

## 03-HTTP请求流程：为什么很多站点第二次打开速度会很快？

在[上一篇文章](#)中我介绍了TCP协议是如何保证数据完整传输的，相信你还记得，一个TCP连接过程包括了建立连接、传输数据和断开连接三个阶段。

而HTTP协议，正是建立在TCP连接基础之上的。**HTTP是一种允许浏览器向服务器获取资源的协议，是Web的基础**，通常由浏览器发起请求，用来获取不同类型的文件，例如HTML文件、CSS文件、JavaScript文件、图片、视频等。此外，**HTTP也是浏览器使用最广的协议**，所以要想学好浏览器，就要先深入了解HTTP。

不知道你是否有过下面这些疑问：

1. 为什么通常在第一次访问一个站点时，打开速度很慢，当再次访问这个站点时，速度就很快了？
2. 当登录过一个网站之后，下次再访问该站点，就已经处于登录状态了，这是怎么做到的呢？

这一切的秘密都隐藏在HTTP的请求过程中。所以，在今天这篇文章中，我将通过分析一个HTTP请求过程中每一步的状态来带你了解完整的HTTP请求过程，希望你看完这篇文章后，能够对HTTP协议有个全新的认识。

### 浏览器端发起HTTP请求流程

如果你在浏览器地址栏里键入极客时间网站的地址：<http://time.geekbang.org/index.html>，那么接下来，浏览器会完成哪些动作呢？下面我们就一步一步详细“追踪”下。

#### 1. 构建请求

首先，浏览器构建**请求行**信息（如下所示），构建好后，浏览器准备发起网络请求。

```
GET /index.html HTTP1.1
```

#### 2. 查找缓存

在真正发起网络请求之前，浏览器会先在浏览器缓存中查询是否有要请求的文件。其中，**浏览器缓存是一种在本地保存资源副本，以供下次请求时直接使用**的技术。

当浏览器发现请求的资源已经在浏览器缓存中存有副本，它会拦截请求，返回该资源的副本，并直接结束请求，而不会再去源服务器重新下载。这样做的好处有：

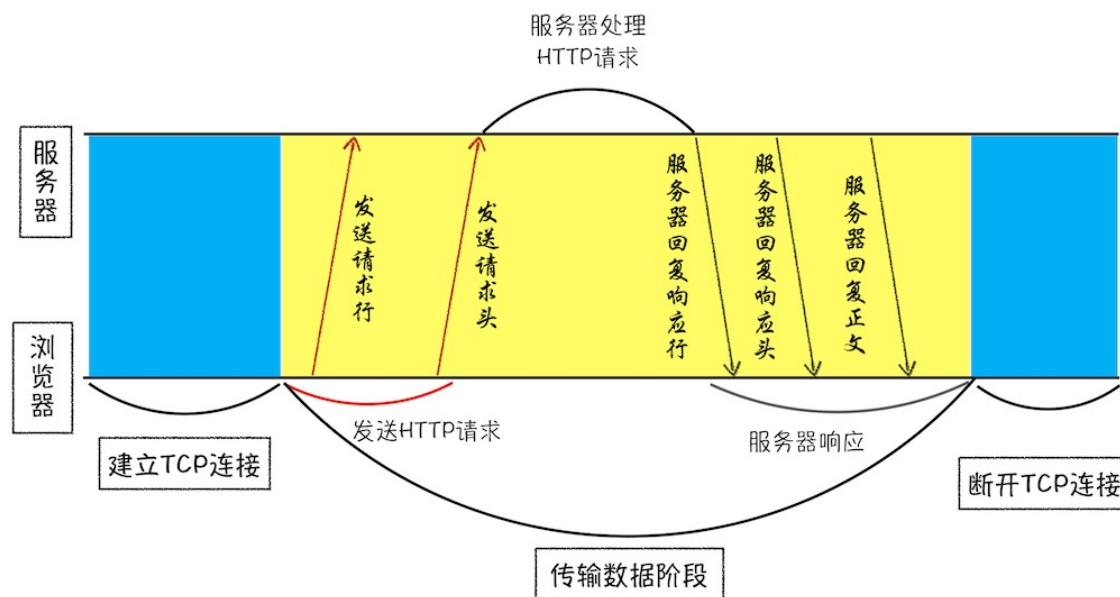
- 缓解服务器端压力，提升性能（获取资源的耗时更短了）；
- 对于网站来说，缓存是实现快速资源加载的重要组成部分。

当然，如果缓存查找失败，就会进入网络请求过程了。

#### 3. 准备IP地址和端口

不过，先不急，在了解网络请求之前，我们需要先看看HTTP和TCP的关系。因为浏览器使用**HTTP协议作为**

**应用层协议**，用来封装请求的文本信息；并使用**TCP/IP作传输层协议**将它发到网络上，所以在HTTP工作开始之前，浏览器需要通过TCP与服务器建立连接。也就是说**HTTP的内容是通过TCP的传输数据阶段来实现的**，你可以结合下图更好地理解这二者的关系。



TCP和HTTP的关系示意图

那接下来你可以思考这么“一连串”问题：

- HTTP网络请求的第一步是做什么呢？结合上图看，是和服务器建立TCP连接。
- 那建立连接的信息都有了？[上一篇文章](#)中，我们讲到建立TCP连接的第一步就是需要准备IP地址和端口号。
- 那怎么获取IP地址和端口号呢？这得看看我们现在有什么，我们有一个URL地址，那么是否可以利用URL地址来获取IP和端口信息呢？

在[上一篇文章](#)中，我们介绍过数据包都是通过IP地址传输给接收方的。由于IP地址是数字标识，比如极客时间网站的IP是39.106.233.176, 难以记忆，但使用极客时间的域名（time.geekbang.org）就好记多了，所以基于这个需求又出现了一个服务，负责把域名和IP地址做一一映射关系。这套域名映射为IP的系统就叫做“**域名系统**”，简称**DNS**（Domain Name System）。

所以，这样一路推导下来，你会发现在**第一步浏览器会请求DNS返回域名对应的IP**。当然浏览器还提供了**DNS数据缓存服务**，如果某个域名已经解析过了，那么浏览器会缓存解析的结果，以供下次查询时直接使用，这样也会减少一次网络请求。

拿到IP之后，接下来就需要获取端口号了。通常情况下，如果URL没有特别指明端口号，那么HTTP协议默认是80端口。

## 4. 等待TCP队列

现在已经把端口和IP地址都准备好了，那么下一步是不是可以建立TCP连接了呢？

答案依然是“不行”。Chrome有个机制，同一个域名同时最多只能建立6个TCP连接，如果在同一个域名下

同时有10个请求发生，那么其中4个请求会进入排队等待状态，直至进行中的请求完成。

当然，如果当前请求数量少于6，会直接进入下一步，建立TCP连接。

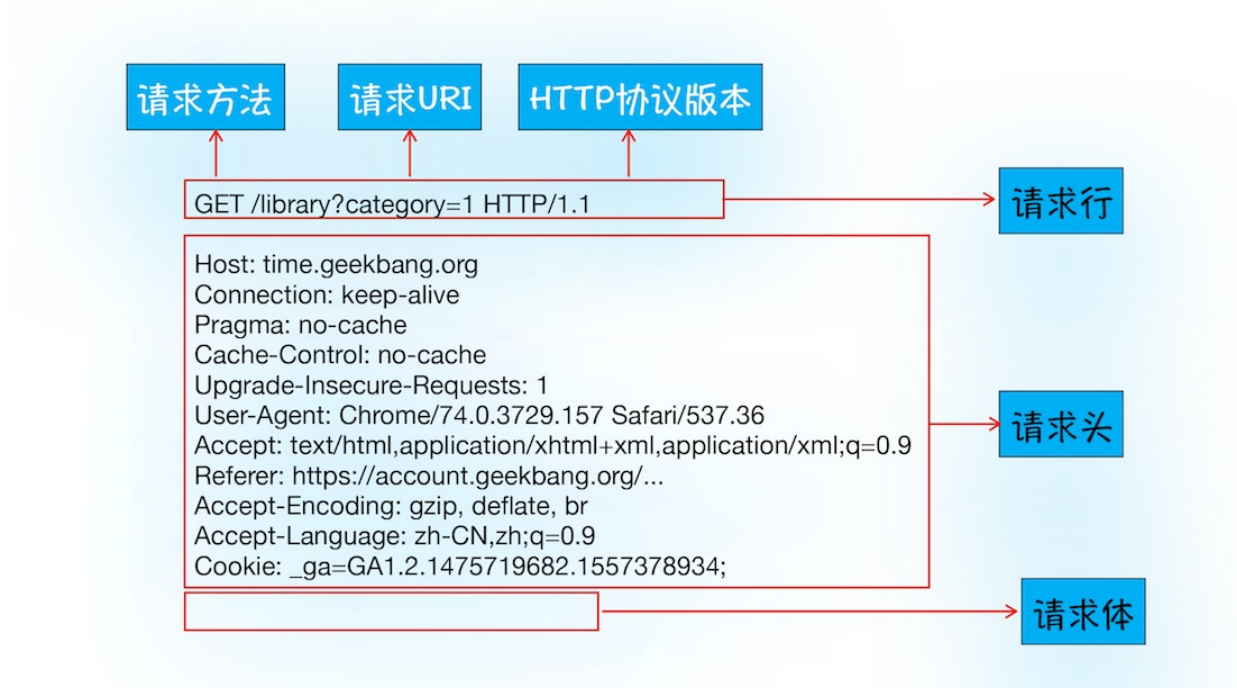
## 5. 建立TCP连接

排队等待结束之后，终于可以快乐地和服务器握手了，在HTTP工作开始之前，浏览器通过TCP与服务器建立连接。而TCP的工作方式，我在[上一篇文章](#)中已经做过详细介绍了，如果有必要，你可以自行回顾下，这里我就不再重复讲述了。

## 6. 发送HTTP请求

一旦建立了TCP连接，浏览器就可以和服务器进行通信了。而HTTP中的数据正是在这个通信过程中传输的。

你可以结合下图来理解，浏览器是如何发送请求信息给服务器的。



HTTP请求数据格式

首先浏览器会向服务器发送**请求行**，它包括了**请求方法**、**请求URI (Uniform Resource Identifier)** 和**HTTP版本协议**。

发送请求行，就是告诉服务器浏览器需要什么资源，最常用的请求方法是**Get**。比如，直接在浏览器地址栏键入极客时间的域名（time.geekbang.org），这就是告诉服务器要Get它的首页资源。

另外一个常用的请求方法是**POST**，它用于发送一些数据给服务器，比如登录一个网站，就需要通过POST方法把用户信息发送给服务器。如果使用POST方法，那么浏览器还要准备数据给服务器，这里准备的数据是通过**请求体**来发送。

在浏览器发送请求行命令之后，还要以**请求头**形式发送其他一些信息，把浏览器的一些基础信息告诉服务

器。比如包含了浏览器所使用的操作系统、浏览器内核等信息，以及当前请求的域名信息、浏览器端的Cookie信息，等等。

## 服务器端处理HTTP请求流程

历经千辛万苦，HTTP的请求信息终于被送达了服务器。接下来，服务器会根据浏览器的请求信息来准备相应的内容。

### 1. 返回请求

一旦服务器处理结束，便可以返回数据给浏览器了。你可以通过工具软件curl来查看返回请求数据，具体使用方法是在命令行中输入以下命令：

```
curl -i https://time.geekbang.org/
```

注意这里加上了-i是为了返回响应行、响应头和响应体的数据，返回的结果如下图所示，你可以结合这些数据来理解服务器是如何响应浏览器的。



服务器响应的数据格式

首先服务器会返回**响应行**，包括协议版本和状态码。

但并不是所有的请求都可以被服务器处理的，那么一些无法处理或者处理出错的信息，怎么办呢？服务器会通过请求行的**状态码**来告诉浏览器它的处理结果，比如：

- 最常用的状态码是200，表示处理成功；
- 如果没有找到页面，则会返回**404**。

状态码类型很多，这里我就不过多介绍了，网上有很多资料，你可以自行查询和学习。

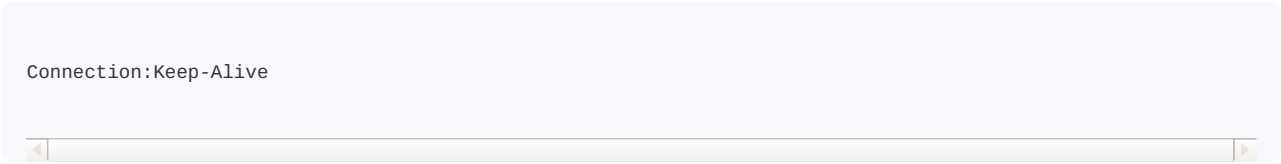
随后，正如浏览器会随同请求发送请求头一样，服务器也会随同响应向浏览器发送**响应头**。响应头包含了服务器自身的一些信息，比如服务器生成返回数据的时间、返回的数据类型（JSON、HTML、流媒体等类型），以及服务器要在客户端保存的Cookie等信息。

发送完响应头后，服务器就可以继续发送**响应体**的数据，通常，响应体就包含了HTML的实际内容。

以上这些就是服务器响应浏览器的具体过程。

## 2. 断开连接

通常情况下，一旦服务器向客户端返回了请求数据，它就要关闭 TCP 连接。不过如果浏览器或者服务器在其头信息中加入了：



```
Connection: Keep-Alive
```

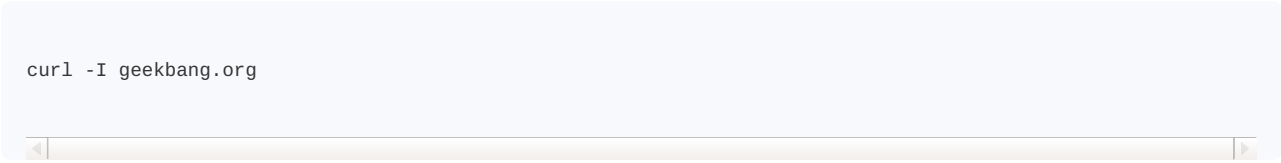
那么TCP连接在发送后将仍然保持打开状态，这样浏览器就可以继续通过同一个TCP连接发送请求。**保持TCP连接可以省去下次请求时需要建立连接的时间，提升资源加载速度**。比如，一个Web页面中内嵌的图片就都来自同一个Web站点，如果初始化了一个持久连接，你就可以复用该连接，以请求其他资源，而不需要重新再建立新的TCP连接。

## 3. 重定向

到这里似乎请求流程快结束了，不过还有一种情况你需要了解下，比如当你在浏览器中打开geekbang.org后，你会发现最终打开的页面地址是 <https://www.geekbang.org>。

这两个URL之所以不一样，是因为涉及到了一个**重定向操作**。跟前面一样，你依然可以使用curl来查看下请求geekbang.org 会返回什么内容？

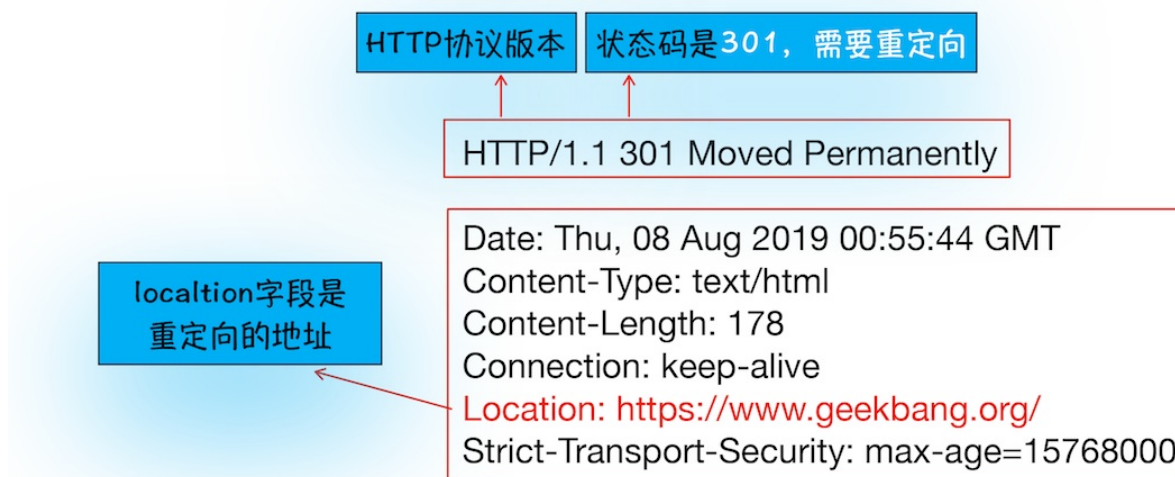
在控制台输入如下命令：



```
curl -I geekbang.org
```

注意这里输入的参数是-I，和-i不一样，-I表示只需要获取响应头和响应行数据，而不需要获取响应体的数据，最终返回的数据如下图所示：





服务器返回响应行和响应头（含重定向格式）

从图中你可以看到，响应行返回的状态码是301，状态301就是告诉浏览器，我需要重定向到另外一个网址，而需要重定向的网址正是包含在响应头的Location字段中，接下来，浏览器获取Location字段中的地址，并使用该地址重新导航，这就是一个完整重定向的执行流程。这也就解释了为什么输入的是geekbang.org，最终打开的却是 <https://www.geekbang.org/> 了。

不过也不要认为这种跳转是必然的。如果你打开 <https://12306.cn>，你会发现这个站点是打不开的。这是因为12306的服务器并没有处理跳转，所以必须要手动输入完整的 <https://www.12306.com> 才能打开页面。

## 问题解答

说了这么多，相信你现在已经了解了HTTP的请求流程，那现在我们再回过头来看看文章开头提出的问题。

### 1. 为什么很多站点第二次打开速度会很快？

如果第二次页面打开很快，主要原因是第一次加载页面过程中，缓存了一些耗时的数据。

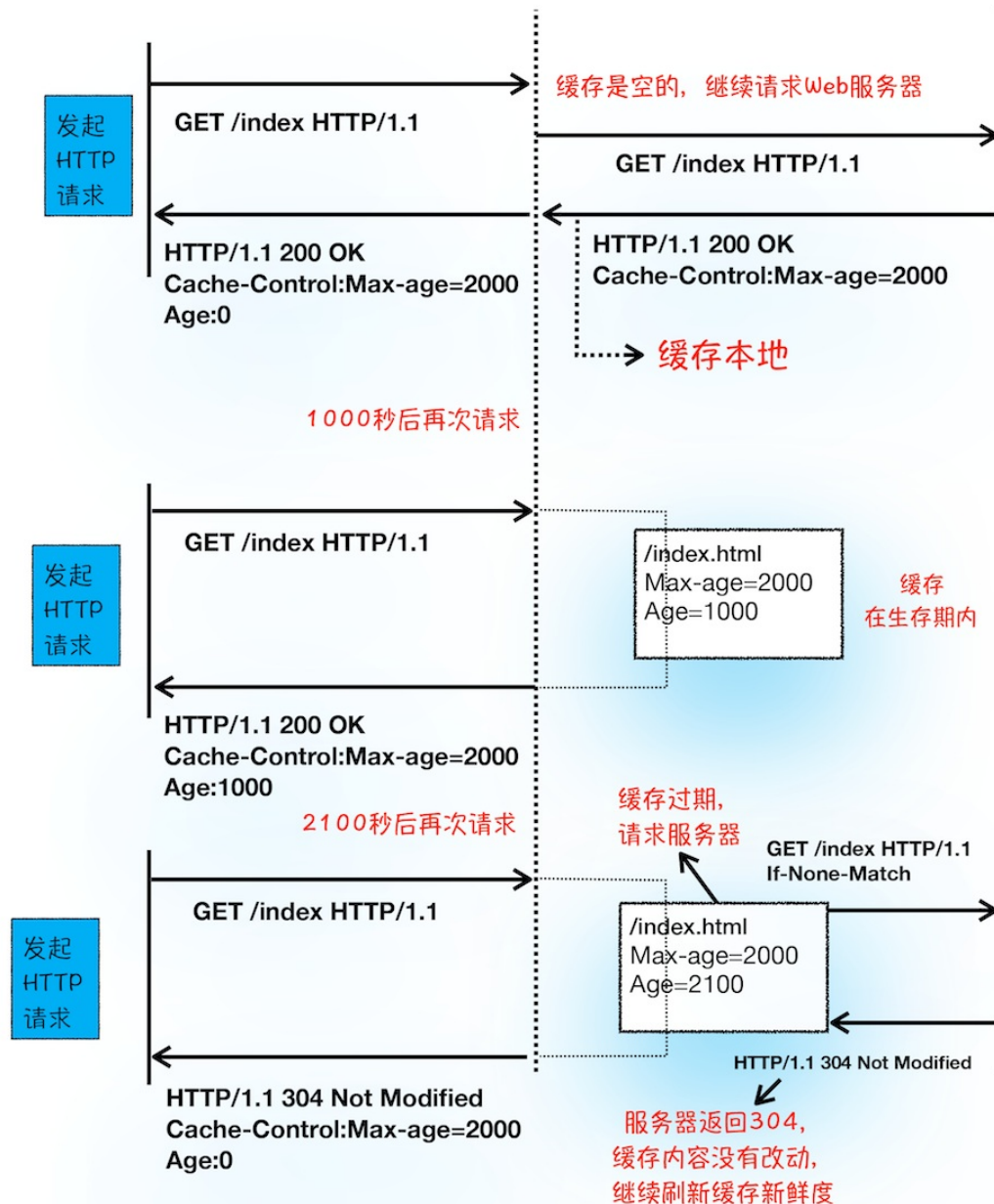
那么，哪些数据会被缓存呢？从上面介绍的核心请求路径可以发现，**DNS缓存**和**页面资源缓存**这两块数据是会被浏览器缓存的。其中，DNS缓存比较简单，它主要就是在浏览器本地把对应的IP和域名关联起来，这里就不做过多分析了。

我们重点看下浏览器资源缓存，下面是缓存处理的过程：

浏览器

Cache

Web服务器



缓存查找流程示意图

首先，我们看下服务器是通过什么方式让浏览器缓存数据的？

从上图的第一次请求可以看出，当服务器返回**HTTP响应头**给浏览器时，浏览器是**通过响应头中的Cache-Control字段来设置是否缓存该资源**。通常，我们还需要为这个资源设置一个缓存过期时长，而这个时长是通过Cache-Control中的Max-age参数来设置的，比如上图设置的缓存过期时间是2000秒。

```
Cache-Control:Max-age=2000
```

这也就意味着，在该缓存资源还未过期的情况下，如果再次请求该资源，会直接返回缓存中的资源给浏览器。

但如果缓存过期了，浏览器则会继续发起网络请求，并且在**HTTP请求头**中带上：

```
If-None-Match: "4f80f-13c-3a1xb12a"
```

服务器收到请求头后，会根据If-None-Match的值来判断请求的资源是否有更新。

- 如果没有更新，就返回304状态码，相当于服务器告诉浏览器：“这个缓存可以继续使用，这次就不重复发送数据给你了。”
- 如果资源有更新，服务器就直接返回最新资源给浏览器。

关于缓存的细节内容特别多，具体细节你可以参考这篇 [HTTP缓存](#)，在这里我就不赘述了。

简要来说，很多网站第二次访问能够秒开，是因为这些网站把很多资源都缓存在了本地，浏览器缓存直接使用本地副本来回应请求，而不会产生真实的网络请求，从而节省了时间。同时，DNS数据也被浏览器缓存了，这又省去了DNS查询环节。

## 2. 登录状态是如何保持的？

通过上面的介绍，你已经了解了缓存是如何工作的。下面我们再一起看下登录状态是如何保持的。

- 用户打开登录页面，在登录框里填入用户名和密码，点击确定按钮。点击按钮会触发页面脚本生成用户登录信息，然后调用POST方法提交用户登录信息给服务器。
- 服务器接收到浏览器提交的信息之后，查询后台，验证用户登录信息是否正确，如果正确的话，会生成一段表示用户身份的字符串，并把该字符串写到响应头的Set-Cookie字段里，如下所示，然后把响应头发送给浏览器。

```
Set-Cookie: UID=3431uad;
```

- 浏览器在接收到服务器的响应头后，开始解析响应头，如果遇到响应头里含有Set-Cookie字段的情况，浏览器就会把这个字段信息保存到本地。比如把UID=3431uad保持到本地。
- 当用户再次访问时，浏览器会发起HTTP请求，但在发起请求之前，浏览器会读取之前保存的Cookie数据，并把数据写进请求头里的Cookie字段里（如下所示），然后浏览器再将请求头发送给服务器。

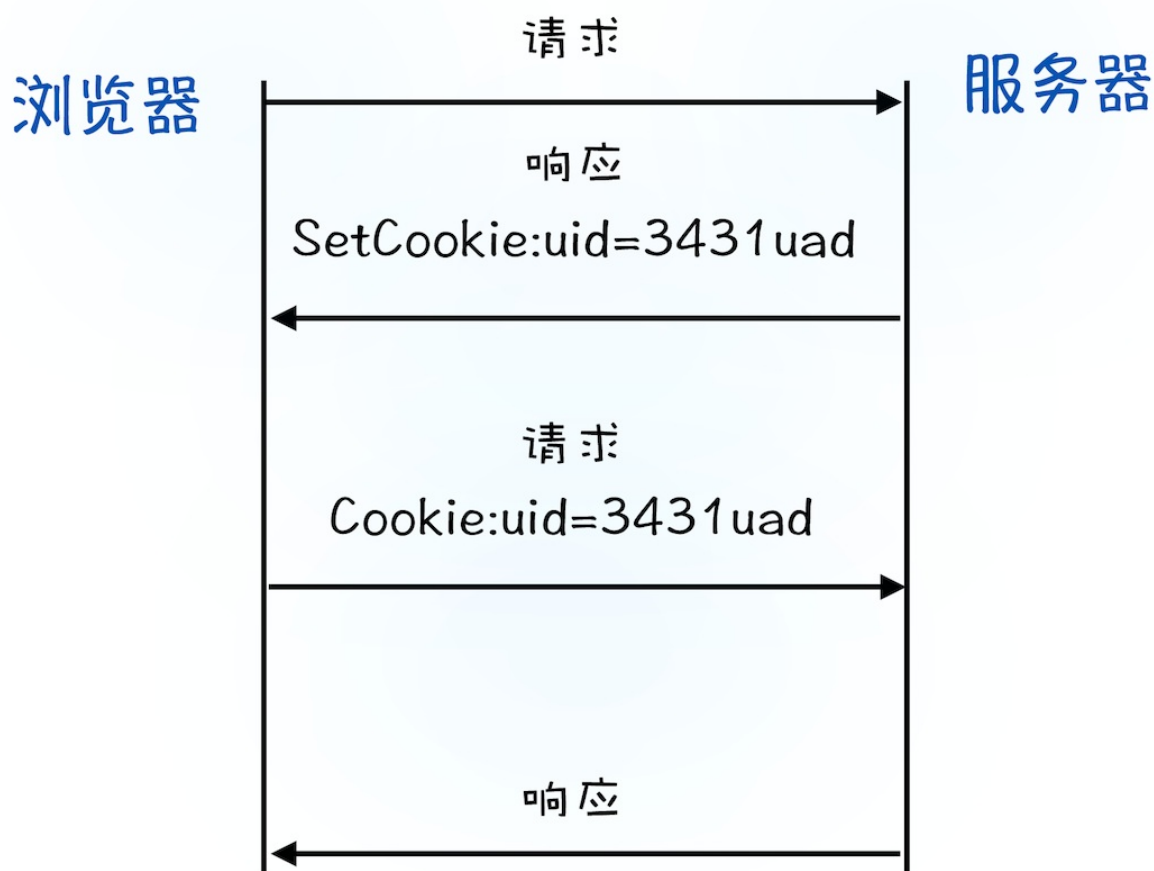
```
Cookie: UID=3431uad;
```

- 服务器在收到HTTP请求头数据之后，就会查找请求头里面的“Cookie”字段信息，当查找到包含UID=3431uad的信息时，服务器查询后台，并判断该用户是已登录状态，然后生成含有该用户信息的页面数据，并把生成的数据发送给浏览器。



- 浏览器在接收到该含有当前用户的页面数据后，就可以正确展示用户登录的状态信息了。

好了，通过这个流程你可以知道浏览器页面状态是通过使用Cookie来实现的。Cookie流程可以参考下图：



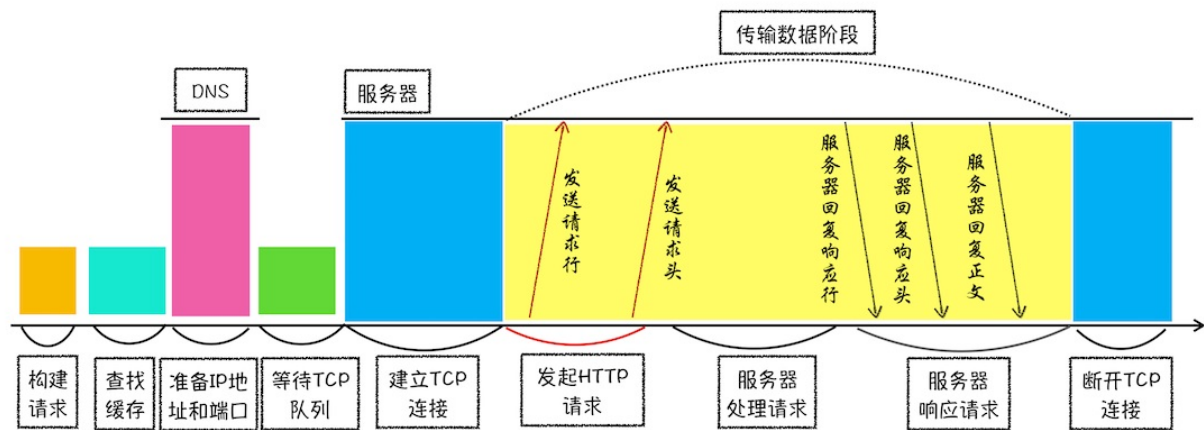
Cookie流程图

简单地说，如果服务器端发送的响应头内有 Set-Cookie 的字段，那么浏览器就会将该字段的内容保持到本地。当下次客户端再往该服务器发送请求时，客户端会自动在请求头中加入 Cookie 值后再发送出去。服务器端发现客户端发送过来的Cookie后，会去检查究竟是从哪一个客户端发来的连接请求，然后对比服务器上的记录，最后得到该用户的状态信息。

## 总结

本篇文章的内容比较多、比较碎，但是非常重要，所以我先来总结下今天的主要内容。

为了便于你理解，我画了下面这张详细的“HTTP请求示意图”，用来展现浏览器中的HTTP请求所经历各个阶段。



HTTP请求流程示意图

从图中可以看到，浏览器中的HTTP请求从发起结束一共经历了如下八个阶段：构建请求、查找缓存、准备IP和端口、等待TCP队列、建立TCP连接、发起HTTP请求、服务器处理请求、服务器返回请求和断开连接。

然后我还通过HTTP请求路径解答了两个经常会碰到的问题，一个涉及到了Cache流程，另外一个涉及到如何使用Cookie来进行状态管理。

通过今天系统的讲解，想必你已经了解了一个HTTP完整的工作流程，相信这些知识点之于你以后的学习或工作会很有帮助。

另外，你应该也看出来了本篇文章是有很多分析问题的思路在里面的。所以在学习过程中，你也要学会提问，通过最终要做什么和现在有什么，去一步步分析并提出一些问题，让疑问带领着你去学习，抓住几个本质的问题就可以学透相关知识点，让你能站在更高维度去查看整体框架。希望它能成为你的一个学习技巧吧！

## 思考时间

最后，还是留给你个思考题：结合今天所讲HTTP请求的各个阶段，如果一个页面的网络加载时间过久，你是如何分析卡在哪个阶段的？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

# 浏览器工作原理与实践

>>> 透过浏览器看懂前端本质

李兵

前盛大创新院高级研究员



新版升级：点击「🔔 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

## 精选留言：

• mfist 2019-08-10 08:03:28

1 首先猜测最可能的出问题的地方，网络传输丢包比较严重，需要不断重传。然后通过ping curl看看对应的时延高不高。

2 然后通过wireshake看看具体哪里出了问题。

3 假如别人访问很快，自己电脑很慢，就要看看自己客户端是否有问题了。

感觉平常碰到很多http问题，基本都能通过上面方式搞定 [3赞]

作者回复2019-08-10 09:37:01



• nissan 2019-08-10 06:48:09

请问老师，使用者的资讯（如UID=3431uad）是放在cookie还是localStorage中好呢？我的理解是存不存cookie是后端决定（要求set-cookie)存localStorage是前端程式控制的。是这样吗？ [2赞]

作者回复2019-08-10 09:40:14

是否要使用localStorage，还是要看具体应用场景。其实使用cookie会很方便，因为它会随着http请求头把cookie内容发送服务器，用localStorage需要重新实现数据上传和下载。

• aaron 2019-08-10 23:01:13

老师，请问https为什么能防止网络劫持？ [1赞]

作者回复2019-08-11 16:54:25

http在传输过程中是明文的，所以数据在传输过程中是能够被截获或者修改的，比如谁在你电脑上安装了一个网络拦截软件，或者你的路由器被谁安装了监听软件，甚至在网络服务提供商都有可能修改你页面的内容，基于这些原因，我们需要在传输过程中加密数据，这就是https出现的原因，即便你拦截到了请求，获取的只是加密后的数据，拿到也没有什么用。

这块在浏览器安全篇会系统介绍。

- 一步 2019-08-10 17:19:02

对于浏览器缓存地方的选择一直搞不明白其中的原理，在浏览器中访问的时候打开network面板，发现缓存的来源有的from disk有的是from memory。对于资源什么情况下缓存到硬盘什么时候缓存到内存，想请教一下老师 [1赞]

作者回复2019-08-10 18:52:48

这是浏览器的三级缓存机制，使用memory cache比disk cache 的访问速度要快，但是具体什么规则等我回头看下源码再来回答你了。

还有另外一种cache，是service worker的cache。

- Snow同學 2019-08-12 09:56:32

希望老师在后续的课程能讲解获取html，css，js后，浏览器是如何解析，并如何生成DOM树，CSS树，最后渲染显示到浏览器上的完整的过程。同时给出前端人员编写代码怎么做性能高效的指导。非常期待呢，也是奔着这个买的老师的课程。希望在指导下，能提升前端功力

作者回复2019-08-12 10:33:36

这块后内容会详细介绍

- peter 2019-08-12 09:56:30

请问：请求行和请求头是发送两次吗？从文章的文字来看，是发送两次。但我感觉是发送一次，即发送一次请求，该请求包含请求行和请求头。

作者回复2019-08-12 10:20:29

对，只发送一次。

问下文章什么地方让你感觉是发送了两次啊？

我检查下。

- Geek\_007 2019-08-12 09:10:40

分析一个访问过慢，可以打开浏览器的开发者模式，分析network的waterfull图

- 安思科 2019-08-11 11:02:50

打开network查看timing tab应该能看到一部分网络耗时情况，再厉害点的应该能用陶辉老师课中讲的方式了

- jjd 2019-08-11 02:24:01

我 h5 引入toB 业务的js sdk 链接，当他们的js 更新后，虽然我的h5 不需要发布版本，但是如何保证手机浏览器“新鲜度”的呢

- 月隐千山 2019-08-10 23:04:15

老师，在做前端页面的时候，是否可以设置当前页面是否可以被缓存，以及哪些部分可以被缓存？还是说整个缓存机制都是由浏览器自己控制的？

- Bence Zhu 2019-08-10 16:07:06

老师好，“缓存查找流程示意图”，最后一次响应到浏览器，状态码为什么不是304呀？\^o^/

作者回复2019-08-10 18:45:40

呀，这是我敲错了，多谢，我改过来

• ytd 2019-08-10 12:49:39

平时做的项目大部分都是基于java的web项目，前后端不分离的那种，在调试时打开chrome的调试工具，修改前端静态文件后我总是习惯性的“清空缓存并硬性加载”，有时就会遇到页面刷新出来的时间久的情况，但没太关注过为什么会加载这么慢。下面我尝试简单分析下卡在哪个阶段，希望老师能够指点下。开发时应用是部署在本地的，那么慢的原因就不会是网络原因，这种项目的特点就是用jsp来渲染页面，然后集成了各种前端的js库，导致如果不利用缓存的话，页面建立的tcp连接很多，需要下载的资源也多，就会导致页面加载起来比较慢，我检查了下响应头，发现没有用到connection: keep-alive这个头，那么连接也就不能复用，那么卡就主要卡在建立tcp连接、等待tcp队列上面。另外，偶尔会遇到加载很久的情况，这种情况好像是后端的服务响应过慢，这时检查浏览器的timing，就会发现TTFB很长，不过这种只是出现在：第一次启动应用或者启动后调试时偶尔会出现（感觉这种情况主要可能是在java web项目debug模式下后端需要做初始化处理之类的事情，只是猜测）。第一次应用启动时加载登录页面，后端做了跳转处理，返回302这个耗时就很长，建立连接其实时间花的不长，主要是后端迟迟不给响应（估计又是后端初始化导致的），再次加载登录页面就会好些，不会再出现302了。

• 许童童 2019-08-10 11:20:29

老师讲得好，这一讲大致讲了一个请求经历的网络过程，虽然我自己对这些知识都已经掌握并了解细节，但是之前收集资料的过程却不容易，如果早点有老师的指引就好了。

作者回复2019-08-11 00:10:20

能自己把这个流程串起来，必须点赞👍

• °Faith book 2019-08-10 09:59:44

浏览器又是怎么对待跨域请求的呢？发送真请求之前不是还有个预请求，老师可否择机插入讲解，跨域对前端的问题对于前端来说还是常见的问题

作者回复2019-08-10 10:53:33

在浏览器安全篇会系统介绍，后面的其它章节中遇到也会穿插介绍一点

• liu\_xm 2019-08-10 09:11:42

同域名只能建立6个tcp链接的话，那加载大量图片或者其他资源的时候不是很卡呢？

作者回复2019-08-10 09:36:03

是的，通常如图片这种静态资源都是直接配置到cdn上的

• Geek\_a942b2 2019-08-10 07:32:59

数据传输阶段，如果网络很慢就要更长时间传输，其中服务器处理阶段需要查询数据库也是比较长，也比较容易出错

作者回复2019-08-10 09:41:02

嗯。服务器也是一个因素

• nissan 2019-08-10 06:45:36

想請問老師使用者的資訊（UID=3431uad）是放在cookie中好還是放在localStorage好呢？Cookie:

• Meeteer 2019-08-10 01:10:33

是服务器会通过 响应行 的状态码来告诉