책에서도 알려주지 않는, 반응형 웹의 비밀!

작성: 2014. 04. 06. 수정: 2014. 04. 08.(3판)
분류: 웹표준, 참고자료
작성: 고남현(Naver ID - isc7981)

- 개요

이 글에서는 일반 시중에 팔리고 있는 웹 개발 서적에서 다루지 않는, '반응형 웹에 대한 역사'를 풀이하는 글이다. 이전의 고정형 웹과 최근의 반응형 웹이 어떠한 관계에 있는지 알아본다. 최근에 이야기되는 반응형 웹의 근간이 어디서 왔는지를 이해하면, 앞으로의 심화 과정을 학습하는데도 큰 도움이 될 것이다.

- 반응형 웹 왜 떴나?

반응형 웹의 근간은 '모바일 퍼스트'에 있다. 모바일을 가장 첫 번째로 염두에 두고 구현하는 방식을 의미한다. 다양한 화면 크기를 가진 스마트폰, 태블릿들이 등장하면서 고정형 레이아웃(Fixed Layout)이 주를 이루고 있던 이전의 웹사이트로는 이용자들의 만족을 충족하기 어려웠다. 다양한 기기를 만족할 수 없던 이전의 웹사이트와는 달리 모바일용 웹 사이트는 소형 기기는 물론 일반 개인용 컴퓨터까지 대응할 수 있었다.

이런 점을 이용하여 웹사이트를 PC용과 모바일용을 구분할 필요 없이 만드는 방법. 반응형 웹(Responsive Web)이 알려지기 시작했다.

반응형 웹이 막 알려진 시기에는, 필요한 요구사항을 온전히 충족하는 웹 브라우저의 부재가 가장 큰 문제였다. 이러한 부재를 해결하기 위해 이전의 기술로 유사하게 구현을 하기 위한 시도가 있었으나, 반응형 웹에 대한 인식이 부족한데다가 개발 시간이 오래 걸리고 브라우저 간 차이도 심했다. 이러한 이유로 관심이 있는 일부 사람들 사이에서 개인적인 연구로 끝나는 경우가 대부분이여서 일반 웹사이트에서 대중화되기란 쉽지 않았다.

이러한 상황은 주요 브라우저가 HTML5와 CSS3에 대한 지원을 강화하면서 반전된다. 특히 반응형 웹 활성화의 핵심은 CSS3의 지원 강화에 있다. 이에 따라 반응형 웹 구현에 필요한 요구사항의 지원이 강화되면서 속속 반응형으로 제작된 웹 사이트들이 생겨났다. 이러한 상황에 맞춰 반응형 웹에 대한 관심도가 증가하고, 스마트폰과 태블릿의 보급률이 거의 정점을 찍으면서 웹 세상은 반응형의 시대가 도래하였다고 할 수 있다.

- 반응형 웹 시기 이전의 레이아웃 구현방법

반응형 웹(Responsive Web)이 본격적으로 대두되기 전, CSS2 시기에는 레이아웃을 다루던 문제로 고정 방식과 가변 방식이 있었다. 최근에 이야기하는 반응형 웹은 시각적으로 보면 가변 방식과 유사하다 볼 수 있다. 하지만 그 근간을 고정 방식에 두고있는 경우도 많아 처음 접근하는데 많은 어려움을 겪을 수 있다는 것이 필자의 생각이다. 우리는 현대적인 반응형 웹을 익히기 전에 CSS2 시기에 쓰였던 고정 방식과 가변 방식에 대해 익힐 필요가 있다.

또한, 고정형 방식에서 쓰이던 내용들을 소개하고 이것이 반응형 방식과 어떻게 연계가 되어 있는지 살펴보고 이 시기에 등장하기 시작해서 현재의 반응형 웹 구현의 근간이 되고있는 모바일 웹 개발의 기초 내용에 대해서도 알아보도록 하겠다.

이전의 고정형 웹의 개발에 쓰이던 방법은 고정형 레이아웃, 가변형 레이아웃, 고정형 CSS 그리드 시스템이 있다. 이 글에서 다룬 고정형과 가변형 레이아웃의 기준은 NHN 웹표준팀에서 운영하는 NULI(html.nhncorp.com)에서 제시한 모델을 기준으로 삼았다. 이것 말고도 다양한 모델이 있으니 검색을 통해 찾아보기를 바란다.

 # 고정형 레이아웃

고정형 레이아웃은 고정된 폭(width)를 두고 그 안에 요소를 놓는 레이아웃 구성 방식이다. 레이아웃의 칼럼의 수에 따라 1단 레이아웃, 2단 레이아웃, 3단 레이아웃이라 부르기도 한다. 여기서는 고정형 3단 레이아웃을 기준으로 설명하도록 하겠다.

<그림 1> 고정폭 3단 레이아웃

아래는 해당 레이아웃을 구성하는 CSS이다. 필자와 함께 이 소스를 분석해보도록 하자.

```
1 body{_text-align:center}
2 #wrap{width:600px;margin:0 auto;_text-align:left}
3 #header{width:100%}
4 #container{*display:inline-block;width:100%}
5 #container:after{display:block;clear:both;content:''}
6 .snb{float:left;width:180px;margin-right:20px}
7 #content{float:left;width:200px}
8 .aside{float:right;width:180px}
9 #footer{width:100%}
```

▶ CSS Hack의 사용

적용된 소스를 잘 보면 일반적인 CSS와 달리 _text-align:center와 *display:inline-block;처럼 일반적인 CSS 속성에 언더바(_) 또는 별(*)이 붙은 부분이 있다. 이는 인터넷 익스플로러(IE)의 버전 차이에서 오는 문제를 해결하기 위한 것이다. 언더바가 붙은 핵은 IE5.5~6에 영향을 미치고 별이 붙은 것은 IE6~7에 영향을 미친다. CSS Hack은 IE 말고도 다른 브라우저에 적용되는 여러 가지 형태가 있다. CSS Hack은 꼭 필요한 상황에서만 쓰도록 하고 남용하는

것은 권장하지 않는다.

▶ 떠있는(Float) 객체의 클리어 문제

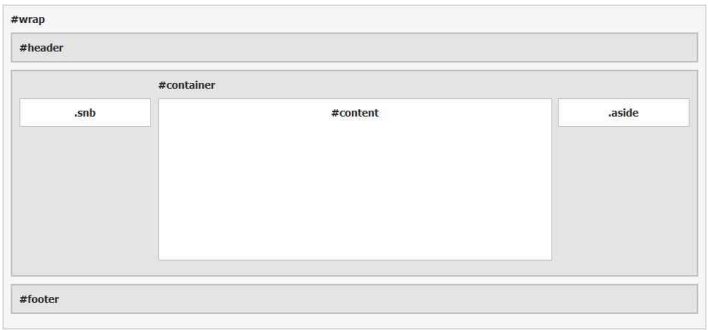
CSS의 Float 속성은 말 그대로 떠있는 상태를 만들어주는 속성이다. 그래서 float: left;는 왼쪽에 떠있다, float: right;는 오른쪽에 떠있다고 해석할 수 있다. 하지만 떠있다보니 하단의 #footer의 위치가 제대로 잡히지 못하고 좌우 요소(.snb와 .aside)와 겹쳐 보이는 상황이 발생한다. 떠있는 요소 아래로 들어가는 상태인 것이다. 이를 해결하기 위해 5번 행과 같은 내용이 적용된 것이다. 이런 방식 외에 #container에 overflow: auto; 속성을 주거나 #footer에 clear: both; 속성, 또는 #container와 #footer 사이에 별도의 .clear 요소를 넣어주는 방법이 있다. 아래는 별도의 .clear 요소에 적용된 CSS 예제이다.

```
.clear { clear: both; display: block; float: none; font-size: 0 !important; height: 0; line-height: 0 !important; margin: 0 !important; overflow: hidden; padding: 0 !important; width: 100%; }
```

이제 가변형 레이아웃으로 넘어가 더 많은 내용을 확인해 볼 것이다. 고정형 및 가변형 레이아웃을 모두 익히면 반응형 레이아웃을 익히는데 도움이 될 것이다.

 # 가변형 레이아웃

가변형 레이아웃은 폭(width)이 브라우저 크기에 따라 바뀌는 방식을 의미하며, 그 안의 요소도 원하는 요소에 한해 부모 요소의 폭에 따라 바뀌도록 설정하는 것이다. 넓은 의미에서 고전적인 반응형 웹의 형태라 볼 수 있다. 여기선 가변폭 3단 레이아웃을 기준으로 설명하겠다.



<그림 2> 가변폭 3단 레이아웃

아래는 해당 레이아웃을 구성하는 CSS이다.

```
1 #wrap{width:100%}
2 #header{width:100%}
3 #container{*display:inline-block;_width /**/:100%;padding-right:200px;padding-left:200px}
4 #container:after{display:block;clear:both;content:''}
5 .snb{float:left;position:relative;left:-200px;width:180px;margin-right:-180px}
6 #content{float:left;width:100%}
7 .aside{float:left;position:relative;left:200px;width:182px;margin-left:-182px}
8 #footer{width:100%}
```

이전 고정형 레이아웃에서 설명한 내용은 생략하겠다. 이 레이아웃은 좌우에 각각 있는 .snb

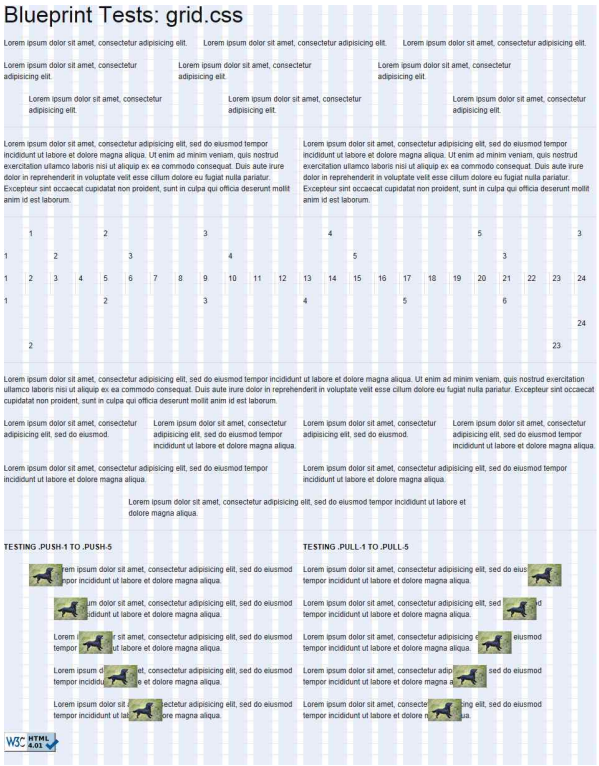
와 .aside는 고정 너비를 가지고, #content는 브라우저에서 너비에 따라 같이 변하는 형태를 가진다. #container에서 좌우 양단에 padding(안쪽 여백)을 200px를 주면서 공간을 확보한다. 좌우의 각 요소에 정의된 속성들을 보면 position: relative;가 적용되어 있는데 이는 위치를 상대적으로 이용하겠다는 뜻으로 현재 위치를 기준점으로 left: -200px;과 right: -200px;를 사용 가능하게 해준다. 위치를 잡았지만 브라우저는 아직 해당 요소의 자취를 그대로 가지고 있어 원하는 위치에 놓이지 않게 된다. 이를 margin-right: -180px;과 margin-left: -182px; 속성을 이용하여 잡아주게 된다.

 # 고정형 CSS 그리드 시스템

그리드를 이용하면 웹 페이지를 일정한 기준의 크기로 잘라 레이아웃 구성이 가능하기 때문에 구현하기가 용이해진다는 장점이 있다. 국내 웹 개발 환경은 레이아웃 구성 방식이 테이블(표)을 쓰느냐 DIV(Division)을 쓰느냐로 이분화 되어있어 국내에는 많이 알려지지 않았다.

이 중 테이블 방식이 가장 그리드 시스템과 유사하다 볼 수 있어 그 편리함 때문에 그동안 많이 이용해왔다. 하지만 일정한 크기로 나누는 것이 아니기 때문에 그리드 시스템이라 부르진 않는다. DIV를 이용한 구성은 구성하고 있는 요소 중에 하나라도 기준 위치가 바뀌면 전체가 틀어지기 때문에 불편함을 느끼기도 한다.

CSS 그리드 시스템은 일정한 크기로 나눈 레이아웃 구조를 제시함으로써 이러한 불편을 해소해준다. 이러한 CSS 그리드 시스템은 대표적으로 BluePrintCSS: A CSS Framework가 있다. 이 글에서 우리는 BluePrintCSS의 그리드 시스템을 살펴보도록 하겠다.



<그림 3> BluePrintCSS 그리드 시스템

아래는 BluePrintCSS의 그리드 시스템을 이용하여 3-칼럼 레이아웃을 생성하는 예제이다.

```
<div class="container">
<div class="span-24">The header</div>
<div class="span-4">The first column</div>
<div class="span-16">The center column</div>
<div class="span-4 last">The last column</div>
<div class="span-24">The footer</div>
</div>
```

한 칸(Grid) 당 30px의 크기가 주어진다면, 좌측 칼럼이 4*30=120px가 되어야 할 것처럼 보이지만 그렇지 않다. 좌측 칼럼의 너비는 150px이다. 30px 차이는 어디서 발생한 것일까? 그리드 시스템을 파헤쳐보면 아래와 같은 내용이 있다.

```
.column, .span-1, .span-2, (생략), .span-24 {
        float: left;
        margin-right: 10px;
}
```

한 그리드 당 10px 씩 오른쪽 바깥 여백이 있다. 이러한 여백을 Gutter라고 한다. 이러한 내용을 토대로 span-n에 따른 너비를 계산하면, width=n*30+10*(n-1) 식이 성립할 것이다. 이 식에 따라 계산하면 전체 너비는 950px, 중앙 칼럼은 630px, 우측 칼럼은 120px가 될 것이다. 여기서 last는 옆으로 나열하는 요소가 끝나서 다음 행의 그리드로 넘어가야한다는 뜻이다. 더 자세한 내용을 알고 싶다면 www.blueprintcss.org을 참고하도록 하자.

 # 모바일 웹의 태동기 - 웹 표준과 접근성부터 모바일 웹 까지

모바일 웹의 태동은 스마트폰과 태블릿의 보급 확산과 웹 표준과 접근성에 뿌리를 두고 있다. 국내에서 스마트 기기 보급이 본격적으로 시작될 즈음, 한편에서는 웹에 대한 시각을 달리 보는 시도가 본격화되고 있었다. HTML과 CSS를 W3C의 표준에 맞게 웹을 개발하는 '웹 표준', 그리고 모두에게 평등한 웹을 제공하자는 '웹 접근성'이다.

간혹 우리나라에선 개정된 장애인차별금지법에 의해 강제된 것이 아니냐는 반론도 있다. 하지만 처음 웹 표준과 접근성에 관심을 둔 소수의 사람들은 사회적 차별보다는 개발 과정의 이로움과 기술적 문제를 해결하기 위한 방안에 초점을 맞췄다. 우리나라 웹 표준과 접근성의 시작은 다양한 브라우저가 하나의 웹사이트를 두고 각기 다른 모습을 보인다는 기술적 문제를 해결하기 위한 것이었다. 이것을 크로스브라우징(Cross-browsing)이라 따로 이야기하기도 한다. 특정 플랫폼에 의존적인 우리나라 웹의 특성을 고려해봤을 때 어떤 이유에서 시작된 것인지는 더 명확해진다 할 수 있다. 하지만, 우리나라에서 웹 접근성을 준수하는 개발 방식의 대중화는 개발자 사이에서도 부정적으로 보는 시각이 많았고 결국 정부 주도로 법이 강제함으로써 이뤄졌기 때문에, 기술을 깊게 생각하지 못하고 단순 찍어내기에 급급한 우리나라 IT의 씁쓸한 단면을 보여주기도 한다.

이러한 웹 브라우저 간 차이를 해결하기 위해 웹 표준으로 제시된 사항을 지키면서 개발을 하는 개발 방법이 시도가 된다. 완벽한 해결책은 아니지만 효과가 있었다. 물론 W3C가 제안하

는 표준을 개발 시 선언하지 않는 쿼크 모드(Quirks Mode)에서도 브라우저 간 차이를 맞출 수 있었으나, 브라우저에 따른 영향이 표준 모드(Standard Mode) 보다 심하고 차후 유지보수가 원활하지 못했기 때문에 이전에는 임시방편으로 쓰이다가 이후에는 특별한 경우를 제외하고는 사용하지 않게 된다.

웹 표준 개발 방식을 따르게 되면서 웹 개발 시 작성한 소스가 얼마나 읽기 좋은지(가독성)에도 점차 신경을 쓰게 되었다. 또한, 유연한 개발에도 역시 관심이 높아졌는데 브라우저에서 웹을 이루는 3요소인 HTML, CSS, Javascript의 분리였다. 처음에는 하나의 웹 페이지 안에서 기능적으로 분리하는 '점진적 향상(Progressive Enhancement)' 방식으로 가다가 점차 파일까지 분리하는 완전한 분리의 형태인 '겸손한 구현(Unobtrusive)' 방식에 까지 이른다.

이러한 방식은 웹 접근성을 고려하여 개발할 시, 웹 페이지 내의 모든 스크립트와 스타일 시트를 배제하고서라도 해당 사이트를 이용할 수 있도록 하는 방안에서 시작된 것이다. 한 때 이러한 개발 방식을 두고 '비스크립트(non-script) 환경에 대응하도록 개발을 한다.'라는 말을 쓰기도 했다. 일반 웹과 마찬가지로 현재의 모바일 웹이나 반응형 웹에서도 스크립트 요소는 웹 사이트의 기능을 강화하는 중요한 역할을 하고 있지만, 반대로 이러한 웹 사이트의 기반에는 스크립트가 배제된 환경에 대응하기 위한 방법이 깔려있다. 서로 극과 극이 다른 상황에서 나온 개념이 오히려 서로에게 도움을 주고 있는 것이다.

각기 다른 요소를 분리시키는 웹 개발 방식을 이용함으로써, 파일 전체를 수정하지 않고도 CSS 수정만으로도 다양한 모양으로 변형시키거나 Javascript 수정만으로도 원하는 기능을 추가시키는 것이 가능하게 된다. 이러한 특징은 차후 모바일 웹 개발에 큰 영향을 미치게 된다.

이러한 변화는 웹 접근성과 목적은 다르지만 유사한 특징을 가진 SEO(검색엔진최적화)에 대한 관심으로 이어졌다. 두 주제는 사용자 편의를 증대한다는 측면은 동일하지만, SEO는 말 그대로 웹 사이트가 가진 내용이 검색 엔진에 잘 노출될 수 있게 최적화 시킨다는 목적이 있고, 웹 접근성은 모든 사람에게 평등한 웹을 제공하는데 목적이 있다는 차이가 있다. SEO에 대한 관심 증가는 장애인도 차별 없이 웹을 이용한다는 취지를 담은 법적인 의미의 '웹 접근성'에 대한 관심 증가에 영향을 미치게 된다. 웹 접근성에 대한 관심 증가는 웹 접근성을 다룬 법 제정에도 영향을 미치게 된다.

위 내용을 모두 이해했다면 모바일용 페이지는 큰 문제가 없어진다. 보통 모바일용 페이지는 PC용 웹과는 별개로 만들어지는 것이 일반적인데 이는 기존에 PC를 기준으로 개발된 내용으로는 모바일에 그대로 대응할 수 없기 때문이다. 하지만, 필자는 PC용 웹에서 웹 접근성을 준수한 기존 코드를 그대로 가지고 모바일 웹에 적용할 수 있었던 경험이 있다.

즉 개발 프로세스상 따로 진행되는건 어쩔 수 없다하여도, 모바일 웹의 작성은 웹 접근성에 그 기본이 있다고 말할 수 있다. 이 글에 나온 이전의 내용을 모두 알고 있다면, 모바일 웹에서 가장 중요한 내용은 단 한 줄로 끝난다. 나머지는 개발자 각자의 역량이 맞기면 된다.

아래는 모바일 웹 페이지의 head 태그 안에 넣는 viewport meta 태그 내용이다.

```
<meta name="viewport" content="width=device-width, height=device-height, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, target-densitydpi=device-dpi" />
```

각각의 속성에 대한 풀어보면 아래와 같다.

속성	설명
width=device-width	너비를 기기의 해상도 너비에 맞춤.
height=device-height	높이를 기기의 해상도 높이에 맞춤.
user-scalable=no	웹페이지 폭 맞춤(스케일링) 제한.
initial-scale=1.0	
maximum-scale=1.0	
minimum-scale=1.0	
target-densitydpi=device-dpi	DPI(Dots per Inch)에 맞춤. 안드로이드만 적용.

참고로 알아두어야 할 내용을 하나 이야기하면, 글자의 크기를 지정할 때는 고정 형태인 픽셀(px)이나 포인트(pt) 대신 상대적 크기를 가진 em이나 % 단위를 쓰는 것을 권장한다. 1em은 100%와 같으며 상위 요소에서 지정된 글자 크기와 같음을 의미한다. 즉, 글자 크기를 지정할 때 고정 단위는 * { font-size: 12px; }와 같이 지정하는 것이 가능하지만, 가변 단위는 * { font-size: .75em; }과 같이 지정하면 안 되고, 하나의 상위 요소를 잡아서 지정해야한다. 그렇지 않으면 자식 요소로 갈수록 75% 만큼 글자 크기가 작아진다.

 # 브라우저 별로 다른 스타일을 적용

이후 내용에서 나오겠지만 CSS3 미디어 쿼리를 주요 브라우저가 원활하게 지원하기 전, 따로 이름이 붙지 않은 유사한 방식이 쓰이기도 했다. 환경에 상관없이 공통적으로 적용되는 스타일 시트와 함께, 접속하는 브라우저나 기기의 종류를 확인하여 각기 다른 스타일 시트를 불러오는 방법이었다. 이러한 방법은 서버에서 확인하는 방법과 사용자 웹 브라우저와 같은 클라이언트 환경에서 확인하는 방법으로 나뉜다.

먼저 클라이언트에서 확인하는 방법에 대해 알아보겠다. 해당 방법의 대표적인 예는 인터넷 익스플로러(IE)의 버전 차이를 맞추기 위해 쓰이는 '조건부 주석(Conditional Comments)'이 있다. 이 방식은 CSS만 제어하는 것이 아닌 브라우저에 따라 다른 Javascript 파일을 불러오거나 특정한 HTML 코드 출력을 제어할 수 있다. 이런 점 때문에 지금도 다양한 종류의 웹을 개발하면서 브라우저 차이로 인한 문제를 해결하기 위해 쓰이고 있다. 아래는 조건부 주석을 이용하여 html5shiv(구형 브라우저에서 HTML5 태그 지원) 스크립트를 불러오는 예제이다.

```
<!--[if lt IE 9]>
<script src="dist/html5shiv.js"></script>
<![endif]-->
```

여기서 if lt IE9의 의미는 IE9 보다 아래 버전임을 의미한다. 여기에 들어갈 수 있는 조건은 gt(선택보다 큰), lt(선택보다 작은), gte(선택보다 같거나 큰), lte(선택보다 같거나 작은), !(느낌표, 선택 이외의)가 있다. 조건부 주석은 IE10부터 표준 모드에서는 지원하지 않는다.

클라이언트에서 확인하는 또 다른 방법으로는 자바스크립트를 이용한 방법이 있다. 결과에 따라 어떤 내용을 보여줄지를 스크립트가 정하도록 할 수 있다. 아래는 자바스크립트로 구현된 클라이언트에서 확인하는 방식의 간단한 예제이다.

```
if( /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(navigator.userAgent) ) {
    alert("Your device is mobile,");
} else {
    alert("Your device is not mobile,");
}
```

▶ 여기서 잠깐! HTML5/CSS3 하위 호환을 위한 라이브러리를 알아보자.

반응형 웹 활성화의 주역은 HTML5가 아닌 CSS3이다. 하지만 웹의 발전 흐름 상 이 둘은 같이 발전하고 있기 때문에 HTML5 역시 무시해서는 안 되는 부분이다. 브라우저 별로 HTML5와 CSS3 지원가능 사항이 많은 차이를 보이고 있는데, 최근의 브라우저가 아니라면 이러한 차이는 더 심해진다. 이러한 차이를 해결하기 위한 주요 라이브러리를 소개하겠다.

1) HTML5Shiv: HTML5에서 지원하는 태그의 하위 호환에 필요한 스크립트이다. 이 스크립트는 https://code.google.com/p/html5shiv/ 에서 받을 수 있다.
2) Modernizr: 브라우저가 HTML5/CSS3의 어떠한 사양을 지원할 수 있는지 확인하는 스크립트이다. 이 스크립트는 http://modernizr.com/ 에서 받을 수 있다.
3) CSS3-MediaQueries.js: 하위 브라우저에서 CSS3의 미디어 쿼리를 사용 가능하게 해주는 스크립트이다. 이 스크립트는 http://code.google.com/p/css3-mediaqueries-js/ 에서 받을 수 있다.

이제부터 알아볼 내용은 서버에서 확인하는 방법이다. 서버에서 확인하는 방법은 주로 모바일 기기로 웹사이트에 접속했을 때 모바일 웹 페이지로 유도하기 위한 방법으로 많이 쓰였다. 하지만 예외는 있었는데, 기존의 HTML 구조는 그대로 놔두고 기기별로 다른 스타일 시트를 불러와 적용하는 용도로 사용되기도 했다. 이는 조건에 따라 다른 스타일을 주는 반응형 웹과 유사한 방식이었지만 이 당시에는 반응형 웹이라는 용어를 사용하지 않았다. 그 이후에 이러한 방식은 반응형 웹에서 최적화(Optimization)이라는 주제로 등장하게 된다. 아래는 서버에서 확인하는 방식의 PHP로 작성된 간단한 예제이다.

```
$mobile_agent = '/(iPod|iPhone|Android|BlackBerry|SymbianOS|SCH-M\d+|Opera Mini|Windows
CE|Nokia|SonyEricsson|webOS|PalmOS)/';
if(preg_match($mobile_agent, $_SERVER['HTTP_USER_AGENT'])) {
    echo 'Your device is mobile.';
} else {
    echo 'Your device is not mobile.';
}
```

- 반응형 웹의 본격적인 시작!

고정형 웹 시기에 이룬 성과를 바탕으로 반응형 웹이 등장하기 시작한다. 반응형 웹 활성화의 일등공신은 CSS의 미디어 쿼리(Media Query)이다. 조건에 따라 다른 CSS를 적용하는 것이 가능하도록 만들어주기 때문에 기기의 너비라는 조건의 변화에 대응해야하는 반응형 웹의 특성에 가장 알맞은 기능이라 할 수 있겠다. 이는 CSS2부터 존재했던 기능이지만 이전에는 all, screen, print와 같은 단순한 조건에 한해 이용되었다. 하지만 CSS3에 들어 기능이 향상되면

서 더 구체적인 조건 지정이 가능해졌다. 아래는 브라우저별 미디어 쿼리 지원 현황이며, 자세한 현황은 http://caniuse.com/css-mediaqueries 에서 확인할 수 있다.

Can I use CSS3 Media Queries? View in interactive mode

Compatibility table for support of CSS3 Media Queries in desktop and mobile browsers.

= Supported = Not supported = Partially supported = Support unknown

CSS3 Media Queries - **Recommendation**

Method of applying styles based on media information. Includes things like page and device dimensions

Resources: WebPlatform Docs | IE demo page with information | Media Queries tutorial | Polyfill for IE

Global user stats*:
Support: 86.92%
Partial support: 0.01%
Total: 86.93%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
29 versions back			4.0										
28 versions back		2.0	5.0										
27 versions back		3.0	6.0										
26 versions back		3.5	7.0										
25 versions back		3.6	8.0										
24 versions back		4.0	9.0										
23 versions back		5.0	10.0										
22 versions back		6.0	11.0										
21 versions back		7.0	12.0										
20 versions back		8.0	13.0										
19 versions back		9.0	14.0										
18 versions back		10.0	15.0										
17 versions back		11.0	16.0										
16 versions back		12.0	17.0		9.0								
15 versions back		13.0	18.0		9.5-9.6								
14 versions back		14.0	19.0		10.0-10.1								
13 versions back		15.0	20.0		10.5								
12 versions back		16.0	21.0		10.6								
11 versions back		17.0	22.0		11.0								
10 versions back		18.0	23.0		11.1								
9 versions back		19.0	24.0		11.5								
8 versions back		20.0	25.0		11.6								
7 versions back		21.0	26.0	3.1	12.0			2.1					
6 versions back	5.5	22.0	27.0	3.2	12.1			2.2		10.0			
5 versions back	6.0	23.0	28.0	4.0	15.0	3.2		2.3		11.0			
4 versions back	7.0	24.0	29.0	5.0	16.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	25.0	30.0	5.1	17.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	26.0	31.0	6.0	18.0	5.0-5.1		4.1		12.0			
Previous version	10.0	27.0	32.0	6.1	19.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	28.0	33.0	7.0	20.0	7.0	5.0-7.0	4.4	10.0	16.0	33.0	26.0	10.0
Near future		29.0	34.0		21.0								
Farther future		30.0	35.0		22.0								
3 versions ahead		31.0	36.0										

Note: Incomplete support by older webkit browsers refers to only acknowledging different media rules on page reload

<그림 4> 미디어 쿼리 지원 현황표

 # 고정형 레이아웃, 반응형으로 변신하다!

미디어 쿼리의 조건 지정이 어떤 것인지 알고 싶다면, 기존의 고정형 레이아웃을 반응형으로 바꿔버릴 수 있다면 이해가 빠르게 될 것이다. 앞서 보았던 고정형 레이아웃의 모델을 그대로 반응형으로 바꿔보도록 하겠다. 무슨 말을 하는지 모르겠다 하시는 분은 이전의 고정형 레이아웃 부분을 다시 읽고 오기를 바란다.

바꾸는 방법은 CSS 안에 아래 코드 하나 넣어주면 곧바로 반응형 레이아웃이 된다.

```
@media screen and (max-width: 480px) {
        #wrap { width: 90%; }
        .snb { width: 100%; }
        #content { width: 100%; }
        .aside { width: 100%; }
}
```

이 코드의 기능은 브라우저의 너비가 480px 아래가 되면 3단 레이아웃이 1단 레이아웃으로 변형되도록 하는 것이다. 작동을 확인하고 싶다면 직접 해보길 바란다. 여기서 screen은 미디어 타입(media type)이고 and 이후에 들어간 내용은 조건문이다. 아래는 조건문에 들어갈 수 있는 속성들에 대한 표이다.

기능	설명
width	기기의 너비.
height	기기의 높이.
device-width	기기의 너비.
device-height	기기의 높이.
orientation	기기의 회전 상태를 의미. portrait, landscape 사용.
aspect-ratio	기기 화면의 비율. 1은 1:1로 가로 세로의 길이가 같음을 의미.
device-aspect-ratio	기기 화면의 비율.
color	단말기에서 사용하는 최대 색상 2^n개 비트 수.
color-index	단말기에서 사용하는 최대 색상 수.
monochrome	흑백 사이에 2^n 단계의 회색을 가진 기기.
resolution	해상도
scan	TV 출력 장치의 검색 프로세스를 의미.
grid	0: 고정된 크기의 글자만 가진 기기. 1: 일반적인 기기.

미디어 쿼리에서 조건문은 위 기능에 max-, min-을 붙여 쓰게된다. 이는 대소비교와 같은 조건을 사용할 수 없기 때문이다. 아래는 미디어 타입으로 쓸 수 있는 내용들이다.

타입	설명
all	모든 미디어 타입
aural	음성 합성 장치
braille	점자 표시 장치
handheld	손으로 들고 다니면서 볼 수 있는 작은 스크린에 대응하는 용도
print	인쇄 용도
projection	프로젝터 표현 용도
screen	컴퓨터 스크린을 위한 용도
tty	디스플레이 능력이 한정된, 텔렉스(teletype), 터미날, 또는 수동 이동 장치 등, 고정 피치(fixed-pitch:폭이 일정) 글자를 사용하는 메디아를 위한 의도 “tty” 메디아 타입에서 제작자는 픽셀(pixel) 단위를 사용하여서는 안됨
tv	음성과 영상이 동시 출력되는 TV와 같은 장치
embossed	페이지에 인쇄된 점자 표지 장치

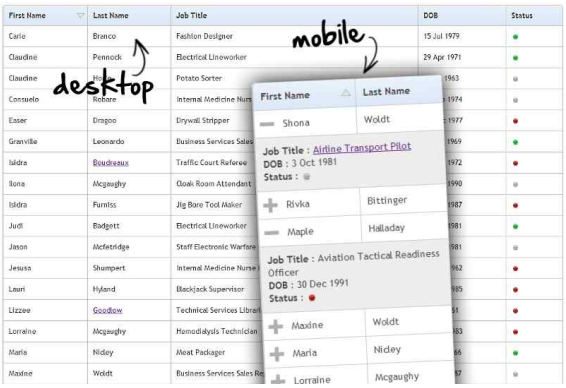
미디어 쿼리는 위에서 봤던 고정형 레이아웃을 반응형으로 바꾸는 예제처럼 CSS 안에서 쓰는 방법, 스타일(style) 태그에 media 속성을 주는 방법, link 태그나 CSS의 @import처럼 다른 스타일 시트 파일을 불러올 때 media 속성을 주거나 명시하는 방법이 있다.

 # 부속 요소들을 어떻게 처리하지?

웹 페이지를 반응형으로 만들다보면 대응이 되지 않는 요소가 있다. 바로 변하는 너비만큼 줄일 수 없는 요소들이다. 대표적으로는 테이블(표)가 있다. 열의 개수가 일정 수준 이상을 넘어가면 아무리 줄이려고 해도 줄일 수가 없다. 이러한 문제를 해결하기 위한 방법은 크게 세 가지로 볼 수 있는데, 일부 내용을 감추거나, 테이블의 구조를 아예 변형시키거나, 구조 변경을 최소화하고 스크롤을 제공하는 방법이다.

이 중 일부 내용을 감추는 방법은 너비에 맞춰 표의 열(칼럼)을 줄이는 방식으로 이해하기 가

장 쉬운 방식이다. 이 글에선 테이블을 변형하는 방법과, 스크롤을 제공하는 방법의 대표적인 예를 하나씩 살펴보도록 하겠다. 아래는 FooTable에서 제안한 반응형 테이블 모델이다.



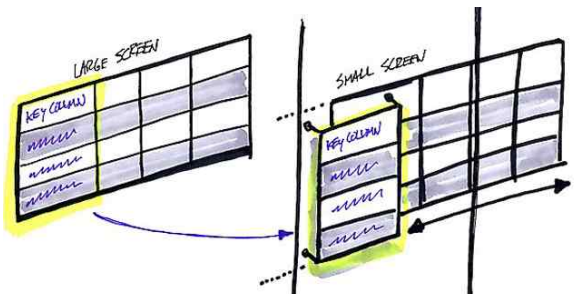
<그림 5> FooTable 반응형 테이블

테이블을 완전히 변형하는 형태를 가지고 있다. 해당 반응형 테이블을 제안하고 만든 개발자는 아래의 다양한 반응형 테이블 방식에서 영향을 받았다고 한다.

제안자	해결책	사이트
Zurb	소형 기기에서 가로 스크롤을 제공한다.	zurb.com
Dave Bushell	테이블을 사이드로 플립(제목의 행렬전환)한다.	dbushell.com
Filament Groups	보여줄 행을 사용자가 직접 선택한다.	filamentgroup.com
Stewart Curry	덜 중요한 열(칼럼)을 감춘다.	irishstu.com
Chris Coyier	행 마다 제목을 반복한다.	css-tricks.com

* FooTable은 http://themergency.com/footable/ 에서 내려받을 수 있다.

위에 제시된 반응형 테이블 방식 중, 스크롤 제공 방식의 대표적인 예시인 Zurb에 대해 알아보겠다. 아래는 Zurb에서 제시한 반응형 테이블 모델이다.



<그림 6> Zrub 반응형 테이블 모델

첫 번째 열(칼럼)은 고정시키고 나머지 내용에 대해 가로 스크롤을 제공하는 형태이다. 제목 열이 되는 첫 번째 열이 항상 같은 위치에 고정되어 스크롤하면서도 표를 읽기가 수월해진다. 표에 스크롤을 제공하는 방법은 간단하게 생각하면 표 전체가 스크롤 되는걸 생각할 수 있는데, 제목이 가려지기 때문에 내용에 대한 분간이 힘들 수 있다. 해당 라이브러리는 이 점을 해결했다 보면 된다.

 # 반응형 CSS 그리드 시스템

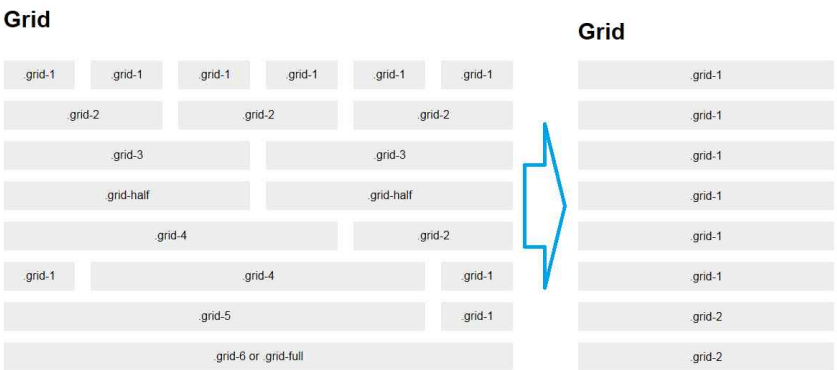
우리나라에선 빛을 보지 못했던 고정형 그리드와는 달리 반응형 그리드는 어떤 결과를 가져올지는 좀 더 지켜봐야 한다. 하지만 그리드를 적용할 명분은 이전보다는 많아졌다 생각한다.

반응형 그리드는 여러 가지 형태가 있다. 이전 예제에서 고정형 레이아웃에 반응형을 적용한 것처럼 기존의 고정형 그리드에 반응형으로 그리드 고정 폭을 조정하는 방법이 있다. 그리드 크기를 가변적으로 조정해 나가다가 어느 조건이 되면 변형되도록 한 반응형 방식도 있다. 이는 이전의 고정형 웹에서 지정된 가변형 레이아웃에 반응형 레이아웃을 적용하는 것과 같다 보면 된다. 이 말을 이해하기 어렵다면 간단하게 이해하자. 반응형은 조건에 따르는 것이다.

다양한 반응형 그리드 중 이해를 돕기 위해 가장 간단한 형태로 구현된 그리드 시스템을 소개하도록 하겠다. 가변형과 반응형을 합쳐놓은 형태의 Foldy960이다. 아래는 Foldy960의 그리드 시스템이 적용된 예제이다. (Foldy960 - https://github.com/davatron5000/Foldy960)

```
@media screen and (min-width: 480px) {
    .grid-1, .grid-2, .grid3, (생략), .grid-unit {
        float: left; width:96.969696969697%; margin:0 1.515151515152% 1em;
    }
@media screen and (min-width: 640px) {
    .grid-1 { width: 13.636363636364%; }
    .grid-2 { width: 30.30303030303%; }
    .grid-3, .grid-half { width: 46.969696969697%; }
}
```

일부 내용은 다 다룰 수 없어 생략하겠다. 화면 너비가 480px을 넘어가면 그리드를 세로로 풀어놓고, 640px를 넘으면 그리드가 위치에 맞게 재배열되도록 구성되어있다. 결과적으로는 480px 아래에서도 그리드들이 아래로 풀리는 상태가 만들어진다. 그리드의 너비를 고정 단위가 아닌 %나 em 등의 가변 단위를 쓰고 반응형을 적용한 형태이다. 알아두어야 할 것은 단위를 고정으로 써도 반응형은 적용 가능하다는 것이다. 반응형은 조건에 따라 스타일의 변화를 주는 것이기 때문에 반드시 가변 단위를 써야하는 것은 아니다. 아래는 방금 전의 예제를 실행한 결과이다.



<그림 7> Foldy960 그리드 시스템

 # 잊혀져가던 방식의 화려한 부활: Optimization

이전의 CSS3 미디어 쿼리 지원이 열악하던 때에도, 웹사이트를 PC용과 모바일을 각각 따로 두지 않고도 두 환경을 동시에 대응하도록 하는 시도가 있었다. 서버 측에서 사용자의 접속 환경을 확인하여 기기에 맞춰진 각기 다른 CSS를 불러오게끔 하여 가능한 일이였다. 이후에 미디어 쿼리로 조건 지정이 가능해지면서 이러한 시도는 자취를 감추는 듯 했다. 하지만 반응형 웹에서 이러한 방식은 최적화(Optimization)이라는 이름으로 다시 등장하게 된다.

이전의 서버에서 적용할 스타일 시트를 결정하던 방식은 모바일은 mobile.css, PC는 main.css와 같이 각기 다른 파일을 불러오도록 하는 것이었다. 이후 미디어 쿼리로 인해 이러한 방식은 사용할 필요가 없어졌는데 차후 문제점이 발견되었다. 한 번에 불러오는 스타일 시트가 mobile.css과 main.css의 특성을 동시에 가지고 있었던 것이다.

이는 모바일에서 접속했을 때에도 PC용 스타일 시트까지 함께 불러들여짐을 의미한다. 사실상 사용하지 않는 CSS를 불러오는 것이기 때문에 트래픽이 낭비되는 것이다. 이러한 문제점을 해결하기 위해 기기에 따라 사용하지 않는 스타일 시트를 배제하는 최적화(Optimization) 방식이 알려지게 되었다.

이러한 방식은 단순히 트래픽 해소로만 끝나지 않고, 기기에 따른 적절한 해상도의 동영상이나 이미지를 선택하는 방법으로도 쓰일 수 있다. 기기에 맞는 적절한 컨텐츠를 선택하여 보여주기 때문에 더 효과적인 반응형 웹 구현이 가능해진다.

- 다루지 못했던 내용들

1) 스크립트로 이루어져있으나 접근성을 향상시키기 여의치 않은 부분에 대해선 noscript 태그를 이용하여 접근성을 향상시키곤 하는데, 이러한 방식을 '적절한 낮춤(Graceful Degradation)'이라 부른다.

- 마치며

지금까지 웹 반응형의 역사에 대해 살펴보았습니다. 더 다양한 내용은 검색을 통해 더 많이 찾아보셔서 학습하시길 바랍니다. 고정형 웹부터 지금의 반응형 웹으로 넘어가는 과정 사이에 어떠한 일들이 있었는지를 중점적으로 서술하려다보니 여러 주제가 겹쳐있다는 느낌을 받을 수 있습니다. 특히 웹 표준과 접근성을 강조하는 모습을 많이 보였는데, 비록 이 글의 주제가 반응형 웹이지만 강조했던 다른 개념들도 검색을 통해 좀 더 많이 찾아보시기를 권장드립니다. 긴 글 읽어주셔서 감사합니다.

- 질문있습니다!

이 문서는 처음 작성 후 3번의 수정을 거쳤습니다. 여기에선 수정을 거치면서 받은 질문들을 풀어보는 시간을 가져보도록 하겠습니다.

Q. 반응형 웹의 근간이 반드시 '모바일 퍼스트'라고 할 수 있나요?

정확하게는 반응형 웹이라는 개념이 나온 배경이 모바일 퍼스트라고 할 수 있습니다. 이 문서

에서 적용된 예시 중에는 고정형 레이아웃을 반응형으로 바꾸는 예시가 있는데, 기본으로 삼는 레이아웃이 모바일용이 아닌 일반 PC용 웹에서 쓰는 레이아웃이기 때문에 모바일 퍼스트를 기반으로 한 반응형 웹의 개발이었다고 말하기 힘듭니다. 이 문서에서 소개된 반응형 그리드 역시 모바일 웹 보다는 그리드에 기본이 있기 때문에 이것이 모바일 퍼스트를 기초로 했다고 보는 것은 각자 개개인의 생각에 따라 다르게 볼 수 있습니다. 하지만, 여기서는 반응형 웹이 나오게 된 시작이 어디에 있는가를 중점적으로 이야기했기 때문에 '반응형 웹은 모바일 퍼스트에 근간이 있다.'라는 말을 쓴 겁니다.

Q. 이미 시중에 반응형 웹을 다루는 책이 나와 있는데, 이 문서는 뭐가 다른거죠?
이미 '마치며' 부분에서도 밝혔듯이 고정형 웹에서 반응형 웹으로 넘어가는 과정에서 있었던 일들을 중점적으로 다루었습니다. 특히 해외가 아닌 우리나라 환경에서 접할 수 있었던 내용들로 구성하였습니다. 반응형 웹에서 쓰이는 기술 위주로 서술하기보다, 기존의 고정형 웹과 반응형 웹의 연관성을 중점적으로 서술하였다는 차이가 있습니다.

- 변경사항
2014. 04. 06. - 첫 문서 작성
2014. 04. 07. - '마치며' 부분의 수정, 최적화(Optimization) 내용 추가.
2014. 04. 08. - 애매했던 부분을 더 구체적으로 수정.

- 참고자료

1) NHN 웹표준팀 NULI UIO Factory - http://html.nhncorp.com/uio_factory
2) BluePrint Wiki - https://github.com/joshuaclayton/blueprint-css/wiki
3) Can I use CSS3 media queries? - http://caniuse.com/css-mediaqueries
4) Media Queries, W3C Recommendation 19 June 2012
http://www.w3.org/TR/css3-mediaqueries/
5) CSS3 media query에 대하여 - http://nuli.navercorp.com/blog/42284
6) FooTable: a jQuery Plugin for Responsive Data Tables
http://css-tricks.com/footable-a-jquery-plugin-for-responsive-data-tables/
7) A New Take on Responsive Tables
http://zurb.com/article/982/a-new-take-on-responsive-tables
8) Foldy960 - https://github.com/davatron5000/Foldy960
9) About conditional comments(Internet Explorer) - MSDN
http://msdn.microsoft.com/en-us/library/ms537512(v=vs.85).aspx
10) What is the best way to detect a handheld device in jQuery?
http://stackoverflow.com/questions/3514784

- 유용한 사이트
1) 웹 접근성 연구소 - http://www.wah.or.kr/
2) W3C Markup Validation Service - http://validator.w3.org/
3) QuirksMode - http://www.quirksmode.org/
4) CSS-Tricks - http://css-tricks.com/