Nginx课程笔记（讲师：应癫）

主要课程内容

- 第一部分：Nginx基础回顾(Nginx是什么？能做什么事情（应用在什么场合）？常用命令是什么？)
- 第二部分：Nginx核心配置文件解读
- 第三部分：Nginx应用场景之反向代理
- 第四部分：Nginx应用场景之负载均衡
- 第五部分：Nginx应用场景之动静分离
- 第六部分：Nginx底层进程机制剖析

  Nginx源代码是使用C语言开发的，所以呢我们不会再去追踪分析它的源代码了。

第一部分 Nginx基础回顾

nginx 编辑

Nginx (engine x) 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务。Nginx是由伊戈尔·赛索耶夫为俄罗斯访问量第二的Rambler.ru站点（俄文：Рамблер）开发的，第一个公开版本0.1.0发布于2004年10月4日。

其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx 1.0.4发布。

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

中文名	Nginx	软件类型	开源软件,网页服务器软件
外文名	Nginx	兼容性	Linux系统,Windows NT系统
软件许可	BSD许可	别　名	engine x

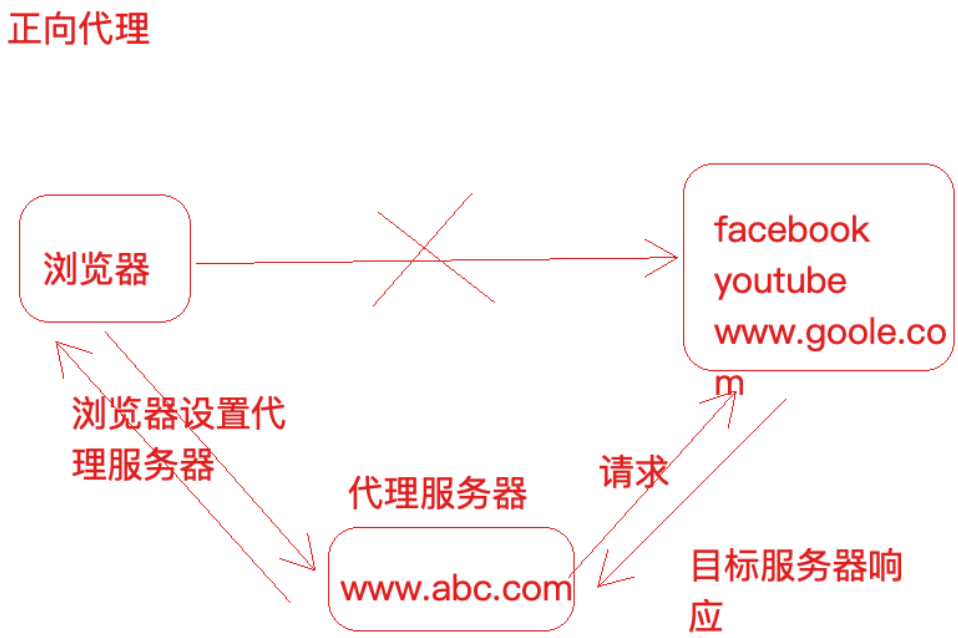
- Nginx 到底是什么?

  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
- Nginx 又能做什么事情（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。

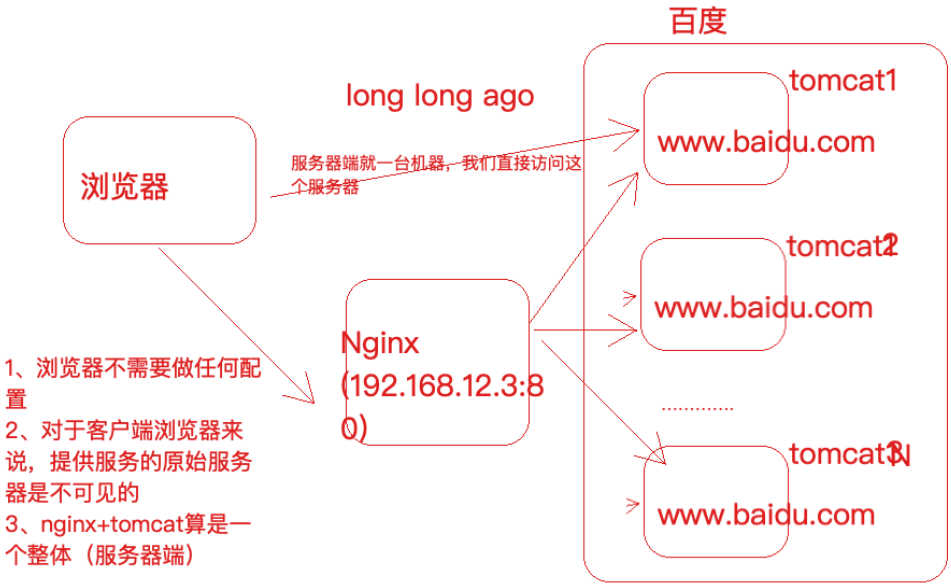
    支持50000个并发连接数，不仅如此，CPU和内存的占用也非常的低，10000个没有活动的连接才占用2.5M的内存。
  - 反向代理服务器
    - 正向代理

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



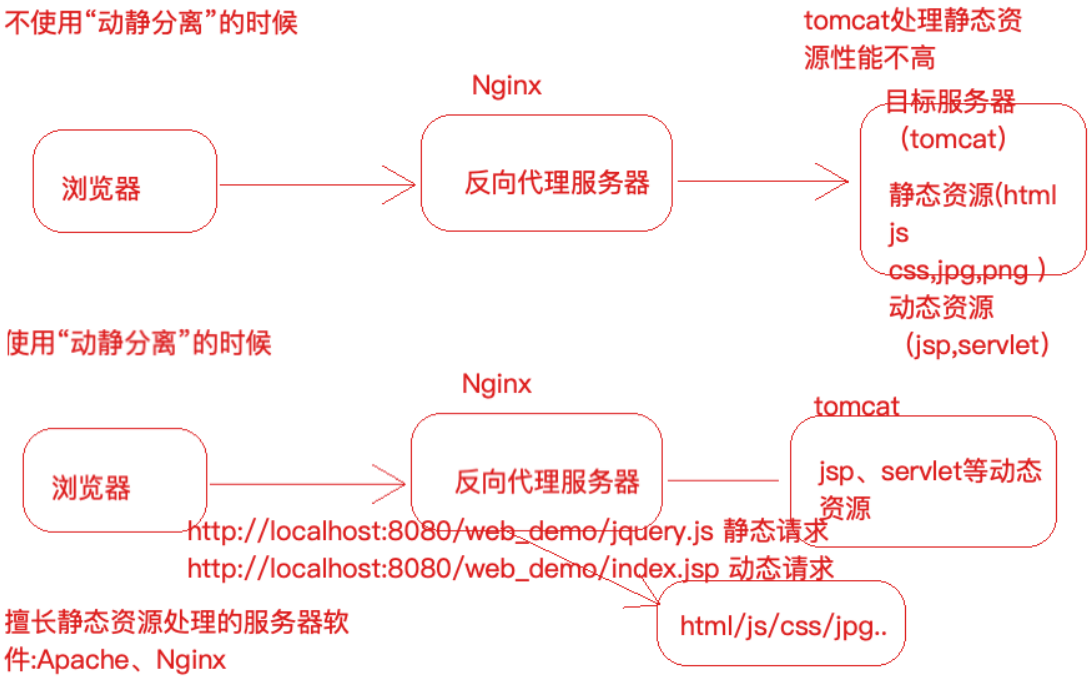
- 负载均衡服务器

  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



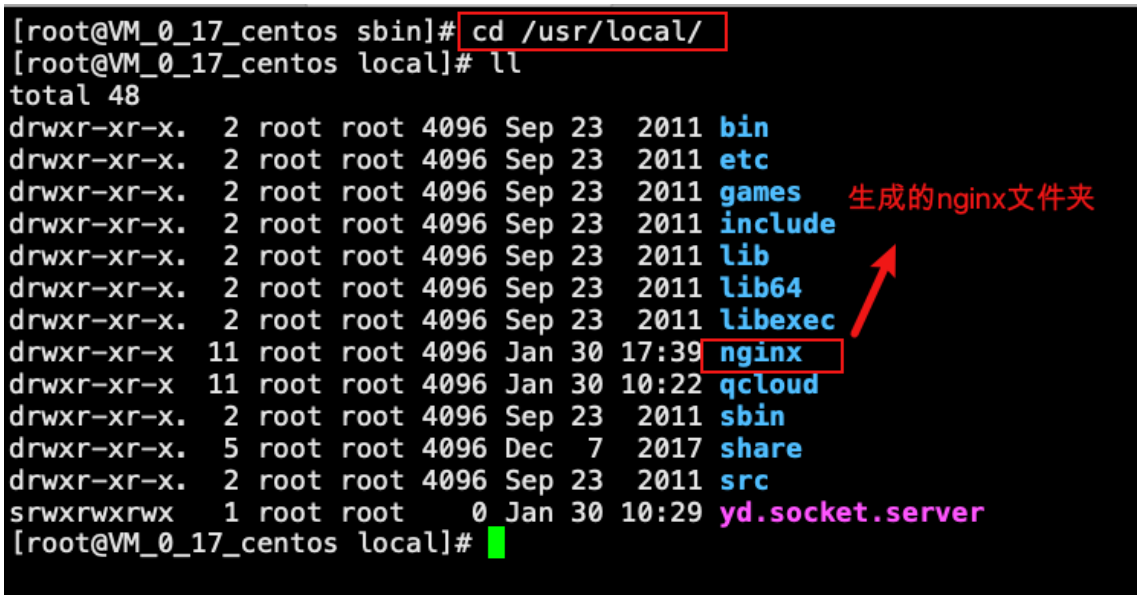
- Nginx 的特点
  - 跨平台：Nginx可以在大多数类unix操作系统上编译运行，而且也有windows版本
  - Nginx的上手非常容易，配置也比较简单
  - 高并发，性能好
  - 稳定性也特别好，宕机概率很低
- Nginx的安装
  - 上传nginx安装包到linux服务器，nginx安装包(.tar文件)下载地址：http://nginx.org

    课程使用1.17.8版本 nginx-1.17.8.tar
  - 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）

- 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

- 全局块

  从配置文件开始到events块之间的内容，此处的配置影响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

```
#==========================================start,全局块，从开始到events块之间的内容============
# 运行用户
#user  nobody;
# worker进程数量，通常设置为和cpu数量相等
worker_processes  1;

# 全局错误日志及pid文件位置
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;
#==========================================end,全局块=================
```

- events块

  events块主要影响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

```
#==========================================start,events事件块（影响nginx服务器和用户的网络连接）=======
events {
        #单个worker进程的最大并发连接数
    worker_connections  1024;
}
#==========================================end,events时间块============
```

- http块

  http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

```
#======================start,http块（nginx服务器中配置最频繁的部分，端口监听，请求转发等）====================
http {
    # 引入mime类型定义文件
    include       mime.types;
    default_type  application/octet-stream;
    # 设定日志格式
    #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;

        # 连接超时时间
    #keepalive_timeout  0;
    keepalive_timeout  65;

        # 开启gzip压缩
    #gzip  on;
```

```
server {
    #监听的端口
    listen       9003;
    #定义使用localhost访问
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    #默认请求
    location / {
        root   html; #默认的网站根目录位置
        index  index.html index.htm;  #索引页，欢迎页
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    #错误提示页面
    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;
    #    }
    #}
```

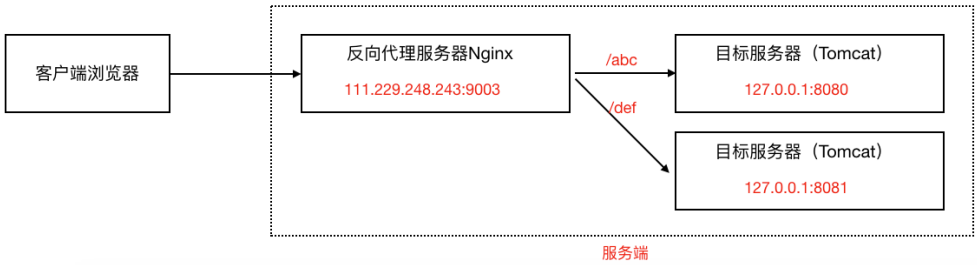
第三部分 Nginx应用场景之反向代理

需求：

by 应癫

反向代理需求一：
浏览器请求nginx（http://111.229.248.243:9003），nginx将请求转发给了目标服务器，我们看到的是目标服务器的响应页面，
在整个过程中目标服务器相当于对客户端是不可见的，服务端向外暴露的就是Nginx的地址

反向代理需求二：
在需求一的基础上，目标服务器有两个，分别是：127.0.0.1:8080,127.0.0.1:8081
当访问http://111.229.248.243:9003/abc 的时候，实际访问目标服务器127.0.0.1:8080
当访问http://111.229.248.243:9003/def 的时候，实际访问目标服务器127.0.0.1:8081



需求一完成

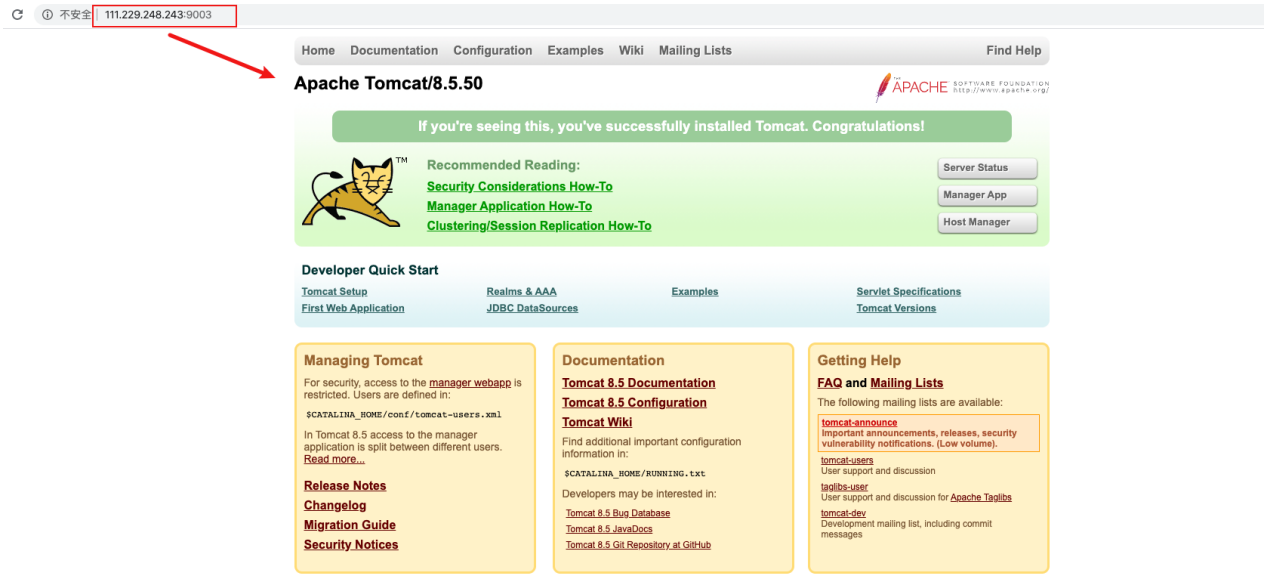
- 部署tomcat，保持默认监听8080端口
- 修改nginx配置，并重新加载
  - 修改nginx配置

```
server {
    #监听的端口
    listen       9003;
    #定义使用localhost访问
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;
                                        反向代理，请求转发到目标服务器
    #默认请求
    location / {
        proxy_pass http://127.0.0.1:8080;
        # root   html; #默认的网站根目录位置
        # index  index.html index.htm;  #索引页，欢迎页
    }
```

  - 重新加载nginx配置

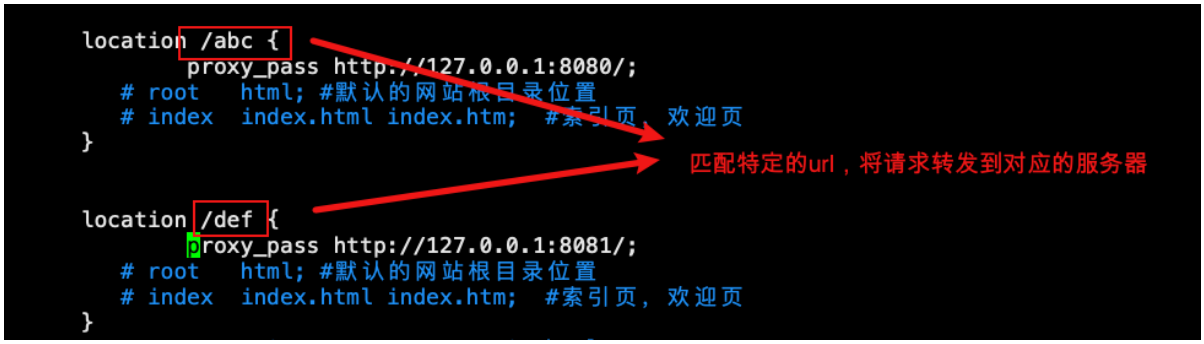
    ./nginx -s reload

- 测试，访问http://111.229.248.243:9003,返回tomcat的页面



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 { }

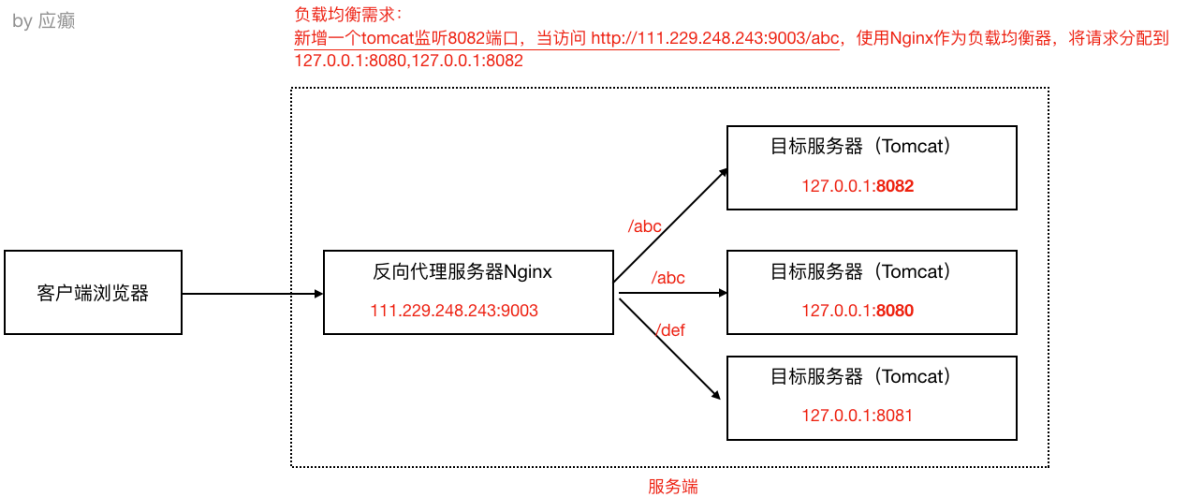
5）普通路径前缀匹配 location /lagou { }

- 优先级

  4 > 3 > 2 > 1 > 5

第四部分 Nginx应用场景之负载均衡

by 应癫



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/;
}
```

- 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

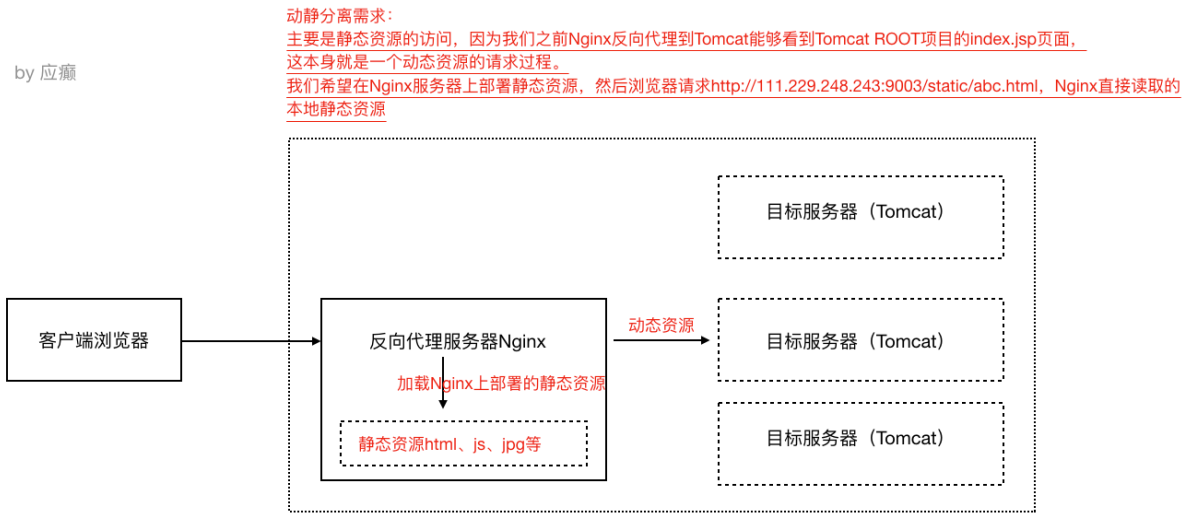
  每个请求按照ip的hash结果分配，每一个客户端的请求会固定分配到同一个目标服务器处理，可以解决session问题

```
upstream lagouServer{
  ip_hash;
  server 111.229.248.243:8080;
  server 111.229.248.243:8082;
}
```

第五部分 Nginx应用场景之动静分离

动静分离就是讲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。



Nginx配置

```
#静态资源处理，直接去nginx服务器目录中加载
location /static/ {
        root staticData;
}
```

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。

```
[root@VM_0_17_centos ~]# ps -ef|grep nginx
nobody   14461 16354  0 13:45 ?        00:00:00 nginx: worker process
root     16354     1  0 Jan30 ?        00:00:00 nginx: master process ./nginx
root     19573 19541  0 14:25 pts/1    00:00:00 grep nginx
```

- master进程

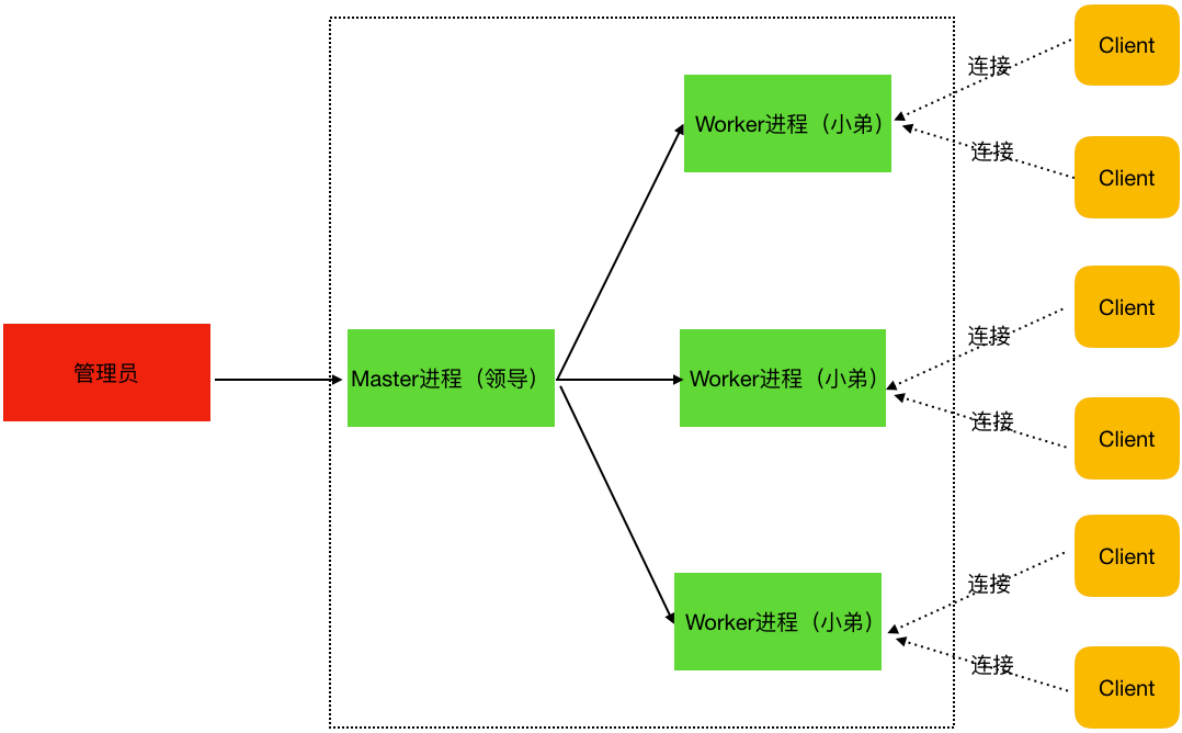
  主要是管理worker进程，比如：

  - 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进程等
- 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

by 应癫　　Nginx进程模型示意图



- 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

```
[root@VM_0_17_centos ~]# ps -ef|grep nginx
root     16354     1  0 Jan30 ?        00:00:00 nginx: master process ./nginx
nobody   19590 16354  0 14:25 ?        00:00:00 nginx: worker process
root     19598 19541  0 14:25 pts/1    00:00:00 grep nginx reload之后worker进程的pid是不同的
[root@VM_0_17_centos ~]# cd /usr/local/nginx/sbin/
[root@VM_0_17_centos sbin]# ./nginx -s reload
[root@VM_0_17_centos sbin]# ps -ef|grep nginx
root     16354     1  0 Jan30 ?        00:00:00 nginx: master process ./nginx
nobody   20337 16354  0 14:31 ?        00:00:00 nginx: worker process
root     20345 19541  0 14:31 pts/1    00:00:00 grep nginx
[root@VM_0_17_centos sbin]#
```

- 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