Nginx

Nginx源代码是使⽤C语⾔开发的，所以不会再去追踪分析它的源代码。

# 第⼀部分 Nginx基础回顾



## 1，Nginx 到底是什么？

Nginx 是⼀个⾼性能的HTTP和反向代理web服务器，核⼼特点是占有内存少，并发能⼒强

## 2，Nginx 应用场景

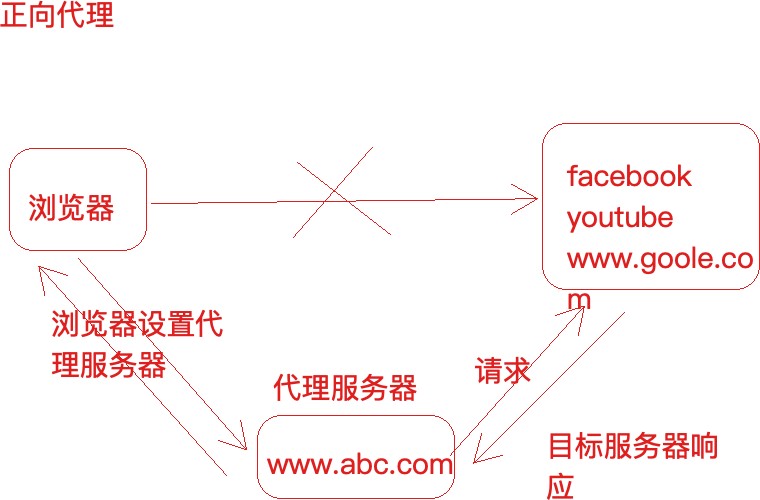
Http服务器（Web服务器）

性能⾮常⾼，⾮常注重效率，能够经受⾼负载的考验。

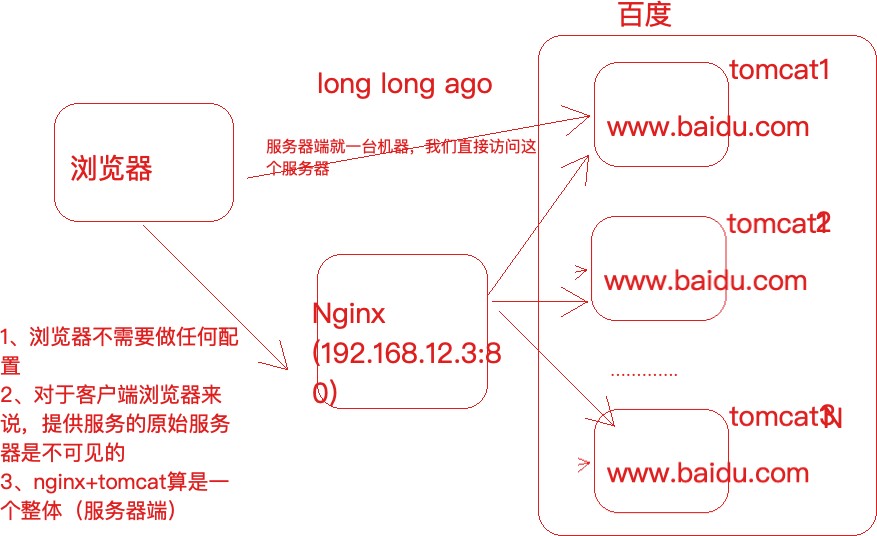
⽀持50000个并发连接数，不仅如此，CPU和内存的占⽤也⾮常的低，10000个没有活动的连 接才占⽤2.5M的内存。

反向代理服务器

正向代理：在浏览器中配置代理服务器的相关信息，通过代理服务器访问⽬标⽹站，代理服务器收到⽬标⽹站的响应之后，会把响应信息返回给我们⾃⼰的浏览器客户端



反向代理：浏览器客户端发送请求到反向代理服务器（⽐如Nginx），由反向代理服务器选择原始服务器提供服务获取结果响应，最终再返回给客户端浏览器

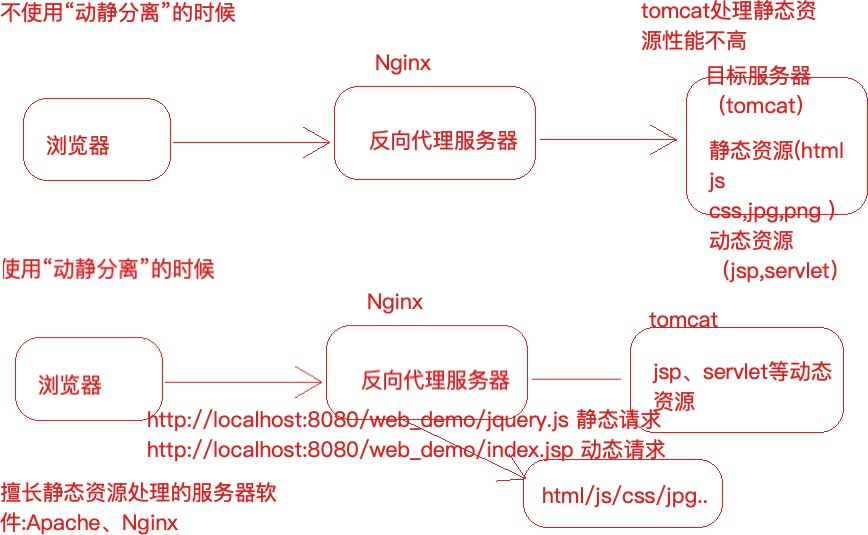


负载均衡服务器

负载均衡，当⼀个请求到来的时候（结合上图），Nginx反向代理服务器根据请求去找到⼀个 原始服务器来处理当前请求，那么这叫做反向代理。那么，如果⽬标服务器有多台（⽐如上图中的tomcat1，tomcat2，tomcat3...），找哪⼀个⽬标服务器来处理当前请求呢，这样⼀ 个寻找确定的过程就叫做负载均衡。

⽣活中也有很多这样的例⼦，⽐如，我们去银⾏，可以处理业务的窗⼝有多个，那么我们会 被分配到哪个窗⼝呢到底，这样的⼀个过程就叫做负载均衡。

负载均衡就是为了解决⾼负载的问题。动静分离



## 3，Nginx 的特点

跨平台：Nginx可以在⼤多数类unix操作系统上编译运⾏，⽽且也有windows版本

Nginx的上⼿⾮常容易，配置也⽐较简单

⾼并发，性能好

稳定性也特别好，宕机概率很低

## 4，Nginx的安装

上传nginx安装包到linux服务器，nginx安装包(.tar⽂件)下载地址：[http://nginx.org](http://nginx.org/) 课程使⽤1.17.8版本 

安装Nginx依赖，pcre、openssl、gcc、zlib（推荐使⽤yum源⾃动安装）

yum -y install gcc zlib zlib-devel pcre-devel openssl openssl-devel

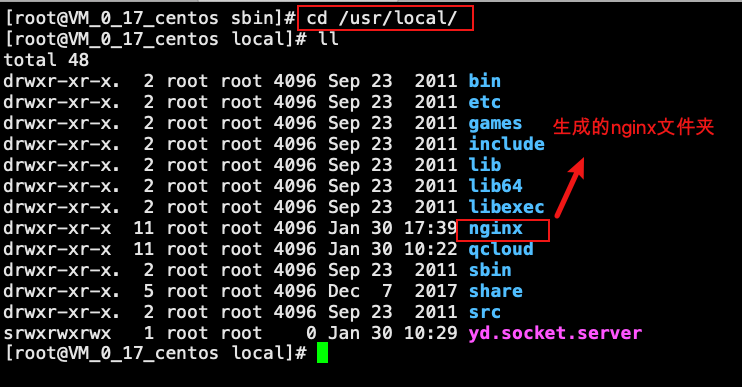
解包Nginx软件包

tar -xvf nginx-1.17.8.tar

进⼊解压之后的⽬录 nginx-1.17.8 cd nginx-1.17.8

命令⾏执⾏./conﬁgure 命令⾏执⾏ make

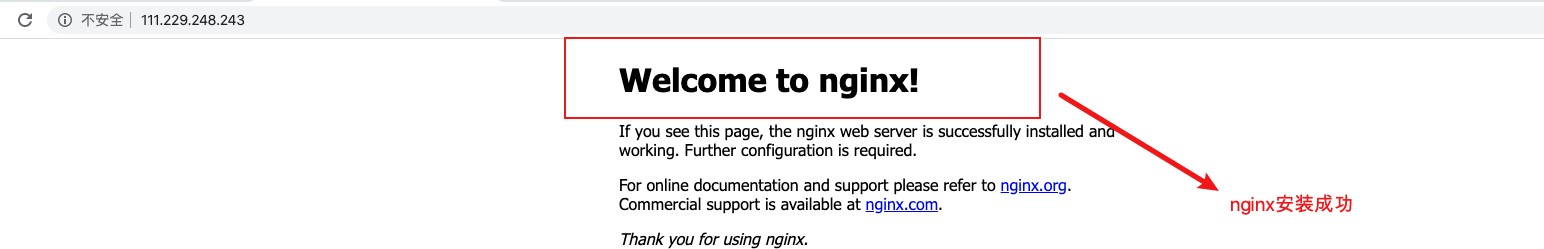
命令⾏执⾏ make install，完毕之后在/usr/local/下会产⽣⼀个nginx⽬录



进⼊sbin⽬录中，执⾏启动nginx命令

cd nginx/sbin

./nginx

然后访问服务器的80端⼝（nginx默认监听80端⼝）

## 5，Nginx主要命令

./nginx 启动nginx

./nginx -s stop 终⽌nginx（当然也可以找到nginx进程号，然后使⽤kill -9 杀掉nginx进程）

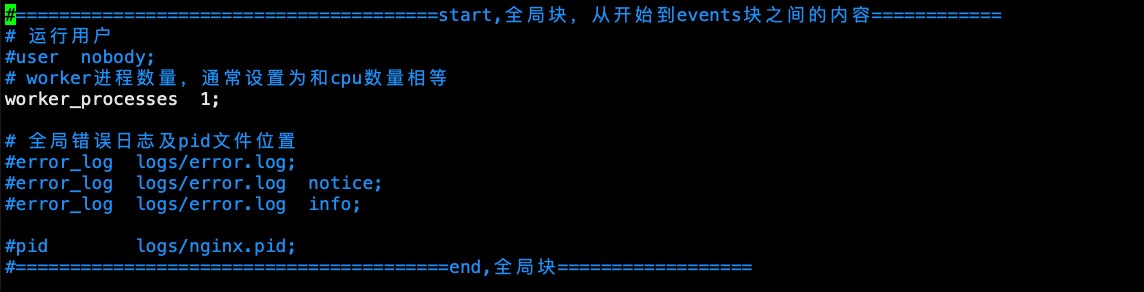
./nginx -s reload (重新加载nginx.conf配置⽂件)

# 第⼆部分 Nginx核⼼配置⽂件nginx.conf解读

Nginx的核⼼配置⽂件conf/nginx.conf包含三块内容：全局块、events块、http块

## 1，全局块

影响nginx服务器整体的运⾏，⽐如worker进程的数量、错误⽇志的位置等

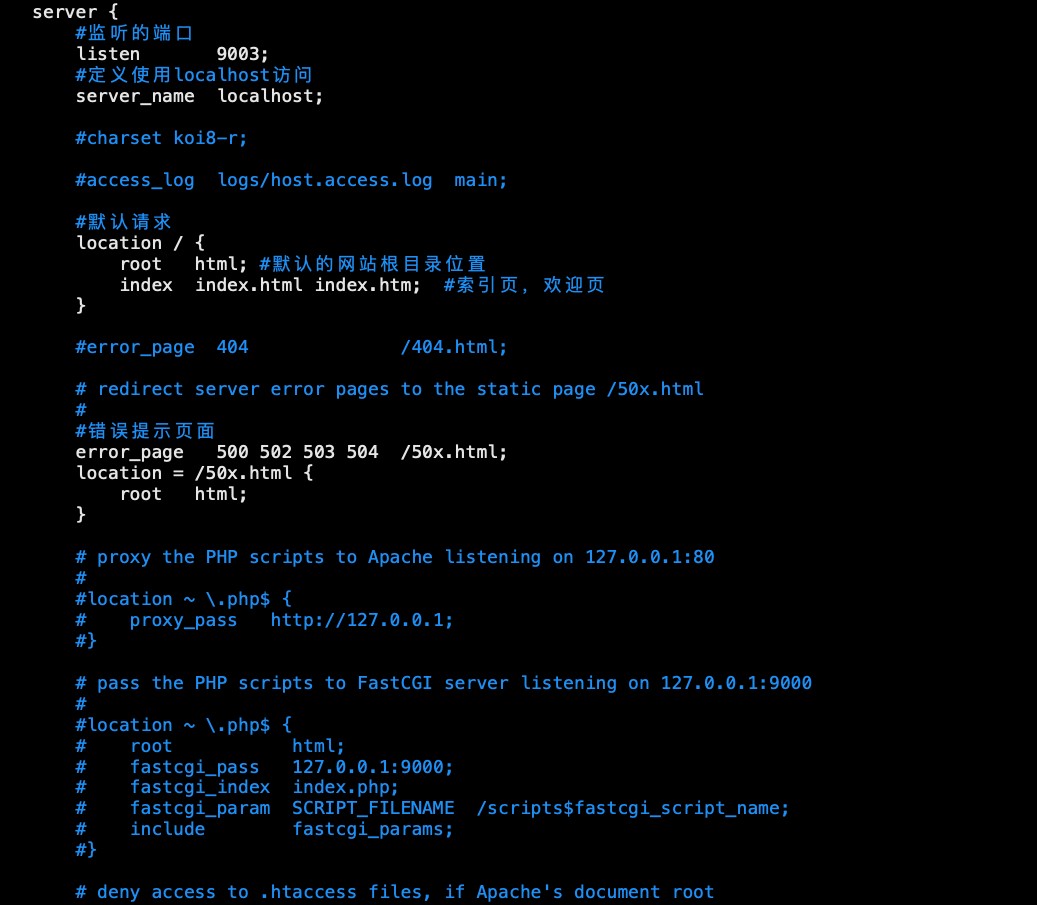


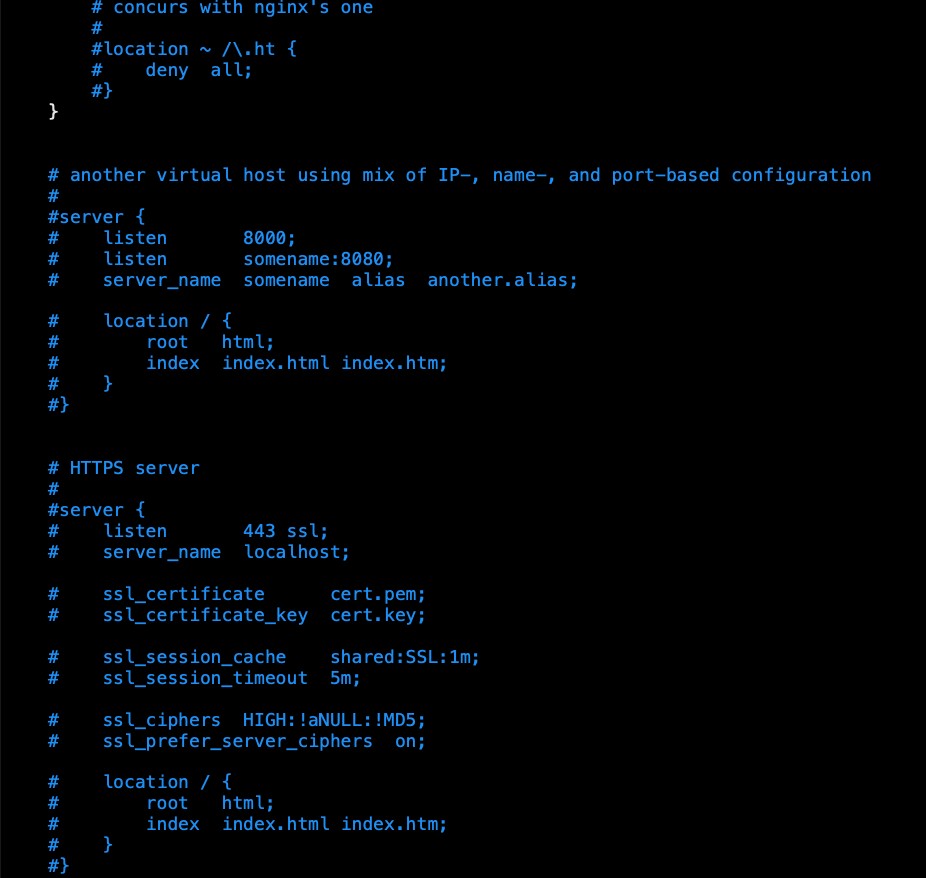
## 2，events块

影响nginx服务器与⽤户的⽹络连接，⽐如worker\_connections 1024，标识每个workderprocess⽀持的最⼤连接数为1024

## 3，http块

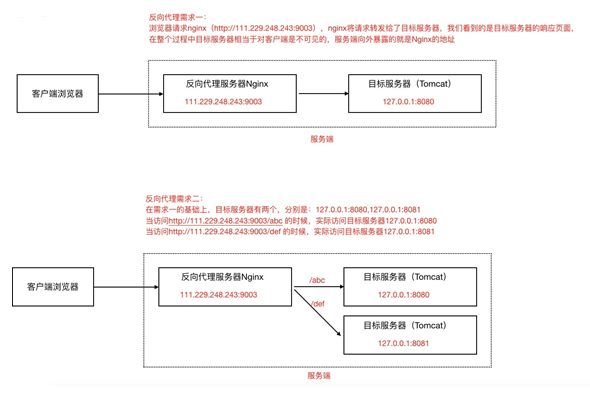
此块是配置最频繁的部分，虚拟主机的配置，监听端⼝的配置，请求转发、反向代理、负载均衡 等





# 第三部分 Nginx - 反向代理

需求：



**需求⼀完成**

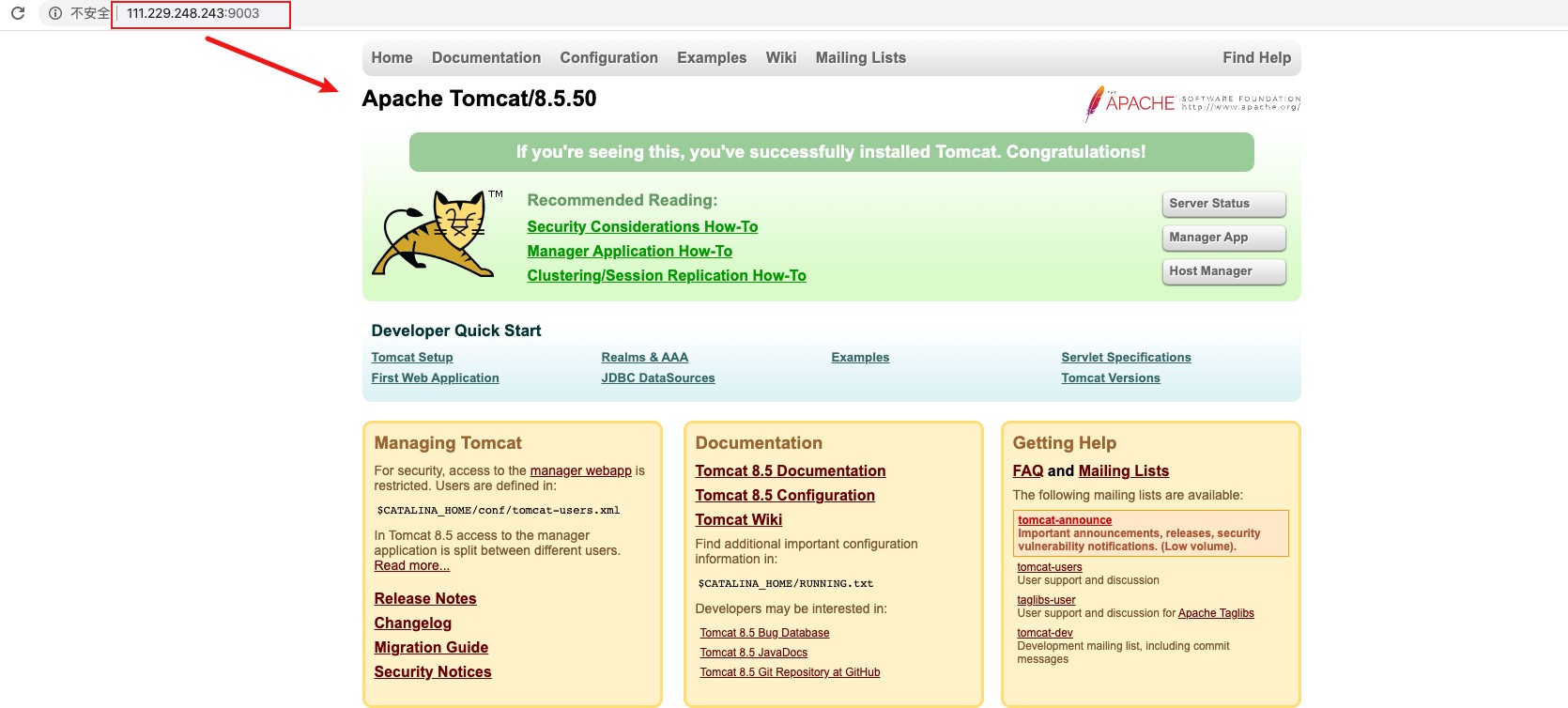
部署tomcat，保持默认监听8080端⼝修改nginx配置，并重新加载

修改nginx配置

重新加载nginx配置

./nginx -s reload

测试，访问http://111.229.248.243:9003,返回tomcat的⻚⾯



需求⼆完成

再部署⼀台tomcat，保持默认监听8081端⼝修改nginx配置，并重新加载



这⾥主要就是多location的使⽤，这⾥的nginx中server/location就好⽐tomcat中的

**Host/Context**

location 语法如下：

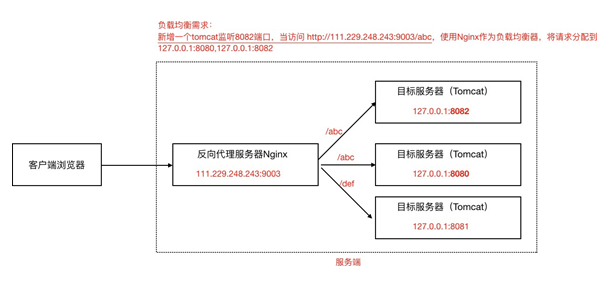
location [=|~|~\*|^~] /uri/ { … }

在nginx配置⽂件中，location主要有这⼏种形式： 1） 正 则 匹 配 location ~ /lagou { } 2）不区分⼤⼩写的正则匹配 location ~\* /lagou { } 3） 匹 配 路 径 的 前 缀 location ^~ /lagou { } 4） 精 确 匹 配 location = /lagou { } 5）普通路径前缀匹配 location /lagou { }

优先级

4 > 3 > 2 > 1 > 5

# 第四部分 Nginx - 负载均衡



Nginx负载均衡策略

轮询

默认策略，每个请求按时间顺序逐⼀分配到不同的服务器，如果某⼀个服务器下线，能⾃动剔除

upstream lagouServer{

server 111.229.248.243:8080;

server 111.229.248.243:8082;

}

location /abc {

proxy\_pass [http://lagouServer/;](http://lagouServer/%3B)

}

weight

weight代表权重，默认每⼀个负载的服务器都为1，权重越⾼那么被分配的请求越多（⽤于服务器 性能不均衡的场景）

upstream lagouServer{

server 111.229.248.243:8080 weight=1;

server 111.229.248.243:8082 weight=2;

}

ip\_hash

每个请求按照ip的hash结果分配，每⼀个客户端的请求会固定分配到同⼀个⽬标服务器处理，可 以解决session问题

upstream lagouServer{ ip\_hash;

server 111.229.248.243:8080;

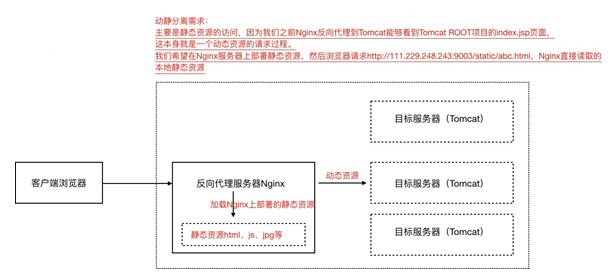
server 111.229.248.243:8082;

}

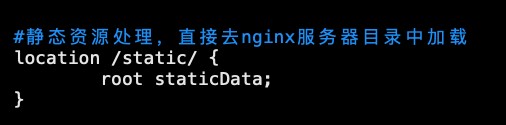
# 第五部分 Nginx - 动静分离

动静分离就是将动态资源和静态资源的请求处理分配到不同的服务器上，⽐较经典的组合就是Nginx+Tomcat架构（Nginx处理静态资源请求，Tomcat处理动态资源请求），那么其实之前的讲解 中，Nginx反向代理⽬标服务器Tomcat，我们能看到⽬标服务器ROOT项⽬的index.jsp，这本身就是Tomcat在处理动态资源请求了。

所以，我们只需要配置静态资源访问即可。

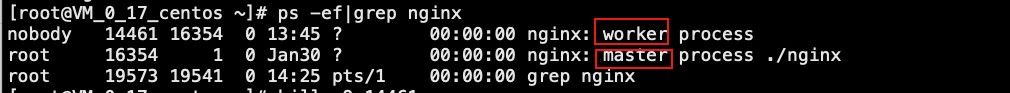


把静态资源费放到Nginx下面的文件夹staticData，然后配置请求过去



# 第六部分 Nginx底层进程机制剖析

Nginx启动后，以**daemon多进程⽅式**在后台运⾏，包括⼀个Master进程和多个Worker进程，Master

进程是领导，是⽼⼤，Worker进程是⼲活的⼩弟。

1，master进程：主要是管理worker进程，⽐如：

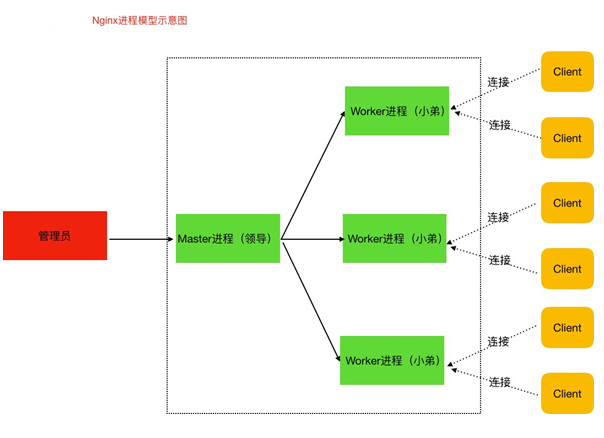
接收外界信号向各worker进程发送信号(./nginx -s reload)

监控worker进程的运⾏状态，当worker进程异常退出后Master进程会⾃动重新启动新的worker进程

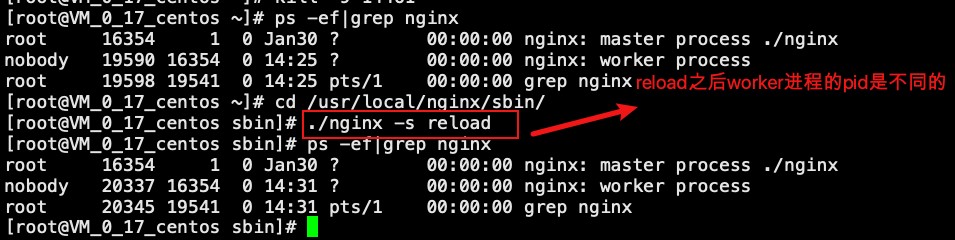
2，worker进程

worker进程具体处理⽹络请求。多个worker进程之间是对等的，他们同等竞争来⾃客户端的请求，**各进程互相之间是独⽴的**。⼀个请求只可能在⼀个worker进程中处理，⼀个worker进程，不可能处理其它进程的请求。worker进程的个数是可以设置的，⼀般设置与机器cpu核数⼀致。

Nginx进程模型示意图如下



以 ./nginx -s reload 来说明nginx信号处理这部分

1. master进程对配置⽂件进⾏语法检查
2. 尝试配置（⽐如修改了监听端⼝，那就尝试分配新的监听端⼝）
3. 尝试成功则使⽤新的配置，新建worker进程
4. 新建成功，给旧的worker进程发送关闭消息
5. 旧的worker进程收到信号会继续服务，直到把当前进程接收到的请求处理完毕后关闭 所以reload之后worker进程pid是发⽣了变化的

worker进程处理请求部分的说明

例如，我们监听9003端⼝，⼀个请求到来时，如果有多个worker进程，那么每个worker进程都有 可能处理这个链接。

master进程创建之后，会建⽴好需要监听的的socket，然后从master进程再fork出多个

worker进程。所以，所有worker进程的监听描述符listenfd在新连接到来时都变得可读。nginx使⽤互斥锁来保证只有⼀个workder进程能够处理请求，拿到互斥锁的那个进程注册listenfd读事件，在读事件⾥调⽤accept接受该连接，然后解析、处理、返回客户端

nginx多进程模型好处

每个worker进程都是独⽴的，不需要加锁，节省开销

每个worker进程都是独⽴的，互不影响，⼀个异常结束，其他的照样能提供服务 多进程模型为reload热部署机制提供了⽀撑