Chap 38 브라우저의 렌더링 과정

▼ 파싱(=구분 분석)

프로그래밍 언어의 문법에 맞게 작성된 텍스트 문서를 읽어 들여 실행하기 위해 텍스트 문서의 문자열을 토큰으로 분해하고 토큰에 문법적 의미와 구조를 반영하여 트리 구조의 자료구조의 파스 트리를 생성하는 일련의 과정

▼ 렌더링

HTML, CSS, 자바스크립트로 작성된 문서를 파싱하여 브라우저에 시각적으로 출력하는

▼ 브라우저 렌더링 과정

1. 브라우저는 HTML, CSS, 자바스크립트, 이미지, 폰트 파일 등 렌더링에 필요한 리소스를 요청하고 서버로부터 응답을 받는다.
2. 브라우저의 렌더링 엔진은 서버로부터 응답된 HTML과 CSS를 파싱하여 DOM과 CSSOM을 생성하고 이들을 결합하여 렌더 트리를 생성한다.
3. 브라우저의 자바스크립트 엔진은 서버로부터 응답된 자바스크립트를 파싱하여 AST를 생성하고 바이트 코드로 변환하여 실행한다. 이때 자바스크립트는 DOM API를 통해 DOM이나 CSSOM을 변경할 수 있다. 변경된 DOM과 CSSOM은 다시 렌더 트리로 결합된다.
4. 렌더 트리를 기반으로 HTML요소의 레이아웃(위치와 크기)를 계산하고 브라우저 화면에 HTML요소를 페인팅한다.

▼ 요청과 응답

브라우저의 핵심 기능은 필요한 **리소스**(HTML, CSS, JS, 이미지, 폰트 등의 정적 파일 또는 서버가 동적으로 생성한 데이터)를 **서버에 요청**하고 **서버로부터 응답받아 브라우저에 시각적으로 렌더링**하는 것이다.

- 서버에 요청을 전송하기 위해서는 브라우저는 주소창을 제공한다.
- 요청과 응답은 개발자 도구의 Network 패널에서 확인할 수 있다.
- 브라우저의 렌더링 엔진이 HTML을 파싱하는 도중에 외부 리소스를 로드하는 태그, 즉 CSS파일을 로드하는 link태그, 이미지 로드하는 img태그, 드을 만나면 HTML파싱을 일시 중단하고 해당 리소스 파일을 요청한다.

▼ HTTP 1.1과 HTTP 2.0

HTTP : 웹에서 브라우저와 서버가 통신하기 위한 프로토콜(규약)

▼ HTTP 1.1

커넥션 당 하나의 요청과 응답만 처리. 즉, 여러개 요청 한번에 처리 불가능

응답시간 증가

▼ HTTP 2.2

커넥션당 여러 개의 요청과 응답 가능. 페이지 로드 속도 1.1보다 약 50%정보 빠름

▼ HTML 파싱과 DOM생성

브라우저의 렌더링 엔진은 응답받은 HTML문서를 파싱하여 브라우저가 이해할 수 있는 자료구조인 **DOM을 생성**한다.

▼ CSS 파싱과 CSSOM 생성

렌더링 엔진은 HTML문서를 파싱하며 DOM을 생성해나가는데 이때 CSS를 로드하는 link태그나 style태그를 만나면 **DOM 생성을 일시 중단**한다.

그리고 CSS파일을 서버에 요청하여 로드한 CSS파일이나 style태그 내의 CSS를 HTML과 동일한 파싱과정(바이트 → 문자 → 토큰 → 노드 → CSSOM)을 거치며 해석하여 **CSSOM을 생성**한다.

그리고 다시 DOM생성을 재개한다.

▼ 렌더 트리 생성

HTML과 CSS를 파싱하여 생성한 DOM과 CSSOM을 렌더링 하기 위해 렌더 트리로 결합된다.

▼ 렌더 트리

렌더링을 위한 트리 구조의 자료구조. 브라우저 화면에 렌더링되지 않는 노드(meta 태그, script태그)와 CSS에 의해 비표시(display:none) 되는 노드들은 포함하지 않는다.

이후 완성된 렌더트리는 HTML 요소의 레이아웃을 계산하는데 사용되며 브라우저 화면에 픽셀을 렌더링하는 페인팅 처리에 입력된다.

▼ 리렌더링

레이아웃 계산과 페인팅을 다시 실행하는 리렌더링은 비용이 많이드는 , 즉 성능에 악영향을 주는 작업이다. 따라서 가급적 리렌더링이 빈번하게 발생하지 않도록 주의하자.

▼ 리렌더링을 일으키는 경우

- 자바스크립트에 의한 노드 추가 또는 삭제
- 브라우저 창의 리사이징에 의한 뷰포트 크기 변경
- HTML 요소의 레이아웃에 변경을 발생시키는 width/height, margin, padding, border, display, position, top/right/bottom/left 등의 스타일 변경

▼ 자바스크립트 파싱과 실행

자바스크립트 파싱과 실행은 브라우저 렌더링 엔진이 아닌 자바스크립트 엔진이 처리한다.

자바스크립트 엔진은 자바스크립트 코드를 파싱하여 CPU가 이해할 수 있는 저수준 언어로 변환하고 실행하는 역할을 한다.

자바스크립트 엔진은 ECMAScript 사양을 준수한다.

CSS파싱 과정도 마찬가지로 렌더링 엔진은 DOM을 생성해 나가다 자바스크립트 파일을 도르하는 script파일을 로드하는 script태그를 만나면 DOM 생성을 일시 중단하고 **자바스크립트 엔진으로 제어권을 넘긴다.** 그리고 추후 자바스크립트 파싱과 실행이 종료되면 다시 DOM 생성을 재개한다.

▼ 리플로우와 리페인트

- 리플로우 : 레이아웃 계산을 다시하는 것. 노드 추가/삭제, 요소의 크기/위치 변경, 윈도우 리사이징 등 레이아웃에 영향을 주는 변경이 발생한 경우에 한하여 실행된다.
- 리페인트 : 재결합된 렌더 트리를 기반으로 다시 페인트를 하는 것을 말한다.
- 리플로우와 리페인트가 반드시 순차적으로 동시에 실행되는 것은 아니다. 레이아웃에 영향이 없는 변경은 리플로우 없이 리페인트만 실행된다.

▼ 자바스크립트 파싱에 의한 HTML 파싱 중단

- 자바스크립트 파싱/실행이 DOM이 다 생성되기 이전에 실행되면 DOM의 생성이 일시 중단된다. 따라서 자바스크립트 코드를 body요소의 가장 아래에 위치시키는 것이 좋다. 이렇게 하면 DOM이 완성되지 않은 상태에서 자바스크립트가 DOM을 조작하는 에러가 발생할 우려도 없고 페이지 로딩시간이 단축되는 이점도 챙길 수 있다.

▼ script 태그의 async/defer 어트리뷰트

자바스크립트 파싱에 의한 DOM 생성이 중단되는 문제를 해결하기 위해 HTML5부터 script태그에 async와 defer 어트리뷰트가 추가되었다.

```
<script async src="extern.js"><script />
<script defer src="extern.js"><script />
```

위와과 같이 src 속성이 있어야만 사용가능. 즉 src속성이 없는 인라인 자바스크립트에는 사용할 수 없다.

async와 defer 어트리뷰트를 사용하면 HTML파싱과 외부 자바슼립트 파일의 로드가 **비동기적으로 동시에 진행**된다. 그러나 자바스크립트 실행 시점에 차이가 있다.

▼ async와 defer의 자바스크립트 실행 시점 차이점

async : 자바스크립트 파싱과 실행은 자바스크립트 파일의 로드가 완료된 직후 진행되며, HTML 파싱이 중단된다. 따라서 순서가 보장되지 않는다.

defer : 자바스크립트 파싱과 실행은 Html파싱이 완료된 직후, 즉 DOM 생성이 완료된 직후 진행된다.