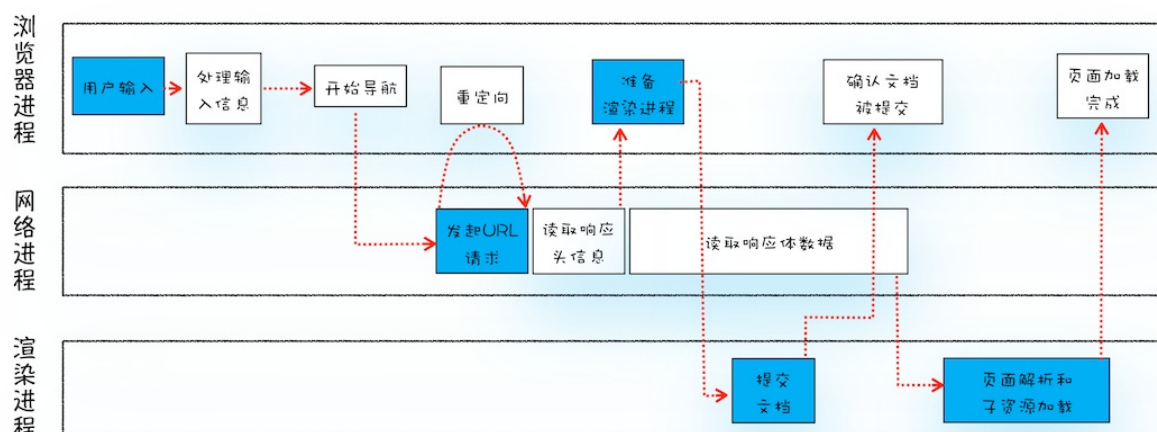


04-导航流程：从输入URL到页面展示，这中间发生了什么？

“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发起攻击，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，用户从浏览器进程里**输入请求信息**；
- 然后，网络进程**发起URL请求**；
- 服务器响应URL请求之后，浏览器进程就又要开始**准备渲染进程**了；
- 渲染进程准备好之后，需要先向渲染进程提交页面数据，我们称之为**提交文档**阶段；
- 渲染进程接收完文档信息之后，便开始**解析页面和加载子资源**，完成页面的渲染。

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**。下面我们来详细分析下这些步骤，同时也就解答了开头所说的那道经典的面试题。

从输入URL到页面展示

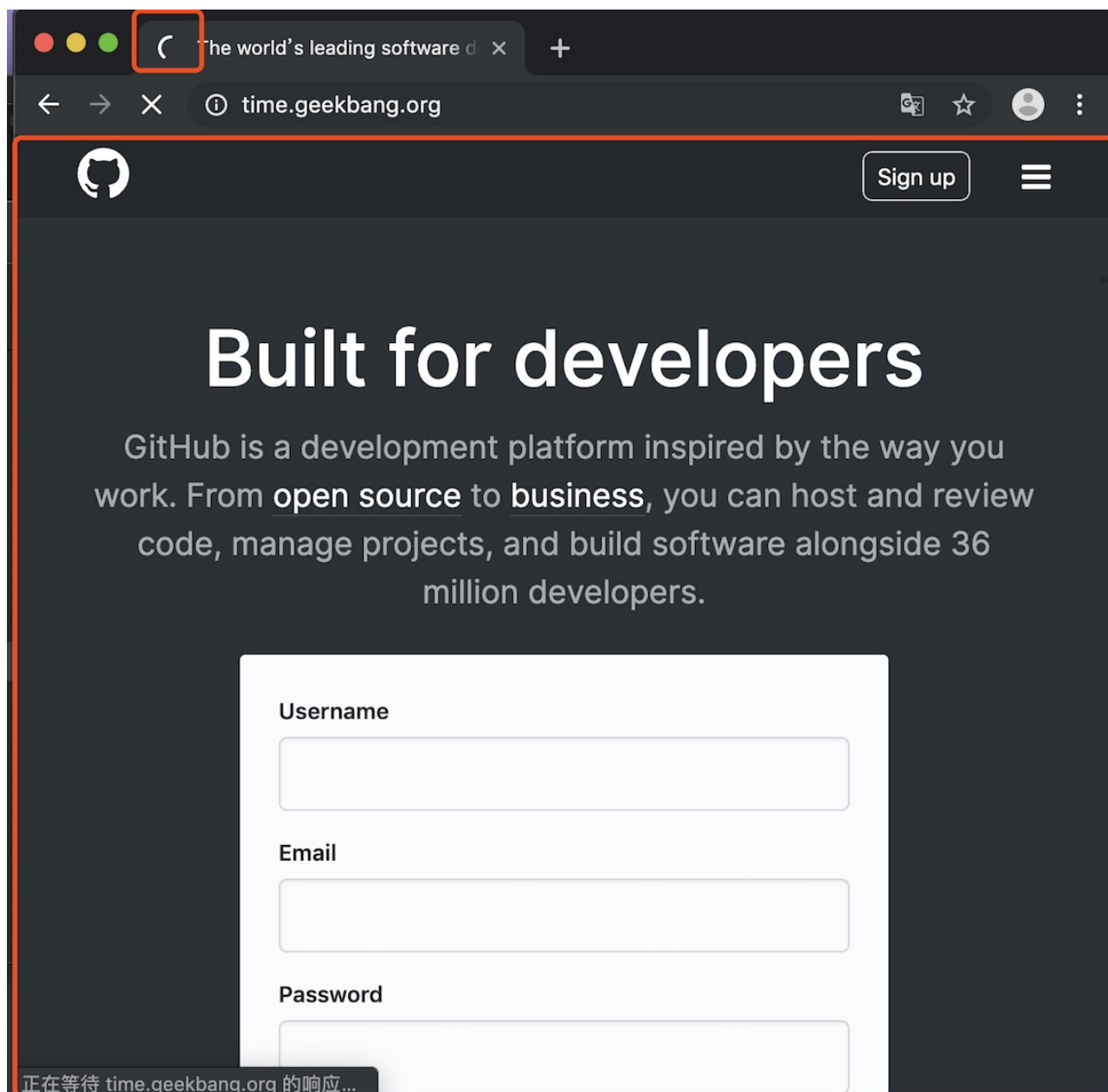
知道了浏览器的几个主要进程的职责之后，那么接下来，我们就从浏览器的地址栏开始讲起。

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>。

当用户输入关键字并键入回车之后，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：

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

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：

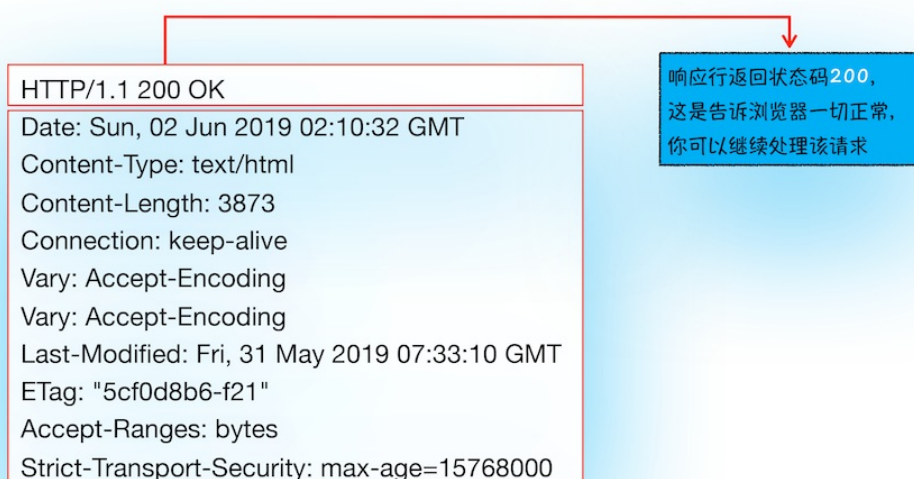


从图中可以看出，极客时间服务器会通过重定向的方式把所有HTTP请求转换为HTTPS请求。也就是说你使用HTTP向极客时间服务器请求时，服务器会返回一个包含有301或者302状态码响应头，并把响应头的Location字段中填上HTTPS的地址，这就是告诉了浏览器要重新导航到新的地址上。

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。

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

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

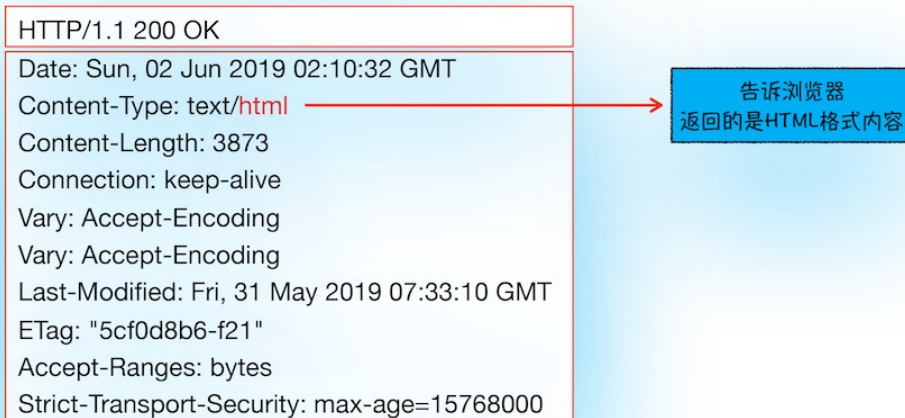
在处理了跳转信息之后，我们继续导航流程的分析。URL请求的数据类型，有时候是一个下载类型，有时候是正常的HTML页面，那么浏览器是如何区分它们呢？

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：

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

返回信息如下图：



The diagram shows a list of HTTP response headers enclosed in a red-bordered box. A red arrow points from the 'Content-Type: text/html' line to a blue box on the right. The blue box contains the text '告诉浏览器 返回的是HTML格式内容'.

```
HTTP/1.1 200 OK
Date: Sun, 02 Jun 2019 02:10:32 GMT
Content-Type: text/html
Content-Length: 3873
Connection: keep-alive
Vary: Accept-Encoding
Vary: Accept-Encoding
Last-Modified: Fri, 31 May 2019 07:33:10 GMT
ETag: "5cf0d8b6-f21"
Accept-Ranges: bytes
Strict-Transport-Security: max-age=15768000
```

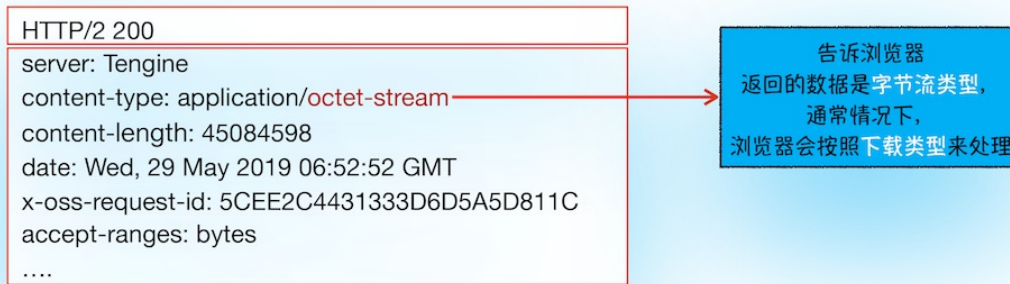
含有HTML格式的Content-Type

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：

```
curl -I https://res001.geekbang.org/apps/geektime/android/2.3.1/official/geektime_2.3.1_20190527-2136_offic
```

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。

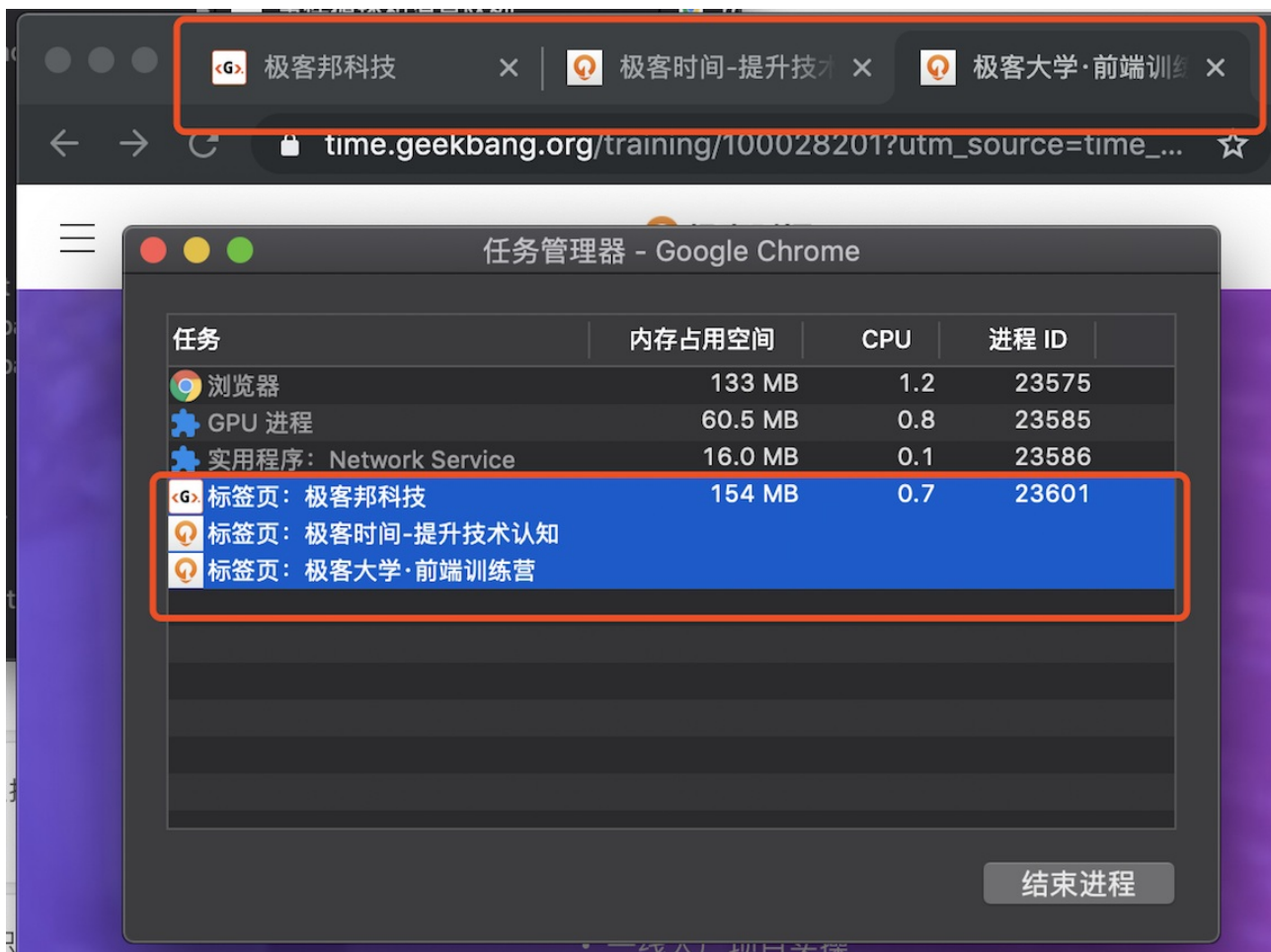
需要注意的是，如果服务器配置Content-Type不正确，比如将text/html类型配置成application/octet-stream类型，那么浏览器可能会曲解文件内容，比如会将一个本来是用来展示的页面，变成了一个下载文件。

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？

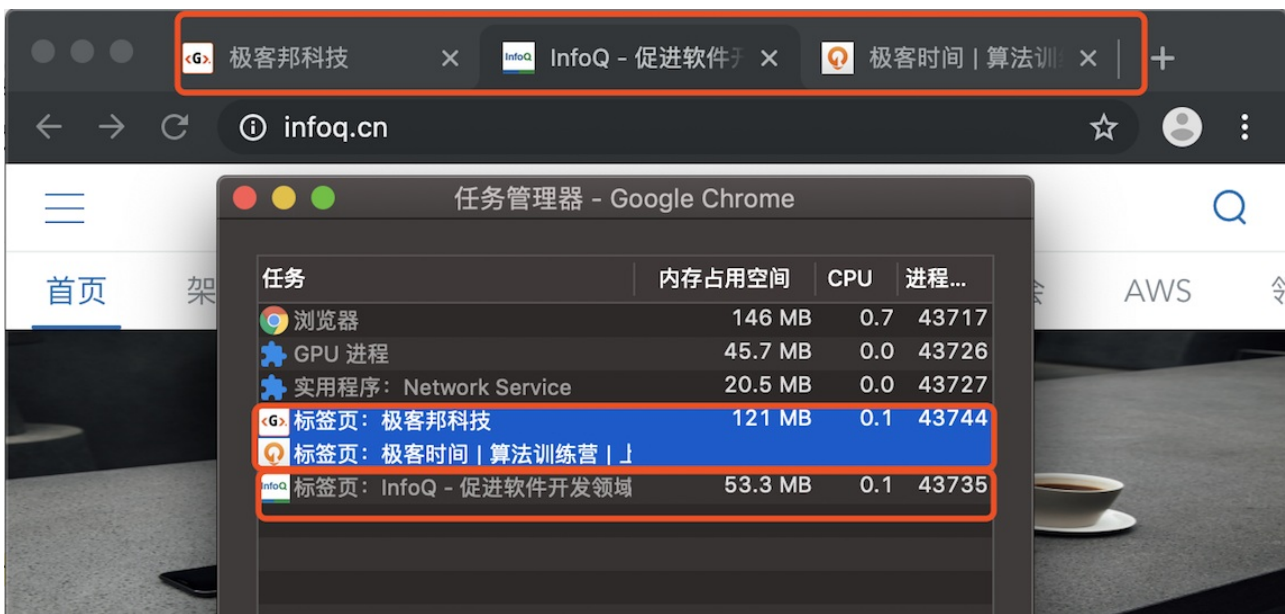
要解决这个问题，我们就需要先了解下什么是同一站点（same-site）。具体地讲，我们将“**同一站点**”定义为**根域名**（例如，geekbang.org）加上**协议**（例如，https:// 或者http://），还包含了该根域名下的所有子域名和不同的端口，比如下面这三个：

```
https://time.geekbang.org
https://www.geekbang.org
https://www.geekbang.org:8080
```

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：



非同一站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。

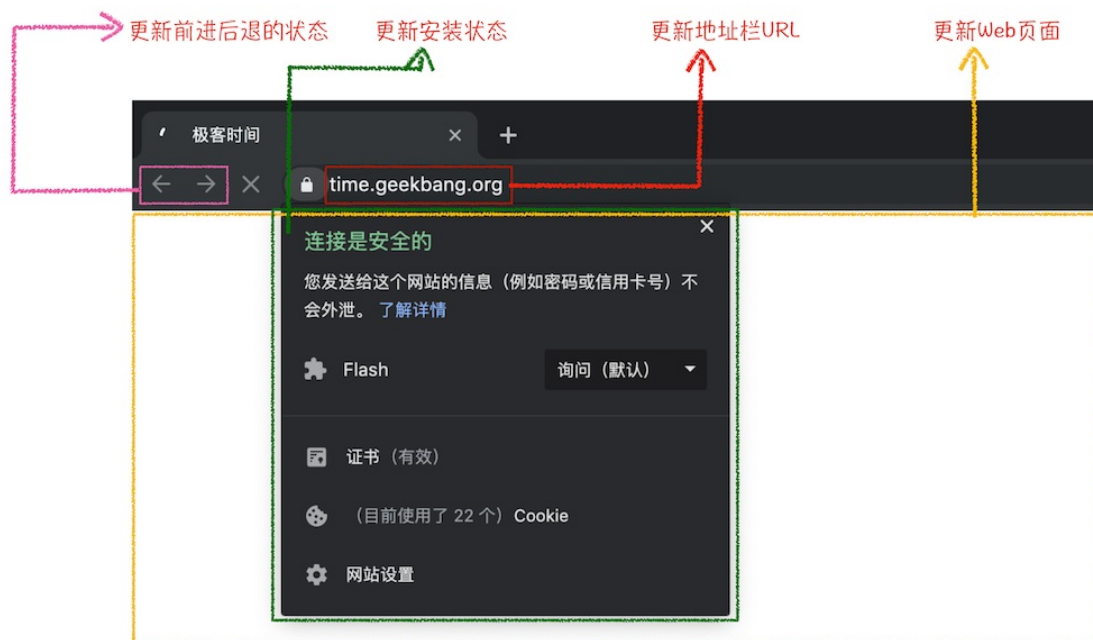
渲染进程准备好之后，还不能立即进入文档解析状态，因为此时的文档数据还在网络进程中，并没有提交给渲染进程，所以下一步就进入了提交文档阶段。

4. 提交文档

首先要明确一点，这里的“文档”是指URL请求的响应体数据。

- “提交文档”的消息是由浏览器进程发出的，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。

更新内容如下图所示：



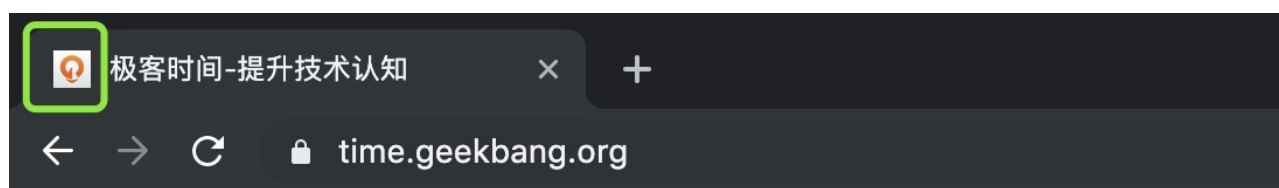
导航完成状态

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？”这个过程极其“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：

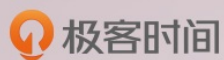
- 服务器可以根据响应头来控制浏览器的行为，如跳转、网络数据类型判断。
- Chrome默认采用每个标签对应一个渲染进程，但是如果两个页面属于同一站点，那这两个标签会使用同一个渲染进程。
- 浏览器的导航过程涵盖了从用户发起请求到提交文档给渲染进程的中间所有阶段。

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。



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：

- ytd 2019-08-13 08:49:03
结合老师的讲义，自己总结了，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。 [3赞]

● mfist 2019-08-13 07:20:53

浏览器打开一个页面需要至少打开四个进程，包括浏览器主进程，渲染进程，插件进程，gpu进程。

1输入url后，浏览器主进程发现变化，通过dns服务解析域名为具体ip地址，如果是https还会通过tls完成密钥交换，进去请求响应。

2收到响应后浏览器进程根据响应状态码判断下一步动作，如果是2xx的话，会将内容提交给渲染进程，当渲染进程反馈给主进程确认收到提交请求后，浏览器的前进后退按钮更衣，显示页面替换为白屏等待渲染进程内容。

3渲染进程根据主进程提交的内容进行解析渲染，碰到资源请求则进行请求加载，当渲染结束时讲内容提交给主进程，主进程收到内容更新界面，将刷新按钮变为完成。

4加载完成，等待下一次任务

这样描述，老师觉得有什么不足吗，谢谢 [2赞]

● Elmer 2019-08-13 10:44:17

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？ [1赞]

● 羽蝶曲 2019-08-13 16:38:33

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 code 301 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

● 羽蝶曲 2019-08-13 16:34:24

请问老师，我刚才从 <https://flutterchina.club/layout/> 打开 <https://flutterchina.club/layout/#> 介绍这个页面，发现两个页面并不是共用一个进程，是为什么呢？要不要同一个站点下的页面共用一个进程可以自己配置的吗？

● 卡卡颂 2019-08-13 14:52:33

导航流程

1. 用户输入阶段：

处理输入，如果是 关键词 调用 默认搜索引擎处理，如果是 url 通过 ipc 传输给 网络进程

2. URL 请求阶段：

1. 本地是否缓存资源，有缓存直接返回缓存资源。没有缓存进入网络请求阶段。

2. 如果没有 DNS 缓存，则解析 DNS 获得目标 IP 地址。如果是 HTTPS 需要建立 TLS 连接。

3. 通过IP地址和端口与目标服务器建立TCP连接。成功后开始构建请求行、请求头（附带域下的COOKIE），向服务器发送构建的信息。
4. 服务器获得信息后，会构建响应信息，发给网络进程。
3. 解析响应阶段：
 1. 解析状态码，如果是301 302，从响应头从获取LOCATION 重新发起请求。如果状态码 200，继续解析响应头。
 2. 解析 Content-Type，判断数据类型，交给不同进程处理(ex: application/octet-stream 交给下载管理器处理，text/html 交给渲染进程)
4. 准备渲染进程阶段：
 1. 如果打开新页面，开新的渲染进程
 2. 如果从A页面打开B页面，且 A、B协议相同，根域名相同，则B复用A的渲染进程。
5. 提交文档阶段：
 1. 浏览器进程发送“提交文档”消息给网络进程，网络进程将请求体发送给渲染进程。
 2. 提交完成后渲染进程返回“确认提交”给浏览器进程，浏览器进程更新前进／后退，tab图标，地址栏URL，web页面。
6. 渲染阶段

● 卡卡颂 2019-08-13 14:51:42
导航流程

1. 用户输入阶段：

处理输入，如果是 关键词 调用 默认搜索引擎处理，如果是 url 通过 ipc 传输给 网络进程
2. URL请求阶段：
 1. 本地是否缓存资源，有缓存直接返回缓存资源。没有缓存进入网络请求阶段。
 2. 如果没有DNS缓存，则解析DNS获得目标IP地址。如果是HTTPS需要建立TLS连接。
 3. 通过IP地址和端口与目标服务器建立TCP连接。成功后开始构建请求行、请求头（附带域下的COOKIE），向服务器发送构建的信息。
 4. 服务器获得信息后，会构建响应信息，发给网络进程。
3. 解析响应阶段：
 1. 解析状态码，如果是301 302，从响应头从获取LOCATION 重新发起请求。如果状态码 200，继续解析响应头。
 2. 解析 Content-Type，判断数据类型，交给不同进程处理(ex: application/octet-stream 交给下载管理器处理，text/html 交给渲染进程)
4. 准备渲染进程阶段：
 1. 如果打开新页面，开新的渲染进程
 2. 如果从A页面打开B页面，且 A、B协议相同，根域名相同，则B复用A的渲染进程。
5. 提交文档阶段：
 1. 浏览器进程发送“提交文档”消息给网络进程，网络进程将请求体发送给渲染进程。
 2. 提交完成后渲染进程返回“确认提交”给浏览器进程，浏览器进程更新前进／后退，tab图标，地址栏URL，web页面。
6. 渲染阶段

● EanCuznaivy 2019-08-13 13:16:41
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2019-08-13 14:34:44
赞

- 许童童 2019-08-13 11:30:10

从输入 URL 到页面展示，这中间发生了什么？

用户输入URL并回车，浏览器进程判断是搜索还是URL地址，是URL地址就通过IPC将URL发送给网络进程，网络进程查找本地缓存，如果没找到，进行DNS解析，利用IP构建TCP连接，三次握手，发送HTTP请求，服务器响应并返回HTTP响应。如果有301，302则执行相应流程。200则继续，判断Content-type如果是text/html，浏览器进程准备渲染进程，发送提交文档消息，让网络进程与渲染进程建立管道，将内容提交给渲染进程，如果是同域建复用父进程的渲染进程。渲染进程接收完毕，向浏览器进程发送确认文档，浏览器安全栏，前进，后退界面更新，渲染进程渲染完后，向浏览器进程提交页面加载完毕。

- Will 2019-08-13 10:41:43

请问老师, Utility: V8 Proxy Resolver是渲染进程吗

- 早起不吃虫 2019-08-13 09:39:24

又回看了第一章，发现老师太贴心了，很多问题都能给予细致的回复，不能更赞，看来下次要先囤货等评论养肥了再一起看了！

- leitong 2019-08-13 01:39:57

有一个疑问，就是如果需要重定向，那不是之前的大段响应体都浪费了，需要销毁了重新获取新的响应资源？

作者回复2019-08-13 07:23:13

对的，就是重头来一次，所以重定向尽量少用。

- leitong 2019-08-13 01:37:29

- 1 浏览器主进程提交url或包含关键字的url给网络进程
- 2 网络进程请求服务器，返回响应头行体，判断是否需要重定向
- 3 网络进程将页面类型的响应资源提交给渲染进程
- 4 渲染进程渲染结束，加载完毕