算法的基础上加上权重(默认是rr+weight),权重轮询和访问成正比,权重越大,转发的请求也就越多,可以根据服务器的配置和性能指定权重值大小,可以有效解决新老服务器性能不均进行请求分配的问题.例如

A服务器配置:i7-7700k 16g内存

B服务器配置:i3-3300 4g内村

我希望在有30个请求到达时其中25个请求交给A处理,剩余5个交个B处理,就可以设置A的权重为5,B的权重为1

#### ip\_hash

每个请求按访问ip的hash结果分配.当新的请求到达时,先将其客户端ip通过hash算出一个值,在随后的请求中只要ip的hash值相同,就会被分配至同一台服务器,**该算法可以解决动态网页session共享问题**,但有时会导致请求分配不均,即无法保证1:1的负载均衡.在国内所有的公司都是NAT上午,多个pc对应一个外部ip,提示必须是最前端的服务器,后端也必须直接接应用服务器多数情况不能和权重参数一起使用

##### 示例

**upstream** rkg{

ip\_hash;

**server** [baidu.com](http://www.baidu.com);

**server** 127.0.0.1:8080 down;

**server** unix:/home/rkg ;

}

##### 注意

当负载调度算法为ip\_hash时,后端服务器在负载均衡配置中的状态不能是weight和backup(配置了也不生效)

##### 缺点

可能导致负载不均

#### fair(第三方插件,需要单独下载安装)

按照后端服务器的响应时间来分配请求,相应时间短的优先分配,比上面两个更加智能的负载均衡算法.此种算法可以依据页面大小和加载时间长短智能的进行负载均衡,也就是根据后端服务器的响应时间来分配请求,响应时间短的优先分配,nginx本身是不支持fair的如果需要使用这种调度算法,必须先下载nginx的upstream\_fair模块

示例

**upstream** rkg{

**server** [baidu.com](http://www.baidu.com);

**server** 127.0.0.1:8080 down;

**server** unix:/home/rkg ;

fair;

}

#### url\_hash(第三方插件,需要单独下载)

按照访问url的hash结果来分配请求,使每个url定向到同一台服务器,当后端服务器是缓存服务器时比较适合该种配置

注意当upstream中加入了hash语句后,server不能写入weight等其他参数了

提示:url\_hash用途cache服务业务,例如memcached,squid等,特点:每个rs都是不同的

#### least\_conn

最少连接数,哪个机器连接数少就分发到哪个机器

## location指令该指令属于server模块

为某个请求uri(路径)建立配置

路径匹配在uri规范化以后进行,所谓规范化,就是先将uri中的%XX等编码字符进行解码,在解析uri中的相对路径.和..部分,另外还可能会压缩相邻的两个或多个斜线成为一个斜线

### [=]精确匹配,如果找到匹配=号的内容,立即停止搜索,并立即处理请求(优先级最高)

### [~]区分大小写

### [^~]只匹配字符串,不匹配正则表达式

### [~\*]不区分大小写

### [@]指定一个命名的location,一般用于内部重定向请求,location @name{}

### 匹配过程

首先对字符串进行匹配查询,最确切的匹配将被使用,然后,正则表达式的匹配查询开始,匹配第一个结果后会停止搜索,如果没有找到正则表达式,将使用字符串的搜索结果,如果字符串和正则都匹配,那么正则优先级较高

示例

location =/ { ##只匹配(根)域名或ip,不匹配自路径

[ config A ]

}

location / { ##匹配所有该域名(ip)下的地址

[ config B ]

}

location /documents/ {

[ config C ]

}

location ^~/images/ {

[ config D ]

}

location ~\*\.(gif|jpg|jpeg)$ { ##不区分大小写,匹配jpg,gif,jpeg

[ config E ]

}

请求”/”匹配配置A

请求”/index.html”匹配配置B

请求”/documents/document.html”匹配配置C

请求”/images/1.gif”匹配配置D

请求”/documents/1.jpg”匹配配置E

匹配结果与location的顺序没有关系

### 匹配顺序(大概结论)

location = /index/1.jpg{}> location ~\* \.jpg&{}

location ~\* \.jpg&{}> location /index/1.jpg{}

location /index/1.jpg{}> location ^~/index/{}

location ~\* /index/{}> location /index/{}

location ^~/index/{}> location ~\* /index/{}

location /index/{}> location /{}

## ngx\_http\_proxy\_module(http\_proxy)模块属于location模块

proxy\_pass指令将请求转发到某台服务器,其值可以为上面配置的upstream名称,也可以是一个地址

proxy\_pass http://testservers;

proxy\_pass http://10.0.6.32/index;

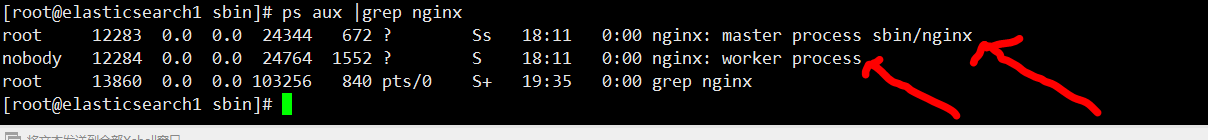
### 常见命令

|  |  |
| --- | --- |
| proxy\_set\_header | 设置由后端的服务器获取用户的主机名或者真实ip地址,以及代理者的真实ip地址 |
| client\_body\_buffer\_size | 用于指定客户端请求主体缓冲区大小,可以理解为先保存到本地再传给用户 |
| proxy\_connect\_timeout | 表示与后端服务器连接的超时时间,即发起握手等候响应的超时时间 |
| proxy\_send\_timeout | 表示后端服务器的数据回传数据,即在规定时间之内后端服务器必须传完所有的数据,否则nginx将断开这个连接 |
| proxy\_read\_timeout | 设置nginx从代理的后端服务器获取信息的时间,表示连接建立成功后,nginx等待后端服务器的响应时间,其实是nginx已经进入后端的排队之中等候处理的时间 |
| proxy\_buffer\_size | 设置缓冲区大小,默认,该缓冲区大小等于指令proxy\_buffers设置的大小 |
| proxy\_buffers | 设置缓冲区的数量和大小,nginx从后端服务器获取的响应信息会放置到缓冲区 |

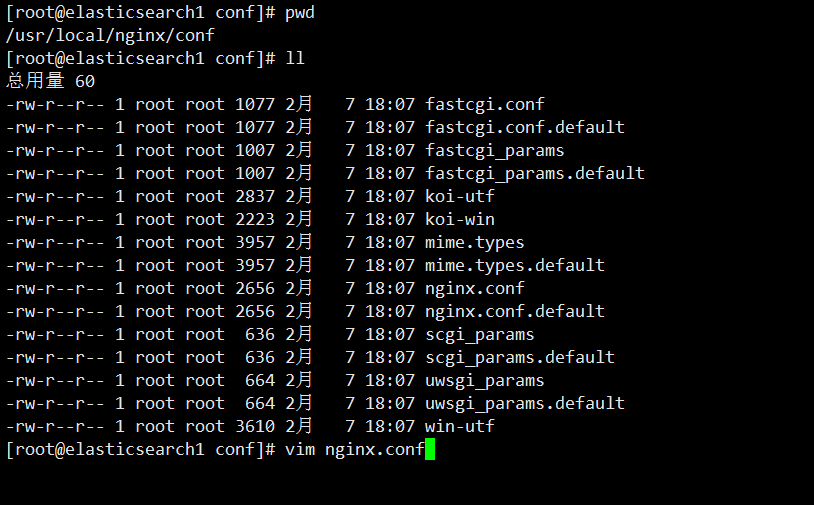
## nginx进程模型

### 进程介绍

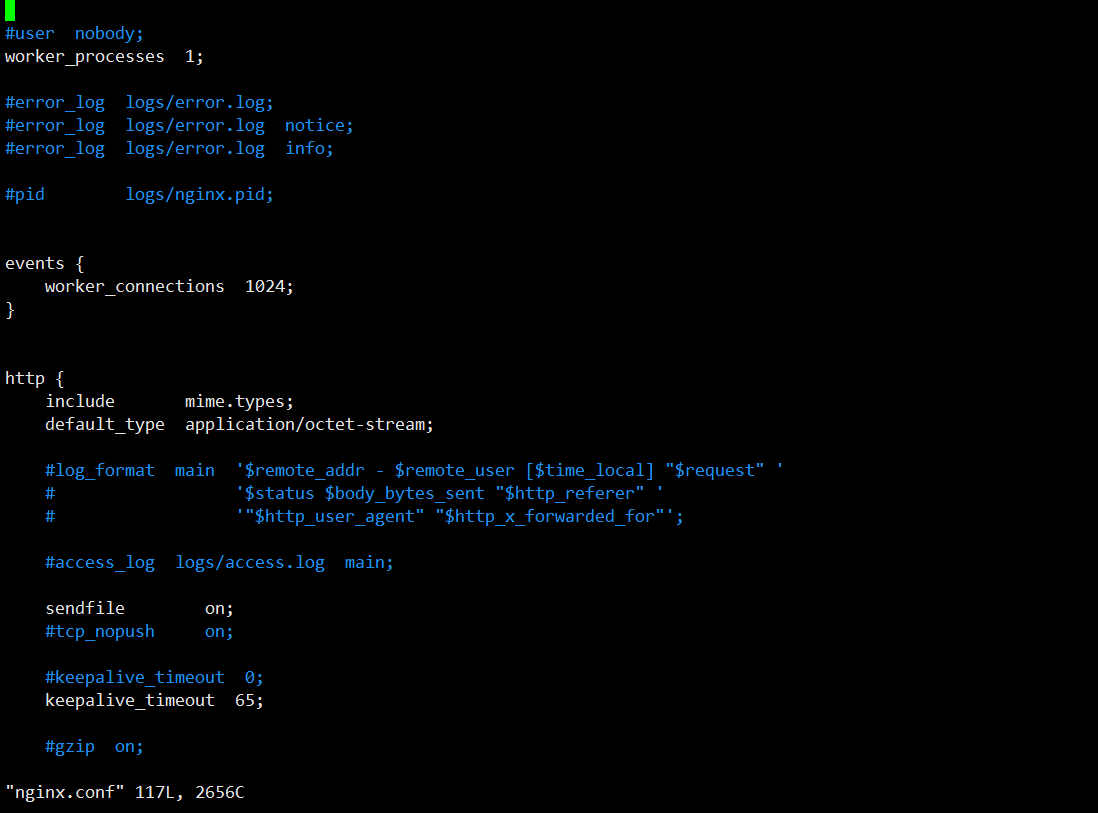
查看nginx进程会发现有两个进程,一个worker一个master



master(主)进程一般只有一个,worker(工作)进程一般根据服务器cpu核心数来确定,cpu是几核的一般配置几个worker,具体配置在nginx.conf中如下



work\_processes:worker进程的个数(一般与cpu核心数相同)



### 为什么这样设计进程模型

因为nginx是热部署方式支持在线更新配置文件等,当更新配置文件时master进程会监测worker进程工作状态,如果worker与客户端在通信则等待通信完毕后在更新配置文件,否则直接更新worker的配置文件,这样不影响程序的正常使用

每个worker进程是相对独立的,不需要加锁,省去了锁的竞争开销

由于是独立的worker,一个worker结束与否与其他worker没有关系,保证了各个worker之间的安全性,当一个worker异常退出时不影响其他worker继续工作

## 事件处理机制

## 通过域名来区分虚拟机

|  |
| --- |
| server {  listen 80;  server\_name www.rkg.com;  location / {  root rkg;  index index.html index.htm;  }  error\_page 500 502 503 504 /50x.html;  location = /50x.html {  root html;  }  }      server {  listen 80;  server\_name www.rkg1.com;  location / {  root rkg1;  index index.html index.htm;  }  error\_page 500 502 503 504 /50x.html;  location = /50x.html {  root html;  }  }      server {  listen 80;  server\_name www.rkg2.com;  location / {  root rkg2;  index index.html index.htm;  }  error\_page 500 502 503 504 /50x.html;  location = /50x.html {  root html;  }  }        server {  listen 80;  server\_name www.rkg3.com;  location / {  root rkg3;  index index.html index.htm;  }  error\_page 500 502 503 504 /50x.html;  location = /50x.html {  root html;  }  } |

# 为安装好的nginx添加https模块支持

## 切换到源码包：

|  |  |
| --- | --- |
| 1 | cd /home/nginxBag/nginx-1.10.2/ |

## 查看nginx原有的模块

|  |  |
| --- | --- |
| 1 | /usr/local/nginx/sbin/nginx -V |

## 在configure arguments:后面显示的原有的configure参数如下：

|  |  |
| --- | --- |
| 1 | --prefix=/usr/local/nginx --with-http\_stub\_status\_module |

## 那么我们的新配置信息就应该这样写：

|  |  |
| --- | --- |
| 1 | ./configure --prefix=/usr/local/nginx --with-http\_stub\_status\_module --with-http\_ssl\_module |

运行上面的命令即可，等配置完

## 配置完成后，运行命令

|  |  |
| --- | --- |
| 1 | make |

这里不要进行make install，否则就是覆盖安装

## 然后备份原有已安装好的nginx

|  |  |
| --- | --- |
| 1 | cp /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.bak |

## 然后将刚刚编译好的nginx覆盖掉原有的nginx（这个时候nginx要停止状态）

|  |  |
| --- | --- |
| 1 | cp ./objs/nginx /usr/local/nginx/sbin/ |

## 然后启动nginx，仍可以通过命令查看是否已经加入成功

|  |  |
| --- | --- |
| 1 | /usr/local/nginx/sbin/nginx -V |

[root@192 nginx-1.10.2]# /usr/local/nginx/sbin/nginx -V

nginx version: nginx/1.10.2

built by gcc 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC)

built with OpenSSL 1.0.2k-fips 26 Jan 2017

TLS SNI support enabled

configure arguments: --prefix=/usr/local/nginx --with-http\_stub\_status\_module --with-http\_ssl\_module

[root@192 nginx-1.10.2]#

# 生成数字证书

最后把client.p12发给用户即可

# 配置nginx.conf



# nginx高可用