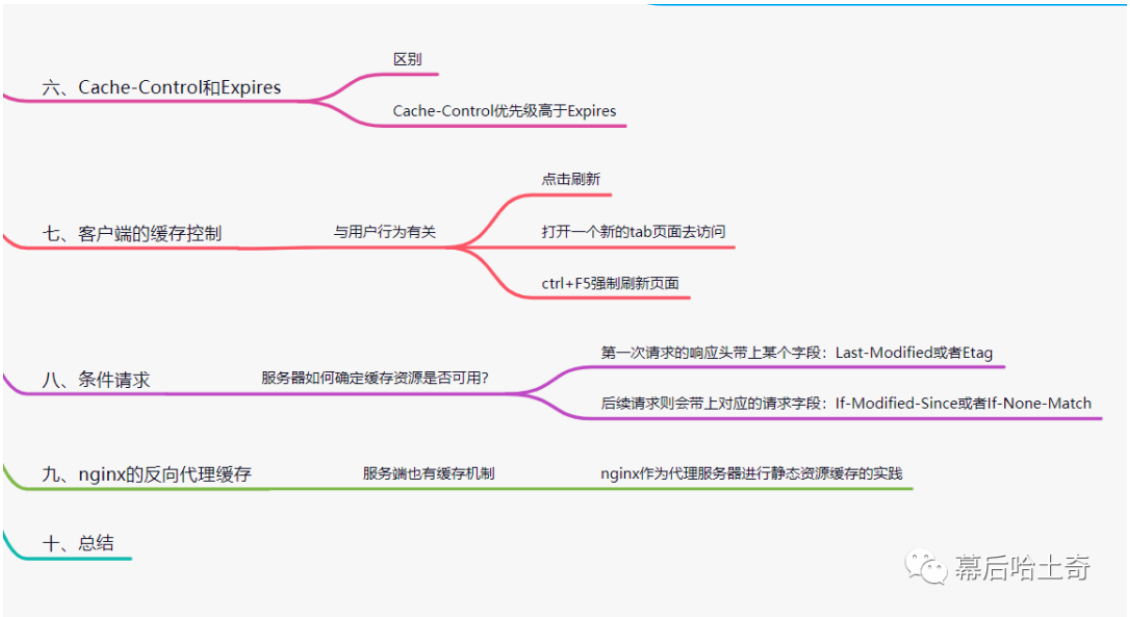
下篇聚焦的部分为：



六、Cache-Control和Expires

在说明关键问题前，我们补充下Cache-Control和Expires两个字段的说明。

虽然我们配置的是Expires，但是我们一直在说Cache-Control字段，好像忽略了Expires字段。

实际上他们是新老两代的接替，他两的区别是：

- 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；

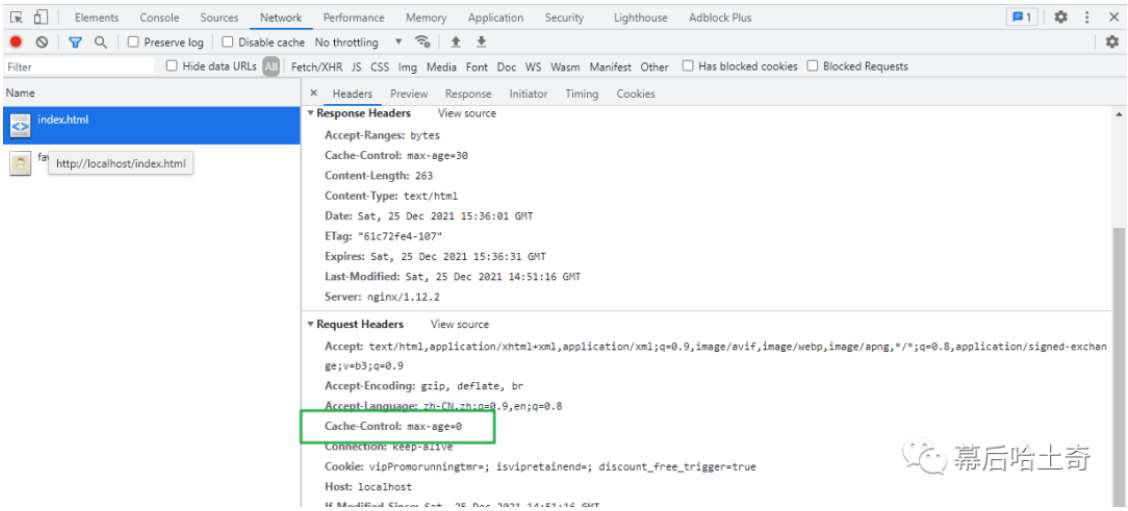
所以在202X年的今天，主要还是看Cache-Control字段咯。

七、客户端的缓存控制

在Cache-Control之max-age这一节中，测试方式是采取重新打开一个新的tab去访问，而不是直接刷新页面，其区别是因为：

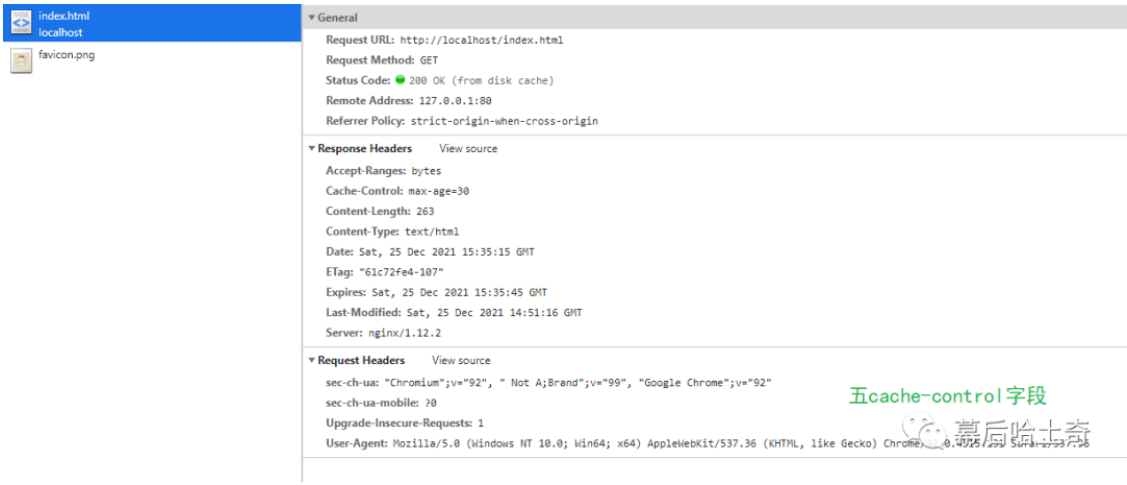
不止服务器可以发“Cache-Control”头，浏览器也可以发“Cache-Control”，也就是说请求 - 应答的双方都可以用这个字段进行缓存控制，互相协商缓存的使用策略。

当点击刷新时，浏览器会在请求头里加一个“Cache-Control: max-age=0”：



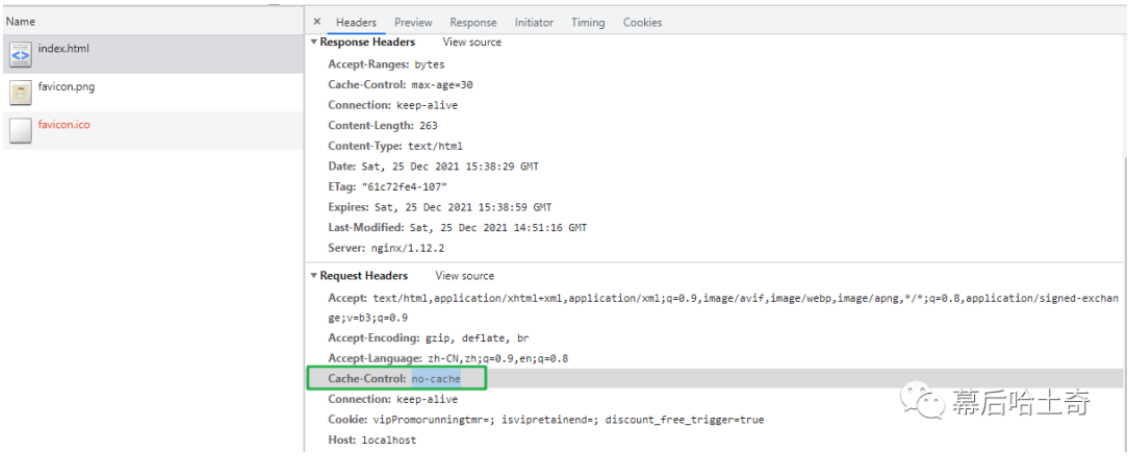
此时我获取到的状态码往往是304，从这个现象看，服务端响应头中的Cache-Control: max-age=30已失效，浏览器会向服务端发起请求，服务端发现其资源未修改则返回304，浏览器直接使用缓存。

而如果是打开一个新的tab页面去访问，请求头中则不会有此字段：



此时若在max-age=30的有效期内，会直接获取缓存，不会向服务端发起任何请求，这是我们上面已经实验验证过的；若超过此时间，则浏览器会向服务端发起请求，服务端发现其资源未修改则返回304，浏览器直接使用缓存。

当ctrl+F5强制刷新页面时，谷歌浏览器中会增加一个“Cache-Control: no-cache”：



从实验结果来看，每次强制刷新，服务端返回的都是200状态码，此时返回了完整数据。

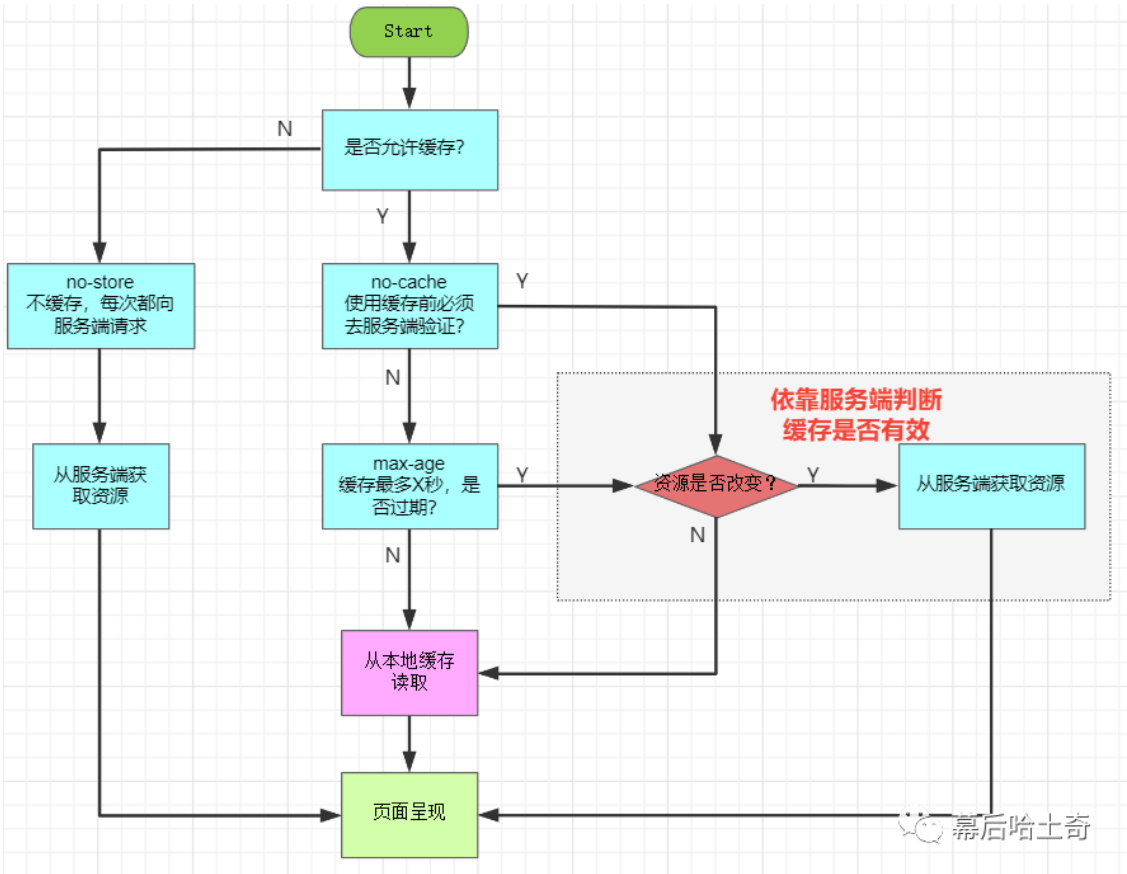
可以显著看到其区别，这就是客户端的缓存控制。

可以看到，上面无数次提到了：**服务端判断其资源是否修改，若未修改则返回304，不返回数据；若已修改则返回全部新数据+200状态码。**

下面来看看其机制。

八、条件请求

核心问题点是：**服务器如何确定缓存资源是否可用**？其实我们就是关注流程图的这一块是啥情况：



一个容易想到的思路是发起两个请求组成“验证动作”：先是一个 HEAD，获取资源的修改时间等元信息，然后与缓存数据比较，如果没有改动就使用缓存，节省网络流量，否则就再发一个 GET 请求，获取最新的版本。

但是两个请求的成本太高了，所以 HTTP 协议就定义了一系列“If”开头的“条件请求”字段，专门用来检查验证资源是否过期，**把两个请求才能完成的工作合并在一个请求里做**。而且，验证的责任也交给服务器，浏览器只需“坐享其成”。

我们只关注最重要的，有两组字段需要关注，这两组搭档都是成对出现的。

- **第一次请求的响应头带上某个字段：Last-Modified或者Etag**
- **后续请求则会带上对应的请求字段：If-Modified-Since或者If-None-Match**

PS：若响应头没有Last-Modified或者Etag字段，则请求头也不会有对应的字段。

响应头的Last-Modified对应请求头的If-Modified-Since，响应头的Etag对应请求头的If-None-Match，这是两组判断缓存是否失效的机制，我们分别来看下其判断原理。

先来说下第一组：**Last-Modified/If-Modified-Since**，其判断流程为：

- 浏览器第一次跟服务器请求一个资源，服务器在返回这个资源的同时，在respone的header加上Last-Modified的header，这个header表示这个资源在服务器上的最后修改时间。
- 浏览器再次跟服务器请求这个资源时，在request的header上加上If-Modified-Since的header，这个header的值就是上一次请求时返回的Last-Modified的值。
- 服务器再次收到资源请求时，根据浏览器传过来If-Modified-Since和资源在服务器上的最后修改时间判断资源是否有变化，如果没有变化则返回304 Not Modified，但是不会返回资源内容；如果有变化，就正常返回资源内容。
- 浏览器收到304的响应后，就会从缓存中加载资源。
- 若不是304，则说明资源已修改，浏览器直接从服务器加载资源时，Last-Modified的Header在重新加载的时候会被更新，下次请求时，If-Modified-Since会启用上次返回最新的Last-Modified值。

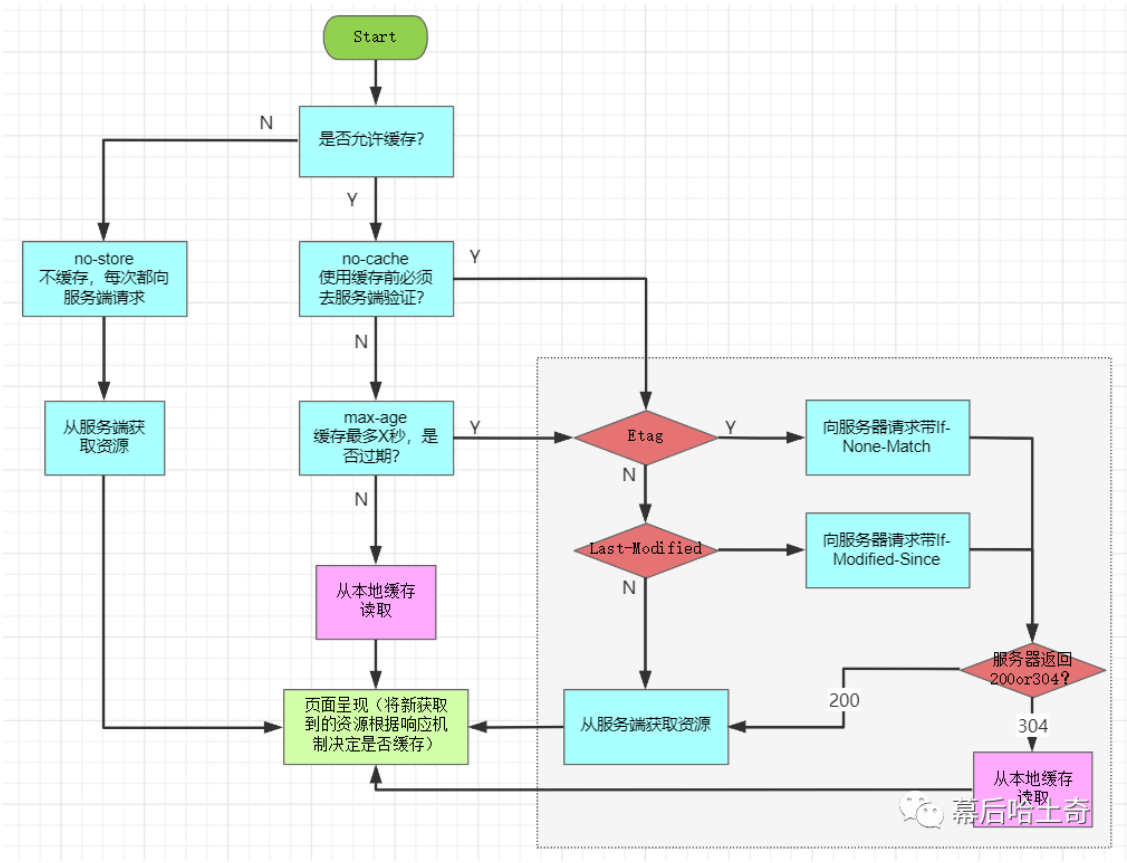
Etag/If-None-Match：

- 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。

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，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。

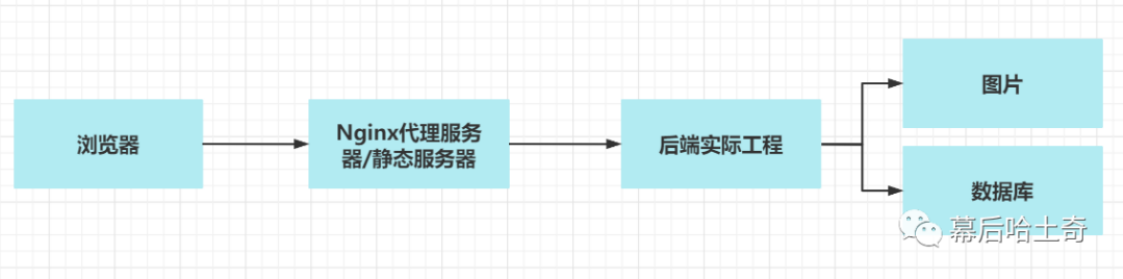
整体流程图将细分为：



九、Nginx的反向代理缓存

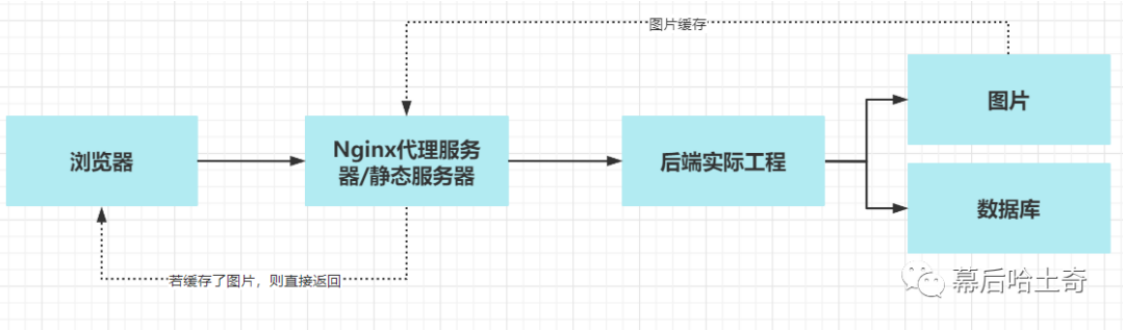
本节是补充点，是思维的延申。

上面说了一大堆，说的都是在浏览器如何如何缓存，即客户端的缓存，而服务端也有缓存机制，尤其是nginx这种反向代理服务器。



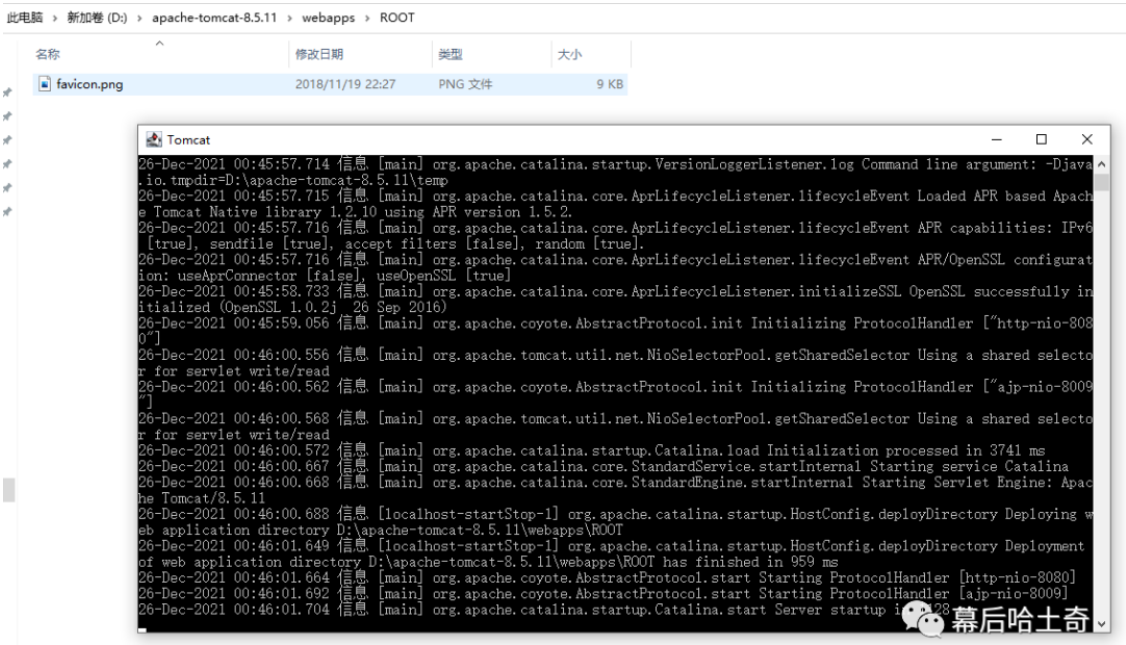
当我访问一张普通的图片，请求则可能会经历：浏览器---》nginx代理服务器---》实际服务器/图片资源，容易想到：**图片尤其是热点图片，能不能缓存到nginx服务器上面呢?**

这样可以大大降低后端压力，也能提高响应速度。



好，我们来实践一把。

首先需要一个tomcat来作为后端实际工程，我在webapps目录下的ROOT目录下放一张图片：favicon.png，默认监听端口为8080，启动tomcat即可：



下面需要配置nginx作为代理服务器，我本地配置为：

```
http {
    include       mime.types;
```

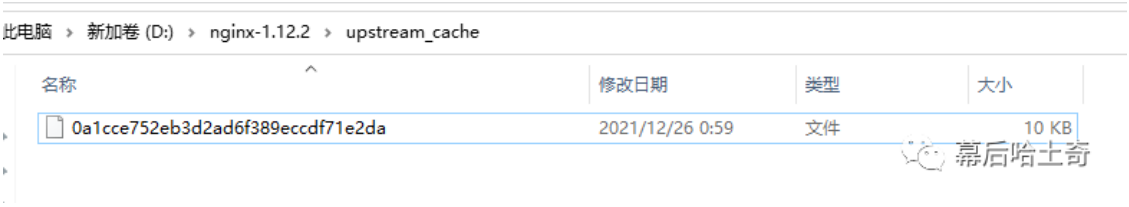
```
    default_type  application/octet-stream;

    sendfile        on;

    keepalive_timeout  65;

    upstream tomcats{
        server 127.0.0.1:8080;
    }

    ## proxy_cache_path: 设置缓存目录为upstream_cache
    ## keys_zone: 设置共享内存以及占用空间大小
    ## max_size: 设置硬盘中最多可以缓存多少数据，当到达该数值时，nginx会删除最少访问的数据
    ## inactive: 设置缓存多长时间就失效，当硬盘上的缓存数据在该时间段内没有被访问过，就会失效了，该数据就会被删除，默认为10s
    proxy_cache_path ./upstream_cache keys_zone=mycache:5m max_size=1g inactive=1m;

    server {
        listen       80;
        server_name  localhost;
        ## 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;

        location / {            ## 反向代理到tomcats这组服务器，与上面的upstream呼应
            proxy_pass http://tomcats;
        }
    }
}
```

核心配置有两点，一个是配置了proxy_pass和upstream，这里配置的是nginx反向代理的上游服务器；另一个核心配置是proxy_cache相关，即反向代理缓存相关，配置了缓存哪种类型、缓存的时间以及保存路径。

当我访问nginx：http://localhost/favicon.png

nginx会去这里获取图片资源：http://localhost:8080/favicon.png

然后返回给浏览器，并且nginx目录下顺利生成了关于这种图片的缓存信息：



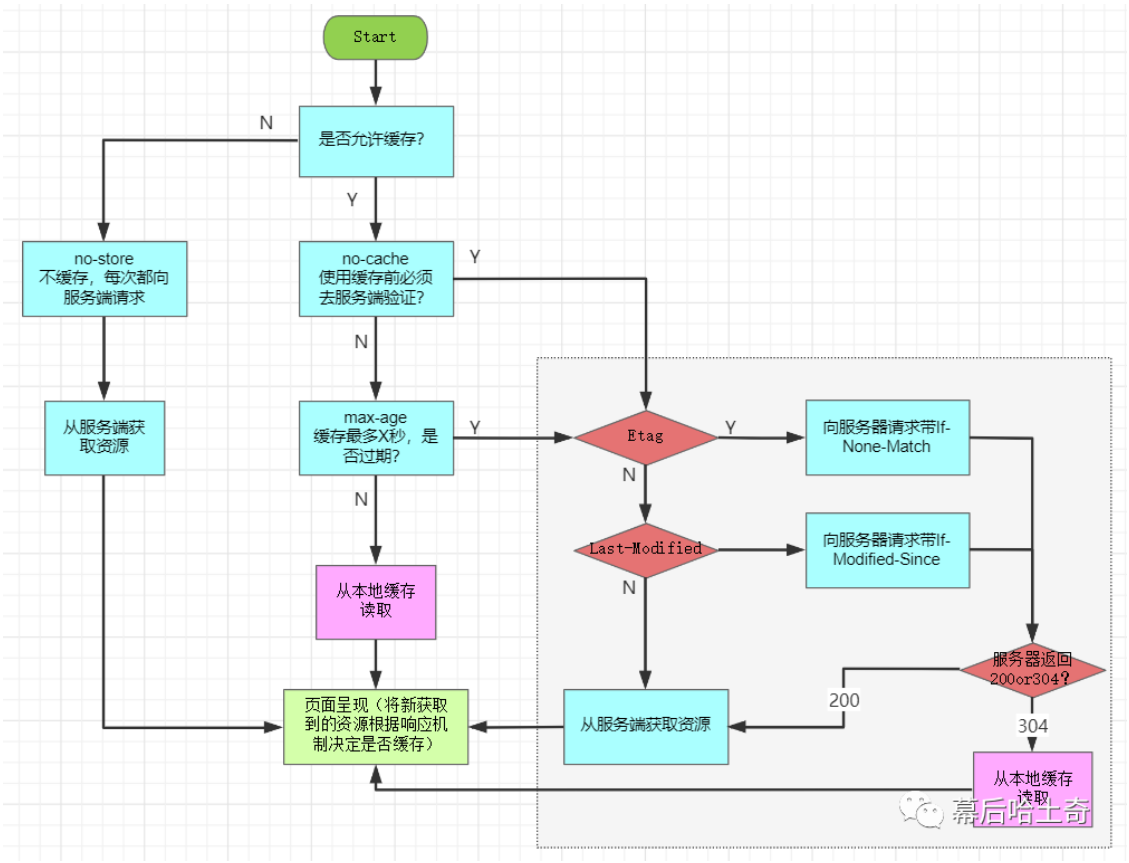
由于inactive配置的是1分钟，所以1分钟不再次去访问这种图片的话，则会被自动删除，即使proxy_cache_valid配置的是8小时，关于他两的区别不在本文讨论范围内。

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。

十、总结

这里总结下缓存的判断流程，相对于流程图稍微做下精简。



- **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，命中缓存，不发请求直接从本地缓存读取该文件；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，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，一致则命中缓存，返回304；不一致则返回新的last-modified和文件并返回200；**

此外，浏览器也可以发送“Cache-Control”字段，使用“max-age=0”或“no_cache”刷新数据，**因此实际情况需要根据浏览器的动作和自身的缓存机制决定，可能会有不一样的结果，我们需要具体问题具体分析了**。

关于HTTP之缓存控制就讨论到这里，读者朋友们，下篇文章见。