

Worklet

- 자료사이트

<https://js-dev-note.netlify.com>

Worklet이란

Worklet 인터페이스는 Web Workers의 경량 버전으로 개발자가 렌더링 파이프 라인의 하위 수준에 접근할 수 있도록 해 줍니다. Worklet을 사용하면 JavaScript 및 WebAssembly 코드를 실행하여 고성능이 필요한 경우 그래픽 렌더링 또는 오디오 처리를 수행할 수 있습니다. - MDN

- 기본적으로 Worker는 한 Thread에 하나가 생성이 가능하지만, Worklet은 한 Thread에 여러 개 생성이 가능합니다.
- 메인 Thread에서 만들 수 있습니다.
- 독립적인 GlobalScope와 Event loop를 가집니다.

Worklet Type

- PaintWorklet
- LayoutWorklet
- AnimationWorklet
- AudioWorklet

그리고

- Typed OM

Houdini(후디니)

Houdini 프로젝트는 Mozilla, Apple, Opera, Microsoft, HP, Intel 그리고 Google의 엔지니어들로 구성되어있습니다.

공식 W3C 표준으로 채택되기 위한 *표준 초안*들을 작성하고 있습니다.

"Houdini"라 하면 표준 문서들의 내용을 의미합니다. 아직 [Houdini 표준안 초안](#)들은 미완성 단계이며, 일부 초안은 가안(임시로 만든 안건)입니다.

들여가기전에

1. `Chrome://flag`
2. `Experimental Web Platform features`
3. `Enable`

CSS Painting API(PaintWorklet)

CSS Painting API를 사용하게 되면 CSS 속성 중 이미지 타입에 사용할 수 있는 모양을 정의할 수 있습니다.

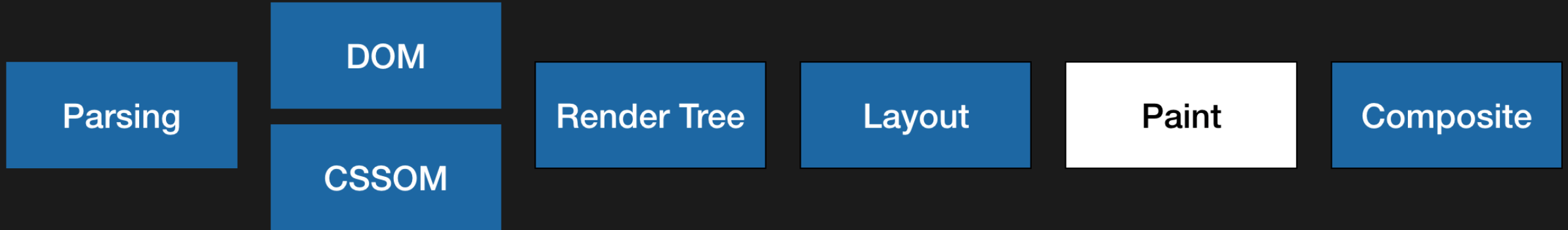
Image Type?

- `background-image`
- `border-image`
- `list-style-image`

이 중 `background-image` 속성을 사용하면 CSS가 적용된 대상이 그려지는 형태를 정의할 수 있습니다.

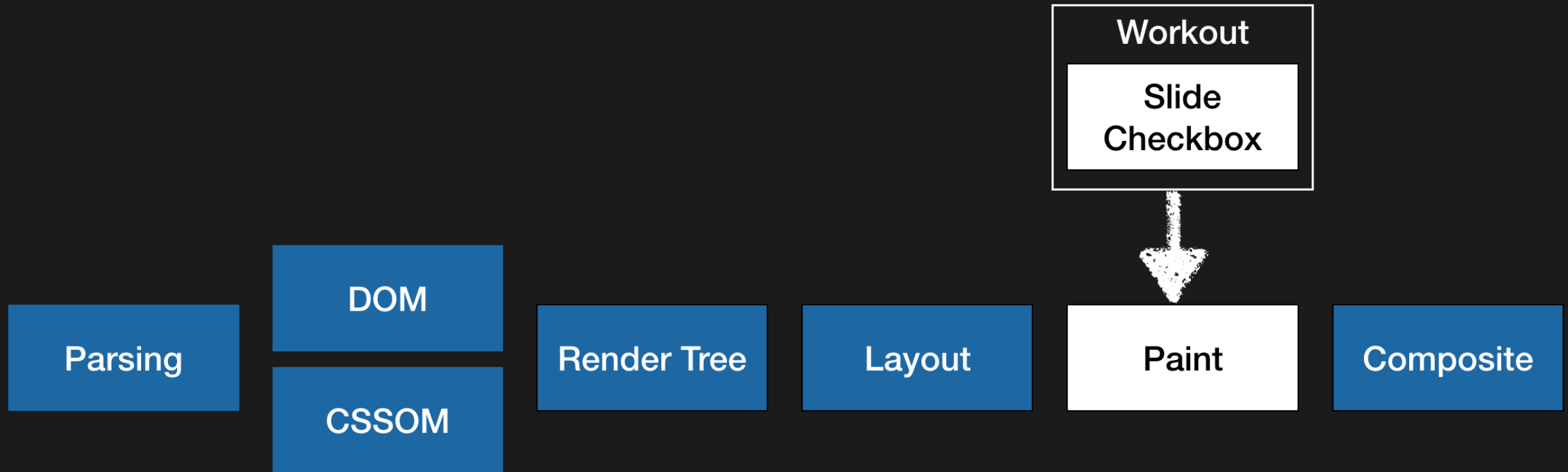
```
.slide-checkbox{  
  background-image: paint(slide);  
}
```

Paint는 대상을 그리는 방법을 다루는 단계입니다.



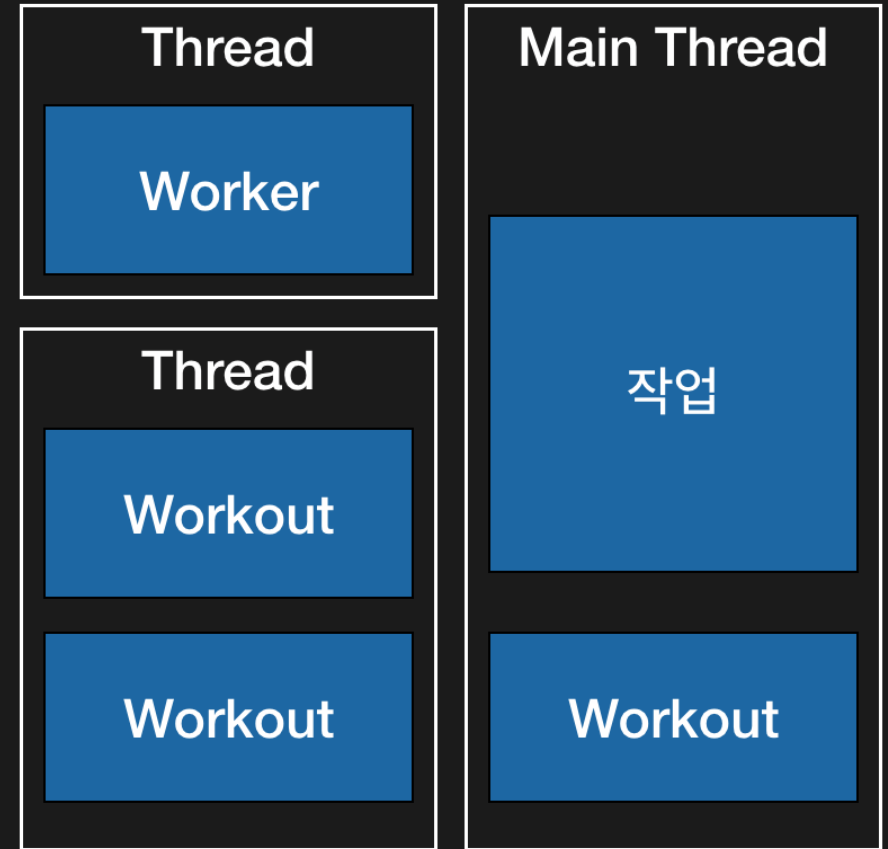
위의 렌더링 파이프라인은 **Chrome** 기준이며 브라우저에 따라 차이가 있을 수 있습니다.

CSS Painting API는 Worklet의 형태로 개발자가 정의한대로 대상을 그리는 코드를 추가합니다.



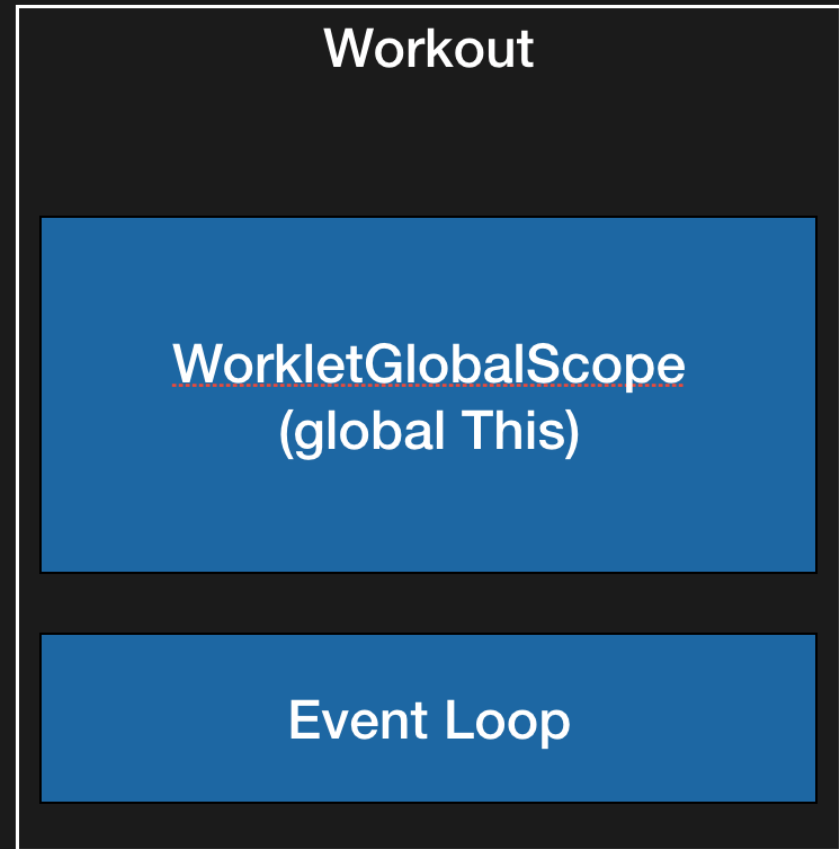
위에서 언급했듯이 Worklet은 Worker의 경량 버전이라고도 합니다.

하지만 Worker와는 다르게 한 스레드에 여러개가 생성될 수 있고, 메인 스레드에서 실행될 수 있습니다.



Worklet은 독립적인 GlobalScope와 Event Loop를 가지고 있습니다.

결국, 그어야 할 대상이 많거나, 성능이 필요한 경우에 여러 개의 Thread에서 병렬로 동작할 수 있게 합니다.



기본 형태

독립적인 파일을 만들어서 addModule을 하는 형태입니다.

```
CSS.paintWorklet.addModule("slideWorklet.js")
```

위의 파일을 class로 작성을 하며 registerPaint 를 사용하여 해당 paint를 등록합니다.

```
class Slide {  
    static get inputArguments() { return []; }  
    static get inputProperties() { return []; }  
    paint(ctx, geom, props, args){ }  
}  
  
registerPaint("slide", Slide);
```

paint method에는 기본 4가지의 인자가 있습니다.

- ctx : PaintRenderingContext2D 객체로, 대상이 어떻게 그려질지 표현합니다.
- geom : 대상의 가로, 세로 크기정보입니다.
- props : 대상에게 적용된 스타일 정보입니다.
- args : CSS에서 전달한 값을 입력받습니다.

props와 args는 각각 inputProperties() , inputArguments() 를 사용하여 입력받을 값(속성명)을 지정하여야 합니다.

예제

index.html

```
<head>
  <link rel="stylesheet" href="index.css">
  <script src="index.js"></script>
</head>

<body>
  <input type="checkbox" class="switch">

  <label class="switch">
    <input type="checkbox">
    <span class="slider round"></span>
  </label>
</body>
```

index.js

```
CSS.registerProperty({  
  name: '--slide-on',  
  syntax: '<number>',  
  inherits: true,  
  initialValue: "0"  
});  
CSS.paintWorklet.addModule("slideWorklet.js")
```

index.css

```
/* CSS Paint API */
.slide-checkbox{
  background-image: paint(slide);
  display: block;
  color: green;
  width: 60px;
  height: 34px;
  --slide-on : 0;
  -webkit-appearance: none;
  transition: --slide-on 200ms
}

.slide-checkbox:checked{
  --slide-on : 1;
  background-image: paint(slide);
}

/* Custom Checkbox for CSS */
.switch {
  position: relative;
  display: inline-block;
  width: 60px;
  height: 34px;
}

.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

.slider {
  position: absolute;
  cursor: pointer;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: #ccc;
  -webkit-transition: .2s;
  transition: .2s;
}

.slider:before {
  position: absolute;
  content: "";
  height: 28px;
  width: 28px;
  left: 3px;
  bottom: 3px;
  background-color: white;
  -webkit-transition: .2s;
  transition: .2s;
}

input:checked + .slider {
  background-color: #2196F3;
}

input:focus + .slider {
  box-shadow: 0 0 1px #2196F3;
}

input:checked + .slider:before {
  -webkit-transform: translateX(26px);
  -ms-transform: translateX(26px);
  transform: translateX(26px);
}

.slider.round {
  border-radius: 34px;
}

.slider.round:before {
  border-radius: 50%;
}
```

slideWorklet.js

```
const DEG_360 = Math.PI * 2;
const FG_COLOR = "white";
const BG_COLOR = "#E1E1E1";
const BG_COLOR_ON = "#FFCD00";
const CIRCLE_MARGIN = 3;

class Slide {
  static get inputProperties() {
    return ['--slide-on'];
  }

  paint(ctx, geom, props, args){
    const {width, height} = geom;
    const halfOfCircleSize = height / 2;
    const innerWidth = width - height;
    const on = parseFloat(props.get('--slide-on')).toString();
    const x = halfOfCircleSize + innerWidth * on

    ctx.fillStyle = on == 1 ? BG_COLOR_ON : BG_COLOR;
    ctx.beginPath();

    // 양쪽에 원을 그린다.
    ctx.arc(halfOfCircleSize, halfOfCircleSize, halfOfCircleSize, 0, DEG_360);
    ctx.arc(width - halfOfCircleSize, halfOfCircleSize, halfOfCircleSize, 0, DEG_360);

    // 가운데 사각형을 그려준다.
    ctx.rect(halfOfCircleSize, 0, innerWidth, height);
    ctx.fill();

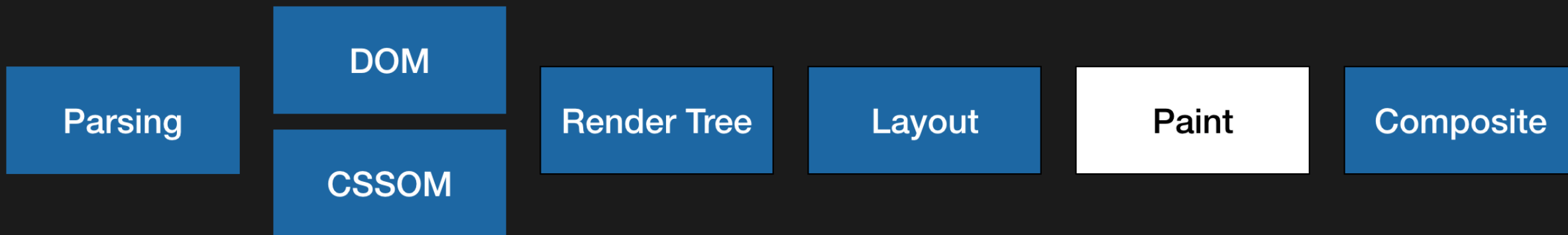
    // 슬라이드시 움직이는 원을 그린다.
    ctx.fillStyle = FG_COLOR;
    ctx.beginPath();
    ctx.arc(x, halfOfCircleSize, halfOfCircleSize - CIRCLE_MARGIN, 0, DEG_360);
    ctx.fill();
  }
}

registerPaint("slide", Slide);
```


성능비교

렌더링 파이프라인은 이전 단계의 결과물이 다음 단계의 입력으로 사용됩니다.

따라서 엘리먼트가 많아지게 되면 DOM 객체가 많아지고 이후 모든 과정들의 연산량과 메모리 사용량이 증가하게 됩니다.



위의 슬라이드 체크박스를 약 5,000개를 만들었을 경우, $5000 * 3$ 개의 엘리먼트가 5000개의 엘리먼트로 대체됩니다.

CSS Layout API(LayoutWorklet)

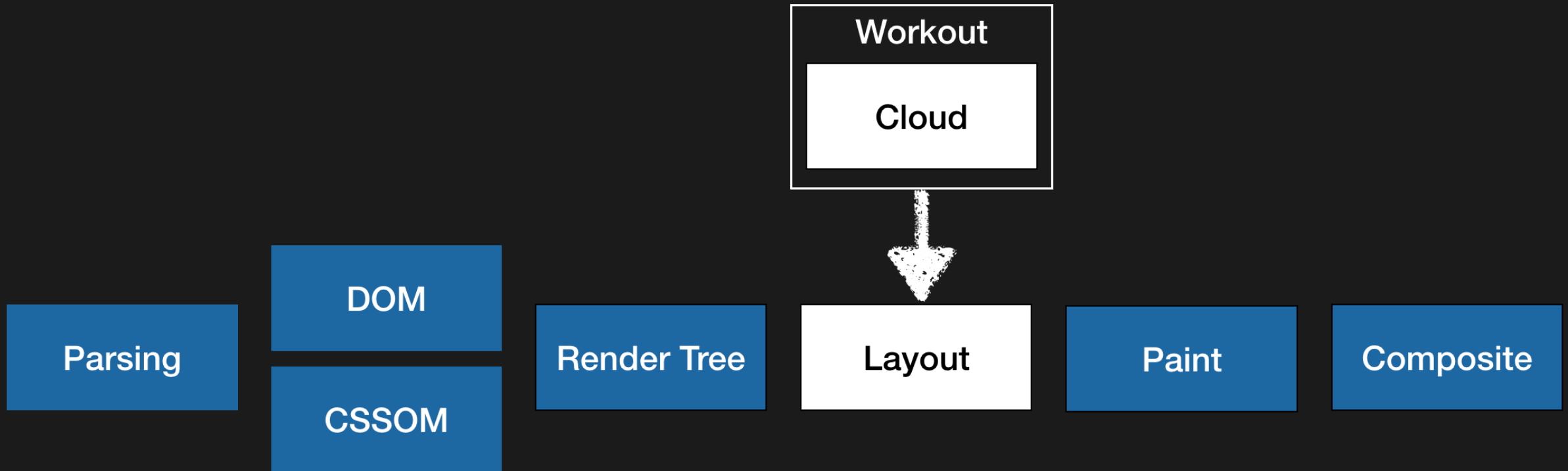
CSS Layout API를 사용하면 CSS가 적용된 대상의 자식 엘리먼트들의 배치를 정의할 수 있습니다.

```
.cloud{  
  display: layout(cloud);  
}
```

렌더링 파이프라인 중 Layout 단계는 대상을 배치하는 방법을 다룹니다.



CSS Layout API는 Worklet의 형태로 자식 엘리먼트를 개발자가 정의한대로 배치하는 코드를 추가합니다.



기존의 방법은, 렌더링이 끝난 후, JS로 재배치하면 렌더링 파이프라인을 한번 더 수행합니다.



기본 형태

```
class CloudLayout {  
    static get inputProperties() {return [];}  
    *intrinsicSizes(children, edges, styleMap) {}  
    *layout(children, edges, constraints, styleMap){}  
}  
registerLayout("cloud", CloudLayout);
```

paint method에는 기본 4가지의 인자가 있습니다.

- children : 자식 요소들의 정보입니다.
- edges : 레이아웃이 적용된 요소의 외곽선 정보입니다.
- constraints : 레이아웃이 적용된 요소의 크기 정보입니다.
- styleMap : 레이아웃이 적용된 요소의 Object Model style Map

paint API와 동일하게 대상 엘리먼트에 적용된 스타일 정보를 읽어올 수 있습니다.

마찬가지로 static inputProperties 필드로 읽어오려는 속성의 이름을 미리 선언해야합니다.

layout() 함수는 자식요소들의 비동기 처리를 위해서 제네레이터 함수로 작성해야 합니다.

```
const childFragments = yield children.map(child => {  
  return child.layoutNextFragment(constraints)  
})
```

각각 자식요소에서 layoutNextFragment() 함수를 호출해서 자식요소의 크기를 알 수 있습니다.

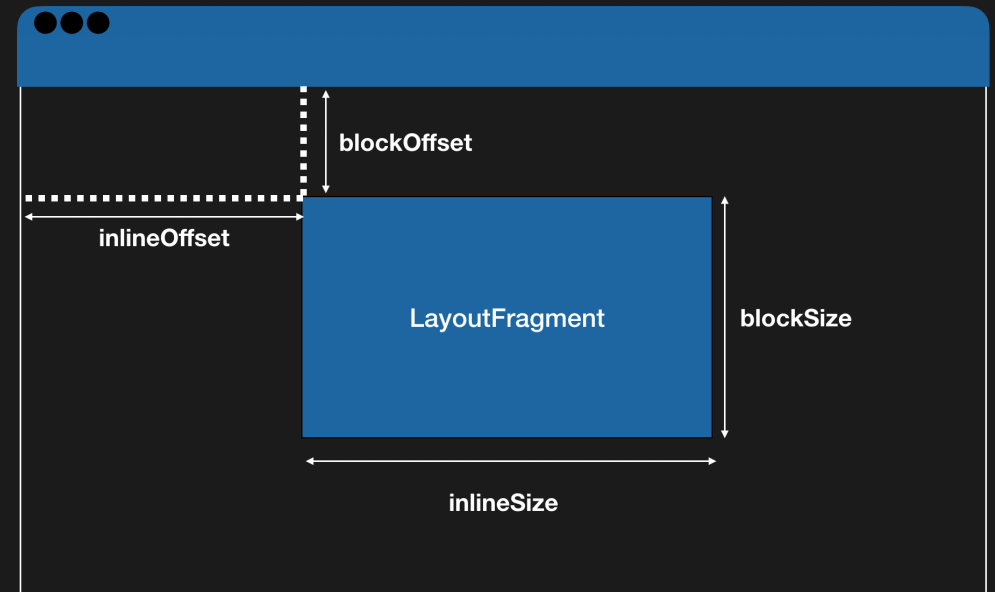
layoutNextFragment 함수는 LayoutFragment 객체를 반환합니다.

LayoutFragment 객체는 4가지의 정보를 가지고 있습니다.

1. blockSize
2. blockOffset
3. inlineSize
4. inlineOffset

blockSize, inlineSize는 fragment의 크기를 나타냅니다.

blockOffset, inlineOffset은 fragment가 원점으로 부터 떨어진 거리를 나타냅니다.



예제

index.html

```
<head>
  <link rel="stylesheet" href="index.css">
  <script src="index.js"></script>
</head>
<body>
  <div class="cloud">
    <div class="child">Websquare sp2</div>
    <div class="child">Websquare sp3</div>
    <div class="child">Websquare sp4</div>
    <div class="child">Websquare sp5</div>
    <div class="child">Inswave</div>
    <div class="child">IWD</div>
    <div class="child">Websquare</div>
    <div class="child">Websquare</div>
    <div class="child">Websquare</div>
  </div>
</body>
```

index.js

```
CSS.layoutWorklet.addModule('cloudLayout.js');
```

index.css

```
.cloud{
  display: layout(cloud);
  --random-seed: 30;
  width : 500px;
  height : 500px;
  border-radius: 25px;
}

.child{
  font-size: 24px;
  font-weight: blod;
  color: #579DDC;
  text-shadow: 0px 2px 2px rgba(0,0,0,0.3);
}
```

cloudLayout.js

```
class CloudLayout {
  static get inputProperties() {
    return ["--random-seed", "--cloud-level"];
  }

  *intrinsicSizes(children, edges, styleMap) {
  }

  *layout(children, edges, constraints, styleMap){
    const childFragments = yield children.map(child => {
      return child.layoutNextFragment(constraints)
    })

    const availableInlineSize = constraints.fixedInlineSize;
    const availableBlockSize = constraints.fixedBlockSize;
    const randomSeed = parseInt(styleMap.get("--random-seed"));

    let seed = randomSeed;
    const random = () => {
      let x = Math.sin(seed++) * 10000;
      return x - Math.floor(x);
    }

    let nextBlockOffset = 0;
    for (const fragment of childFragments) {
      let i = 0;
      console.log(fragment)
      fragment.blockOffset = random() * availableBlockSize;
      fragment.inlineOffset = random() * availableInlineSize;
    }

    return{
      childFragments
    }
  }
}

registerLayout("cloud", CloudLayout);
```

Typed OM

CSS에는 CSSOM이 있습니다.

CSSOM은 JavaScript에서 CSS를 조작할 수 있게 해주는 API입니다. CSSOM은 웹 페이지에서 발견되는 CSS 스타일의 기본 '맵'으로, DOM과 결합된 CSSOM은 브라우저에서 웹 페이지를 표현하는데 사용됩니다.

JavaScript에서 .style을 read 또는 set할 때 항상 아래와 같이 사용해왔습니다.

```
// 요소의 스타일
el.style.opacity = 0.3;
typeof el.style.opacity === 'string' // true??

// 스타일시트 규칙
document.styleSheets[0].cssRules[0].style.opacity = 0.3;
```

CSS Typed OM이란?

새로 나온 CSS Typed Object Model(Typed OM)은 CSS 값에 타입과 메소드, 적절한 객체모델을 추가함으로써 세계관을 넓혔습니다.

값이 문자열이 아닌 JavaScript 객체로 나타나기 때문에 CSS를 효율적으로(정상적으로) 조작할 수 있습니다.

기존의 사용하던 `element.style` 대신, `.attributeStyleMap` 속성을 사용하여 스타일에 접근할 수 있습니다.

스타일시트 규칙에는 `.styleMap` 속성을 사용합니다.

두 속성 모두 `StylePropertyMap` 객체를 반환합니다.

```
// 요소의 스타일
el.attributeStyleMap.set('opacity', 0.3);
typeof el.attributeStyleMap.get('opacity').value === 'number' // true 숫자값이다!!!!

// 스타일시트 규칙
const stylesheet = document.styleSheets[0];
stylesheet.cssRules[0].styleMap.set('background', 'blue');
```

StylePropertyMap은 Map과 유사한 객체이기 때문에, 일반적인 함수(get/set/keys/values/entries)를 전부 지원합니다. 따라서 아래와 같이 유연하게 작업할 수 있습니다.

```
// 아래 3가지가 모두 동일하다.
el.attributeStyleMap.set('opacity', 0.3);
el.attributeStyleMap.set('opacity', '0.3');
el.attributeStyleMap.set('opacity', CSS.number(0.3)); // 'Unit values' 파트 참고
// el.attributeStyleMap.get('opacity').value === 0.3

// StylePropertyMaps은 반복 가능하다.
for (const [prop, val] of el.attributeStyleMap) {
  console.log(prop, val.value);
} // → opacity, 0.3

el.attributeStyleMap.has('opacity') // true
el.attributeStyleMap.delete('opacity') // opacity 제거
el.attributeStyleMap.clear(); // 모든 스타일 제거
```

두 번째 예에서 opacity를 문자열 '0.3'으로 set 했지만 속성을 read 할 때는 숫자로 읽힌다는 것을 명심하세요.

주어진 CSS 속성이 숫자를 지원한다면, Typed OM은 문자열 값을 입력하더라도 항상 숫자값을 반환합니다!

이점

CSS Typed OM이 해결하려는 문제가 무엇일까요? CSS Typed OM이 이전의 Object Model보다 훨씬 장황하다고 주장할 수도 있습니다.

Typed OM을 작성하기 전에 아래의 몇 가지 주요 특징을 고려하세요.

1. 적은 버그 – 예) 숫자 값은 문자열이 아니라 항상 숫자로 반환됩니다.

// CSSOM은 문자열로 붙는다!// CSSOM은 문자열로 붙는다!

2. 산술 연산 및 단위 변환 – 절대 길이 단위를 변환하고(px → cm), 기본 수학 연산을 수행할 수 있습니다.








3. 값 클램핑 & 반올림 – Typed OM은 값을 반올림 및 클램핑해서 속성의 허용 범위 내에 있을 수 있습니다. ex) opacity <= 1

컴퓨터 그래픽에서 '클램핑'이란, 어떤 위치를 범위 안으로 한정시키는 방법입니다. 위치를 제일 가까운 사용 가능한 값으로 옮깁니다.

4. 성능 향상 – 브라우저는 문자열 값을 직렬화, 병렬화하는 작업을 줄여야 합니다. 이제 엔진은 JS, C++과 비슷한 방식으로 CSS 값을 이해합니다. Tab Akins는 초기 CSS 벤치마크에서 Typed OM이 기존의 CSSOM을 사용할 때보다 초당 작동 속도가 30%까지 빠르다는 것을 입증했습니다. 이는 `requestAnimationFrame()`를 사용하여 빠른 CSS 애니메이션을 구현할 때 중요합니다.
5. 오류 처리 – 새로운 파싱 메소드는 CSS 세계에서 오류 처리를 제공합니다.
6. CSSOM은 이름이 camel-case인지 문자열인지 가늠할 수 없었습니다(ex. `el.style.backgroundColor` vs `el.style['background-color']`). Typed OM의 CSS 속성 이름은 항상 문자열이며, 실제 CSS에서 작성한 것과 일치시키면 됩니다.

그러나...

Is houdini ready yet

							
	Google Chrome	Mozilla Firefox	Opera	Microsoft Edge	Samsung Internet	Apple Safari	Spec
Layout API (Explainer Demos)	Partial support (Canary) Details	Intent to implement Details	No signal	No signal	No signal	No signal	First Public Working Draft Spec
Paint API (Explainer Demos Article)	Shipped (Chrome 65) Details	Intent to implement (Servo) Details	Shipped (Opera 52) Details	No signal	Shipped (Internet 9.2) Details	In Development Details	Candidate Recommendation Spec
Parser API (Explainer)	No signal	No signal	No signal	No signal	No signal	No signal	No spec
Properties & Values API (Demos)	Partial support (Canary) Details	In Development Details	No signal	No signal	No signal	Partial support (Safari TP 67) Details	Working Draft Spec
AnimationWorklet (Explainer Demos Article)	Partial support (Canary) Details	No signal	No signal	No signal	No signal	No signal	First Public Working Draft Spec
Typed OM (Explainer Article)	Shipped (Chrome 66) Details	Intent to implement Details	Shipped (Opera 53) Details	No signal	Shipped (Internet 9.2) Details	In Development Details	Working Draft Spec
Font Metrics API (Explainer)	No signal	No signal	No signal	No signal	No signal	No signal	No spec

Maintained by [Surma](#), source code & license in the [repository](#) on GitHub, last updated on Mon Jun 03 2019 22:47:49 GMT+0100 (British Summer Time)

Reference

- [houdini-draft](#)
- [googlechromelabs-sample](#)
- [masonry - sample](#)
- [css-houdini.rocks](#)
- [Is houdini ready yet](#)