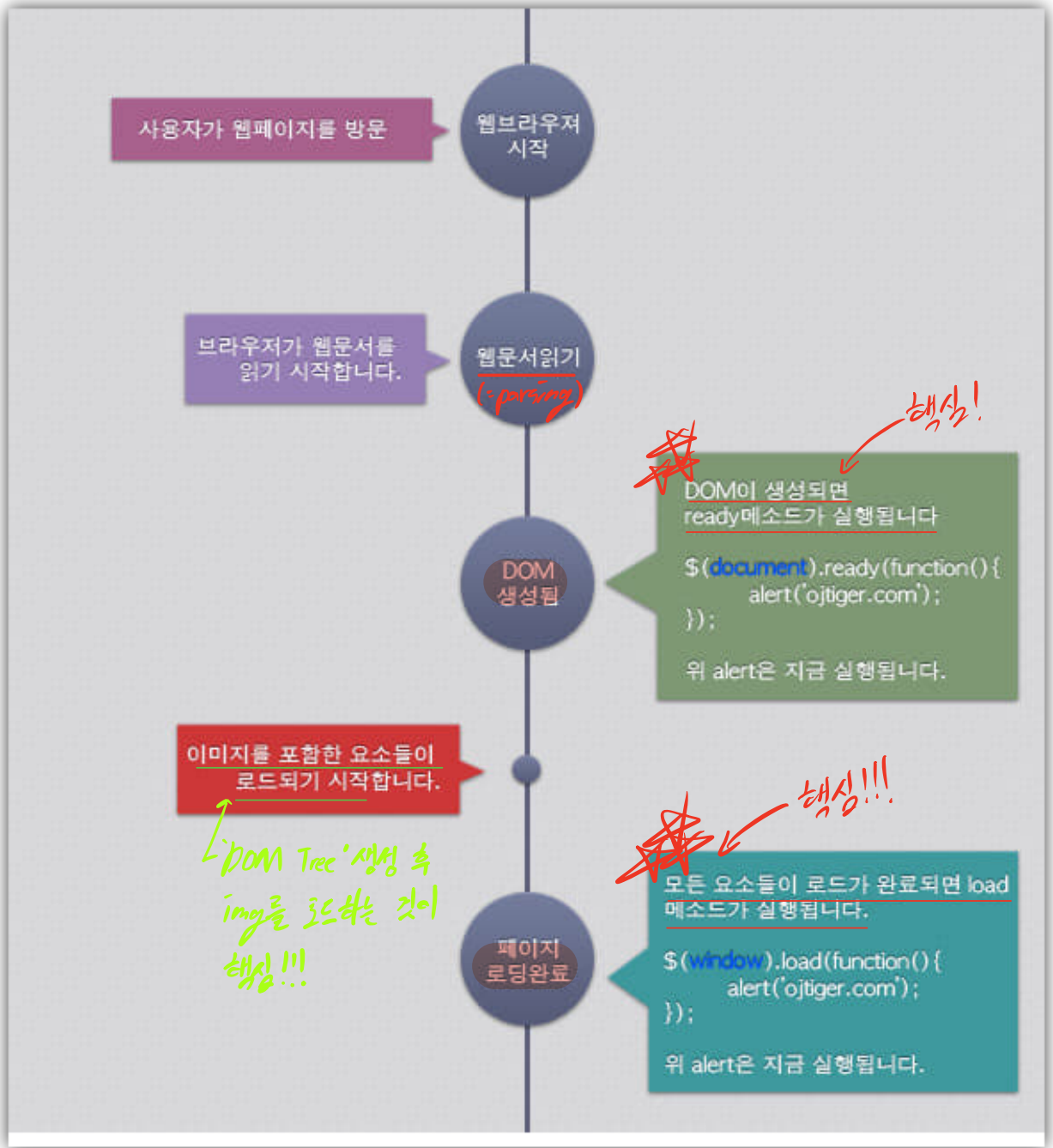

사용자가 웹페이지를 방문
웹브라우저 시작
브라우저가 웹문서를 읽기 시작합니다.
웹문서읽기 (=parsing)
DOM 생성됨
DOM이 생성되면 ready메소드가 실행됩니다
$(document).ready(function(){
alert('ojtiger.com');
});
위 alert은 지금 실행됩니다.
핵심!
이미지를 포함한 요소들이 로드되기 시작합니다.
'DOM Tree' 생성 후 img를 로드하는 것이 핵심!!!
페이지 로딩완료
모든 요소들이 로드가 완료되면 load 메소드가 실행됩니다.
$(window).load(function(){
alert('ojtiger.com');
});
위 alert은 지금 실행됩니다.
핵심!!!

렌더링 엔진

렌더링 엔진의 역할은 요청 받은 내용을 브라우저 화면에 표시하는 일을 수행합니다.
렌더링 엔진은 HTML 및 XML 문서와 이미지를 표시할 수 있다. 물론 플러그인이나 브라우저 확장 기능을 이용해 PDF와 같은 다른 유형도 표시할 수 있습니다.
대표적 렌더링 엔진으로 파이어폭스와 웹킷 엔진이 있는데 파이어폭스는 모질라에서 직접 만든 게코(Gecko) 엔진을 사용하고 사파리와 크롬은 웹킷(Webkit) 엔진을 사용합니다.
웹킷은 최초 리눅스 플랫폼에서 동작하기 위해 제작된 오픈소스 엔진인데 애플이 맥과 윈도우즈에서 사파리 브라우저를 지원하기 위해 수정을 가했습니다. 더 자세한 내용은 webkit.org를 참조하기 바랍니다.

파싱(parse or parsing)이란?

문서를 파싱한다는 것은 브라우저가 코드를 이해하고 사용할 수 있는 구조로 변환하는 것을 의미합니다.
파싱 결과는 보통 문서 구조를 나타내는 노드 트리인데 파싱 트리(parse tree) 또는 문법 트리(syntax tree)라고 부릅니다.

(=DOM 트리)

브라우저가 문서(HTML)를 해석하면서 하는일

브라우저는 기본적으로 다음과 같은 작업을 수행합니다.

1. **불러오기(Loading)** - 불러오기는 HTTP 모듈 또는 파일시스템으로 전달 받은 리소스 스트림(resource stream)을 읽는 과정으로 로더(Loader)가 이 역할을 맡고 있음. 로더는 단순히 읽는 것이 아니라, 이미 데이터를 읽었는지도 확인하고, 팝업창을 열지 말지, 또는 파일을 다운로드 받을지를 결정한다.
2. **파싱(Parsing)** - 파싱은 DOM(Document Object Model) 트리를 만드는 과정으로 일반적으로 HTML, XML 파서를 각각 가지고 있음. HTML 파서는 말 그대로 HTML 문서를 해석하는데 사용되고, XML 파서는 XML 형식을 따르는 SVG, MathML 등을 처리하는데 사용함.
3. **렌더링 트리(Rendering Tree)** 만들기 - 파싱으로 생성된 DOM 트리는 HTML/XML 문서의 내용을 트리 형태로 자료 구조화 한 것을 말한다. 다시 말해,
   DOM 트리는 내용 자체를 저장하고 있고, 화면에 표시하기 위한 위치와 크기 정보, 그리는 순서 등을 저장하기 위한 별도의 트리 구조가 필요한데 이를 일반적으로 렌더링 트리라고 부른다.
4. **CSS 스타일 결정** - CSS 는 을 나타내기 위해 만들어 졌음
   HTML 문서 내용과 별도로 표현
5. **레이아웃(Layout)** - 렌더링 트리가 생성될 때, 각 렌더(Render) 객체가 위치와 크기를 갖게 되는 과정을 레이아웃이라고 한다.
6. **그리기(Painting)** - 그리기 단계는 렌더링 트리를 탐색하면서 특정 메모리 공간에 RGB 값을 채우는 과정이다.

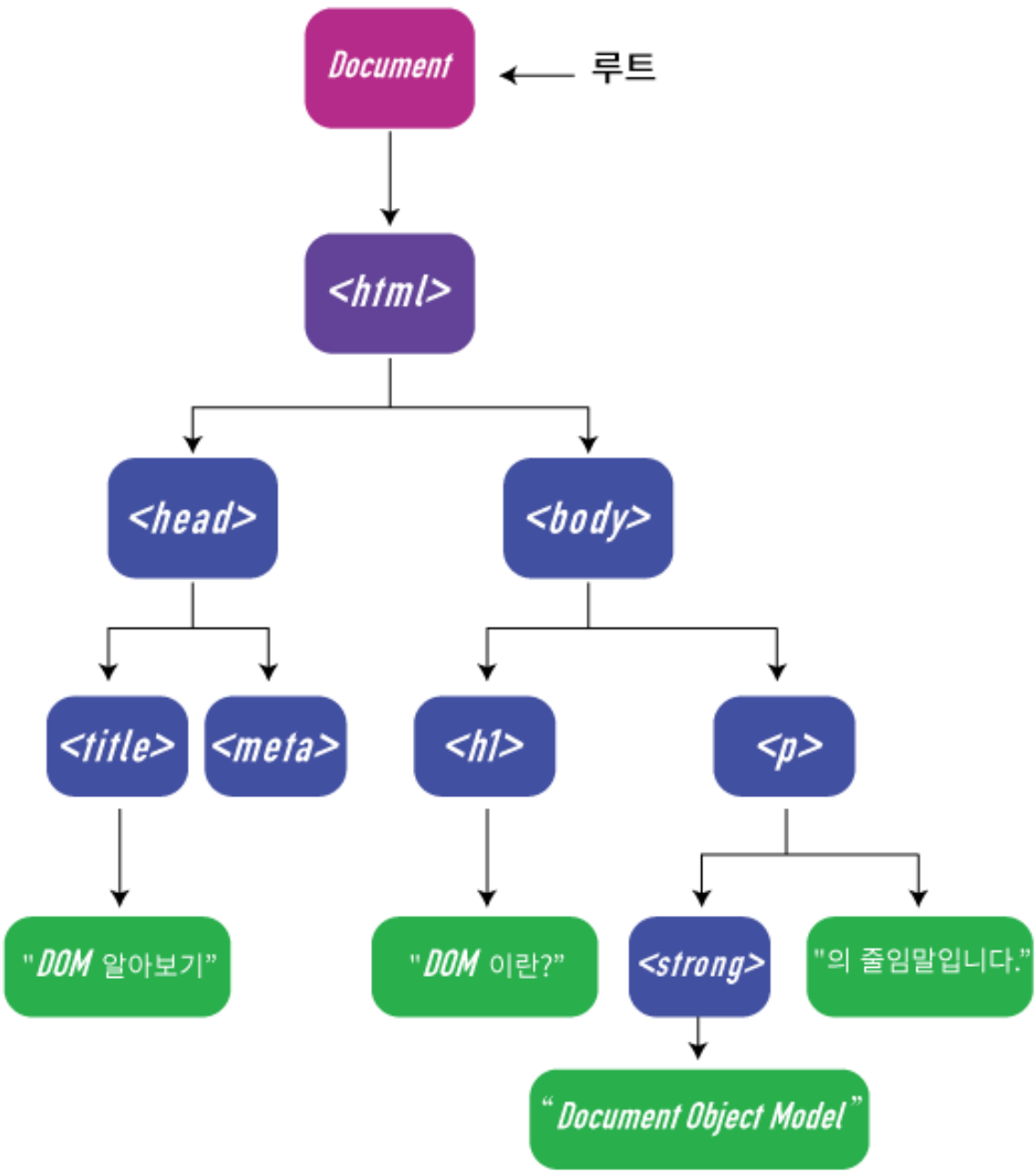
DOM 트리

웹 페이지의 모든 요소를 Document 객체가 관리합니다. 때문에 웹 페이지의 요소들을 관리하고 제어하기 위해서는 Document 객체가 웹 페이지 요소들을 잘 반영하는 자료구조를 가지고 있어야 합니다. 그래서 Document 객체 모델인 DOM은 트리 자료구조 형태를 가지고 있습니다. 트리 자료 구조는 HTML 문서를 읽어 들이고 제어하기 가장 좋은 자료구조입니다.

```html
<!DOCTYPE html>
<html lang="ko">
<head>
    <title>DOM 알아보기</title>
    <meta charset="UTF-8">
</head>
<body>
    <h1>DOM이란?</h1>
    <p><strong>Document Object Model</strong>의 줄임말입니다.</p>
</body>
</html>
```

dom.html hosted with ❤ by GitHub view raw

위의 html 파일을 브라우저에서 실행한 결과입니다.



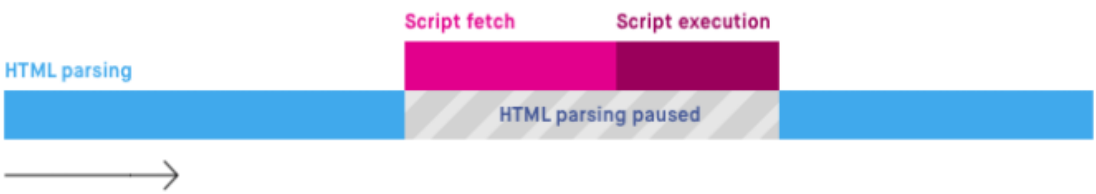
루트란? 뿌리를 뜻합니다. 최상위 계층의 노드(덩어리)입니다.

웹 브라우저가 HTML 문서를 읽어 들이면 위에 그림처럼 Document 객체로 시작하는 DOM 트리가 만들어집니다. 트리 자료구조에서는 트리를 구성하고 있는 객체 하나를 노드(Node)라고 부릅니다. 위 그림에서 볼 수 있는 기본적인 세 가지 노드를 먼저 정리하겠습니다.

1. 문서 노드: 보라색 도형으로 표현된 제일 위에 있는 노드가 문서노드입니다. 트리의 최상위 계층이면서 전체 문서를 가리키는 Document 객체입니다. document로 참조할 수 있으며 DOM 트리로 웹 페이지를 접근하는 시작점입니다.
2. 요소 노드: 파란색 도형으로 표현된 노드들이 요소 노드입니다. HTML 태그에 해당되는 요소들입니다. 요소 노드는 속성 노드와 텍스트 노드를 자식으로 가질 수 있습니다.
3. 텍스트 노드: 초록색 도형으로 표현된 노드들이 텍스트 노드입니다. HTML 태그 안에 있는 텍스트들이 텍스트 노드이며 이들은 요소 노드와 달리 자식 노드를 가질 수 없습니다.

🔍 브라우저의 동작 방식

1. HTML을 읽기 시작한다.
2. HTML을 파싱한다.
3. DOM 트리를 생성한다.
4. Render 트리(DOM tree + CSS의 CSSOM 트리 결합)가 생성되고
5. Display에 표시한다



여기서 주목해야 할 부분은 1, 2, 3의 순서입니다. HTML을 파싱한 다음 DOM 트리를 생성하죠. 그런데 브라우저는 HTML 태그들을 읽어나가는 도중 `<script>` 태그를 만나면 파싱을 중단하고 javascript 파일을 로드 후 javascript 코드를 파싱합니다. 완료되면 그 후에 HTML 파싱이 계속 됩니다.

⚙ body 태그 최하단이 가장 좋은 이유

이로인해 HTML태그들 사이에 script 태그가 위치하면 두가지 문제가 발생합니다.

1. HTML을 읽는 과정에 스크립트를 만나면 중단 시점이 생기고 그만큼 Display에 표시되는 것이 지연된다.
2. DOM 트리가 생성되기전에 자바스크립트가 생성되지도 않은 DOM의 조작을 시도할 수 있다.

위와 같은 상황을 막기 위해 script 태그는 body 태그 최하단에 위치하는 게 가장 좋습니다.

2. script 내부에서 로딩 순서 제어하기

DOMContentLoaded 와 onload

`DOMContentLoaded` 와 `onload` 를 활용하면 javascript 자체에서 로딩 순서를 제어할 수도 있습니다.

`DOMContentLoaded` 내부의 코드는 **DOM 생성이 끝난 후**에 실행되고

`onload` 내부의 코드는 **문서에 포함된 모든 콘텐츠(images, script, css, ...)가 전부 로드된 후**에 실행됩니다.

따라서

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <title>DOMContentLoaded</title>
</head>
<body>
    <script>
        // window.onload가 가장 앞에 위치!
        window.onload = function(){
            console.log("afterwindowload");
            var target = document.querySelector("#test");
            console.log(target);
        }
                // DOMContentLoaded가 두번째에 위치!
        document.addEventListener("DOMContentLoaded", function() {
            console.log("afterdomload");
            var target = document.querySelector('#test');
            console.log(target);
        });
                // 일반 script 코드가 가장 끝에 위치
        console.log("바로시작")
        var target = document.querySelector('#test');
        console.log(target);
    </script>
    <div id="test">test</div>
</body>
</html>
```

위 코드의 console에 출력된 결과는

```
바로시작
null
afterdomload
  <div id="test">test</div>
afterwindowload
  <div id="test">test</div>
```