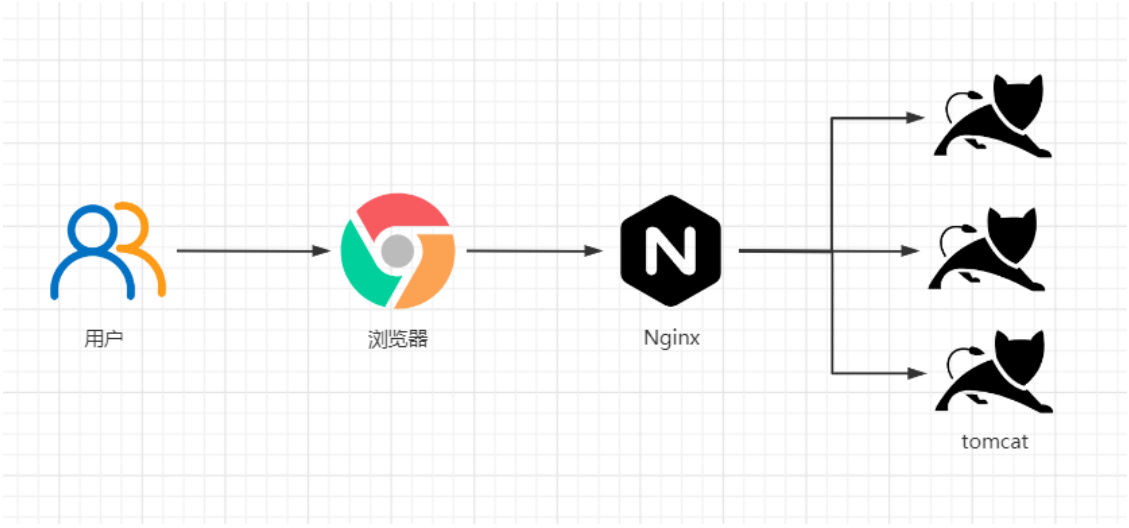
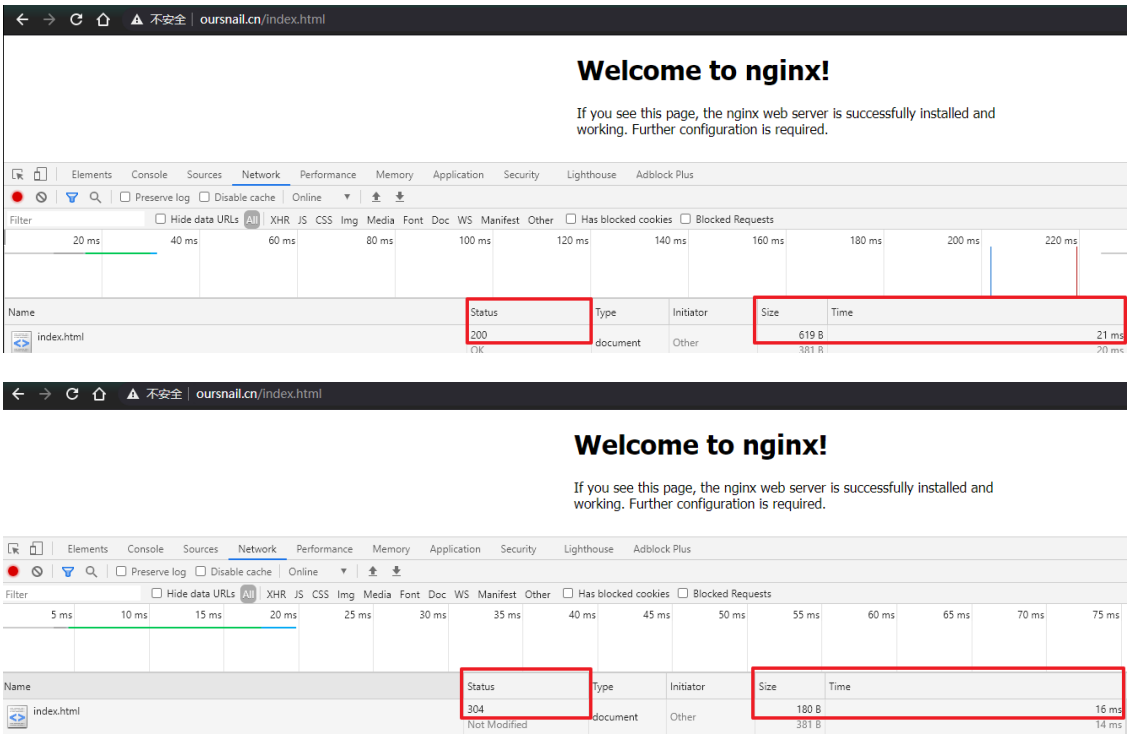
由于tomcat这种应用服务器本身吞吐量有限，缓存的出现不仅可大大提高响应速度，也可降低对应用服务器的访问压力。



上图的架构，可以设计成两级缓存，诸如html、css、js等静态资源可缓存于浏览器中，tomcat中的一些静态资源也可以缓存到nginx侧。前者可避免客户端重复获取静态资源提高响应速度、降低服务器压力；后者也可降低tomcat压力、降低内网传输带宽损耗。不过针对图片、大的js文件，最好的方案是走CDN加速。

首先是浏览器缓存。我们以访问nginx默认的 `index.html` 为例。

一、浏览器缓存-200和304问题



第一次直接访问，第二次是F5刷新，发现响应状态码、响应的大小和时间都是不一样的。这里就涉及到浏览器默认的缓存机制了。

首先来说下200和304两个状态码的机制。

状态码200：请求已成功，请求所希望的响应头或数据体将随此响应返回。即返回的数据为全量的数据，如果文件不通过GZIP压缩的话，文件是多大，则要有多大传输量。

状态码304：如果客户端发送了一个带条件的 GET 请求且该请求已被允许，而文档的内容（自上次访问以来或者根据请求的条件）并没有改变，则服务器应当返回这个状态码。即客户端和服务器端只需要传输很少的数据量来做文件的校验，如果文件没有修改过，则不需要返回全量的数据。

在客户端强制刷新，如ctrl+f5这种情况下，所有的缓存策略失效，服务器端都会返回200；

在客户端非强制刷新，如点击刷新按钮或按f5的情况下，服务器端会根据request头中：If-Modified-Since字段的时间与文件的实际修改时间进行比较，

如果修改时间比If-Modified-Since时间要新，则服务器会认为文件已经修改过了，向客户端返回全量的数据，客户端本地的缓存失效，状态码为200。

如果修改时间比If-Modified-Since时间要旧，则服务器会认为文件并未修改过，并且只会向客户端写回头文件，不返回文件数据，客户端使用本地缓存，状态码为304。

状态为304的请求要比状态为200的请求的数据量小很多，因为304只需要返回响应头，并不需要返回整个文件，所以只需要几字节就可以了，这样能够节省大量的网络带宽，并减少了页面的渲染时间。

因此当我修改文件时，文件的时间就会发生更改，再次f5刷新页面则又变为了200响应码。读者朋友们可以自己试试哟。

二、Nginx控制浏览器缓存

关于缓存，我们在之前学习HTTP协议时已经学习过，我们再来简单地复习一下。

浏览器缓存分为两大类：强制缓存：Expires，cache-control 和缓存协商：Last-modified ，Etag。

关于强制缓存中的Expires和Cache-Control的区别，我们只要知道：

- Expires 是http1.0的产物，Cache-Control是http1.1的产物
- 两者同时存在的话，Cache-Control优先级高于Expires；
- 在某些不支持HTTP1.1的环境下，Expires就会发挥用处。所以Expires其实是过时的产物，现阶段它的存在只是一种兼容性的写法

- Expires是一个具体的服务器时间，这就导致一个问题，如果客户端时间和服务器时间相差较大，缓存命中与否就不是开发者所期望的。Cache-Control是一个时间段，控制就比较容易

关于缓存协商中的ETag和If-None-Match：

- Etag是上一次加载资源时，服务器返回的response header，是对该资源的一种唯一标识，只要资源有变化，Etag就会重新生成。
- 浏览器在下一次加载资源向服务器发送请求时，会将上一次返回的Etag值放到request header里的If-None-Match里
- 服务器接受到If-None-Match的值后，会拿来跟该资源文件的Etag值做比较，如果相同，则表示资源文件没有发生改变，命中协商缓存。

关于缓存协商中的Last-Modified和If-Modified-Since：

- Last-Modified是该资源文件最后一次更改时间，服务器会在response header里返回，同时浏览器会将这个值保存起来
- 在下一次发送请求时，放到request header里的If-Modified-Since里，服务器在接收到后也会做比对，如果相同则说明文件没有改动，返回304

那么就涉及ETag和Last-Modified区别：

- 在方式上，Etag是对资源的一种唯一标识，而Last-Modified是该资源文件最后一次更改时间
- 在精确度上，Etag要优于Last-Modified。Last-Modified的时间单位是秒，如果某个文件在1秒内改变了多次，那么他们的Last-Modified其实并没有体现出来修改，但是Etag每次都会改变确保了精度；如果是负载均衡的服务器，各个服务器生成的Last-Modified也有可能不一致。
- 在性能上，Etag要逊于Last-Modified，毕竟Last-Modified只需要记录时间，而Etag需要服务器通过算法来计算出一个hash值。
- 在优先级上，服务器校验优先考虑Etag。

此外，关于控制缓存开关的Pragma 和 Cache-Control区别为：

- Pragma的值为no-cache时，表示禁用缓存
- Pragma是旧产物，已经逐步抛弃，有些网站为了向下兼容还保留了这两个字段。如果一个报文中同时出现Pragma和Cache-Control时，以Pragma为准。
- 同时出现Cache-Control和Expires时，以Cache-Control为准。即优先级从高到低是 Pragma -> Cache-Control -> Expires

好了，关于缓存的涉及的基础字段全部说明完毕，下面就是说浏览器缓存的过程：

- 浏览器第一次加载资源，服务器返回200，浏览器将资源文件从服务器上请求下载下来，并把response header及该请求的返回时间一并缓存；
- 下一次加载资源时，先比较当前时间和上一次返回200时的时间差，如果没有超过cache-control设置的max-age，则没有过期，命中强缓存，不发请求直接从本地缓存读取该文件（如果浏览器不支持HTTP1.1，则用expires判断是否过期）；如果时间过期，则向服务器发送header带有If-None-Match和If-Modified-Since的请求；
- 服务器收到请求后，优先根据Etag的值判断被请求的文件有没有做修改，Etag值一致则没有修改，命中协商缓存，返回304；如果不一致则有改动，直接返回新的资源文件带上新的Etag值并返回200；
- 如果服务器收到的请求没有Etag值，则将If-Modified-Since和被请求文件的最后修改时间做比对，一致则命中协商缓存，返回304；不一致则返回新的last-modified和文件并返回200；

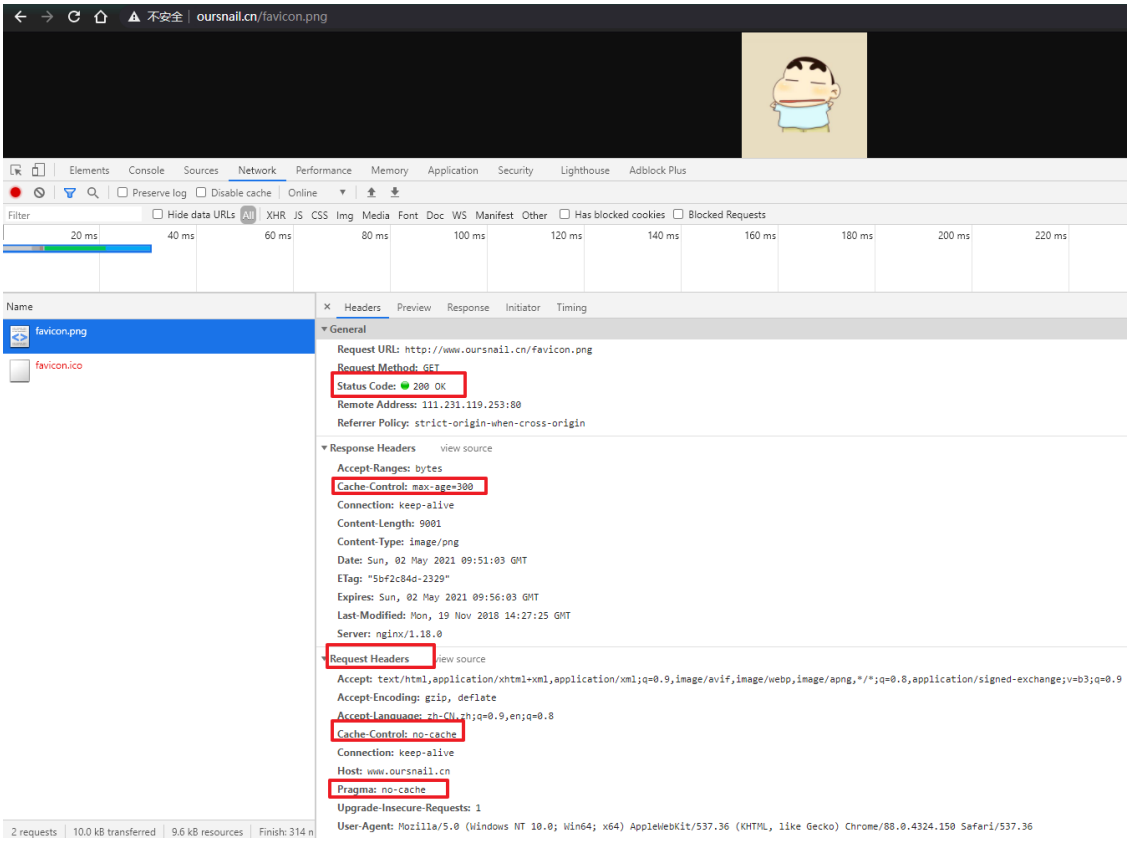
注意注意，用户的不同行为会触发不同的缓存机制：

- 地址栏访问，链接跳转是正常用户行为，将会触发浏览器缓存机制；
- F5刷新，浏览器会设置max-age=0，跳过强缓存判断，会进行协商缓存判断；
- ctrl+F5刷新，跳过强缓存和协商缓存，直接从服务器拉取资源。

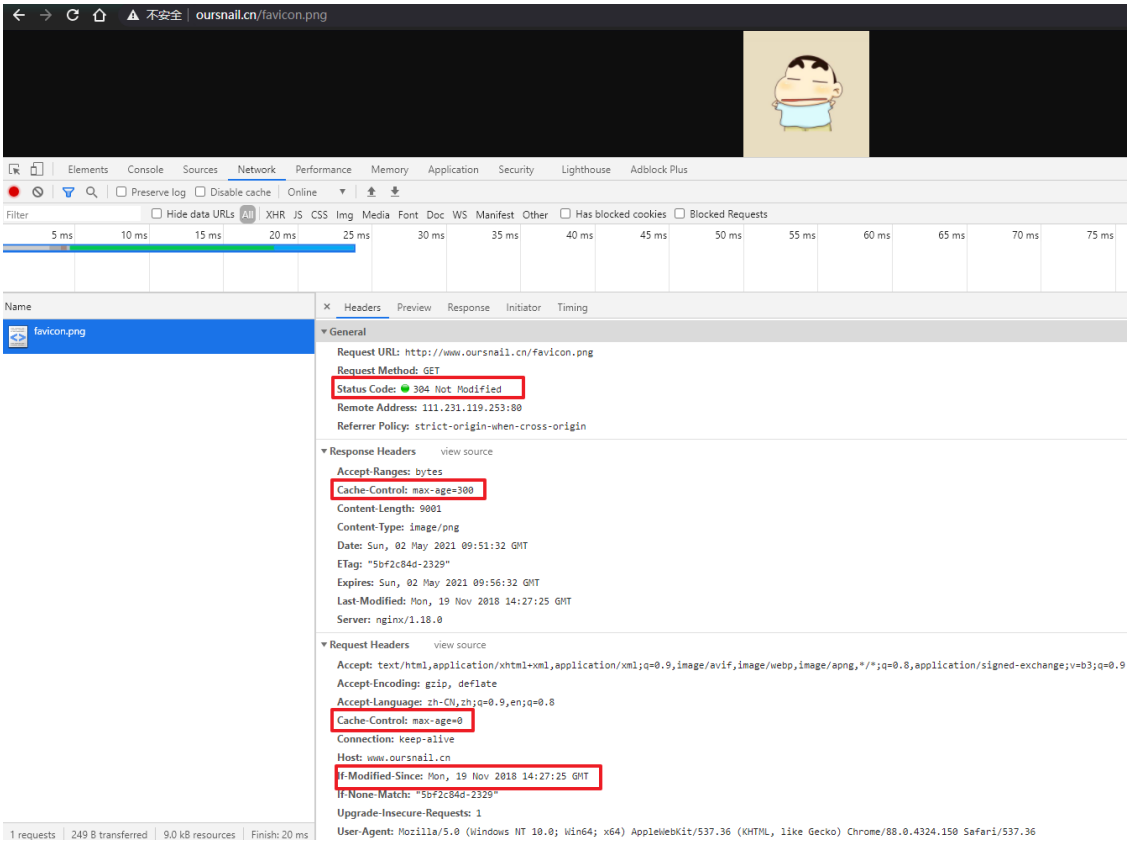
在 `nginx` 中配置 `expires` 字段即可实现Expires和Cache-Control的控制，示例：

```
location /files {
    alias /home/imooc;
    # Cache-Control: max-age=10，即缓存时间为10秒
    # expires 10s;

    # 这里制定到22点30分过期
    # expires @22h30m;

    # 缓存提前1小时过期，即expires时间比当前时间早1个小时，Cache-Control:
no-cache
    # expires -1h;

    # 不设置缓存，Cache-Control: no-cache，expires的时间是1970年
    # expires epoch;

    # nginx默认选项，响应头中没有Cache-Control或expires字段显示了
    # expires off;

    # 最大缓存时间，315360000秒
    expires max;
}
```

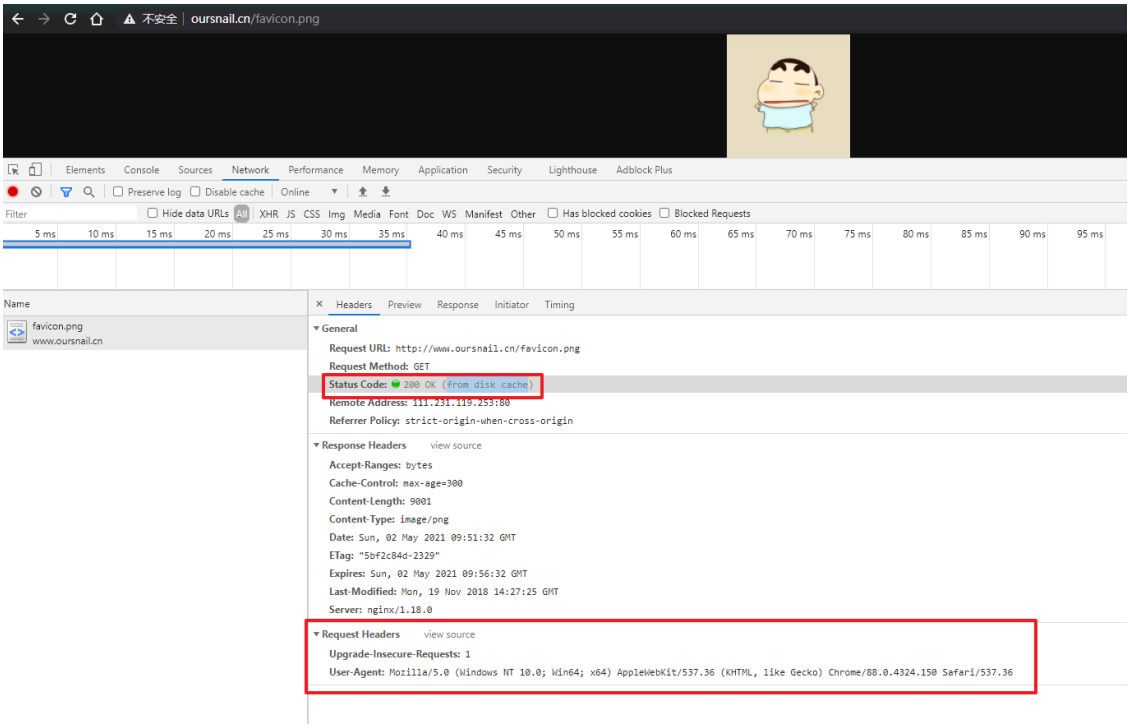
首先第一次访问，响应为200，第一次访问是直接从服务器拉取资源的：



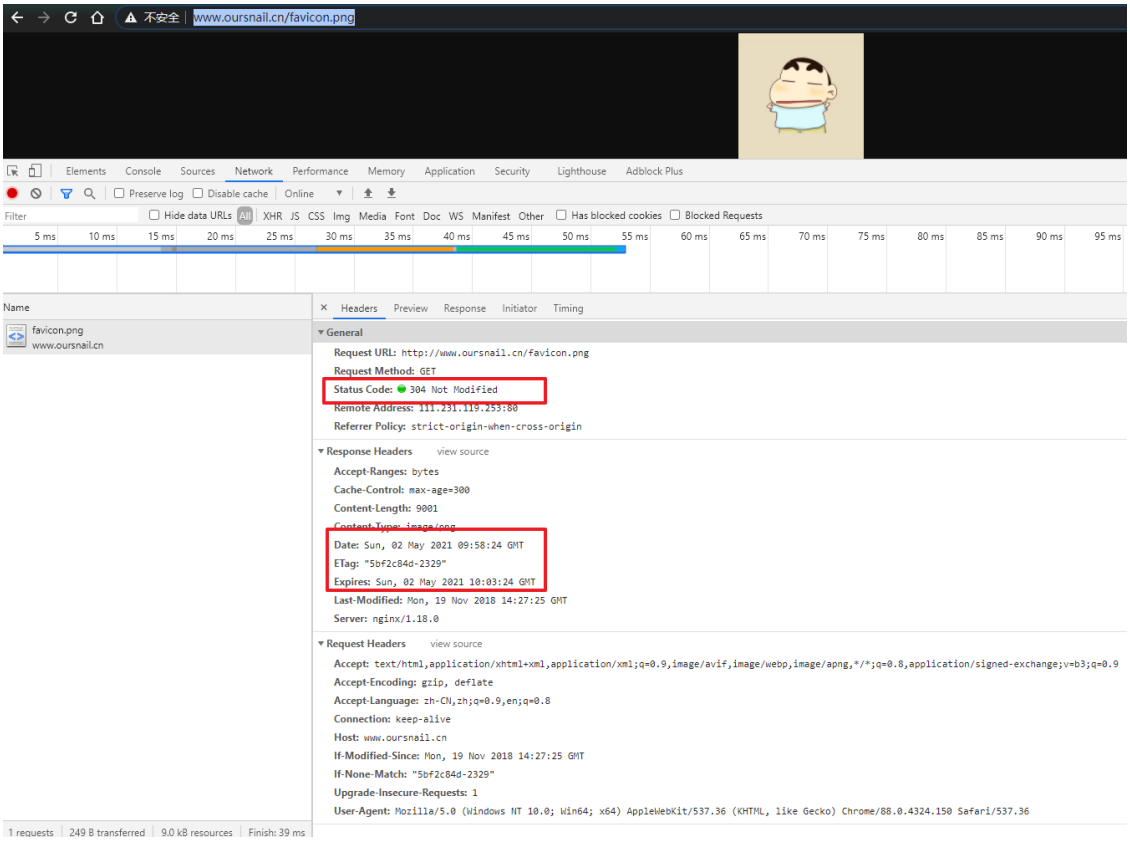
然后F5刷新页面，响应304，表示走的是协商协议，因为请求体的cache-control:max-age=0，所以会跳过强缓存走协商缓存：



重新打开一个tab输入网址，看到的是响应200状态码，并且显示from disk cache，可以看到并没有实际向服务器去请求资源，而是直接读取的硬盘：



我这里的max-age设置的是300秒，当超过300秒后再次打开新的tab去请求页面时，由于已经超时，因此强缓存已经消失，这个时候就要走协商缓存了（忽略Expires时间相差8个小时问题，只要跟上图对比就行，发现expires又刷新了，那么紧接着再去打开新的网页请求的话，就会再次走到读取disk这一步）：

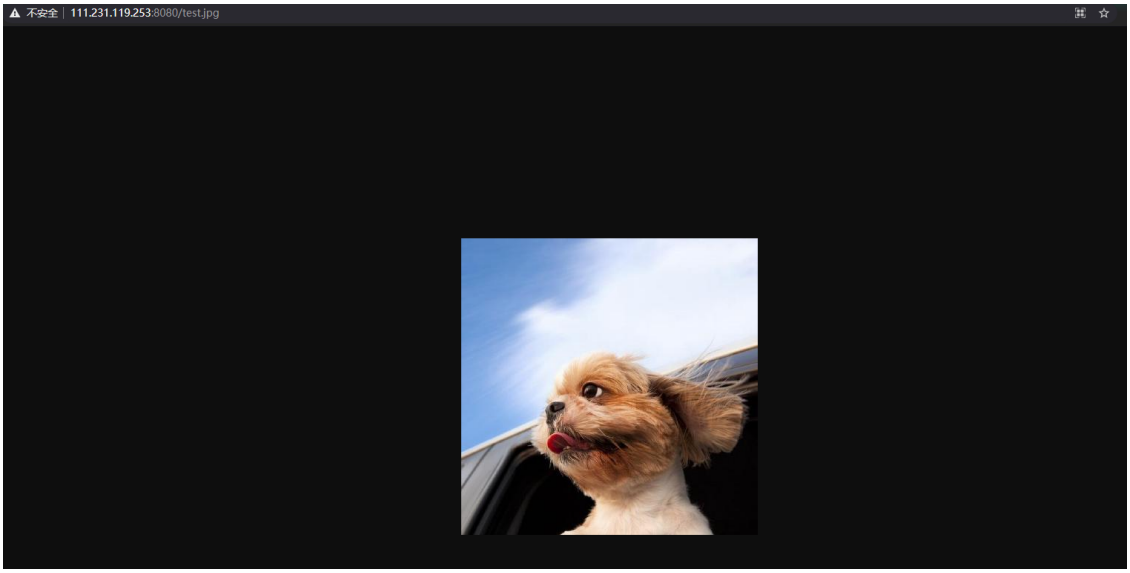


浏览器缓存的学问还是挺多的，这里核心上要说明的就是nginx如何来控制浏览器缓存的。

三、nginx的反向代理缓存

对于upstream上游服务器也有一些静态资源，nginx也可以对其进行缓存，提高用户请求的响应速度。

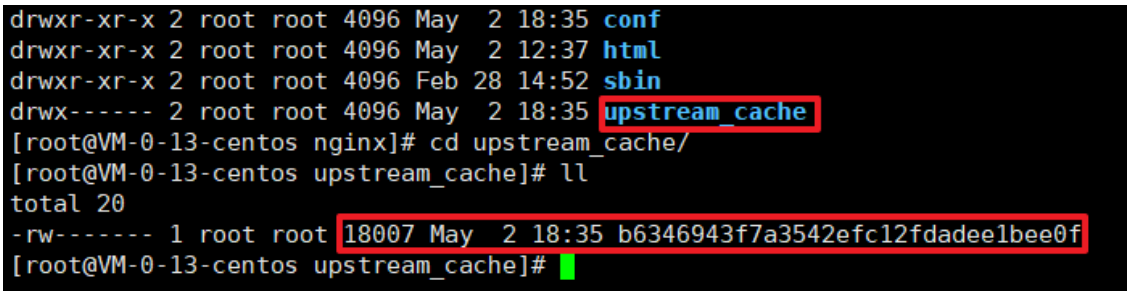
为了测试，我提前往服务器的ROOT下上传了一张图片：



下面对nginx进行配置：

```
upstream tomcats{
      #server 111.231.119.253:8080;
      server 127.0.0.1:8080;
}

 # proxy_cache_path 设置缓存目录
 # # keys_zone 设置共享内存以及占用空间大小
 # # max_size 设置缓存大小
 # # inactive 超过此时间则被清理
 # # use_temp_path 临时目录，使用后会影响nginx性能
proxy_cache_path /usr/local/nginx/upstream_cache keys_zone=mycache:5m max_size=1g inactive=1m use_temp_path=off;

  server {
      listen       80;
      server_name  www.oursnail.cn;

      # 启用缓存，和keys_zone一致
      proxy_cache mycache;
      # 针对200和304状态码缓存时间为8小时
      proxy_cache_valid 200 304 8h;

      #charset koi8-r;

      #access_log  logs/host.access.log  main;
      location / {
#         root   html;
#         index  index.html index.htm;
#        expires 300s;
#        autoindex_localtime on; #GMT时间 改为 本地时间
#        autoindex on;
         proxy_pass http://tomcats;
      }
```

当我访问：`http://www.oursnail.cn/test.jpg` 时也可以看到这张图片，不过特别的是在我的nginx目录下生成了 `upstream_cache` 文件夹：



由于缓存时间设置的是1分钟，过了一分钟后这个文件就自动消失了，此外值得注意的是，如果超出了max_size参数设置的最大值，使用LRU算法移除缓存数据。

这样就实现了对上游服务器的静态资源的缓存，属于nginx的一个优化思路。