Day 05 (2019-07-03)
3Javascript trouble shooting - 응답없는 페이지

프로그래스 바 구현

→ 작업 진행 상황을 프로그래스 바로 확인할 수 있다

→ 중간에 진행 상태를 확인할 수 있다

→ 최초 상태는 Initial이며 진행중은 In progress, 완료는 Finished이다.

→ 모든 작업이 끝나면 'Job was finished' 알림이 발생한다

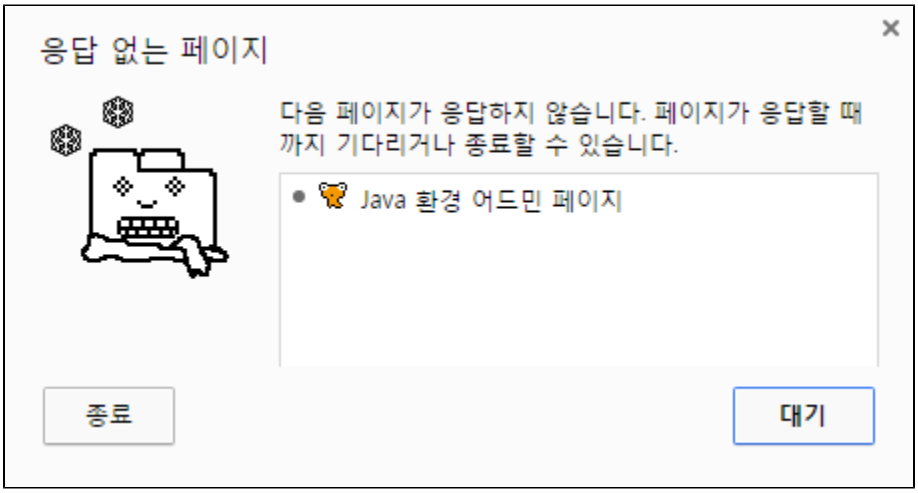
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        #progressBar {
            position: relative;
            width: 100%;
            height: 30px;
            background-color: #ddd;
        }

        #progressPercent {
            position: absolute;
            width: 0%;
            height: 100%;
            background-color: #0dff97;
        }
    </style>
</head>
<body>
<input type="button" value="" onclick="checkStatus();">
<input type="button" value="" onclick="execute();">

<div id="progressBar" style="margin-top:20px;">
    <div id="progressPercent"></div>
</div>

</body>
<script>
    var status = 'Initial';
    var loopSize = 10000000;
    var $progressPercent = document.getElementById('progressPercent');

    function onSuccess() {
        status = 'Finished';
        alert('Job was finished');
    }

    function checkStatus() {
        alert(status);
    }

    function execute() {
        var $progressPercent = document.getElementById('progressPercent');
        status = 'In progress';

        for (var i = 0; i < loopSize; i++) {
            $progressPercent.style.width = (i / loopSize) * 100 + '%';
        }

        onSuccess();
    }

</script>
</html>
```

제대로 실행되지 않음

→ 모든 작업이 완료될 때까지 버튼의 이벤트 반응이 없음

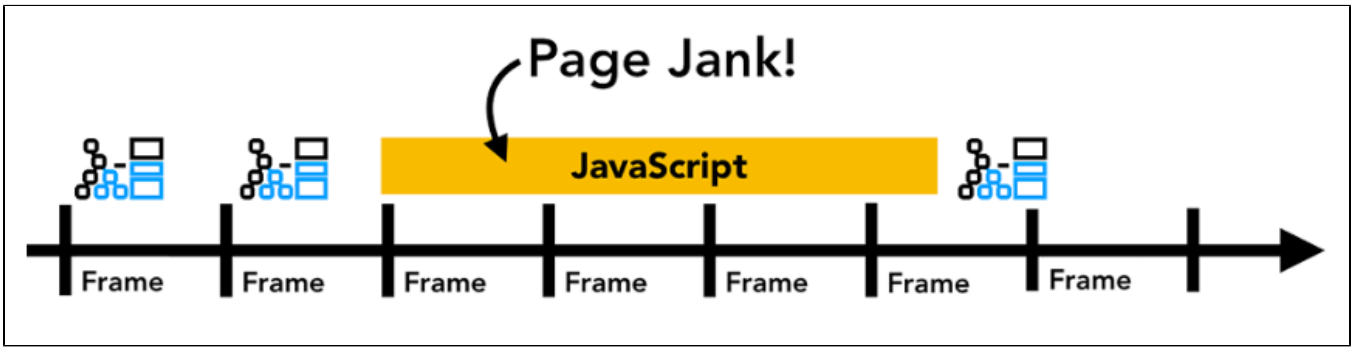
→ 프로그래스 바의 움직임이 보이지 않음

→ 더 많은 시간이 소용되는 작업이라면?



→ 심각한 경우 gif와 이미지도 움직이지 않을 수 있음

Page Jank



대부분의 화면 주사율은 60 FPS

브라우저는 이에 맞추어 초당 60 프레임으로 페이지를 렌더링 함

CSS 스타일링,Layout, Painting 등의 모든 과정을 거쳐 다음 프레임을 렌더링 하기 위한 시간을 프레임 예산(Frame budget)이라고 함

프레임 예산은 16.67ms를 반드시 유지해야 함

만약 프레임 예산 내에서 작업이 완료되지 않으면?
→ 프레임 드롭, 다음 페이지 사이의 전환이 매끄럽지 않아 애니메이션이 버벅거리거나 점렌ㄷ

Javascript를 통해 시각적 업데이트를 처리할 때 브라우저에서 일어나는 단계

렌더러 프로세스의 Main thread가 하는일

- Layout
- DOM 렌더링 (Painting, Composite)
- Javascript 실행

결과적으로 오랜시간 작업이 걸리는 Javascript code 때문에 Page jank 발생

타이머 분할 수행방법

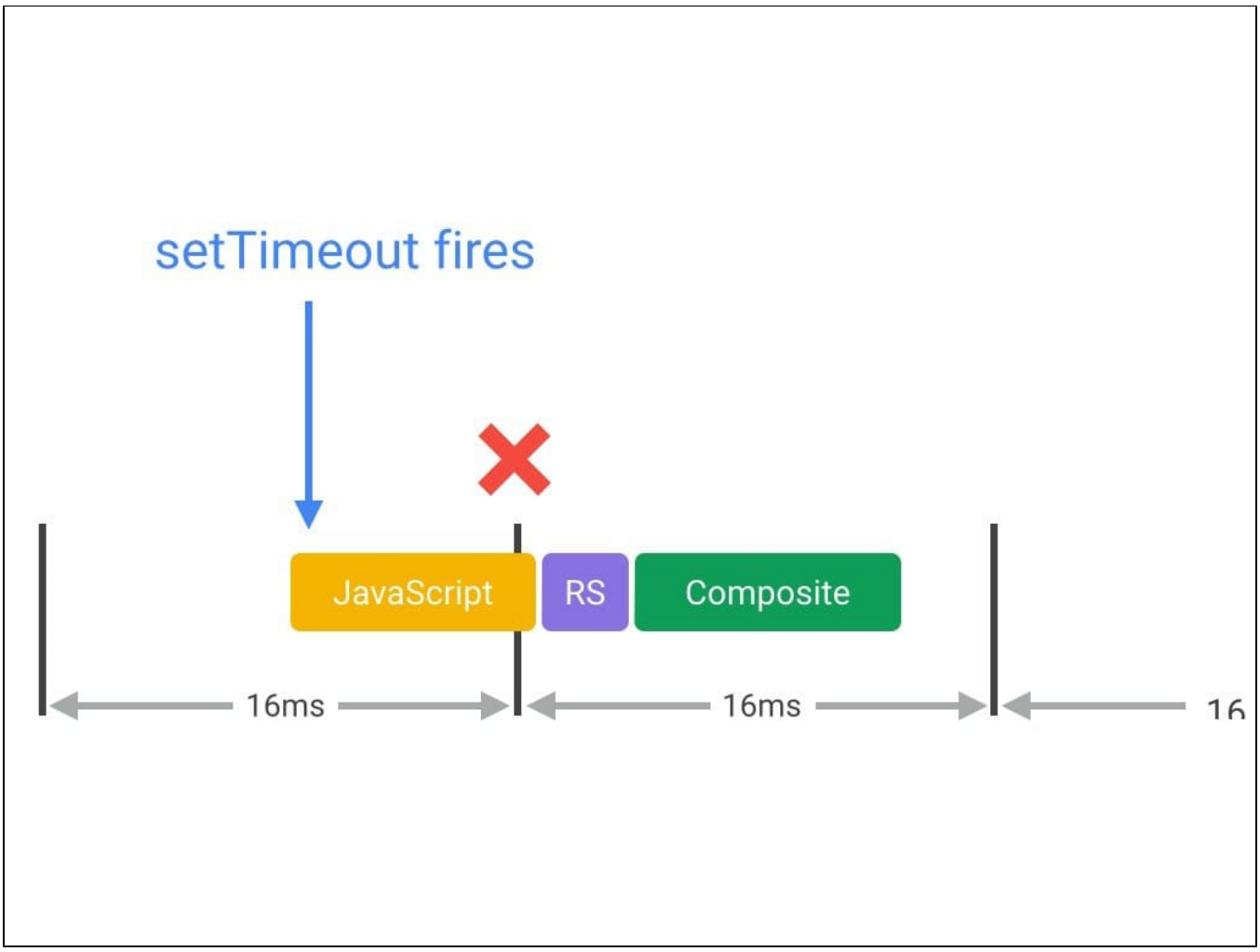
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        #progressBar {
            position: relative;
            width: 100%;
            height: 30px;
            background-color: #ddd;
        }

        #progressPercent {
            position: absolute;
            width: 0%;
            height: 100%;
            background-color: #0dff97;
        }
    </style>
</head>
<body>
<input type="button" value="" onclick="checkStatus();">
<input type="button" value="(setTimeout)" onclick="executeBySetTimeout();">

<div id="progressBar" style="margin-top:20px;">
    <div id="progressPercent"></div>
</div>

</body>
<script>
    var status = 'Initial';
    var loopSize = 10000000;
    var $progressPercent = document.getElementById('progressPercent');

    function onSuccess() {
        status = 'Finished';
        alert('Job was finished');
    }

    function checkStatus() {
        alert(status);
    }

    var startIndex = 0;
    var maxProcessCount = 1000;
    function job() {
        var endIndex = startIndex + maxProcessCount;
        for (var i = startIndex; i < endIndex; i++) {
            $progressPercent.style.width = (i / loopSize) * 100 + '%';
        }

        startIndex = endIndex;
        if (status !== 'Finished' && startIndex >= loopSize) {
            onSuccess();
        }
    }

    function executeBySetTimeout() {
        status = 'In progress';

        var jobCount = loopSize / maxProcessCount;
        for (var i = 0; i < jobCount; i++) {
            setTimeout(job, 1000 / 60);
        }
    }
</script>
</html>
```

큰 작업을 분할하고(Paging 하듯이) setTimeout을 통해서 나누어진 작업을 적재

Main thread가 여러 일을 할 수 있음
→ 작은 단위의 javascript 작업과 Rendering을 할 수 있음
→ CPU의 Time slice 처리와 비슷
→ Page jank를 줄일 수 있음
→ setTimeout의 지정된 시간은 60 FPS (1000ms / 60 frame, 약 16ms)에 맞추는 것이 최적

하지만 약간의 버벅거림
→ 반드시 60 FPS에 맞추어서 동작한다는 보장이 없음
→ Callback이 프레임의 종료 시점에 실행되어 Drop 될 수 있음

만약 작업을 나누는 데에도 시간이 오래 걸리면...?
→ 답 없음 or Callback hell

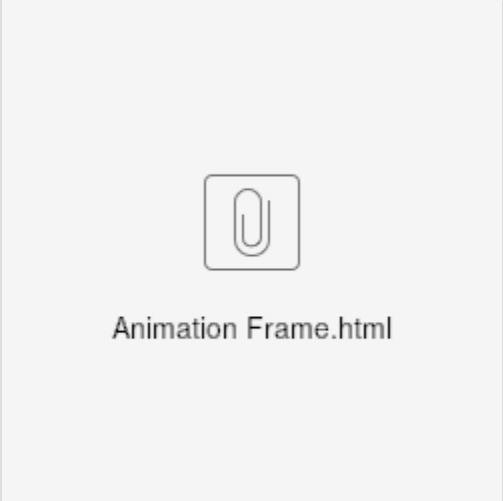


화면의 시각적 변화에서 버벅거리지 않는 방법의 핵심은 프레임의 시작 시점에 시각적 업데이트 Javascript 작업이 진행 되어야한다는 것.

requestAnimationFrame 사용

화면의 시각적 변화를 적용하기 위해 javascript가 프레임의 시작 시 실행되도록 보장하는 유일항 방법

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        #progressBar {
            position: relative;
            width: 100%;
            height: 30px;
            background-color: #ddd;
        }

        #progressPercent {
            position: absolute;
            width: 0%;
            height: 100%;
            background-color: #0dff97;
        }
    </style>
</head>
<body>
<input type="button" value="" onclick="checkStatus();">
<input type="button" value="(AnimationFrame)" onclick="executeByAnimationFrame();">

<div id="progressBar" style="margin-top:20px;">
    <div id="progressPercent"></div>
</div>

</body>
<script>
    var handle = 0;
    var lPos = 0;
    var status = 'Initial';
    var loopSize = 10000000;
    var $progressPercent = document.getElementById('progressPercent');


    function onSuccess() {
        status = 'Finished';
        alert('Job was finished');
    }

    function checkStatus() {
        alert(status);
    }


    function renderLoop() {
        lPos += 10000;
        $progressPercent.style.width = (lPos / loopSize) * 100 + '%';

        if (lPos === loopSize) {
            window.cancelAnimationFrame(handle);
            onSuccess();
            return;
        }

        handle = window.requestAnimationFrame(renderLoop);
    }

    function executeByAnimationFrame() {
        status = 'In progress';
        renderLoop();
    }
</script>
</html>
```



window.requestAnimationFrame(callback);

callback은 브라우저가 Repaint 하기 전에 실행 됨

실행되는 주기는 보통 1초에 60회, 즉 60 FPS이지만 W3C 권장사항에 따라 디스플레이 주사율과 일치하게 함 (대부분 모니터는 60 FPS를 따름)

함수 실행후 해당 요청의 고유한 ID를 반환 함.
→ window.cancelAnimationFrame(ID)를 통해 콜백 요청을 취소할 수 있음

백그라운드 탭이나 hidden인 〈iframe〉에서는 실행이 중단 됨

하지만 (~~IE 너란녀석...~~)

> 노트
>
> 인터넷 익스플로러와 Edge 17 버전 이하는 페인트 사이클 이전에 `requestAnimationFrame` 실행을 보장하지 않습니다.

게다가 브라우저마다 약간씩 이름이다름...

Polyfill

```
window.requestAFrame = (function () {
        return window.requestAnimationFrame ||
            window.webkitRequestAnimationFrame ||
            window.mozRequestAnimationFrame ||
            window.oRequestAnimationFrame ||
            function (callback) {
                return window.setTimeout(callback, 1000 / 60); // shoot for 60 fps
            };
    })();
```

→ 모든 케이스를 지원하지 않을 경우엔 setTimeout을 통해서 처리

만약 시각적 변화가 아닌, DOM에 접근이 필요하지 않은, 순수한 계산 작업이 필요하다면?

→ Web Workers를 이용할 수 있음

```
var dataSortWorker = new Worker("sort-worker.js");
dataSortWorker.postMesssage(dataToSort);

// The main thread is now free to continue working on other things...

dataSortWorker.addEventListener('message', function(evt) {
   var sortedData = evt.data;
   // Update data on screen...
});
```

더 읽어보기

WebRender가 Page jank를 방지하는 방법(GPU)

CSS GPU 애니메이션 제대로 하기

참고자료

https://developer.mozilla.org/ko/docs/Web/API/Window/requestAnimationFrame

https://github.com/gnarf/jquery-requestAnimationFrame (jQuery animate를 사용하고 있다면...)