Technology Report

작성자 : 오다은

```css
body {
  opacity: 0;
  transition-duration: 3s;
  transition-property: opacity text-shadow;
  text-shadow: 0 0 10px transparent;
}

body.fade_out {
  opacity: 1;
  text-shadow: 0 0 0 transparent;
}
```

```js
window.addEventListener("load", () => {
  document.body.classList.add("fade_out")
});
```

▶ 위의 기술을 적용한 이유는 무엇인가?

메뉴 클릭 시 페이지를 이동시킬 때 바로 나타나는 것이 아닌 브라우저상에 아무것도 보이지 않다가 기다린 뒤에 html 전체가 자연스럽게 나타나는 것을 구현하고 싶었다.

body {

opacity

: (요소를 투명하게 만드는 속성, 없다가 나타난 것처럼 보이기 위해 0 값과 1 값 적용),

transition-duration

: (애니메이션 완료되는데 걸리는 시간, 시간을 두고 나타나게 하기 위해 3s 적용),

transition-property

: (전환 효과를 적용할 css 속성 설정, opacity text-shadow 두 가지 작성했기 때문에 값 적용)

text-shadow

: (텍스트 그림자 효과, 블러처럼 보이고자 transparent:투명한 색 적용),

}

▶ 문제를 해결하기 위해 어떤 노력을 했는가?

어떤 이벤트 값을 줘야 할지 몰라 “페이지 이동 시 부드럽게” 등 다양한 검색을 했다. 부드러운 스크롤 이동이 가장 많이 나왔고 내가 원하는 정보와 비슷한 내용을 찾아서 적용시켰다.

첫 번째 시도,

```
$(function () {
    $("body").fadeIn(500, function () {
      $(this).animate(
        {
          top: "100px",
        },
        1000
      );
    });

    $(".moving").click(function () {
      var url = $(this).attr("href");
      $("body").animate(
        {
          opacity: "0",
          top: "100px",
        },
        5000,
        function () {
          document.location.href = url;
        }
      );

      return false;
    });
 });
```

html에서 header nav에 class를 주고 javascript에서 body에 fadeIn 값을 넣어 animate를 적용시키는 방법이다. 실행되기는 하되 버벅거리고 모든 'a[href]' 값에 적용되어 방법을 아예 바꾸기로 했다.

두 번째 시도,

```
document.addEventListener("DOMContentLoaded", () => {
     window.setTimeout(() => {
     document.body.classList.add("fade_out")
  },500);
});
```

적용했을 때는 자연스럽게 나타나되 다른 javascript나 css 와 충돌이 생긴 것처럼 브라우저 상 오류가 나타났고 메뉴 간 이동 시 값이 적용되지 않고 나타날 때가 생겼다. 무슨 문제일지

자바스크립트에서 찾아보니 “DOMContentLoaded” 이 이벤트가 갖고 있는 성격 때문에 그랬고 대체할 수 있는 load를 적용시켰고 오류 없이 작동하는 걸 확인할 수 있었다.

▶ 인상 깊었던 점

뒤로 가기, 새로고침, html 로드 되는 것, 페이지를 떠나는 것, 다른 탭으로 이동하는 것들도 전부 브라우저가 가지고 있는 이벤트이고 document 자체로 이 모든 것들을 감지할 수 있다는 점이 재밌었다. 이로 인해 브라우저 상에서 할 수 있는 모든 행동은 전부 이벤트라는 것을 알게 되었고 “DOMContentLoaded” 와 “load”의 차이를 알게된 게 가장 값진 결과물이었다. “load” 대신 “DOMcontentLoaded”를 적용시켰을 때는 될 때도 있고 안될 때도 있는 오류가 있었다. 오류가 나와 “DOMcontentLoaded” 검색을 해봤는데 “DOMcontentLoaded”은 html을 읽는 즉시 스타일 값을 기다려주지 않고 즉시 실행시키는 요소였다. 그래서 css에 내가 적용한 값을 불러오기도 전에 시작하다 보니 오류가 발생했었고 load는 html을 읽고 스타일 값을 전부 불러온 다음에 시작하는 요소였기 때문에 무사히 오류 없이 실행되었다.
아직 자바스크립트가 낯설고 어렵게만 느껴져서 코드도 당연히 길 것이라 생각했는데 많이 고민했던 거에 비해 js 코드는 짧았고 css에 적용할게 상대적으로 더 많아서 작업하면서 재밌으면서도 신기했던 코드였다.