

04 | 导航流程：从输入URL到页面展示，这中间发生了什么？

2019-08-13 李兵

《浏览器工作原理与实践》

课程介绍 >



讲述：李兵

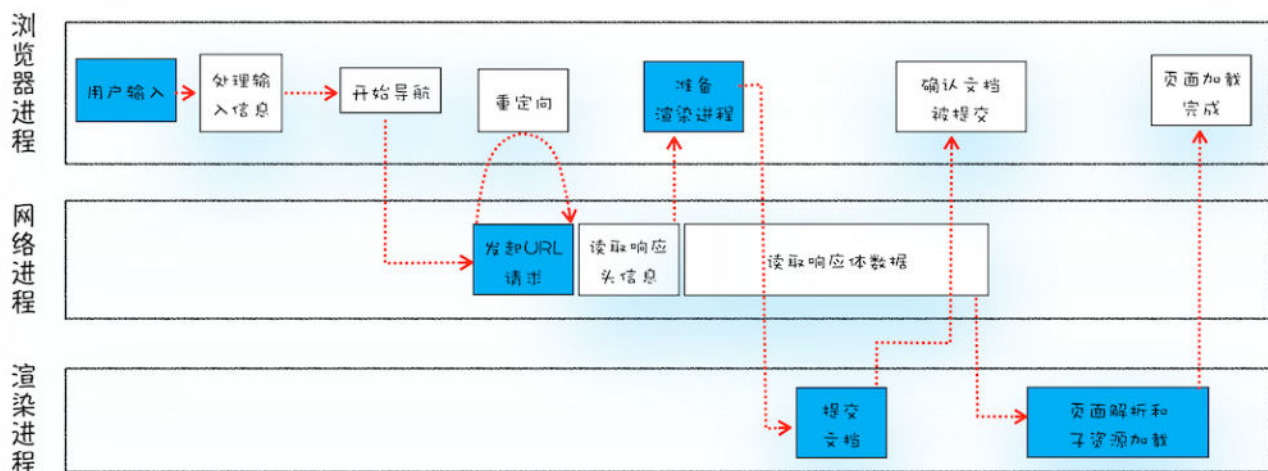
时长 15:36 大小 12.51M



“在浏览器里，从输入 URL 到页面展示，这中间发生了什么？”这是一道经典的面试题，能比较全面地考察应聘者知识的掌握程度，其中涉及到了网络、操作系统、Web 等一系列的知识。所以我在面试应聘者时也必问这道题，但遗憾的是大多数人只能回答其中部分零散的知识点，并不能将这些知识点串联成线，无法系统而又全面地回答这个问题。

那么今天我们就一起来探索下这个流程，下图是我梳理出的“从输入 URL 到页面展示完整流程示意图”：





从输入 URL 到页面展示完整流程示意图

从图中可以看出，**整个过程需要各个进程之间的配合**，所以在开始正式流程之前，我们还是先来快速回顾下浏览器进程、渲染进程和网络进程的主要职责。

- 浏览器进程主要负责用户交互、子进程管理和文件储存等功能。
- 网络进程是面向渲染进程和浏览器进程等提供网络下载功能。
- 渲染进程的主要职责是把从网络下载的 HTML、JavaScript、CSS、图片等资源解析为可以显示和交互的页面。因为渲染进程所有的内容都是通过网络获取的，会存在一些恶意代码利用浏览器漏洞对系统进行攻击，所以运行在渲染进程里面的代码是不被信任的。这也是为什么 Chrome 会让渲染进程运行在安全沙箱里，就是为了保证系统的安全。

当然，你也可以先回顾下前面的 [《01 | Chrome 架构：仅仅打开了 1 个页面，为什么有 4 个进程？》](#) 这篇文章，来全面了解浏览器多进程架构。

回顾了浏览器的进程架构后，我们再结合上图来看下这个完整的流程，可以看出，整个流程包含了许多步骤，我把其中几个核心的节点用蓝色背景标记出来了。这个过程可以大致描述为如下。

- 首先，浏览器进程接收到用户输入的 URL 请求，浏览器进程便将该 URL 转发给网络进程。
- 然后，在网络进程中发起真正的 URL 请求。
- 接着网络进程接收到了响应头数据，便解析响应头数据，并将数据转发给浏览器进程。



- 浏览器进程接收到网络进程的响应头数据之后，发送“提交导航 (CommitNavigation)”消息到渲染进程；
- 渲染进程接收到“提交导航”的消息之后，便开始准备接收 HTML 数据，接收数据的方式是直接和网络进程建立数据管道；
- 最后渲染进程会向浏览器进程“确认提交”，这是告诉浏览器进程：“已经准备好接受和解析页面数据了”。
- 浏览器进程接收到渲染进程“提交文档”的消息之后，便开始移除之前旧的文档，然后更新浏览器进程中的页面状态。

这其中，用户发出 URL 请求到页面开始解析的这个过程，就叫做导航。

从输入 URL 到页面展示

现在我们知道了浏览器几个主要进程的职责，还有在导航过程中需要经历的几个主要的阶段，下面我们就来详细分析下这些阶段，同时也就解答了开头所说的那道经典的面试题。

1. 用户输入

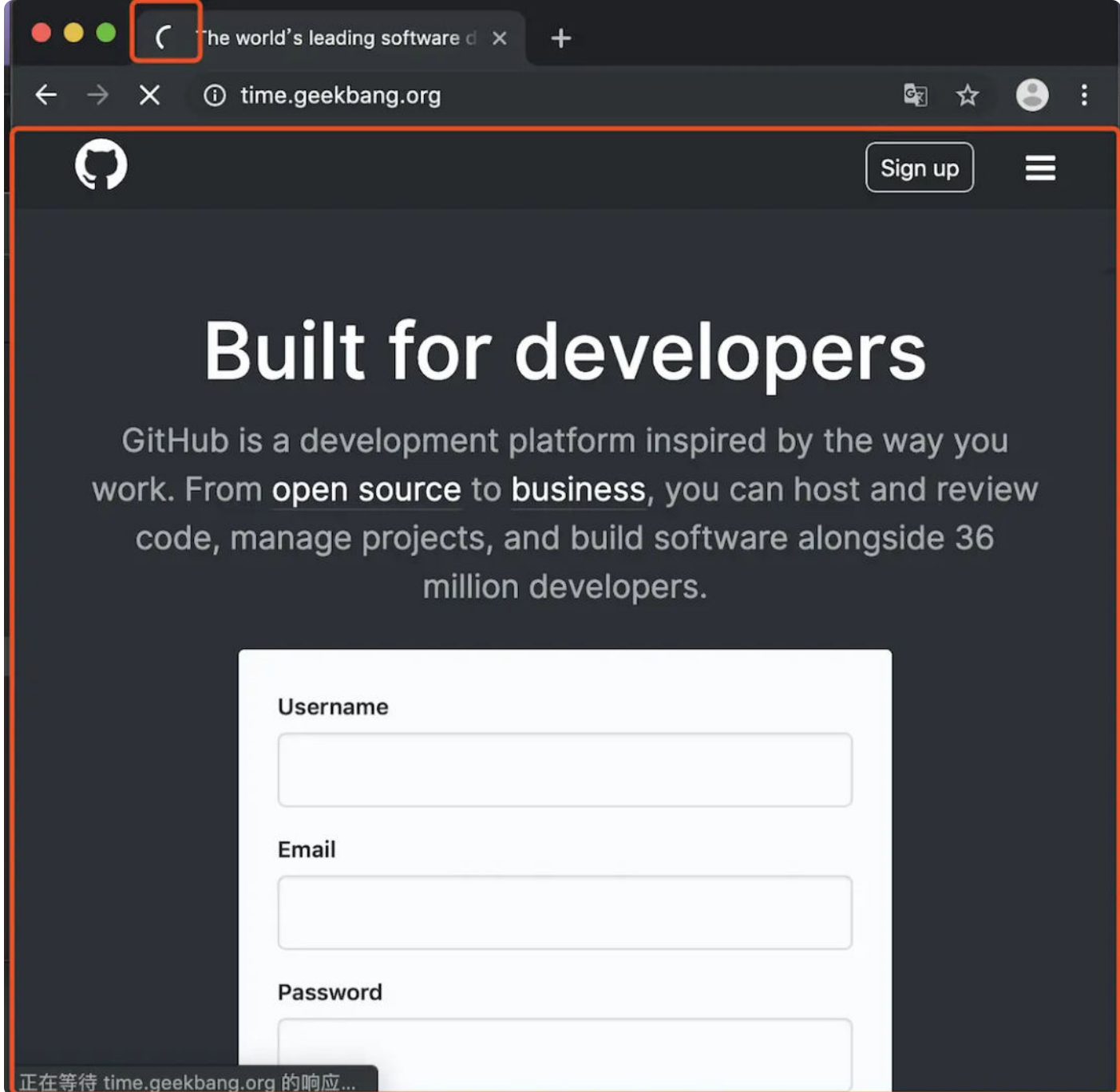
当用户在地址栏中输入一个查询关键字时，地址栏会判断输入的关键字是**搜索内容**，还是**请求的 URL**。

- 如果是搜索内容，地址栏会使用浏览器默认的搜索引擎，来合成新的带搜索关键字的 URL。
- 如果判断输入内容符合 URL 规则，比如输入的是 time.geekbang.org，那么地址栏会根据规则，把这段内容加上协议，合成为完整的 URL，如 [🔗https://time.geekbang.org](https://time.geekbang.org)。

当用户输入关键字并键入回车之后，这意味着当前页面即将要被替换成新的页面，不过在这个流程继续之前，浏览器还给了当前页面一次执行 beforeunload 事件的机会，beforeunload 事件允许页面在退出之前执行一些数据清理操作，还可以询问用户是否要离开当前页面，比如当前页面可能有未提交完成的表单等情况，因此用户可以通过 beforeunload 事件来取消导航，让浏览器不再执行任何后续工作。

当前页面没有监听 beforeunload 事件或者同意了继续后续流程，那么浏览器便进入下图的状态：





开始加载 URL 浏览器状态

从图中可以看出，当浏览器刚开始加载一个地址之后，标签页上的图标便进入了加载状态。但此时图中页面显示的依然是之前打开的页面内容，并没立即替换为极客时间的页面。因为需要等待提交文档阶段，页面内容才会被替换。

2. URL 请求过程

接下来，便进入了页面资源请求过程。这时，浏览器进程会通过进程间通信（IPC）把 URL 请求发送至网络进程，网络进程接收到 URL 请求后，会在这里发起真正的 URL 请求流程。那具体流程是怎样的呢？



首先，网络进程会查找本地缓存是否缓存了该资源。如果有缓存资源，那么直接返回资源给浏览器进程；如果在缓存中没有查找到资源，那么直接进入网络请求流程。这请求前的第一步是要进行 DNS 解析，以获取请求域名的服务器 IP 地址。如果请求协议是 HTTPS，那么还需要建立 TLS 连接。

接下来就是利用 IP 地址和服务器建立 TCP 连接。连接建立之后，浏览器端会构建请求行、请求头等信息，并把和该域名相关的 Cookie 等数据附加到请求头中，然后向服务器发送构建的请求信息。

服务器接收到请求信息后，会根据请求信息生成响应数据（包括响应行、响应头和响应体等信息），并发给网络进程。等网络进程接收了响应行和响应头之后，就开始解析响应头的内容了。（为了方便讲述，下面我将服务器返回的响应头和响应行统称为响应头。）

(1) 重定向

在接收到服务器返回的响应头后，网络进程开始解析响应头，如果发现返回的状态码是 301 或者 302，那么说明服务器需要浏览器重定向到其他 URL。这时网络进程会从响应头的 Location 字段里面读取重定向的地址，然后再发起新的 HTTP 或者 HTTPS 请求，一切又重头开始了。

比如，我们在终端里输入以下命令：

```
1 curl -I http://time.geekbang.org/
```

 复制代码

`curl -I + URL`的命令是接收服务器返回的响应头的信息。执行命令后，我们看到服务器返回的响应头信息如下：





响应行返回状态码 301

从图中可以看出，极客时间服务器会通过重定向的方式把所有 HTTP 请求转换为 HTTPS 请求。也就是说你使用 HTTP 向极客时间服务器请求时，服务器会返回一个包含有 301 或者 302 状态码响应头，并把响应头的 Location 字段中填上 HTTPS 的地址，这就是告诉了浏览器要重新导航到新的地址上。

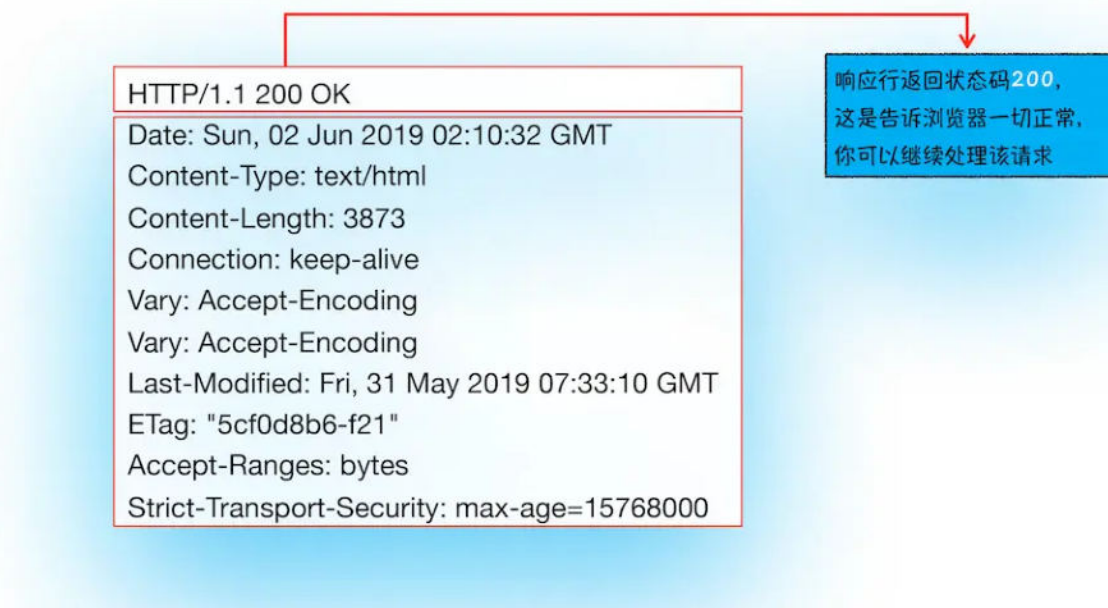
下面我们再使用 HTTPS 协议对极客时间发起请求，看看服务器的响应头信息是什么样子的。

```
1 curl -I https://time.geekbang.org/
```

复制代码

我们看到服务器返回如下信息：





响应行返回状态码 200

从图中可以看出，服务器返回的响应头的状态码是 200，这是告诉浏览器一切正常，可以继续往下处理该请求了。

好了，以上是重定向内容的介绍。现在你应该理解了，在导航过程中，如果服务器响应行的状态码包含了 301、302 一类的跳转信息，浏览器会跳转到新的地址继续导航；如果响应行是 200，那么表示浏览器可以继续处理该请求。

(2) 响应数据类型处理

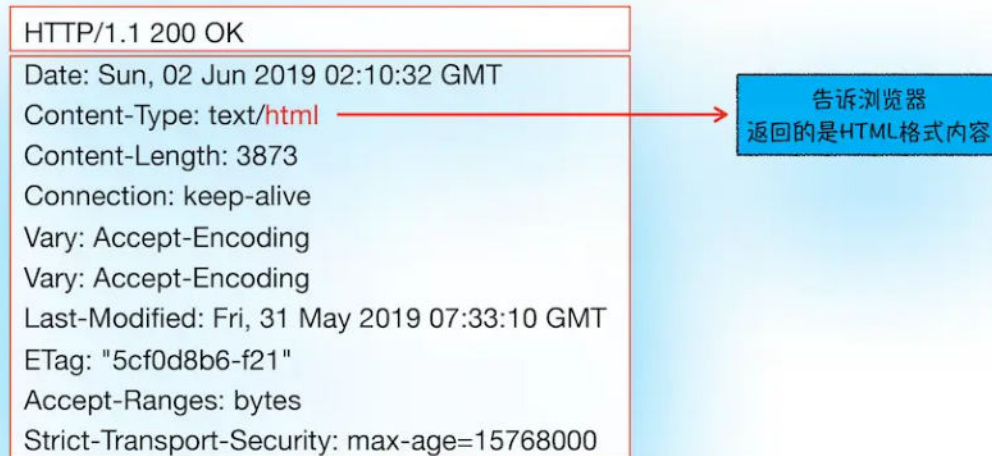
在处理了跳转信息之后，我们继续导航流程的分析。URL 请求的数据类型，有时候是一个下载类型，有时候是正常的 HTML 页面，那么浏览器是如何区分它们呢？

答案是 Content-Type。Content-Type 是 HTTP 头中一个非常重要的字段，它告诉浏览器服务器返回的响应体数据是什么类型，然后浏览器会根据 Content-Type 的值来决定如何显示响应体的内容。

这里我们还是以极客时间为例，看看极客时间官网返回的 Content-Type 值是什么。在终端输入以下命令：

```
1 curl -I https://time.geekbang.org/
```

返回信息如下图：



含有 HTML 格式的 Content-Type

从图中可以看到，响应头中的 Content-type 字段的值是 text/html，这就是告诉浏览器，服务器返回的数据是 **HTML 格式**。

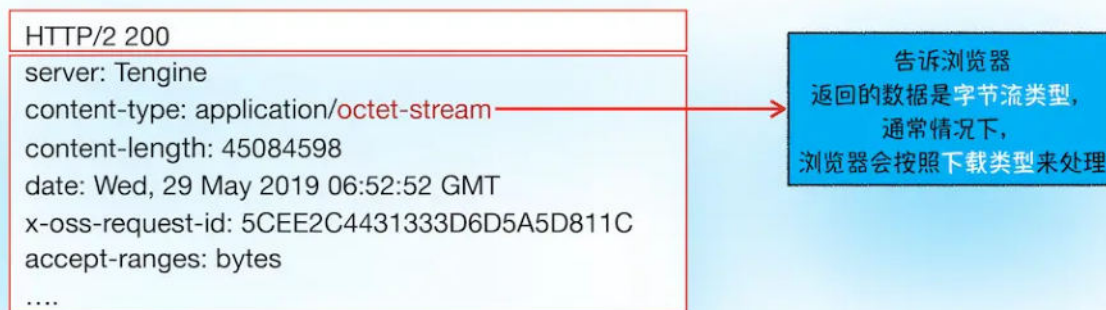
接下来我们再来利用 curl 来请求极客时间安装包的地址，如下所示：

```
1 curl -I https://res001.geekbang.org/apps/geektime/android/2.3.1/official/geektime
```

复制代码

请求后返回的响应头信息如下：





含有 stream 格式的 Content-Type

从返回的响应头信息来看，其 Content-Type 的值是 application/octet-stream，显示数据是**字节流类型**的，通常情况下，浏览器会按照**下载类型**来处理该请求。

需要注意的是，如果服务器配置 Content-Type 不正确，比如将 text/html 类型配置成 application/octet-stream 类型，那么浏览器可能会曲解文件内容，比如会将一个本来是用来展示的页面，变成了一个下载文件。

所以，不同 Content-Type 的后续处理流程也截然不同。如果 Content-Type 字段的值被浏览器判断为**下载类型**，那么该请求会被提交给浏览器的下载管理器，同时该 URL 请求的**导航流程就此结束**。但如果是 HTML，那么浏览器则会继续进行导航流程。由于 Chrome 的页面渲染是运行在渲染进程中的，所以接下来就需要准备渲染进程了。

3. 准备渲染进程

默认情况下，Chrome 会为每个页面分配一个渲染进程，也就是说，每打开一个新页面就会配套创建一个新的渲染进程。但是，也有一些例外，在某些情况下，浏览器会让多个页面直接运行在同一个渲染进程中。

比如我从极客时间的首页里面打开了另外一个页面——算法训练营，我们看下图的 Chrome 的任务管理器截图：





多个页面运行在一个渲染进程中

从图中可以看出，打开的这三个页面都是运行在同一个渲染进程中，进程 ID 是 23601。

那什么情况下多个页面会同时运行在一个渲染进程中呢？

要解决这个问题，我们就需要先了解下什么是同一站点（same-site）。具体地讲，我们将“同一站点”定义为根域名（例如，geekbang.org）加上协议（例如，https:// 或者 http://），还包含了该根域名下的所有子域名和不同的端口，比如下面这三个：

```
1 https://time.geekbang.org
2 https://www.geekbang.org
3 https://www.geekbang.org:8080
```

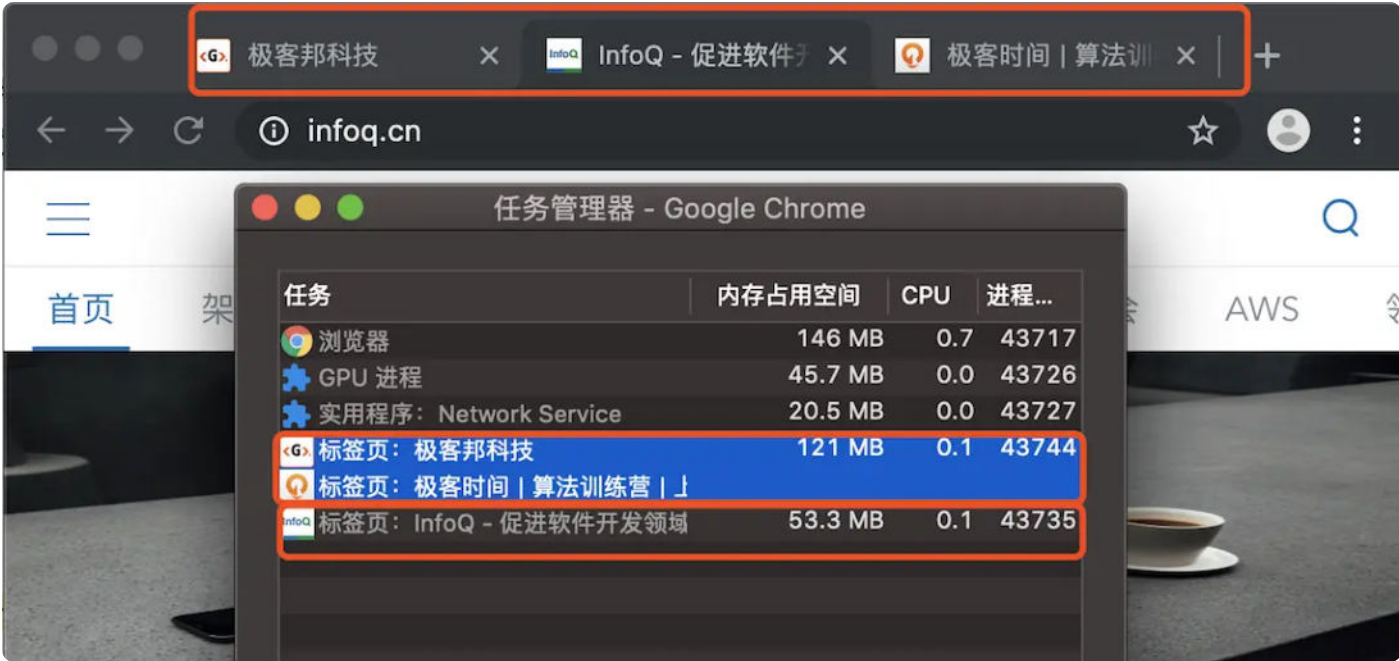
复制代码



它们都是属于同一站点，因为它们的协议都是 HTTPS，而且根域名也都是 geekbang.org。

Chrome 的默认策略是，每个标签对应一个渲染进程。但如果从一个页面打开了另一个新页面，而新页面和当前页面属于同一站点的话，那么新页面会复用父页面的渲染进程。官方把这个默认策略叫 process-per-site-instance。

那若新页面和当前页面不属于同一站点，情况又会发生什么样的变化呢？比如我通过极客邦页面里的链接打开 InfoQ 的官网（<https://www.infoq.cn/>），因为 infoq.cn 和 geekbang.org 不属于同一站点，所以 infoq.cn 会使用一个新的渲染进程，你可以参考下图：



非同一起点使用不同的渲染进程

从图中任务管理器可以看出：由于极客邦和极客时间的标签页拥有**相同的协议和根域名**，所以它们属于**同一站点**，并运行在同一个渲染进程中；而 infoq.cn 的根域名不同于 geekbang.org，也就是说 InfoQ 和极客邦不属于同一站点，因此它们会运行在两个不同的渲染进程之中。

总结来说，打开一个新页面采用的**渲染进程策略**就是：

- 通常情况下，打开新的页面都会使用单独的渲染进程；
- 如果从 A 页面打开 B 页面，且 A 和 B 都属于**同一站点**的话，那么 B 页面复用 A 页面的渲染进程；如果是其他情况，浏览器进程则会为 B 创建一个新的渲染进程。



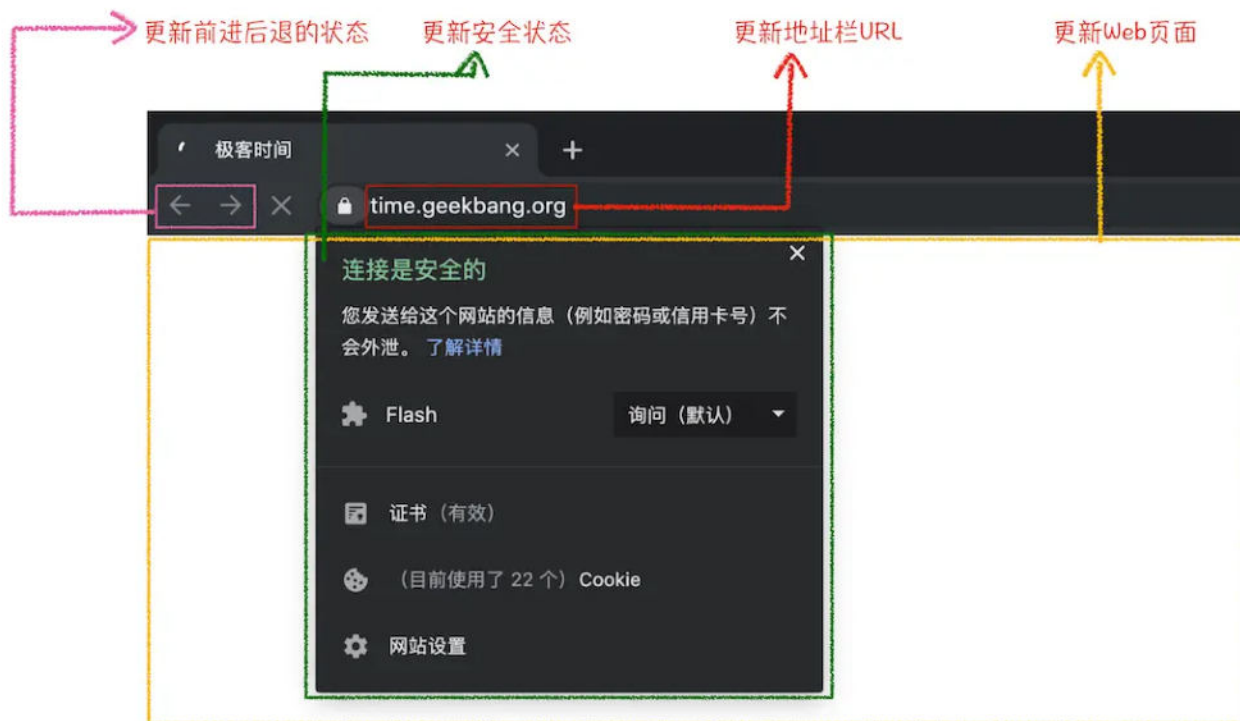
渲染进程准备好之后，还不能立即进入文档解析状态，因为此时的文档数据还在网络进程中，并没有提交给渲染进程，所以下一步就进入了提交文档阶段。

4. 提交文档

所谓提交文档，就是指浏览器进程将网络进程接收到的 HTML 数据提交给渲染进程，具体流程是这样的：

- 首先当浏览器进程接收到网络进程的响应头数据之后，便向渲染进程发起“提交文档”的消息；
- 渲染进程接收到“提交文档”的消息后，会和网络进程建立传输数据的“管道”；
- 等文档数据传输完成之后，渲染进程会返回“确认提交”的消息给浏览器进程；
- 浏览器进程在收到“确认提交”的消息后，会更新浏览器界面状态，包括了安全状态、地址栏的 URL、前进后退的历史状态，并更新 Web 页面。

其中，当渲染进程**确认提交**之后，更新内容如下图所示：



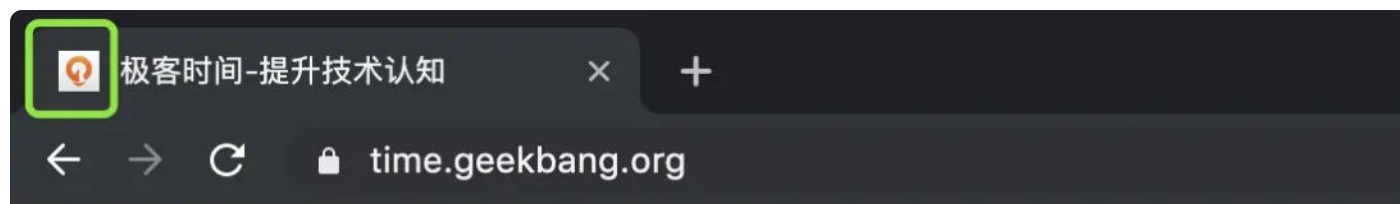
导航完成状态

这也就解释了为什么在浏览器的地址栏里面输入了一个地址后，之前的页面没有立马消失，而是要加载一会儿才会更新页面。

到这里，一个完整的导航流程就“走”完了，这之后就要进入渲染阶段了。

5. 渲染阶段

一旦文档被提交，渲染进程便开始页面解析和子资源加载了，关于这个阶段的完整过程，我会在下一篇文章中来专门介绍。这里你只需要先了解一旦页面生成完成，渲染进程会发送一个消息给浏览器进程，浏览器接收到消息后，会停止标签图标上的加载动画。如下所示：



渲染结束

至此，一个完整的页面就生成了。那文章开头的“从输入 URL 到页面展示，这中间发生了什么？”这个过程及其“串联”的问题也就解决了。

总结

好了，今天就到这里，下面我来简单总结下这篇文章的要点：

- 服务器可以根据响应头来控制浏览器的行为，如跳转、网络数据类型判断。
- Chrome 默认采用每个标签对应一个渲染进程，但是如果两个页面属于同一站点，那这两个标签会使用同一个渲染进程。
- 浏览器的导航过程涵盖了从用户发起请求到提交文档给渲染进程的中间所有阶段。



导航流程很重要，它是网络加载流程和渲染流程之间的一座桥梁，如果你理解了导航流程，那么你就能完整串起来整个页面显示流程，这对于你理解浏览器的工作原理起到了点睛的作用。


思考时间

最后，还是留给你个小作业：在上一篇文章中我们介绍了 HTTP 请求过程，在本文我们又介绍了导航流程，那么如果再有面试官问你“从输入 URL 到页面展示，这中间发生了什么？”这个问题，你知道怎么回答了吗？可以用你自己的语言组织下，就当为你的面试做准备。

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。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 66  提建议

© 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。

上一篇 03 | HTTP请求流程：为什么很多站点第二次打开速度会很快？

下一篇 05 | 渲染流程（上）：HTML、CSS和JavaScript，是如何变成页面的？



JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



精选留言 (127)

写留言



ytd

2019-08-13

结合老师的讲义，自己总结了下，不考虑用户输入搜索关键字的情况：

- 1，用户输入url并回车
- 2，浏览器进程检查url，组装协议，构成完整的url
- 3，浏览器进程通过进程间通信（IPC）把url请求发送给网络进程
- 4，网络进程接收到url请求后检查本地缓存是否缓存了该请求资源，如果有则将该资源返回给浏览器进程
- 5，如果没有，网络进程向web服务器发起http请求（网络请求），请求流程如下：
 - 5.1 进行DNS解析，获取服务器ip地址，端口（端口是通过dns解析获取的吗？这里有个疑问）
 - 5.2 利用ip地址和服务器建立tcp连接
 - 5.3 构建请求头信息
 - 5.4 发送请求头信息
 - 5.5 服务器响应后，网络进程接收响应头和响应信息，并解析响应内容
- 6，网络进程解析响应流程；
 - 6.1 检查状态码，如果是301/302，则需要重定向，从Location自动中读取地址，重新进行第4步
(301/302跳转也会读取本地缓存吗？这里有个疑问)，如果是200，则继续处理请求。
 - 6.2 200响应处理：



检查响应类型Content-Type，如果是字节流类型，则将该请求提交给下载管理器，该导航流程结束，不再进行

后续的渲染，如果是html则通知浏览器进程准备渲染进程准备进行渲染。

7. 准备渲染进程

7.1 浏览器进程检查当前url是否和之前打开的渲染进程根域名是否相同，如果相同，则复用原来的进程，如果不同，则开启新的渲染进程

8. 传输数据、更新状态

8.1 渲染进程准备好后，浏览器向渲染进程发起“提交文档”的消息，渲染进程接收到消息和网络进程建立传输数据的“管道”

8.2 渲染进程接收完数据后，向浏览器发送“确认提交”

8.3 浏览器进程接收到确认消息后更新浏览器界面状态：安全、地址栏url、前进后退的历史状态、更新web页面。

作者回复: 赞的，回头补上05-06节所讲的渲染流程，整个过程就完整了

共 23 条评论 >

👍 291



羽蝶曲

2019-08-13

1. 用户输入URL，浏览器会根据用户输入的信息判断是搜索还是网址，如果是搜索内容，就将搜索内容+默认搜索引擎合成新的URL；如果用户输入的内容符合URL规则，浏览器就会根据URL协议，在这段内容上加上协议合成合法的URL
2. 用户输入完内容，按下回车键，浏览器导航栏显示loading状态，但是页面还是呈现前一个页面，这是因为新页面的响应数据还没有获得
3. 浏览器进程浏览器构建请求行信息，会通过进程间通信（IPC）将URL请求发送给网络进程 GET /index.html HTTP1.1
4. 网络进程获取到URL，先去本地缓存中查找是否有缓存文件，如果有，拦截请求，直接200返回；否则，进入网络请求过程
5. 网络进程请求DNS返回域名对应的IP和端口号，如果之前DNS数据缓存服务缓存过当前域名信息，就会直接返回缓存信息；否则，发起请求获取根据域名解析出来的IP和端口号，如果没有端口号，http默认80，https默认443。如果是https请求，还需要建立TLS连接。
6. 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。如果当前请求数量少于6个，会直接建立TCP连接。
7. TCP三次握手建立连接，http请求加上TCP头部——包括源端口号、目的程序端口号和用于校验数据完整性的序号，向下传输
8. 网络层在数据包上加上IP头部——包括源IP地址和目的IP地址，继续向下传输到底层
9. 底层通过物理网络传输给目的服务器主机
10. 目的服务器主机网络层接收到数据包，解析出IP头部，识别出数据部分，将解开的数据包向上传输到传输层
11. 目的服务器主机传输层获取到数据包，解析出TCP头部，识别端口，将解开的数据包向上



传输到应用层

12. 应用层HTTP解析请求头和请求体，如果需要重定向，HTTP直接返回HTTP响应数据的状态code301或者302，同时在请求头的Location字段中附上重定向地址，浏览器会根据code和Location进行重定向操作；如果不是重定向，首先服务器会根据 请求头中的If-None-Match的值来判断请求的资源是否被更新，如果没有更新，就返回304状态码，相当于告诉浏览器之前的缓存还可以使用，就不返回新数据了；否则，返回新数据，200的状态码，并且如果想要浏览器缓存数据的话，就在相应头中加入字段：

Cache-Control:Max-age=2000

响应数据又顺着应用层——传输层——网络层——网络层——传输层——应用层的顺序返回到网络进程

13. 数据传输完成，TCP四次挥手断开连接。如果，浏览器或者服务器在HTTP头部加上如下信息，TCP就一直保持连接。保持TCP连接可以省下下次需要建立连接的时间，提示资源加载速度

Connection:Keep-Alive

14. 网络进程将获取到的数据包进行解析，根据响应头中的Content-type来判断响应数据的类型，如果是字节流类型，就将该请求交给下载管理器，该导航流程结束，不再进行；如果是text/html类型，就通知浏览器进程获取到文档准备渲染

15. 浏览器进程获取到通知，根据当前页面B是否是从页面A打开的并且和页面A是否是同一个站点（根域名和协议一样就被认为是同一个站点），如果满足上述条件，就复用之前网页的进程，否则，新创建一个单独的渲染进程

16. 浏览器会发出“提交文档”的消息给渲染进程，渲染进程收到消息后，会和网络进程建立传输数据的“管道”，文档数据传输完成后，渲染进程会返回“确认提交”的消息给浏览器进程

17. 浏览器收到“确认提交”的消息后，会更新浏览器的页面状态，包括了安全状态、地址栏的URL、前进后退的历史状态，并更新web页面，此时的web页面是空白页

18. 渲染进程对文档进行页面解析和子资源加载，HTML 通过HTML 解析器转成DOM Tree（二叉树类似结构的东西），CSS按照CSS 规则和CSS解释器转成CSSOM TREE，两个tree结合，形成render tree（不包含HTML的具体元素和元素要画的具体位置），通过Layout可以计算出每个元素具体的宽高颜色位置，结合起来，开始绘制，最后显示在屏幕中新页面显示出来

作者回复: 总结的很详细，用心了！

共 15 条评论>

👍 175



芳华年月

2019-08-15

请问老师，<https://linkmarket.aliyun.com>内新开的页面都是新开一个渲染进程，能帮忙解释下吗



作者回复: 我看了下代码，因为连接里面使用了 rel="noopener noreferrer"这个属性。

这个涉及到安全了，要完整解释起来就话长了，我长话短说，先看阿里这个网站的连接是下面这种形式：

```
<a target="_blank" rel="noopener noreferrer" class="hover" href="https://linkmarket.aliyun.com/hardware_store?spm=a2c3t.11219538.iot-navBar.62.4b5a51e7u2sXtw" data-spm-anchor-id="a2c3t.11219538.iot-navBar.62">硬件商城</a>
```

使用noopener noreferrer就是告诉浏览器，新打开的子窗口不需要访问父窗口的任何内容，这是为了防止一些钓鱼网站窃取父窗口的信息。

浏览器在打开新页面时，解析到含有noopener noreferrer时，就知道他们不需要共享页面内容，所以这时候浏览器就会让新链接在一个新页面中打开了。

共 2 条评论 >

👍 109



Geek_East

2019-11-28

参考google官方文档，貌似提交文档的原文是commit navigation，提交是从浏览器进程发起，至渲染进程，渲染进程确认导航，然后开始正式渲染；Now that the data and the renderer process is ready, an IPC is sent from the browser process to the renderer process to commit the navigation. It also passes on the data stream so the renderer process can keep receiving HTML data. Once the browser process hears confirmation that the commit has happened in the renderer process, the navigation is complete and the document loading phase begins.

作者回复：

多谢，看了你的提示，我重新看了一遍源码的执行过程，原文的确有两个地方描述不妥当，我现在评论区做下修订，稍后更正原文。

1:浏览器进程发出URL请求给网络进程

2:网络进程接收到URL请求之后，便发起网络请求，然后服务器返回HTTP数据到网络进程，网络进程解析HTTP出来响应头数据，并将其转发给浏览器进程

3:浏览器进程接收到网络进程的响应头数据之后，发送CommitNavigation消息到渲染进程，发送CommitNavigation时会携带响应头、等基本信息。

4:渲染进程接收到CommitNavigation消息之后，便开始准备接收HTML数据，接收数据的方式是直接和网络进程建立数据管道

5:最后渲染进程会像浏览器进程“确认提交”，这是告诉浏览器进程，说我已经准备好接受和解析页面



数据了

6:最后浏览器进程更新页面状态

共 14 条评论 >

👍 72



👉(●°V°●)👈 🏆

2019-08-14

那么浏览器的http的keepalive的connection是什么粒度复用的呢？也是域名加协议头级别吗？

作者回复：

由于正文篇幅有限，无法对keep-alive做详细解释，刚好借着这个问题，我把keep-alive讲清楚。

首先keep-alive是为了解决连接效率不高的问题，http1.0时代，http请求都是短连接的形式，也即是每次请求一个资源都需要和服务器建立连接+传输数据+断开连接，通常，建立连接和断开连接的时间就有可能超过传输数据的时间了，这种短连接的效率是异常的低效。

针对短连接低效的问题，后面就出现了长连接，也就是这里要讲的keep-alive。

你可以把长连接看成是一个管道，一个http请求结束之后，不会关闭连接，下个请求可以复用该连接，这样就省去建立连接和断开连接的时间了，但是他们请求是按照顺序，也就是符合IP+端口规则的资源都可以复用该连接，这就回答了上面提的这个问题。

但是，使用keep-alive同样存在问题，比如一个页面可能有100张图片素材，假设这些图片素材都保存在同一个域名下面，如果只复用一个http管道的话，那么传输100张图片的素材也是非常耗时间的，这就出现了同一时刻并发连接服务器的需求，也就是文中提到同一时刻，对同一域名下面，只能发起6个请求，这样就可以大大提升请求效率了。

为什么是6个请求而不是更多了，这是为了服务器性能考虑，如果同一时刻无限制连接，那么可能会导致服务器忙不过来。

共 6 条评论 >

👍 56



JawQ_

2019-08-15

同一站点共用一个渲染进程，那假设有2个标签页是同一站点，我在A标签页面写个死循环，导致页面卡死，B页面是否也是卡死了呢？



作者回复：你能想到这个问题，说明你已经快思考到最核心的---事件循环机制了，非常好。

多个页面公用一个渲染进程，也就意味着多个页面公用同一个主线程，所有页面的任务都是在同一个主线程上执行，这些任务包括渲染流程，JavaScript执行，用户交互的事件的响应等等，@@@但是@@@ 如果一个标签页里面执行一个死循环，那么意味着该JavaScript代码会一直霸占主线程，这样就导致了其它的页面无法使用该主线程，从而让所有页面都失去响应！

关于循环系统，我会在后续章节做详细分析！

共 5 条评论 >

👍 43



前端队长Daotin

2019-08-15

- 1、用户输入关键词，地址栏判断是搜索内容还是url地址。
如果是搜索内容，会使用浏览器默认搜索引擎加上搜索内容合成url；
如果是域名会加上协议（如https）合成完整的url。
- 2、然后按下回车。浏览器进程通过IPC（进程间通信）把url传给网络进程（网络进程接收到url才发起真正的网络请求）。
- 3、网络进程接收到url后，先查找有没有缓存。
有缓存，直接返回缓存的资源。
没有缓存。（进入真正的网络请求）。首先获取域名的IP，系统会首先自动从hosts文件中寻找域名对应的 IP 地址，一旦找到，和服务器建立TCP连接；如果没有找到，则系统会将网址提交 DNS 域名解析服务器进行 IP 地址的解析。
- 4、利用IP地址和服务器建立TCP连接（3次握手）。
- 5、建立连接后，浏览器构建数据包（包含请求行，请求头，请求正文，并把该域名相关Cookie等数据附加到请求头），然后向服务器发送请求消息。
- 6、服务器接收到消息后根据请求信息构建响应数据（包括响应行，响应头，响应正文），然后发送回网络进程。
- 7、网络进程接收到响应数据后进行解析。
如果发现响应行的返回的状态码为301，302，说明服务器要我们去找别人要数据，找谁呢？
找响应头中的Location字段要，Location的内容是需要重定向的地址url。获取到这个url一切重新来过。
如果返回的状态码为200，说明服务器返回了数据。



8、好了，获取到数据以什么方式打开呢？打开的方式不对的话也不行。打开的方式就是 Content-Type。这个属性告诉浏览器服务器返回的数据是什么类型的。如果返回的是网页类型则为 text/html，如果是下载文件类型则为 application/octet-stream 等等。打开的方式不对，则得到的结果也不对。

如果是下载类型，则该请求会被提交给浏览器的下载管理器，同时该请求的流程到此结束。如果是网页类型，那么浏览器就要准备渲染页面了。

9、渲染页面开始。浏览器进程发出“提交文档”（文档是响应体数据）消息给渲染进程，渲染进程接收到消息后会和网络进程建立传输数据的通道，网络进程将“文档”传输给渲染进程。

10、一旦开始传输，渲染进程便开始渲染界面（详细渲染过程待续。。。)

11、传输完毕，渲染进程会发出“确认提交”消息给浏览器进程。

12、浏览器在接收到“确认提交”消息后，更新浏览器界面状态（包括地址栏信息，仟前进后退历史，web页面和网站安全状态）。

13、页面此时可能还没有渲染完毕，而一旦渲染完毕，渲染进程会发送一个消息给浏览器进程，浏览器接收到这个消息后会停止标签图标的加载动画。

自此，一个完整的页面形成了。

作者回复: 超级赞

共 12 条评论 >

👍 31



Oliver

2019-08-13

这也就解释了为什么在浏览器的地址栏里面输入了一个地址后，之前的页面没有立马消失，而是要加载一会儿才会更新页面。

第一步应该是触发当前页的卸载事件和收集需要释放内存，这也占用了一些时间，但大头应该是请求新的url时的返回

作者回复: 补充的好 👍

共 2 条评论 >

👍 24



tokey

2019-09-03

老师！如果一个页面发出请求后就关闭了，那么这个页面的进程就关闭了吧？那么 tcp 的连接还会有（请求能不能到达服务端），如果连接成功服务端处理过后 tcp 断开需要四次挥手，此时服务器收不到客户端的断开确认消息，服务器会处于什么状态（一直等待么）？

作者回复: 页面进程关闭后，浏览器进程会接收到关闭的消息，然后浏览器进程会通知网络进程主动断开该页面的所有tcp连接。

所以你不用担心页面关闭会导致网络问题！

共 2 条评论 >

👍 18



Mr. Cheng

2019-08-14

作为一个从其他行业跳到web前端的，网络方面的知识欠缺太多，虽然在其他地方也看了很多，但这个专栏讲的清晰易懂，大赞

作者回复: 谢谢，这也让我很开心



👍 15



Robert小七

2019-08-15

我打开百度，然后右键点击百度文库以新页面打开的方式，发现两个标签是两个不同的渲染进程！

作者回复: 要左键直接点击才开的才算

共 5 条评论 >

👍 13



无名

2019-08-24

如：打开<https://www.baidu.com/index.html>，index.html页面中肯定有很多css和js链接文件，那么当浏览器进程通过IPC把URL请求发送至网络进程，网络进程将index.html页面下载完成后，告诉浏览器进程，浏览器进程创建渲染进程，发出“提交文档”的消息，然后渲染进程与网络进程建立管道，渲染进程接收完数据后，向浏览器进程发出“确认提交”的消息，浏览器进程做停止加载条，更新页面等操作。

老师，这个过程有2个疑问：

1、只要下载完成index.html文件(不包括里面的css和js等文件)，浏览器进程就会发出“提交文档”的消息是吗？



2、渲染进程接收完数据，怎么算接收完数据？是把index.html中的响应数据接收了就算完吗？包不包括里面的css、js等文件下载完才算是接收完？

作者回复: 第一个就是接受到第一批index.html的数据就会发送提交文档的消息

第二个要等所有资源加载完毕，js css image等

共 8 条评论 >

👍 9



EanCuznaivy

2019-08-13

想起来一个github repo: <https://github.com/alex/what-happens-when>，讲了从浏览器地址框输入google.com按下回车之后会发生什么.....

作者回复: 赞

共 2 条评论 >

👍 9



tokey

2019-08-22

老师！我 curl -I http://www.baidu.com 返回的信息是200，但是浏览器直接输入 http 协议的时候百度会跳转到 https，这个是因为什么啊？那百度是怎么做的重定向呢？

作者回复: 那说明百度依然支持http协议，浏览器里面可能你之前有过https的访问记录，所以浏览器在请求之前将你的协议切换成https了。



👍 6



月上秦少

2019-08-15

哇，有一次面试问题遇到了，答得一塌糊涂，没想到大佬这里竟然有这个专题。

作者回复: 哈哈，01篇到06篇都可以看看



👍 6



常金

2020-02-22

这篇中的URL请求部分和第三篇中的内容有出入
第三篇表示先构建请求，再查询缓存



这篇中说先查询缓存，再构建请求。。。搞晕了。。。

共 4 条评论 >

👍 5



Elmer

2019-08-13

提交文档后向浏览器确认提交，这个时候更新的web页面只是一个空白页面吗？等页面渲染阶段完成后才会展示解析内容吗？

作者回复: 是的，提交文档后便开始解析DOM，解析CSS，生成布局树，然后绘制等操作，从解析HTML到绘制页面的第一个像素，是需要时间的！

优化好的页面这个时间会很短，优化差的页面这个空白时间会比较久！



👍 5



乃乎

2020-01-09

建议看原文？https://developers.google.com/web/updates/2018/09/inside-browser-part2#a_simple_navigation



👍 4



Geek_Leon

2019-08-20

李老师您好，关于下面这句话：“提交文档”的消息由浏览器进程发出，渲染进程接收到“提交文档”消息后，和网络进程建立传输数据的“管道”。

这句话是否可以理解为：此时响应报文在网络进程那里，需要浏览器进程做一个统一调度，跟渲染进程说，你可以去接收响应报文了。然后渲染进程就通过IPC与网络进程通信，让网络进程把响应报文全部发过来。渲染进程拿到所有响应报文，就会回复浏览器进程全部响应报文都拿到了，你浏览器进程可以更新界面，我渲染进程可以进行渲染。

作者回复: 可以这么理解

共 2 条评论 >

👍 4



Snow同學

2019-08-15

提交文档，这里的“文档”是指 URL 请求的响应体。

请问老师：网络进程向渲染进程提交文档，是网络进程下载完整了响应体，一次完整的提交给渲染进程呢？还是网络进程边下载边提交给渲染进程呢？



作者回复: 对 文档就是指响应体的数据。

边下载边解析的，接收到第一批数据，便开始做DOM解析了！

共 3 条评论 >



4

