

# 浏览器：一个浏览器是如何工作的？（阶段一）

winter 2019-02-09



对于前端开发来说，我们平时与浏览器打交道的时间是最多的。可浏览器对前端同学来说更多像一个神秘黑盒子的存在。我们仅仅知道它能做什么，而不知道它是如何做到的。

在我面试和接触过的前端开发者中，70% 的前端同学对这部分的知识内容只能达到“一知半解”的程度。甚至还有一部分同学会质疑这部分知识是否重要：这与我们的工作相关吗，学多了会不会偏移前端工作的方向？

事实上，我们这里所需要了解的浏览器工作原理只是它的大致过程，这部分浏览器工作原理不但是前端面试的常考知识点，它还会辅助你的实际工作，学习浏览器的内部工作原理和个中缘由，对于我们做性能优化、排查错误都有很大的好处。

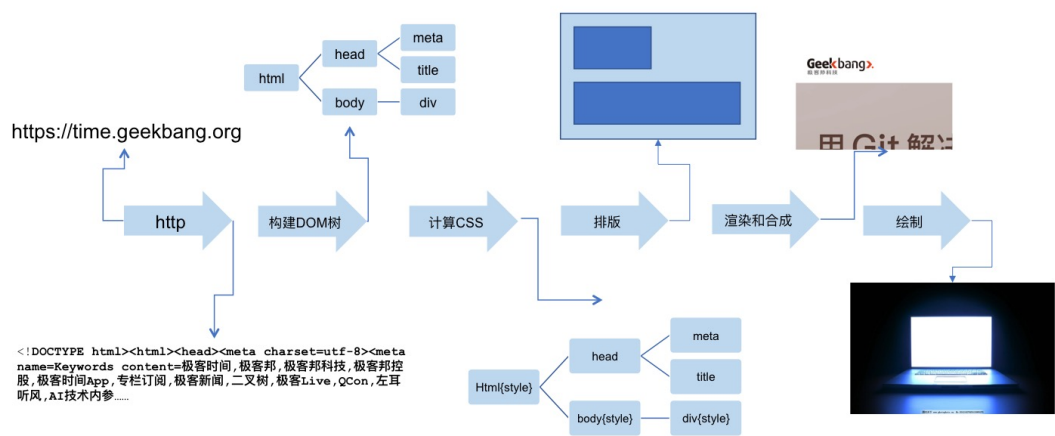
在我们的课程中，我也会控制浏览器相关知识的粒度，把它保持在“给前端工程师了解浏览器”的水准，而不是详细到“给浏览器开发工程师实现浏览器”的水准。

那么，我们今天开始，来共同思考一下。一个浏览器到底是如何工作的。

实际上，对浏览器的实现者来说，他们做的事情，就是把一个 URL 变成一个屏幕上显示的网页。

这个过程是这样的：

- 1. 浏览器首先使用 HTTP 协议或者 HTTPS 协议，向服务端请求页面；
- 2. 把请求回来的 HTML 代码经过解析，构建成 DOM 树；
- 3. 计算 DOM 树上的 CSS 属性；
- 4. 最后根据 CSS 属性对元素逐个进行渲染，得到内存中的位图；
- 5. 一个可选的步骤是对位图进行合成，这会极大地增加后续绘制的速度；
- 6. 合成之后，再绘制到界面上。



我们在开始详细介绍之前，要建立一个感性认识。我们从 HTTP 请求回来开始，这个过程并非一般想象中的一步做完再做下一步，而是一条流水线。

从 HTTP 请求回来，就产生了流式的数据，后续的 DOM 树构建、CSS 计算、渲染、合成、绘制，都是尽可能地流式处理前一步的产出：即不需要等到上一步骤完全结束，就开始处理上一步的输出，这样我们在浏览网页时，才会看到逐步出现的页面。

首先我们来介绍下网络通讯的部分。

## HTTP 协议

浏览器首先要做的事就是根据 URL 把数据取回来，取回数据使用的是 HTTP 协议，实际上这个过程之前还有 DNS 查询，不过这里就不详细展开了。

我先来了解下 HTTP 的标准。

HTTP 标准由 IETF 组织制定，跟它相关的标准主要有两份：

HTTP1.1 [🔗https://tools.ietf.org/html/rfc2616](https://tools.ietf.org/html/rfc2616)

HTTP1.1 [🔗https://tools.ietf.org/html/rfc7234](https://tools.ietf.org/html/rfc7234)

HTTP 协议是基于 TCP 协议出现的，对 TCP 协议来说，TCP 协议是一条双向的通讯通道，HTTP 在 TCP 的基础上，规定了 Request-Response 的模式。这个模式决定了通讯必定是由浏览器端首先发起的。

大部分情况下，浏览器的实现者只需要用一个 TCP 库，甚至一个现成的 HTTP 库就可以搞定浏览器的网络通讯部分。HTTP 是纯粹的文本协议，它是规定了使用 TCP 协议来传输文本格式的一个应用层协议。


下面，我们试着用一个纯粹的 TCP 客户端来手工实现 HTTP 一下：

## 实验

我们的实验需要使用 telnet 客户端，这个客户端是一个纯粹的 TCP 连接工具（安装方法）。


首先我们运行 telnet，连接到极客时间主机，在命令行里输入以下内容：

```
1 telnet time.geekbang.org 80
```


 复制代码

这个时候，TCP 连接已经建立，我们输入以下字符作为请求：

```
1 GET / HTTP/1.1
2 Host: time.geekbang.org
```

 复制代码

按下两次回车，我们收到了服务端的回复：

 复制代码

```
1 HTTP/1.1 301 Moved Permanently
2 Date: Fri, 25 Jan 2019 13:28:12 GMT
3 Content-Type: text/html
4 Content-Length: 182
5 Connection: keep-alive
6 Location: https://time.geekbang.org/
7 Strict-Transport-Security: max-age=15768000
8
9 <html>
10 <head><title>301 Moved Permanently</title></head>
11 <body bgcolor="white">
12 <center><h1>301 Moved Permanently</h1></center>
13 <hr><center>openresty</center>
14 </body>
15 </html>
```

这就是一次完整的 HTTP 请求的过程了，我们可以看到，在 TCP 通道中传输的，完全是文本。

在请求部分，第一行被称作 request line，它分为三个部分，HTTP Method，也就是请求的“方法”，请求的路径和请求的协议和版本。

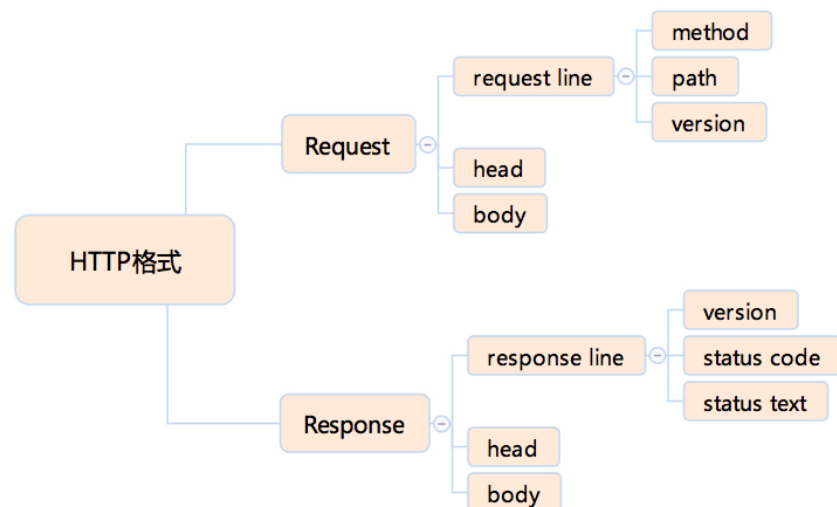
在响应部分，第一行被称作 response line，它也分为三个部分，协议和版本、状态码和状态文本。

紧随在 request line 或者 response line 之后，是请求头 / 响应头，这些头由若干行组成，每行是用冒号分隔的名称和值。

在头之后，以一个空行（两个换行符）为分隔，是请求体 / 响应体，请求体可能包含文件或者表单数据，响应体则是 HTML 代码。

## HTTP 协议格式

根据上面的分析，我们可以知道 HTTP 协议，大概可以划分成如下部分。



我们简单看一下，在这些部分中，path 是请求的路径完全由服务端来定义，没有很多的特别内容；而 version 几乎都是固定字符串；response body 是我们最熟悉的 HTML，我在后面会有专门的课程介绍，这里也就不多讲了。

下面我们就来逐个给你介绍其它部分。

## HTTP Method（方法）

我们首先来介绍一下 request line 里面的方法部分。这里的方法跟我们编程中的方法意义类似，表示我们此次 HTTP 请求希望执行的操作类型。方法有以下几种定义：

GET

POST

HEAD

PUT

DELETE

CONNECT

OPTIONS

TRACE

浏览器通过地址栏访问页面都是 GET 方法。表单提交产生 POST 方法。

HEAD 则是跟 GET 类似，只返回请求头，多数由 JavaScript 发起

PUT 和 DELETE 分别表示添加资源和删除资源，但是实际上这只是语义上的一种约定，并没有强约束。

CONNECT 现在多用于 HTTPS 和 WebSocket。

OPTIONS 和 TRACE 一般用于调试，多数线上服务都不支持。

## HTTP Status code（状态码）和 Status text（状态文本）

接下来我们看看 response line 的状态码和状态文本。常见的状态码有以下几种。

1xx：临时回应，表示客户端请继续。

2xx：请求成功。

- 200：请求成功。

3xx：表示请求的目标有变化，希望客户端进一步处理。

- 301&302：永久性与临时性跳转。

- 304：跟客户端缓存没有更新。

4xx：客户端请求错误。

- 403：无权限。

- 404：表示请求的页面不存在。

- 418：It's a teapot. 这是一个彩蛋，来自 ietf 的一个愚人节玩笑。（[☞ 超文本咖啡壶控制协议](#)）

5xx：服务端请求错误。

- 500：服务端错误。

- 503：服务端暂时性错误，可以一会再试。

对我们前端来说，1xx 系列的状态码是非常陌生的，原因是 1xx 的状态被浏览器 HTTP 库直接处理掉了，不会让上层应用知晓。

2xx 系列的状态最熟悉的的就是 200，这通常是网页请求成功的标志，也是大家最喜欢的状态码。

3xx 系列比较复杂，301 和 302 两个状态表示当前资源已经被转移，只不过一个是永久性转移，一个是临时性转移。实际上 301 更接近于一种报错，提示客户端下次别来了。

304 又是一个每个前端必知必会的状态，产生这个状态的前提是：客户端本地已经有缓存的版本，并且在 Request 中告诉了服务端，当服务端通过时间或者 tag，发现没有更新的时候，就会返回一个不含 body 的 304 状态。

## HTTP Head (HTTP 头)

HTTP 头可以看作一个键值对。原则上，HTTP 头也是一种数据，我们可以自由定义 HTTP 头和值。不过在 HTTP 规范中，规定了一些特殊的 HTTP 头，我们现在就来了解一下它们。

在 HTTP 标准中，有完整的请求 / 响应头规定，这里我们挑几个重点的说一下：

我们先来看看 Request Header。

Request Header	规定
Accept	浏览器端接受的格式。
Accept-Encoding:	浏览器端接收的编码方式。
Accept-Language	浏览器端接受的语言，用于服务端判断多语言。
Cache-Control	控制缓存的时效性。
Connection	连接方式，如果是keep-alive，且服务端支持，则会复用连接。
Host	HTTP访问使用的域名。
If-Modified-Since	上次访问时的更改时间，如果服务端认为此时间后自己没有更新，则会给出304响应。
If-None-Match	次访问时使用的E-Tag，通常是页面的信息摘要，这个比更改时间更准确一些。
User-Agent	客户端标识，因为一些历史原因，这是一笔糊涂账，多数浏览器的这个字段都十分复杂，区别十分微妙。
Cookie	客户端存储的cookie字符串。

接下来看一下 Response Header。



Response Header	规定
Cache-Control	缓存控制，用于通知各级缓存保存的时间，例如max-age=0，表示不要缓存。
Connection	连接类型，Keep-Alive表示复用连接。
Content-Encoding	内容编码方式，通常是gzip。
Content-Length	内容的长度，有利于浏览器判断内容是否已经结束。
Content-Type	内容类型，所有请求网页的都是text/html。
Date	当前的服务器日期。
ETag	页面的信息摘要，用于判断是否需要重新到服务端取回页面。
Expires	过期时间，用于判断下次请求是否需要到服务端取回页面。
Keep-Alive	保持连接不断时需要的一些信息，如timeout=5, max=100。
Last-Modified	页面上次修改的时间。
Server	服务端软件的类型。
Set-Cookie	设置cookie，可以存在多个。
Via	服务端的请求链路，对一些调试场景至关重要的一个头。

这里仅仅列出了我认为比较常见的 HTTP 头，这些头是我认为前端工程师应该做到不需要查阅，看到就可以知道意思的 HTTP 头。完整的列表还是请你参考我给出的 rfc2616 标准。

## HTTP Request Body

HTTP 请求的 body 主要用于提交表单场景。实际上，HTTP 请求的 body 是比较自由的，只要浏览器端发送的 body 服务端认可就可以了。一些常见的 body 格式是：

- application/json
- application/x-www-form-urlencoded
- multipart/form-data
- text/xml

我们使用 HTML 的 form 标签提交产生的 HTML 请求，默认会产生 application/x-www-form-urlencoded 的数据格式，当有文件上传时，则会使用 multipart/form-data。

## HTTPS

在 HTTP 协议的基础上，HTTPS 和 HTTP2 规定了更复杂的内容，但是它基本保持了 HTTP 的设计思想，即：使用上的 Request-Response 模式。



我们首先来了解下 HTTPS。HTTPS 有两个作用，一是确定请求的目标服务端身份，二是保证传输的数据不会被网络中间节点窃听或者篡改。

HTTPS 的标准也是由 RFC 规定的，你可以查看它的详情链接：

🔗 <https://tools.ietf.org/html/rfc2818>

HTTPS 是使用加密通道来传输 HTTP 的内容。但是 HTTPS 首先与服务端建立一条 TLS 加密通道。TLS 构建于 TCP 协议之上，它实际上是对传输的内容做一次加密，所以从传输内容上看，HTTPS 跟 HTTP 没有任何区别。

## HTTP 2

HTTP 2 是 HTTP 1.1 的升级版，你可以查看它的详情链接。

🔗 <https://tools.ietf.org/html/rfc7540>

HTTP 2.0 最大的改进有两点，一是支持服务端推送，二是支持 TCP 连接复用。

服务端推送能够在客户端发送第一个请求到服务端时，提前把一部分内容推送给客户端，放入缓存当中，这可以避免客户端请求顺序带来的并行度不高，从而导致的性能问题。

TCP 连接复用，则使用同一个 TCP 连接来传输多个 HTTP 请求，避免了 TCP 连接建立时的三次握手开销，和初建 TCP 连接时传输窗口小的问题。

Note: 其实很多优化涉及更下层的协议。IP 层的分包情况，和物理层的建连时间是需要被考虑的。

## 结语

在这一节内容中，我们一起学习了浏览器的第一步工作，也就是“浏览器首先使用 HTTP 协议或 HTTPS 协议，向服务端请求页面”的这一过程。

在这个过程中，掌握 HTTP 协议是重中之重。我从一个小实验开始，带你体验了一次完整的 HTTP 请求过程。我们一起先分析了 HTTP 协议的结构。接下来，我分别介绍了 HTTP

方法、HTTP 状态码和状态文本、HTTP Head 和 HTTP Request Body 几个重点需要注意的部分。

最后，我还介绍了 HTTPS 和 HTTP 2 这两个补充版本，以便你可以更好地熟悉并理解新的特性。

你在工作中，是否已经开始使用 HTTPS 和 HTTP 2 协议了呢？用到了它们的哪些特性，请留言告诉我吧。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言(29)



CC

现在工作中暂时只使用到 HTTPS。

去年，看到一些公司已经开始着手更新 HTTP2，不少敬佩的前端高手做了分享。

他们利用 HTTP2 服务端推送，以及 TCP 连接复用的特性，把 CSS 和 JS 文件分拆成更小的文件，分批下载。

更小的文件意味着用户可以更快看到页面，以及性能的改善。与此同时，这种处理方式也对生产环境的工具链带来调整 and 改变。

很有意思的变化。

2019-02-09



95



yy不会笑

自己可以扩展补充以下

> DNS查询得到IP

> tcp/ip的并发限制

> get和post的区别

> 五层因特网协议栈

> 长连接与短连接

> http2.0与http1.1的显著不同点:

> 强缓存与协商缓存

2019-02-09



50



徐言

老师真的很喜欢猫呐

2019-02-09



16



一步

HTT2 还有一个很重要的特性：使用二进制代理文本进行传输，极大提高了传输的效率

作者回复: 嗯 是 这块每太细讲 我觉得大家了解即可 主要是服务端的工作。

2019-02-23



13



oillie

h2的头部会用哈夫曼编码压缩大小

2019-02-09



9



umaru

老师，浏览器渲染听说有重绘，回流什么的，哪里可以找到这一部分相关资料？

2019-02-09



1

9



新哥

option请求在线上也会存在吧？非简单请求中的预请求

2019-02-26



1



8



hhk

那么 HTTP2 的 TCP 连接复用, 和 keep-alive 有什么区别呢？

作者回复: 嗯，这块我没细讲，其实就是HTTP2要加这个头才能复用连接。

2019-02-17



7



瞧，这个人

当5G来临，http小优化都不用找了

作者回复: 这完全是一种误解，物理层优化没法解决上层协议的问题。

2019-02-16



1



7



Shaoyao·琚

棒！已经开始阅览 HTTP3 的文档了😄

2019-02-09



7



温木

看了您的流式数据的解释，是不是可以理解当看到页面渲染的时候，DOM是不是有可能都还没构建完成呢？即使是理论上，有这种可能吗？

在网上看别人写的网页渲染原理，都是说dom构建完成了，再和构建css，都完成了才进行渲染的。

老师能详细解释下吗？

2019-04-28



1



5



Russell

关于缓存，304这块我整理了一片文章。<https://github.com/XiaodongTong/blog/blob/master/%E6%B5%8F%E8%A7%88%E5%99%A8/%E7%BC%93%E5%AD%98%E6%9C%BA%E5%88%B6.md>

2019-04-11



3



YY

软件有bug，听上一个音频后切换下一个音频时，下一个音频开始播放的时间不是从零开始，是上一个音频的暂停时间点开始

2019-03-23



3



芳玥

3xx状态码，真的是给我补了一下。从来没踩过这个坑。

2019-02-23



3



维维

找到了与老师的共同爱好，喜欢猫。

2019-02-12



3



索

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title></title>
<style>
/* var()函数 */
:root {
  --bg-color: #cdcdcd;
  --font-color: blue;
}
body {
  background-color: var(--bg-color);
}
.title {
```

```
color: var(--font-color);
}
/* attr()函数 */
p:before {
  content: attr(data-foo) " ";
}
/* max()函数 */
.test-max {
  height: 30px;
  background-color: #f00;
  width: max(30px, 50px);
}
/* toggle()函数 */
ul {
  list-style-type: disk;
}
li > ul {
  list-style-type: toggle(disk, circle, square, box);
}
</style>
</head>
<body>
<h1 class="title">标题引人注目</h1>
<p data-foo="hello">world</p>
<div class="test-max"></div>
<ul>
<li>1
  <ul>
    <li>1-1
      <ul>
        <li>1-1-1
          <ul>
            <li>1-1-1-1</li>
          </ul>
        </li>
      </ul>
    </li>
  </ul>
</li>
</ul>
</body>
</html>
```

max(),min(),clamp() 浏览器基本都不支持



yeoman

那位提问tag是什么的同学应该是问304请求那块提到的tag，温大指的是协商缓存中的etag（entity tag

2019-03-09



2



...

老师，网站做https升级后，由于TLS加密，第一次连接，速度慢一点是可以理解，但是之后的连接速度也变得挺慢的，这个用户体验不好，这个是不是跟我申请的https证书有很大关系，除此之外，有没有好的建议？

作者回复: 怀疑是没有复用连接，这个凭你给的信息我没法判断，得具体拿log分析时间都花在哪里了，一般来说跟证书关系不大。

2019-02-17



2



苗

请教老师：前端页面发送跨域ajax请求时，能正常发送和接受返回数据，但通过Chrome发现Request Header有警告信息：Provisional headers are shown；请问老师有什么解决的方法呢？

我的ajax代码如下：

```
$.ajax({
  method : 'POST',
  xhrFields: {
    withCredentials: true
  },
  crossDomain:true, //允许跨域
  contentType : 'application/json',
  dataType : 'json',
  url : 'http://localhost:8083/add',
  data:JSON.stringify({
    custCode:'icbc11',
    custName:'工商银行11',
    custAbbrEnglish:'icbc bank11'
  }),
  success : function(result){
    console.log(result);
  }
});
```



```
}  
});
```

2019-06-30



👍 1



Helen

为什么对位图的合成是一个可选的过程，浏览器的composite 不是一定会执行的么，我以为这一步是无法不执行的

作者回复: 可以不合成直接画，效率低而已。  
当然现代浏览器是不可能跳过这一步的。

2019-06-29



👍 1