What happens when you type a URL in browser

Pankaj Pal (/Member/Profile/1/Pankypal) 30 May, 2014 11 (http://edusagar.com/articles/view/70/What-happens-when-you-type-a-URL-in-browser#disqus_thread) Comments 112.06K

Apart from being a very common interview question, this is one of the very first query which lingers around in our mind every time we type a URL in a browser. Here is an attempt to satiate this quest while we delve into the details of what happens in the background when we type a URL in our browsers.

Step 1. URL is typed in the browser.

Step 2. If requested object is in browser cache and is fresh, move on to Step 8.

Step 3. DNS lookup to find the ip address of the server

when we want to connect to ***google.com (http://google.com)***, we actually want to reach out to a server where google web services are hosted. One such server is having an ip address of ***74.125.236.65***. Now, if you type "***http://74.125.236.65 (http://74.125.236.65)***" in your browser, this will take you to google home page itself. Which means, "***http://google.com (http://google.com)***" and "***http://74.125.236.65 (http://74.125.236.65)***" are nothing but same stuff. But, it is not so. Google has multiple servers in multiple locations to cater to the huge volume of requests they receive per second. Thus we should let Google decide which server is best suited to our needs. Using "google.com (http://google.com)" does the job for us. When we type "google.com (http://google.com)", DNS(Domain Name System) (http://en.wikipedia.org/wiki/Domain_Name_System) services comes into play and resolves the URL to a proper ip address.

Following is a summary of steps happening while DNS service is at work:

- ***Check browser cache***: browsers maintain cache of DNS records for some fixed duration. So, this is the first place to resolve DNS queries.
- ***Check OS cache***: if browser doesn't contain the record in its cache, it makes a system call to underlying Operating System to fetch the record as OS also maintains a cache of recent DNS queries.
- ***Router Cache***: if above steps fail to get a DNS record, the search continues to your router which has its own cache.
- ***ISP cache***: if everything fails, the search moves on to your ISP. First, it tries in its cache, if not found - ISP's DNS ***recursive search*** comes into picture. DNS lookup is again a complex process which finds the appropriate ip address from a list of many options available for websites like Google. You can read more about this here (http://en.wikipedia.org/wiki/Domain_Name_System#Address_resolution_mechanism).

For the DNS enthusiasts - here (https://webhostinggeeks.com/guides/dns/) is a great guide worth reading.

Step 4. Browser initiates a TCP connection with the server.

Step 5. Browser sends a HTTP request to the server.

Browser sends a ***GET*** request to the server according to the specification of HTTP(Hyper Text Transfer Protocol) (http://www.w3.org/Protocols/) protocol.

```
GET http://google.com/ (http://google.com/) HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:29.0) Gecko/20100101 Firefox/29.0
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: google.com (http://google.com)
Cookie: datr=1265876274-[...]; locale=en_US; lsd=WW[...]; c_user=2101[...]
```

Here, browser passes some meta information in the form of headers to the server along with the URL - "http://google.com (http://google.com)". ***User-Agent*** header specifies the browser properties, ***Accept-Encoding*** headers specify the type of responses it will accept. ***Connection*** header tells the server to keep open the TCP connection established here. The request also contains ***Cookies***, which are meta information stored at the client end and contain previous browsing session information for the same website in the form of key-value pairs e.g. the login name of the user for Google.

A quick guide to HTTP specification can be found here (http://www.jmarshall.com/easy/http/).

Step 6. Server handles the incoming request

HTTP request made from browsers are handled by a special software running on server - commonly known as ***web servers*** e.g. ***Apache***, ***IIS*** etc. Web server passes on the request to the proper request handler - a program written to handle web services e.g. ***PHP, ASP.NET, Ruby, Servlets*** etc.

For example URL- ***http://edusagar.com/index.php (http://edusagar.com/index.php)*** is handled by a program written in PHP file - ***index.php***. As soon as GET request for index.php is received, Apache(our webserver at edusagar.com (http://edusagar.com)) prepares the environment to execute index.php file. Now, this php program will generate a response - in our case a HTML response. This response is then sent back to the browser according to HTTP guidelines.

Step 7. Browser receives the HTTP response

```
HTTP/1.1 200 OK
Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Connection: Keep-Alive
Content-length: 1215
Date: Fri, 30 May 2014 08:10:15 GMT

.........<some blob> ..............
```

HTTP response starts with the returned status code from the server. Following is a very brief summary of what a status code denotes:

- 1xx indicates an informational message only
- 2xx indicates success of some kind
- 3xx redirects the client to another URL
- 4xx indicates an error on the client's part
- 5xx indicates an error on the server's part

Server sets various other headers to help browser render the proper content. ***Content-Type*** tells the type of the content the browser has to show, ***Content-length*** tells the number of bytes of the response. Using the ***Content-Encoding*** header's value, browsers can decode the blob data present at the end of the response.

Step 8. Browsers displays the html content

Rendering of html content is also done in phases. The browser first renders the bare bone html structure, and then it sends multiple GET requests to fetch other hyper linked stuff e.g. If the html response contains an image in the form of img tags such as ******, browser will send a HTTP GET request to the server to fetch the image following the complete set of steps which we have seen till now. But this isn't that bad as it looks. Static files like ***images, javascript, css*** files are all cached by the browser so that in future it doesn't have to fetch them again.

Step 9. Client interaction with server

Once a html page is loaded, there are several ways a user can interact with the server. For example, he call fill out a login form to sign in to the website. This also follows all the steps listed above, the only difference is that the HTTP request this time would be a ***POST*** instead of GET and along with that request, browser will send the form data to the server for processing (username and password in this case).

Once server authenticates the user, it will send the proper HTML content(may be user's profile) back to the browser and thus user will see that new webpage after his login request is processed.

Step 10. AJAX queries

Another form of client interaction with server is through AJAX(Asynchronous JavaScript And XML) (http://en.wikipedia.org/wiki/Ajax_%28programming%29) requests. This is an asynchronous GET/POST request to which server can send a response back in a variety of ways - ***json, xml, html*** etc. AJAX requests doesn't hinder the current view of the webpage and work in the background. Because of this, one can dynamically modify the content of a webpage by calling an AJAX request and updating the web elements using Javascript.

Hopefully, that gives you an idea of what happens in the background when we do a really simple operation which is type a URL in a browser.