HTML5와 오픈소스 기반의 Web Components 기술

신정규

래블업
lablup.com

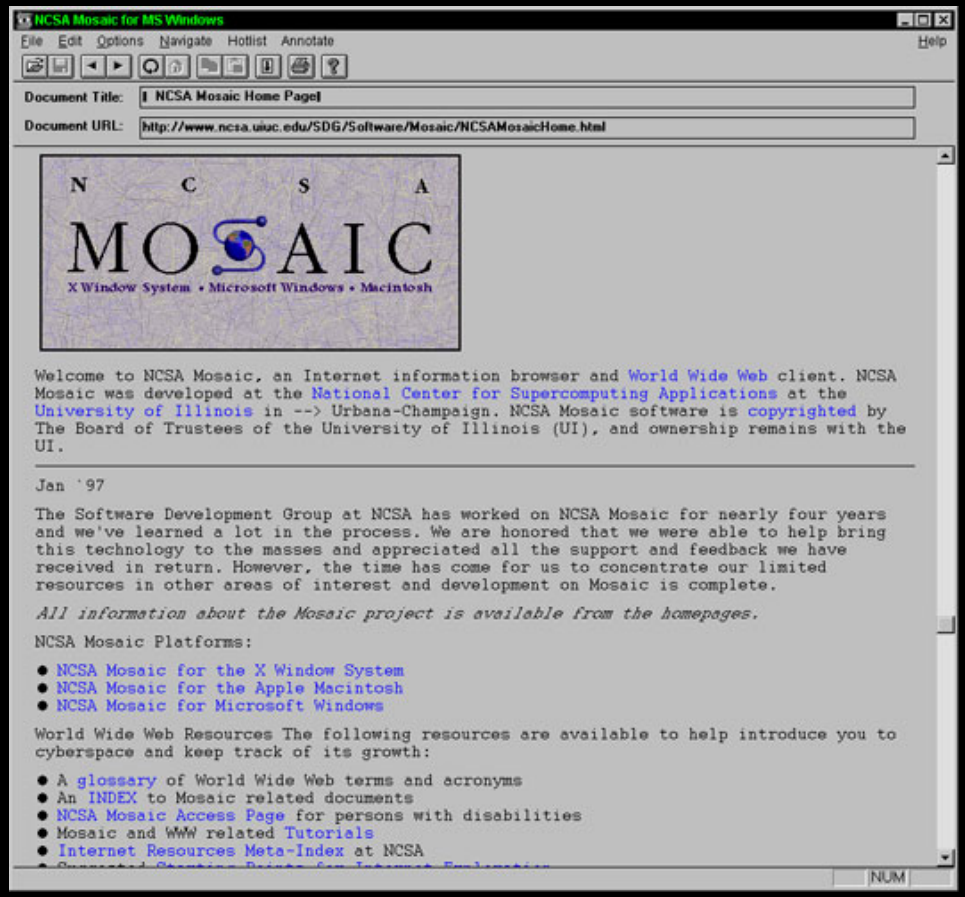
발표자

- 신정규
  - 복잡계 이론 물리학자
    - 통계물리학 / 뇌과학
  - 텍스트큐브 개발
    - 오픈소스 블로그 소프트웨어 / 서비스
    - TNF/ Needlworks
  - Automotive security
  - 래블업
    - 코딩 교육 시스템 / 연구 지원 시스템 개발중
  - 기타 등등...

한마디로 요약하면 카오스죠
확실한건 우리 아기는 귀엽습니다.

오늘의 주제

Web Components



태초에 HTML이 있었다

브라우저 전쟁

- Netscape 4 *versus* Internet Explorer 3/4

웹 중세시대

Firefox

- XHTML 1 / CSS 2
- “철학” – 왜 우리에게 대안이 필요한가?

크롬이 크롬크롬

- 구글 기어로 고생한 후 그들의 접근
- 렌더링 엔진을 구분하기 시작하다
  - Webkit + V8 (by Google Inc.)

V8!

V8!

구.. 구글님께서 날 보셨어

V8!

V8!

모바일

- iPhone OS 1.0: 손안에 들어오는 데스크탑 급의 브라우저
- “웹앱은 애플리케이션을 대체할 정도로 충분한 잠재력을 지니고 있다.” – 스티브 잡스

...물론 대박은 2.0에서 네이티브 앱용 앱스토어를 넣어서 났습니다만.

근본적인 질문이 폭발하다:

웹이 뭐냐?

HTML5

IT 전쟁의 새로운 전장

웹앱

- 긴 역사가 있습니다
  - Scriptacul.us
  - Dojo
  - Prototype.js
  - jQuery
  - Angular.js
  - Flux
- 대한민국
  - 웹 (위의) 앱!
  - Active X!
  - 아니 왜 우리 애를 기를 죽이고 그래요!

문제의식

- 웹은 문서인가 런타임 플랫폼인가?
  - Mozilla의 접근
  - Google의 접근
  - Apple의 접근
  - Facebook의 접근
- 충돌
  - WHATWG (web hypertext application technology working group)

충돌: 종교 전쟁의 시작

- W3C
  - Next HTML specification
- WHATWG (web Hypertext application technology working group)
  - Web application 1.0 (2004. 7)

- 2007 ~ 2012
  - HTML5 로 리브랜딩 및 규격 확정 작업에 동의
  - HTML5에 대한 양 집단의 “정의 (definition)” 가 다르다

More recently, the goals of the W3C and the WHATWG on the HTML front have diverged a bit as well. The WHATWG effort is focused on developing the canonical description of HTML and related technologies, meaning fixing bugs as we find them adding new features as they become necessary and viable, and generally tracking implementations. The W3C effort, meanwhile, is now focused on creating a snapshot developed according to the venerable W3C process. This led to the chairs of the W3C HTML working group and myself deciding to split the work into two, with a different person responsible for editing the W3C HTML5, canvas, and microdata specifications than is editing the WHATWG specification.

Ian Hickson (WHATWG editor)

불안한 동거 (지만 가내 별거)

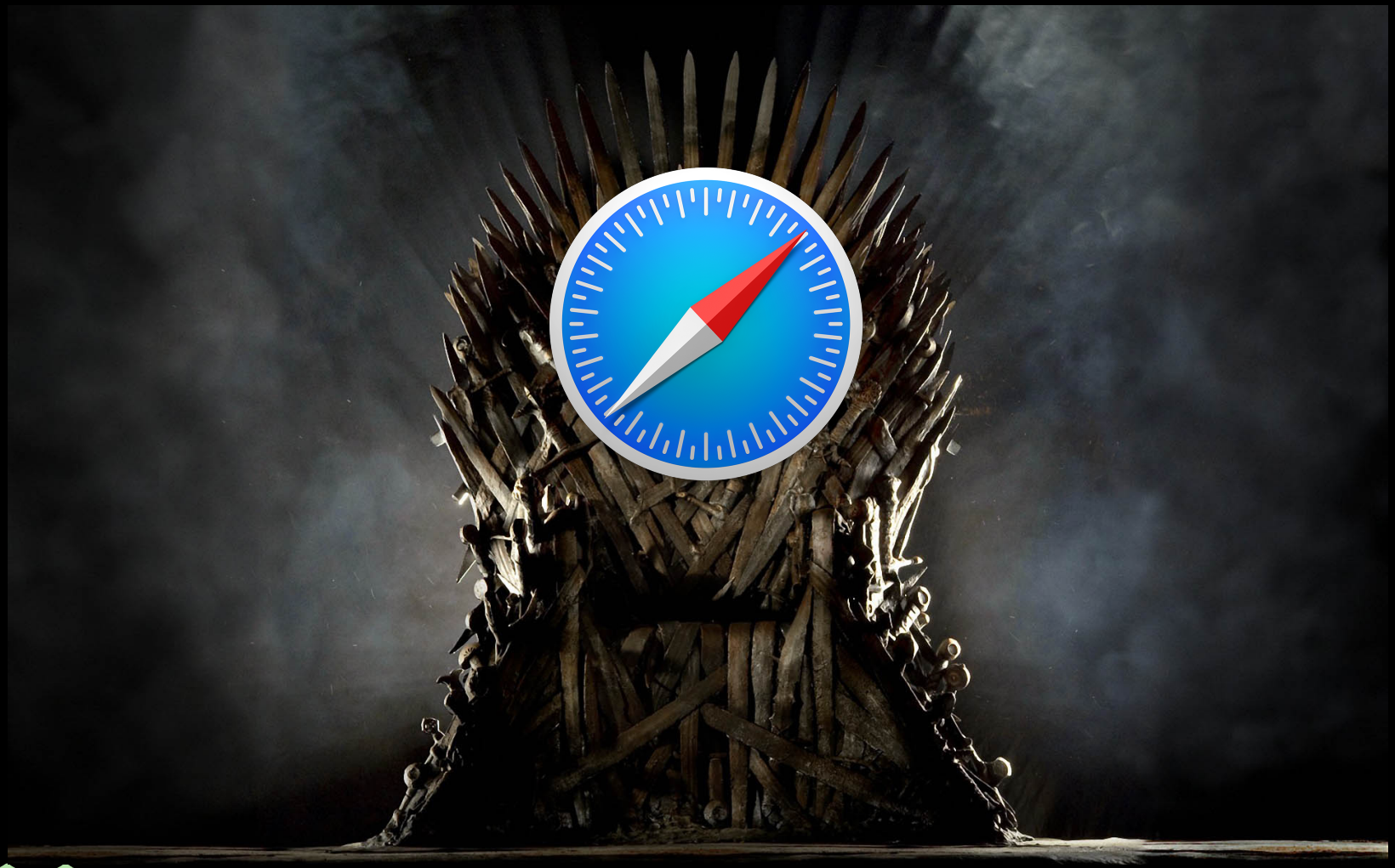
- 문서다 (W3C)
  - 고정된, 한정된 형식과 포맷
  - 영속성에 중점
  - 앱은 앱으로 만들지 왜 웹 위에 만드냐?

- 플랫폼이다 (WHATWG)
  - 변화하고 발전하는 규약
  - 확장성 및 진화에 중점
  - 너희는 우리 규격의 스냅샷일 뿐

현재

- (좋은/나쁜놈1) 기업이 브라우저를 갖고 있음
  - Chrome / Edge
  - 새 규격을 계속 새 브라우저에 부어 넣음
- (좋은/나쁜놈1) 기업 외 브라우저
  - Firefox
  - 규격을 추가하고 있긴 하지만 여전히 의문 제기
  - XUL 기반의 웹 플랫폼을 말아먹은 전력이 있음
- 이상한 놈
  - Safari
  - 기업이 주도하는 브라우저인데 엔진을 공개해 놓음
  - W3C만 지원할까?+독자 명령셋 (--webkit-) 지원

- 세상은 이상한 놈이 바꾼다...?



Web Components

HTML5 앱의 기어박스

플랫폼이라 칩시다

- (WHATWG의 HTML5에 추가된) 플랫폼적 요소들
  - HTML Import
    - 외부 HTML을 참조할 수 있음
  - Shadow DOM
    - HTML의 서브셋이 독립된 DOM을 가질 수 있음
  - Custom Elements
    - 브라우저 기본 DOM의 leaf object /class 를 개발자가 임의로 만들 수 있음
  - Templates
    - 템플릿을 만들고 디자인할 수 있음

HTML Import

- 임의의 HTML 파일을 CSS 불러오듯 import 할 수 있음
- 반복되는 코드 블럭 조각을 불러오는데도 쓸 수가 있지만, custom element 등을 불러오는 용도가 적당합니다.

```
<link rel="import" href="name-card.html">
```

Shadow DOM

- DOM Tree 하위의 어떤 leaf 엘리먼트 안에 독립된 DOM Tree를 만들 수 있음
- Root DOM (우리가 일반적으로 생각하는 DOM) 에서 보이지 않는다고 해서 Shadow DOM
  - 예) Root DOM와 Shadow DOM 내의 같은 id가 중복되어도 상관이 없음

```
this.createShadowRoot().appendChild(new_element);
```

Custom Element

- DOM에서 HTML5 규격에 미리 정의된 object type 이외의 타입을 생성하고 사용할 수 있음
  - 예) , <a> 태그처럼 <name-card> 라는 custom element를 만들어 사용할 수 있습니다.

```
document.registerElement('name-card', {
        prototype: prototype
});
var ncElement = document.createElement('name-card');
```

HTML Template

- HTML에서 디자인 요소를 분리해 낸 것이 CSS
- 구조를 분리해 낸 것이 Template

```
<style>
    .card {
      width : 200px;
      height: 35px;
      border-radius: 3px;
    }
</style>
<template id="namecard-root">
  <div class="card">
    <h2>Name : <span>{{ name }}</span></h2>
  </div>
</template>
```

HTML imports!

Reusable web components!

Shadow DOM!

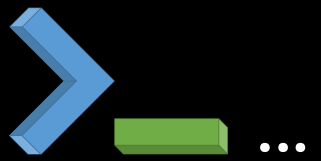
Polyfill!

본 적이 없어요

- 실제 구현 사례?



저희도 CodeOn (https://codeonweb.com) 에서 구현하고 있습...



Webcomponents.js

일단 자리를 폅시다

http://webcomponents.org

Webcomponents.js

- “Polyfill”
  - HTML5 webcomponents 스펙을 완전히 지원하지 않는 브라우저들을 위한 대체 구현
- Webcomponents.js
  - Custom elements / HTML imports / Shadow DOM
  - Mutation observers
    - DOM mutation 을 추적하고 효율적으로 실행함
  - ES6 Weakmap type
    - Key 를 지정하지 않은 k-v Map

브라우저 지원

Polyfill	IE10	IE11+	Chrome*	Firefox*	Safari 7+*	Chrome Android*	Mobile Safari*
Custom Elements	~	✓	✓	✓	✓	✓	✓
HTML Imports	~	✓	✓	✓	✓	✓	✓
Shadow DOM	✓	✓	✓	✓	✓	✓	✓
Templates	✓	✓	✓	✓	✓	✓	✓

- Webcomponents.js : 모든 polyfill 지원
  - 115kbytes
- Webcomponents-lite.js : Shadow DOM 지원 제외
  - 38kbytes

준비

- 코드 한 줄 추가!
- <script src="webcomponents-lite.min.js"></script>
- 이제 본격적으로 떠나볼까요?

Polymer

http://polymer-project.org

구글의 대답

질문도 구글이 했다는게 함정

Polymer library

- HTML5 컴포넌트 기반 라이브러리
  - HTML imports / Custom elements / Shadow DOM
- Web components에 기반함
  - “Polyfill” 을 지원 : 브라우저 호환 레이어
- 재사용 가능한 컴포넌트 개발
  - ‘DOMelements’ 라는 이름 사용
  - 원래 이름으로는 표준화하기 좀 그랬던듯
    - (PolymerElement)

Polymer : 요약

- Polyfill library + MORE
  - 브라우저가 안 되는 부분을 메꿔줍니다.
- Template + style + code 로 딱딱 나누어짐
- Two-way binding 지원 (느림..)
- 컴포넌트 상속 지원
  - 정확히는 상속이라기보다는 메소드 바인딩이라고 보는 것이 타당함
- 굉장히 직관적인 구조

예: 패널

- 동기
  - 코드 안에 패널을 보여주는 부분의 반복이 너무 많다.
- 사양
  - 링크, 제목, 요약, 커버 이미지 경로

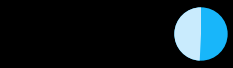
스타일

이건 바로 뒤에서 소개하겠습니다.

```
<link rel="import" href="../polymer/polymer.html">
<link rel="import" href="../lablup-piechart/lablup-piechart.html">
<dom-module id="lablup-course-item">
        <style>
            paper-card {
                width: 280px;
                margin: 15px 20px;
                --paper-card-header-image-text: {
                    width: 100%;
                    color: #eee;
                    padding-left: 10px;
                    padding-right: 0;
                    text-shadow: 0 0 4px #444, 0 0 8px #000;
                    font-weight: 400;
                    overflow-x: hidden;
                };
            }

            .card-actions-item {
                padding-right: 15px;
            }

            .card-actions-item span {
                display: block;
                height: 25px;
                line-height: 25px;
            }
        </style>
```

템플릿

```
<template>
      <a href="{{ url }}">
          <paper-card heading="{{ title }}"
                      image="{{ coverUrl }}">
              <div class="card-content">{{ summary }}</div>
              <div class="card-actions horizontal left layout">
                  <section class="vertical center layout card-actions-item">
                      <lablup-piechart url="" number="{{ lectureCount }}"
                           maxnumber="20” chartcolor="#29B6F6" unit="" size="40">
                      </lablup-piechart>
                      <span>{{ messageLectures }}</span>
                  </section>
                  <section class="vertical center layout card-actions-item">
                      <lablup-piechart url="" number="{{ memberCount }}"
                           maxnumber="100” chartcolor="#cddc39" unit="" size="40">
                      </lablup-piechart>
                      <span class="black">{{ messageMembers }}</span>
                  </section>
              </div>
          </paper-card>
      </a>
  </template>
```



코드


```
<script>
    Polymer({
        is: "lablup-course-item",
        properties: {
            title: {
                type: String
            },
            summary: {
                type: String
            },
            url: {
                type: String
            },
            coverUrl: {
                type: String
            },
            messageMembers: {
                type: String
            },
            lectureCount: {
                type: Number
            },
            memberCount: {
                type: Number
            }
        },
        ready: function () {
        }
    });
</script>
</dom-module>
```

사용

HTML template의 경우:

```
<lablup-course-item url="/course/view/HelloWorld"
       title=”Hello world!”
       summary=”About basic grammar”
       cover-url=”img/helloworld.png”
       lecture-count=”3”
       member-count=”12”
       message-members="Members“
       message-lectures="Lectures">
</lablup-course-item>
```

사용

Django HTML template의 경우:

```
<lablup-course-item url="/course/view/{{ course.id }}"
        title="{{ course.title }}”
        summary="{{ course.summary }}”
        cover-url="{{ cover.url }}”
        lecture-count="{{ course.entries.count }}”
        member-count="{{ course.users.count }}”
        message-members="{% trans "Members" %}“
        message-lectures="{% trans "Lectures" %}">
</lablup-course-item>
```

짠!

테스트 코스입니다.

4 Lectures

3 구성원

테스트 3입니다.

0 Lectures

2 구성원

예: 간단한 파이 차트

- 동기
  - 파이 차트 그래프를 svg로 매번 그리는 것은 코드 중복이 심하다
  - HTML 문법에 piechart 등이 있어서 간단하게 파라미터만 줘서 파이 차트를 원하는대로 그릴 수 있었으면 좋겠다
- 사양
  - 속성: 색상, 사이즈, 전체 값 크기, 현재 값 크기, 폰트 사이즈
  - 폰트는 자동으로 그래프 크기에 맞게 리스케일링되게

스타일 / 템플릿

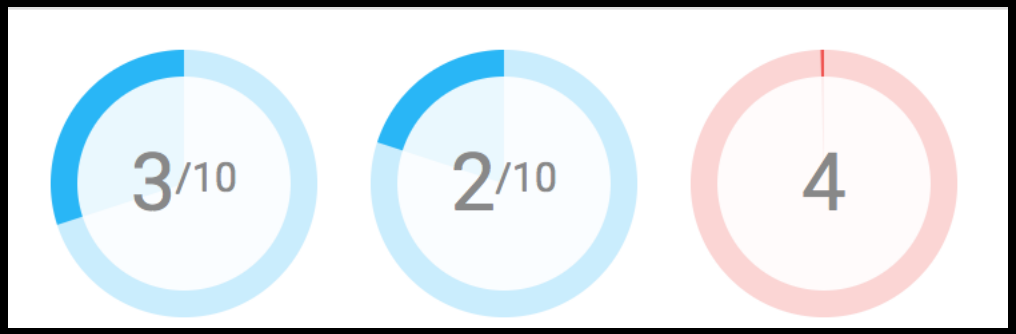
```
<link rel="import" href="../polymer/polymer.html">
<link rel="import" href="../paper-styles/paper-styles.html">
<link rel="import" href="../iron-icon/iron-icon.html">

<dom-module id="lablup-piechart">
    <style>
        #chart {
            cursor: pointer;
        }
    </style>
    <template>
      <svg id="chart"
            xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xl
ink" version="1.1“ viewBox="0 0 1 1">
          <g id="piechart">
            <circle cx=0.5 cy=0.5 r=0.5 />
             <circle cx=0.5 cy=0.5 r=0.40 fill="rgba(255,255,255,0.9)" />
               <path id="pievalue" stroke="none" fill=“rgba(255,255,255,0.9)" />
               <text id="chart-text" x="0.5" y="0.5" font-family="Roboto” font-size="
0.3" text-anchor="middle" dy="0.1">{{ number }}<tspan id="unit-text" font-size="0.2" dy="-0
.07">{{ unit }}</tspan></text>
            </g>
        </svg>
     </template>
```

코드

```
    <script>
    Polymer({
        is: "lablup-piechart",
        properties: {
            number: {
                type: Number,
                value: 50
            },
            maxnumber: {
                type: Number,
                value: 100
            },
            unit: {
                type: String,
                value: '%'
            },
            ....
            ready: function () {
                this.sizeParam = this.size + "px";
                Polymer.dom(this.root).querySelector("#chart").setAttribute("fill", this.c
hartcolor);
                Polymer.dom(this.root).querySelector("#chart-text").setAttribute("fill", t
his.textcolor);
            ....
      });
   </script>
</dom-module>
```

실제 사용!

```
<link rel="import" href=”lablup-piechart.html">
…
<lablup-piechart
        url="/dashboard/circle"
        number=”3"
        maxnumber="10”
        chartcolor="#cddc39"
        unit="/10" size="100">
</lablup-piechart>
```



Polymer : 장점

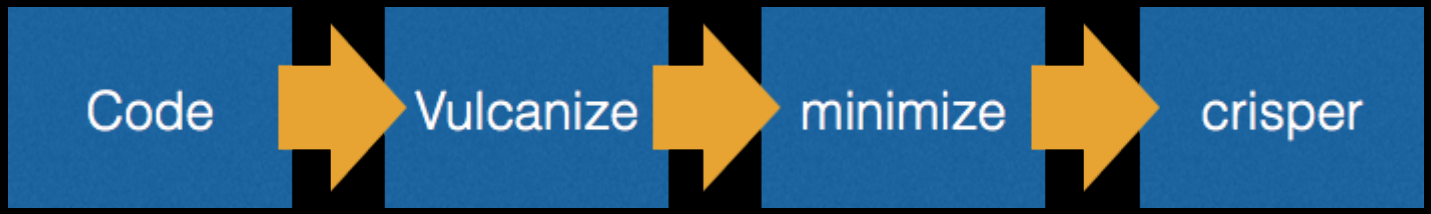
- 올인원 솔루션
  - 이것저것 다 구현해 놓았음
- Webcomponents.js에 지대한 공헌
  - 사실상 멱살 잡고 끌고 간다고 봐도 과장은 아님
- Component-driven
  - customelements.js 등에서 공유되는 custom elements들이 많음
  - 그냥 가져와서 쓰면 된다
- 크롬 브라우저의 네이티브 지원
  - Polyfill이 필요하지 않다! (크롬에서만)

Polymer : 단점

- 엄청난 덩치
- 느림
- 무지막지한 xhr 의존성 체크 오버헤드
  - Vulcanize : 미리 몽땅 체크해서 중복 import 구조를 없애고 파일 하나로 통합
  - Crisper : CORS 이슈 해결을 위해 Javascript를 HTML에서 뜯어냄
- 불안정
  - 디폴트 컴포넌트로 주는 컴포넌트들의 변화가 심함
  - Iron- 시리즈들은 멋짐. Paper- 시리즈들은 버립시다.

Polymer: 프로덕션

- Vulcanize
  - 라이브러리를 중복 제거하고 합침: 일종의 컴파일
- Minimize
  - 코드 부피를 줄임 (그냥 만들면 메가 단위가 나옴)
- Crisper (선택)
  - CORS 문제를 해결하는 도구



X-Tag

https://x-tag.readme.io

모질라 재단도 손에 카드를 들고 있(긴 하)다...

X-Tag

- 모질라 재단의 일부가 참여하여 개발한 web components 라이브러리
  - 모질라가 web components 개념을 좋아했는지는 별개
  - 개발자들이라면 새 제안이 나오면 손 대 보고 싶죠
- Thin wrapper 추구
  - Shadow DOM 및 그에 기반한 template을 사용하지 않음
- Web components를 날것으로 다뤄보고 싶은 경우 괜찮은 선택이 될 수 있음
  - 실제 써먹기는 애매함
  - 자동차를 주문했더니 매뉴얼과 재료가 택배로 온 느낌임

Bosonic

http://bosonic.github.io

자동차 부품보다는 모터달린 킥보드를

Bosonic : 요약

- X-Tag과 Polymer의 중간 정도의 속도
- 철학
  - 다른 라이브러리의 기반 컴포넌트를 추구함
  - 특히 UI 컴포넌트
  - 지향점은 React.js 와 유사하다고 볼 수가 있음
    - 구현 형태가 완전히 상이하지만.
- 실제 사용하려면 react나 angular.js 등이 필요
  - UI 컴포넌트에 집중 - JQuery UI와 타겟이 유사함
  - 데이터 바인딩의 측면에서는 Bosonic이 훨씬 좋다

Bosonic - 단점

- 작년 시점에선 써먹기 어려운 물건
  - 컴포넌트가 적다
  - 너무 과거 브라우저까지 (IE9) 커버하려다 보니 오히려 호환성 문제가 좀 심각하다
- 종종 기반 라이브러리인 webcomponents.js 와의 호환이 깨짐
  - 기다리거나, 패치하거나
- Thin wrapper로서의 기능에 충실함
- 리부트 한다고 합니다...

React

https://facebook.github.io/react

페북: 우리는 당신들의 주장에 동의하지 않습니다

저흰 심지어 PHP도 자체 컴파일러 만들어서 써요 *HHVM

React : 같은 문제에 대한 완전히 다른 대답

- 뷰에만 특화된 라이브러리
  - Function들은 view에 바인딩되어 있다.
  - MVC의 V.
- 모든게 컴포넌트
  - 컴포넌트를 가볍게 정의하고, 앱은 컴포넌트들의 컴포넌트가 된다.
- Functional style library
  - 룰이나 코딩 컨벤션을 제공하지 않음
  - 맘대로 써라. (페북은 flux를 적용해서 씁니다)

React : 특징

- 템플릿과 컴포넌트가 완전히 믹스됨
  - 근본론자들은 기겁할 코드
  - 그럴 필요가 없지 않나?

React : 특징

- JSX: 자바스크립트와 HTML의 믹스

```
var nameCard = React.createClass({
    …
    render: function() {
      return (
       <div class="card">
         <h2>Name : <span>{this.state.name}</span></h2>
       </div>
      )
    }
  });

  React.render(
    <nameCard name={userName} />,
    document.querySelector('#name-section')
  );
```

React : 특징

- Fake/Virtual DOM
  - 브라우저의 DOM element의 비정상적으로 거대한 속성 상속 레거시에서 벗어남
  - 빠르다!
- Shadow DOM이 아니다.
  - 실제 DOM이 shadow DOM의 루트 역할에 해당됨
  - Virtual dom은 인스턴트하게만 존재

React : 특징

- 뮤테이션 구현을 할 필요가 없음
  - DOM mutation : 동적 페이지 갱신의 필수 요소
  - 속도 문제는 대부분 여기서 발생함
    - 대부분의 라이브러리들이 권장하지 않...
    - 지만 Polymer만 하더라도 dom-if 나 two-way data binding, updateStyles() API를 지원함 (느림)
  - Virtual DOM에 diff 적용이 됨
    - 바로 변경 사항 적용이 됨
  - 추가 장점: 밖에서 보면 immutable element처럼 보인다

React : 단점

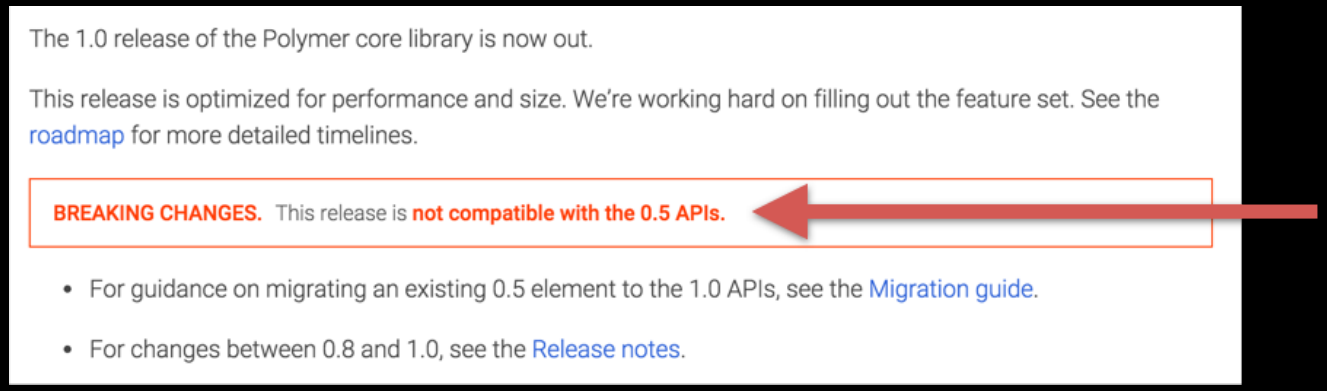
- 스타일링 구현에 대한 불만
  - 스타일이 컴포넌트 밖에 있어야 함
  - 장점으로 보는 입장도 있지만, 재사용 가능한 엘리먼트라는 측면에서는 단점이라고 보는 것이 타당함
- React 의존성
  - React component는 react 안에서만 사용 가능함
- 모바일 환경에서는 일반 javascript 구현보다 느림

문제점?

솔직하게 말하면, 뭐가 문제가 아닌가? 를 말하는게 쉽습니다;

Polymer: it is too google to be true

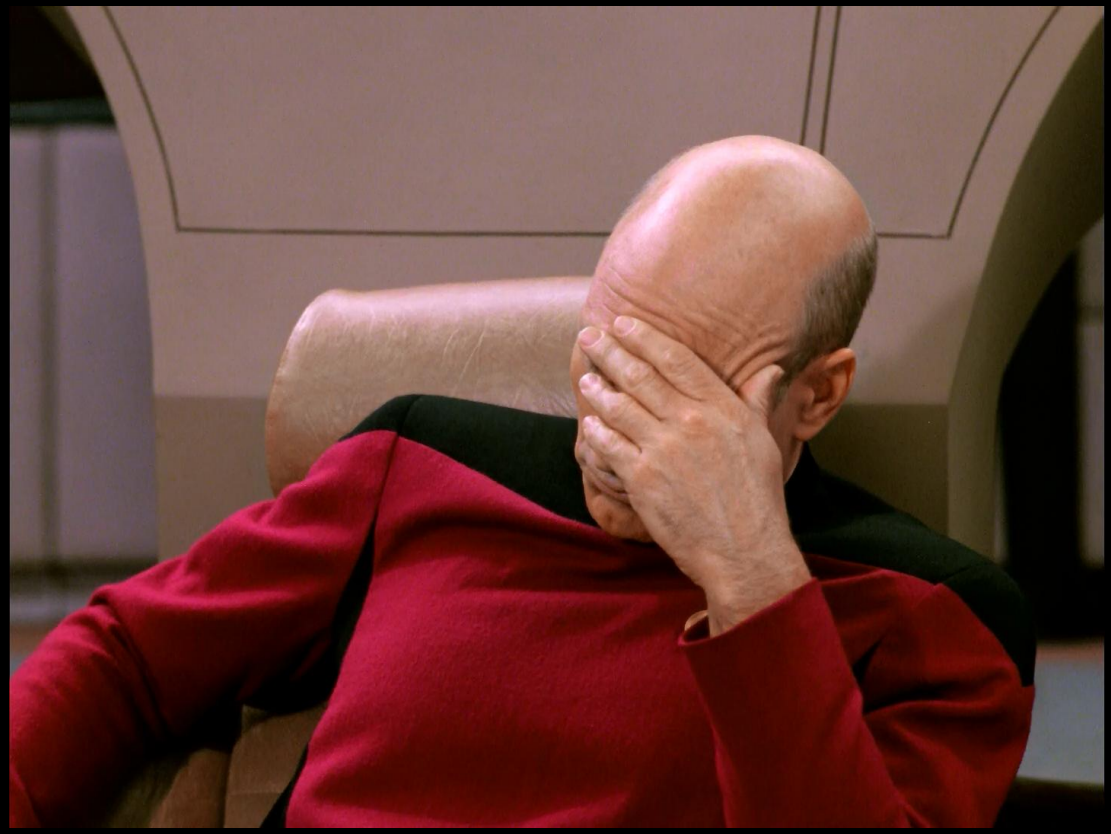
- 격변하는 스펙



  - 자동 마이그레이션 툴 지원 (잘 안 됨)
- 1.0에서 1.1로?
  - 10주만에 업데이트 되었는데 그 새 또 좀 바뀌었음 (Iron element)

근본적 이슈

- 브라우저 호환성
  - 왜 윈도우 7에선 안돼요? 왜 익스플로러에선 안돼요?

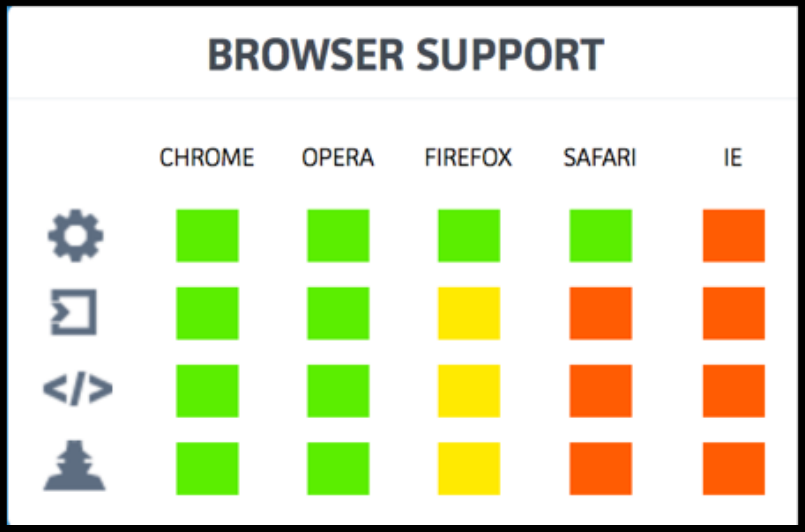


바다를 채워야 하는 polyfill

- 페이지 재렌더링 횟수
- 익스플로러
  - 거의 모든 웹컴포넌트 기능(html import 및 shadow DOM)을 polyfill에 의존해야 함
- 사파리
  - 있던 규격도 지우는 중

IE	Safari	Firefox	Chrome
79	47	34	9

*Tested with our project (without vulcanize)



FOUC prevention handling

- FOUC (Flash Of Unstyled Content)
- 재렌더링이 가져오는 필연
- 해결 방안
  - 최종 렌더링이 끝날 때 까지 body의 display 속성을 none으로 (*unresolved* attribute)
  - `<body unresolved class="fullbleed">`
  - 결과적으로 사이트가 느린 것 처럼 보임
    - 준비가 다 되어야 사용 가능하다는 관점에서는 맞음;
  - 단일 페이지 웹앱일 때 문제가 좀 있음
    - 예) Polymer + Backbone.js

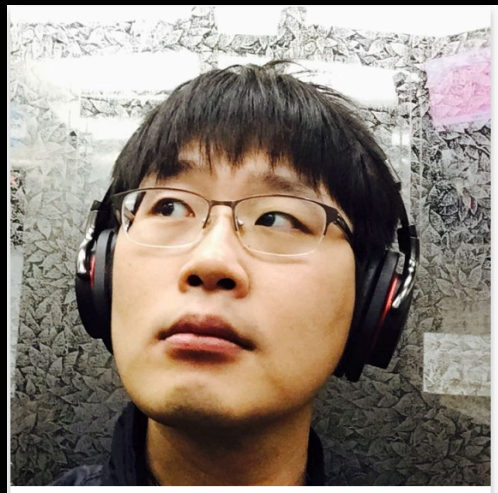
Electron: 현실적인 해결 방안

- 해결책 : cross-platform webapp container
- Electron
  - Github에서 개발한 Node.js 기반의 웹 앱 프레임웍
  - Node 앱 또는 웹 앱을 Chromium 런타임으로 wrapping
  - 윈도우 / 맥 / 리눅스 지원
  - ATOM 에디터의 기반
- 앱은요?
  - Apache Cordova

요약

- 고전 강독회
- HTML5 Web Components 개요
  - 미래의 프론트엔드
- 오픈소스 구현체들인 Polymer (및 기타 등등) 소개
- React 소개 및 비교
- 그래서 쓰면 됩니까? – 네.
- 정말요? – 음...

요약



Jeongkyu Shin

Jonghyun Park

Joongi Kim

Thank you

Lablup Inc.

http://www.lablup.com

코드온 구현체: https://codeonweb.com