# **브라우저 렌더링 원리**

# **브라우저(browser) ?**

특정 웹사이트를 접근한다고 하자.

1. 사용자가 브라우저를 실행한다.

2. 접속하고 싶은 사이트의 url를 주소창에 입력한다.

3. 브라우저는 입력된 url의 서버에게 사이트 정보를 받아와 이를 화면에 표현한다.

4. 표현된 정보를 사용자가 활용한다.

  
사용자가 보고자 하는 페이지를 서버에 요청하고 서버로 부터 받은 응답(HTML,CSS,JavaScript, 이미지 등..) 을 브라우저에 표시하는 것이다.

주로 알고 있는 웹 브라우저로는 파이어폭스, 구글크롬, 인터넷 익스플로러, 사파리 등이 있다.

웹 브라우저라고도 하며 웹 서버에 이동하며 쌍방향으로 통신하고 HTML문서나 파일을 출력하는 그래픽 사용자 인터페이스 기반의 응용 소프트 웨어이다. [- 웹 브라우저 정의 위키백과](https://ko.wikipedia.org/wiki/%EC%9B%B9_%EB%B8%8C%EB%9D%BC%EC%9A%B0%EC%A0%80)

## **브라우저의 기본 구조**

**1. 사용자 인터페이스**주소 표시줄, 이전/다음 버튼, 북마크 메뉴 등 요쳥한 페이지를 보여주는 창을 제외한 나머 지 모든 부분.

사용자 인터 페이스 예시

* URI(Uniform Resource Identifier)를 입력할 수 있는 주소 표시줄
* 이전 버튼과 다음 버튼
* 북마크
* 새로 고침 버튼과 현재 문서의 로드를 중단할 수 있는 정지 버튼
* 홈 버튼

**2. 브라우저 엔진**사용자 인터페이스와 렌더링 엔진 사이의 동작을 제어

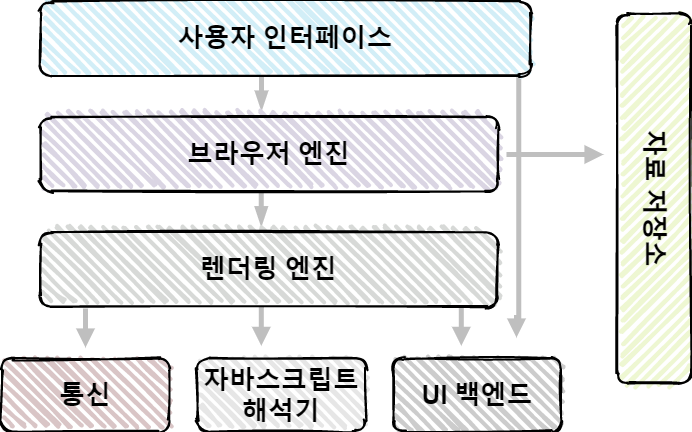
**3. 렌더링 엔진**요청한 콘테츠를 표시. 예를 들면 HTML을 요청하면 HTML과 CSS를 파싱 하여 화면에 표시한다.

**4. 통신**HTTP요청과 같은 네트워크 호출에 사용됨. 이것은 플랫폼 독립적인 인터페이스이고 각 플랫폼 하부에서 실행된다.

**5. UI 백엔드**콤보 박스와 창 같은 기본적인 장치를 그린다. 플랫폼에서 명시하지 않은 일반적인 인터페이스로, OS사용자 인터페이스 체계를 사용.

**6. 자바스크립트 해석기**자바스크립트 코드를 해석하고 실행.

**7. 자료 저장소**자료를 저장. 쿠키를 저장하는 것과 같이 모든 종류의 자원을 하드 디스크에 저장할 필요가 있다. HTML5명세에는 브라우저가 지원하는 웹데이터 베이스가 정의되어 있다.



# **렌더링 엔진**

렌더링 엔진의 역할은 요청 받은 내용을 브라우저 화면에 표시하는 일이다.

렌더링 엔진은 HTML및 XML 문서와 이미지를 표시 할수 있다. 물론 플러그인이나 브라우저 확장 기능을 이용해 PDF와 같은 다른 유형도 표시할 수 있다.

각 브라우저 마다 사용하는 렌더링 엔진은 다른데 다음과 같다.

| **브라우저** | **엔진** |
| --- | --- |
| 파이어 폭스 | 게코(Gecko) 엔진 |
| 사파리, 크롬 | 웹킷(Webkit) |

## **렌더링 과정**

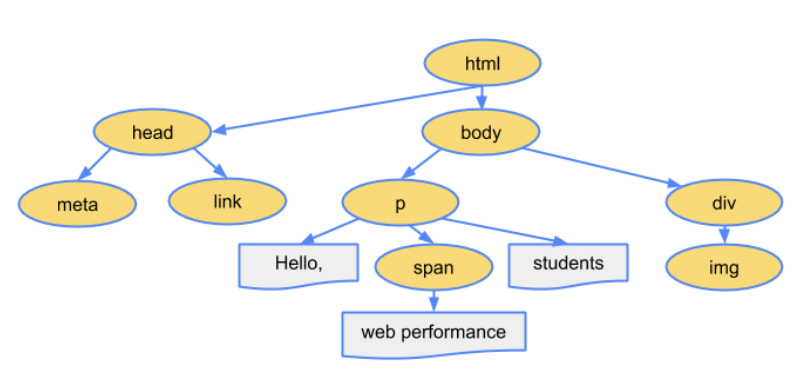
#### **1.불러오기(Loading)**

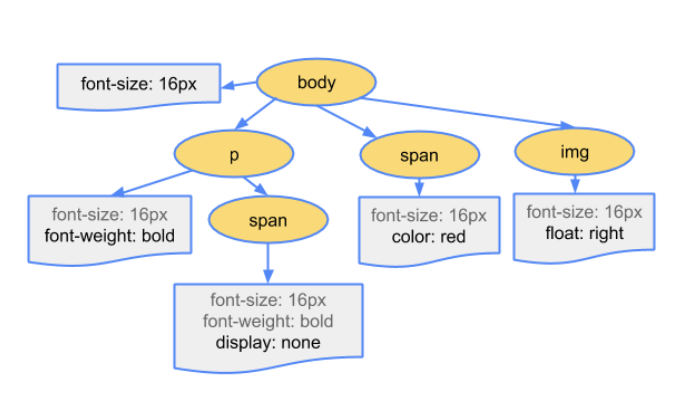
HTTP 모듈 또는 파일 시스템으로 전달 받은 리소스 스트림을 읽는 과정으로 로더가 이 역할을 맡고 있다. 로더는 단순히 읽는 것이 아니라, 이미 데이터를 읽었는지도 확인하고, 팝업창을 열지 말지, 또는 파일 다운로드 받을지를 결정한다.

#### **2.다운받은 HTML, CSS를 Object Model로 만든다.**

**DOM**(Document Object Model) 이란?

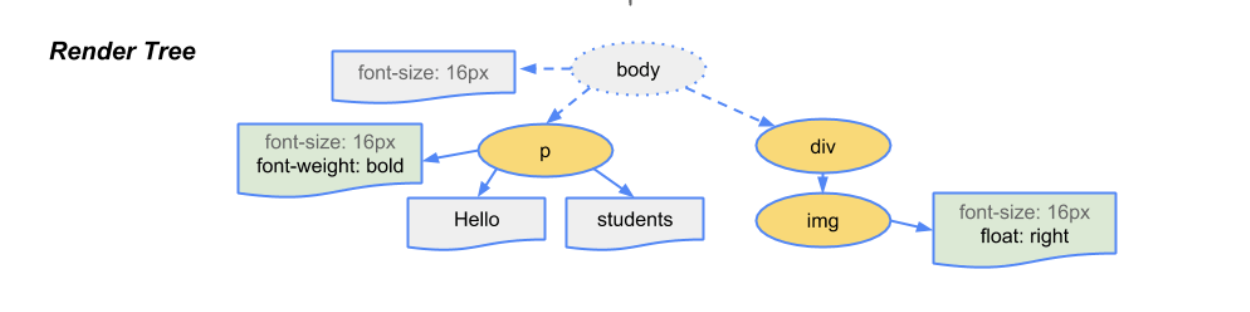
HTML태그를 JS에서 이용할수 있는 객체로 만드는 것이다. 즉 HTML문서의 객체 기반 표현 방식이다.

◾ HTML -> DOM  
HTML파일은 HTML파서에 의해 파싱 되어 DOM트리로 변환된다.  


CSS -> CSSOM (<link> ,<style> 를 통하여 생성)  
CSS파일은 CSS파서에 의해 파싱 되어 CSSOM트리로 변환된다.  


**파싱(parse or parsing)?**문서를 파싱한다는 것은 브라우저가 코드를 이해하고 사용할 수 있는 구조로 변환하는 것이다.  
파싱 결과는 보통 문서 구조를 나타내는 노드 트리인데 파싱 트리 또는 문법 트리 라고 부른다.

#### **3. DOM과 CSSOM을 합친다.**

◾Render Tree  


#### **4. Layout**

렌더트리가 만들어 졌으면 이것을 토대로 그려질 노드와 그의 스타일값 그리고 치수까지 계산한다.

Viewport

그래픽이 표시되는 브라우저 영역, 크기  
뷰포트는 모바일의 경우 디스플레이의 크기, pc의 경우 브라우저 창의 크기에 따라 달라진다.

#### **5. Paint**

이 정보들을 페인팅 단계로 전달해서 렌더트리의 각 노드를 화면산의 실제 픽셀로 변환한다.

## **HTML파서가 <script> 태그를 만나면?**

자바스크립트는 자바스크립트 엔진이 처리하는데 HTML파서는 <script>태그 를 만나면 자바스크립트 코드를 실행하기 위해 DOM생성 프로세스를 중지하고 자바스크립트 엔진으로 제어 권한을 넘긴다.

제어권을 넘겨받은 **자바스크립트 엔진**은 **<script> 태그 내의 자바스크립트 코드 혹은 script의 src attribute에 정의된 js파일을 로드하고 파싱/기계어로 컴파일하여 실행**한다.

자바스크립트 실행이 왼료되면 다시 HTML파서로 권한을 넘겨서 아까 중지했던 시점 부터 다시 DOM생성을 재개 한다.

브라우저는 동기적으로 HTML, CSS, Javascript를 처리하는데, 이는 <script>태그의 위치에 따라 DOM생성이 지연될 수 있다는 뜻이다.

보통 <body>요소의 가장 아래에 <script>태그를 위치하는게 좋은 방법이다. 그이유의  
첫째로는 HTML 요소들의 렌더링이 지연되는 일이 없기 때문에 화면 로딩이 줄고 사용자에게 더 좋은 경험을 줄 수 있기 때문이고, 둘째로는 DOM이 완성되지 않은 상태에서 자바스크립트가 DOM을 조작 한다면 에러가 발생하기 때문이다.

### **스크립트 로드 시점 - async, defer**

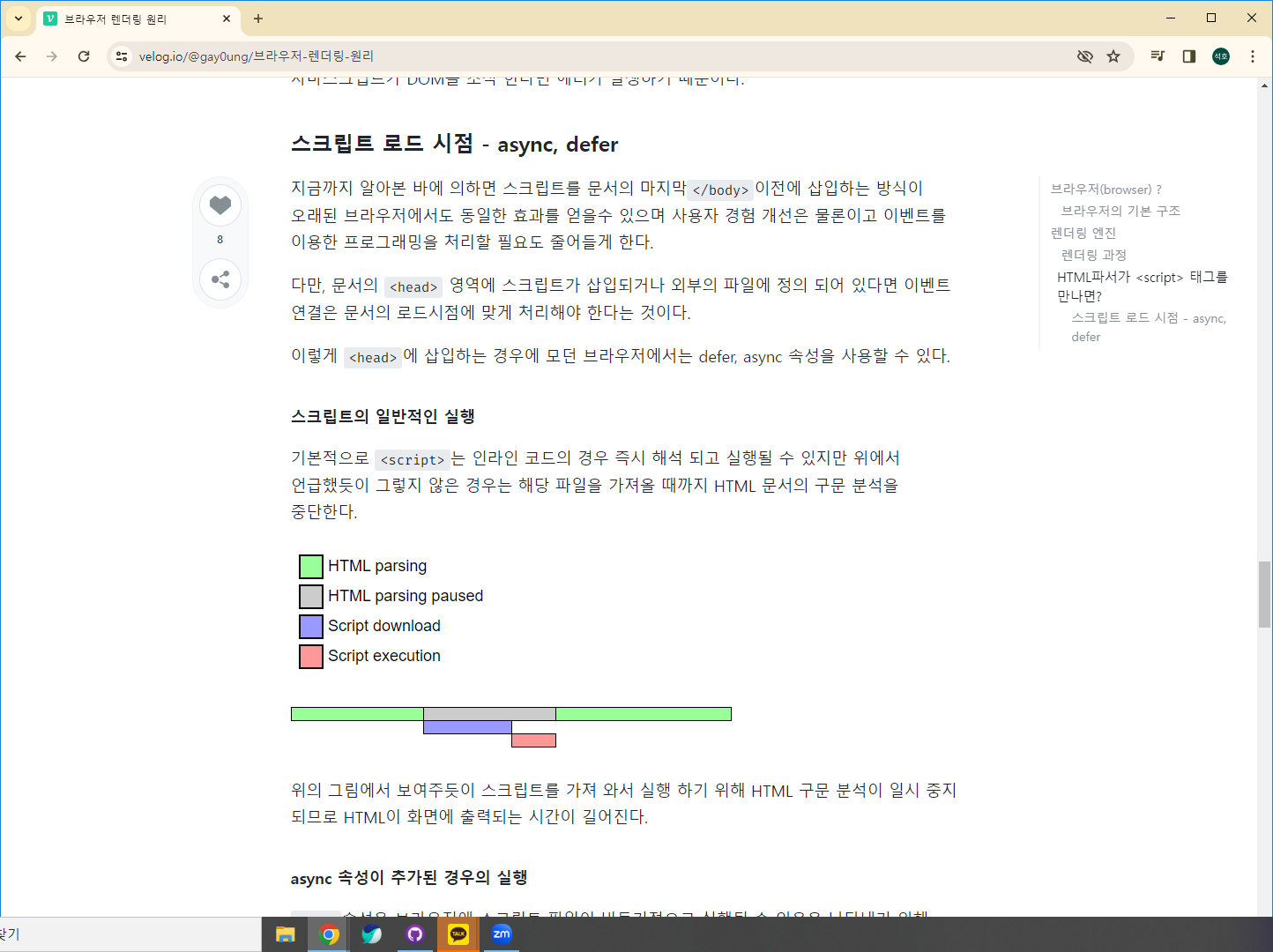
지금까지 알아본 바에 의하면 스크립트를 문서의 마지막</body>이전에 삽입하는 방식이 오래된 브라우저에서도 동일한 효과를 얻을수 있으며 사용자 경험 개선은 물론이고 이벤트를 이용한 프로그래밍을 처리할 필요도 줄어들게 한다.

다만, 문서의 <head> 영역에 스크립트가 삽입되거나 외부의 파일에 정의 되어 있다면 이벤트 연결은 문서의 로드시점에 맞게 처리해야 한다는 것이다.

이렇게 <head>에 삽입하는 경우에 모던 브라우저에서는 defer, async 속성을 사용할 수 있다.

#### **스크립트의 일반적인 실행**

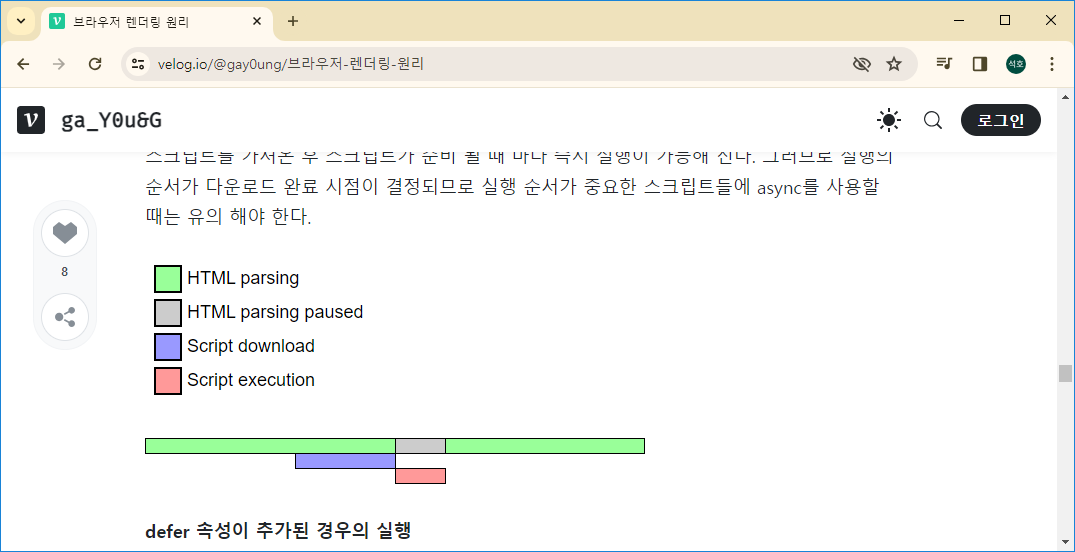
기본적으로 <script>는 인라인 코드의 경우 즉시 해석 되고 실행될 수 있지만 위에서 언급했듯이 그렇지 않은 경우는 해당 파일을 가져올 때까지 HTML 문서의 구문 분석을 중단한다.



위의 그림에서 보여주듯이 스크립트를 가져 와서 실행 하기 위해 HTML 구문 분석이 일시 중지 되므로 HTML이 화면에 출력되는 시간이 길어진다.

#### **async 속성이 추가된 경우의 실행**

async속성은 브라우저에 스크립트 파일이 비동기적으로 실행될 수 있음을 나타내기 위해 사용할 수 있다. 즉, HTML 구문 해석기는 스크립트 태그에 도달한 지점에서 스크립트를 가져오고 실행하기 위해 일시 중지 할 필요가 없다. 따라서 HTML 구문 분석과 병행하여 스크립트를 가져온 후 스크립트가 준비 될 때 마다 즉시 실행이 가능해 진다. 그러므로 실행의 순서가 다운로드 완료 시점이 결정되므로 실행 순서가 중요한 스크립트들에 async를 사용할 때는 유의 해야 한다.



#### 

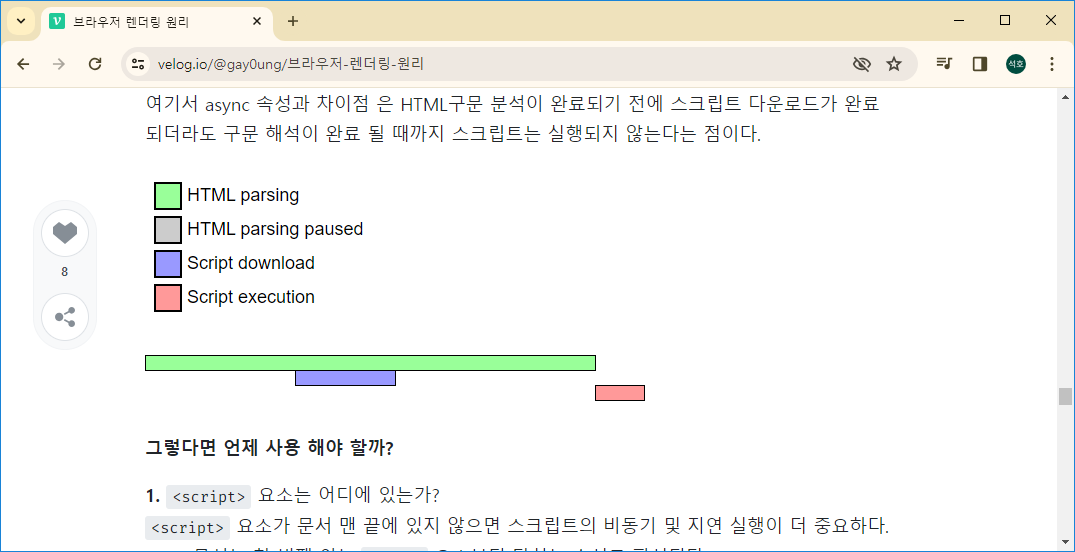
#### **defer 속성이 추가된 경우의 실행**

async(기본값 true)인 경우와 다르게 defer 속성은 HTML 구문 분석이 완전히 완료되면 스크립트 파일을 실행하도록 브라우저에 지시하게 된다.

<script defer src = "index.js"></script>

비동기적으로 로드된 스크립트와 마찬가지고, HTML 구문분석이 실행되는 동안 (HTML해석을 방해하지 않고) 스크립트 파일을 다운로드 할 수 있따.

여기서 async 속성과 차이점 은 HTML구문 분석이 완료되기 전에 스크립트 다운로드가 완료 되더라도 구문 해석이 완료 될 때까지 스크립트는 실행되지 않는다는 점이다.



#### **그렇다면 언제 사용 해야 할까?**

**1.** <script> 요소는 어디에 있는가?  
<script> 요소가 문서 맨 끝에 있지 않으면 스크립트의 비동기 및 지연 실행이 더 중요하다. HTML문서는 첫 번째 여는 <html> 요소부터 닫히는 순서로 파싱된다.  
외부 소스 JavaScript 파일이 닫는 </body> 요소 바로 앞에 있으면, async 또는 defer 속성을 사용하는 것이 큰 효과가 없을 것이다.(HTML 파서가 그 시점까지 문서의 대다수를 완성 했기 때문에 지연에 대한 의미가 크게 없음).

**2.** 스크립트 파일의 의존성 여부에 따라  
다른 파일들에 종속적이지 않거나 종속성 자체가 없는 스크립트 파일의 경우 async 속성이 특히 유용하다. 파일이 어느 지점에서 실행되는지 정확히 알 필요가 없기 때문에 비동기 로드가 가장 적합할 수 있다.  
그리고 의존성을 가지고 있는 스크립트 파일이거나 어떤 이유로 든 문제의 파일을 다른 위치에 배치해야 하는 상황에서는 defer를 사용하면된다.