브라우저 렌더링 과정

1. www.google.com 을 입력하면 URL 주소 중, 도메인 이름에 해당하는 google.com 을 DNS 서버에서 검색한다.

> **들어가기 전에! DNS란?**
>
> 네트워크 상의 모든 PC는 127.1.1.1과 같은 고유한 IP 주소를 가지고 있다. 그러나 모든 IP 주소가 도메인 이름을 가지는 것은 아니기 때문에 보통 도메인 이름은 일정 기간 동안 대여하여 사용한다. 예를 들어 IP 주소가 1.1.1.1 이면 이는 사용자가 보기 편하지 않기 때문에 naver.com 과 같은 도메인 이름을 일정 기간 대여해 사용한다.
>
> 브라우저에서 도메인 이름을 입력해 해당 사이트로 이동하기 위해서는 해당 도메인 이름과 매칭된 IP 주소를 확인하는 작업이 반드시 필요하고, 네트워크에는 이를 위한 서버가 별도로 있다. 이 서버를 Domain Name System Server 즉, **DNS Server** 라고 한다.
>
> 간단하게 말하자면 **DNS는 호스트의 도메인 이름을 IP 주소로 변환하거나 반대의 경우를 수행할 수 있도록 개발된 시스템**인 것이다.

웹 브라우저는 DNS 서버에 도메인 이름을 검색하기 전에 캐싱된 DNS 기록들을 먼저 확인한다. 이전에 해당 도메인을 찾아 들어간 적이 있다면 캐싱된 기록이 있을 것이고 이를 반환하면 훨씬 빠른 속도로 해당 사이트에 입장할 수 있다.

일치하는 IP 주소가 존재하지 않는다면, 다음 과정인 DNS 서버 요청으로 넘어간다.

2. 가장 가까운 DNS 서버에서 해당 도메인 이름에 해당하는 IP 주소를 찾아 사용자가 입력한 URL 정보와 함께 전달한다.

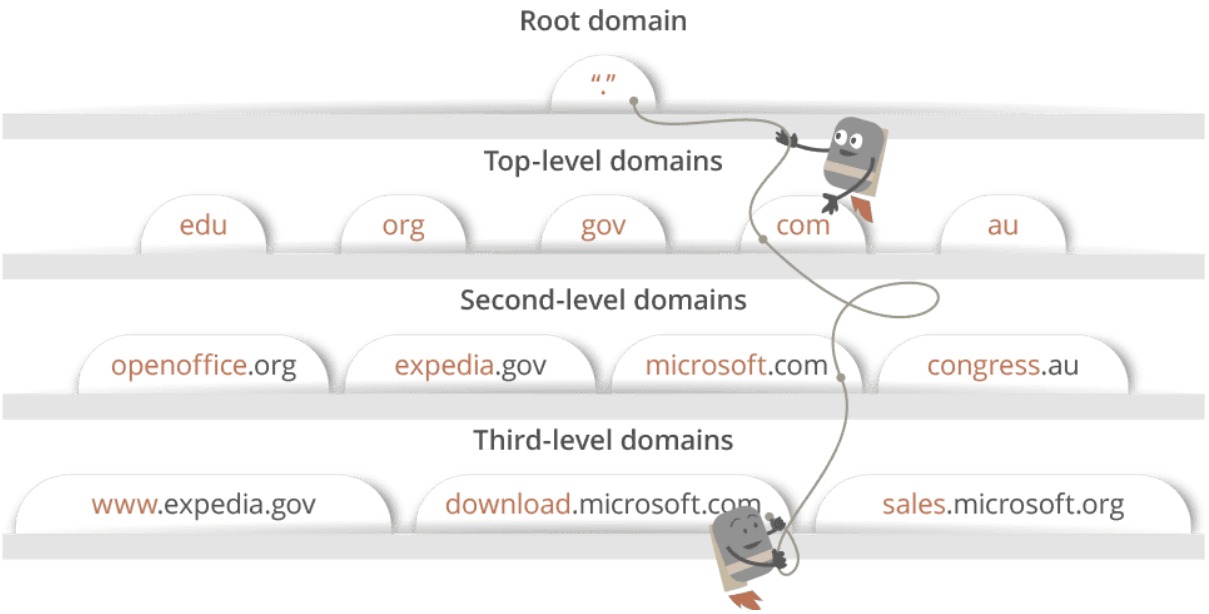
ISP(Internet Service Provider. 예시로는 SK 브로드밴드, KT, ...) 를 통해 DNS 서버가 호스팅하고 있는 서버의 IP 주소를 찾기 위한 DNS query를 전달한다.

DNS query는 현재 DNS 서버에 원하는 IP 주소가 없다면 다른 DNS 서버를 방문하는 과정을 원하는 IP 주소를 찾을 때까지 반복한다.

이제 찾았으면 도메인 이름을 찾은 IP 주소로 변환해야 한다. 이때, 변환 과정은 점(.)을 기준으로 계층적으로 진행된다.

예를 들어, google.com 이면 . → com → google 순으로 탐색하며 IP 주소로 변환된다.

아래 사진과 같이 Root domain에서 시작해 Top → Second → Third 레벨 도메인 순서로 탐색하며 IP 주소로 변환된다.



위와 같이 Local DNS 서버가 여러 DNS 서버에게 차례대로 물어봐 답을 찾는 과정을 **Recursive Query** 라고 한다.

3. 전달받은 IP 주소를 이용해 웹 브라우저는 웹 서버에게 해당 웹 사이트에 맞는 html 문서를 요청한다.

HTTP 요청 메세지는 TCP/IP 프로토콜을 사용해 서버로 전송된다.

> **TCP/IP?**
>
> **TCP** 는 전송 제어 프로토콜로 데이터의 전송을 제어하고 데이터를 어떻게 보낼 지 정한다.
>
> **IP** 는 비신뢰성과 비연결성이라는 특성으로 인해 그 자체만으로는 통신이 불가능하다. 그렇기에, 신뢰성과 연결성을 책임지는 TCP를 활용해 통신을 한다.

TCP는 3 way handshake 과정을 통해 연결 및 데이터를 수신받고, 4 way handshake 과정을 통해 연결을 종료한다.

> **3 way handshake?**
>
> 1. A 클라이언트는 B 서버에 접속을 요청하는 SYN 패킷을 전송한다.
> 2. B 서버는 SYN 요청을 받고 A 클라이언트에게 요청을 수락한다는 SYN_ACK flag가 설정된 패킷을 전송한다.
> 3. A 클라이언트는 B 서버에게 ACK를 전송 후, 연결이 이루어지고 데이터가 오고 가게 된다.
>
> **여기서 SYN 은 연결 요청 플래그, ACK는 상대방으로부터 패킷을 받았다는 것을 알려주는 패킷이다.**

> **4 way handshake**
>
> 1. 클라이언트가 연결을 종료하겠다는 FIN 플래그를 전송한다.
> 2. 서버는 확인 메세지 ACK 를 보낸 후, 자신의 통신이 끝날 때까지 기다린다.
> 3. 서버의 통신이 끝났으면 연결이 종료되었다고 클라이언트에 FIN 플래그를 전송한다.
> 4. 클라이언트는 확인했다는 메세지 ACK를 보낸다.
>
> **여기서 FIN은 연결 종료 요청 플래그 이다.**

만약 클라이언트에서 세션을 종료시킨 뒤 도착하는 패킷이 있다면 해당 패킷은 Drop되고 데이터는 유실되게 된다.

이러한 현상을 방지하기 위해 클라이언트는 서버로부터 FIN 패킷을 수신하더라도 일정 시간 동안 세션을 남겨놓고 잉여 패킷을 기다리는 **TIME_WAIT** 과정을 마지막으로 거치게 된다.

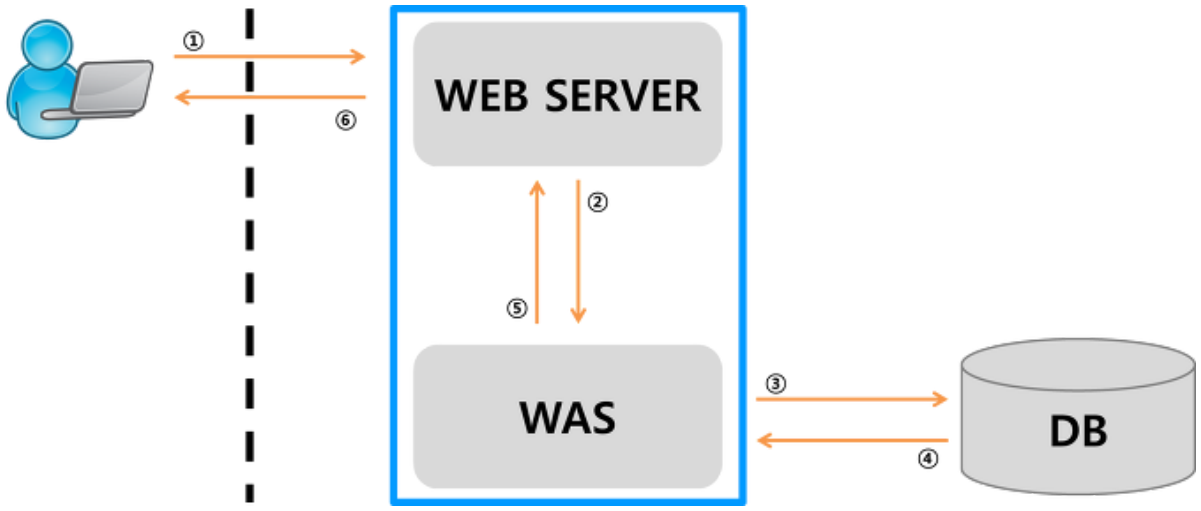
4. WAS와 데이터베이스에서 웹페이지 작업을 처리한다.

웹 서버 혼자서 모든 로직 처리 및 데이터 관리를 하게 되면 서버에 과부하가 일어날 가능성이 높다. 그렇기에 서버의 일을 돕는 조력자 역할을 하는 것이 WAS 입니다.

WAS는 사용자의 컴퓨터나 장치에 웹 애플리케이션을 수행해주는 미들웨어 입니다.

특정 데이터 요청을 브라우저로부터 받게 되면, 웹 서버는 페이지의 로직이나 데이터베이스의 연동을 위해 WAS에게 이들의 처리를 요청합니다. WAS는 해당 요청을 통해 동적인 페이지 처리를 담당하고, DB에서 필요한 데이터 정보를 받아 그에 맞는 파일을 생성합니다.

즉, 웹 서버는 정적인 파일(HTML, CSS, 이미지 파일)을 처리하고 WAS는 동적인 파일(JS, TS)을 처리합니다.



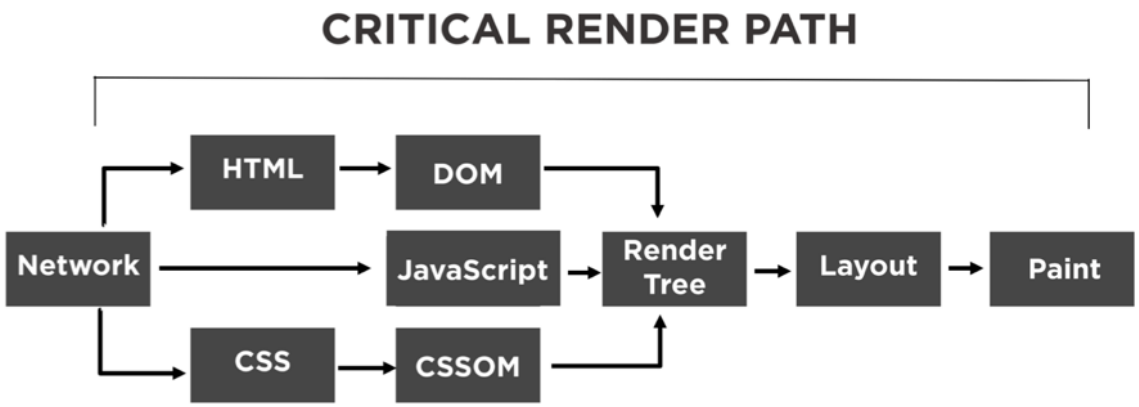
5. WAS에서의 작업 처리 결과들을 웹 서버로 전송하고, 웹 서버는 웹 브라우저에게 html 문서 결과를 전달합니다.

전달 과정에서 status code를 통해 서버 요청에 따른 결과 및 상태를 전달합니다.

- 1xx : 정보가 담긴 메세지
- 2xx : 성공
- 3xx : 클라이언트를 다른 URL로 리다이렉트
- 4xx : 클라이언트 측 에러 발생
- 5xx : 서버 측 에러 발생

6. Critical Rendering Path를 통해 웹 브라우저 화면에 웹 페이지 내용을 출력한다.

웹 브라우저에 출력되는 단계를 Critical Rendering Path라고 하며 크게 6단계로 분류된다. 성능 최적화를 위해선 수신 받은 HTML, CSS, JS 파일들이 어떤 단계에서 어떻게 되는지 파악한 후, 해당 과정을 최소화하는 것이 매우 중요하다.



이제 Critical Rendering Path 과정을 뜯어본다.

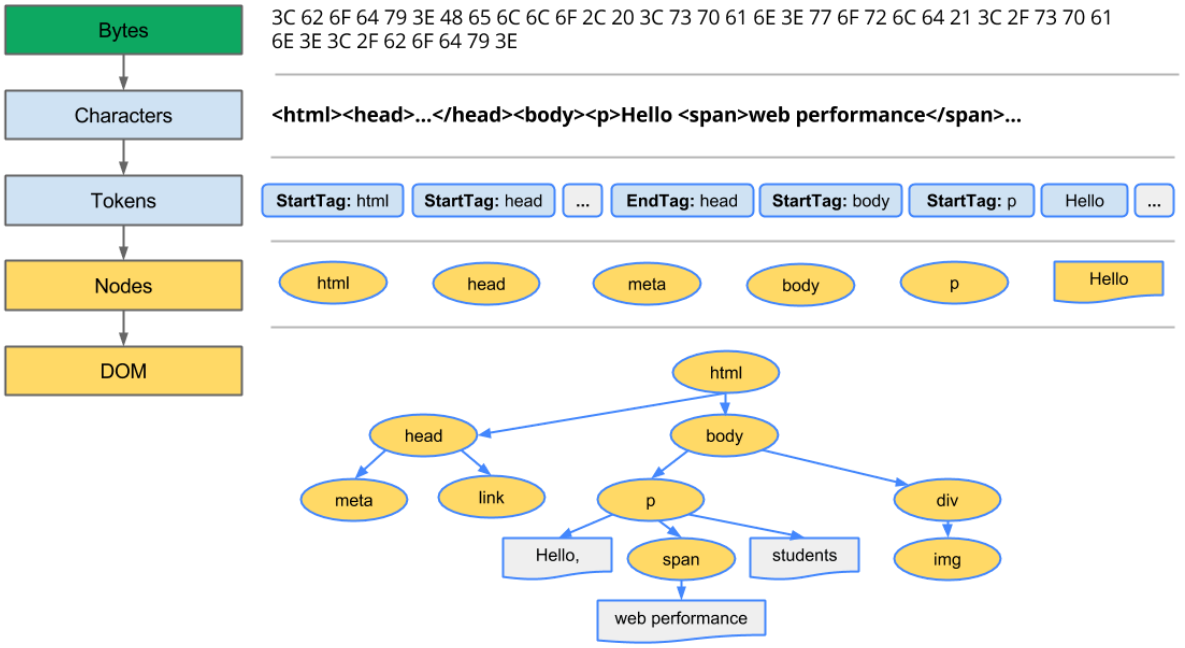
7. DOM 트리 빌드

DOM은 Document Object Model의 약자로 트리 구조를 이룬다. 아래와 같은 HTML을 받아왔다고 하자.

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta ... />
        <link ... />
    </head>
    <body>
        <p>Hello, <span>web performance</span> students</p>
        <div>
            <img />
        <div>
```

```
    </body>
</html>
```

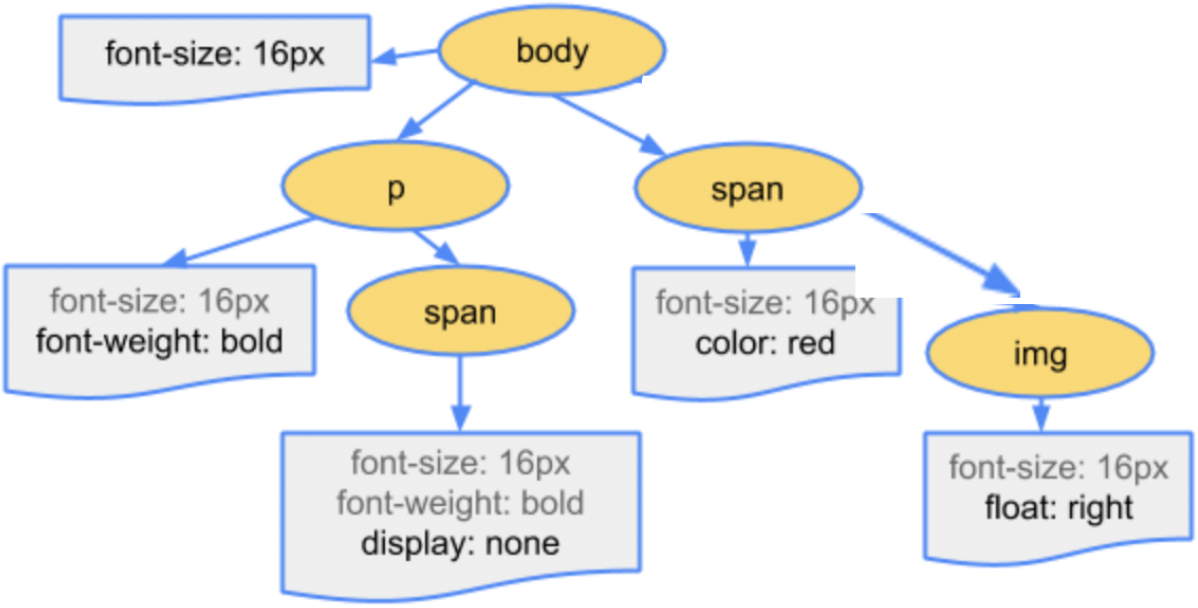
HTML 파일들은 바이트 형태로 전달되게 된다. 이를 트리 구조로까지 바꾸는데 몇몇 과정을 거치는데 그 과정은 바이트 → 문자 → 토큰 → 노드 → DOM 순서이다.



1. 바이트 → 문자
    - 바이트 형태의 파일을 지정된 인코딩에 따라 개별 문자로 변환한다. 그냥 바이트 형태로 받은 내용을 한 줄의 문자열로 바꿔준다고 생각한다.
2. 문자 → 토큰
    - 위 사진에서 알 수 있다시피 시작 태그, 종료 태그, 안의 내용들이 모두 토큰화 되어 저장된다.
3. 토큰 → 노드
    - 생성된 토큰들을 규칙 및 속성에 맞는 객체로 변환시킨다. 위의 예시에서는 태그의 경우 긴 원, 내용의 경우 찢어진 사각형으로 변환했다.
4. 노드 → DOM
    - 생성된 노드를 트리 형태로 구성해주면 끝

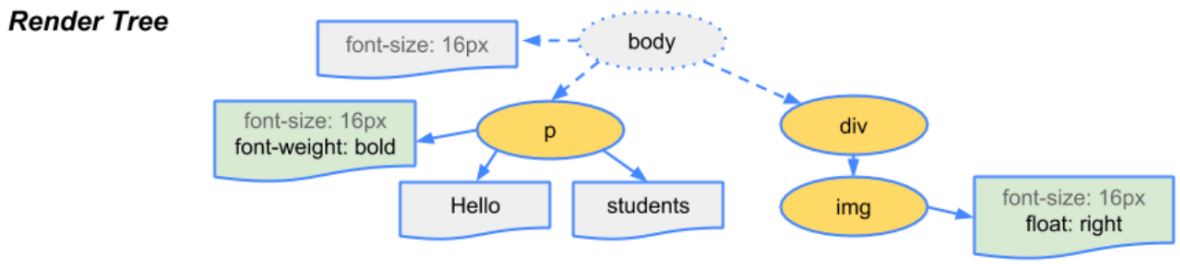
8. CSSOM 트리 빌드

CSS 또한 위의 HTML → DOM 과정과 같은 과정을 밟는다.



9. Render Tree 생성

기존에 제작된 DOM과 CSSOM을 결합해 Render Tree를 생성한다. Render Tree는 렌더링에 필요한 노드만 선택하여 페이지를 렌더링하는데 사용한다.



10. Layout

Render Tree의 노드들에 대한 위치와 크기를 계산하는 단계이다. 페이지 상에 존재하는 객체의 크기를 렌더링 트리의 루트부터 시작해 모든 객체의 정확한 위치와 크기를 계산한다.

11. Paint

계산된 값들을 기반으로 화면에 필요한 요소들을 실제로 그리는 작업을 실행한다. 레이아웃 단계에서 계산된 모든 위치, 크기를 실제 픽셀로 변환해 화면에 출력한다.

11. 5. Reflow & Repaint

특정 액션과 이벤트에 따라 html 요소의 크기나 위치를 변경해야 하는 경우가 발생하는데 해당 과정을 **reflow** 라고 한다. **reflow** 가 발생하면 렌더링 트리와 각 요소들의 크기 및 위치를 다시 계산해야 하며, 다시 페인팅을 해주는 **repaint** 단계 역시 수행된다.

하지만, reflow가 발생해야 repaint가 발생하는 것은 아니며, **css만 간단하게 바뀐 경우에는 repaint만 단독으로 수행되기도 한다.**

당연히 Reflow, Repaint는 최대한 줄여야 성능이 최적화 된다.

12. Composition

레이아웃과 페인트를 수행하지 않고 레이어의 합성만 실행시키는 단계이다. Transform, opacity와 같은 요소들을 의미한다.

13. 끝!!

브라우저 렌더링 과정