HTML 페이지 안에 자바스크립트 코드는 아무 위치에나 올 수 있습니다.

심지어는 <html></html> 태그 바깥에도 위치할 수 있습니다.

HTML 페이지 안에 자바스크립트 코드를 넣을 때는 "<script></script>" 태그를 사용합니다.

"<scrip><script>" 태그 안에 직접 자바스크립트 코드는 작성해 넣는 것과, 외부 자바스크립트 파일의 링크를 "src" 속성에 표시를 해서 "<script src='경로'></script>"로 연결을 하는 방법 두 가지가 있습니다.

두 가지 방법 모두 자바스크립트 코드를 실행하는 정상적인 방법입니다.

우리가 보는 유명 사이트들은 모두 <head></head> 태그 안에 자바스크립트 코드 태그가 위치합니다.

기본적으로 자바스크립트 코드는 모두 <head><head> 태그 안에 작성을 한다고 생각하면 됩니다.

다만 그것은 권장 사항이고 필수 사항은 아닙니다.

필요에 따라 자바스크립트 태그는 HTML 문서 중간에 위치할 수도 있으며, 문서 맨 끝에 위치할 수도 있습니다.

웹 브라우저가 웹 페이지에서 자바스크립트 태그를 파싱(Parsing) 하는 순서는 다음과 같습니다.

1. HTML 파일 열기.

2. HTML 페이지의 태그를 **위에서부터 순차적으로 파싱 시작.**

3. <script> 태그를 발견하면 자바스크립트 코드 블록을 읽음. 외부 자바스크립트 파일 링크이면 파일을 요청함. **HTML 태그 파싱은 일단 멈춥니다**. 중요합니다!

4. <script> 태그의 자바스크립트 코드, 또는 외부 자바스크립트 파일의 코드를 인터프리팅(Interpreting)해서 실행합니다.

5. 종료 태그(</script>) 이후의 HTML 태그를 파싱 시작.

중요한 것은 3번의 자바스크립트 코드를 읽어서 실행하는 동안 HTML 파싱이 멈추는 것입니다.

다운로드하는데 오랜 시간이 걸리는 큰 파일을 가진 자바스크립트 라이브러리나, 실행 시간이 오래 걸리는 자바스크립트 코드를 HTML 페이지 중간에 넣으면 웹 페이지 로딩과 렌더링 완료까지 오랜 시간이 걸리게 되고, SEO에 최악의 결과를 가져오게 됩니다.

<head></head> 안에 자바스크립트 태그를 넣으면 웹 브라우저는 이 자바스크립트 태그의 코드, 또는 링크한 파일(들)을 모두 읽어서 실행한 후 가 다음 <body> 태그의 내용을 읽기 시작합니다.

웹 브라우저가 자바스크립트 코드 실행 완료 시점까지 파싱을 멈추고 기다리는 이유는 자바스크립트 코드로 HTML 태그를 생성할 수 있기 때문입니다. "document.write()" 함수로 HTML 태그를 생성하게 되면, 자바스크립트 코드 실행 위치에 새로운 HTML 태그가 생겨나게 되고, 만약 웹 브라우저가 자바스크립트 코드 이후의 HTML 태그 파싱을 진행한 경우 다시 되돌아와서 자바스크립트 코드 위치부터 다시 파싱을 시작해야 합니다.

그래서 자바스크립트 코드가 HTML 파싱 후 DOM이 생성된 이후에 실행되도록 하는 여러가지 방법을 사용하게 됩니다.

1. HTML 페이지 맨 끝에 자바스크립트 코드 적용

<script> 태그는 </html> 태그 뒤에 올 수 있습니다.

</html> 태그 뒤에 자바스크립트 코드가 오면 HTML DOM이 생성된 후 자바스크립트 코드가 실행됩니다.

자바스크립트 코드의 실행 순서에 이해가 없거나 DOM 생성을 체크하는 코드를 사용할 줄 모를 때 많이 사용합니다.

확실한 방법이지만, HTML 페이지 작성시에 권장하지 않는 방법입니다.

2. DOM 생성이 완료되었는지 체크

자바스크립트 코딩을 실제로 하게 되면 가장 많이 보게 되는 코드 중의 하나입니다.

HTML DOM이 생성되었는지를 체크하는 기본 이벤트 리스너 코드이고, 표준 사용방법입니다.

"DOMContentLoaded" 이벤트는 DOM 생성이 완료되면 웹 브라우저가 발생시키는 이벤트입니다. "DOM 생성이 완료됨. 님 이제 웹페이지 컨텐츠에 접근할 수 있음."입니다.

이벤트가 발생하면 이벤트 리스너 함수의 두 번째 인자로 정의한 콜백 함수 "function(){}"을 실행하는 이벤트 리스너를 등록하는 코드입니다.

document.addEventListener("DOMContentLoaded", function() {

*// DOM 생성이 완료된 후 실행*

});

 onload = function(){{

*// body 태그를 모두 읽은 후 실행*

});

$(document).ready(function(){

*// body 태그를 모두 읽은 후 실행*

});

$(function(){

*// body 태그를 모두 읽은 후 실행*

});

**3. async**

외부 자바스크립트 파일을 링크해서 가져올 때 태그 속성으로 "async"를 추가합니다.

"async" 속성이 추가된 자바스크립트 파일은 HTML 페이지 파싱과 비동기로 로딩되고 실행됩니다.

"async" 속성으로 인해 HTML 페이지 파싱이 블록 되지 않고 계속 진행되는 것이 중요합니다.

<script src="파일경로/myscript1.js" async></script>

<script src="파일경로/myscript2.js" async></script>

"async"는 한 가지 주의해야 할 점이 있습니다.

비동기로 로딩되기 때문에 자바스크립트 파일의 링크 순서에 상관없이 로딩됩니다.

뒤에 있는 자바스크립트 파일이 먼저 로딩되어 실행될 수 있습니다.

상호 의존적이거나 연관 관계가 있는 자바스크립트 파일이나 코드들은 "async"를 사용하면 안 됩니다.

어떤 경우에는 잘 실행되겠지만, 어떤 경우에는 참조 에러가 나면서 자바스크립트 코드 실행이 멈출 수도 있습니다.

같은 이유로 HTML 페이지 상단에서 비동기(async)로 로딩되어 실행되는 자바스크립트 코드는 HTML 페이지 파싱이 끝나기 전에 실행될 가능성도 있습니다. 그래서 테스트 및 개발을 할 때는 잘 실행되다, 막상 실 서비스에 적용하면 어떤 때는 실행되고, 어떤 때는 에러가 나는 이상한 현상이 발생하기도 합니다.

비동기로 가져오는 자바스크립트 코드는 2번의 DOM 생성 체크를 하는 코드가 필수적으로 필요하며, 실행 순서에 주의해서 상호 의존적인 실행 코드들은 하나의 파일에 모아서 가져오는 방식을 사용해야 합니다.

**4. defer**

async의 문제를 해결하기 위해 만들어진 속성입니다.

비동기로 자바스크립트 코드를 로딩해서 실행하지만, 순서대로 처리된다.

"async" 와의 가장 중요한 차이점이 "순서대로"입니다. 스크립트 태그가 위치한 순서대로 차례로 실행됩니다.

더 중요한 차이가 있습니다. "defer" 속성으로 정의한 자바스크립트는 DOM이 생성된 후에 실행됩니다.

따라서 2번의 DOM 생성 체크 이벤트 핸들러 처리를 할 필요가 없습니다.

다음 자바스크립트 링크는 헤더 영역에 정의되었지만 자바스크립트 파일 안에 2번의 DOM 생성 체크 코드를 넣을 필요도 없고, 두 번째 스크립트 파일에서 첫 번째 스크립트 파일의 함수를 호출해도 아무런 문제 없이 실행됩니다.

<head>

<script src="파일경로/myscript1.js" defer></script>

<script src="파일경로/myscript2.js" defer></script>

</head>

"defer"는 가장 최신의 자바스크립트 코드 실행과 관련된 기술이고, 최신 웹 브라우저에서는 모두 지원하기 때문에 빠르게 정착되어가고 있습니다.

아직 "defer"를 사용하고 있지 않으면 "defer"를 사용해서 자바스크립트 파일을 링크하는 것을 추천합니다.