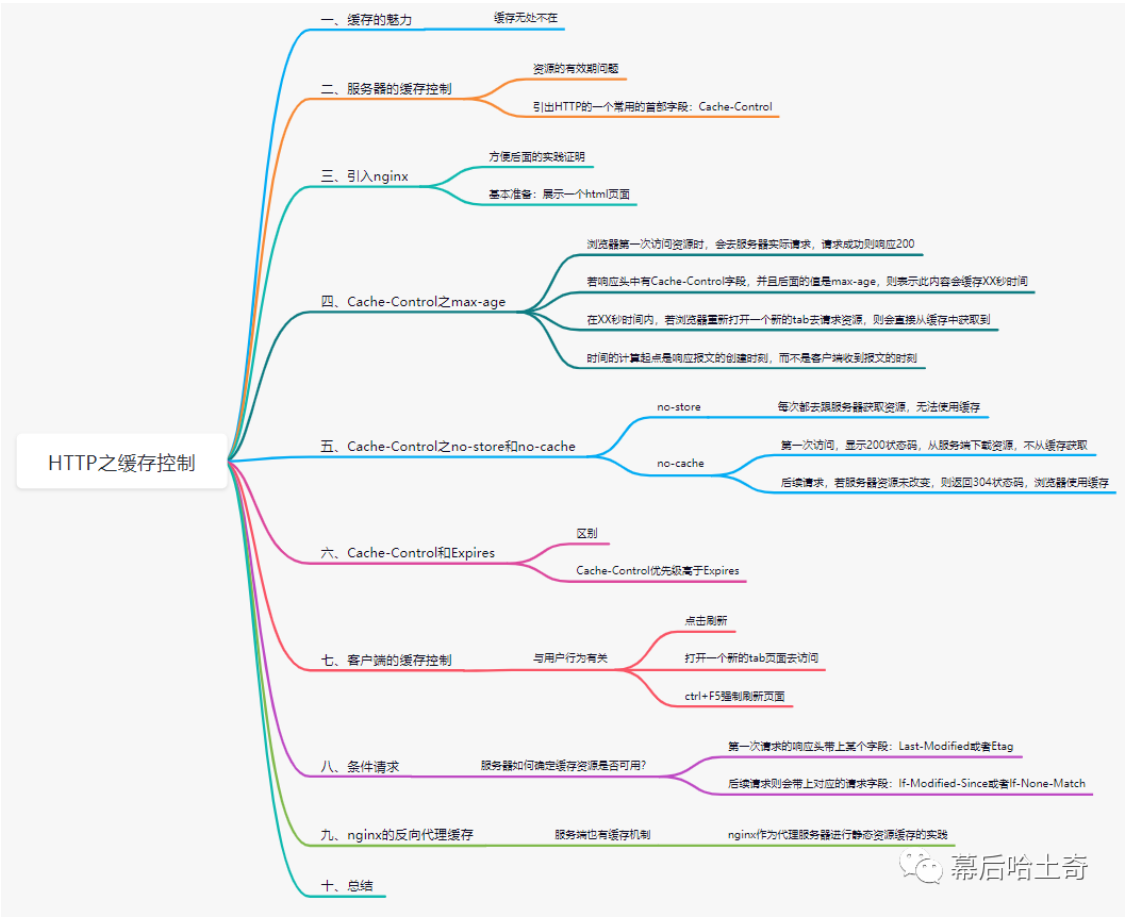
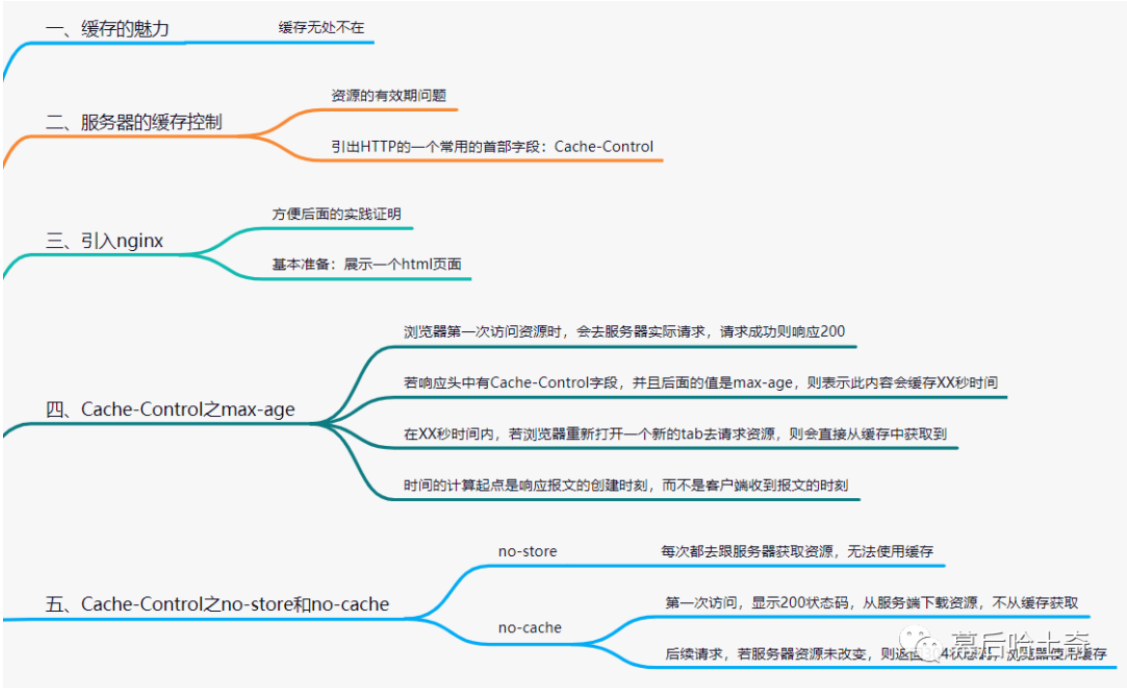
关于缓存，内容较多，分为上下两篇，整体思维导图如下：



上篇聚焦的部分为：



一、缓存的魅力

在计算机的世界里，缓存无处不在，比如我们的CPU和内存之间有L1、L2、L3等多级缓存，目的就是为了解决CPU运算速度与内存读写速度不匹配的矛盾；我们服务端的代码经常会使用缓存，目的也是为了提高读取速度，因为数据放在内存里往往会读的更快。

拿CDN来说，CDN就是典型的缓存服务器，我们可以将热门的图片和视频资源放在CDN上，为用户提供服务。

所谓CDN加速，实际上就是依靠分布在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。

如果没有分布式的CDN，可想而知中心平台的压力会有多大！用户的体验也将一塌糊涂，这都可以归纳于缓存的魅力。

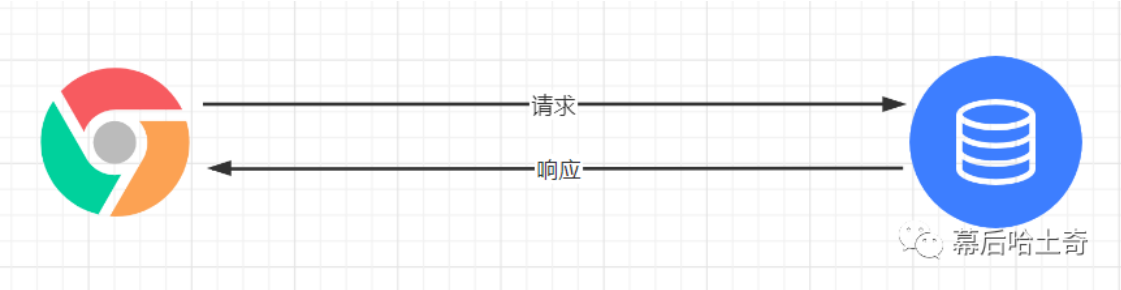
然而，缓存也会引入新的问题，当数据频繁更新，如果有缓存，那就需要同样刷新缓存，否则用户得到的还是“老”数据，所以，缓存带来了好处，也引入了维护的困难。

因此必须对缓存有所控制，典型的是设置有效期，拥有过期更新的特性，以适应千变万化的场景，我们回归正文，来看看在HTTP协议中缓存是如何使用和控制的。

二、服务器的缓存控制

我们来看下浏览器访问服务器的场景，我们以前往往会简单归纳于下面这个流程：

- 浏览器发起请求信息
- 服务器返回响应信息

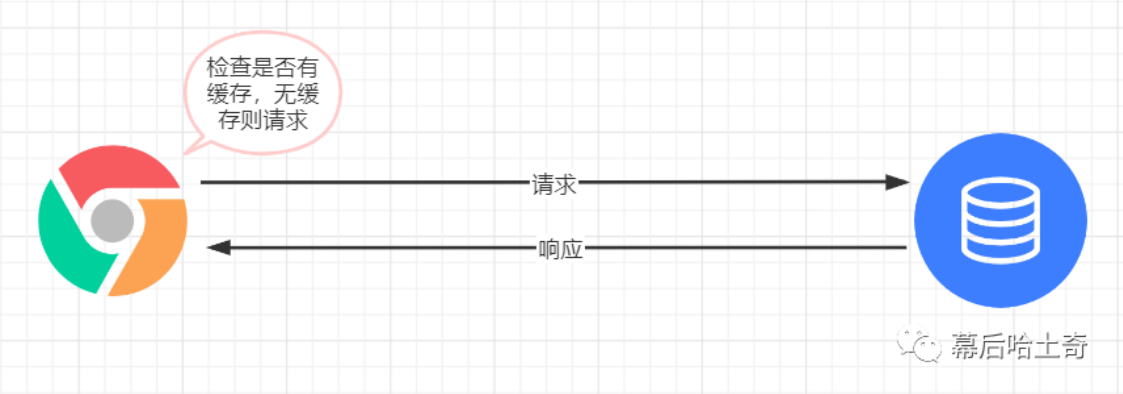


不过只有第一次访问网站时，浏览器才会老老实实去请求，但是它才不会那么老实，**有些资源我能缓存下来的话，不就不需要每次都费劲去请求新的了吗？**

于是流程变成：

- 浏览器检查是否有缓存，有则直接用，若发现缓存无数据，于是发送请求，向服务器获取资源
- 服务器响应请求，返回资源，同时标记资源的有效期；

- 浏览器缓存资源，等待下次重用



这里提到了一个关键字：资源的有效期，没错，很容易想到，服务器给浏览器返回资源时，给资源一个有效期，浏览器缓存下来后，在这个有效期内即可使用此缓存，否则就需要老老实实向服务器去重新请求。

首先，为什么要加有效期呢？直接给浏览器一直缓存一直用呗？

正如上面所提到的，这是因为网络上的数据随时都在变化，不能保证它稍后的一段时间还是原来的样子。

就像从超市买回来的零食，一直放着不吃，过了保质期可就不能再乱吃了，容易坏肚子。

好了，解释了为什么浏览器的缓存要有有效期后，我们来看看双方是如何约定这个有效期的呢？

就不得不提HTTP的一个常用的首部字段：Cache-Control

服务器可在响应首部中通过Cache-Control字段实现对客户端缓存时长的控制，下面我们结合实践来看看是如何控制的。

三、引入Nginx

为了更好地说明问题，我下载了nginx解压到本机（windows10）。

> 关于Nginx，在后续文章中将增加对其介绍，读者朋友可以跳到后续章节学习，也可以借助搜索引擎先去查看Nginx基础入门。

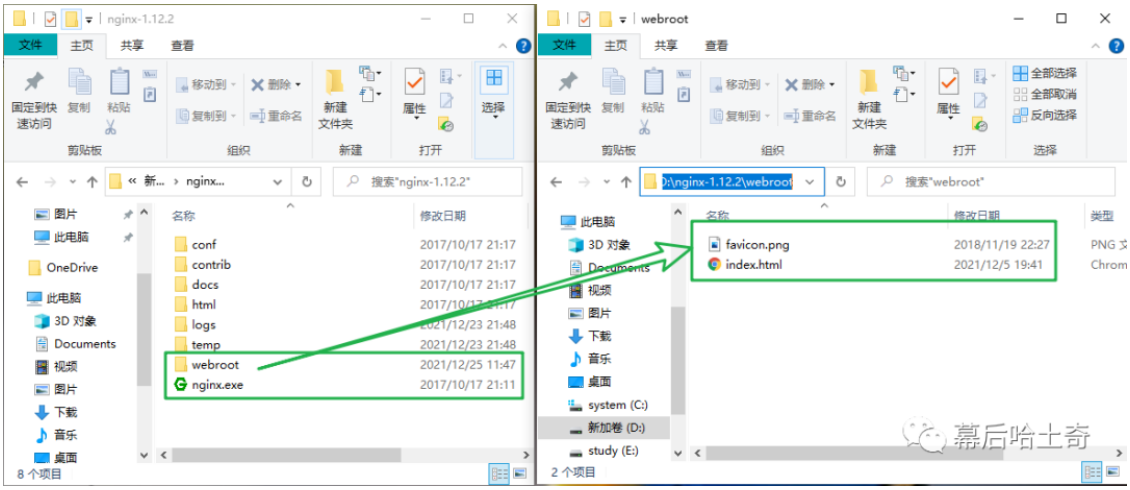
对conf/nginx.conf文件进行精简，即将注释的部分删干净，核心的配置如下：

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

```
    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

        location / {
            alias ./webroot/;
        }
    }
}
```

这里简单说下配置文件的意思，服务监听80端口，访问/默认会转向设置的./webroot/目录下的资源，其他就不再赘述了。

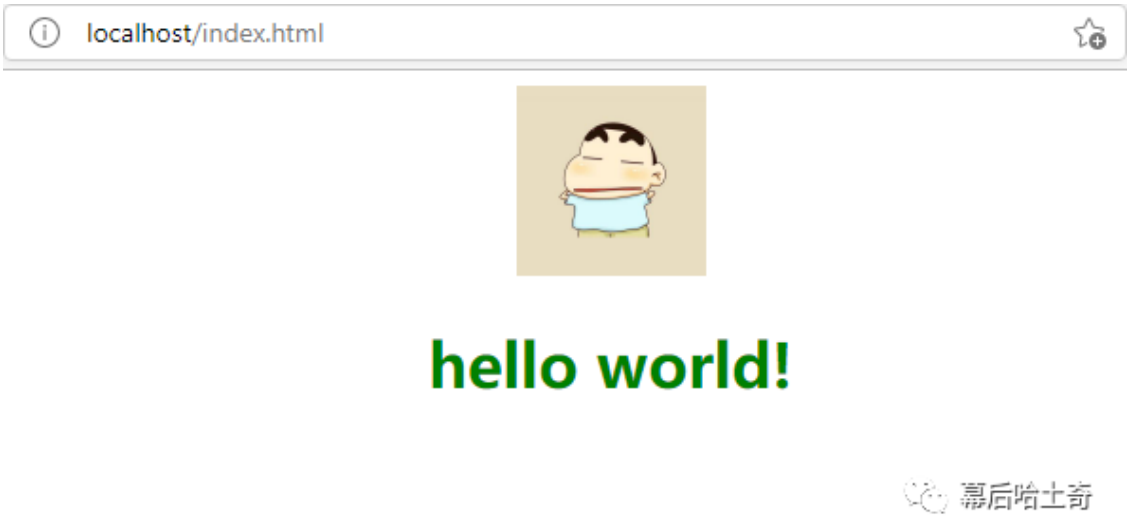
我们在nginx.exe可执行文件统计目录下新建一个文件夹叫做webroot，然后在文件夹里面放一个HTML文件叫做：index.html，此页面极其简单，是一个文字+图片的组合。



我们启动nginx：

```
Xshell 7 (Build 0085)
Copyright (c) 2020 NetSarang Computer, Inc. All rights reserved.

Type `help' to learn how to use Xshell prompt.
[C:\~]$ d:
[D:\]$ cd D:\nginx-1.12.2
[D:\nginx-1.12.2]$ nginx.exe -t 检查配置文件是否有误
nginx: the configuration file D:\nginx-1.12.2/conf/nginx.conf syntax is ok
nginx: configuration file D:\nginx-1.12.2/conf/nginx.conf test is successful

[D:\nginx-1.12.2]$ nginx.exe    启动nginx
```

通过浏览器访问 `http://localhost/xiaohuangren.jpg` 即可访问到静态页面：



好了，nginx是一个上手极其容易的代理服务器，它可以轻松完成很多事情，在本文中不再过多赘述，如果读者朋友完全不知道nginx为何物，也没有关系，本文只是通过它来方便试验和说明问题。

四、Cache-Control之max-age

通过nginx我可以很方便地进行缓存的控制，比如这里的Cache-Control字段，我简单加了一行配置：

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

    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

        location / {
            alias ./webroot/;
            ## 新增的一行
            expires 30s;
        }
    }
}
```

这个配置意思很简单，即过期时间是30秒，我们通过实验来具体看下其表现机制。

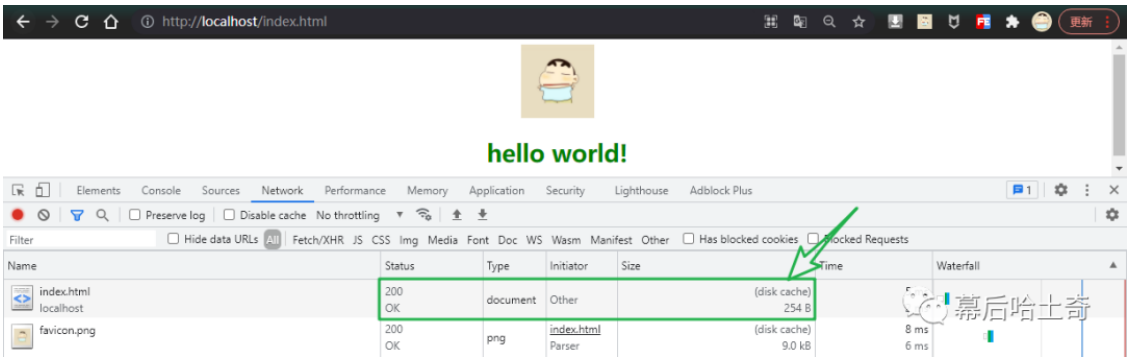
下面我使用谷歌浏览器进行实验，各个浏览器表现可能会有点差异，主要是浏览器关于缓存的实现机制可能略有不同。

我们输入url访问，第一次请求的响应状态码是200：

响应头中多了一个Cache-Control，后面30表示30秒：

当在30秒内，重新打开一个新的tab去访问时，将会直接从缓存中获取内容：

> 注意我的操作！！！不是F5刷新，更不是crtl+F 5，也不是其他操作，认准这个操作，下面还会说到不同操作对应会有不一样的结果。



因此我们获得到第一个知识点：

浏览器第一次访问资源时，会去服务器实际请求，请求成功则响应200，若响应头中有Cache-Control字段，并且后面的值是max-age，则表示此内容会缓存XX秒时间，在XX秒时间内，若浏览器重新打开一个新的tab去请求资源，则会直接从缓存中获取到。

此XX秒即缓存的生命周期，注意，**时间的计算起点是响应报文的创建时刻（即 Date 字段，也就是离开服务器的时刻），而不是客户端收到报文的时刻，也就是说包含了在链路传输过程中所有节点所停留的时间。**

比如，服务器设定“max-age=5”，但因为网络质量很糟糕，等浏览器收到响应报文已经过去了 4 秒，那么这个资源在客户端就最多能够再存 1 秒钟，之后就会失效。

截止目前，我们学习的还很简单，流程图可以表示为：

五、Cache-Control之no-store和no-cache

“max-age”是 HTTP 缓存控制最常用的属性，此外在响应报文里还可以用其他的属性来更精确地指示浏览器应该如何使用缓存：

- no-store：**直接禁止游览器缓存数据，每次用户请求该资源，都会向服务器发送一个请求，每次都会下载完整的资源。**
- no-cache：**它的字面含义容易与 no_store 搞混，实际的意思并不是不允许缓存，而是可以缓存，但在使用之前必须要去服务器验证是否过期，是否有最新的版本；**

- public：可以被所有的用户缓存，包括终端用户和CDN等中间代理服务器；
- private：只能被终端用户的浏览器缓存，不允许CDN等中继缓存服务器对其缓存；

我们主要看下no-store和no-cache。

修改下nginx的配置：

```
## 新增的一行
add_header    Cache-Control  no-store;
```

重启nginx后，无论我怎么访问此url，比如刷新、用新的tab来访问等等，只会给我一个结果：**每次都去跟服务器获取资源，无法使用缓存**。

no-store这个小伙子比较直，没啥可说的，我们来说说no-cache，修改nginx配置：

```
## 新增的一行
add_header    Cache-Control  no-cache;
```

我们来看下神奇现象。

首先ctrl+F5强制刷新页面，显示200状态码，从服务端下载资源，不从缓存获取。

当我重新刷新页面或在一个新的tab也访问该链接时：

这次返回的是神奇的304状态码。

304状态码可能对于很多人不是很熟悉，因为它确实比较低调，304的含义其实很简单：自从上次请求后，请求的网页未修改过。服务器返回此响应时，不会返回网页内容。

更直白点说就是，客户端来请求时，服务端发现资源没有改变，直接告诉客户端：资源没有变化，还使用之前获取到的资源即可。

可以理解304 Not Modified这个含义了吧。

不过客户端和服务端如何协商呢？这个将是下面要说明的内容，我们这里暂且卖个关子。

既然说资源没改变就会返回304，那我修改下HTML文件呗？修改好之后，再次刷新页面的效果：

可以看到，我在html文件中加了一点文本，返回又变成了200，显然浏览器又向服务端重新请求了资源，因为资源发生了变化。

抛开背后实现机制，我们可以理解no-cache的背后逻辑了。

我们完善下流程图：

剩下一个核心问题：资源是否改变是如何协商的？限于篇幅，下篇继续讨论。