1장

1.1 웹 플랫폼을 이루는 근간들

이 명세는 웹 플랫폼의 상당 부분을 매우 상세하게 정의합니다. 다른 명세들과 비교했을 때 웹 플랫폼 명세 스택에서 이 명세가 차지하는 위치는 다음과 같이 가장 잘 요약될 수 있습니다:
웹 플랫폼은 여러 가지 스펙으로 구성되며, 각각의 스펙은 특정한 기능과 역할을 합니다. 이 이미지는 특정 스펙이 다른 스펙들과의 관계를 어떻게 가지는지 보여줍니다.

상단 (프론트엔드 기술)

- **CSS**: 웹 페이지의 스타일과 레이아웃을 정의.
- **SVG**: 벡터 이미지를 표시.
- **MathML**: 수학 기호와 방정식을 표시.
- **Service Workers**: 백그라운드에서 서비스 작업을 수행.

중간 (프론트엔드와 기본 웹기술을 연결하는 기술들)

- **IDB (IndexedDB)**: 브라우저에서 대용량 데이터를 비동기적으로 저장하고 관리할 수 있는 데이터베이스.
- **Fetch**: 네트워크 요청을 간단하고 강력하게 처리할 수 있는 최신 API.
- **CSP (Content Security Policy)**: 웹 페이지에서 허용되는 콘텐츠의 출처를 제어하여 보안을 강화.
- **AV1**: 높은 압축률과 우수한 화질을 제공하는 비디오 코덱.
- **Opus**: 실시간 오디오 전송을 위한 고품질 오디오 코덱.
- **PNG**: 손실 없는 이미지 압축을 제공하는 이미지 포맷.

하단 (기본 웹 기술)

- **HTTP**: 클라이언트와 서버 간에 데이터를 전송하는 프로토콜.
- **TLS**: 인터넷 통신의 보안을 제공하는 프로토콜.
- **DOM**: HTML 및 XML 문서의 구조를 표현하는 인터페이스.
- **Unicode**: 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하는 문자 인코딩 표준.
- **Web IDL**: 웹 API를 정의하는 인터페이스 정의 언어.
- **MIME**: 인터넷에서 파일의 형식과 내용을 식별하는 표준.
- **URL**: 웹에서 자원의 위치를 지정하는 문자열.
- **XML**: 데이터 구조를 정의하고 전달하는 마크업 언어. `JSON으로 표준의 변화`
- **JavaScript**: 웹 페이지에 동적인 기능을 추가하는 프로그래밍 언어.
- **Encoding**: 데이터를 특정 형식으로 변환하는 과정.

1.2 위에 것은 HTML5 인가?

오피셜한 것은 아니지만 짧게 말하면 `YES`
HTML5란 해당 버전의 웹을 이루는 플랫폼의 근간의 스펙같은 것이다. 즉 HTML5에 DOM이 있고 HTML 태그가 있게 되는 것이다.
즉 HTML5란 것은 단순하게 HTML의 버전을 의미하는 것이 뿐만 아니라 유행처럼 그 버전의 웹 스펙(기술들)을 지칭하는 말이기도 하다.

1.3 배경

HTML은 월드 와이드 웹의 핵심 마크업 언어입니다. 원래 HTML은 주로 과학 문서를 의미론적으로 기술하기 위한 언어로 설계되었습니다. 그러나 그 일반적인 설계 덕분에 이후 몇 년 동안 다른 유형의 문서들과 심지어 애플리케이션까지도 기술할 수 있도록 적응해왔습니다.

1.4 대상 독자

이 명세는 다음과 같은 대상을 위해 작성되었습니다:

- 이 명세에 정의된 기능들을 사용하는 문서와 스크립트의 작성자
- 이 명세에 정의된 기능들을 사용하는 페이지를 다루는 도구의 구현자
- 이 명세의 요구사항에 따라 문서나 구현의 정확성을 확인하고자 하는 개인

이 문서는 웹 기술에 대해 최소한의 기본 지식이 없는 독자에게는 적합하지 않을 수 있습니다. 일부 내용에서는 명확성보다는 정확성을, 간결성보다는 완전성을 우선시하기 때문입니다. 더 쉽게 접근할 수 있는 튜토리얼과 저작 가이드가 이 주제에 대한 더 부드러운 소개를 제공할 수 있습니다.

특히, 이 명세의 더 기술적인 부분을 완전히 이해하려면 DOM의 기본에 대한 친숙함이 필요합니다. 또한 Web IDL, HTTP, XML, 유니코드, 문자 인코딩, JavaScript, CSS에 대한 이해도 곳곳에서 도움이 되겠지만 필수적이지는 않습니다.

1.5 범위

이 명세는 정적 문서부터 동적 애플리케이션에 이르기까지 웹상의 접근 가능한 페이지를 작성하기 위한 의미론적 수준의 마크업 언어와 관련된 의미론적 수준의 스크립팅 API를 제공하는 것으로 제한됩니다.

이 명세의 범위에는 미디어별 프레젠테이션 커스터마이징을 위한 메커니즘 제공은 포함되지 않습니다(웹 브라우저를 위한 기본 렌더링 규칙은 이 명세의 끝부분에 포함되어 있으며, CSS와 연결하기 위한 여러 메커니즘이 언어의 일부로 제공됩니다).

이 명세의 범위는 전체 운영 체제를 설명하는 것이 아닙니다. 특히 하드웨어 구성 소프트웨어, 이미지 조작 도구, 사용자가 고성능 워크스테이션에서 매일 사용할 것으로 예상되는 애플리케이션은 범위에서 벗어납니다. 애플리케이션 측면에서, 이 명세는 특히 사용자가 가끔 사용하거나, 정기적으로 사용하더라도 다양한 위치에서 낮은 CPU 요구사항으로 사용할 것으로 예상되는 애플리케이션을 대상으로 합니다. 이러한 애플리케이션의 예로는 온라인 구매 시스템, 검색 시스템, 게임(특히 다중 사용자 온라인 게임), 공공 전화번호부나 주소록, 통신 소프트웨어(이메일 클라이언트, 인스턴트 메시징 클라이언트, 토론 소프트웨어), 문서 편집 소프트웨어 등이 있습니다.

1.6 역사

처음 5년 동안(1990-1995), HTML은 여러 번의 개정과 확장을 거쳤으며, 주로 CERN에서 시작하여 나중에는 IETF에서 호스팅되었습니다.

W3C의 창설과 함께 HTML의 개발 장소가 다시 바뀌었습니다. 1995년에 HTML 3.0으로 알려진 첫 번째 실패한 HTML 확장 시도 이후, HTML 3.2라고 알려진 더 실용적인 접근 방식이 1997년에 완성되었습니다. HTML4가 같은 해 후반에 빠르게 뒤를 이었습니다.

다음 해, W3C 회원들은 HTML의 발전을 중단하고 대신 XHTML이라고 불리는 XML 기반의 대안 작업을 시작하기로 결정했습니다. 이 노력은 HTML4를 XML로 재구성한 XHTML 1.0으로 시작되었는데, 이는 새로운 직렬화를 제외하고는 새로운 기능을 추가하지 않았으며 2000년에 완성되었습니다. XHTML 1.0 이후, W3C의 초점은 XHTML 모듈화라는 기치 아래 다른 작업 그룹들이 XHTML을 더 쉽게 확장할 수 있도록 하는 것으로 바뀌었습니다. 이와 병행하여 W3C는 이전의 HTML 및 XHTML 언어와 호환되지 않는 새로운 언어인 XHTML2 작업도 진행했습니다.

1998년 HTML의 발전이 중단될 무렵, 브라우저 벤더들이 개발한 HTML용 API의 일부가 DOM Level 1(1998년), DOM Level 2 Core 및 DOM Level 2 HTML(2000년 시작하여 2003년 완성)이라는 이름으로 명세화되고 발표되었습니다. 이러한 노력들은 이후 점차 사그라들어, 2004년에 일부 DOM Level 3 명세가 발표되었지만 모든 Level 3 초안이 완성되기 전에 작업 그룹이 폐쇄되었습니다.

2003년, 웹 폼의 차세대 기술로 자리매김한 XForms의 발표는 HTML을 대체하는 것이 아니라 HTML 자체를 발전시키는 데 대한 관심을 다시 불러일으켰습니다. 이러한 관심은 XML의 웹 기술로서의 배포가 기존의 배포된 기술(HTML과 같은)을 대체하는 것이 아니라 완전히 새로운 기술(RSS와 나중의 Atom과 같은)에 국한된다는 인식에서 비롯되었습니다.

이렇게 새롭게 관심이 생긴 첫 번째 결과물은 기존 HTML 웹 페이지와 호환되지 않는 렌더링 엔진을 브라우저가 구현할 필요 없이, HTML4의 폼을 확장하여 XForms 1.0이 도입한 많은 기능을 제공할 수 있다는 것을 보여주는 개념 증명이었습니다. 이 초기 단계에서는 초안이 이미 공개적으로 이용 가능했고 모든 출처로부터 의견을 구하고 있었지만, 명세는 오페라 소프트웨어의 저작권 하에 있었습니다.

HTML의 발전을 재개해야 한다는 아이디어는 2004년 W3C 워크샵에서 테스트되었습니다. 이 워크샵에서 Mozilla와 Opera가 공동으로 W3C에 HTML5 작업의 기초가 되는 일부 원칙들(아래에서 설명)과 앞서 언급한 초기 초안 제안(폼 관련 기능만 다룸)을 발표했습니다. 이 제안은 웹 발전의 이전에 선택된 방향과 충돌한다는 이유로 거부되었습니다. W3C 직원들과 회원들은 대신 XML 기반 대체물을 계속 개발하기로 투표했습니다.

그 직후, Apple, Mozilla, Opera는 WHATWG라는 새로운 장소에서 이 노력을 계속하겠다는 의향을 공동으로 발표했습니다. 공개 메일링 리스트가 만들어졌고, 초안은 WHATWG 사이트로 옮겨졌습니다. 이후 저작권은 세 벤더가 공동으로 소유하도록 수정되었고, 명세의 재사용을 허용하게 되었습니다.

WHATWG는 몇 가지 핵심 원칙에 기반을 두고 있었습니다. 특히 기술은 후방 호환성이 있어야 하고, 명세와 구현이 일치해야 하며(이는 구현을 바꾸기보다는 명세를 바꾸는 것을 의미할 수 있음), 명세는 구현들이 서로를 역설계하지 않고도 완전한 상호운용성을 달성할 수 있을 만큼 상세해야 한다는 것이었습니다.

특히 후자의 요구사항은 HTML5 명세의 범위가 이전에 세 개의 별도 문서(HTML4, XHTML1, DOM2 HTML)에서 명세화되었던 내용을 포함해야 함을 의미했습니다. 또한 이전에 표준으로 여겨졌던 것보다 훨씬 더 상세한 내용을 포함해야 한다는 의미이기도 했습니다.

2006년, W3C는 결국 HTML5 개발에 참여하겠다는 의사를 표명했고, 2007년에는 WHATWG와 함께 HTML5 명세 개발 작업을 하도록 위임받은 작업 그룹을 구성했습니다. Apple, Mozilla, Opera는 W3C가 W3C 저작권으로 명세를 발행하도록 허용하면서도, WHATWG 사이트에는 덜 제한적인 라이선스의 버전을 유지했습니다.

수년 동안 두 그룹은 함께 작업했습니다. 그러나 2011년, 두 그룹은 서로 다른 목표를 가지고 있다는 결론에 도달했습니다: W3C는 "HTML5"의 "완성된" 버전을 발표하기를 원했고, WHATWG는 알려진 문제가 있는 상태로 고정하는 대신 계속해서 명세를 유지하고 필요에 따라 새로운 기능을 추가하여 플랫폼을 발전시키는 HTML의 Living Standard를 계속 작업하기를 원했습니다.

2019년, WHATWG와 W3C는 앞으로 단일 버전의 HTML에 대해 협력하기로 합의했습니다: 바로 이 문서입니다.

1.7 설계

HTML의 많은 측면이 처음 보기에는 비논리적이고 일관성이 없어 보인다는 점을 인정해야 합니다. 이는 HTML의 발전 과정과 밀접한 관련이 있습니다.

예를 들어, `<table>` 요소는 원래 데이터를 표 형식으로 표시하기 위해 설계되었지만, 한때 웹 페이지 레이아웃을 구성하는 데 널리 사용되었습니다. 이는 본래의 의도와는 다른 용도였지만, 당시의 기술적 한계로 인해 널리 채택되었습니다.

HTML, 이를 지원하는 DOM API, 그리고 많은 지원 기술들은 수십 년에 걸쳐 서로 다른 우선순위를 가진 광범위한 사람들에 의해 개발되었으며, 많은 경우 이들은 서로의 존재를 알지 못했습니다. 이로 인해 때로는 일관성 없는 기능들이 생겨났습니다.

예를 들어, HTML에서 이미지를 삽입할 때는 `<img>` 태그를 사용하지만, 비디오나 오디오를 삽입할 때는 각각 `<video>`와 `<audio>` 태그를 사용합니다. 이 태그들은 서로 다른 시기에 도입되어 약간 다른 구문을 가지고 있습니다.

```
<img src="image.jpg" alt="설명">
<video src="video.mp4" controls></video>
<audio src="audio.mp3" controls></audio>
```

더욱이 웹의 독특한 특성 때문에, 구현 버그들이 종종 사실상의, 그리고 이제는 법률상의 표준이 되었습니다. 이는 콘텐츠가 종종 의도치 않게 이러한 버그들에 의존하는 방식으로 작성되어 수정되기 전에 이미 널리 사용되었기 때문입니다.

예를 들어, 초기 버전의 Internet Explorer에서는 `<table>` 요소 내부의 공백을 무시하는 버그가 있었습니다. 많은 웹 개발자들이 이 버그에 의존하여 레이아웃을 만들었기 때문에, 다른 브라우저들도 이 동작을 모방해야 했습니다.

이러한 모든 상황에도 불구하고, 특정 설계 목표를 준수하려는 노력이 있었습니다. 이에 대해서는 다음 몇 개의 하위 섹션에서 설명합니다.

1.7.1 스크립트 실행의 직렬화 가능성

웹 작성자들이 멀티스레딩의 복잡성에 노출되는 것을 피하기 위해, HTML과 DOM API는 어떤 스크립트도 다른 스크립트의 동시 실행을 감지할 수 없도록 설계되었습니다.

예를 들어, 다음과 같은 코드에서는 두 스크립트가 순차적으로 실행됩니다:

```
<script>
        console.log("첫 번째 스크립트 시작");
        // 시간이 오래 걸리는 작업
        for(let i = 0; i < 1000000000; i++) {}
        console.log("첫 번째 스크립트 종료");
</script>
<script>
        console.log("두 번째 스크립트 실행");
</script>
```

이 경우, "두 번째 스크립트 실행"은 항상 "첫 번째 스크립트 종료" 후에 출력됩니다.

워커를 사용하더라도, 구현의 동작은 모든 전역에서 모든 스크립트의 실행을 완전히 직렬화하는 것으로 생각할 수 있도록 의도되었습니다.

이 일반적인 설계 원칙의 예외는 JavaScript의 `SharedArrayBuffer` 클래스입니다. `SharedArrayBuffer` 객체를 사용하면 실제로 다른 에이전트의 스크립트가 동시에 실행되고 있다는 것을 관찰할 수 있습니다. 더욱이 JavaScript 메모리 모델로 인해, 직렬화된 *스크립트* 실행을 통해 표현할 수 없을 뿐만 아니라 해당 스크립트들 간의 직렬화된 *문장* 실행을 통해서도 표현할 수 없는 상황들이 있습니다.

1.7.2 다른 명세와의 준수

이 명세는 다양한 다른 명세들과 상호작용하고 의존합니다. 예를 들어, HTML은 CSS, JavaScript, DOM API 등과 밀접하게 연관되어 있습니다.

불행히도 특정 상황에서는 상충되는 요구사항으로 인해 이 명세가 다른 명세들의 요구사항을 위반하게 되었습니다. 예를 들어, HTML의 `<iframe>` 요소는 보안상의 이유로 동일 출처 정책을 위반할 수 있습니다.

이런 일이 발생할 때마다, 각 위반 사항은 "**의도적 위반**"으로 표시되었으며, 위반 이유가 기록되었습니다.

1.7.3 확장성

HTML은 안전한 방식으로 의미를 추가하는 데 사용할 수 있는 광범위한 확장성 메커니즘을 가지고 있습니다:

1. `class` 속성을 사용한 확장: 작성자는 `class` 속성을 사용하여 요소를 확장할 수 있으며, 실질적으로 자신만의 요소를 만들 수 있습니다. 예제:

```
<div class="custom-card">
<h2 class="card-title">제목</h2>
<p class="card-content">내용</p>
</div>
```

2. `data-*` 속성 사용: 작성자는 `data-*=""` 속성을 사용하여 인라인 클라이언트 측 스크립트나 서버 측 사이트 전체 스크립트가 처리할 데이터를 포함할 수 있습니다.

```
    <article data-author="김철수" data-date="2023-07-30">
    <h1>기사 제목</h1>
    <p>기사 내용...</p>
    </article>
```

3. `<meta>` 태그를 이용한 메타데이터 포함: 작성자는 `<meta name="" content="">` 메커니즘을 사용하여 페이지 전체 메타데이터를 포함할 수 있습니다. 예제:

```
<meta name="description" content="이 페이지는 HTML의 확장성에 대해 설명합니다."> <meta name="keywords" content="HTML, 웹 개발, 확장성">`
```

4. `rel` 속성을 이용한 링크 의미 확장: 작성자는 `rel=""` 메커니즘을 사용하여 미리 정의된 링크 유형 집합에 확장을 등록함으로써 특정 의미로 링크에 주석을 달 수 있습니다. 예제:

```
<link rel="stylesheet" href="styles.css">
<a rel="nofollow" href="http://example.com"> 외부 링크 </a>
```

5. 사용자 정의 `<script>` 타입: 작성자는 사용자 정의 유형의 `<script type="">` 메커니즘을 사용하여 원시 데이터를 삽입하고, 이를 인라인 또는 서버 측 스크립트에서 추가로 처리할 수 있습니다. 예제:

```
<script type="application/json">
        { "name": "John Doe",  "age": 30 }
</script>
```

6. JavaScript 프로토타이핑: 작성자는 JavaScript 프로토타이핑 메커니즘을 사용하여 API를 확장할 수 있습니다. 예제:

```
HTMLElement.prototype.hide = function() {
        this.style.display = 'none';
        }; // 사용 document.getElementById('myElement').hide();`
```

7. 마이크로데이터: 작성자는 마이크로데이터 기능(`itemscope=""` 및 `itemprop=""` 속성)을 사용하여 다른 애플리케이션 및 사이트와 공유할 중첩된 이름-값 쌍의 데이터를 삽입할 수 있습니다. 예제:

```
<div itemscope itemtype="http://schema.org/Person">
        <span itemprop="name">Jane Doe</span>
        <span itemprop="jobTitle">Professor</span>
        <div itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
                <span itemprop="streetAddress">20 Main St</span>
                <span itemprop="addressLocality">San Francisco</span>
        </div>
</div>`
```

8. 사용자 정의 요소: 작성자는 사용자 정의 요소를 정의, 공유 및 사용하여 HTML의 어휘를 확장할 수 있습니다. 유효한 사용자 정의 요소 이름의 요구사항은 앞으로의 호환성을 보장합니다(앞으로 하이픈이 포함된 로컬 이름을 가진 요소가 HTML, SVG 또는 MathML에 추가되지 않을 것이기 때문입니다). 예제:

```
<script>
        class PopUpInfo extends HTMLElement {
                constructor() {
                        super();    // 요소의 기능 정의
                        }
                }
        customElements.define('popup-info', PopUpInfo);
</script> <!-- 사용 -->

<popup-info></popup-info>`
```

이러한 확장성 메커니즘들은 HTML을 더욱 유연하고 강력하게 만들어, 개발자들이 웹의 진화하는 요구사항에 맞춰 새로운 기능을 구현할 수 있게 해줍니다.

1.8 HTML 대 XML 구문

이 명세는 웹 문서와 애플리케이션을 설명하기 위한 추상적인 언어와, 이 언어를 사용하는 리소스의 메모리 내 표현 및 이와 상호작용하기 위한 API를 정의합니다.

1. DOM (Document Object Model): 메모리 내 표현은 "DOM HTML" 또는 줄여서 "DOM"이라고 알려져 있습니다. DOM은 문서의 구조화된 표현이며, 프로그래밍 언어가 문서 구조, 스타일, 내용 등을 변경할 수 있게 해주는 인터페이스를 제공합니다. 예시:

```
 // DOM을 사용하여 새로운 요소 생성 및 추가
let newDiv = document.createElement("div");
newDiv.innerHTML = "Hello, DOM!";
document.body.appendChild(newDiv);
```

2. HTML 구문: HTML 구문은 대부분의 웹 개발자에게 권장되는 형식입니다. 이는 대부분의 레거시 웹 브라우저와 호환되며, `text/html` MIME 타입으로 전송됩니다. 예시:

```
<!DOCTYPE html>
<html lang="ko">
        <head>
                <meta charset="UTF-8">
                <title>HTML 예시</title>
        </head>
        <body>
                <h1>안녕하세요, HTML!</h1>
                <p>이것은
                        <strong>HTML</strong>
                구문의 예시입니다.
                </p>
        </body>
</html>
```

```
HTML 구문의 특징:
- 태그 이름은 대소문자를 구분하지 않습니다 (예: `<div>` 또는 `<DIV>`).
- 일부 요소는 종료 태그가 선택적입니다 (예: `<p>텍스트</p>` 또는 `<p>텍스트`).
- 속성 값은 따옴표 없이 사용할 수 있습니다 (예: `<input type=text>`).
```

3. XML 구문: XML 구문은 더 엄격한 규칙을 가지고 있으며, `application/xhtml+xml`과 같은 XML MIME 타입으로 전송됩니다. XML 프로세서에 의해 파싱되며, 사소한 구문 오류조차도 문서 렌더링을 방해할 수 있습니다. 예시:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko">
        <head>
                <title>XML 예시</title>
        </head>
        <body>
                <h1>안녕하세요, XML!</h1>
                <p>이것은
                        <strong>XML</strong>
                        구문의 예시입니다.
                </p>
        </body>
</html>
```

```
XML 구문의 특징:
- 모든 태그는 대소문자를 구분합니다 (예: `<div>`만 유효, `<DIV>`는 유효하지 않음).
- 모든 요소는 종료 태그가 필요합니다 (예: `<p>텍스트</p>`).
- 모든 속성 값은 따옴표로 묶어야 합니다 (예: `<input type="text">`).
```

DOM, HTML 구문, XML 구문은 각각 표현할 수 있는 내용에 차이가 있습니다:

결론적으로, HTML과 XML은 각각 고유한 특징과 용도를 가지고 있습니다. HTML은 더 유연하고 오류에 관대하여 웹 개발에 널리 사용되는 반면, XML은 더 엄격하고 구조화된 데이터 교환에 적합합니다. DOM은 이러한 문서 구조를 프로그래밍 방식으로 조작할 수 있게 해주는 인터페이스를 제공합니다.