

## 02. 브라우저 렌더링

➤ 프로젝트



000. 인터넷

### 👁️ 학습 목표

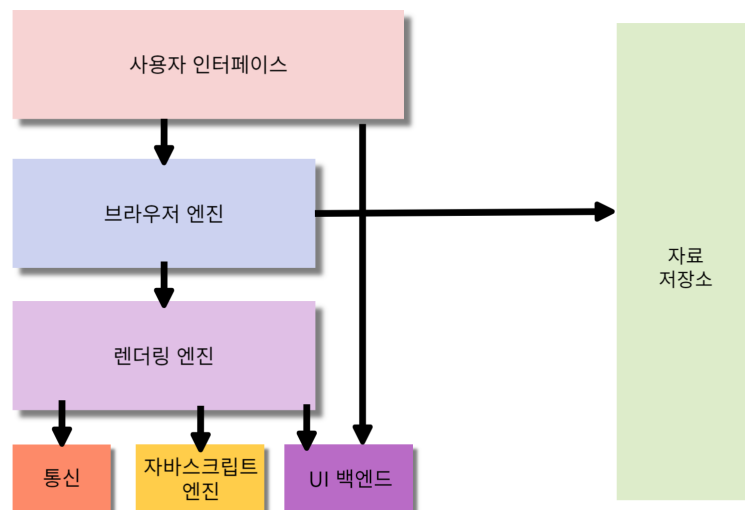
- 웹 브라우저의 구조를 이해한다.

### ☁️ 학습내용

#### ▼ 웹 브라우저의 구조

WEB 개발을 함에 있어 사용자에게 화면을 만들기 위해 HTML, CSS, JAVASCRIPT 코드를 이용하여 개발을 하게 되면 브라우저는 코드를 기반으로 웹 페이지를 그려주는 작업을 하게된다.

브라우저에서 코드를 이용하여 화면을 그리는 과정은 다음과 같은 순서로 진행이 되게 된다.



### 1. 사용자 인터페이스

사용자가 접근할 수 있는 영역을 의미하며, 주소 표시줄 및 새로고침, 다음, 이전, 즐겨찾기 등 웹페이지를 제외한 사용자와 상호작용을 지원하는 부분을 의미한다.

### 2. 브라우저 엔진

유저 인터페이스와 렌더링 엔진을 연결하는 역할을 담당하며 자료 저장 공간(Data Storage)를 참조하여 사용자 브라우저에 데이터를 읽고 쓰기를 하며 작업을 하게된다. 자료저장소는 Html5 부터 추가된 기능이다.

### 3. 렌더링 엔진

서버에서 전달받은 html, css를 해석하여 사용자에게 화면을 표시하는 기능을 담당하며 해당 렌더링 엔진의 동작 원리를 이해해야 이후 SPA 방식의 프레임 워크의 이점을 이해할 수 있게 된다.

또한 브라우저는 서버에서 html, css, javascript 문서를 전달받게 되면 html 파서와 css 파싱에 의해 DOM, CSSOM 트리로 변환되어 렌더 트리를 결합하는 과정을 거치게 되며 형성 이후 자바스크립트 엔진에 의해 동적인 이벤트를 형성하여 웹 페이지를 표시한다.

### 4. Networking

서버와 통신이 가능하도록 하는 통신은 인터넷에서 리소스를 가져오며 사용자 인터페이스의 주소창에 검색어를 입력하게 되면 데이터를 페칭하게 된다.

우리가 주소창에 타이핑하면, 브라우저는 북마크나 검색 기록에서 의미있는 제안들을 보여줍니다.

- URL을 파싱하고(도메인, params 등) http 또는 https 프로토콜을 반환합니다.
- NON-ASCII 코드가 변환되어야 한다면, URL을 변경하기 위해 인코딩합니다. (이 때문에 한국어나 특수문자가 변형됩니다.)
- 브라우저는 \*HSTS (HTTP Strict Transport Security) 목록을 체크하고, URL이 이 목록에 있으면 https 프로토콜을 추가합니다. 아니면 http로 보냅니다.
- HSTS는 보안을 위해 https로만 통신해야한다고 알려주는 기능입니다.
- DNS 룩업을 시작합니다. \*브라우저에 DNS 캐시가 있으면 해당 캐시에서 IP 주소를 찾고, 그래도 없으면 운영체제 cache, 그래도 없으면 로컬 라우터 또는 ISP(Internet Service Provider)에 있는 DNS 서버에 요청합니다.
- DNS가 무엇이고 동작 방식이 궁금하다면 이 포스팅을 참고해주세요. [another-light.tistory.com/35](https://another-light.tistory.com/35) [another-light.tistory.com/36](https://another-light.tistory.com/36)
- DNS 서버 또는 ARP(Address Resolution Protocol)를 사용한 default gateway로부터 IP 주소를 받습니다.
- 53번 포트를 열고 UDP(User Datagram Protocol) 프로토콜을 통해 DNS 서버와 통신합니다(응답 사이즈에 따라 TCP 또는 UDP 프로토콜 사용). 만약 default gateway라면, 재귀적으로 이 절차를 진행합니다.

- 만약 DNS가 이 주소에 대해 모르면, 브라우저는 검색 엔진에서 이에 대해 찾아보고 상위 10개의 결과를 보여줍니다.
- 반대로 DNS가 이 주소에 대해 알게 되면 서버의 IP 주소로 http(80 포트)/https(443 포트) 요청을 시작합니다. 그리고 TCP 소켓 연결을 요청합니다.
- TCP 헤더를 채우기 위해 네트워크 계층, 다음은 IP 헤더를 채우기 위해 전송 계층, 이더넷 프레임 헤더를 위해 데이터 링크 계층까지 요청이 도달합니다.
- 네트워크의 데이터 패킷이 디지털에서 전기 신호로 변환됩니다.
- 서버와 클라이언트 간의 3 way 핸드셰이크가 일어나고, 데이터를 클라이언트에게 보냅니다.
- 전송 계층에서 마지막으로 핸드셰이크가 일어나면 브라우저는 패킷에서 데이터를 얻게 됩니다.

## 05. 자바스크립트 인터프리

자바스크립트 코드를 실행하는 인터프리터로 크롬 브라우저의 경우 V8을 이용하고 있다.

## 06. UI 백엔드

체크박스과 버튼과 같은 위젯을 형성하는 부분

## 07. 데이터 저장소

로컬 스토리지와 세션 스토리지로 구분되어 사용자의 브라우저에 데이터를 저장하는 기술

### ▼ 브라우저 렌더링

현재 대표적으로 많이 사용되는 브라우저는 총 3가지 종류가 있으며 각각 다른 렌더링 엔진을 기반으로 웹 표준을 지키면서 조금씩 다르게 동작하게 된다.

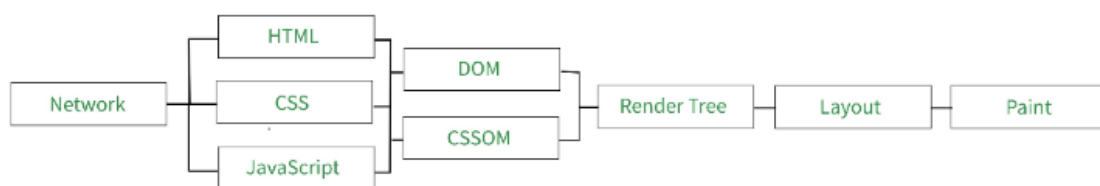
1. Safari : 사파리는 apple 사의 브라우저로 Webkit 렌더링 엔진을 이용한다.
2. Firefox는 Mozilla사의 브라우저로 Gecko 렌더링 엔진을 이용한다.
3. Chrome은 Google 사의 브라우저로 Blink 렌더링 엔진을 이용한다.

위의 브라우저 중 우리는 Chrome을 기준으로 브라우저 렌더링 과정에 대하여 알아볼 것이다.

렌더링 엔진은 서버로부터 전달받은 html, CSS, JS 이미지 등의 리소스를 사용자의 화면에 보여주는 것을 목표로 동작하게 되며 업데이트가 필요시 효율적으로 렌더링을 할 수 있도록 자료구조를 생성하게 된다.

렌더링은 단순히 코드를 읽고 사용자의 화면에 그려주는 방식이 아닌 자료구조를 생성한다는 말처럼 우선 서버로부터 전달받은 html, CSS, JS 코드를 기반으로 Critical Rendering Path라는 작업을 하게 된다. 해당 작업은 다음과 같은 순서로 이루어진다.

## Critical Rendering Path



<https://www.geeksforgeeks.org/critical-rendering-path-flow/> 이미지 참조

1. 네트워크 : 사용자가 웹페이지를 방문하게 되면 웹페이지 파일을 호스팅하는 서버의 IP주소를 확인하기 위해 DNS 조회가 수행되며 IP를 얻은 이후 해당 파일을 서버로 요청하게 되는데 이때 서버에서 파일을 전달하는 것은 한 번에 전달하는 것이 아닌 여러 개의 패킷으로 분할되어 네트워크를 통해 전송하게 된다.
2. DOM(Document Object Model) : Dom은 모든 웹 페이지의 중요한 부분으로 Html은 웹 페이지가 로드될 때 서버에서 전송되는 첫 번째 파일로 브라우저가 서버에서 전달받은 패킷을 받기 시작하면 전달받은 패킷의 구문을 분석하여 증분 DOM 업데이트 방식을 따르게 된다. 구문 분석에서 먼저 바이트를 문자로 변환하여 다음 토큰으로 변환한 뒤 노드로 변환하여 DOM 트리를 형성하게 되며 해당 주기는 새 패킷이 수신될 때마다 반복이 되는 형식이다.

♥ 토큰화 : W3C 표준에 지정된 고유한 토큰으로 변환하는 것으로 html 코드에서 <를 만나면 태그를 열림 상태로 전환 한뒤 a~Z문자를 만나면 시작 태그를 생성하고 상태는 태그 이름 상태로 변하는데 > 를 만날 때까지 유지를 한데 이러한 방식으로 새로운 문자 탐색 > 상태변경 > 처리를 반복하여 토큰화를 수행하게 되는 것이다.

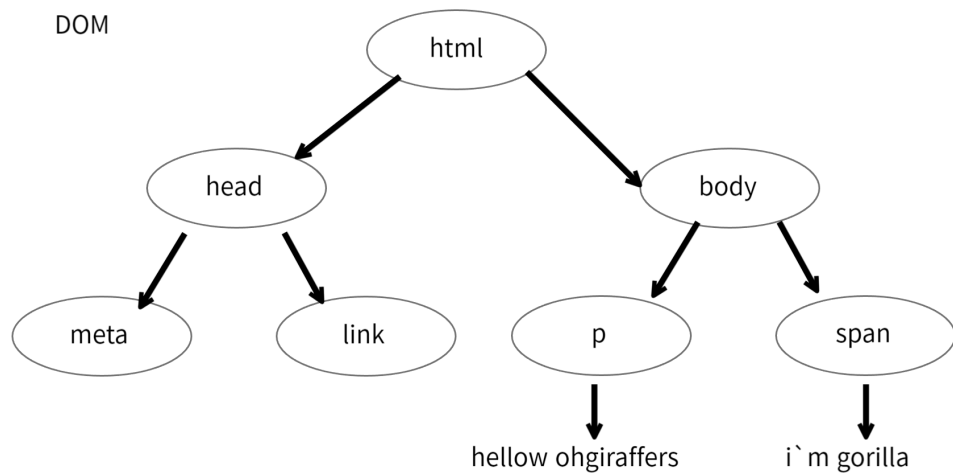


Bytes → Characters → Tokens → Node → Dom

```

<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style.css">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    <title>Document</title>
</head>
<body>
    <p>hellow ohgiraffers</p>
    <span>i`m gorilla</span>
</body>
</html>

```



DOM(Document Object Model) : Dom은 모든 웹 페이지의 중요한 부분으로 Html은 웹 페이지가 로드될 때 서버에서 전송되는 첫 번째 파일로 브라우저가 서버에서 전달받은 패킷을 받기 시작하면 전달받은 패킷의 구문을 분석하여 증분 DOM 업데이트 방식을 따르게 된다.

구문 분석에서 먼저 바이트를 문자로 변환하여 다음 토큰으로 변환한 뒤 노드로 변환하여 DOM 트리를 형성하게 되며 해당 주기는 새 패킷이 수신될 때마다 반복이 되는 형식이다.

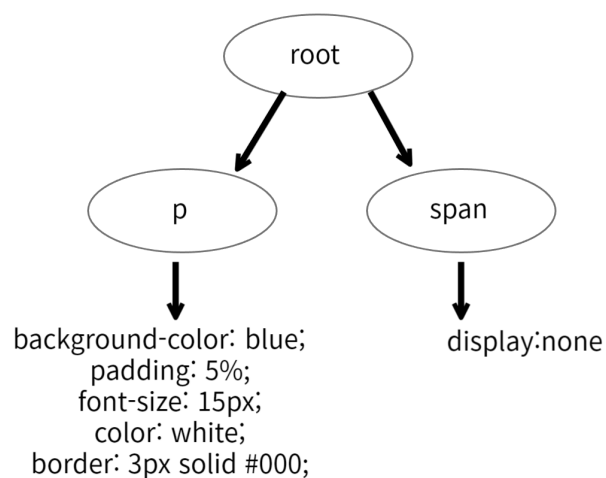
3. CSSOM(Cascading Style Sheet Object Model) : 객체 모델은 웹 페이지의 스타일 부분에 중점을 두게 되는 부분으로 CSSOM 트리의 경우 DOM 트리 구성과 달리 렌더링을 차단하는 방식으로 브라우저에서 네트워크를 통해 모든 패킷이 수신되기를 기다리고 있다가 모든 패킷이 수신되면 구문 분석을 시작하여 스타일 속성을 교체하거나 덮어쓰는 방식이다.

Bytes → Characters → Tokens → Node → CSSOM

```
p{
  background-color: blue;
  padding: 5%;
  font-size: 15px;
  color: white;
  border: 3px solid #000;
}

span{
  display:none
}
```

CSSOM



4. 렌더 트리 : 렌더 트리는 DOM 및 CSSOM 트리를 하나로 결합하는 역할을 담당한다.

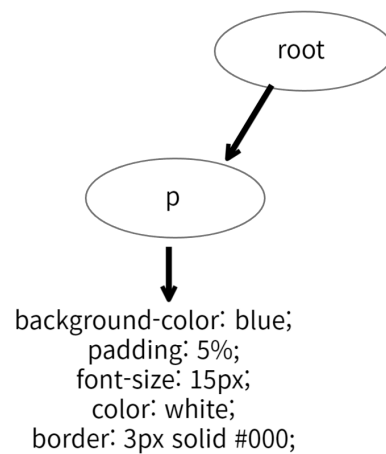
DOM 트리에서 각 노드를 가져와 CSS 개체를 여기에 매핑을 하게된다.

렌더 트리는 각 노드에 특정 노드에 대해 정의된 스타일 콘텐츠가 있는 경우 스타일 속성이 할당되어 있으며 렌더 트리 구성에서 중요한 부분은 사용자 측에 보여지는 부분만 트리에 추가한다는 것이다. 헤드의 경우 가시성이 없기 때문에 렌더트리에서 관리가 되지 않는다.

또한 노드의 스타일 속성이 `display:none` 속성을 가지고 있는 경우에도 추가되지 않는다.

만약 위의 Span 태그의 스타일 속성으로 `display:none`을 추가하게 되면 다음과 같이 렌더 트리를 형성할 것이다.


Render



5. Layout(Visula Formatting model) : 렌더 트리를 구축한 후 이제 레이아웃을 그림으로 나타내며 레이아웃은 화면 크기에 따라 화면에 요소를 배치하는 데 중점을 두고 높이와 너비가 100%인 요소는 전체 화면과 같은 크기를 갖는다. 레이아웃은 DOM 트리의 노드에 따라 다르기 때문에 노드의 개수가 많을 수록 브라우저가 레이아웃을 만드는데 더 많은 시간을 소비하게 된다.

브라우저는 렌더링을 위해 Visual Formatting Model을 사용하게 된다.

6. 페인트 : 렌더링 트리 및 레이아웃을 만든 후 브라우저가 화면의 픽셀을 페인트하는 마지막 단계로 페인팅이 완료된 후 웹 페이지를 보여주게 된다.  
해당 페이팅 과정은 렌더트리를 기반으로 속성을 갖지 않는 레이블은 표시하지 않는다.



hellow ohgiraffers

페인트 후 화면

## 자바스크립트 해석

위의 과정은 브라우저에서 사용자의 화면을 그리는 것을 설명하였다.  
그러나 위의 내용에서는 자바스크립트가 동작하는 부분에 대하여 설명하지 않고 있다.  
그렇다면 자바스크립트는 어느 시점에 어떻게 동작이 되는 것인가?

우선 자바스크립트는 자바스크립트 엔진이 별도로 처리하게 되는데 HTML 파서는 `<script>` 태그를 만나게 되면 자바스크립트의 실행을 위해서 DOM의 생성 프로세스를 중단하고 자바스크립트 엔진으로 권한을 넘겨주게 되는데 이때 제어권을 받는 엔진은 (브라우저 마다 다름) 자바스크립트 파일을 로드 후 파싱하여 실행을 하게 된다.

### | <Script> 태그 위치의 중요성

지나가는 말로 자바스크립트 태그는 `<body>`의 하단에 작성하라는 말을 들은 적이 있을 것이다.

이 부분 브라우저가 스크립트를 동작 시키는 구문과 관련이 있는데 스크립트의 파일이 많거나 파일이 커서 읽어오는 시간이 오래 걸리는 경우 사용자의 화면서 표시되는 것이 지연되거나 멈추는 경우가 발생된다.

이외로 엘리먼트를 참조하지 못하는 오류도 발생이 되는데 이는 html 코드가 형성되는 시점보다 자바스크립트의 코드가 생성되는 시점이 빠르기 때문에 스크립트가 동작을 하면서 해당 node를 찾지 못하는 에러가 발생이 되는 것이다.

```
<!DOCTYPE html>
<html lang="en">
```



```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>Document</title>
  <link rel="stylesheet" href="style.css"/>
</head>
<body>
  <h1>시작</h1>
  <script type="text/javascript">
    var $testNode = document.getElementById("node");
    console.log($testNode); // null
  </script>
  <h1 id="node"> 중간 </h1>
  <p>hellow ohgiraffers</p>
  <span>i`m gorilla</span>
</body>
</html>

```

## 스크립트 속성으로 로딩 순서 제어하기

스크립트 태그 속성에 로딩의 흐름을 제어하는 방법을 제시하고 있다.

async :

**스크립트의 로드와 html parsing이 함께** 이루어지다가 script의 로드가 끝나면 script가 실행되는 시점에 html 파싱을 중단하고 스크립트를 실행하여 종료 후 다시 제어권을 브라우저 렌더링 엔진에게 반환함

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>Document</title>
  <link rel="stylesheet" href="style.css"/>
</head>

```

```

<body>
  <h1>시작</h1>
  <script async src="index.js"></script> // <h1 id="node

  <h1 id="node"> 중간 </h1>
  <p>hellow ohgiraffers</p>
  <span>i`m gorilla</span>
</body>
</html>

```

defer : 스크립트 태그를 만나도 html 파싱이 중단되지 않고 파싱이 끝난 후에 script를 실행하는 방식으로 동작이 된다.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>Document</title>
  <link rel="stylesheet" href="style.css"/>
</head>
<body>
  <h1>시작</h1>

  <script defer src="index.js"></script> // <h1 id="node

  <h1 id="node"> 중간 </h1>
  <p>hellow ohgiraffers</p>
  <span>i`m gorilla</span>
</body>
</html>

```

## 스크립트 내부에서 흐름 제어하기

위와 같은 오류를 해결하고자 하는 경우 자바스크립트 내부에서 코드의 흐름을 제어하는 방법이 있으며 다음과 같이 이벤트의 실행 흐름을 제어할 수 있으나 꼭 필요한 경우가 아니면 권장되지 않는다고 하며 이유는 정확하게 모르겠다.

DOMContentLoaded 이벤트는 html 문서가 완전히 구문 분석되고 다른 다운로드 파일 빌드가 완료된 이후에 실행되는 옵션으로 이미지, 비동기 스크립트와 같은 항목 및 스타일시트의 로드는 기다리지 않는다는 특징을 갖는다.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>Document</title>
  <link rel="stylesheet" href="style.css"/>
</head>
<body>
  <h1>시작</h1>

  <script type="text/javascript">
    addEventListener("DOMContentLoaded", (event) =>{
      var $testNode = document.getElementById("node"
      console.log($testNode); // <h1 id="node"> 중간
    })
  </script>

  <h1 id="node"> 중간 </h1>
  <p>hellow ohgiraffers</p>

  <span>i`m gorilla</span>
</body>
</html>
```

#### ▼ 리플로우와 리페인트

reflow, repaint을 이해하기 위해 우리는 브라우저의 렌더링 과정에 대하여 공부를 하게 되었다.

Layout에서 진행되는 작업의 내용을 화면에 그리는 과정을 다시 반복하는 것을 reflow라 하며 repaint는 paint 작업을 다시 시작하는 것을 의미한다.

## 리플로우

리플로우는 레이아웃에서 진행되었던 요소의 너비, 높이, 위치 등 레이아웃을 잡기 위해서 계산하였던 전반적인 내용을 다시 작업하는 것을 말한다.

리플로우는 다음과 같은 경우 발생이 되는데 DOM의 속성이 변경되거나 브라우저의 사이즈가 변경될 때 수행하게 된다. 해당 작업은 layout을 새롭게 연산하여 그리는 작업으로 많은 비용을 발생 시키는 작업이며 주변까지 영향을 주기 때문이다.

리플로우의 경우 다음과 자바스크립트에서 속성을 접근할 때도 발생이 되며 요소의 현재 최신값을 확인하기 위해서 발생하는 것이다.

100

렌더트리 생성 → 레이아웃 → 리페인트 → 레이어 업데이트 → 합성

100

리플로우 발생 원인

1. 윈도우 사이즈 조절
2. 폰트의 변화 (해당 노드의 height에 영향을 주게 되어 Global Layout 변화)
3. 스타일 추가 또는 제거
4. 내용 변경 (input box 텍스트 변경 및 스크립트를 활용한 텍스트 노드 변경)
5. hover 이벤트
6. 클래스 속성의 동적 변화
7. 자바스크립트를 통해 dom 조작
8. 엘리먼트의 크기, 넓이, 위치 계산
9. 스타일 속성 변화

## 리페인트

변경된 요소를 화면에 그려주는 작업을 리페인트라 부른다.

리페인트는 리플로우가 발생하거나 요소의 스타일(색상, 배경색과 같은 속성)이 변경되거나 visibility: hidden;과 같은 속성에서 발생이 된다.

visibility: hidden;의 경우 사용자의 화면에 그려주지 않는 것이 요소가 display: none과 같이 완전히 렌더링 대상에서 제외하는 것이 아닌 속성으로 해당 속성은 리플로우가 발생되지 않는다.

100

랜더트리 생성 → 리페인트 → 레이어 업데이트 → 합성

## 리플로우 최적화 하기

우리가 만든 사이트의 성능을 최적화 하기 위해서 리플로우를 발생 시키지 않아야 한다는 것을 알아보게 되었다. 그렇다면 어떻게 하면 리플로우를 줄일 수 있는지 확인할 것이다.

### 1. DOM 속성 변경 코드 묶어서 실행

아래의 코드는 div1, div2, div3의 id를 가진 노드의 높이를 변경하는 코드이다. 높이가 변경될 때 노드는 총 3번의 리플로우가 발생되게 되게 된다.

```
<script type="text/javascript">
    var div1 = document.getElementById("div1").clientHeight;
    document.getElementById("div1").style.height = div1 + "px";

    var div2 = document.getElementById("div2").clientHeight;
    document.getElementById("div2").style.height = div2 + "px";

    var div3 = document.getElementById("div3").clientHeight;
    document.getElementById("div3").style.height = div3 + "px";
</script>
```

위의 코드를 브라우저의 특성을 이용하여 개선을 하면 리플로우를 1번으로 감소하는 것이 가능하다.

```
<script type="text/javascript">
    var div1 = document.getElementById("div1").clientHeight;
    var div3 = document.getElementById("div3").clientHeight;
    var div2 = document.getElementById("div2").clientHeight;
```

```

        document.getElementById("div2").style.height =
            document.getElementById("div1").style.h
        document.getElementById("div3").style.height =
    </script>

```

브라우저는 기본적으로 리플로우를 줄이기 위해서 요소의 변경을 바로 실행하는 것이 아닌 큐에 일정시간 저장을 하였다가 처리하는 방식으로 리플로우를 수행하게 되는데 clientHeight 속성 자체가 현재의 상태를 확인하기 위해서 리플로우를 발생시키는 속성이다.

이로 인해 처음 작성한 코드는 노드의 높이에 변경이 발생됨에 따라 전체 레이아웃에 영향을 줄 수 있다 인지하여 매번 리플로우를 발생 시키게 되는 것이다.

## 2. 리플로우가 유발 될 수 있는 메서드는 별도 저장 후 사용

실행시 현재의 상태를 확인하기 위해 리플로우를 발생 시키는 메서드를 사용하는 경우 변수에 저장을 하여 캐시를 이용하여 작업을 할 수 있도록 처리를 하는 것이다.

## 3. CSS 속성은 한번에 처리하기

스크립트로 CSS 속성을 하나씩 수정하는 것은 리플로우, 리페인팅을 여러번 유발하는 것으로 클래스를 별도로 정의하여 한번에 수정을 해주는 방식으로 한다.

```

<script type="text/javascript">
    let test = `width: 30%; height: 200px; backgrou
    document.getElementById("div1").style.cssText =
</script>

```

## 4. Render 엔진 확용하기

Render에서 display:none의 경우 노드를 그리지 않는다는 것을 확인하였다. 이러한 특성을 이용하여 다음과 같은 작업을 하는 전략을 이용한다.

```

<script type="text/javascript">
    let div1 = document.getElementById("div1");
    div1.style.display = "none";

```

```
div1.style.width = "30%";  
div1.style.height = "200px";  
div1.style.backgroundColor = "red";  
div1.style.display = "block";  
</script>
```

#### 5. 애니메이션은 position과 absolute fixed를 사용하기

애니메이션의 효과는 다른 노드의 영향을 주는 것이 대부분이며 이를 통해 리플로우가 발생되는데 다른 노드에 영향이 되지 않도록 해당 노드만 분리하여 동작되도록 한다면 리플로우를 제한하는 것이 가능하다.

#### 6. 테이블 레이아웃 자제

테이블은 기본적으로 전체 노드의 관계를 맺고 있기 때문에 리플로우가 쉽게 발생된다.

#### 7. 인라인 스타일 사용자제

인라인 스타일의 경우 html이 파싱되는 과정에서 레이아웃에 영향을 주게 되며 이를 통해 리플로우가 발생하는 문제가 있다.

#### 8. CSS 하위 선택자 최소화

CSS 하위 선택자를 최소화 하는 것은 reflow의 횟수를 줄이기 보다는 렌더트리의 계산을 최소화 하는 방법이다.

### Repaint 발생 속성

background	color
background-image	line-style
background-position	outline
background-repeat	outline-color
background-size	outline-style
border-radius	outline-width
border-style	text-decoration
box-shadow	...

## Reflow 발생 속성

```
position
width
height
left
top
right
bottom
margin
padding
border
border-width
display
float
font-family
font-size
font-weight
line-height
min-height
overflow
text-align
vertical-align
...
```

객체	속성 및 메서드
HTMLElement	clientHeight, clientLeft, clientTop, clientWidth, focus(), getBoundingClientRect(), getClientRects(), innerText, offsetHeight, offsetLeft, offsetParent, offsetTop, offsetWidth, outerText, scrollByLines(), scrollByPages(), scrollHeight, scrollIntoView(), scrollIntoViewIfNeeded(), scrollLeft, scrollTop, scrollWidth
Frame, Image	height, width
Range	getBoundingClientRect(), getClientRects()
SVGLocatable	computeCTM(), getBBox()
SVGTextContent	getCharNumAtPosition(), getComputedTextLength(), getEndPositionOfChar(), getExtentOfChar(), getNumberOfChars(), getRotationOfChar(), getStartPositionOfChar(), getSubStringLength(), selectSubString()
SVGUse	instanceRoot
window	getComputedStyle(), scrollBy(), scrollTo(), scrollX, scrollY, webkitConvertPointFromNodeToPage(), webkitConvertPointFromPageToNode()

### ▼ Virtual Dom

Facebook에서 개발한 REACT의 경우 가상의 돔을 이용하여 리플로우와 리페인팅이 발생하는 것을 최소화하였다고 한다. 여기서 우리가 알아야 하는 것은 간단한 소개페이지와 같은 작은 규모의 페이지에서는 일반 DOM이 더 좋은 성능을 나타내지만, 대규모 애플리케이션의 경우 REACT와 같은 SPA 프레임워크가 높은 성능을 보여주고 있다.



SPA 프레임워크가 우수한 성능을 보이는 이유는 현재 시대가 빠르게 변화하기 때문으로 볼 수 있는데 초기 웹의 경우 정적인 콘텐츠를 보여주는 WEB 1.0에서 서비스 로직이 추가되어 서버와 클라이언트 상호작용을 만든 WEB 2.0으로 발전되었으며 현재의 WEB 3.0은 Client 쪽에 많은 서비스가 추가 되면서 더 이상 과거의 방식으로 서비스를 이용하는 것이 힘들어졌기 때문이다.

현재의 시대적 흐름을 맞추기 위해서 Virtual Dom이 고안되었으며 해당 기술은 DOM의 노드 하나를 조작할 때마다 리플로우나 리 페인트 작업이 수행되면서 발생하였던 브라우저의 성능 저하 문제를 해결하기 위해 실제의 DOM과 같은 형태로 Virtual DOM을 형성하여 변경되는 내용이 존재하면 화면에 그려주는 과정을 거치지 않고 메모리상에서만 그리는 방식으로 오버헤드를 작게 만든 기술이다.

React의 경우 위의 virtual Dom의 기술을 채택한 대표적인 라이브러리로 React에서는 문서의 특정 부분이 리 렌더링 되는 것을 최소화하여 Virtual Dom 트리를 메모리에서 새로 생성하고 이전에 존재하는 Virtual Dom 트리과 같은 휴리스틱 알고리즘으로 비교하여 차이를 분석한다.

이후 차이점을 모아서 실제 DOM에 전달하면 이를 기반으로 리 렌더링 과정을 거치게 되며 이는 단 한 번만 일어나기 때문에 성능상 큰 이슈를 가질 수 있다.

자바스크립트에서는 Dom Fragment를 활용하여 리액트와 같이 한 번에 변경할 수 있는 기능을 만드는 것이 가능하지만 쉽지 않다....