Nginx进阶篇

Nginx服务器基础配置实例

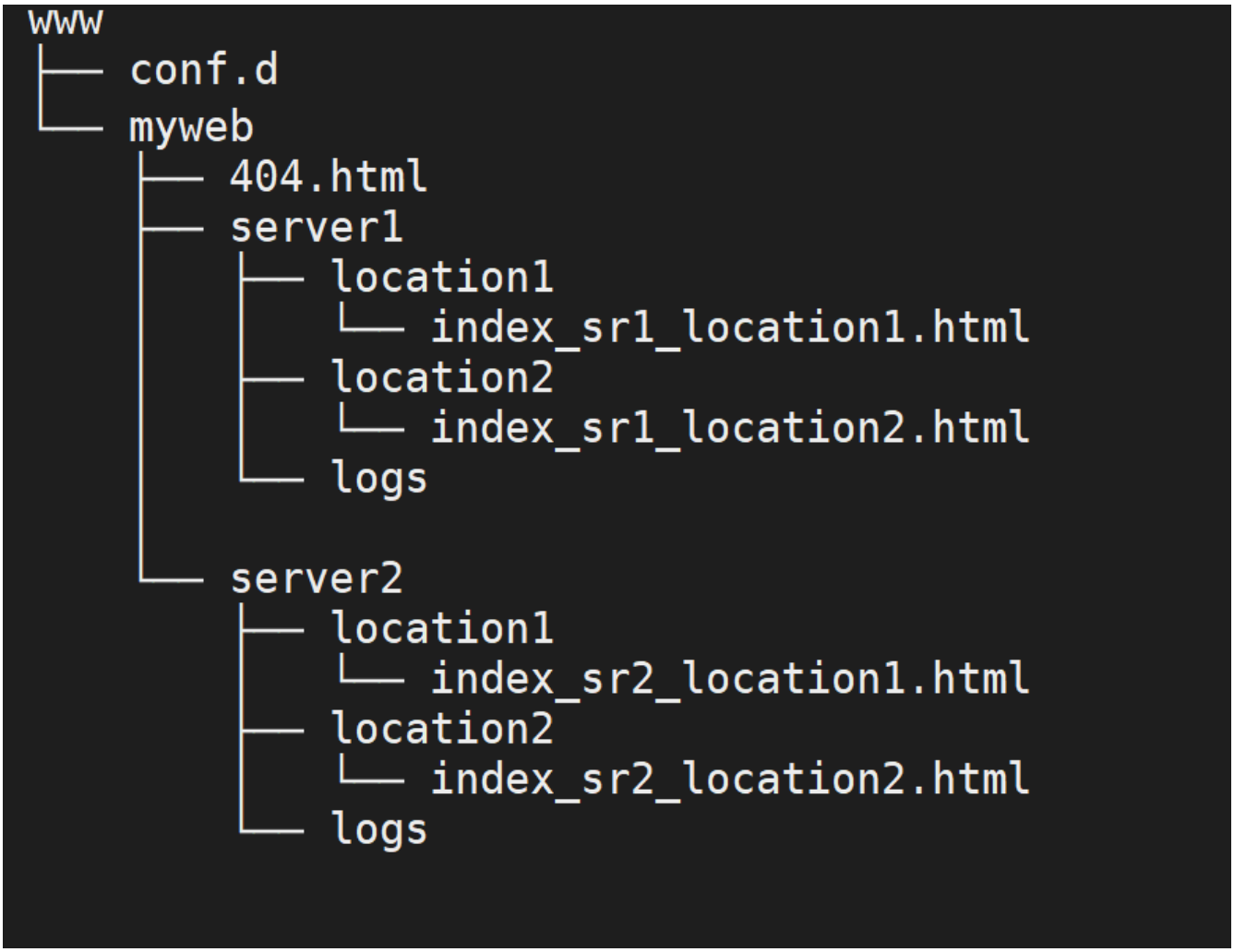
前面我们已经对Nginx服务器默认配置文件的结构和涉及的基本指令做了详细的阐述。通过这些指令的合理配置，我们就可以让一台Nginx服务器正常工作，并且提供基本的web服务器功能。

接下来我们将通过一个比较完整和最简单的基础配置实例，来巩固下前面所学习的指令及其配置。

需求如下:

```
(1) 有如下访问:
    http://192.168.200.133:8081/server1/location1
        访问的是: index_sr1_location1.html
    http://192.168.200.133:8081/server1/location2
        访问的是: index_sr1_location2.html
    http://192.168.200.133:8082/server2/location1
        访问的是: index_sr2_location1.html
    http://192.168.200.133:8082/server2/location2
        访问的是: index_sr2_location2.html
(2) 如果访问的资源不存在,
    返回自定义的404页面
(3) 将/server1和/server2的配置使用不同的配置文件分割
    将文件放到/home/www/conf.d目录下，然后使用include进行合并
(4) 为/server1和/server2各自创建一个访问日志文件
```

准备相关文件，目录如下：



配置的内容如下：

```
1
2  ##全局块 begin##
3  #配置允许运行Nginx工作进程的用户和用户组
4  user www;
5  #配置运行Nginx进程生成的worker进程数
6  worker_processes 2;
7  #配置Nginx服务器运行对错误日志存放的路径
8  error_log logs/error.log;
9  #配置Nginx服务器允许时记录Nginx的master进程的PID文件路径和名
   称
10 pid logs/nginx.pid;
```

```
#配置Nginx服务是否以守护进程方法启动
#daemon on;
##全局块 end##

##events块 begin##
events{
    #设置Nginx网络连接序列化
    accept_mutex on;
    #设置Nginx的worker进程是否可以同时接收多个请求
    multi_accept on;
    #设置Nginx的worker进程最大的连接数
    worker_connections 1024;
    #设置Nginx使用的事件驱动模型
    use epoll;
}
##events块 end##
##http块 start##
http{
    #定义MIME-Type
    include mime.types;
    default_type application/octet-stream;
    #配置允许使用sendfile方式运输
    sendfile on;
    #配置连接超时时间
    keepalive_timeout 65;
    #配置请求处理日志格式
    log_format server1 '===>server1 access log';
    log_format server2 '===>server2 access log';
    ##server块 开始##
    include /home/www/conf.d/*.conf;
    ##server块 结束##
}
```

```
43 ##http块 end##
```

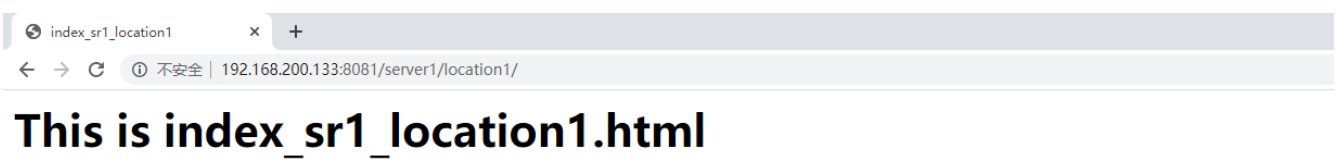
server1.conf

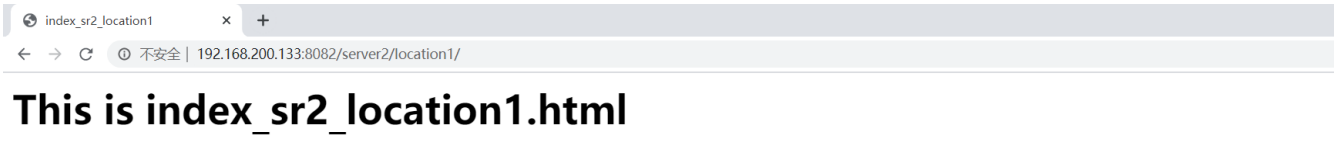
```
server{
        #配置监听端口和主机名称
        listen 8081;
        server_name localhost;
        #配置请求处理日志存放路径
        access_log
/home/www/myweb/server1/logs/access.log server1;
        #配置错误页面
        error_page 404 /404.html;
        #配置处理/server1/location1请求的location
        location /server1/location1{
            root /home/www/myweb;
            index index_sr1_location1.html;
        }
        #配置处理/server1/location2请求的location
        location /server1/location2{
            root /home/www/myweb;
            index index_sr1_location2.html;
        }
        #配置错误页面转向
        location = /404.html {
            root /home/www/myweb;
            index 404.html;
        }
}
```

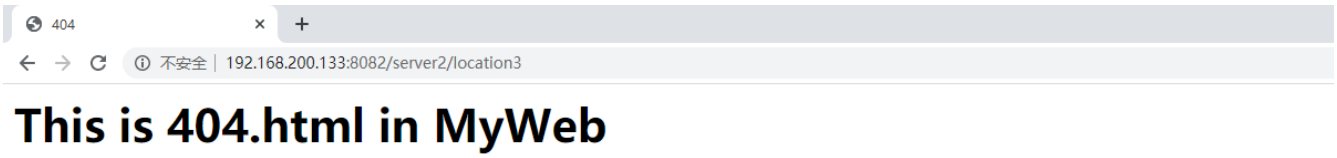
server2.conf

```
server{
        #配置监听端口和主机名称
        listen 8082;
        server_name localhost;
        #配置请求处理日志存放路径
        access_log
/home/www/myweb/server2/logs/access.log server2;
        #配置错误页面,对404.html做了定向配置
        error_page 404 /404.html;
        #配置处理/server1/location1请求的location
        location /server2/location1{
            root /home/www/myweb;
            index index_sr2_location1.html;
        }
        #配置处理/server2/location2请求的location
        location /server2/location2{
            root /home/www/myweb;
            index index_sr2_location2.html;
        }
        #配置错误页面转向
        location = /404.html {
            root /home/www/myweb;
            index 404.html;
        }
    }
```

访问测试：







Nginx服务操作的问题

经过前面的操作，我们会发现，如果想要启动、关闭或重新加载nginx配置文件，都需要先进入到nginx的安装目录的sbin目录，然后使用nginx的二级制可执行文件来操作，相对来说操作比较繁琐，这块该如何优化？另外如果我们想把Nginx设置成随着服务器启动就自动完成启动操作，又该如何来实现?这就需要用到接下来我们要讲解的两个知识点：

```
Nginx配置成系统服务
Nginx命令配置到系统环境
```

Nginx配置成系统服务

把Nginx应用服务设置成为系统服务，方便对Nginx服务的启动和停止等相关操作，具体实现步骤:

(1) 在`/usr/lib/systemd/system`目录下添加nginx.service,内容如下:

```
vim /usr/lib/systemd/system/nginx.service
```

```
[Unit]
Description=nginx web service
Documentation=http://nginx.org/en/docs/
After=network.target

[Service]
Type=forking
PIDFile=/usr/local/nginx/logs/nginx.pid
ExecStartPre=/usr/local/nginx/sbin/nginx -t -c
/usr/local/nginx/conf/nginx.conf
ExecStart=/usr/local/nginx/sbin/nginx
ExecReload=/usr/local/nginx/sbin/nginx -s reload
ExecStop=/usr/local/nginx/sbin/nginx -s stop
PrivateTmp=true

[Install]
WantedBy=default.target
```

(2)添加完成后如果权限有问题需要进行权限设置

```
chmod 755 /usr/lib/systemd/system/nginx.service
```

(3)使用系统命令来操作Nginx服务

```
启动: systemctl start nginx
停止: systemctl stop nginx
重启: systemctl restart nginx
重新加载配置文件: systemctl reload nginx
查看nginx状态: systemctl status nginx
开机启动: systemctl enable nginx
```

Nginx命令配置到系统环境

前面我们介绍过Nginx安装目录下的二级制可执行文件`nginx`的很多命令，要想使用这些命令前提是需要进入sbin目录下才能使用，很不方便，如何去优化，我们可以将该二进制可执行文件加入到系统的环境变量，这样的话在任何目录都可以使用nginx对应的相关命令。具体实现步骤如下:

演示可删除

```
/usr/local/nginx/sbin/nginx -V
cd /usr/local/nginx/sbin  nginx -V
如何优化? ? ?
```

(1)修改`/etc/profile`文件

```
vim /etc/profile
在最后一行添加
export PATH=$PATH:/usr/local/nginx/sbin
```

(2)使之立即生效

```
source /etc/profile
```

(3)执行nginx命令

```
nginx -V
```

Nginx静态资源部署

Nginx静态资源概述

上网去搜索访问资源对于我们来说并不陌生，通过浏览器发送一个HTTP请求实现从客户端发送请求到服务器端获取所需要内容后并把内容回显展示在页面的一个过程。这个时候，我们所请 求的内容就分为两种类型，一类是静态资源、一类是动态资源。 静态资源即指在服务器端真实存在并且能直接拿来展示的一些文件，比如常见的html页面、css文件、js文件、图 片、视频等资源； 动态资源即指在服务器端真实存在但是要想获取需要经过一定的业务逻辑处理，根据不同的条件展示在页面不同这 一部分内容，比如说报表数据展示、根据当前登录用户展示相关具体数据等资源；

Nginx处理静态资源的内容，我们需要考虑下面这几个问题：

```
（1）静态资源的配置指令
（2）静态资源的配置优化
（3）静态资源的压缩配置指令
（4）静态资源的缓存处理
（5）静态资源的访问控制，包括跨域问题和防盗链问题
```

Nginx静态资源的配置指令

listen指令

listen:用来配置监听端口。

语法	**listen address[:port] [default_server]...; listen port [default_server]...;**
默认值	listen *:80 | *:8000
位置	server

listen的设置比较灵活，我们通过几个例子来把常用的设置方式熟悉下：

```
listen 127.0.0.1:8000; // listen localhost:8000 监听指定的IP和端口
listen 127.0.0.1;    监听指定IP的所有端口
listen 8000;     监听指定端口上的连接
listen *:8000;   监听指定端口上的连接
```

default_server属性是标识符，用来将此虚拟主机设置成默认主机。所谓的默认主机指的是如果没有匹配到对应的address:port，则会默认执行的。如果不指定默认使用的是第一个server。

```
server{
    listen 8080;
    server_name 127.0.0.1;
    location /{
        root html;
        index index.html;
    }
}
server{
    listen 8080 default_server;
    server_name localhost;
    default_type text/plain;
    return 444 'This is a error request';

```

```
14 }
```

server_name指令

server_name：用来设置虚拟主机服务名称。

127.0.0.1 、 localhost 、域名[www.baidu.com | www.jd.com]

语法	**server_name name ...; name可以提供多个中间用空格分隔**
默认值	server_name "";
位置	server

关于server_name的配置方式有三种，分别是：

```
1 精确匹配
2 通配符匹配
3 正则表达式匹配
```

配置方式一：精确匹配

如：

```
server {
    listen 80;
    server_name www.itcast.cn www.itheima.cn;
    ...
}
```

补充小知识点:

```
hosts是一个没有扩展名的系统文件，可以用记事本等工具打开，其作用就是将一些常用的网址域名与其对应的IP地址建立一个关联“数据库”，当用户在浏览器中输入一个需要登录的网址时，系统会首先自动从hosts文件中寻找对应的IP地址，一旦找到，系统会立即打开对应网页，如果没有找到，则系统会再将网址提交DNS域名解析服务器进行IP地址的解析。
```

windows:C:\Windows\System32\drivers\etc

centos：/etc/hosts

因为域名是要收取一定的费用，所以我们可以使用修改hosts文件来制作一些虚拟域名来使用。需要修改 `/etc/hosts` 文件来添加

```
vim /etc/hosts
127.0.0.1 www.itcast.cn
127.0.0.1 www.itheima.cn
```

配置方式二:使用通配符配置

server_name中支持通配符"*",但需要注意的是通配符不能出现在域名的中间，只能出现在首段或尾段，如：

```
server {
    listen 80;
    server_name  *.itcast.cn    www.itheima.*;
    # www.itcast.cn abc.itcast.cn www.itheima.cn www.itheima.com
    ...
}
```

下面的配置就会报错

```
server {
    listen 80;
    server_name  www.*.cn www.itheima.c*
    ...
}
```

配置三:使用正则表达式配置

server_name中可以使用正则表达式，并且使用`~`作为正则表达式字符串的开始标记。

常见的正则表达式

代码	说明
^	匹配搜索字符串开始位置
$	匹配搜索字符串结束位置
.	匹配除换行符\n之外的任何单个字符
\	转义字符，将下一个字符标记为特殊字符
[xyz]	字符集，与任意一个指定字符匹配
[a-z]	字符范围，匹配指定范围内的任何字符
\w	与以下任意字符匹配 A-Z a-z 0-9 和下划线,等效于[A-Za-z0-9_]
\d	数字字符匹配，等效于[0-9]
{n}	正好匹配n次
{n,}	至少匹配n次
{n,m}	匹配至少n次至多m次
*	零次或多次，等效于{0,}
+	一次或多次，等效于{1,}
?	零次或一次，等效于{0,1}

配置如下：

```
server{
        listen 80;
        server_name ~^www\.(\w+)\.com$;
        default_type text/plain;
        return 200 $1  $2 ..;
}
注意 ~后面不能加空格，括号可以取值
```

匹配执行顺序

由于server_name指令支持通配符和正则表达式，因此在包含多个虚拟主机的配置文件中，可能会出现一个名称被多个虚拟主机的server_name匹配成功，当遇到这种情况，当前的请求交给谁来处理呢?

```
server{
    listen 80;
    server_name ~^www\.\w+\.com$;
    default_type text/plain;
    return 200 'regex_success';
}

server{
    listen 80;
    server_name www.itheima.*;
    default_type text/plain;
    return 200 'wildcard_after_success';
}

server{
    listen 80;
    server_name *.itheima.com;
```

```
    default_type text/plain;
    return 200 'wildcard_before_success';
}

server{
    listen 80;
    server_name www.itheima.com;
    default_type text/plain;
    return 200 'exact_success';
}

server{
    listen 80 default_server;
    server_name _;
    default_type text/plain;
    return 444 'default_server not found server';
}
```

结论:

```
exact_success
wildcard_before_success
wildcard_after_success
regex_success
default_server not found server!!
```

```
No1:准确匹配server_name

No2:通配符在开始时匹配server_name成功

No3:通配符在结束时匹配server_name成功

No4:正则表达式匹配server_name成功

No5:被默认的default_server处理，如果没有指定默认找第一个server
```

location指令

```
server{
    listen 80;
    server_name localhost;
    location / {

    }
    location /abc{

    }
    ...
}
```

location:用来设置请求的URI

语法	**location [= | ~ | ~* | ^~ |@] uri{...}**
默认值	—
位置	server,location

uri变量是待匹配的请求字符串，可以不包含正则表达式，也可以包含正则表达式，那么nginx服务器在搜索匹配location的时候，是先使用不包含正则表达式进行匹配，找到一个匹配度最高的一个，然后在通过包含正则表达式的进行匹配，如果能匹配到直接访问，匹配不到，就使用刚才匹配度最高的那个location来处理请求。

属性介绍:

不带符号，要求必须以指定模式开始

```
server {
    listen 80;
    server_name 127.0.0.1;
    location /abc{
        default_type text/plain;
        return 200 "access success";
    }
}
以下访问都是正确的
http://192.168.200.133/abc
http://192.168.200.133/abc?p1=TOM
http://192.168.200.133/abc/
http://192.168.200.133/abcdef
```

= : 用于不包含正则表达式的uri前，必须与指定的模式精确匹配

```
server {
    listen 80;
    server_name 127.0.0.1;
    location =/abc{
        default_type text/plain;
        return 200 "access success";
```

```
    }
}
可以匹配到
http://192.168.200.133/abc
http://192.168.200.133/abc?p1=TOM
匹配不到
http://192.168.200.133/abc/
http://192.168.200.133/abcdef
```

~： 用于表示当前uri中包含了正则表达式，并且区分大小写 ~*: 用于表示当前uri中包含了正则表达式，并且不区分大小写

换句话说，如果uri包含了正则表达式，需要用上述两个符合来标识

```
server {
    listen 80;
    server_name 127.0.0.1;
    location ~^/abc\w${
        default_type text/plain;
        return 200 "access success";
    }
}
server {
    listen 80;
    server_name 127.0.0.1;
    location ~*^/abc\w${
        default_type text/plain;
        return 200 "access success";
    }
}
```

^~: 用于不包含正则表达式的uri前，功能和不加符号的一致，唯一不同的是，如果模式匹配，那么就停止搜索其他模式了。

```
server {
    listen 80;
    server_name 127.0.0.1;
    location ^~/abc{
        default_type text/plain;
        return 200 "access success";
    }
}
```

设置请求资源的目录root / alias

root：设置请求的根目录

语法	**root path;**
默认值	root html;
位置	http、server、location

path为Nginx服务器接收到请求以后查找资源的根目录路径。

alias：用来更改location的URI

语法	**alias path;**
默认值	—
位置	location

path为修改后的根路径。

以上两个指令都可以来指定访问资源的路径，那么这两者之间的区别是什么?

举例说明:

（1）在`/usr/local/nginx/html`目录下创建一个 images目录,并在目录下放入一张图片`mv.png`图片

```
location /images {
    root /usr/local/nginx/html;
}
```

访问图片的路径为:

```
http://192.168.200.133/images/mv.png
```

（2）如果把root改为alias

```
location /images {
    alias /usr/local/nginx/html;
}
```

再次访问上述地址，页面会出现404的错误，查看错误日志会发现是因为地址不对，所以验证了:

```
root的处理结果是: root路径+location路径
/usr/local/nginx/html/images/mv.png
alias的处理结果是:使用alias路径替换location路径
/usr/local/nginx/html/images
```

需要在alias后面路径改为

```
location /images {
    alias /usr/local/nginx/html/images;
}
```

（3）如果location路径是以/结尾,则alias也必须是以/结尾，root没有要求

将上述配置修改为

```
location /images/ {
    alias /usr/local/nginx/html/images;
}
```

访问就会出问题，查看错误日志还是路径不对，所以需要把alias后面加上 /

小结:

```
root的处理结果是: root路径+location路径
alias的处理结果是:使用alias路径替换location路径
alias是一个目录别名的定义，root则是最上层目录的含义。
如果location路径是以/结尾,则alias也必须是以/结尾，root没有要求
```

index指令

index:设置网站的默认首页

语法	**index file ...;**
默认值	index index.html;
位置	http、server、location

index后面可以跟多个设置，如果访问的时候没有指定具体访问的资源，则会依次进行查找，找到第一个为止。

举例说明：

```
location / {
    root /usr/local/nginx/html;
    index index.html index.htm;
}
访问该location的时候，可以通过 http://ip:port/，地址后面如果不添加任何内容，则默认依次访问index.html和index.htm，找到第一个来进行返回
```

error_page指令

error_page:设置网站的错误页面

语法	error_page code ... [=[response]] uri;
默认值	—
位置	http、server、location……

当出现对应的响应code后，如何来处理。

举例说明：

（1）可以指定具体跳转的地址

```
server {
    error_page 404 http://www.itcast.cn;
}
```

（2）可以指定重定向地址

```
server{
    error_page 404 /50x.html;
    error_page 500 502 503 504 /50x.html;
    location =/50x.html{
        root html;
    }
}
```

(3) 使用location的@符合完成错误信息展示

```
server{
    error_page 404 @jump_to_error;
    location @jump_to_error {
        default_type text/plain;
        return 404 'Not Found Page...';
    }
}
```

可选项`=[response]`的作用是用来将相应代码更改为另外一个

```
server{
    error_page 404 =200 /50x.html;
    location =/50x.html{
        root html;
    }
}
这样的话，当返回404找不到对应的资源的时候，在浏览器上可以看到，最终返回的状态码是200，这块需要注意下，编写error_page后面的内容，404后面需要加空格，200前面不能加空格
```

静态资源优化配置语法

Nginx对静态资源如何进行优化配置。这里从三个属性配置进行优化：

```
sendfile on;
tcp_nopush on;
tcp_nodeplay on;
```

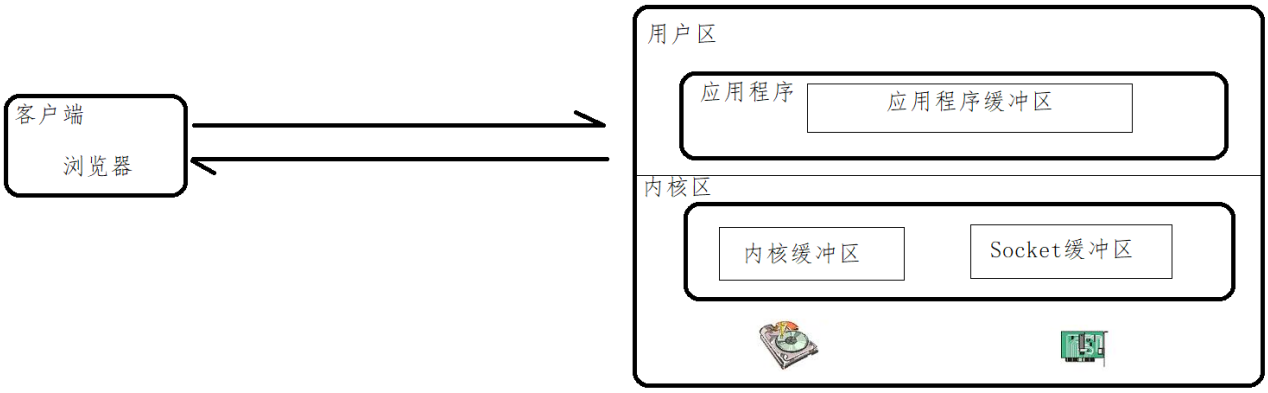
（1）sendfile，用来开启高效的文件传输模式。

语法	**sendfile on |off;**
默认值	sendfile off;
位置	http、server、location...

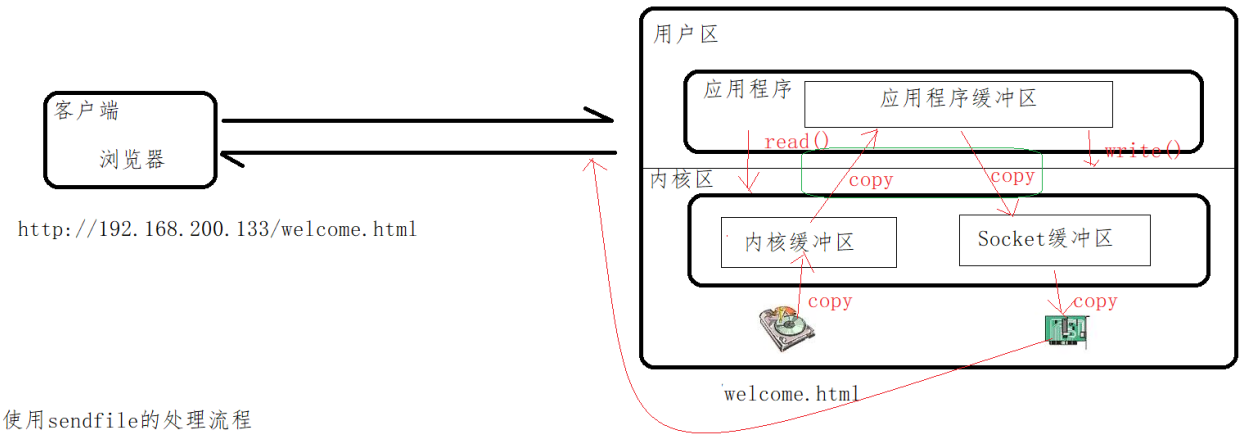
请求静态资源的过程：客户端通过网络接口向服务端发送请求，操作系统将这些客户端的请求传递给服务器端应用程序，服务器端应用程序会处理这些请求，请求处理完成以后，操作系统还需要将处理得到的结果通过网络适配器传递回去。

如：

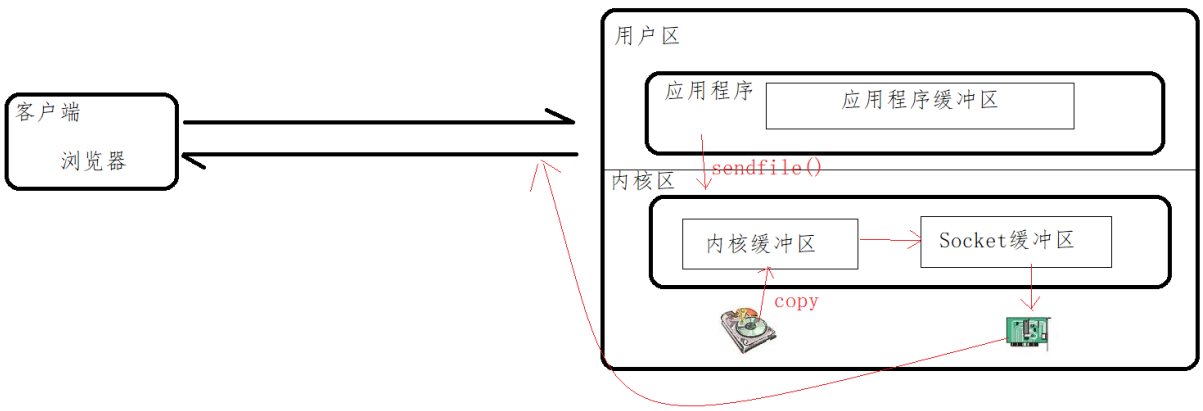
```
server {
    listen 80;
    server_name localhost;
    location / {
        root html;
        index index.html;
    }
}
在html目录下有一个welcome.html页面，访问地址
http://192.168.200.133/welcome.html
```



未使用sendfile的处理流程



使用sendfile的处理流程

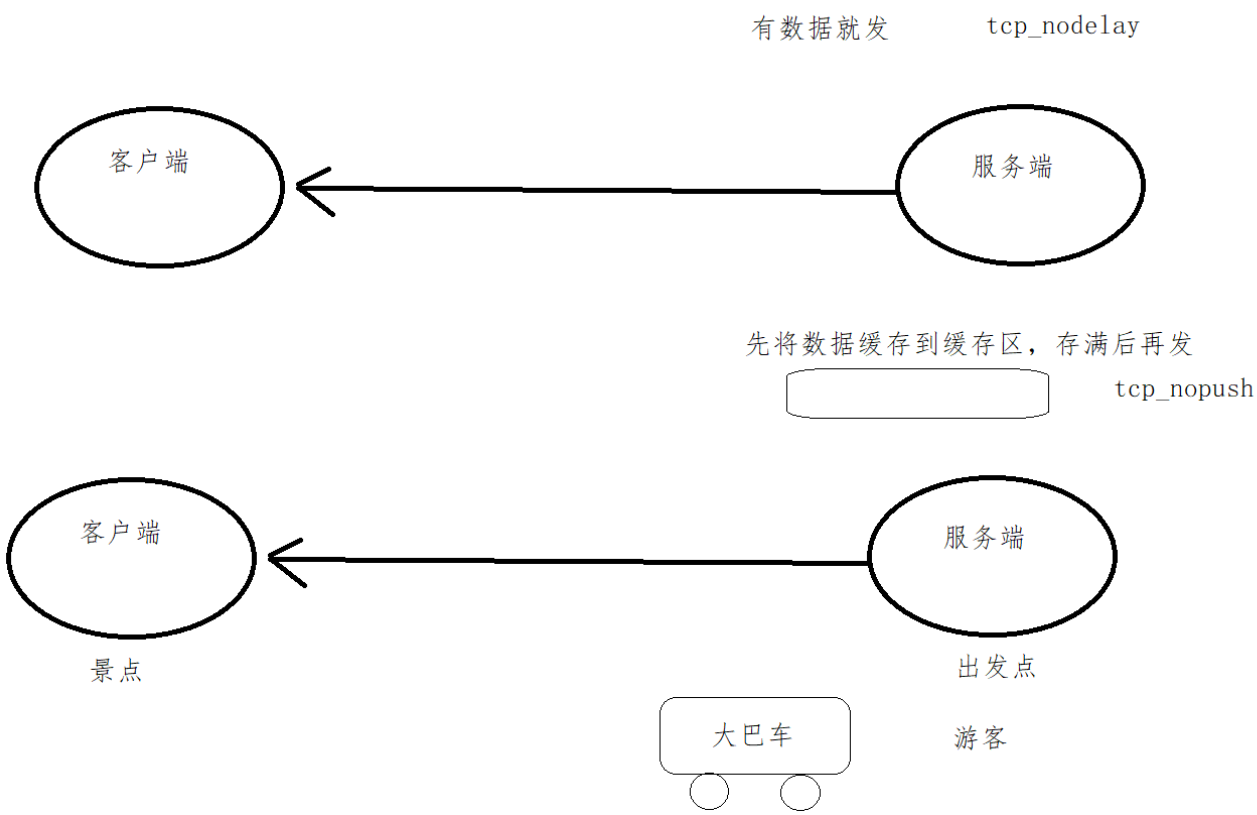


（2）tcp_nopush：该指令必须在sendfile打开的状态下才会生效，主要是用来提升网络包的传输'效率'

语法	tcp_nopush on|off;
默认值	tcp_nopush off;
位置	http、server、location

（3）tcp_nodelay：该指令必须在keep-alive连接开启的情况下才生效，来提高网络包传输的'实时性'

语法	tcp_nodelay on|off;
默认值	tcp_nodelay on;
位置	http、server、location



经过刚才的分析，"tcp_nopush"和”tcp_nodelay“看起来是"互斥的"，那么为什么要将这两个值都打开，这个大家需要知道的是在linux2.5.9以后的版本中两者是可以兼容的，三个指令都开启的好处是，sendfile可以开启高效的文件传输模式，tcp_nopush开启可以确保在发送到客户端之前数据包已经充分“填满”， 这大大减少了网络开销，并加快了文件发送的速度。 然后，当它到达最后一个可能因为没有“填满”而暂停的数据包时，Nginx会忽略tcp_nopush参数， 然后，tcp_nodelay强制套接字发送数据。由此可知，TCP_NOPUSH可以与TCP_NODELAY一起设置，它比单独配置TCP_NODELAY具有更强的性能。所以我们可以使用如下配置来优化Nginx静态资源的处理

```
sendfile on;
tcp_nopush on;
tcp_nodelay on;
```

Nginx静态资源压缩实战

经过上述内容的优化，我们再次思考一个问题，假如在满足上述优化的前提下，我们传送一个1M的数据和一个10M的数据那个效率高?，答案显而易见，传输内容小，速度就会快。那么问题又来了，同样的内容，如果把大小降下来，我们脑袋里面要蹦出一个词就是"压缩"，接下来，我们来学习Nginx的静态资源压缩模块。

在Nginx的配置文件中可以通过配置gzip来对静态资源进行压缩，相关的指令可以配置在http块、server块和location块中，Nginx可以通过

```
ngx_http_gzip_module模块
ngx_http_gzip_static_module模块
ngx_http_gunzip_module模块
```

对这些指令进行解析和处理。

接下来我们从以下内容进行学习

```
(1) Gzip各模块支持的配置指令
(2) Gzip压缩功能的配置
(3) Gzip和sendfile的冲突解决
(4) 浏览器不支持Gzip的解决方案
```

Gzip模块配置指令

接下来所学习的指令都来自ngx_http_gzip_module模块，该模块会在nginx安装的时候内置到nginx的安装环境中，也就是说我们可以直接使用这些指令。

1. gzip指令：该指令用于开启或者关闭gzip功能

语法	**gzip on|off;**
默认值	gzip off;
位置	http、server、location...

注意只有该指令为打开状态，下面的指令才有效果

```
http{
   gzip on;
}
```

2. gzip_types指令：该指令可以根据响应页的MIME类型选择性地开启Gzip压缩功能

语法	gzip_types mime-type ...;
默认值	gzip_types text/html;
位置	http、server、location

所选择的值可以从mime.types文件中进行查找，也可以使用"*"代表所有。

```
http{
    gzip_types application/javascript;
}
```

3. gzip_comp_level指令：该指令用于设置Gzip压缩程度，级别从1-9,1表示要是程度最低，要是效率最高，9刚好相反，压缩程度最高，但是效率最低最费时间。

语法	gzip_comp_level level;
默认值	gzip_comp_level 1;
位置	http、server、location

```
http{
    gzip_comp_level 6;
}
```

4. gzip_vary指令：该指令用于设置使用Gzip进行压缩发送是否携带“Vary:Accept-Encoding”头域的响应头部。主要是告诉接收方，所发送的数据经过了Gzip压缩处理

语法	gzip_vary on|off;
默认值	gzip_vary off;
位置	http、server、location

5. gzip_buffers指令：该指令用于处理请求压缩的缓冲区数量和大小。

语法	gzip_buffers number size;
默认值	gzip_buffers 32 4k|16 8k;
位置	http、server、location

其中number:指定Nginx服务器向系统申请缓存空间个数，size指的是每个缓存空间的大小。主要实现的是申请number个每个大小为size的内存空间。这个值的设定一般会和服务器的操作系统有关，所以建议此项不设置，使用默认值即可。

```
gzip_buffers 4 16K;    #缓存空间大小
```

6. gzip_disable指令：针对不同种类客户端发起的请求，可以选择性地开启和关闭Gzip功能。

语法	gzip_disable regex ...;
默认值	—
位置	http、server、location

regex:根据客户端的浏览器标志(user-agent)来设置，支持使用正则表达式。指定的浏览器标志不使用Gzip.该指令一般是用来排除一些明显不支持Gzip的浏览器。

```
gzip_disable "MSIE [1-6]\.";
```

7. gzip_http_version指令：针对不同的HTTP协议版本，可以选择性地开启和关闭Gzip功能。

语法	gzip_http_version 1.0|1.1;
默认值	gzip_http_version 1.1;
位置	http、server、location

该指令是指定使用Gzip的HTTP最低版本，该指令一般采用默认值即可。

8. gzip_min_length指令：该指令针对传输数据的大小，可以选择性地开启和关闭Gzip功能

语法	gzip_min_length length;
默认值	gzip_min_length 20;
位置	http、server、location

```
nignx计量大小的单位: bytes[字节] / kb[千字节] / M[兆]
例如: 1024 / 10k|K / 10m|M
```

Gzip压缩功能对大数据的压缩效果明显，但是如果要压缩的数据比较小的化，可能出现越压缩数据量越大的情况，因此我们需要根据响应内容的大小来决定是否使用Gzip功能，响应页面的大小可以通过头信息中的`Content-Length`来获取。但是如何使用了Chunk编码动态压缩，该指令将被忽略。建议设置为1K或以上。

9. gzip_proxied指令：该指令设置是否对服务端返回的结果进行Gzip压缩。

语法	**gzip_proxied off|expired|no-cache|no-store|private|no_last_modified|no_etag|auth|any;**
默认值	gzip_proxied off;
位置	http、server、location

off - 关闭Nginx服务器对后台服务器返回结果的Gzip压缩 expired - 启用压缩，如果header头中包含 "Expires" 头信息 no-cache - 启用压缩，如果header头中包含 "Cache-Control:no-cache" 头信息 no-store - 启用压缩，如果header头中包含 "Cache-Control:no-store" 头信息 private - 启用压缩，如果header头中包含 "Cache-Control:private" 头信息 no_last_modified - 启用压缩,如果header头中不包含 "Last-Modified" 头信息 no_etag - 启用压缩 ,如果header头中不包含 "ETag" 头信息 auth - 启用压缩 , 如果header头中包含 "Authorization" 头信息 any - 无条件启用压缩

Gzip压缩功能的实例配置

```
gzip on;                #开启gzip功能
gzip_types *;           #压缩源文件类型,根据具体的访问资源类型设定
gzip_comp_level 6;      #gzip压缩级别
gzip_min_length 1024; #进行压缩响应页面的最小长度,content-length
gzip_buffers 4 16K;     #缓存空间大小
gzip_http_version 1.1; #指定压缩响应所需要的最低HTTP请求版本
gzip_vary  on;          #往头信息中添加压缩标识
gzip_disable "MSIE [1-6]\."; #对IE6以下的版本都不进行压缩
gzip_proxied  off;  #nginx作为反向代理压缩服务端返回数据的条件
```

这些配置在很多地方可能都会用到，所以我们可以将这些内容抽取到一个配置文件中，然后通过include指令把配置文件再次加载到nginx.conf配置文件中，方法使用。

nginx_gzip.conf

```
gzip on;
gzip_types *;
gzip_comp_level 6;
gzip_min_length 1024;
gzip_buffers 4 16K;
gzip_http_version 1.1;
gzip_vary  on;
gzip_disable "MSIE [1-6]\.";
gzip_proxied  off;
```

nginx.conf

```
include nginx_gzip.conf
```

Gzip和sendfile共存问题

前面在讲解sendfile的时候，提到过，开启sendfile以后，在读取磁盘上的静态资源文件的时候，可以减少拷贝的次数，可以不经过用户进程将静态文件通过网络设备发送出去，但是Gzip要想对资源压缩，是需要经过用户进程进行操作的。所以如何解决两个设置的共存问题。

可以使用ngx_http_gzip_static_module模块的gzip_static指令来解决。

gzip_static指令

gzip_static: 检查与访问资源同名的.gz文件时，response中以gzip相关的header返回.gz文件的内容。

语法	**gzip_static on | off | always;**
默认值	gzip_static off;
位置	http、server、location

添加上述命令后，会报一个错误，`unknown directive "gzip_static"`主要的原因是Nginx默认是没有添加ngx_http_gzip_static_module模块。如何来添加?

添加模块到Nginx的实现步骤

(1)查询当前Nginx的配置参数

```
nginx -V
```

(2)将nginx安装目录下sbin目录中的nginx二进制文件进行更名

```
cd /usr/local/nginx/sbin
mv nginx nginxold
```

(3) 进入Nginx的安装目录

```
cd /root/nginx/core/nginx-1.16.1
```

(4)执行make clean清空之前编译的内容

```
make clean
```

(5)使用configure来配置参数

```
./configure --with-http_gzip_static_module
```

(6)使用make命令进行编译

```
make
```

(7) 将objs目录下的nginx二进制执行文件移动到nginx安装目录下的sbin目录中

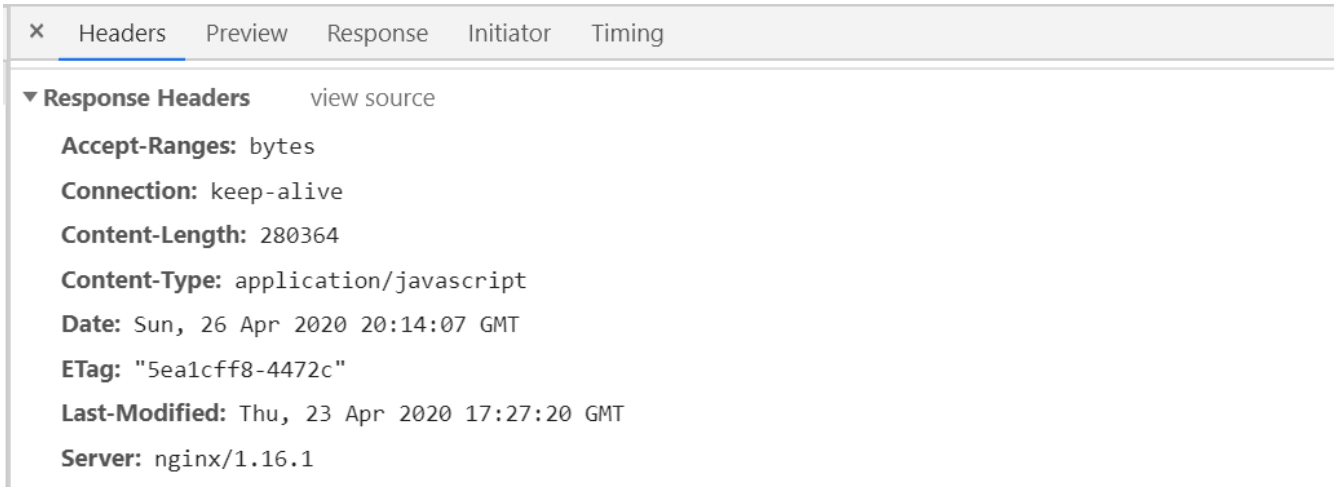
```
mv objs/nginx /usr/local/nginx/sbin
```

(8)执行更新命令

```
make upgrade
```

gzip_static测试使用

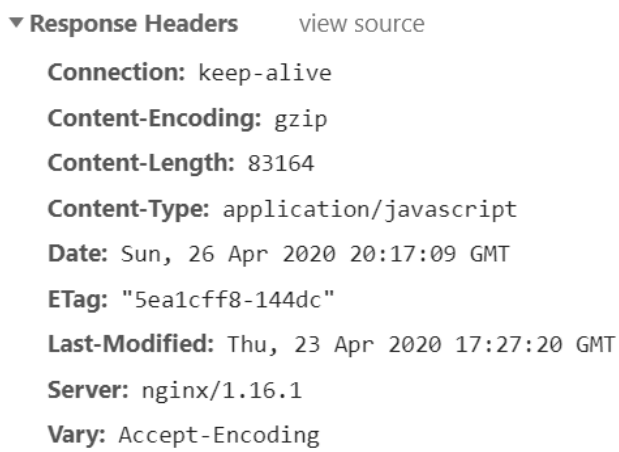
(1)直接访问`http://192.168.200.133/jquery.js`



(2)使用gzip命令进行压缩

```
cd /usr/local/nginx/html
gzip jquery.js
```

(3)再次访问`http://192.168.200.133/jquery.js`



静态资源的缓存处理

什么是缓存

缓存（cache），原始意义是指访问速度比一般随机存取存储器（RAM）快的一种高速存储器，通常它不像系统主存那样使用DRAM技术，而使用昂贵但较快速的SRAM技术。缓存的设置是所有现代计算机系统发挥高性能的重要因素之一。

什么是web缓存

Web缓存是指一个Web资源（如html页面，图片，js，数据等）存在于Web服务器和客户端（浏览器）之间的副本。缓存会根据进来的请求保存输出内容的副本；当下一个请求来到的时候，如果是相同的URL，缓存会根据缓存机制决定是直接使用副本响应访问请求，还是向源服务器再次发送请求。比较常见的就是浏览器会缓存访问过网站的网页，当再次访问这个URL地址的时候，如果网页没有更新，就不会再次下载网页，而是直接使用本地缓存的网页。只有当网站明确标识资源已经更新，浏览器才会再次下载网页

web缓存的种类

```
客户端缓存
    浏览器缓存
服务端缓存
    Nginx / Redis / Memcached等
```

浏览器缓存

是为了节约网络的资源加速浏览，浏览器在用户磁盘上对最近请求过的文档进行存储，当访问者再次请求这个页面时，浏览器就可以从本地磁盘显示文档，这样就可以加速页面的阅览.

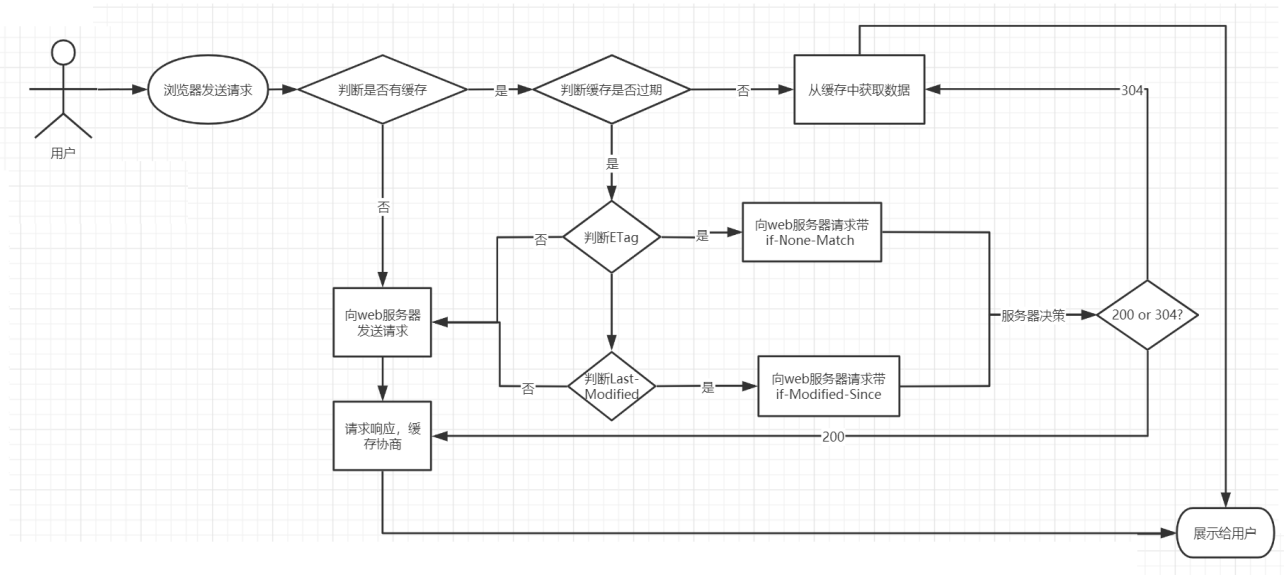
为什么要用浏览器缓存

```
成本最低的一种缓存实现
减少网络带宽消耗
降低服务器压力
减少网络延迟，加快页面打开速度
```

浏览器缓存的执行流程

HTTP协议中和页面缓存相关的字段，我们先来认识下：

header	说明
Expires	缓存过期的日期和时间
Cache-Control	设置和缓存相关的配置信息
Last-Modified	请求资源最后修改时间
ETag	请求变量的实体标签的当前值，比如文件的MD5值



（1）用户首次通过浏览器发送请求到服务端获取数据，客户端是没有对应的缓存，所以需要发送request请求来获取数据；

（2）服务端接收到请求后，获取服务端的数据及服务端缓存的允许后，返回200的成功状态码并且在响应头上附上对应资源以及缓存信息；

（3）当用户再次访问相同资源的时候，客户端会在浏览器的缓存目录中查找是否存在响应的缓存文件

（4）如果没有找到对应的缓存文件，则走(2)步

（5）如果有缓存文件，接下来对缓存文件是否过期进行判断，过期的判断标准是(Expires),

（6）如果没有过期，则直接从本地缓存中返回数据进行展示

（7）如果Expires过期，接下来需要判断缓存文件是否发生过变化

（8）判断的标准有两个，一个是ETag(Entity Tag),一个是Last-Modified

（9）判断结果是未发生变化，则服务端返回304，直接从缓存文件中获取数据

（10）如果判断是发生了变化，重新从服务端获取数据，并根据缓存协商(服务端所设置的是否需要进行缓存数据的设置)来进行数据缓存。

浏览器缓存相关指令

Nginx需要进行缓存相关设置，就需要用到如下的指令

expires指令

expires:该指令用来控制页面缓存的作用。可以通过该指令控制HTTP应答中的“Expires"和”Cache-Control"

语法	**expires [modified] time** **expires epoch|max|off;**
默认值	expires off;
位置	http、server、location

time:可以整数也可以是负数，指定过期时间，如果是负数，Cache-Control则为no-cache,如果为整数或0，则Cache-Control的值为max-age=time;

epoch: 指定Expires的值为'1 January,1970,00:00:01 GMT'(1970-01-01 00:00:00)，Cache-Control的值no-cache

max:指定Expires的值为'31 December2037 23:59:59GMT' (2037-12-31 23:59:59)，Cache-Control的值为10年

off:默认不缓存。

add_header指令

add_header指令是用来添加指定的响应头和响应值。

语法	**add_header name value [always];**
默认值	—
位置	http、server、location...

Cache-Control作为响应头信息，可以设置如下值：

缓存响应指令：

```
Cache-control: must-revalidate
Cache-control: no-cache
Cache-control: no-store
Cache-control: no-transform
Cache-control: public
Cache-control: private
Cache-control: proxy-revalidate
Cache-Control: max-age=<seconds>
Cache-control: s-maxage=<seconds>
```

指令	说明
must-revalidate	可缓存但必须再向源服务器进行确认
no-cache	缓存前必须确认其有效性
no-store	不缓存请求或响应的任何内容
no-transform	代理不可更改媒体类型
public	可向任意方提供响应的缓存
private	仅向特定用户返回响应
proxy-revalidate	要求中间缓存服务器对缓存的响应有效性再进行确认
max-age=<秒>	响应最大Age值
s-maxage=<秒>	公共缓存服务器响应的最大Age值

max-age=[秒]:

Nginx的跨域问题解决

这块内容，我们主要从以下方面进行解决：

```
什么情况下会出现跨域问题?
实例演示跨域问题
具体的解决方案是什么?
```

同源策略

浏览器的同源策略：是一种约定，是浏览器最核心也是最基本的安全功能，如果浏览器少了同源策略，则浏览器的正常功能可能都会受到影响。

同源: 协议、域名(IP)、端口相同即为同源

```
http://192.168.200.131/user/1
https://192.168.200.131/user/1
不

http://192.168.200.131/user/1
http://192.168.200.132/user/1
不

http://192.168.200.131/user/1
http://192.168.200.131:8080/user/1
不

http://www.nginx.com/user/1
http://www.nginx.org/user/1
不

```

```
http://192.168.200.131/user/1
http://192.168.200.131:8080/user/1
不

http://www.nginx.org:80/user/1
http://www.nginx.org/user/1
满足
```

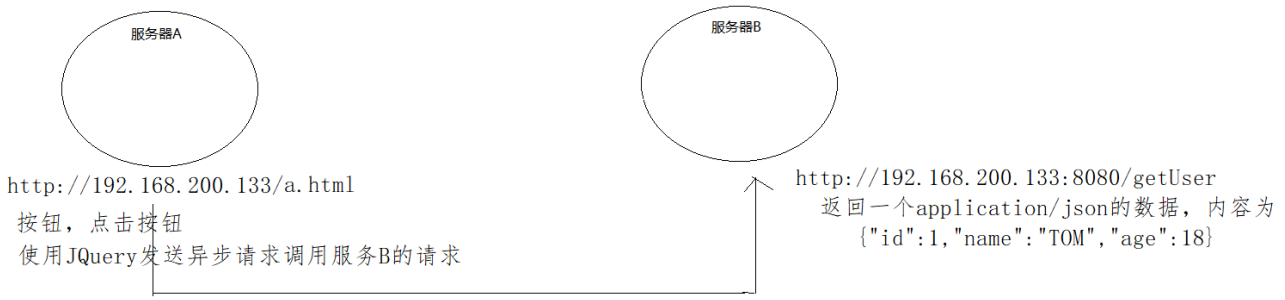
跨域问题

简单描述下:

```
有两台服务器分别为A,B,如果从服务器A的页面发送异步请求到服务器B获取数据，如果服务器A和服务器B不满足同源策略，则就会出现跨域问题。
```

跨域问题的案例演示

出现跨域问题会有什么效果?,接下来通过一个需求来给大家演示下:



（1）nginx的html目录下新建一个a.html

```
<html>
  <head>
        <meta charset="utf-8">
        <title>跨域问题演示</title>
        <script src="jquery.js"></script>
```

```
        <script>
            $(function(){
                $("#btn").click(function(){

 $.get('http://192.168.200.133:8080/getUser',function(data){

 alert(JSON.stringify(data));
                        });
                });
            });
        </script>
  </head>
  <body>
        <input type="button" value="获取数据" id="btn"/>
  </body>
</html>

```

(2) 在nginx.conf配置如下内容

```
server{
        listen  8080;
        server_name localhost;
        location /getUser{
                default_type application/json;
                return 200 '{"id":1,"name":"TOM","age":18}';
        }
}
server{

```

```
    listen  80;
    server_name localhost;
    location /{
        root html;
        index index.html;
    }
}
```

(3)通过浏览器访问测试

解决方案

使用add_header指令，该指令可以用来添加一些头信息

语法	add_header name value...
默认值	—
位置	http、server、location

此处用来解决跨域问题，需要添加两个头信息，一个是`Access-Control-Allow-Origin`,`Access-Control-Allow-Methods`

Access-Control-Allow-Origin: 直译过来是允许跨域访问的源地址信息，可以配置多个(多个用逗号分隔)，也可以使用`*`代表所有源

Access-Control-Allow-Methods:直译过来是允许跨域访问的请求方式，值可以为 GET POST PUT DELETE...,可以全部设置，也可以根据需要设置，多个用逗号分隔

具体配置方式

```
location /getUser{
    add_header Access-Control-Allow-Origin *;
    add_header Access-Control-Allow-Methods GET,POST,PUT,DELETE;
    default_type application/json;
    return 200 '{"id":1,"name":"TOM","age":18}';
}
```

静态资源防盗链

什么是资源盗链

资源盗链指的是此内容不在自己服务器上，而是通过技术手段，绕过别人的限制将别人的内容放到自己页面上最终展示给用户。以此来盗取大网站的空间和流量。简而言之就是用别人的东西成就自己的网站。

效果演示

京东:https://img14.360buyimg.com/n7/jfs/t1/101062/37/2153/254169/5dcbd410E6d10ba22/4ddbd212be225fcd.jpg

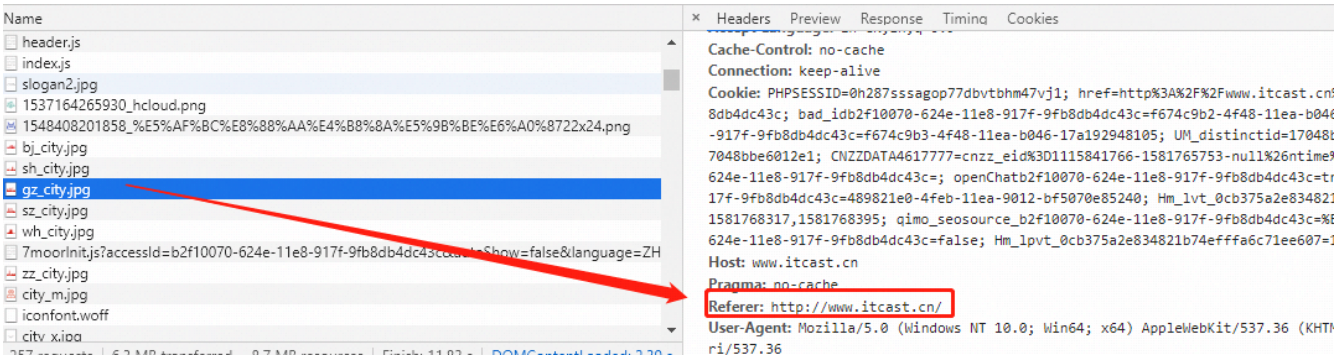
百度:https://pics7.baidu.com/feed/cf1b9d16fdfaaf516f7e2011a7cda1e8f11f7a1a.jpeg?token=551979a23a0995e5e5279b8fa1a48b34&s=BD385394D2E963072FD48543030030BB

我们自己准备一个页面，在页面上引入这两个图片查看效果

从上面的效果，可以看出来，下面的图片地址添加了防止盗链的功能，京东这边我们可以直接使用其图片。

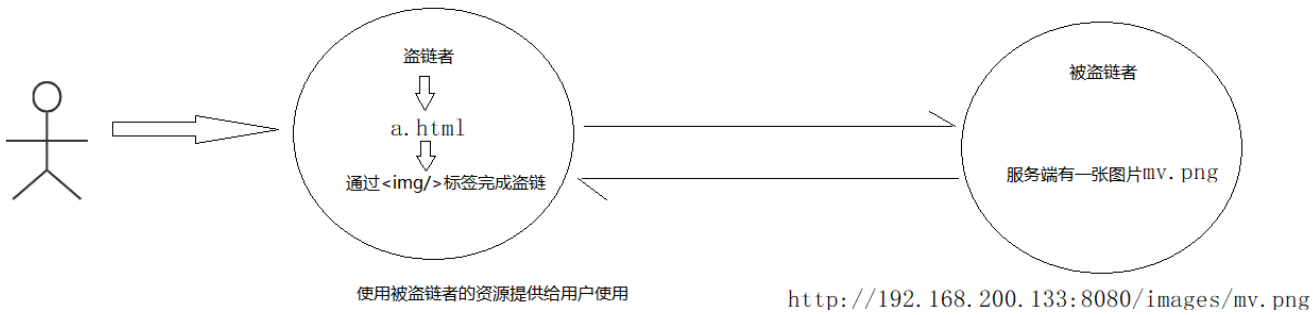
Nginx防盗链的实现原理：

了解防盗链的原理之前，我们得先学习一个HTTP的头信息Referer,当浏览器向web服务器发送请求的时候，一般都会带上Referer,来告诉浏览器该网页是从哪个页面链接过来的。



后台服务器可以根据获取到的这个Referer信息来判断是否为自己信任的网站地址，如果是则放行继续访问，如果不是则可以返回403(服务端拒绝访问)的状态信息。

在本地模拟上述的服务器效果：



Nginx防盗链的具体实现:

valid_referers:nginx会通就过查看referer自动和valid_referers后面的内容进行匹配，如果匹配到了就将$invalid_referer变量置0，如果没有匹配到，则将$invalid_referer变量置为1，匹配的过程中不区分大小写。

语法	**valid_referers none|blocked|server_names|string...**
默认值	—
位置	server、location

none: 如果Header中的Referer为空，允许访问

blocked:在Header中的Referer不为空，但是该值被防火墙或代理进行伪装过，如不带"http://" 、"https://"等协议头的资源允许访问。

server_names:指定具体的域名或者IP

string: 可以支持正则表达式和*的字符串。如果是正则表达式，需要以~开头表示，例如

```
location ~*\.(png|jpg|gif){
            valid_referers none blocked www.baidu.com
192.168.200.222 *.example.com example.*
www.example.org  ~\.google\.;
            if ($invalid_referer){
                 return 403;
            }
            root /usr/local/nginx/html;

}
```

遇到的问题:图片有很多，该如何批量进行防盗链?

针对目录进行防盗链

配置如下:

```
location /images {
            valid_referers none blocked www.baidu.com
192.168.200.222 *.example.com example.*
www.example.org  ~\.google\.;
            if ($invalid_referer){
                 return 403;
            }
            root /usr/local/nginx/html;

}
```

这样我们可以对一个目录下的所有资源进行翻到了操作。

遇到的问题：Referer的限制比较粗，比如随意加一个Referer，上面的方式是无法进行限制的。那么这个问题改如何解决?

此处我们需要用到Nginx的第三方模块`ngx_http_accesskey_module`，第三方模块如何实现盗链，如果在Nginx中使用第三方模块的功能，这些我们在后面的Nginx的模块篇再进行详细的讲解。

Rewrite功能配置

Rewrite是Nginx服务器提供的一个重要基本功能，是Web服务器产品中几乎必备的功能。主要的作用是用来实现URL的重写。

注意:Nginx服务器的Rewrite功能的实现依赖于PCRE的支持，因此在编译安装Nginx服务器之前，需要安装PCRE库。Nginx使用的是ngx_http_rewrite_module模块来解析和处理Rewrite功能的相关配置。

"地址重写"与"地址转发"

重写和转发的区别:

```
地址重写浏览器地址会发生变化而地址转发则不变
一次地址重写会产生两次请求而一次地址转发只会产生一次请求
地址重写到的页面必须是一个完整的路径而地址转发则不需要
地址重写因为是两次请求所以request范围内属性不能传递给新页面而地址转发因为是一次请求所以可以传递值
地址转发速度快于地址重写
```

Rewrite规则

set指令

该指令用来设置一个新的变量。

语法	set $variable value;
默认值	—
位置	server、location、if

variable:变量的名称，该变量名称要用"$"作为变量的第一个字符，且不能与Nginx服务器预设的全局变量同名。

value:变量的值，可以是字符串、其他变量或者变量的组合等。

Rewrite常用全局变量

变量	说明
$args	变量中存放了请求URL中的请求指令。比如http://192.168.200.133:8080?arg1=value1&arg2=value2中的"arg1=value1&arg2=value2"，功能和$query_string一样
$http_user_agent	变量存储的是用户访问服务的代理信息(如果通过浏览器访问，记录的是浏览器的相关版本信息)
$host	变量存储的是访问服务器的server_name值
$document_uri	变量存储的是当前访问地址的URI。比如http://192.168.200.133/server?id=10&name=zhangsan中的"/server"，功能和$uri一样
$document_root	变量存储的是当前请求对应location的root值，如果未设置，默认指向Nginx自带html目录所在位置
$content_length	变量存储的是请求头中的Content-Length的值
$content_type	变量存储的是请求头中的Content-Type的值
$http_cookie	变量存储的是客户端的cookie信息，可以通过add_header Set-Cookie 'cookieName=cookieValue'来添加cookie数据

变量	说明
$limit_rate	变量中存储的是Nginx服务器对网络连接速率的限制，也就是Nginx配置中对limit_rate指令设置的值，默认是0，不限制。
$remote_addr	变量中存储的是客户端的IP地址
$remote_port	变量中存储了客户端与服务端建立连接的端口号
$remote_user	变量中存储了客户端的用户名，需要有认证模块才能获取
$scheme	变量中存储了访问协议
$server_addr	变量中存储了服务端的地址
$server_name	变量中存储了客户端请求到达的服务器的名称
$server_port	变量中存储了客户端请求到达服务器的端口号
$server_protocol	变量中存储了客户端请求协议的版本，比如"HTTP/1.1"
$request_body_file	变量中存储了发给后端服务器的本地文件资源的名称
$request_method	变量中存储了客户端的请求方式，比如"GET","POST"等
$request_filename	变量中存储了当前请求的资源文件的路径名

变量	说明
$request_uri	变量中存储了当前请求的URI，并且携带请求参数，比如http://192.168.200.133/server?id=10&name=zhangsan中的"/server?id=10&name=zhangsan"

if指令

该指令用来支持条件判断，并根据条件判断结果选择不同的Nginx配置。

语法	if (condition){...}
默认值	—
位置	server、location

condition为判定条件，可以支持以下写法：

1. 变量名。如果变量名对应的值为空或者是0，if都判断为false,其他条件为true。

```
if ($param){

}
```

```
2. 使用"="和"!="比较变量和字符串是否相等，满足条件为true，不满足为false
```

```
if ($request_method = POST){
    return 405;
}
```

注意：此处和Java不太一样的地方是字符串不需要添加引号。

3. 使用正则表达式对变量进行匹配，匹配成功返回true，否则返回false。变量与正则表达式之间使用"~","~*","!~","!~*"来连接。

   "~"代表匹配正则表达式过程中区分大小写，

   "~*"代表匹配正则表达式过程中不区分大小写

   "!~"和"!~*"刚好和上面取相反值，如果匹配上返回false,匹配不上返回true

```
if ($http_user_agent ~ MSIE){
    #$http_user_agent的值中是否包含MSIE字符串，如果包含返回true
 }
```

注意：正则表达式字符串一般不需要加引号，但是如果字符串中包含"}"或者是";"等字符时，就需要把引号加上。

4. 判断请求的文件是否存在使用"-f"和"!-f",

   当使用"-f"时，如果请求的文件存在返回true，不存在返回false。

   当使用"!f"时，如果请求文件不存在，但该文件所在目录存在返回true,文件和目录都不存在返回false,如果文件存在返回false

```
if (-f $request_filename){
    #判断请求的文件是否存在
}
if (!-f $request_filename){
    #判断请求的文件是否不存在
}
```

5. 判断请求的目录是否存在使用"-d"和"!-d",

   当使用"-d"时，如果请求的目录存在，if返回true，如果目录不存在则返回false

   当使用"!-d"时，如果请求的目录不存在但该目录的上级目录存在则返回true，该目录和它上级目录都不存在则返回false,如果请求目录存在也返回false.

6. 判断请求的目录或者文件是否存在使用"-e"和"!-e"

   当使用"-e",如果请求的目录或者文件存在时，if返回true,否则返回false.

   当使用"!-e",如果请求的文件和文件所在路径上的目录都不存在返回true,否则返回false

7. 判断请求的文件是否可执行使用"-x"和"!-x"

   当使用"-x",如果请求的文件可执行，if返回true,否则返回false

   当使用"!-x",如果请求文件不可执行，返回true,否则返回false

break指令

该指令用于中断当前相同作用域中的其他Nginx配置。与该指令处于同一作用域的Nginx配置中，位于它前面的指令配置生效，位于后面的指令配置无效。

语法	**break;**
默认值	—
位置	server、location、if

例子:

```
location /{
    if ($param){
        set $id $1;
        break;
        limit_rate 10k;
    }
}
```

return指令

该指令用于完成对请求的处理，直接向客户端返回响应状态代码。在return后的所有Nginx配置都是无效的。

语法	**return code [text];** **return code URL;** **return URL;**
默认值	—
位置	server、location、if

code:为返回给客户端的HTTP状态代理。可以返回的状态代码为0~999的任意HTTP状态代理

text:为返回给客户端的响应体内容，支持变量的使用

URL:为返回给客户端的URL地址

rewrite指令

该指令通过正则表达式的使用来改变URI。可以同时存在一个或者多个指令，按照顺序依次对URL进行匹配和处理。

URL和URI的区别：

```
URI:统一资源标识符
URL:统一资源定位符
```

语法	rewrite regex replacement [flag];
默认值	—
位置	server、location、if

regex:用来匹配URI的正则表达式

replacement:匹配成功后，用于替换URI中被截取内容的字符串。如果该字符串是以"http://"或者"https://"开头的，则不会继续向下对URI进行其他处理，而是直接返回重写后的URI给客户端。

flag:用来设置rewrite对URI的处理行为，可选值有如下：

- last:
- break
- redirect
- permanent

rewrite_log指令

该指令配置是否开启URL重写日志的输出功能。

语法	rewrite_log on|off;
默认值	rewrite_log off;
位置	http、server、location、if

开启后，URL重写的相关日志将以notice级别输出到error_log指令配置的日志文件汇总。

Rewrite的案例

域名跳转

》问题分析

先来看一个效果，如果我们想访问京东网站，大家都知道我们可以输入`www.jd.com`,但是同样的我们也可以输入`www.360buy.com`同样也都能访问到京东网站。这个其实是因为京东刚开始的时候域名就是www.360buy.com，后面由于各种原因把自己的域名换成了www.jd.com, 虽然说域名变量，但是对于以前只记住了www.360buy.com的用户来说，我们如何把这部分用户也迁移到我们新域名的访问上来，针对于这个问题，我们就可以使用Nginx中Rewrite的域名跳转来解决。

》环境准备

- 准备两个域名 www.360buy.com | www.jd.com

```
vim /etc/hosts
```

```
192.168.200.133 www.360buy.com
192.168.200.133 www.jd.com
```

- 在/usr/local/nginx/html/hm目录下创建一个访问页面

```
<html>
    <title></title>
    <body>
        <h1>欢迎来到我们的网站</h1>
    </body>
</html>
```

- 通过Nginx实现当访问www.访问到系统的首页

```
server {
    listen 80;
    server_name www.hm.com;
    location /{
        root /usr/local/nginx/html/hm;
        index index.html;
    }
}
```

》通过Rewrite完成将www.360buy.com的请求跳转到www.jd.com

```
server {
    listen 80;
    server_name www.360buy.com;
    rewrite ^/ http://www.jd.com permanent;
}
```

问题描述:如何在域名跳转的过程中携带请求的URI?

修改配置信息

```
server {
    listen 80;
    server_name www.itheima.com;
    rewrite ^(.*) http://www.hm.com$1 permanent;
}
```

问题描述:我们除了上述说的www.jd.com 、www.360buy.com其实还有我们也可以通过www.jingdong.com来访问，那么如何通过Rewrite来实现多个域名的跳转?

添加域名

```
vim /etc/hosts
192.168.200.133 www.jingdong.com
```

修改配置信息

```
server{
    listen 80;
    server_name www.360buy.com www.jingdong.com;
    rewrite ^(.*) http://www.jd.com$1 permanent;
}
```

域名镜像

上述案例中，将www.360buy.com 和 www.jingdong.com都能跳转到www.jd.com，那么www.jd.com我们就可以把它起名叫主域名，其他两个就是我们所说的镜像域名，当然如果我们不想把整个网站做镜像，只想为其中某一个子目录下的资源做镜像，我们可以在location块中配置rewrite功能，比如:

```
server {
    listen 80;
    server_name rewrite.myweb.com;
    location ^~ /source1{
        rewrite ^/resource1(.*)
http://rewrite.myweb.com/web$1 last;
    }
    location ^~ /source2{
        rewrite ^/resource2(.*)
http://rewrite.myweb.com/web$1 last;
    }
}
```

独立域名

一个完整的项目包含多个模块，比如购物网站有商品商品搜索模块、商品详情模块已经购物车模块等，那么我们如何为每一个模块设置独立的域名。

需求:

```
http://search.hm.com  访问商品搜索模块
http://item.hm.com    访问商品详情模块
http://cart.hm.com    访问商品购物车模块
```

```
server{
    listen 80;
    server_name search.hm.com;
    rewrite ^(.*) http://www.hm.com/bbs$1 last;
}
server{
    listen 81;
```

```
    server_name item.hm.com;
    rewrite ^(.*) http://www.hm.com/item$1 last;
}
server{
    listen 82;
    server_name cart.hm.com;
    rewrite ^(.*) http://www.hm.com/cart$1 last;
}
```

目录自动添加"/"

问题描述

通过一个例子来演示下问题:

```
server {
    listen  80;
    server_name localhost;
    location / {
        root html;
        index index.html;
    }
}

```

要想访问上述资源，很简单，只需要通过http://192.168.200.133直接就能访问，地址后面不需要加/,但是如果将上述的配置修改为如下内容:

```
server {
    listen  80;
    server_name localhost;
    location /hm {
        root html;
        index index.html;
    }
}
```

这个时候，要想访问上述资源，按照上述的访问方式，我们可以通过http://192.168.200.133/hm/来访问,但是如果地址后面不加斜杠，页面就会出问题。如果不加斜杠，Nginx服务器内部会自动做一个301的重定向，重定向的地址会有一个指令叫server_name_in_redirect on|off;来决定重定向的地址：

```
如果该指令为on
    重定向的地址为:  http://server_name/目录名/;
如果该指令为off
    重定向的地址为:  http://原URL中的域名/目录名/;
```

所以就拿刚才的地址来说，http://192.168.200.133/hm如果不加斜杠，那么按照上述规则，如果指令server_name_in_redirect为on，则301重定向地址变为 http://localhost/hm/,如果为off，则301重定向地址变为http://192.168.200.133/ht/。后面这个是正常的，前面地址就有问题。

注意server_name_in_redirect指令在Nginx的0.8.48版本之前默认都是on，之后改成了off,所以现在我们这个版本不需要考虑这个问题，但是如果是0.8.48以前的版本并且server_name_in_redirect设置为on，我们如何通过rewrite来解决这个问题?

解决方案

我们可以使用rewrite功能为末尾没有斜杠的URL自动添加一个斜杠

```
server {
    listen  80;
    server_name localhost;
    server_name_in_redirect on;
    location /hm {
        if (-d $request_filename){
            rewrite ^/(.*)([^/])$ http://$host/$1$2/ permanent;
        }
    }
}
```

合并目录

搜索引擎优化(SEO)是一种利用搜索引擎的搜索规则来提供目的网站的有关搜索引擎内排名的方式。我们在创建自己的站点时，可以通过很多中方式来有效的提供搜索引擎优化的程度。其中有一项就包含URL的目录层级一般不要超过三层，否则的话不利于搜索引擎的搜索也给客户端的输入带来了负担，但是将所有的文件放在一个目录下又会导致文件资源管理混乱并且访问文件的速度也会随着文件增多而慢下来，这两个问题是相互矛盾的，那么使用rewrite如何解决上述问题?

举例，网站中有一个资源文件的访问路径时/server/11/22/33/44/20.html,也就是说20.html存在于第5级目录下，如果想要访问该资源文件，客户端的URL地址就要写成`http://www.web.name/server/11/22/33/44/20.html`,

```
server {
    listen 80;
    server_name www.web.name;
    location /server{
        root html;
    }
}
```

但是这个是非常不利于SEO搜索引擎优化的，同时客户端也不好记.使用rewrite我们可以进行如下配置:

```
server {
    listen 80;
    server_name www.web.name;
    location /server{
        rewrite ^/server-([0-9]+)-([0-9]+)-([0-9]+)-([0-9]+)\.html$ /server/$1/$2/$3/$4/$5.html last;
    }
}
```

这样的花，客户端只需要输入http://www.web.name/server-11-22-33-44-20.html就可以访问到20.html页面了。这里也充分利用了rewrite指令支持正则表达式的特性。

防盗链

防盗链之前我们已经介绍过了相关的知识，在rewrite中的防盗链和之前将的原理其实都是一样的，只不过通过rewrite可以将防盗链的功能进行完善下，当出现防盗链的情况，我们可以使用rewrite将请求转发到自定义的一张图片和页面，给用户比较好的提示信息。下面我们就通过根据

文件类型实现防盗链的一个配置实例:

```
server{
    listen 80;
    server_name www.web.com;
    locatin ~* ^.+\.(gif|jpg|png|swf|flv|rar|zip)${
        valid_referers none blocked server_names
*.web.com;
        if ($invalid_referer){
            rewrite ^/
http://www.web.com/images/forbidden.png;
        }
    }
}
```

根据目录实现防盗链配置:

```
server{
    listen 80;
    server_name www.web.com;
    location /file/{
        root /server/file/;
        valid_referers none blocked server_names
*.web.com;
        if ($invalid_referer){
            rewrite ^/
http://www.web.com/images/forbidden.png;
        }
    }
}
```