一、 Nginx简介

1.1 讲在 Nginx之前

1.1.1 同步与异步

同步与异步的重点在消息通知的方式上，也就是调用结果的通知方式不同。

同步：当一个同步调用发出去后，调用者要一直等待调用的结果通知后，才能进行后续的执行。

异步：当一个异步调用发出去后，调用者不必一直等待调用结果的返回，异步调用，要想获得结果，

一般有两种方式：

1、主动轮询异步调用的结果;

2、被调用方通过 callback（回调通知）来通知调用方调用结果。

实例解释：

同步取快递：小明收到快递将送达的短信，在楼下一直等到快递送达。

异步取快递：小明收到快递将送达的短信，快递到楼下后，小明再下楼去取。

异步取快递，小明知道快递到达楼下有两种方式：

1、不停的电话问快递小哥到了没有，即主动轮询；

2、快递小哥到楼下后，打电话通知小明，然后小明下楼取快递，即回调通知。

1.1.2 阻塞与非阻塞

阻塞与非阻塞的重点在于进/线程等待消息时候的行为，也就是在等待消息的时候，当前进/线程是挂起状态，还是非挂起状态。

阻塞：调用在发出去后，在消息返回之前，当前进/线程会被挂起，直到有消息返回，当前进/线程才会被激活

非阻塞：调用在发出去后，不会阻塞当前进/线程，而会立即返回。

实例解释：

阻塞取快递：小明收到快递即将送达的信息后，什么事都不做，一直专门等快递。

非阻塞取快递：小明收到快递即将送达的信息后，等快递的时候，还一边敲代码、一边刷微信。

同步与异步，重点在于消息通知的方式；阻塞与非阻塞，重点在于等消息时候的行为。

所以，就有了下面 4种组合方式

- 同步阻塞：小明收到信息后，啥都不干，等快递；
- 同步非阻塞：小明收到信息后，边刷微博，边等着取快递；
- 异步阻塞：小明收到信息后，啥都不干，一直等着快递员通知他取快递；
- 异步非阻塞：小明收到信息后，边刷着微博，边等快递员通知他取快递。

大部分程序的 I/O模型都是同步阻塞的，单个进程每次只在一个文件描述符上执行 I/O操作，每次 I/O系统调用都会阻塞，直到完成数据传输。传统的服务器采用的就是同步阻塞的多进程模型。一个 server采用一个进程负责一个 request 的方式，一个进程负责一个 request，直到会话结束。进程数就是并发数，而操作系统支持的进程数是有限的，且进程数越多，调度的开销也越大，因此无法面对高并发。

Nginx采用了异步非阻塞的方式工作。我们先来先了解一下 I/O多路复用中的 epoll模型。

1.1.3 epoll模型

当连接有 I/O事件产生的时候，epoll就会去告诉进程哪个连接有 I/O事件产生，然后进程就去处理这个事件。

例如：小明家楼下有一个收发室，每次有快递到了，门卫就先代收并做了标记；然后通知小明去取送给小明的快递。

1.2 Nginx概述

Nginx ("engine x") 是一个**高性能的 HTTP 和反向代理服务器，特点是占有内存少，并发能**力强**，事实上nginx的并发能力确实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。

1.3 Nginx作为web服务器

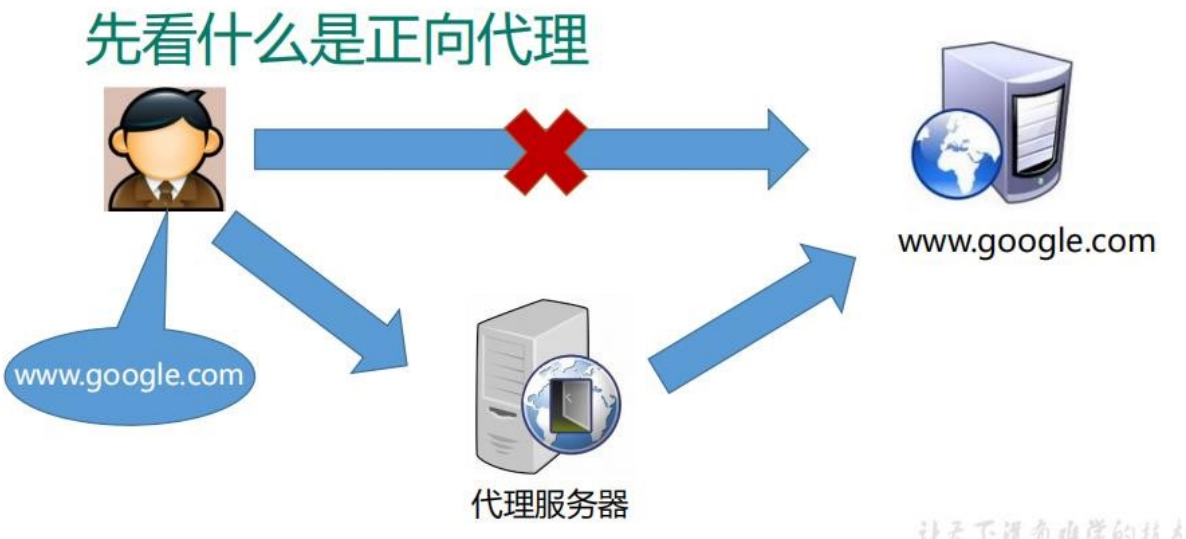
Nginx 可以作为静态页面的 web 服务器，同时还支持 CGI 协议的动态语言，比如 perl、php等。但是不支持 java。Java 程序只能通过与 tomcat 配合完成。Nginx 专为性能优化而开发，性能是其最重要的考量,实现上非常注重效率 ，能经受高负载的考验,有报告表明**能支持高**达 50,000 个并发连接数**。

https://lnmp.org/nginx.html

1.4 正向代理

Nginx 不仅可以做反向代理，实现负载均衡。还能用作正向代理来进行上网等功能。

正向代理：如果把局域网外的 Internet 想象成一个巨大的资源库，则局域网中的客户端要访问 Internet，则需要通过代理服务器来访问，这种代理服务就称为正向代理。



1.5 反向代理

反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，**暴露的是代理服务器**地址，隐藏了真实服务器 IP 地址**。

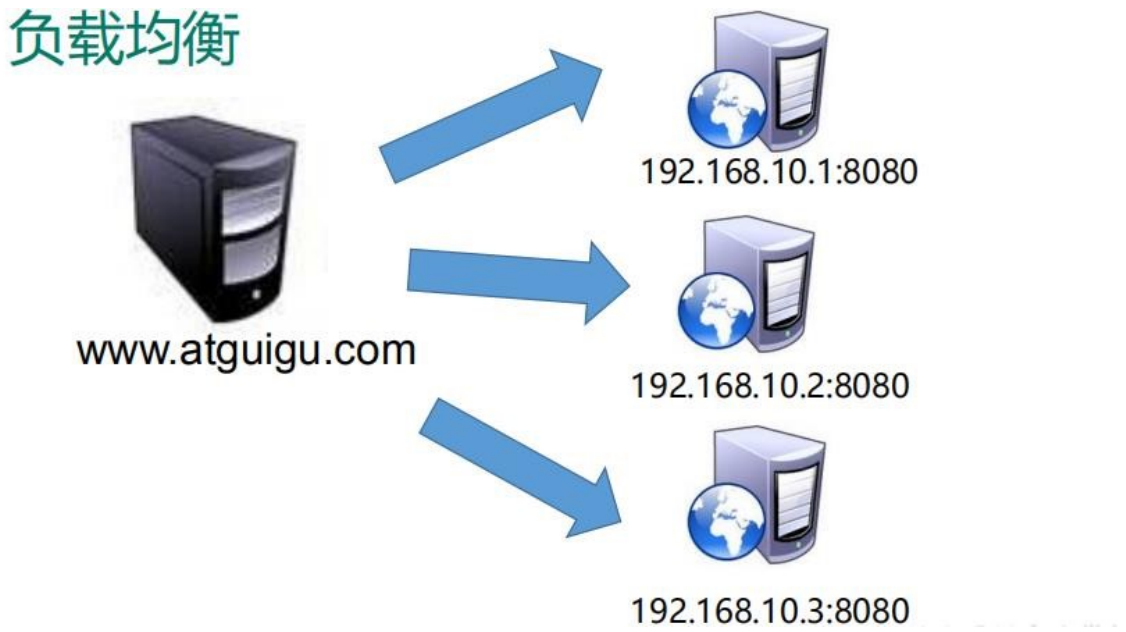
1.6 负载均衡

客户端发送多个请求到服务器，服务器处理请求，有一些可能要与数据库进行交互，服务器处理完毕后，再将结果返回给客户端。

这种架构模式对于早期的系统相对单一，并发请求相对较少的情况下是比较适合的，成本也低。但是随着信息数量的不断增长，访问量和数据量的飞速增长，以及系统业务的复杂度增加，这种架构会造成服务器相应客户端的请求日益缓慢，并发量特别大的时候，还容易造成服务器直接崩溃。很明显这是由于服务器性能的瓶颈造成的问题，那么如何解决这种情况呢？

我们首先想到的可能是升级服务器的配置，比如提高 CPU 执行频率，加大内存等提高机器的物理性能来解决此问题，但是我们知道摩尔定律的日益失效，硬件的性能提升已经不能满足日益提升的需求了。最明显的一个例子，天猫双十一当天，某个热销商品的瞬时访问量是极其庞大的，那么类似上面的系统架构，将机器都增加到现有的顶级物理配置，都是不能够满足需求的。那么怎么办呢？

上面的分析我们去掉了增加服务器物理配置来解决问题的办法，也就是说纵向解决问题的办法行不通了，那么横向增加服务器的数量呢？这时候集群的概念产生了，单个服务器解决不了，我们增加服务器的数量，然后将请求分发到各个服务器上，**将原先请求集中到单个**服务器上的情况改为将请求分发到多个服务器上，将负载分发到不同的服务器，也就是我们所说的 负载均衡**。



1.7 动静分离

为了加快网站的解析速度，可以**把动态页面和静态页面由不同的服务器来解析**，加快解析速度。降低原来单个服务器的压力。

二、Nginx安装

为知笔记地址：CentOS7 下安装nginx

GitHub地址：
https://github.com/wangliu1102/StudyNotes/tree/master/%E5%B0%9A%E7%A1%85%E8%B0%B7Java/%E5%9B%9B%E3%80%81JavaEE%E9%AB%98%E7%BA%A7/6%E3%80%81Nginx/%E5%AE%89%E8%A3%85

三、Nginx常用的命令和配置文件

3.1 nginx 常用的命令

3.1.1 查看版本号

```
# 进入 nginx 目录中
cd /usr/local/nginx/sbin
# 查看 nginx 版本号
./nginx -v
```

```
root@localhost sbin]# ./nginx -v
ginx version: nginx/1.12.2
```

3.1.2 启动命令

```
# 在/usr/local/nginx/sbin 目录下执行
./nginx
```

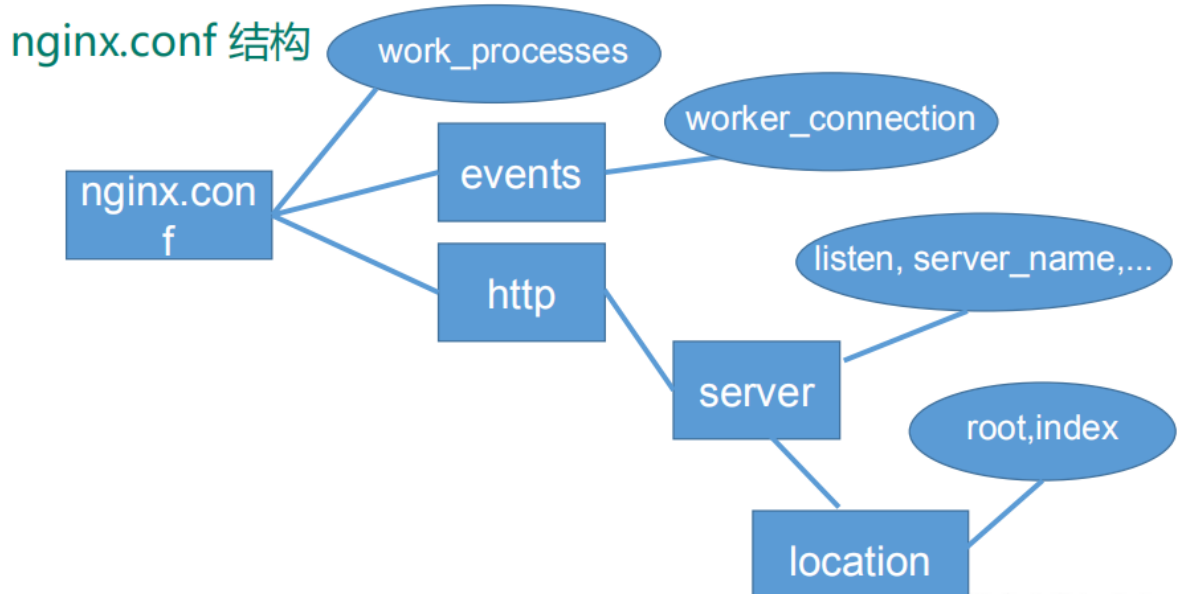
3.1.3 关闭命令

```
# 在/usr/local/nginx/sbin 目录下执行
./nginx -s stop
```

3.1.4 重新加载命令

```
# 在/usr/local/nginx/sbin 目录下执行
./nginx -s reload
```

3.2 nginx.conf 配置文件



nginx 安装目录下，其默认的配置文件都放在/usr/local/nginx/conf 目录下，而主配置文件nginx.conf 也在其中，后续对 nginx 的使用基本上都是对此配置文件进行相应的修改。

```
[root@slave1 /]# cd /usr/local/nginx/conf/
[root@slave1 conf]# ll
总用量 68
-rw-r--r--. 1 root root 1077 7月  29 21:48 fastcgi.conf
-rw-r--r--. 1 root root 1077 7月  29 21:48 fastcgi.conf.default
-rw-r--r--. 1 root root 1007 7月  29 21:48 fastcgi_params
-rw-r--r--. 1 root root 1007 7月  29 21:48 fastcgi_params.default
-rw-r--r--. 1 root root 2837 7月  29 21:48 koi-utf
-rw-r--r--. 1 root root 2223 7月  29 21:48 koi-win
-rw-r--r--. 1 root root 5170 7月  29 21:48 mime.types
-rw-r--r--. 1 root root 5170 7月  29 21:48 mime.types.default
-rw-r--r--. 1 root root 2656 7月  29 21:48 nginx.conf
-rw-r--r--. 1 root root 2656 7月  29 21:48 nginx.conf.default
-rw-r--r--. 1 root root  636 7月  29 21:48 scgi_params
-rw-r--r--. 1 root root  636 7月  29 21:48 scgi_params.default
-rw-r--r--. 1 root root  664 7月  29 21:48 uwsgi_params
-rw-r--r--. 1 root root  664 7月  29 21:48 uwsgi_params.default
-rw-r--r--. 1 root root 3610 7月  29 21:48 win-utf
[root@slave1 conf]#
```

配置文件中有很多#， 开头的表示注释内容，我们去掉所有以 # 开头的段落，精简之后的内容如下：

```
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

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

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
```

根据上述文件，我们可以很明显的将 nginx.conf 配置文件分为三部分：

3.2.1 第一部分：全局块

从配置文件开始到 events 块之间的内容，主要会设置一些影响 nginx 服务器整体运行的配置指令，主要包括配置运行 Nginx 服务器的用户（组）、允许生成的 worker process 数、进程 PID 存放路径、日志存放路径和类型以及配置文件的引入等。

```
# 程序运行用户和组
#user  nobody;

# 启动进程，指定 nginx 启动的工作进程数量，建议按照 cpu 数目来指定，一般等于 cpu 核心数目
# 这是 Nginx 服务器并发处理服务的关键配置，worker_processes 值越大，可以支持的并发处理量也越多，但是会受到硬件、软件等设备的制约。
worker_processes  1;

# 全局错误日志
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

# 主进程 PID 保存文件
#pid        logs/nginx.pid;

# 文件描述符数量
worker_rlimit_nofile 51200;
```

3.2.2 第二部分： events块

比如上面的配置：

```
events {
    # 使用 epoll 模型，对于 2.6 以上的内核，建议使用 epoll 模型以提高性能
    use epoll;
    # 工作进程的最大连接数量
    worker_connections  1024;
}
```

events 块涉及的指令主要影响 Nginx 服务器与用户的网络连接，常用的设置包括是否开启对多 work process下的网络连接进行序列化，是否允许同时接收多个网络连接，选取哪种事件驱动模型来处理连接请求，每个 word process 可以同时支持的最大连接数等。

上述例子就表示每个 work process 支持的最大连接数为 1024。

这部分的配置对 Nginx 的性能影响较大，在实际中应该灵活配置。

3.2.3 第三部分：http块

```
http {
    include       mime.types;
    default_type  application/octet-stream;


    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

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

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }

    }

}
```

这算是 Nginx 服务器配置中最频繁的部分，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这里。

需要注意的是：http 块也可以包括 http 全局块、server 块。

1、http全局块

http 全局块配置的指令包括文件引入、MIME-TYPE 定义、日志自定义、连接超时时间、单链接请求数上限等。

2、server块

这块和虚拟主机有密切关系，虚拟主机从用户角度看，和一台独立的硬件主机是完全一样的，该技术的产生是为了节省互联网服务器硬件成本。

每个 http 块可以包括多个 server 块，而每个 server 块就相当于一个虚拟主机。

而每个 server 块也分为全局 server 块，以及可以同时包含多个 locaton 块。

（1）全局 server 块

最常见的配置是本虚拟机主机的监听配置和本虚拟主机的名称或 IP 配置。

（2）location 块

一个 server 块可以配置多个 location 块。

这块的主要作用是基于 Nginx 服务器接收到的请求字符串（例如 server_name/uri-string），对虚拟主机名称（也可以是 IP 别名）之外的字符串（例如 前面的 /uri-string）进行匹配，对特定的请求进行处理。地址定向、数据缓存和应答控制等功能，还有许多第三方模块的配置也在这里进行。

location 指令说明：该指令用于匹配 URL。

语法如下：

```
1 location [ = | ~ | ~* | ^~] uri {
2
3 }
```

- = ：用于不含正则表达式的 uri 前，要求请求字符串与 uri 严格匹配，如果匹配成功，就停止继续向下搜索并立即处理该请求。
- ~：用于表示 uri 包含正则表达式，并且区分大小写。
- ~*：用于表示 uri 包含正则表达式，并且不区分大小写。
- ^~：用于不含正则表达式的 uri 前，要求 Nginx 服务器找到标识 uri 和请求字符串匹配度最高的 location 后，立即使用此 location 处理请求，而不再使用 location块中的正则 uri 和请求字符串做匹配。

注意：如果 uri 包含正则表达式，则必须要有 ~ 或者 ~* 标识。

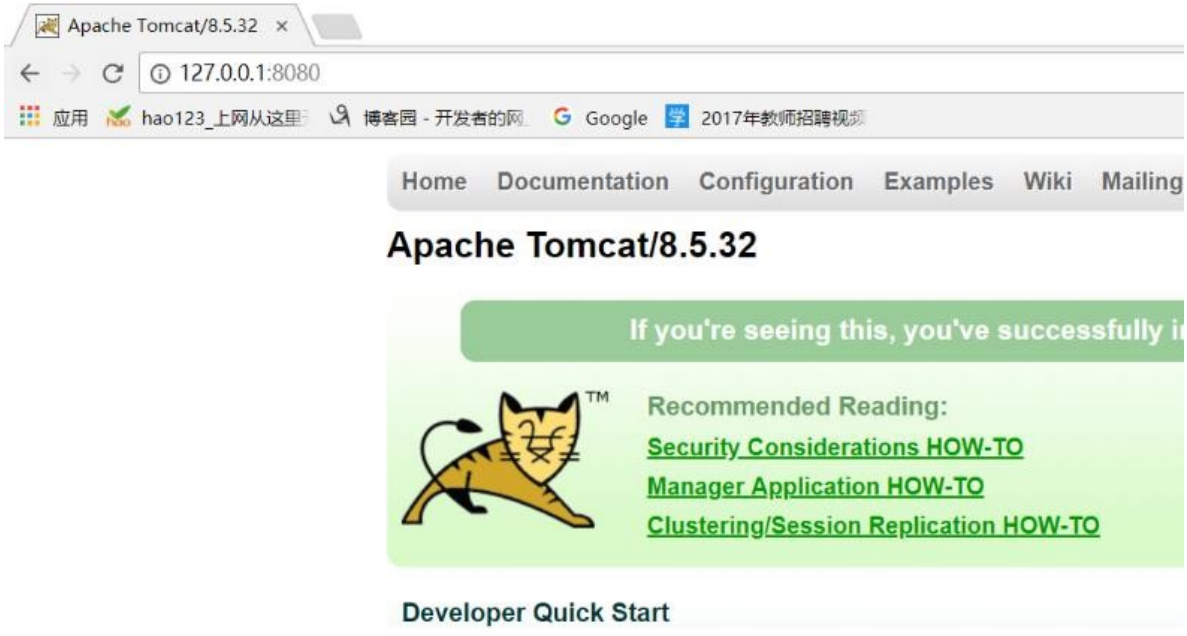
四、nginx配置实例- 反向代理

4.1 反向代理实例一

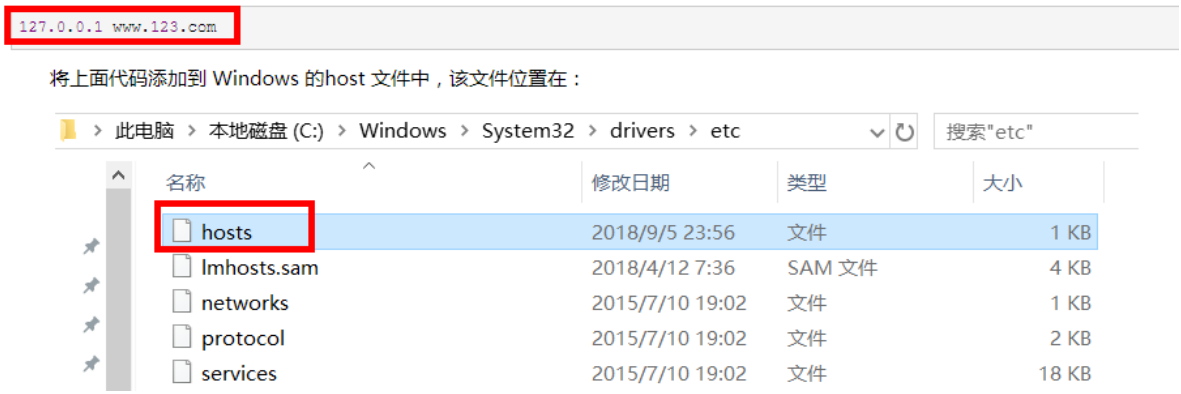
实现效果：使用 nginx 反向代理，访问 www .123 .com 直接跳转到 127.0.0.1:8080

4.1.1 实验代码

1、 启动一个 tomcat，浏览器地址栏输入 127.0.0.1:8080，出现如下界面



2、通过修改本地 host 文件，将 www.123.com 映射到 127.0.0.1

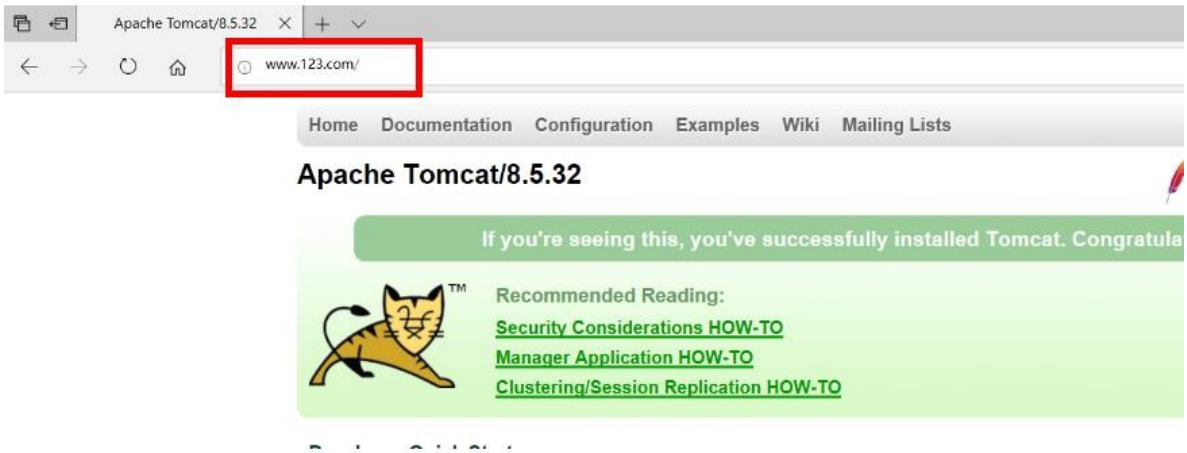


配置完成之后，我们便可以通过 www.123.com:8080 访问到第一步出现的 Tomcat 初始界面。那么如何只需要输入 www.123.com 便可以跳转到 Tomcat 初始界面呢？便用到 nginx的反向代理。

3、 在 nginx.conf 配置文件中增加如下配置

```
server {
    listen       80;
    server_name  www.123.com;

    location / {
        proxy_pass http://127.0.0.1:8080;
        index  index.html index.htm index.jsp;
    }
}
```

如上配置，我们监听 80 端口，访问域名为 www.123.com，不加端口号时默认为 80 端口，故访问该域名时会跳转到 127.0.0.1:8080 路径上。在浏览器端输入 www.123.com 结果如下：



4.3 反向代理实例二

实现效果：使用 nginx 反向代理，根据访问的路径跳转到不同端口的服务中nginx 监听端口为 9001，

访问 http://127.0.0.1:9001/edu/ 直接跳转到 127.0.0.1:8001

访问 http://127.0.0.1:9001/vod/ 直接跳转到 127.0.0.1:8002

4.3.1 实验代码

第一步，准备两个 tomcat，一个 8001 端口，一个 8002 端口，并准备好测试的页面

第二步，修改 nginx 的配置文件

在 http 块中添加 server{}

```
server {
        listen 9001;
        server_name localhost;

        location ~ /edu/ {
                proxy_pass http://localhost:8001;
        }

        location ~ /vod/ {
                proxy_pass http://localhost:8002;
        }

}
```

五、nginx配置实例- 负载均衡

实现效果： 配置负载均衡

5.1 实验代码

1、首先准备两个同时启动的 Tomcat

2、 在 nginx.conf 中进行配置

```
http {
......
    upstream myserver{
        ip_hash;
        server 115.28.52.63:8080 weight=1;
        server 115.28.52.63:8180 weight=1;
     }
.....
     server{
        location / {
                .........
                proxy_pass http://myserver;
                proxy_connect_timeout 10;
        }
            .........
      }
}
```

随着互联网信息的爆炸性增长，负载均衡（load balance）已经不再是一个很陌生的话题，顾名思义，**负载均衡即是将负载分摊到不同的服务单元，既保证服务的可用性，又保证响应**足够快，给用户很好的体验**。

快速增长的访问量和数据流量催生了各式各样的负载均衡产品，很多专业的负载均衡硬件提供了很好的功能，但却价格不菲，这使得负载均衡软件大受欢迎，nginx 就是其中的一个，在 linux 下有 Nginx、LVS、Haproxy 等等服务可以提供负载均衡服务，而且 Nginx 提供了几种分配方式(策略)：

1 、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉，能自动剔除。

2 、weight

weight 代表权重，默认为 1,权重越高被分配的客户端越多

指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况。 例如：

```
upstream server_pool {
    server 192.168.5.21 weight = 10 ;
    server 192.168.5.22 weight = 10 ;
}
```

3 、ip_hash

每个请求按访问 ip 的 hash 结果分配，这样每个访客固定访问一个后端服务器，可以解决 session 的问题。 例如：

```
upstream server_pool {
    ip_hash ;
    server 192.168.5.21:80 ;
    server 192.168.5.22:80 ;
}
```

4 、fair （第三方）

按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。

```
upstream server_pool {
    server 192.168.5.21:80 ;
    server 192.168.5.22:80 ;
    fair ;
}
```

六、 nginx配置实例- 动静分离

Nginx 动静分离简单来说就是把动态跟静态请求分开，不能理解成只是单纯的把动态页面和静态页面物理分离。严格意义上说应该是**动态请求跟静态请求分开**，可以理解成使用 Nginx处理静态页面，Tomcat 处理动态页面。

动静分离从目前实现角度来讲大致分为两种。

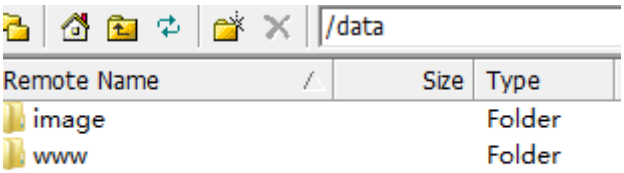
- 一种是纯粹把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案；
- 另外一种方法就是动态跟静态文件混合在一起发布，通过 nginx 来分开。通过 location 指定不同的后缀名实现不同的请求转发。通过 expires 参数设置，可以使浏览器缓存过期时间，减少与服务器之前的请求和流量。

具体 Expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量。此种方法非常适合不经常变动的资源。（如果经常更新的文件，不建议使用 Expires 来缓存），我这里设置 3d，表示在这 3 天之内访问这个 URL，发送一个请求，比对服务器该文件最后更新时间没有变化，则不会从服务器抓取，返回状态码304，如果有修改，则直接从服务器重新下载，返回状态码 200。

6.1 实验代码

6.1.1 项目资源准备

（1）在 liunx 系统中准备静态资源，用于进行访问



6.1.2 进行 nginx 配置

找到 nginx 安装目录，打开/conf/nginx.conf 配置文件，

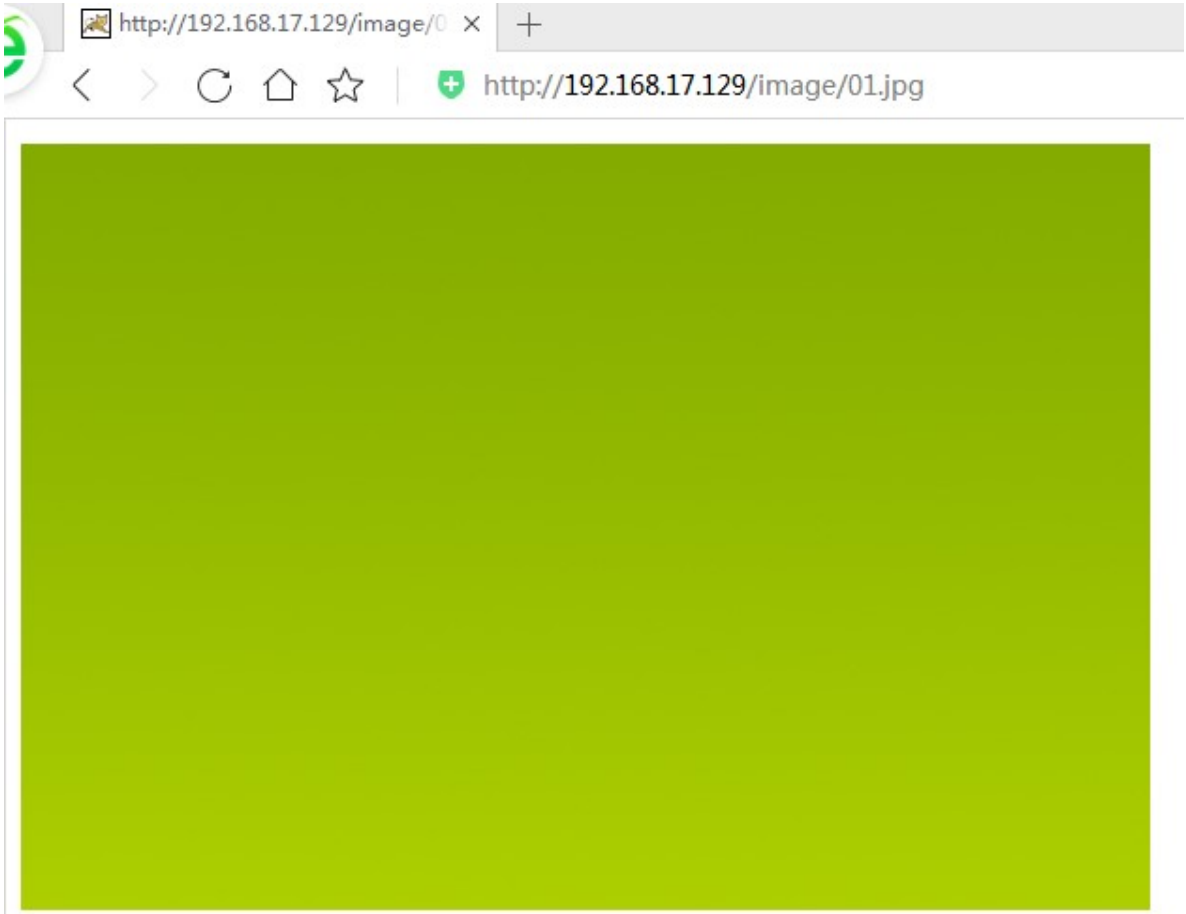
```
server {
    listen       80;
    server_name  192.168.17.129;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location /www/ {
            root /data/;
            index index.html index.htm;
    }

    location /image/ {
            root /data/;
            autoindex on;
    }
```

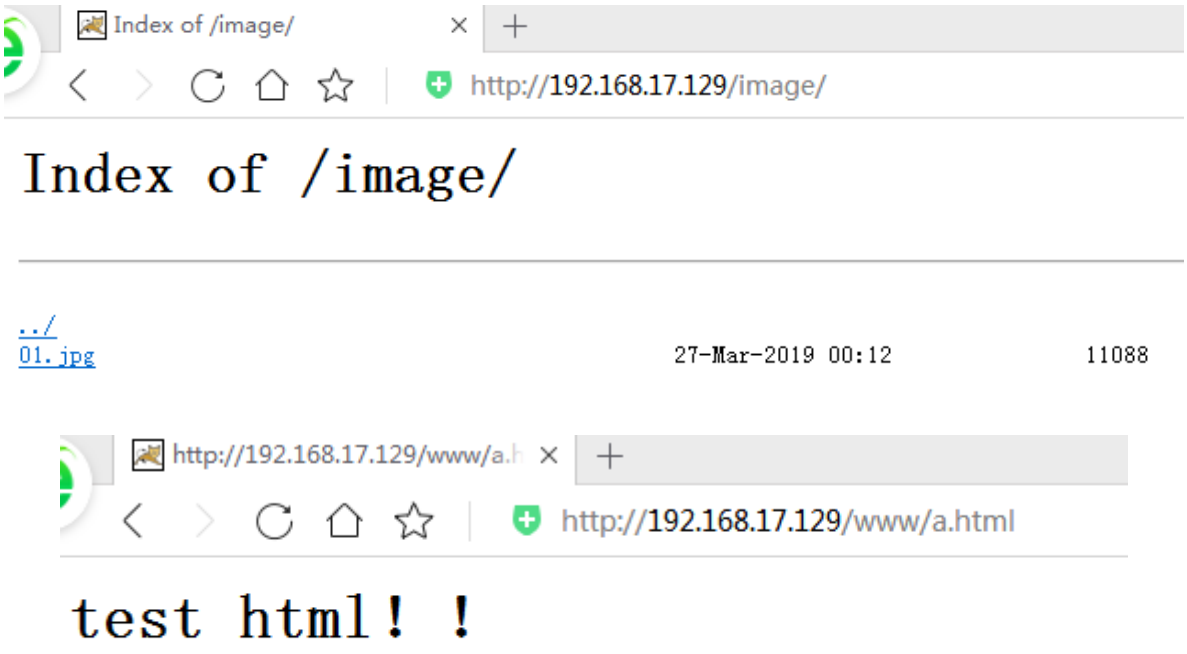
添加监听端口、访问名字。重点是添加 location，最后检查 Nginx 配置是否正确即可，然后测试动静分离是否成功，需要删除后端 tomcat服务器上的某个静态文件，查看是否能访问，如果可以访问说明静态资源 nginx 直接返回了，不走后端 tomcat 服务器。

6.1.3 测试

（1）浏览器中输入地址

http://192.168.17.129/image/01.jpg

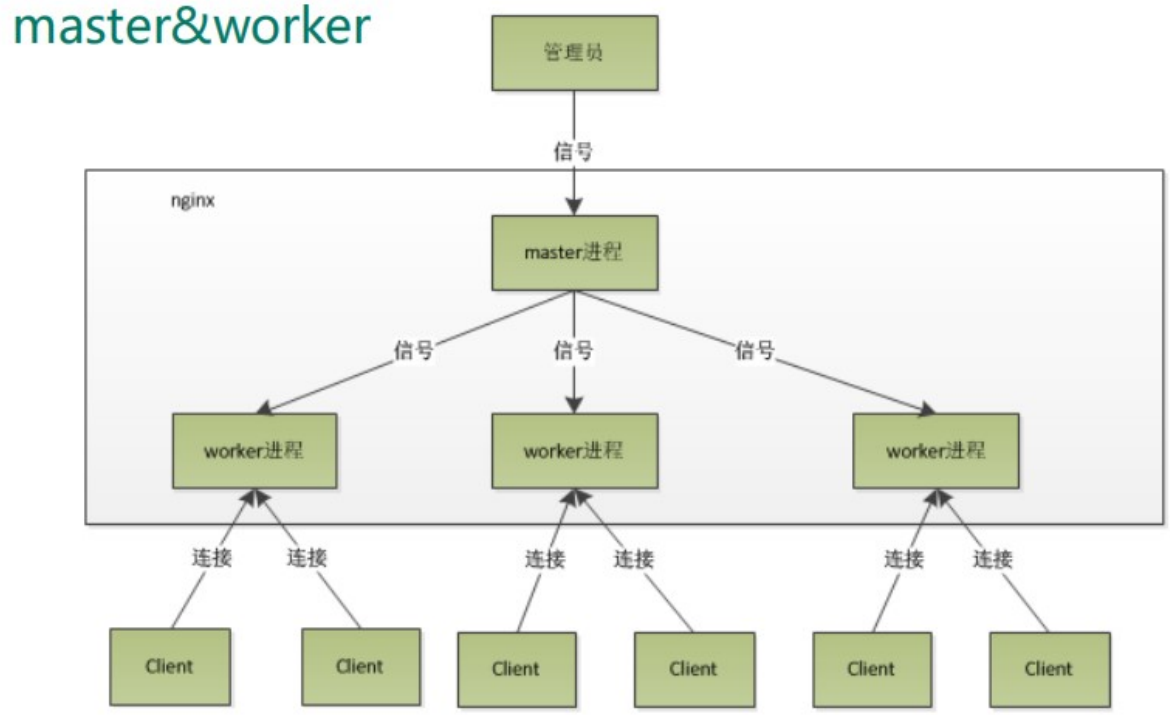




七、nginx 原理与优化参数配置

7.1 工作模式

nginx 有两种工作模式:master-worker 模式和单进程模式。在 master-worker 模式下，有一个 master 进程和至少一个的 worker 进程，单进程模式顾名思义只有一个进程。这两种模式有各自的特点和适用场景。



7.1.1 master-worker

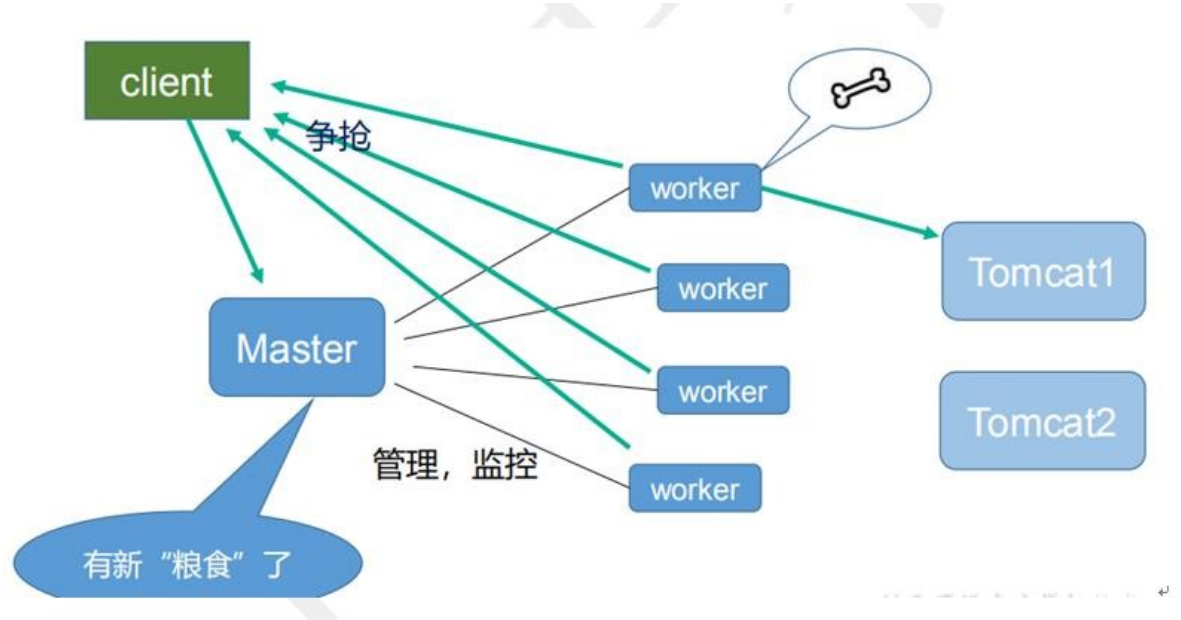
该模式下，nginx 启动成功后，会有一个 master 进程和至少一个的 worker 进程。master 进程负责处理系统信号，加载配置，管理 worker 进程（启动，杀死，监控，发送消息/信号等）。worker 进程负责处理具体的业务逻辑，也就是说，对外部来说，真正提供服务的是 worker 进程。生产环境下一般使用这种模式，因为这种模式有以下优点：

- 稳定性高，只要还有 worker 进程存活，就能够提供服务，并且一个 worker 进程挂掉 master 进程会立即启动一个新的 worker 进程，保证 worker 进程数量不变，降低服务中断的概率。
- 配合 linux 的 cpu 亲和性配置，可以充分利用多核 cpu 的优势，提升性能。
- 处理信号/配置重新加载/升级时可以做到尽可能少或者不中断服务（热重启）。

7.1.2 单进程模式

单进程模式下，nginx 启动后只有一个进程，nginx 的所有工作都由这个进程负责。由于只有一个进程，因此可以很方便地利用 gdb 等工具进行调试。该模式不支持 nginx 的平滑升级功能，任何的信号处理都可能造成服务中断，并且由于是单进程，进程挂掉后，在没有外部监控的情况下，无法重启服务。因此，该模式一般只在开发阶段和调试时使用，生产环境下不会使用。（了解）

7.2 worker如何进行工作的



7.3 master-workers的机制的好处

首先，对于每个 worker 进程来说，独立的进程，不需要加锁，所以省掉了锁带来的开销，同时在编程以及问题查找时，也会方便很多。其次，采用独立的进程，可以让互相之间不会影响，一个进程退出后，其它进程还在工作，服务不会中断，master 进程则很快启动新的worker 进程。当然，worker 进程的异常退出，肯定是程序有 bug 了，异常退出，会导致当前 worker 上的所有请求失败，不过不会影响到所有请求，所以降低了风险。

（1 ）可以使用 nginx –s reload 热部署，利用 nginx 进行热部署操作；

（2 ）每个 woker 是独立的进程，如果有其中的一个 woker 出现问题，其他 woker 独立的，继续进行争抢，实现请求过程，不会造成服务中断。

7.4 需要设置多少**个 worker**

Nginx 同 redis 类似都采用了 io 多路复用机制，每个 worker 都是一个独立的进程，但每个进程里只有一个主线程，通过**异步非阻塞的方式**来处理请求， 即使是千上万个请求也不在话下。每个 worker 的线程可以把一个 cpu 的性能发挥到极致。所以 worker 数和服务器的 cpu数相等是最为适宜的。设少了会浪费 cpu，设多了会造成 cpu 频繁切换上下文带来的损耗。

7.5 设置 worker 数量

```
worker_processes 4

#work 绑定 cpu(4 work 绑定 4cpu)
worker_cpu_affinity 0001 0010 0100 1000

#work 绑定 cpu (4 work 绑定 8cpu 中的 4 个)
worker_cpu_affinity 0000001 00000010 00000100 00001000
```

7.6 连接数 worker_connection

这个值是表示每个 worker 进程所能建立连接的最大值，所以，一个 nginx 能建立的最大连接数，应该是 worker_connections * worker_processes。当然，这里说的是最大连接数，对于HTTP 请求本地资源来说， 能 够支持的最大并发数量是 worker_connections *worker_processes，如果是支持 http1.1 的浏览器每次访问要占两个连接，所以**普通的静态访问最大并发数是： worker_connections * worker_processes /2**，而如果是 **HTTP 作为反向代理来说，最大并发数量应该是 worker_connections *worker_processes/4。**因为作为反向代理服务器，每个并发会建立与客户端的连接和与后端服务的连接，会占用两个连接。

7.6.1 发送请求，占用了 woker 的几个连接数

答案：2 或者 4 个

7.6.2 nginx支持的最大并发数是多少

- 普通的静态访问最大并发数是： worker_connections * worker_processes /2
- 而如果是 HTTP 作 为反向代理来说，最大并发数量应该是 worker_connections *worker_processes/4

7.7 为什么 Nginx比其他 web服务器并发高（Nginx工作原理）

Nginx配置 use epoll后，以异步非阻塞方式工作，能够轻松处理百万级的并发连接。

处理过程：每进来一个 request，会有一个 worker进程去处理。但不是全程的处理，处理到可能发生阻塞的地方。比如向后端服务器转发 request，并等待请求返回。那么，这个处理的 worker不会这么傻等着，他会在发送完请求后，注册一个事件：“如果后端服务器返回了，告诉我一声，我再接着干”。于是他就休息去了。此时，如果再有新的 request 进来，他就可以很快再按这种方式处理。而一旦后端服务器返回了，就会触发这个事件，worker才会来接手，这个 request才会接着往下走。通过这种快速处理，快速释放请求的方式，达到同样的配置可以处理更大并发量的目的。

八、其他Nginx配置

8.1 nginx 负载均衡

```
#user  nobody;
worker_processes  auto;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;

events {
    worker_connections  1024;
}

http {
    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;

    #用于负载均衡
     upstream webservers {
        #weigth参数表示权值，权值越高被分配到的几率越大
        server 10.186.128.202:8081 weight=10;
        server 10.186.128.200:8081 weight=10;
     }

   #设置监听端口
    server {
        listen       8089;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        location / {
             #直接复制替换就可以了
            proxy_set_header  X-Forwarded-For $remote_addr;
            proxy_set_header  X-Forwarded-Host $server_name;
            # 重写请求头部，保证网站所有页面都可访问成功
            proxy_set_header Host $host;
            #添加反向代理，代理地址填写 upstream 声明的名字
            proxy_pass http://webservers;
        }

        #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;
        }
    }
}
```

负载均衡地址：

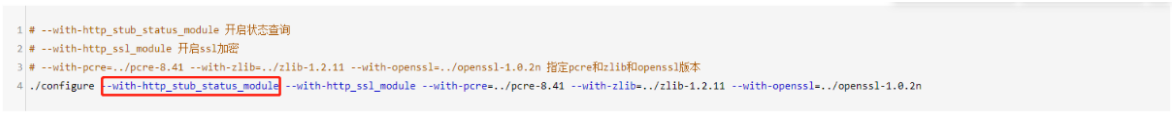
```
    #gzip  on;

    #用于负载均衡
     upstream webservers {
        #weigth参数表示权值，权值越高被分配到的几率越大
        server 10.186.128.202:8081 weight=10;
        server 10.186.128.200:8081 weight=10;
     }
```

监听服务:

```

    #设置监听端口
     server {
         listen       8089;
         server_name  localhost;

         #charset koi8-r;

         #access_log  logs/host.access.log  main;

         location / {
         #直接复制替换就可以了
             proxy_set_header  X-Forwarded-For $remote_addr;
             proxy_set_header  X-Forwarded-Host $server_name;
             proxy_set_header Host $host;
             proxy_pass http://webservers;
         }
```

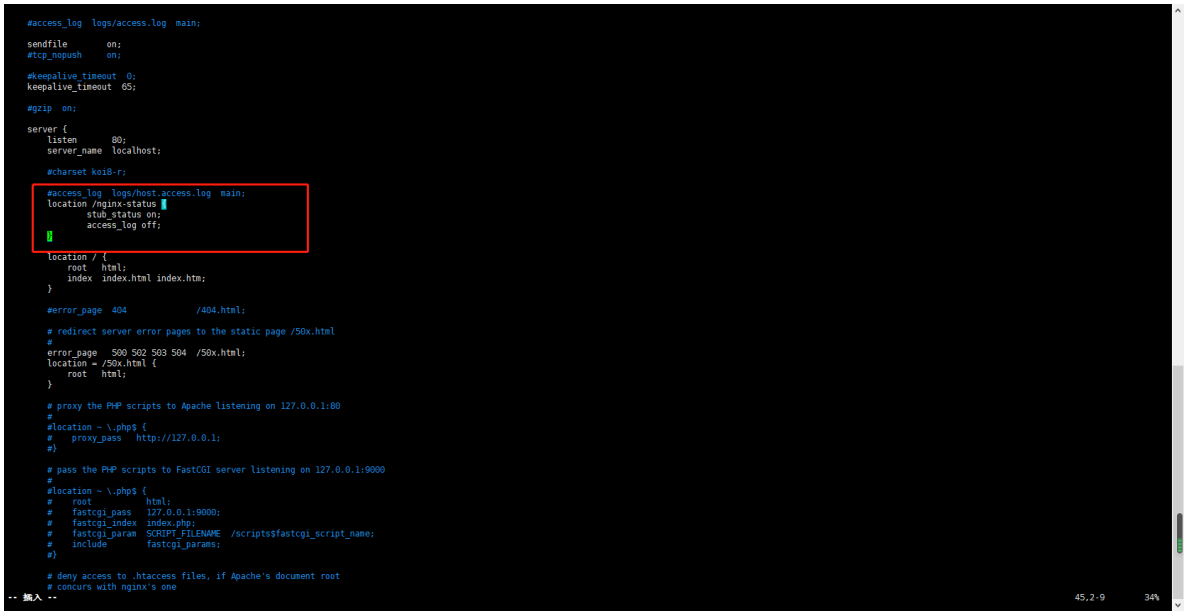
8.2 nginx 状态统计

1、安装 nginx 时将 --with-http_stub_status_module 模块开启

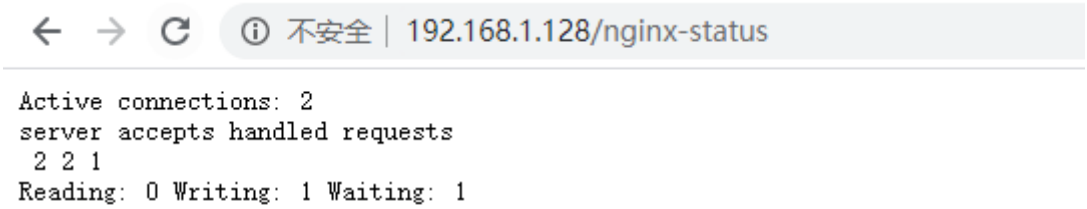


2、修改 nginx 配置文件（写入要访问的 server 标签中）

```
cd /usr/local/nginx/conf

vim nginx.conf

location /nginx_status {
    stub_status on;
    access_log off;
}
```



3、客户端访问网址:http://IP/nginx_status



结果说明:

- Active connections：正在处理的活动连接数
- server accepts handled requests
- - 第一个 server 表示Nginx启动到现在共处理了9个连接
  - 第二个 accepts 表示Nginx启动到现在共成功创建 9 次握手
  - 第三个 handled requests 表示总共处理了 21 次请求
  - 请求丢失数 = 握手数 - 连接数， 可以看出目前为止没有丢失请求
- Reading: 0 Writing: 1 Waiting: 1
- - Reading：Nginx 读取到客户端的 Header 信息数
  - Writing：Nginx 返回给客户端 Header 信息数
  - Waiting：Nginx 已经处理完正在等候下一次请求指令的驻留链接（开启keep-alive的情况下，这个值等于Active - (Reading+Writing)

8.3 nginx 目录保护

1、原理和 apache 的目录保护原理一样（利用 nginx状态统计 接着完成）

2、在状态统计的 location 中添加:

```
auth_basic "Welcome to nginx_status!";
auth_basic_user_file /usr/local/nginx/html/htpasswd.nginx;
```

```
location /nginx_status {
    stub_status on;
    access_log off;
    auth_basic "Welcome to nginx_status!";
    auth_basic_user_file /usr/local/nginx/html/htpasswd.nginx;
}
```

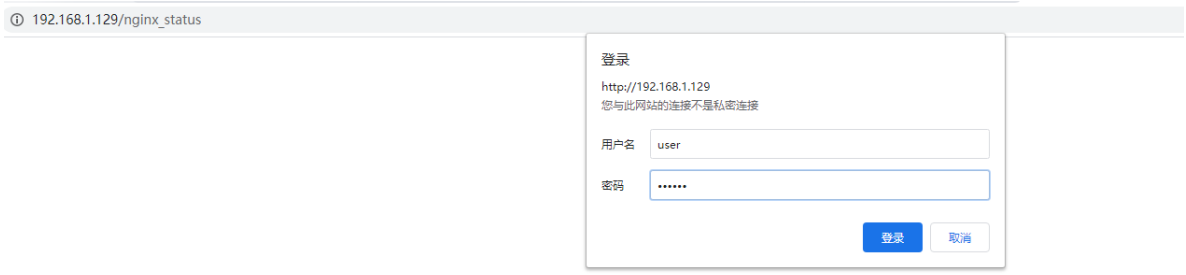
3、使用 http 的命令 htpasswd 进行用户密码文件的创建（生成在上面指定的位置）

```
# 安装httpd
yum -y install httpd

# 配置nginx密码文件，用户名为user
htpasswd -c /usr/local/nginx/html/htpasswd.nginx user
```

4、重启 nginx 并再次访问统计页面

```
nginx -s stop
nginx
```

输入配置的用户名和密码才能进入页面：



8.4 nginx基于IP的身份验证（访问控制）

1、接着上一个实验：nginx 目录保护，完成操作

2、在状态统计的 location 中添加：

```
# 允许某个ip访问
allow 192.168.1.102;
# 拒绝一个ip段访问
deny 192.168.1.0/24;
```

```
location /nginx_status {
    stub_status on;
    access_log off;
    auth_basic "Welcome to nginx_status!";
    auth_basic_user_file /usr/local/nginx/html/htpasswd.nginx;
    allow 192.168.1.102;
    deny 192.168.1.0/24;
}
```

仅允许 192.168.1.102 访问服务器

8.5 nginx 的虚拟主机（基于域名）

1、提前准备好两个网站的域名，并且规划好两个网站网页存放目录

/etc/hosts文件中配置：

```
192.168.88.10 blog.atguigu.com
192.168.88.10 bbs.atguigu.com
```

/usr/local/nginx/html目录下新建blog目录和bbs目录，并在各个目录下新建index.html

```
mkdir /usr/local/nginx/html/blog
cd /usr/local/nginx/html/blog
vim index.html
# 输入
blog.atguigu.com


mkdir /usr/local/nginx/html/bbs
cd /usr/local/nginx/html/bbs
vim index.html
# 输入
bbs.atguigu.com
```

2、在 Nginx 主配置文件中并列编写两个 server 标签，并分别写好各自信息

```
server {
    listen 80;
    server_name blog.atguigu.com;
    index index.html index.htm index.php;
    root html/blog;
    access_log logs/blog-access.log main;
}
server {
    listen 80;
    server_name bbs.atguigu.com;
    index index.html index.htm index.php;
    root html/bbs;
    access_log logs/bbs-access.log main;
}
```

3、分别访问两个不同的域名验证结果

8.6 nginx 实现 https {证书+rewrite}

继续上个实验：nginx 的虚拟主机（基于域名）。

1、安装 nginx 时，需要将--with-http_ssl_module 模块开启

```
# --with-http_stub_status_module 开启状态查询
# --with-http_ssl_module 开启ssl加密
# --with-pcre=../pcre-8.41 --with-zlib=../zlib-1.2.11 --with-openssl=../openssl-1.0.2n 指定pcre和zlib和openssl版本
./configure --with-http_stub_status_module --with-http_ssl_module --with-pcre=../pcre-8.41 --with-zlib=../zlib-1.2.11 --with-openssl=../openssl-1.0.2n
```

2、在对应要进行加密的 server 标签中添加以下内容开启 SSL

```
server {
    .......;
    ssl on;
    ssl_certificate /usr/local/nginx/conf/ssl/atguigu.crt;
    ssl_certificate_key /usr/local/nginx/conf/ssl/atguigu.key;
    ssl_session_timeout 5m;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers
    "EECDH+CHACHA20:EECDH+CHACHA20-
draft:EECDH+AES128:RSA+AES128:EECDH+AES256:RSA+AES256:EECDH+3DES:RSA+3DES:!MD5";
}
```

3、生成证书和秘钥文件

注意：在实验环境中可以用命令生成测试，在生产环境中必须要在 https 证书厂商注册

建立服务器私钥，生成 RSA 密钥：

```
mkdir /usr/local/nginx/conf/ssl
cd /usr/local/nginx/conf/ssl

openssl genrsa -out atguigu.key 1024
```

需要依次输入国家，地区，组织，email。最重要的是有一个 common name，可以写你的名字或者域名。如果为了 https 申请，这个必须和域名吻合，否则会引发浏览器警报。生成的 csr 文件交给 CA 签名后形成服务端自己的证书：

```
openssl req -new -key atguigu.key -out atguigu.csr
```

```
[root@localhost ssl]# openssl req -new -key atguigu.key -out atguigu.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:CN
State or Province Name (full name) []:BJ
Locality Name (eg, city) [Default City]:BJ
Organization Name (eg, company) [Default Company Ltd]:atguigu
Organizational Unit Name (eg, section) []:YJS
Common Name (eg, your name or your server's hostname) []:bbs.atguigu.com
Email Address []:
```

生成签字证书：

```
openssl x509 -req -days 365 -sha256 -in atguigu.csr -signkey atguigu.key -out
atguigu.crt
```

将私钥和证书复制到指定位置：

```
cp atguigu.crt /usr/local/nginx/conf/ssl/atguigu.crt
cp atguigu.key /usr/local/nginx/conf/ssl/atguigu.key
```

4、设置 http 自动跳转 https 功能

原有的 server 标签修改监听端口

```
server {
    ..........;
    listen 443;
}
```

新增以下 server 标签（利用虚拟主机+rewrite 的功能）

```
server{
    listen 80;
    server_name bbs.atguigu.com;
    rewrite ^(.*)$ https://bbs.atguigu.com permanent;
    root html;
    index index.html index.htm;
}
```

5、重启 nginx，并测试

访问的主机修改C:\Windows\System32\drivers\etc\hosts文件，添加虚拟机ip映射：

```
192.168.88.10    bbs.atguigu.com
```

访问bbs.atguigu.com后自动域名重定向：

九、nginx搭建高可用集群

为知笔记地址：Nginx+keepalive高可用解决方案

GitHub地址：
https://github.com/wangliu1102/StudyNotes/tree/master/%E5%B0%9A%E7%A1%85%E8%B0%B7Java/%E5%9B%9B%E3%80%81JavaEE%E9%AB%98%E7%BA%A7/6%E3%80%81Nginx/%E5%AE%89%E8%A3%85