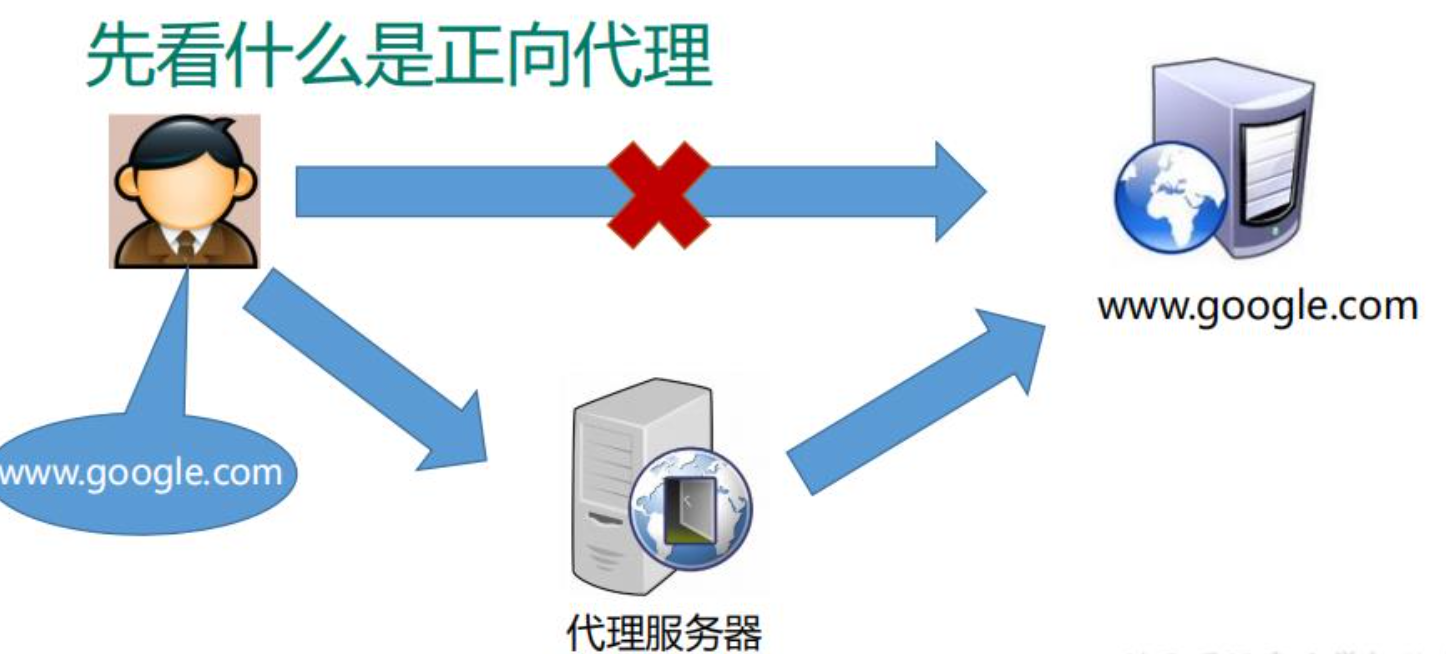
# Nginx

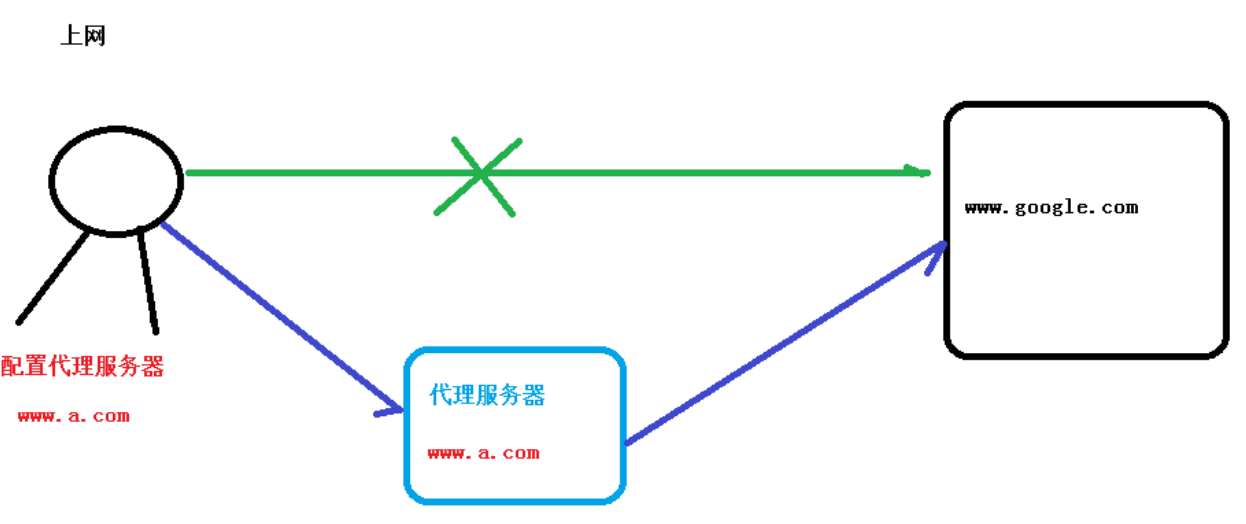
高性能的HTTP和反向代理的服务器，处理高并发能力是十分强大的，能经受高负载的考验，有报告表明能支持高达50,000个并发连接数

## 代理

### 正向代理

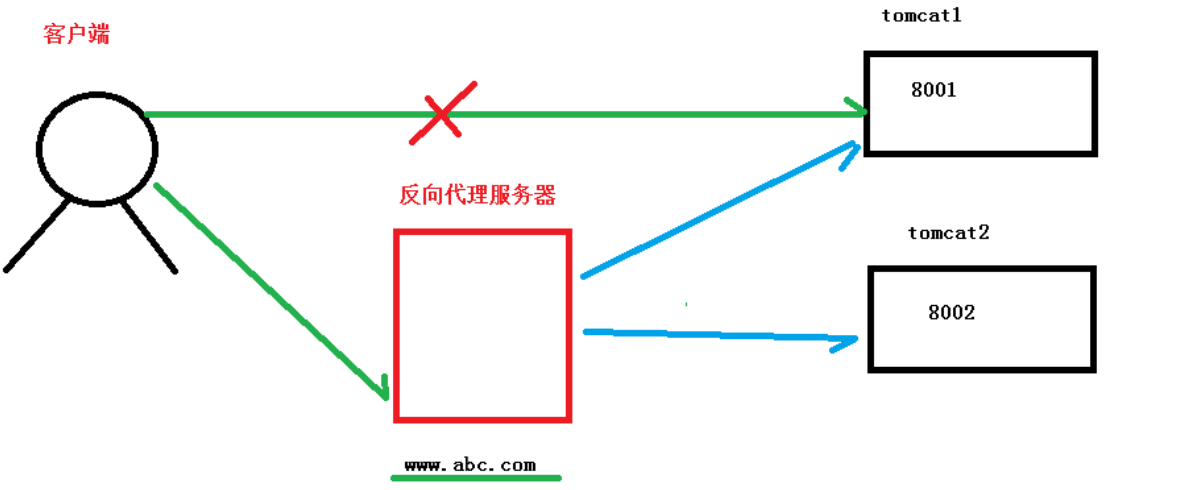
如果把局域网外的 Internet 想象成一个巨大的资源库，则局域网中的客户端要访问 Internet，则需要通过代理服务器来访问，这种代理服务就称为正向代理。通俗来讲，需要在客户端配置代理服务器进行指定网站访问



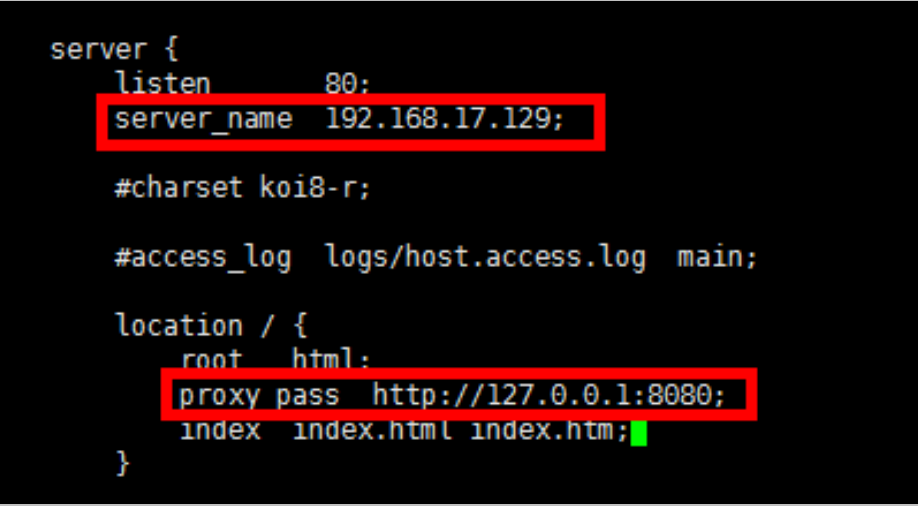


### 反向代理

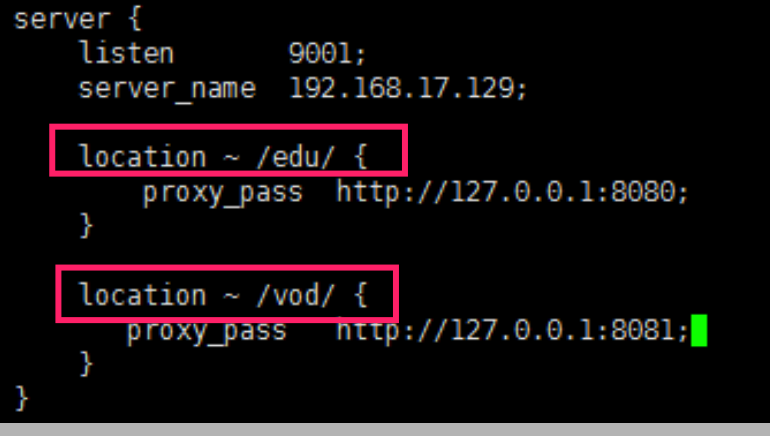
反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，再返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器 IP 地址



1. nginx配置实例1，代理到后台tomcat



1. nginx配置实例2，根据访问地址代理到不同的后台tomcat



1. location配置语法规则，语法规则很简单，一个location关键字，后面跟着可选的修饰符，后面是要匹配的字符，花括号中是要执行的操作。

location的配置有两种形式，前缀字符和正则。查找匹配的时候，先查找前缀字符，选择最长匹配项，再查找正则。正则的优先级高于前缀字符。正则的查找是按照在配置文件中的顺序进行的。因此正则的顺序很重要，建议越精细的放的越靠前。使用=精准匹配可以加快查找的顺序，如果根域名经常被访问的话建议使用=



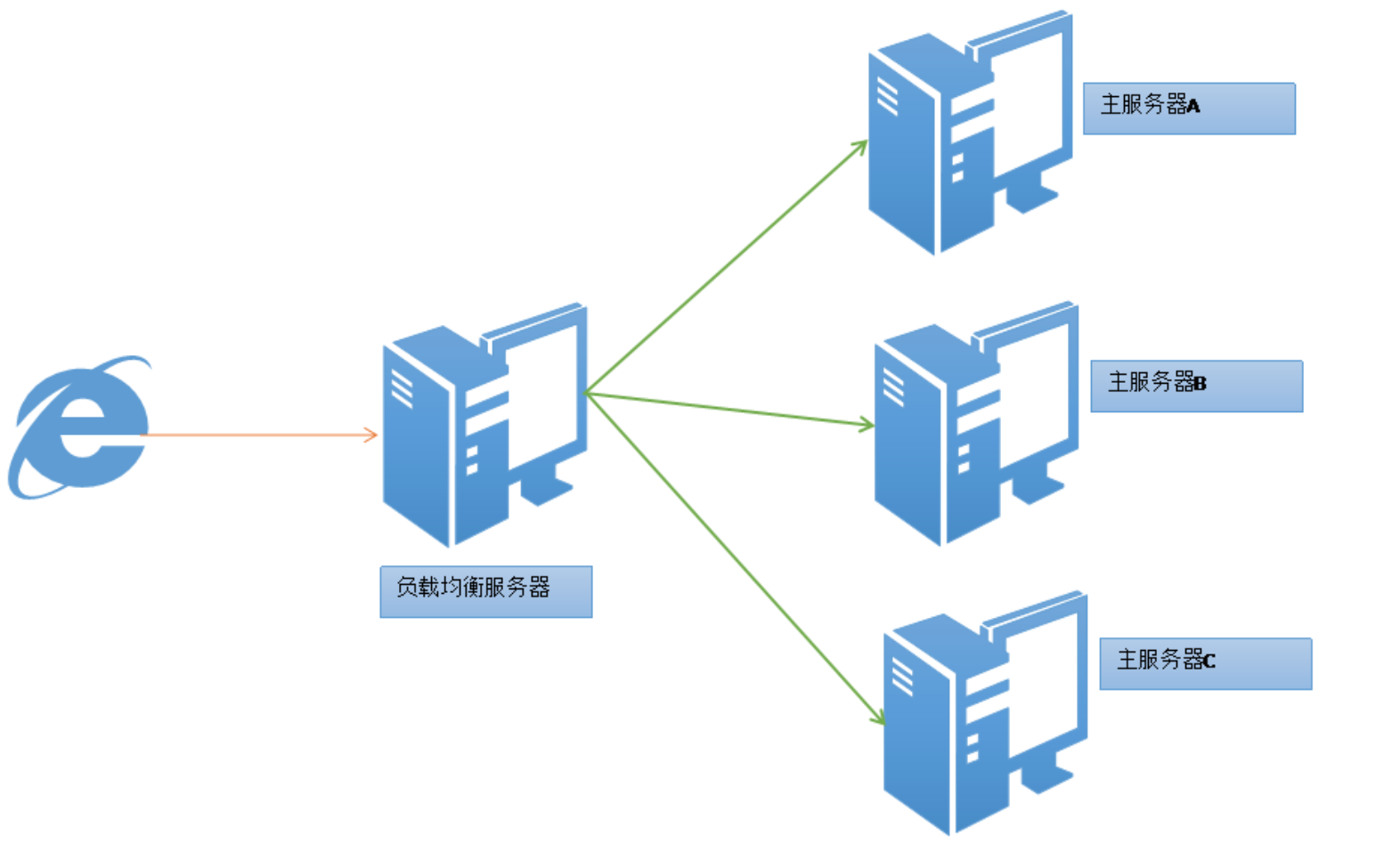
* = 表示**精确**匹配，只有请求的url路径与后面的字符串完全相等时，才会命中。
* ~ 表示该规则是使用**正则**定义的，区分大小写。
* ~\* 表示该规则是使用**正则**定义的，不区分大小写。
* ^~ 表示如果该符号后面的字符是最佳匹配，采用该规则，不再进行后续的查找
* @用来定义一个命名location。主要用于内部重定向，不能用来处理正常的请求。其用法如下

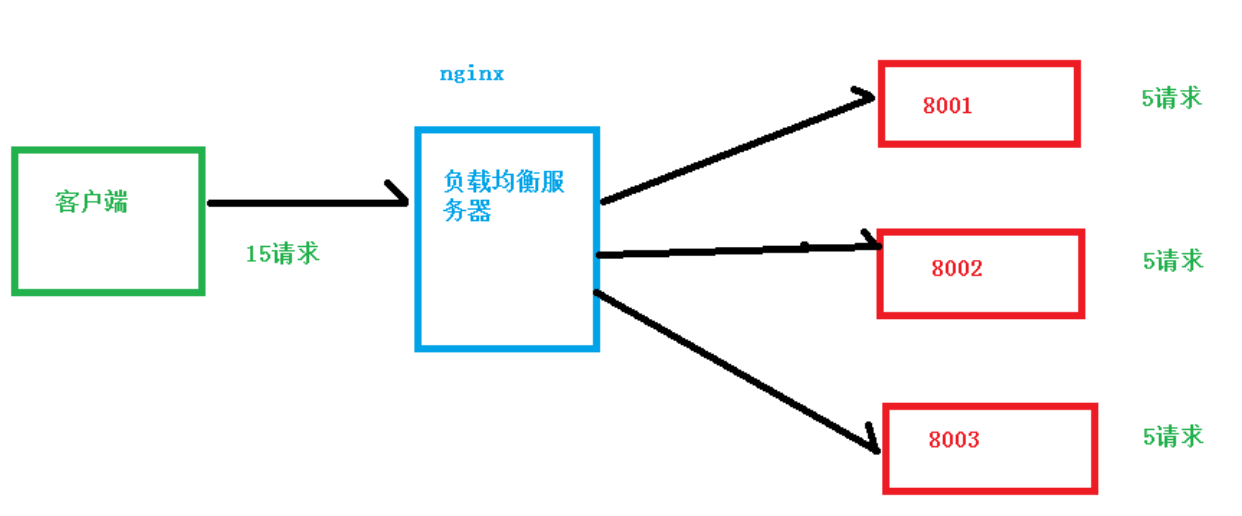


上例中，当尝试访问url找不到对应的文件就重定向到我们自定义的命名location（此处为custom）。值得注意的是，命名location中不能再嵌套其它的命名location

## 负载均衡

1. 通过增加服务器的数量，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
2. 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





1. nginx负载均衡配置



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 提供了几种负载均衡策略，内置策略和扩展策略

* + 1. 轮询(轮询加权round-robin)，内置策略，每个请求按轮询方式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



weight 代表权重默认为 1，权重越高被分配的客户端越多

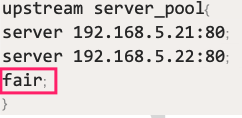
* + 1. Ip hash，内置策略，每个请求按访问客户端ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题



* + 1. 最少连接，内置策略，下一个请求将被分派到活动连接数量最少的服务器

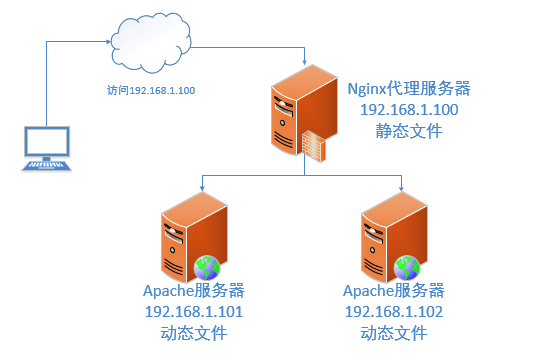


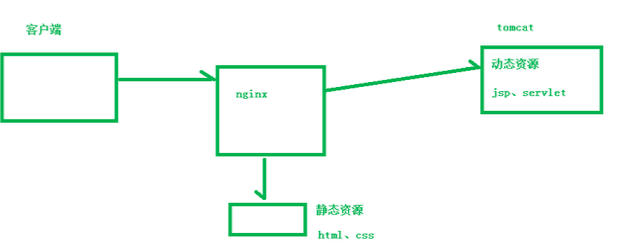
* + 1. fair，扩展策略，按后端服务器的响应时间来分配请求，响应时间短的优先分配



## 动静分离

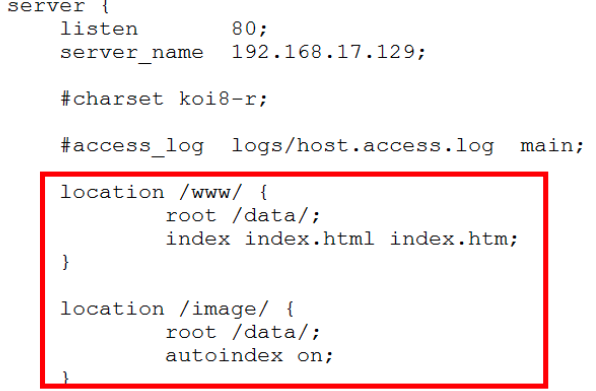
动静分离是将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问。为了加快网站的解析速度，可以把动态页面和静态页面由不同的服务器来解析，加快解析速度。降低原来单个服务器的压力





动静分离从目前实现角度来讲大致分为两种

* 1. 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
  2. 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



添加autoindex on配置后

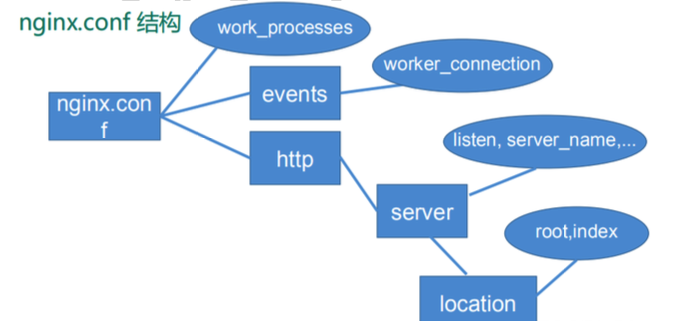


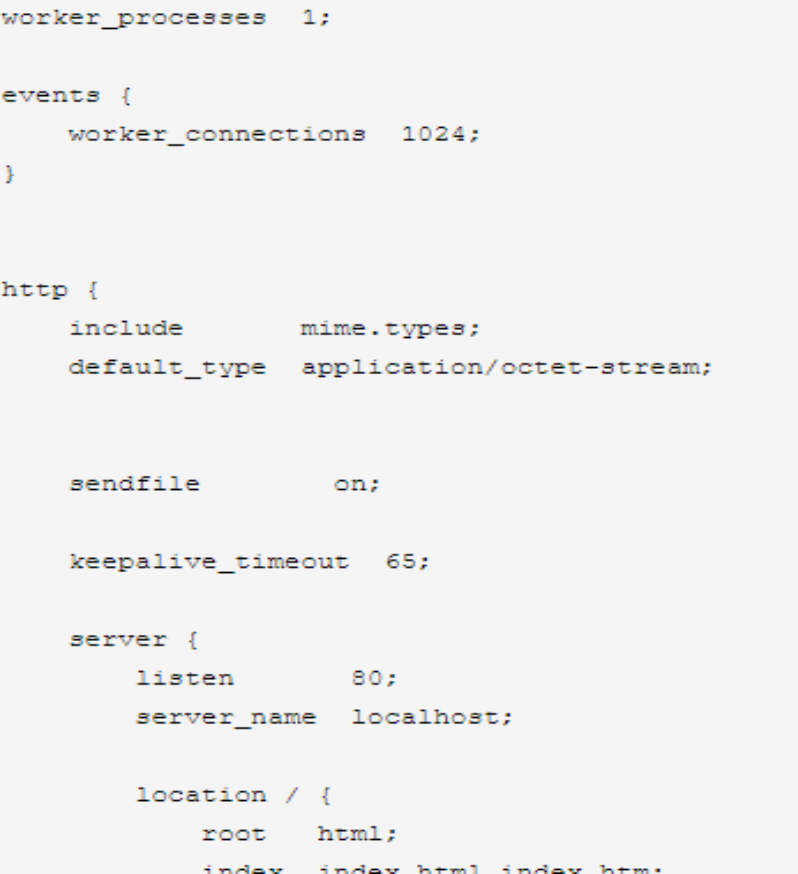
具体配置



## Nginx配置

nginx安装目录下，其默认的配置文件都放在安装目录的conf目录下，主配置文件nginx.conf 也在其中，对nginx的使用基本上都是对此配置文件进行相应的修改。Nginx配置文件包含三部分内容





1. **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

比如worker\_processes 1 #处理并发数的配置

1. **events块**，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 worker process 可以同时支持的最大连接数等

比如worker\_connections 1024 #支持的最大连接数为 102

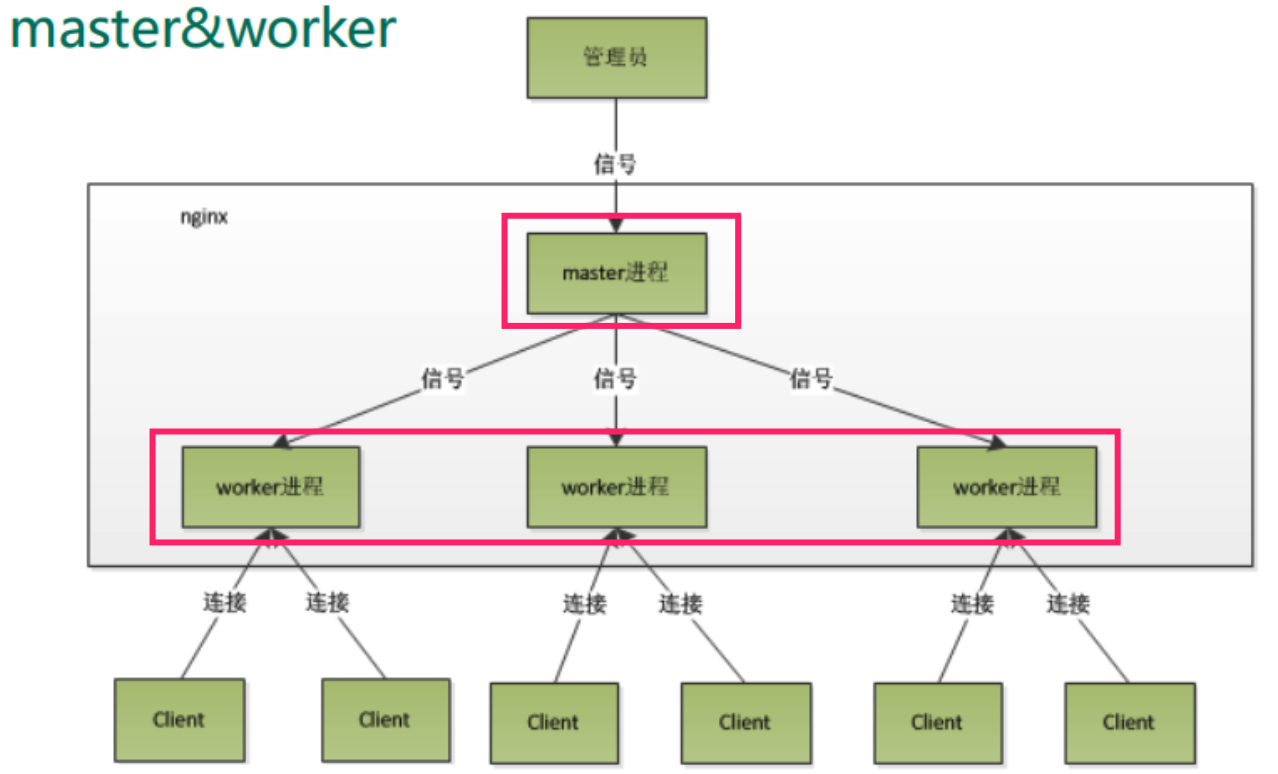
1. **http块**，是 Nginx 服务器配置中最频繁的部分，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这里，分为http全局块和server块

****

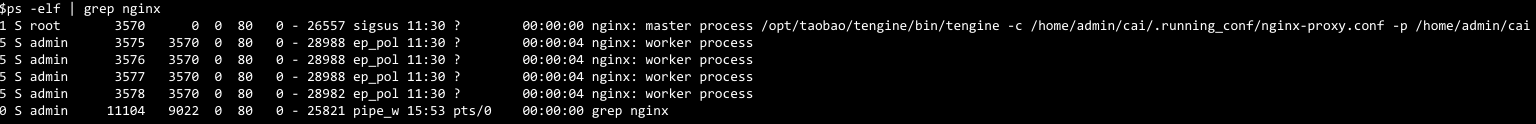
* 1. **http全局块**，http全局块配置的指令包括文件引入、MIME-TYPE 定义、日志自定义、连接超时时间、单链接请求数上限等
  2. **server块**，这块和虚拟主机有密切关系，虚拟主机从用户角度看，和一台独立的硬件主机是完全一样的，该技术的产生是为了节省互联网服务器硬件成本。每个 http 块可以包括多个server 块，而每个server 块就相当于一个虚拟主机。而每个server 块也分为全局server块，以及可以同时包含多个location块
     1. server全局块，最常见的配置是本虚拟机主机的监听配置和本虚拟主机的名称或 IP 配置
     2. 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

## nginx原理

1. master-worker机制



1. 一个master多个worker机制



* 1. 可以使用nginx -s reload热部署，利用nginx进行热部署操作
  2. 每个worker都是独立的进程，其中一个出现问题不会影响其它worker造成服务中断

1. 设置多少个worker合适

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

1. 设置worker数量worker\_processes

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

1. worker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
   1. 发送请求，占用了worker的几个连接数

2或者4个

* 1. Nginx有一个master，四个worker，每个worker支持最大的连接数1024，那么支持的最大并发数是多少
     1. 普通静态访问，最大并发数：worker\_processes \* worker\_connection / 2
     2. Http反向代理，最大并发数worker\_processes \* worker\_connection / 4