HTML5& JavaScript 요약본

문서 객체 모델(DOM) a.k.a 클라이언트 측 JavaScript (⇔ 코어 JavaScript : 기본이 되는 부분만 정의)

↳ 웹 브라우저에 관련된 JavaScript

1. DOM의 이해와 Window 객체

① DOM 객체 계층 : 웹 브라우저와 웹 문서의 기능과 내용을 대부분 객체화 property로 정보를 읽거나 설정 가능
↓
구조화
↓
계층으로 구성

② 현재 웹 브라우저 윈도우
- Window 객체
- window 프레임 객체 배열
- Navigator 객체
- Location 객체
- History 객체
- Document 객체
- Screen 객체

window 객체 이외의 객체는 window 객체의 property로 접근

③ Window 객체 : 다른 DOM 객체에 접근할 수 있는 것 이외에 현재 웹 브라우저 창에 대한 여러 가지 정보 제공 ex) 크기, 위치, 이동(이미지확대관찰기능구현시 중요)

④ Window 생성 : 팝업 윈도우 생성 (open 메소드), 윈도우 닫기 (close 메소드), alert
↳ 부정적인 인식이 강함
thus 팝업차단기능은 기본!

↑ 실제에선 사용하지않지만 디버깅 시 유용하게 사용

2. 웹 문서 접근

① DOM을 통하여 웹 브라우저, HTML 문서의 모든 기능과 요소에 접근 가능

② 문서 객체 배열로 접근
- document.anchors[] : 문서 내 앵커 객체
- document.applets[] : Java 애플릿
- document.forms[] : 폼 객체
- document.images[] : 이미지 객체
- document.links[] : 링크 객체

③ 이름 속성으로 접근 : name 속성

④ DOM 문서 트리 : HTML 문서의 모든 요소 접근 가능, 중첩된 요소들 부모, 자손 관계로 계층화
새로운 DOM 표준은 각 노드에 접근하는 방식을 제공 ① 태그명 ② id속성

3. 웹 문서의 조작

① write/writeln 메소드 : 가장 쉽게 문서 내 콘텐츠 추가 writeln은 줄바꿈표시 이외 write와 동일 but HTML은 줄바꿈 무시하므로 <pre> 요소를 포함해야함

② 노드의 생성과 삽입 → 기존 contents를 없앰
- createElement() / createTextNode() 메소드 : 요소와 텍스트 노드 생성
- appendChild() 메소드 : 자손 노드 삽입 기존 요소의 끝에 생성
- insertBefore() 메소드 : 자손 요소 중 기존 요소의 ~~끝~~ 앞에 생성
- replaceChild() 메소드 : 자손 노드 중 특정 노드 새로운 노드로 치환

③ innerHTML 프로퍼티 : createElement()와 createTextNode() 그리고 appendChild()를 한꺼번에 처리

★ CSS 가정의되어있는 style property ⇒ CSS속성을 표시할때 '-'(하이픈)은 모두 대문자표기로 바꿔줘야함 ★

ex) background-color ⇒ backgroundColor

↳ 접근한 노드의 CSS속성을 직접 적용하거나 변경가능 노드.style.css = css 속성값

이벤트

1. 이벤트의 이해
   - ? 이벤트 : 웹 브라우저에서 웹 문서에 특별한 일이 있을 때 발생하는 신호
   - ? 이벤트 핸들러 : DOM 객체에 할당되어 해당 객체의 이벤트 반응에 호출되어 처리되는 프로퍼티
     이벤트 발생 감지 → 이벤트 감지 → 지정된 자바스크립트 또는 함수 호출
   - ? 이벤트 모델의 종류 (이벤트 핸들러 + 이벤트 API)
     - ? 기본 이벤트 모델 (original event model) ? DOM Level 0 이벤트 모델 (기술적으로 부족)
     - ? 표준 이벤트 모델 (standard event model) ? DOM Level 2 이벤트 모델
     - ? 인터넷익스플로러 이벤트 모델 (Internet Explorer event model) _ 기본 이벤트 모델과 함께 (더 이상 사용 X)
   - ? 기본 이벤트 모델의 이벤트 핸들러
     - ? 마우스 관련 : onclick, onmousedown, onmouseup, onmouseover, onmouseout
     - ? 로딩 관련 : onload, onunload
     - ? 서식 관련 : onsubmit, onreset, onfocus, onblur
     - ? 키보드 관련 : onkeydown, onkeypress, onkeyup
   - ? 기본 이벤트 모델 적용
     - ? HTML 요소에 속성으로 이벤트 핸들러 적용

       ```
       <a href="#" onclick="var i=3, j=5; calc=i*j; alert(calc);">3 x 5 의 값은?</a>
       <body onload="popup();">
       ```

     - ? 자바스크립트 DOM 객체 프로퍼티로 이벤트 핸들러와 호출함수 지정

       ```
       window.onload = init;   ← 괄호 붙이지 않음 ⇒ JavaScript 에선 함수 자체를 호출
       document.getElementById("next").onclick = goNext;
       ```

       DOM 객체의 property로 적용된 이벤트핸들러는

     - ? 이벤트 기본 기능 막기

       웹 브라우저에 기본으로 정의 되어 있는 이벤트 기능의 작동을 막을 필요성 있음

       기본 이벤트 모델에서는 이벤트 핸들러 호출 함수가 false를 리턴하면 기본 기능이 작동을 하지 않음

       ```
       <a href="second.html" onclick="alert('두 번째 페이지로 갈까요?'); return false;">Second Page</a>
       ```

       위 코드는 second.html 로 이동하지 않는다.

     - ? 이벤트 API

       이벤트 핸들러 핸들러는 호출함수에 함수 인자로 이벤트 객체를 전달한다.

       이벤트 객체를 전달받은 호출 함수는 다양한 이벤트 API를 이용하여 이벤트 정보를 얻을 수 있다.

       이벤트 발생 좌표 정보 : clientX, clientY, screenX, screenY,

       이벤트 발생 키보드 정보 : keyCode, altKey, shiftKey, type

2. 표준 이벤트 모델 (a.k.a DOM LEVEL 2 EVENT MODEL) 기본이벤트모델의 문제점 보완, 추가적인 API로를 포함하여 표준으로 공표
   - ? 특징
     - ? 표준화 된 기능 제공
     - ? 고급 이벤트 처리 방식
     - ? 복잡하고 정밀한 이벤트 전파 제어

     + 다양한 이벤트 API
   - ? 이벤트 핸들러 적용 (이벤트핸들러 / 전파되는이벤트감지)
     - ? window.addEventListener('load', init, false); ← 논리값

       이벤트를 감지하고자 하는 객체에 프로퍼티로 addEventListener를 적용

       실행될 함수

첫 번째 인자 'bad'는 이벤트 핸들러
기본 이벤트 모델의 이벤트 핸들러에 앞에 on을 제거하고 사용
두 번째 인자 'init'는 호출 함수
세 번째 인자 false는 이벤트 전파 방법

- 적용 예

```
document.getElementById("drag").addEventListener('mousedown', mouseDown, false);
document.getElementsByTagName("a")[0].addEventListener(click, linkClick, false);
```

- 이벤트 기본 기능 막기
  - 표준 이벤트 모델에서는 preventDefault(); 이벤트API 제공
  - function mouseDown (event) {

```
event.preventDefault();→ 이벤트 기본 기능 작동을 멈추게함
}
```

- 이벤트 API
  - 기본 이벤트 모델에 비해 많은 수의 이벤트 API 제공
  - 기본 이벤트 모델의 이벤트 API와 동일한 사용법
  - target : 이벤트 발생 객체

```
event.target.style.display = "none";
```

  - eventType : 발생한 이벤트타입
  - timeStamp : 밀리초 단위의 이벤트 발생 시간(날짜와 시간)
  - Button: 마우스 이벤트 발생시 눌려진 버튼.
    1은 왼쪽, 2은 가운데, 3은 오른쪽 버튼
- this : 이벤트 핸들러 호출 함수에서 사용되며 이벤트 발생 객체를 의미함
  - 이벤트 API인 target과 동일한 결과 event.target == this
  - 함수를 호출하는 기본 이벤트 모델과 표준 이벤트 모델에서 모두 사용 가능
  - 표준 이벤트 모델에서는 이벤트 전파로 인한 혼란으로 target을 사용하는 것을 권장함
- 적용된 이벤트 핸들러 함수의 제거
  - removeEventListener : 적용된 이벤트 핸들러 함수를 제거하는데 사용한다.
  - 이벤트 핸들러 호출 함수 적용 시 사용되었던 addEventListener와 동일한 인자로 구성된다.

```
document.getElementById("apple").addEventListener('click', bang, true);
document.getElementById("apple").removeEventListener('click', bang, true);
```

3. 이벤트 전파
   - 이벤트 전파의 이해
     - 이벤트는 발생한 객체에만 머물지 않고 이벤트 발생 객체를 포함하는 모든 상위 객체에 전파된다.
     - 이벤트 전파는 시각적으로 겹쳐 있는 객체에서 일어나는 것이 아니라 부모 자식 관계로 연결된 노드에서 발생하는 것.
   - 이벤트 전파의 3단계
     - 1단계 :document 노드 에서 이벤트 발생 노드까지 내려가면서 이벤트 전파
     - 2단계 : 이벤트 발생 노드
     - 3단계 : 이벤트 발생 노드에서 document 노드까지 올라가면서 이벤트 전파
   - addEventListener 이벤트 전파 조절
     - addEventListener의 세번째 인자를 참으로 해 놓으면 이벤트 전파의 전체 과정에서 이벤

트 핸들러가 이벤트를 감지한다 .?캡춰링 capturing 또는 cascade down 방향

? 거짓이면 2단계 3단계에서 이벤트 핸들러가 이벤트를 감지한다 . - 버블링 bubbling 또는 Bubble UP 방향

? 이벤트 전파 방지

? stopPropagation :이벤트 전파를 막는 이벤트 API

? 이벤트 전파에 관련된 이벤트 API

? currentTarget :이벤트 전파로 인해 현재 도달한 객체

? eventPhase: 세 단계의 이벤트 전파 과정 중 현 단계가 몇 단계인지 알려준다 .