**經典面試：**

**當你輸入一個網址後回車，實際會發生什麼？**

**Laravel MVC**

**API、restfulAPI、json**

**Session 、cookie 差別**

**常見jQuery的三種不同架構 $(function(), (function($), $.fn.myFunctionName**

# 當你在瀏覽器輸入baidu.com 並按下回車後發生了什麼？

[**Aceyclee**](https://www.zhihu.com/people/Aceyclee)

喧嘩者不真誠

1,146人讚同了該文章

**簡評：**對於這個經典的問題，本文將不再局限於平常的回答，而是想辦法回答地更具體，不遺漏任何細節。[根據「What-happens-when」英文原版，補充修訂了部分內容的翻譯，去掉了一些多餘的章節標題。](https://link.zhihu.com/?target=https%3A//github.com/skyline75489/what-happens-when-zh_CN/commit/0d39aba32b8f493a4a2138b42ef053491965d435)  
**注：**英文原版是某個不存在的網站，故更換為[baidu.com](https://link.zhihu.com/?target=http%3A//baidu.com/)，大家理解原理就可以了，不要diss我，love and peace :)

## 目錄

* 按下「g」鍵
* 回車鍵按下
* 產生中斷（非USB 鍵盤）
* (Windows) 一個WM\_KEYDOWN 消息被發往應用程式
* (Mac OS X) 一個KeyDown NSEvent 被發往應用程式
* (GNU/Linux)Xorg 服務器監聽鍵碼值
* 解析URL
* 輸入的是URL 還是搜索的關鍵字？
* 轉換非ASCII 的Unicode 字元
* 檢查HSTS 列表
* DNS 查詢
* ARP 過程
* 使用通訊端
* TLS 握手
* HTTP 協議
* HTTP 服務器請求處理
* 瀏覽器背後的故事
* 瀏覽器
* HTML 解析
* CSS 解析
* 頁面渲染
* GPU 渲染
* Window Server
* 後期渲染與用戶引發的處理

## ▎按下「g」鍵

接下來的內容介紹了物理鍵盤和系統中斷的工作原理，但是有一部分內容沒有涉及。

當你按下「g」鍵，瀏覽器接收到這個消息之後，會觸發自動完成機制。瀏覽器根據自己的演算法，以及你是否處於隱私瀏覽模式，會在瀏覽器的位址框下方給出輸入建議。大部分演算法會優先考慮根據你的搜索歷史和書籤等內容給出建議。你打算輸入「[http:// google.com](https://link.zhihu.com/?target=http%3A//google.com)」，因此給出的建議並不匹配。但是輸入過程中仍然有大量的代碼在後臺運行，你的每一次按鍵都會使得給出的建議更加準確。甚至有可能在你輸入之前，流覽器就將 「[http://google.com](https://link.zhihu.com/?target=http%3A//google.com" \t "_blank)」建議給你。

## ▎回車鍵按下

為了從零開始，我們選擇鍵盤上的回車鍵被按到最低處作為起點。在這個時刻，一個專用於回車鍵的電流回路被直接地或者通過電容器間接地閉合了，使得少量的電流進入了鍵盤的邏輯電路系統。

這個系統會掃描每個鍵的狀態，對於按鍵開關的電位彈跳變化進行噪音消除 (debounce)，並將其轉化為鍵盤碼值。在這裡，回車的碼值是13。鍵盤控制器在得到碼值之後，將其編碼，用於之後的傳輸。現在這個傳輸過程幾乎都是通過通用序列匯流排 (USB) 或者藍牙 (Bluetooth) 來進行的，以前是通過 PS/2 或者 ADB 連接進行。

**USB 鍵盤：**

* 鍵盤的 USB 元件通過電腦上的 USB 介面與 USB 控制器相連接，USB 介面中的第一號針為它提供了 5V 的電壓
* 鍵碼值存儲在鍵盤內部電路一個叫做「endpoint」的寄存器內
* USB 控制器大概每隔 10ms 便查詢一次「endpoint」以得到存儲的鍵碼值資料，這個最短時間間隔由鍵盤提供
* 鍵值碼值通過 USB 序列介面引擎被轉換成一個或者多個遵循低層 USB 協定的 USB 資料包
* 這些資料包通過 D+ 針或者 D- 針（中間的兩個針），以最高 1.5Mb/s 的速度從鍵盤傳輸至電腦。速度限制是因為人機交互設備總是被聲明成「低速設備」（USB 2.0 compliance）
* 這個串列信號在電腦的 USB 控制器處被解碼，然後被人機交互設備通用鍵盤驅動進行進一步解釋。之後按鍵的碼值被傳輸到作業系統的硬體抽象層

**虛擬鍵盤（觸屏設備）：**

* 在現代電容屏上，當使用者把手指放在螢幕上時，一小部分電流從傳導層的靜電域經過手指傳導，形成了一個回路，使得螢幕上觸控的那一點電壓下降，螢幕控制器產生一個中斷，報告這次「點擊」的座標
* 然後移動作業系統通知當前活躍的應用，有一個點擊事件發生在它的某個 GUI 部件上了，現在這個部件是虛擬鍵盤的按鈕
* 虛擬鍵盤引發一個軟中斷，返回給 OS 一個「按鍵按下」消息
* 這個消息又返回來向當前活躍的應用通知一個「按鍵按下」事件

## ▎產生中斷（非 USB 鍵盤）

鍵盤在它的插斷要求線 (IRQ) 上發送信號，信號會被中斷控制器映射到一個中斷向量，實際上就是一個整型數 。CPU 使用中斷描述符表 (IDT) 把中斷向量映射到對應函數，這些函數被稱為中斷處理器，它們由作業系統內核提供。當一個中斷到達時，CPU 根據 IDT 和中斷向量索引到對應的中斷處理器，然後作業系統內核出場了。

## ▎(Windows) 一個 WM\_KEYDOWN 消息被發往應用程式

HID 把鍵盤按下的事件傳送給 KBDHID.sys 驅動，把 HID 的信號轉換成一個掃描碼(Scancode)，這裡回車的掃描碼是 VK\_RETURN(0x0d)。 KBDHID.sys 驅動和 KBDCLASS.sys（鍵盤類驅動，keyboard class driver）進行交互，這個驅動負責安全地處理所有鍵盤和小鍵盤的輸入事件。之後它又去調用 Win32K.sys ，在這之前有可能把消息傳遞給安裝的協力廠商鍵盤篩檢程式。這些都是發生在核心模式。

Win32K.sys 通過 GetForegroundWindow() API 函數找到當前哪個視窗是活躍的。這個 API 函數提供了當前流覽器的地址欄的控制碼。Windows 系統的「message pump」機制調用 SendMessage (hWnd, WM\_KEYDOWN, VK\_RETURN, lParam) 函數， lParam 是一個用來指示這個按鍵的更多資訊的遮罩，這些資訊包括按鍵重複次數（這裡是 0），實際掃描碼（可能依賴于OEM 廠商，不過通常不會是 VK\_RETURN ），功能鍵（alt, shift, ctrl）是否被按下（在這裡沒有），以及一些其他狀態。

Windows 的 SendMessage API 直接將消息添加到特定視窗控制碼 hWnd 的訊息佇列中，之後賦給 hWnd 的主要消息處理函數 WindowProc 將會被調用，用於處理佇列中的消息。

當前活躍的控制碼 hWnd 實際上是一個edit control控制項，這種情況下，WindowProc 有一個用於處理 WM\_KEYDOWN 消息的處理器，這段代碼會查看 SendMessage 傳入的第三個參數 wParam ，因為這個參數是 VK\_RETURN ，於是它知道用戶按下了回車鍵。

## ▎(Mac OS X) 一個 KeyDown NSEvent 被發往應用程式

中斷信號引發了 I/O Kit Kext 鍵盤驅動的中斷處理事件，驅動把信號翻譯成鍵碼值，然後傳給OS X 的 WindowServer 進程。然後， WindowServer 將這個事件通過 Mach 埠分發給合適的（活躍的，或者正在監聽的）應用程式，這個信號會被放到應用程式的訊息佇列裡。佇列中的消息可以被擁有足夠高許可權的執行緒使用 mach\_ipc\_dispatch 函數讀取到。這個過程通常是由 NSApplication 主事件迴圈產生並且處理的，通過 NSEventType 為 KeyDown 的 NSEvent 。

## ▎(GNU/Linux)Xorg 伺服器監聽鍵碼值

當使用圖形化的 X Server 時，X Server 會按照特定的規則把鍵碼值再一次映射，映射成掃描碼。當這個映射過程完成之後， X Server 把這個按鍵字元發送給視窗管理器（DWM，metacity, i3 等等），視窗管理器再把字元發送給當前視窗。當前視窗使用有關圖形 API 把文字列印在輸入框內。

## ▎解析URL

流覽器通過 URL 能夠知道下面的資訊：

* Protocol "http" 使用 HTTP 協定
* Resource "/" 請求的資源是主頁 (index)

## ▎輸入的是 URL 還是搜索的關鍵字？

當協議或主機名稱不合法時，流覽器會將位址欄中輸入的文字傳給預設的搜尋引擎。大部分情況下，在把文字傳遞給搜尋引擎的時候，URL 會帶有特定的一串字元，用來告訴搜尋引擎這次搜索來自這個特定流覽器。

## ▎轉換非 ASCII 的 Unicode 字元

* 流覽器檢查輸入是否含有不是 a-z， A-Z，0-9， - 或者 . 的字元
* 這裡主機名稱是 [http://google.com](https://link.zhihu.com/?target=http%3A//google.com) ，所以沒有非 ASCII 的字元；如果有的話，流覽器會對主機名稱部分使用 [Punycode](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/Punycode) 編碼

## ▎檢查 HSTS 列表

* 流覽器檢查自帶的「預載入 HSTS（HTTP嚴格傳輸安全）」列表，這個列表裡包含了那些請求流覽器只使用 HTTPS 進行連接的網站
* 如果網站在這個列表裡，流覽器會使用 HTTPS 而不是 HTTP 協定，否則，最初的請求會使用 HTTP 協定發送
* 注意，一個網站哪怕不在 HSTS 列表裡，也可以要求流覽器對自己使用 HSTS 政策進行訪問。流覽器向網站發出第一個 HTTP 請求之後，網站會返回流覽器一個回應，請求流覽器只使用 HTTPS 發送請求。然而，就是這第一個 HTTP 請求，卻可能會使用戶受到 [downgrade attack](https://link.zhihu.com/?target=http%3A//en.wikipedia.org/wiki/SSL_stripping) 的威脅，這也是為什麼現代流覽器都預置了 HSTS 列表。

## ▎DNS 查詢

* 流覽器檢查功能變數名稱是否在緩存當中（要查看 Chrome 當中的緩存， 打開 chrome://net-internals/#dns）。
* 如果緩存中沒有，就去調用 gethostbyname 庫函數（作業系統不同函數也不同）進行查詢。
* gethostbyname 函數在試圖進行DNS解析之前首先檢查功能變數名稱是否在本地 Hosts 裡，Hosts 的位置[不同的作業系統有所不同](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/Hosts_%2528file%2529%23Location_in_the_file_system" \t "_blank)
* 如果gethostbyname 沒有這個功能變數名稱的緩存記錄，也沒有在hosts 裡找到，它將會向DNS 服務器發送一條DNS 查詢請求。DNS 服務器是由網絡通信棧提供的，通常是本地路由器或者ISP 的緩存DNS 服務器。
* 查詢本地DNS 服務器
* 如果DNS 服務器和我們的主機在同一個子網內，系統會按照下面的ARP 過程對DNS 服務器進行ARP查詢
* 如果DNS 服務器和我們的主機在不同的子網，系統會按照下面的ARP 過程對默認網關進行查詢

## ▎ARP 過程

要想發送ARP（位址解析協議）廣播，我們需要有一個目標IP 位址，同時還需要知道用於發送ARP 廣播的介面的MAC 位址。

* 首先查詢ARP 緩存，如果緩存命中，我們返回結果：目標IP = MAC

如果緩存沒有命中：

* 查看路由表，看看目標IP 地址是不是在本地路由表中的某個子網內。是的話，使用跟那個子網相連的介面，否則使用與默認網關相連的介面。
* 查詢選擇的網絡介面的MAC 位址
* 我們發送一個二層（[OSI模型](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/OSI_model)中的數據鏈路層）ARP請求：

ARP Request:

Sender MAC: interface:mac:address:here

Sender IP: interface.ip.goes.here

Target MAC: FF:FF:FF:FF:FF:FF (Broadcast)

Target IP: target.ip.goes.here

根據連接主機和路由器的硬體類型不同，可以分為以下幾種情況：

直連：

* 如果我們和路由器是直接連接的，路由器會返回一個ARP Reply （見下麵）。

集線器：

* 如果我們連接到一個集線器，集線器會把ARP 請求向所有其它埠廣播，如果路由器也“連接”在其中，它會返回一個ARP Reply 。

交換機：

* 如果我們連接到了一個交換機，交換機會檢查本地CAM/MAC 表，看看哪個埠有我們要找的那個MAC 地址，如果沒有找到，交換機會向所有其它埠廣播這個ARP 請求。
* 如果交換機的MAC/CAM 表中有對應的條目，交換機會向有我們想要查詢的MAC 位址的那個埠發送ARP 請求
* 如果路由器也“連接”在其中，它會返回一個ARP Reply

ARP Reply:

Sender MAC: target:mac:address:here

Sender IP: target.ip.goes.here

Target MAC: interface:mac:address:here

Target IP: interface.ip.goes.here

現在我們有了DNS 服務器或者默認網關的IP 位址，我們可以繼續DNS 請求了：

* 使用53 埠向DNS 服務器發送UDP 請求包，如果響應包太大，會使用TCP 協議
* 如果本地/ISP DNS 服務器沒有找到結果，它會發送一個遞歸查詢請求，一層一層向高層DNS 服務器做查詢，直到查詢到起始授權機構，如果找到會把結果返回

## ▎使用通訊端

當瀏覽器得到了目標服務器的IP 地址，以及URL 中給出來埠號（http 協議默認埠號是80， https 默認埠號是443），它會調用系統庫函數socket ，請求一個TCP 流通訊端，對應的參數是AF\_INET/AF\_INET6 和SOCK\_STREAM 。

* 這個請求首先被交給傳輸層，在傳輸層請求被封裝成TCP segment。目標埠會被加入頭部，源埠會在系統內核的動態埠範圍內選取（Linux下是ip\_local\_port\_range）
* TCP segment 被送往網絡層，網絡層會在其中再加入一個IP 頭部，裡麵包含了目標服務器的IP 地址以及本機的IP 地址，把它封裝成一個TCP packet。
* 這個TCP packet 接下來會進入鏈路層，鏈路層會在封包中加入frame 頭部，裡麵包含了本地內置網卡的MAC 地址以及網關（本地路由器）的MAC 地址。像前面說的一樣，如果內核不知道網關的MAC 地址，它必須進行ARP 廣播來查詢其地址。

到了現在，TCP 封包已經準備好了，可以使用下面的方式進行傳輸：

* [乙太網](https://link.zhihu.com/?target=http%3A//en.wikipedia.org/wiki/IEEE_802.3)
* [WiFi](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/IEEE_802.11)
* [蜂窩數據網絡](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/Cellular_data_communication_protocol)

對於大部分家庭網絡和小型企業網絡來說，封包會從本地計算機出發，經過本地網絡，再通過調製解調器把數字信號轉換成模擬信號，使其適於在電話線路，有線電視光纜和無線電話線路上傳輸。在傳輸線路的另一端，是另外一個調製解調器，它把模擬信號轉換回數字信號，交由下一個[網絡節點](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/Computer_network%23Network_nodes)處理。節點的目標地址和源地址將在後面討論。

大型企業和比較新的住宅通常使用光纖或直接乙太網連接，這種情況下信號一直是數位的，會被直接傳到下一個[網路節點](https://link.zhihu.com/?target=https%3A//en.wikipedia.org/wiki/Computer_network%23Network_nodes)進行處理。

最終封包會到達管理本地子網的路由器。在那裡出發，它會繼續經過自治區域（autonomous system, 縮寫 AS）的邊界路由器，其他自治區域，最終到達目標伺服器。一路上經過的這些路由器會從IP資料報頭部裡提取出目標位址，並將封包正確地路由到下一個目的地。IP 資料報頭部 time to live (TTL) 域的值每經過一個路由器就減 1，如果封包的 TTL 變為 0，或者路由器由於網路擁堵等原因封包佇列滿了，那麼這個包會被路由器丟棄。

上面的發送和接受過程在 TCP 連接期間會發生很多次：

* 用戶端選擇一個初始序號 (ISN)，將設置了 SYN 位的封包發送給伺服器端，表明自己要建立連接並設置了初始序號
* 伺服器端接收到 SYN 包，如果它可以建立連接：
  + 伺服器端選擇它自己的初始序號
  + 伺服器端設置 SYN 位，表明自己選擇了一個初始序號
  + 伺服器端把（用戶端ISN + 1）複製到 ACK 域，並且設置 ACK 位，表明自己接收到了用戶端的第一個封包
* 用戶端通過發送下面一個封包來確認這次連接：
  + 自己的序號+1
  + 接收端 ACK+1
  + 設置 ACK 位
* 資料通過下面的方式傳輸：
  + 當一方發送了N個 Bytes 的資料之後，將自己的 SEQ 序號也增加N
  + 另一方確認接收到這個資料包（或者一系列資料包）之後，它發送一個 ACK 包，ACK 的值設置為接收到的資料包的最後一個序號
* 關閉連接時：
  + 要關閉連接的一方發送一個 FIN 包
  + 另一方確認這個 FIN 包，並且發送自己的 FIN 包
  + 要關閉的一方使用 ACK 包來確認接收到了 FIN

## ▎TLS 握手

* 用戶端發送一個 ClientHello 消息到伺服器端，消息中同時包含了它的 Transport Layer Security (TLS) 版本，可用的加密演算法和壓縮演算法。
* 伺服器端向用戶端返回一個 ServerHello 消息，消息中包含了伺服器端的 TLS 版本，伺服器所選擇的加密和壓縮演算法，以及數位憑證認證機構（Certificate Authority，縮寫 CA）簽發的伺服器公開證書，證書中包含了公開金鑰。用戶端會使用這個公開金鑰加密接下來的握手過程，直到協商生成一個新的對稱金鑰
* 用戶端根據自己的信任 CA 清單，驗證伺服器端的證書是否可信。如果認為可信，用戶端會生成一串偽亂數，使用伺服器的公開金鑰加密它。這串亂數會被用於生成新的對稱金鑰
* 伺服器端使用自己的私密金鑰解密上面提到的亂數，然後使用這串亂數產生自己的對稱主金鑰
* 用戶端發送一個 Finished 消息給伺服器端，使用對稱式金鑰密碼編譯這次通訊的一個散列值
* 伺服器端生成自己的 hash 值，然後解密用戶端發送來的資訊，檢查這兩個值是否對應。如果對應，就向用戶端發送一個 Finished 消息，也使用協商好的對稱式金鑰密碼編譯
* 從現在開始，接下來整個 TLS 會話都使用對稱秘鑰進行加密，傳輸應用層（HTTP）內容

## ▎HTTP 協議

如果流覽器是 Google 出品的，它不會使用 HTTP 協定來獲取頁面資訊，而是會與伺服器端發送請求，商討使用 SPDY 協定。

如果流覽器使用 HTTP 協定而不支援 SPDY 協定，它會向伺服器發送這樣的一個請求:

GET / HTTP/1.1

Host: google.com

Connection: close

[other headers]

[other headers] 包含了一系列的由冒號分割開的鍵值對，它們的格式符合 HTTP 協議標準，它們之間由一個分行符號分割開來。（這裡我們假設流覽器沒有違反 HTTP 協議標準的 bug，同時假設流覽器使用 HTTP/1.1 協議，不然的話頭部可能不包含 Host 欄位，同時 GET 請求中的版本號會變成 HTTP/1.0 或者 HTTP/0.9 。）

HTTP/1.1 定義了「關閉連接」的選項「close」，發送者使用這個選項指示這次連接在回應結束之後會斷開。例如：

Connection:close

不支援持久連接的 HTTP/1.1 應用必須在每條消息中都包含「close」選項。

在發送完這些請求和頭部之後，流覽器發送一個分行符號，表示要發送的內容已經結束了。

伺服器端返回一個回應碼，指示這次請求的狀態，回應的形式是這樣的:

200 OK

[response headers]

然後是一個換行，接下來有效載荷 (payload)，也就是 [http://www.google.com](https://link.zhihu.com/?target=http%3A//www.google.com) 的 HTML 內容。伺服器下面可能會關閉連接，如果用戶端請求保持連接的話，伺服器端會保持連接打開，以供之後的請求重用。

如果流覽器發送的 HTTP 頭部包含了足夠多的資訊（例如包含了 Etag 頭部），以至於伺服器可以判斷出，流覽器緩存的檔版本自從上次獲取之後沒有再更改過，伺服器可能會返回這樣的回應:

304 Not Modified

[response headers]

這個回應沒有有效載荷，流覽器會從自己的緩存中取出想要的內容。

在解析完 HTML 之後，流覽器和用戶端會重複上面的過程，直到 HTML 頁面引入的所有資源（圖片，CSS，favicon.ico 等等）全部都獲取完畢，區別只是頭部的 GET / HTTP/1.1 會變成 GET /$(相對[http://www.google.com](https://link.zhihu.com/?target=http%3A//www.google.com)的URL) HTTP/1.1 。

如果 HTML 引入了 [http://www.google.com](https://link.zhihu.com/?target=http%3A//www.google.com) 功能變數名稱之外的資源，流覽器會回到上面解析功能變數名稱那一步，按照下面的步驟往下一步一步執行，請求中的 Host 頭部會變成另外的功能變數名稱。

## ▎HTTP 伺服器請求處理

HTTPD(HTTP Daemon) 在伺服器端處理請求 / 回應。最常見的 HTTPD 有 Linux 上常用的 Apache 和 nginx，以及 Windows 上的 IIS。

* HTTPD 接收請求
* 伺服器把請求拆分為以下幾個參數：
  + HTTP 請求方法（GET, POST, HEAD, PUT, DELETE, CONNECT, OPTIONS, 或者 TRACE）。直接在位址欄中輸入 URL 這種情況下，使用的是 GET 方法
  + 功能變數名稱：[http://google.com](https://link.zhihu.com/?target=http%3A//google.com)
  + 請求路徑/頁面：/ (我們沒有請求[http://google.com](https://link.zhihu.com/?target=http%3A//google.com)下的指定的頁面，因此 / 是預設的路徑)
* 伺服器驗證其上已經配置了 [http://google.com](https://link.zhihu.com/?target=http%3A//google.com) 的虛擬主機
* 伺服器驗證 [http://google.com](https://link.zhihu.com/?target=http%3A//google.com) 接受 GET 方法
* 伺服器驗證該使用者可以使用 GET 方法（根據 IP 位址，身份資訊等）
* 如果伺服器安裝了 URL 重寫模組（例如 Apache 的 mod\_rewrite 和 IIS 的 URL Rewrite），伺服器會嘗試匹配重寫規則，如果匹配上的話，伺服器會按照規則重寫這個請求
* 伺服器根據請求資訊獲取相應的回應內容，這種情況下由於訪問路徑是「/」，會訪問首頁檔（你可以重寫這個規則，但是這個是最常用的）。
* 伺服器會使用指定的處理常式分析處理這個檔，假如 Google 使用 PHP，伺服器會使用 PHP 解析 index 檔，並捕獲輸出，把 PHP 的輸出結果返回給請求者

## ▎流覽器背後的故事

當伺服器提供了資源之後（HTML，CSS，JS，圖片等），流覽器會執行下麵的操作：

* 解析 —— HTML，CSS，JS
* 渲染 —— 構建 DOM 樹 -> 渲染 -> 佈局 -> 繪製

## ▎流覽器

流覽器的功能是從伺服器上取回你想要的資源，然後展示在流覽器視窗當中。資源通常是 HTML 檔，也可能是 PDF，圖片，或者其他類型的內容。資源的位置通過用戶提供的 URI (Uniform Resource Identifier) 來確定。

流覽器解釋和展示 HTML 檔的方法，在 HTML 和 CSS 的標準中有詳細介紹。這些標準由 Web 標準組織 W3C (World Wide Web Consortium) 維護。

不同流覽器的使用者介面大都十分接近，有很多共同的 UI 元素：

* 一個位址欄
* 後退和前進按鈕
* 書簽選項
* 刷新和停止按鈕
* 主頁按鈕

**流覽器高層架構**

組成流覽器的元件有：

* **使用者介面** 使用者介面包含了位址欄，前進後退按鈕，書簽功能表等等，除了請求頁面之外所有你看到的內容都是使用者介面的一部分
* **流覽器引擎** 流覽器引擎負責讓 UI 和渲染引擎協調工作
* **渲染引擎** 渲染引擎負責展示請求內容。如果請求的內容是 HTML，渲染引擎會解析 HTML 和 CSS，然後將內容展示在螢幕上
* **網路元件** 網路元件負責網路調用，例如 HTTP 請求等，使用一個平臺無關介面，下層是針對不同平臺的具體實現
* **UI後端** UI 後端用於繪製基本 UI 元件，例如下拉式清單方塊和視窗。UI 後端暴露一個統一的平臺無關的介面，下層使用作業系統的 UI 方法實現
* **Javascript 引擎** Javascript 引擎用於解析和執行 Javascript 代碼
* **資料存儲** 資料存儲元件是一個持久層。流覽器可能需要在本機存放區各種各樣的資料，例如 Cookie 等。流覽器也需要支援諸如 localStorage，IndexedDB，WebSQL 和 FileSystem 之類的存儲機制

## ▎HTML 解析

流覽器渲染引擎從網路層取得請求的文檔，一般情況下文檔會分成 8kB 大小的分塊傳輸。

HTML 解析器的主要工作是對 HTML 文檔進行解析，生成解析樹。

解析樹是以 DOM 元素以及屬性為節點的樹。DOM 是文檔物件模型 (Document Object Model)的縮寫，它是 HTML 文檔的物件表示，同時也是 HTML 元素面向外部（如 JavaScript）的介面。樹的根部是「Document」對象。整個 DOM 和 HTML 文檔幾乎是一對一的關係。

**解析演算法**

HTML 不能使用常見的自頂向下或自底向上方法來進行分析。主要原因有以下幾點:

* 語言本身的「寬容」特性
* HTML 本身可能是殘缺的，對於常見的殘缺，流覽器需要有傳統的容錯機制來支持它們
* 解析過程需要反復。對於其他語言來說，源碼不會在解析過程中發生變化，但是對於 HTML 來說，動態代碼，例如腳本元素中包含的 document.write() 方法會在源碼中添加內容，也就是說，解析過程實際上會改變輸入的內容

由於不能使用常用的解析技術，流覽器創造了專門用於解析 HTML 的解析器。解析演算法在 HTML5 標準規範中有詳細介紹，演算法主要包含了兩個階段：標記化（tokenization）和樹的構建。

**解析結束之後**

流覽器開始載入網頁的外部資源（CSS，圖像，Javascript 檔等）。

此時流覽器把文檔標記為可交互的（interactive），流覽器開始解析處於「推遲（deferred）」模式的腳本，也就是那些需要在文檔解析完畢之後再執行的腳本。之後文檔的狀態會變為「完成（complete）」，流覽器會觸發「載入（load）」事件。

注意解析 HTML 網頁時永遠不會出現「無效語法（Invalid Syntax）」錯誤，流覽器會修復所有錯誤內容，然後繼續解析。

## ▎CSS 解析

* 根據 [CSS詞法和句法](https://link.zhihu.com/?target=http%3A//www.w3.org/TR/CSS2/grammar.html) 分析 CSS 檔和 <style> 標籤包含的內容以及 style 屬性的值
* 每個 CSS 檔都被解析成一個樣式表物件（StyleSheet object），這個物件裡包含了帶有選擇器的 CSS 規則，和對應 CSS 語法的物件
* CSS 解析器可能是自頂向下的，也可能是使用解析器生成器生成的自底向上的解析器

## ▎頁面渲染

* 通過遍歷 DOM 節點樹創建一個「Frame 樹」或「渲染樹」，並計算每個節點的各個 CSS 樣式值
* 通過累加子節點的寬度，該節點的水準內邊距 (padding)、邊框 (border) 和外邊距(margin)，自底向上的計算「Frame 樹」中每個節點的首選 (preferred) 寬度
* 通過自頂向下的給每個節點的子節點分配可行寬度，計算每個節點的實際寬度
* 通過應用文字折行、累加子節點的高度和此節點的內邊距 (padding)、邊框 (border) 和外邊距 (margin)，自底向上的計算每個節點的高度
* 使用上面的計算結果構建每個節點的座標
* 當存在元素使用 floated，位置有 absolutely 或 relatively 屬性的時候，會有更多複雜的計算，詳見 [http://dev.w3.org/csswg/css2/](https://link.zhihu.com/?target=http%3A//dev.w3.org/csswg/css2/) 和 [http://www.w3.org/Style/CSS/current-work](https://link.zhihu.com/?target=http%3A//www.w3.org/Style/CSS/current-work)
* 創建 layer (層) 來表示頁面中的哪些部分可以成組的被繪製，而不用被重新柵格化處理。每個幀物件都被分配給一個層
* 頁面上的每個層都被分配了紋理
* 每個層的幀物件都會被遍歷，電腦執行繪圖命令繪製各個層，此過程可能由 CPU 執行柵格化處理，或者直接通過 D2D / SkiaGL 在 GPU 上繪製
* 上面所有步驟都可能利用到最近一次頁面渲染時計算出來的各個值，這樣可以減少不少計算量
* 計算出各個層的最終位置，一組命令由 Direct3D/OpenGL 發出，GPU 命令緩衝區清空，命令傳至 GPU 並非同步渲染，幀被送到 Window Server。

## ▎GPU 渲染

* 在渲染過程中，圖形處理層可能使用通用用途的 CPU，也可能使用圖形處理器 GPU
* 當使用 GPU 用於圖形渲染時，圖形驅動軟體會把任務分成多個部分，這樣可以充分利用 GPU 強大的平行計算能力，用於在渲染過程中進行大量的浮點計算。

## ▎Window Server

## ▎後期渲染與用戶引發的處理

渲染結束後，流覽器根據某些時間機制運行 JavaScript 代碼（比如 Google Doodle 動畫）或與用戶交互（在搜索欄輸入關鍵字獲得搜索建議）。類似 Flash 和 Java 的外掛程式也會運行，儘管 Google 主頁裡沒有。這些腳本可以觸發網路請求，也可能改變網頁的內容和佈局，產生又一輪渲染與繪製。