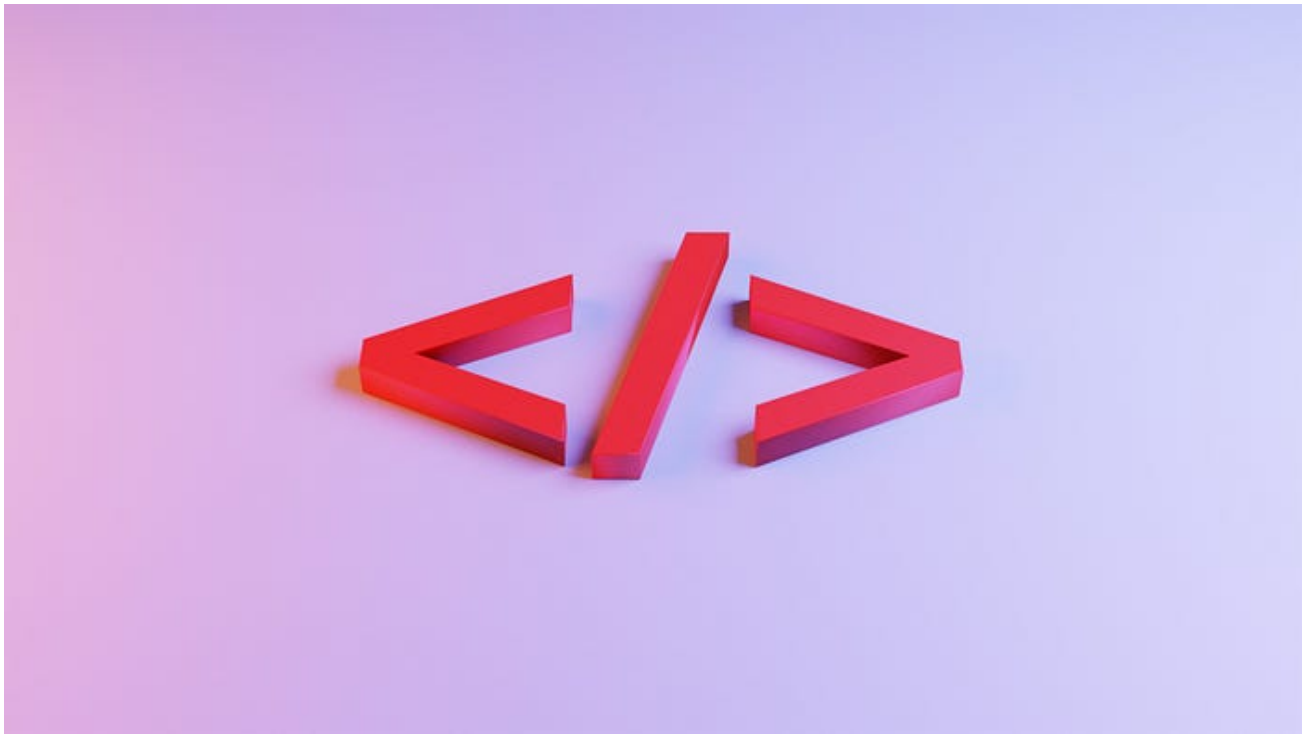
Home

2022-09-06

CSS 안에 자바스크립트 넣기

> 원문: Putting JavaScript In Your CSS

> 물론입니다. CSS 안에 JS를 사용하는 것은 가능합니다.



> Photo by Jackson Sophat on Unsplash

개발자 입장에서 Javascript 안에 CSS를 사용해본 적이 있을 겁니다. 하지만 오늘은 CSS 내부에 Javascript를 작성하는 것을 보여드리고자 합니다.

그 방법으로 후디니(Houdini)라고 알려진 CSS low-level API가 있으며, 개발자는 브라우저 렌더링 엔진의 styling과 layout 프로세스를 후킹하여 CSS를 확장가능하도록 합니다.

후디니는 `HTMLElement.style` 같은 자바스크립트를 통한 스타일 변경보다 파싱을 더 빠르게 할 수 있습니다. 브라우저는 스크립트에서 어떠한 스타일 변경을 적용하기 전에 CSSOM을 파싱하고 레이아웃, 페인트, 컴포지트(compsite)하는 프로세스를 거칩니다. 후디니 코드는 첫번째 렌더링 사이클이 끝나기를 기다리지 않습니다. 오히려 첫번째 사이클에 포함되어 렌더링 가능하고 이해하기 쉬운 스타일을 만듭니다. 후디니는 자바스크립트 내에 CSS 값으로 작업하기 위한 객체 기반의 API를 제공합니다.

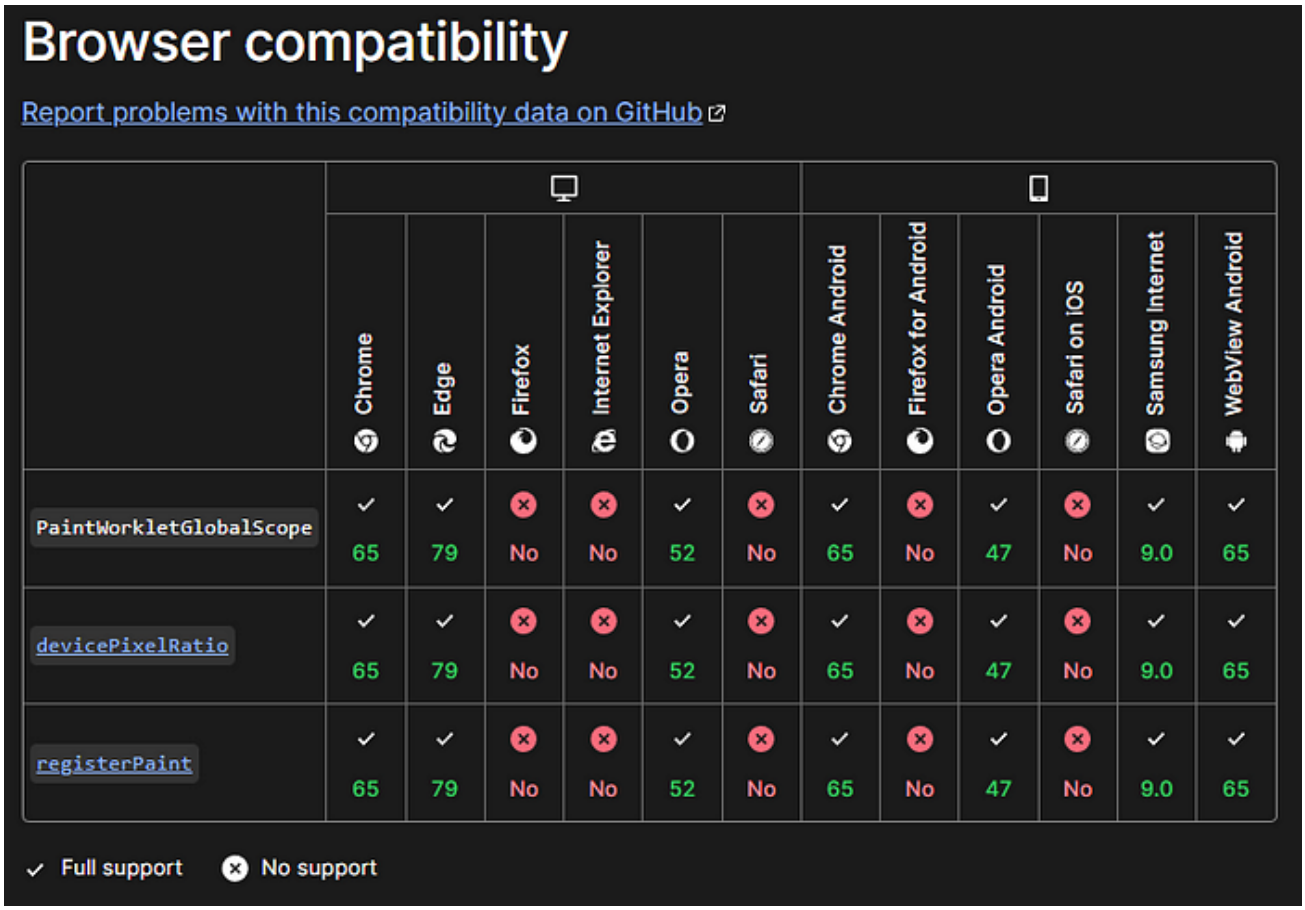
후디니의 `CSS Typeed OM API` 는 타입과 메서드가 포함된 CSS 객체 모델이며, 값을 자바스크립트 객체로 노출하여 이전 문자열 기반 `HTMLElement.style` 조작보다 직관적인 CSS 조작을 가능하게 합니다. 모든 요소 및 스타일 시트 규칙에는 `StylePropertyMap` 을 통해 접근 할 수 있는 스타일 맵이 있습니다.

CSS 후디니의 기능 중 하나는 `Worklet` 입니다. worklet을 사용하면 CSS 모듈을 생성할 수 있는데요. 이 모듈 방식은 전처리 프로세서, 후처리 프로세서 또는 자바스크립트 프레임워크 없이 한 줄의 자바스크립트로 가져올 수 있습니다.

```
CSS.paintWorklet.addModule("csscomponent.js");
```

이 모듈은 완전히 구성 가능한 worklet을 등록하는 `PaintWorklet.registerPaint` 함수가 포함되어 있습니다.

Paint Worklets 지원 여부



> CSS Paint API 호환성 통계

CSS `paint()` 함수는 `<image>` 타입에서 지원되는 추가 기능입니다. worklet의 이름과 함께 worklet에 추가적으로 필요한 매개변수를 인자로 받습니다. worklet은 사용자 지정 속성에 대한 권한도 가지고 있습니다. 따라서 이 속성들은 함수 인수로 전달할 필요가 없습니다.

이 예시에서는 `paint()` 함수에 `myComponent` 라는 worklet이 전달됩니다.

```
li {
  background-image: paint(myComponent, stroke, 10px);
  --highlights: blue;
  --lowlights: green;
}
```

Houdini APIs

아래에서 Houdini의 범위에 있는 API를 찾을 수 있습니다.

CSS 프로퍼티 그리고 Values API

새로운 CSS 프로퍼티를 등록하기 위한 API를 정의합니다. 이 API를 사용하여 등록된 프로퍼티는 타입, 상속 초기값을 정의한 해석된 구문을 제공합니다.

CSS Typed OM

CSSOM 값 문자열을 의미 있는 자바스크립트 표현으로 변환하고 다시 변환하면 상당한 성능 오버헤드가 발생할 수 있습니다. CSS Typed OM은 CSS 값을 형식화된 자바스크립트 객체로 보여주고 성능 조작을 허용합니다.

CSS Painting API

CSS의 확장성을 향상시키기 위해 개발된 Painting API는 개발자가 `paint()` CSS 함수를 통해 요소의 배경, 테두리 또는 콘텐츠로 직접 그릴 수 있는 자바스크립트 함수를 작성할 수 있도록 합니다.

Worklet

기본 자바스크립트 실행 환경과 독립적인 렌더링 파이프라인의 다양한 단계에서 스크립트를 실행하기 위한 API입니다. 워크렛은 개념적으로 웹 워커와 유사하며 렌더링 엔진에 의해 호출되고 확장됩니다.

CSS Layout API

CSS의 확장성을 향상시키기 위해 설계된 이 API를 통해 개발자는 masonary 또는 line snapping과 같은 자체 레이아웃 알고리즘을 작성할 수 있습니다.

> (역주) masonary: 핀터레스트에서 대표적으로 사용하는 레이아웃으로 이미지를 효과적으로 배치하는 레이아웃입니다. 자세한 내용은 MDN을 참고하세요. (역주) line snapping: 자세한 내용은 w3c를 참고하세요.

CSS Parser API

이 API는 임의의 CSS 유사 언어를 약간 유형화된 표현으로 구문 분석하기 위해 CSS 파서를 더 직접적으로 노출합니다.

Font Metrics API

폰트 메트릭을 노출하는 API를 통해 타이포그래픽 레이아웃 결과에 접근할 수 있습니다.

> ("CSS Custom Paint" 또는 "Houdini의 paint worklet"로 알려지기도 한)CSS Paint API는 기본적으로 활성화되어 있습니다.

CSS Paint API를 사용하면 CSS 프로퍼티가 이미지를 예상할 때마다 프로그래밍 방식으로 이미지를 생성할 수 있습니다. `background-image` 또는 `border-image` 와 같은 속성은 일반적으로 이미지 파일을 로드하기 위해 `url()` 또는 `linear-gradient()` 와 같은 CSS 내장 함수와 함께 사용됩니다. 이러한 기능을 사용하는 대신 `paint(myPainter)` 를 사용하여 *paint worklet*을 참조할 수 있습니다.

Paint worklet 작성하기

`myPainter` 라는 paint worklet을 정의하려면 `CSS.paintWorklet.addModule('my-paint-worklet.js')` 을 사용하여 CSS paint worklet 파일을 로드해야 합니다. 해당 파일에서 `registerPaint` 함수를 사용하여 paint worklet 클래스를 등록할 수 있습니다.

```
class MyPainter {
  paint(ctx, geometry, properties) {
    // ...
  }
}
registerPaint("myPainter", MyPainter);
```

`paint()` callback에서 `<canvas>` 로부터 알고 있는 `CanvasRenderingContext2D` 와 같은 방법으로 `ctx` 를 사용할 수 있습니다. 만약 `<canvas>` 안에서 어떻게 그리는지 알고 있다면, paint worklet 안에서도 그릴 수 있습니다! `geometry` 는 우리가 마음대로 쓸 수 있는 캔버스의 폭과 높이를 말해줍니다.

paint worklet의 문맥은 `<canvas>` 와 100% 동일하지는 않습니다. 현재로서는 텍스트 렌더링 방법이 없으며 보안상의 이유로 캔버스에서 픽셀을 다시 읽을 수 없습니다.

입문 예시로, 체커보드 paint worklet을 작성하여 `<textarea>` 의 배경 이미지를 만들어봅니다. (기본적으로 크기가 조정가능하기 때문에 textarea를 사용하였습니다)

```
<!-- index.html -->
<!DOCTYPE html>
<style>
  textarea {
    background-image: paint(checkerboard);
  }
```

```
</style>
<textarea></textarea>
<script>
  CSS.paintWorklet.addModule("checkerboard.js");
</script>



// checkerboard.js
class CheckerboardPainter {
  paint(ctx, geom, properties) {
    // 보통의 canvas에서 사용하듯이 `ctx`를 사용합니다
    const colors = ["red", "green", "blue"];
    const size = 32;
    for (let y = 0; y < geom.height / size; y++) {
      for (let x = 0; x < geom.width / size; x++) {
        const color = colors[(x + y) % colors.length];
        ctx.beginPath();
        ctx.fillStyle = color;
        ctx.rect(x * size, y * size, size, size);
        ctx.fill();
      }
    }
  }
}
// 만든 class를 등록합니다
registerPaint("checkerboard", CheckerboardPainter);
```

이전에 `<canvas>` 를 사용했다면 이 코드가 익숙해 보일 것입니다. 라이브 데모를 확인해보세요.

Paint worklet을 지원하지 않는 브라우저

작성할 당시에 Chrome만이 paint worklet을 구현했습니다. 다른 모든 브라우저 공급업체로부터 긍정적인 신호가 있지만, 큰 진전이 없습니다. 최신 정보를 얻으려면 Is Houdini Ready Yet?을 정기적으로 확인하십시오. 그동안 paint worklet을 지원하지 않더라도 점진적인 개선 기능을 사용하여 코드를 계속 실행해도 됩니다. 예상대로 작동하게 하려면 CSS와 JS의 두 위치에서 코드를 조정해야 합니다.

JS에서 paint worklet에 대한 지원 감지는 `CSS` 객체를 확인하여 수행할 수 있습니다.

```
if ("paintWorklet" in CSS) {
  CSS.paintWorklet.addModule("mystuff.js");
}
```

CSS 에서는 두가지 옵션이 있습니다. `@supports` 를 사용할 수 있습니다.

```
@supports (background: paint(id)) {
  /* ... */
```

```
}
```

더 간결한 트릭은 CSS가 알 수 없는 기능이 있을 경우 전체 프로퍼티 선언을 무효화하고 무시한다는 사실을 이용하는 것입니다. 먼저 paint worklet을 사용하지 않고 그 뒤에 두며 프로퍼티를 두 번 지정하면 점진적인 향상 효과를 볼 수 있습니다.

```
textarea {
  background-image: linear-gradient(0, red, blue);
  background-image: paint(myGradient, red, blue);
}
```

paint worklet을 지원하는 브라우저에서 `background-image` 의 두 번째 선언이 첫 번째 선언을 덮어씁니다. paint worklet을 *지원하지 않는* 브라우저에서는 두 번째 선언이 유효하지 않으며, 첫 번째 선언이 그대로 적용되어 버립니다.

CSS Paint 폴리필

다양한 용도로 CSS Paint Polyfill을 사용하여 최신 브라우저에 CSS Custom Paint 및 Paint Worklets 지원을 추가할 수도 있습니다.

이 글이 마음에 드신다면 아래에서 더 확인해 보세요 :)

- https://medium.com/@jatin.krr/your-ultimate-guide-to-kubernetes-78836ef73072
- https://medium.com/@jatin.krr/your-ultimate-guide-to-docker-9c6f66963e4b
- https://javascript.plainenglish.io/connect-bluetooth-devices-using-javascript-891d586750c8
- https://medium.com/@jatin.krr/top-45-newsletters-for-developers-4542007b149b
- https://medium.com/@jatin.krr/everything-you-need-to-know-about-markdown-8f9239524af3
- https://medium.com/@jatin.krr/ultimate-nodejs-packages-100-resources-2022-edition-list-72561e134a83

즐거운 코딩되세요!

Styling with Markdown is supported

Sign in with GitHub