2019년
FE 프레임워크를 배우는 기분

FE 인싸들이라면 알고 있어야 하는 프레임워크 기술들

DEVIEW 2019

2019.10.29

NAVER 박재성

[BG Source] https://codepen.io/vcomics/pen/aGmoae

슬라이드의 온라인 버전은 https://netil.github.io/에서 확인 하실 수 있습니다.

내가 JS 프레임워크 생활을 2013년 시작했다.
그 때 시작한 프레임워크가 백개다 치면은..
지금 나만큼 사는 프레임워크는 나 혼자 뿐이야.

나는 어떻게 여기까지 왔느냐?

잘난  ANGULARJS by Google 제끼고.

못난  jQuery write less, do more. 보내고...

 React

오늘의 목표

프레임워크가 사용하는 기술적 요소들에 대한

"A-ha moment"

를 발견하는 것.

Q. 야, 내가 새로운 프로젝트를 받았는데, 한동안 FE 개발을 안했더니 뭐가 어떻게 달라졌는지 잘 모르겠어. 요즘 FE 트렌드 좀 알고 있지?

A. 일단, 프레임워크를 선택해야 하는데, VDOM 지원되는 걸 써야지.
뭐 React나 Vue.js면 될꺼야.

초기 로딩 성능을 높이려면 CSR보단 SSR은 기본이야.
그리고 Code-splitting을 해서 로딩 파일 크기 줄여야 하는거 알지?
번들링할 때 Tree-shaking 하는거 잊지 말고.
근데, 이거 다 일일이 환경 구성하기 힘드니까
일단, CRA 같은 CLI를 사용해. 모바일 앱도 만들꺼면
RN이나 NativeScript를 사용하면 되고.

참, 요새 타입이 기본인거 알지? TS 구성은
Babel + preset-typescript로 하면 돼.
그리고 Hooks이 핫한거 알지? 이제 컴포넌트는 클래스가
아닌 함수형으로 작성해야 돼. Hooks은 함수형에서만 사용할 수 있거든
...

[참고] How it feels to learn JavaScript in 2016 (한글번역)

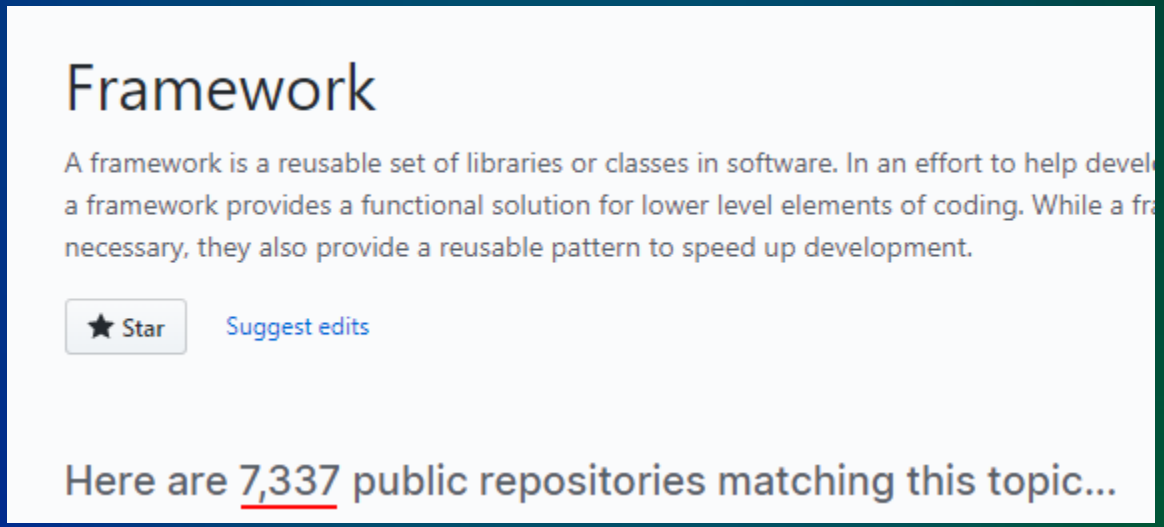
Framework Fatigue

2019년 프레임워크를 배우는 기분은,
피로감의 심화 그리고
"남들이 하는대로 따라하기"

전략이 최고의 전략이 되어버림.

넘쳐나는 프레임워크들

GitHub에는 JavaScript로 작성된 "framework"
주제를 갖는 프로젝트는 약 7K개 이상



[참고] https://github.com/topics/framework?l=javascript

2019년 사용해야/배워야 하는

xxx 프레임워크

- 2019 Top/Best Framework...
- The 2019 Roadmap To...
- Most interesting to Learn in 2019
- Top trending Frameworks in 2019 ...

아마도? 2020년에도 반복해서 보게될 것

XXX Vs. OOO 프레임워크

[참고] I created the exact same app in React and Vue. Here are the differences
Angular vs React vs Vue: Which is the Best Choice for 2019?

프레임워크 선택의 기준은?

성능? 기능? 생태계?

단순하게 어떤 '요소'만을 기준한 비교는 어렵다.

모든 것은 "trade-off"가 있기 때문.

[참고] JS Frameworks Benchmark
RealWorld example apps

얼마 만큼의 작업 영역을 프레임워크가 처리하나?

Small vs. Large Scope

[참고] Evan You on Vue.js: Seeking the Balance in Framework Design | JSConf.Asia 2019

Small Scope: 

장점

- 단순한 컨셉과 유연성 → active한 생태계
- 작은 메인터넌스 영역 → 개발팀은 새로운 아이디어 구현에 보다 집중

단점

- 단순한 컨셉은 오히려 복잡한 문제 해결에 많은 작업 필요
- 패턴들(추상화된), 아키텍처들은 시간 지남에 따라 'semi-required' 형태가 됨
  ex) Redux, HOC, CSS-in-JS, etc.
- 생태계가 너무 빠르게 발전해 파편화를 발생시킬 수 있다.

Flux 사례

2014년 발표된 단방향 데이터 흐름을 갖는 아키텍처

매일 새로운 아이디어를 구현한 도구들이 등장

[참고] Flux Comparison by Example
We Compared 13 Top Flux Implementations — You Won't Believe Who Came Out On Top!

Large Scope: 

장점

- 일반적인 문제들은 빌트인을 통해 해결
- '중앙 집중화식 디자인'은 일관성(생태계) 유지하게 함

단점

- 높은 학습비용
- 빌트인 해결책이 사용자 상황과 맞지 않는 경우에 대한 비유연성
- 새로운 'fundamental' 아이디어의 도입 비용이 높아진다.

Frameworks are not tools for organising your code,
they are tools for **organising your mind.**

[참고] Rich Harris - Rethinking reactivity

오늘날의 기본적 접근 방법

Reactivity

그리고

Observability

[참고] Evan You on Vue.js: Seeking the Balance in Framework Design
Rethinking reactivity (Slide)

(Functional) Reactive?

The basic idea behind reactive programming is that
there are certain datatypes that represent a value "over time".

[Source] What is (functional) reactive programming?

사용자가 해당 소프트웨어를 사용하기 위해서
어떤 입력을 발생 시켰을 때 꾸물거리지 않고
최대한 빠른 시간 내에 응답을 한다는 의미다. - 임백준

[Source] 리액티브 개발 패러다임에 담긴 메시지

The essence of FRP is to specify the dynamic behavior
of a value completely at the time of delcaration.

[Source] Heinrich Apfelmus

[참고] What is Reactive Programming?

Observable

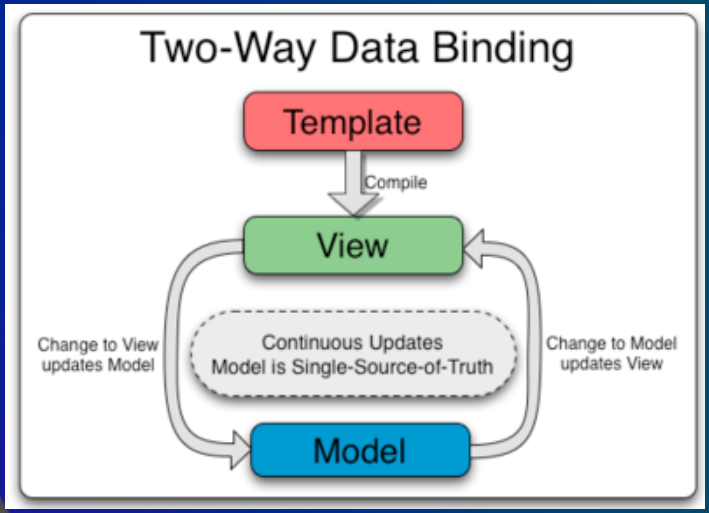
Functions like a spreadsheet, where cells (like formulas)
run automatically whenever their referenced values change.

	A	B	C	D	E
1	20	70			
2	50	-30			
3					
4					
5					
6					
7					

LANPAR(LANguage for Programming Arrays at Random) (1969)
VisiCalc (1979) / Lotus 1-2-3 (1983) / Quattro Pro (1988) / MS Excel (1985)

[참고] How Observable Runs
스프레드시트 탄생과 엑셀의 미래

Two-Way Data Binding




```
<div>
    <!-- 'name'은 모델명 -->
    <input type="text" ng-model="name">

    <!-- 템플릿 표현 식 -->
    <h2>{{name}}</h2>
</div>
```

React

```
this.setState({ name: "John Doe" });
```

enqueues changes to the component state and tells React that this component and its children need to be re-rendered with the updated state.

Vue.js

```
var vm = new Vue({ data: { a: 1 } });
// 'vm.a' is now reactive
```

When you pass a plain JavaScript object ... as its data option, will walk through all of its properties and convert them to getter/setters using Object.defineProperty.

[참고] React: setState(), Vue.js: Reactivity in Depth
Updating UIs: value comparison VS mutation tracking

이젠 React, Vue.js 그리고 Angular 중 하나.
(어쩌면 Svelte도?)

현실은?

[Source] [이육대 1부] 번개맨, 뿡뿡이, 펭수까지 EBS 인기 스타 총출동!

같은듯 다른 듯한
기술적 접근 방법들

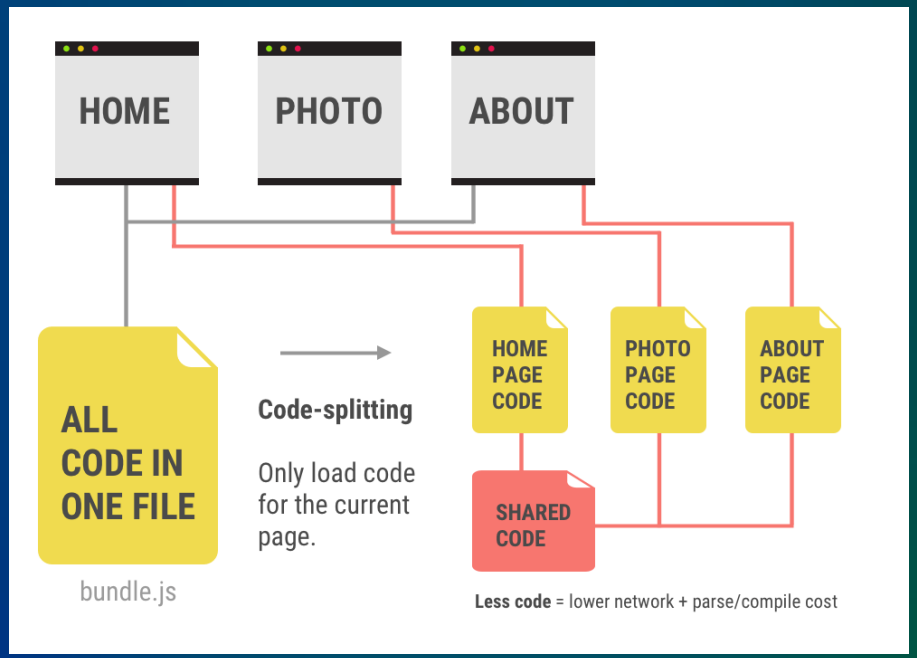
더 빠르게 빠르게

Loading/Build

Key Point: 필요한 것만 로딩

Code splitting

초기 번들 사이즈를 줄이고, 분할(모듈 or 라우팅 기준)된 파일(chunk)의 점진적 로딩 전략



[Source] https://developers.google.com/web/fundamentals/performance/webpack

[참고] Reduce JavaScript Payloads with Code Splitting

분리 기준 전략

- Vendor: 벤더(lodash, React) 기준으로 분리
- Entry Point: 번들러의 초기 진입 파일 기준으로 분리
- Dynamic: 동적 import() 사용 기준으로 분리

```
async getModule() {
    const {default: _} = await import(
        /* webpackChunkName: "lodash" */ 'lodash'
    );
}
```

- Routing: 페이지의 routing 기준으로 분리

[참고] webpack: Code Splitting
next.js: Automatic code splitting

Tree Shaking
(Dead code elimination)

실제 사용되는 코드만 번들링하고, 불필요한 코드는 제거

[Source] https://developers.google.com/web/fundamentals/performance/webpack

[참고] rollup.js에서 처음 도입한 기능

코드의 정적 분석을 통해
index.js에서 사용되지 않는 'module.inner.b'는 제거

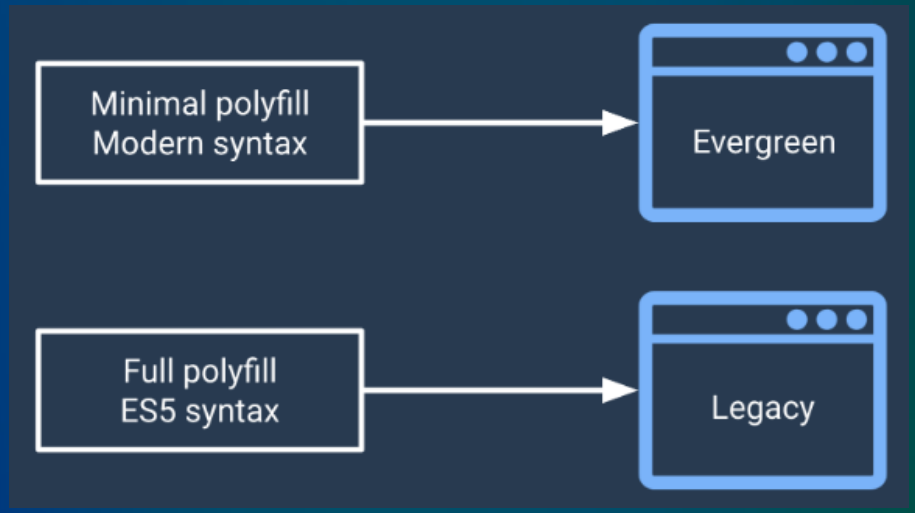
```
// inner.js
export const a = 1;
export const b = 2;

// module.js
import * as inner from "./inner";
export { inner }

// index.js
import * as module from "./module";
console.log(module.inner.a);
```

그러나, 적용은 아주 '간단'하지만은 않다.

[참고] Reduce JavaScript Payloads with Tree Shaking
Webpack 4의 Tree Shaking에 대한 이해

Differential Loading

2개의 빌드를 생성하고, 모던/레거시로 나누는 전략
레거시에는 Polyfill 등을 포함




```
<script type="module" src="esm.build.js">
<script nomodule src="legacy.build.js">
```

[참고] Angular: Differential Loading

Rendering

Key Point:

- 렌더링 시점을 앞당기자
- 렌더링 비용을 줄이자

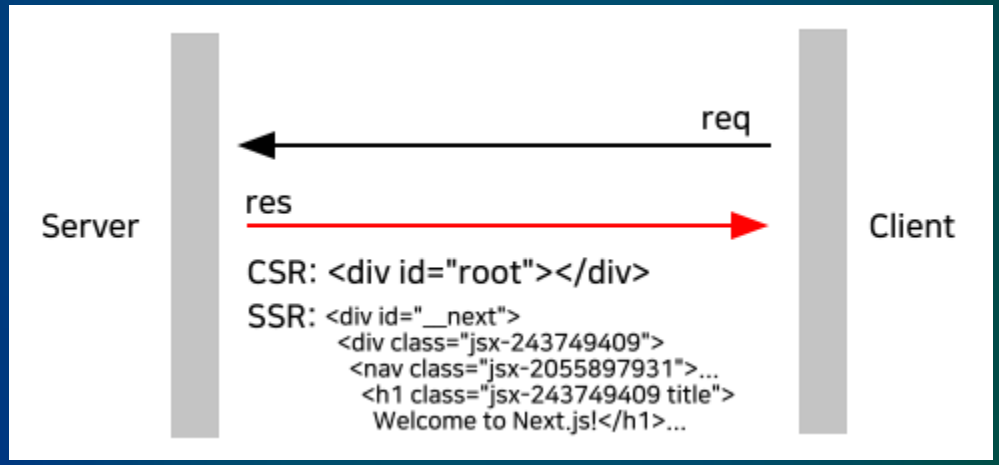
SSR
(Server Side Rendering)

초기 화면 구성에 필요한 코드를
서버에서 구성(렌더링)해 전달하는 전략

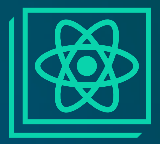
```
renderer.renderToString(vm); // Vue.js
ReactDOMServer.renderToString(element); // React
```

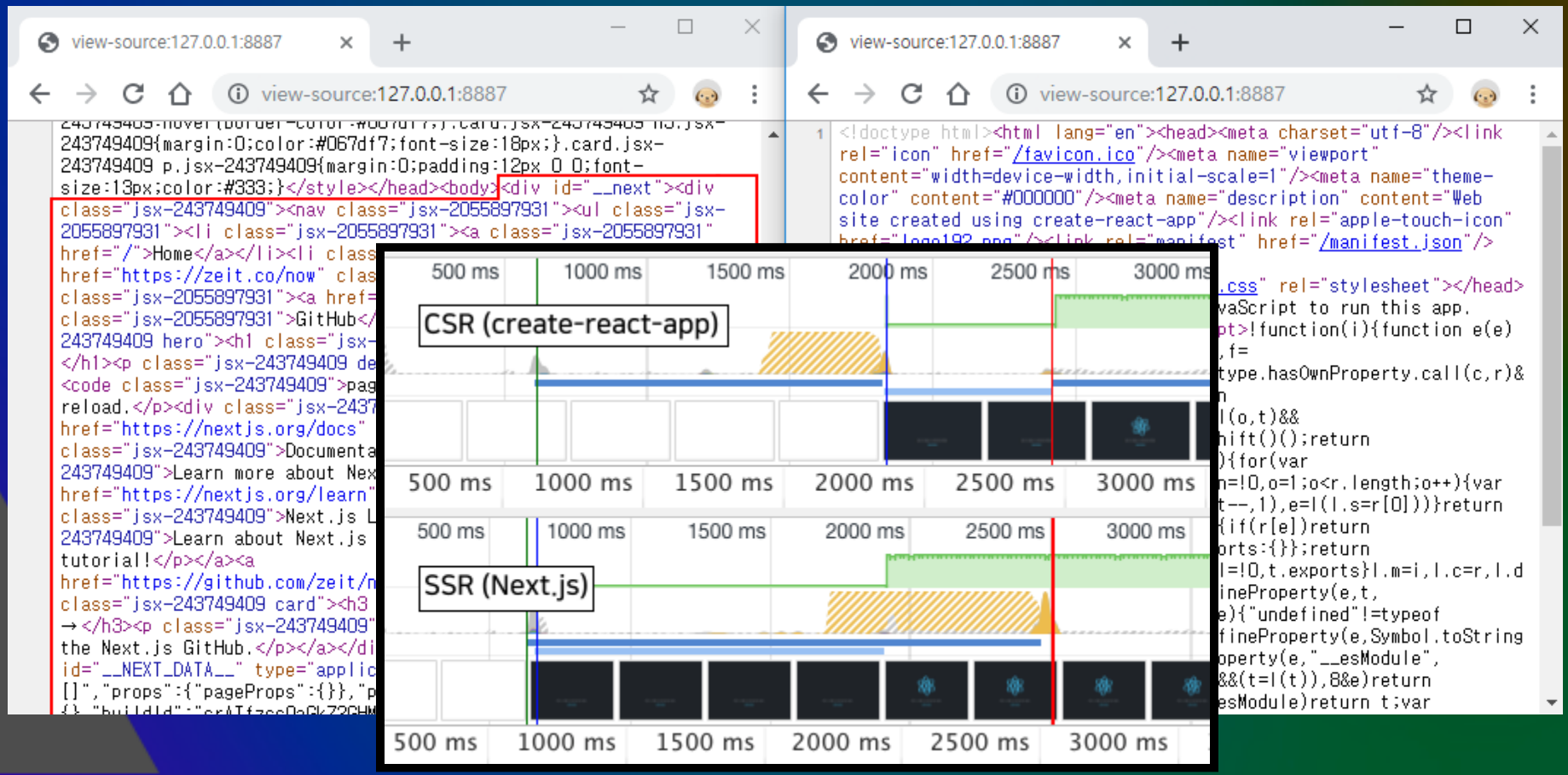
[참고] ReactDOMServer
Vue.js SSR Guide
Server-side Rendering (SSR): An intro to Angular Universal

SSR vs CSR



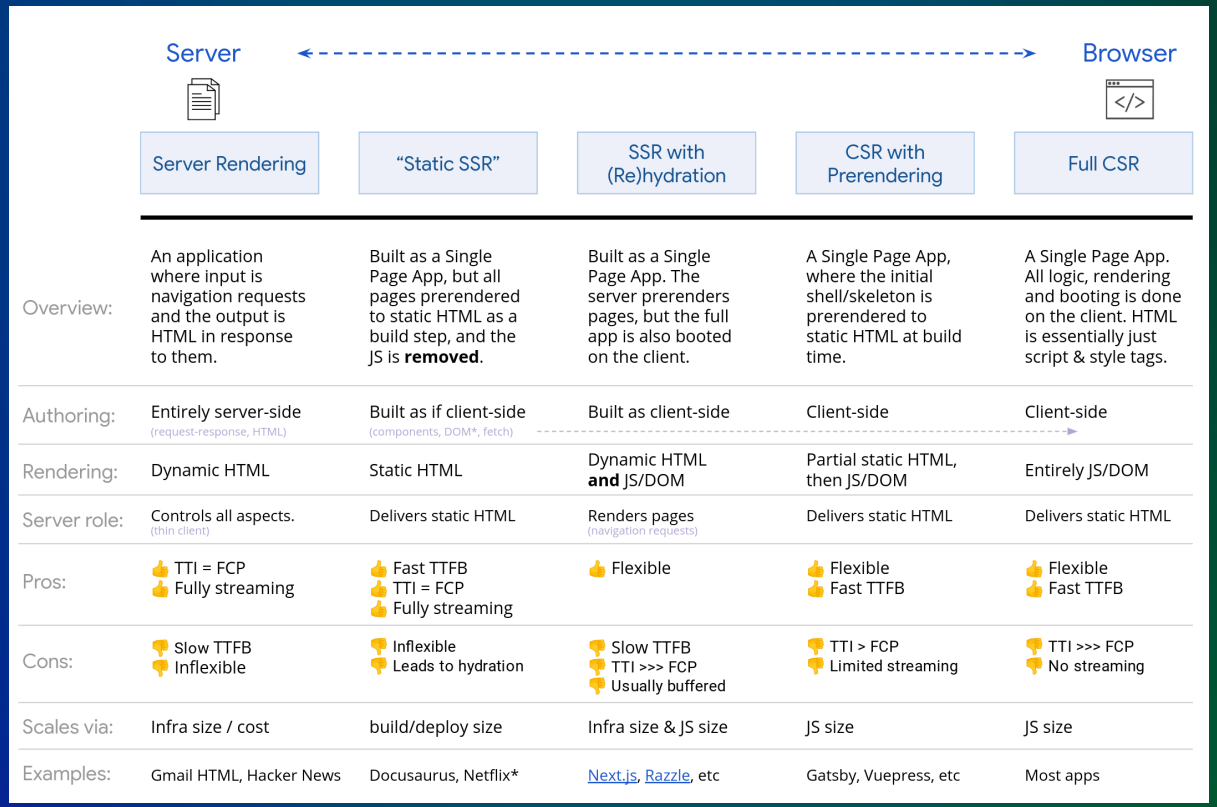
[참고] Next.js (SSR) vs. Create React App (CSR)
The Benefits of Server Side Rendering Over Client Side Rendering
Why it's tricky to measure Server-side Rendering performance

NEXT.JS vs. 


view-source:127.0.0.1:8887
view-source:127.0.0.1:8887
view-source:127.0.0.1:8887
view-source:127.0.0.1:8887
CSR (create-react-app)
SSR (Next.js)
500 ms
1000 ms
1500 ms
2000 ms
2500 ms
3000 ms

다양한 렌더링 기법

SSR, Static SSR, SSR with (Re)hydration,
CSR with Prerendering, Full CSR



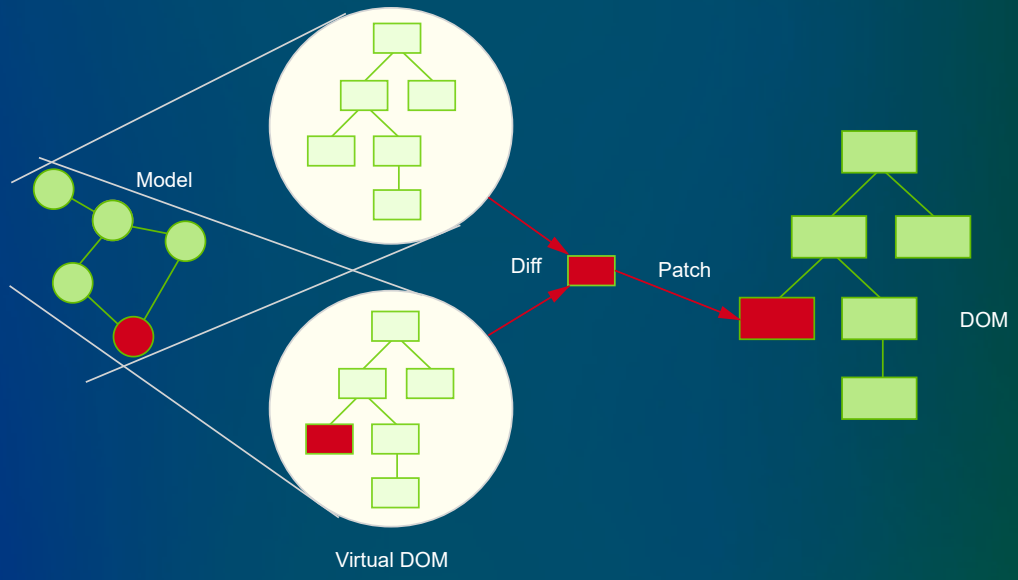
	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is **removed**.	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side (request-response, HTML)	Built as if client-side (components, DOM*, fetch)	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML **and** JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. (thin client)	Delivers static HTML	Renders pages (navigation requests)	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js, Razzle, etc	Gatsby, Vuepress, etc	Most apps

[Source] https://developers.google.com/web/updates/2019/02/rendering-on-the-web

[참고] The Cost Of Client-side Rehydration
Rendering on the Web: Performance Implications of Application Architecture (Google I/O '19)

VDOM (Virtual DOM)

VDOM은 프로그래밍 컨셉으로
JavaScript 객체로 표현된 DOM(UI 표현식) 구조



[Source] http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

모델(데이터) 변경시 실제 DOM 트리와 비교해
업데이트가 필요한 부분만 렌더링하는 기법

[참고] React's diff algorithm
Virtual DOM and diffing algorithm

VNode 표현 객체

```
<div class='holder'>
  <button class='blue'></button>
  <button class='red'></button>
</div>
```

```
// Simplified React's VNode shape
{
  type: 'div',
  props: {
    children: [{
      type: 'button',
      props: {className: 'blue'}
    }, {
      type: 'button',
      props: {className: 'red'}
    }],
    className: 'holder'
  }
}
```

[참고] React: createElement()
Vue.js: createElement()

필요한 영역만 업데이트

0ms

시작 초기화

```
render() {
  return (
    <label>
      {lapse.toLocaleString()}ms
    </label>
  );
}
```

Reconciliation?

VDOM diff

The algorithm React uses to diff one tree with another to determine which parts need to be changed.

[참고] React Docs: Reconciliation
React Fiber Architectur: What is reconciliation?

VDOM의 diffing 비용

상태가 변경되면 매번 diff 프로세스 수행을 거친다.

```
<template>
  <div id="content">
    <p class="text">Lorem ipsum</p>
    <p class="text">Lorem ipsum</p>
    <p class="text">{{ message }}</p>
    <p class="text">Lorem ipsum</p>
    <p class="text">Lorem ipsum</p>
  </div>
</template>
```

- Diff
  - Diff props of
  - Diff children of
    - Diff <p>
      - Diff props of <p>
      - Diff children of <p>
    - Repeat n times...

- Standard vdom diffing cost is relative to the total size of your view, rather than the number of nodes that may change.

diffing 비용은 실제 변경되는 노드 개수가 아닌 뷰의 크기에 비례

[참고] Seeking the Balance in Framework Design: VDOM inherently expensive
Rethinking reactivity: How Virtual DOM library works

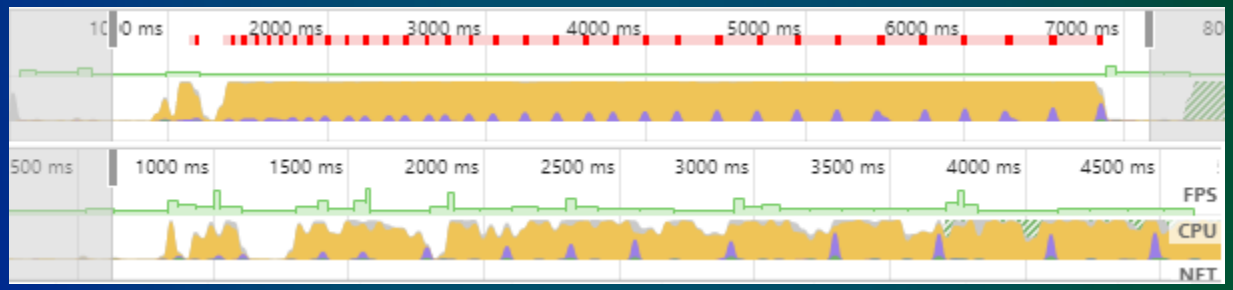
Diffing 비용 개선에 대한 React의 접근방법

"Perceived performance"

Time Slicing

The period of time for which a process is allowed to run in a preemptive multitasking system.

- Scheduling
  - Prioritization
  - Concurrency



특정 작업 실행이 전체 앱을 블럭하지 않도록 작업들을 잘게 나누어 처리하는 기법

[참고] Preemption_(computing) - Time Slice
Vue 3.0 Updates - Evan You | VueConfTO 2018

JavaScript는 single-thread(main thread)에서 작업이 수행되기 때문에,
작업이 진행 중이라면 새로운 작업은 현재의 작업이 완료될 때까지
실행이 차단(block)된다.

one thread = one call stack = one thing at a time

```
|--------------- 40 ----------------| frame
|-- 10 --|-- 10 --|-- 10 --|-- 10 --| frame
```

하나의 작업이 길게 수행되어
다른 작업의 실행이 차단되지 않도록

작업을 작은 조각(slice)으로 나누어

다른 작업이 수행될 틈새를 만들어 주는 것.

Scheduling

Push vs. Pull

React controls 'how' and 'when' to update UI.

setState() 호출시 즉시 실행되는 것은 아니며, 판단하기에
가장 이상적인 시점에 UI를 업데이트 하라고 전달하는 것

Push based

App(programmer) decide when to perform.

Pull based

Framework(React) make scheduling decision for you.

[참고] Andrew Clark: What's Next for React — ReactNext 2016

Prioritization

Scheduling은 다른 유형의 작업들간의
우선순위를 정할 수 있게 한다.

어떤 작업이 먼저 처리되어야 할까?

이벤트 처리 vs. 데이터 구독(Redux) vs. 애니메이션

사용자 경험을 해칠 수(Jank) 있기 때문에
'애니메이션'을 우선 순위를 높여 처리해야 한다.

Concurrency

우선순위가 낮은 작업이 아직 진행중 일때,
높은 순위 작업을 처리해야 하는 상황이라면?

1) 낮은 순위의 현재 작업을 일시 중지
- 현재 렌더링 stack을 일시 중지
- 현재 call stack을 "Stash"

2) 높은 순위 작업을 처리
- 별도의 call stack을 갖는 높은 순위 작업을 실행

3) 중단되었던 작업의 해당 지점부터 다시 실행
- 원래의 call stack으로 돌아가 실행을 재개

렌더링 컴포넌트를 어떻게
중지시킬 수 있을까?

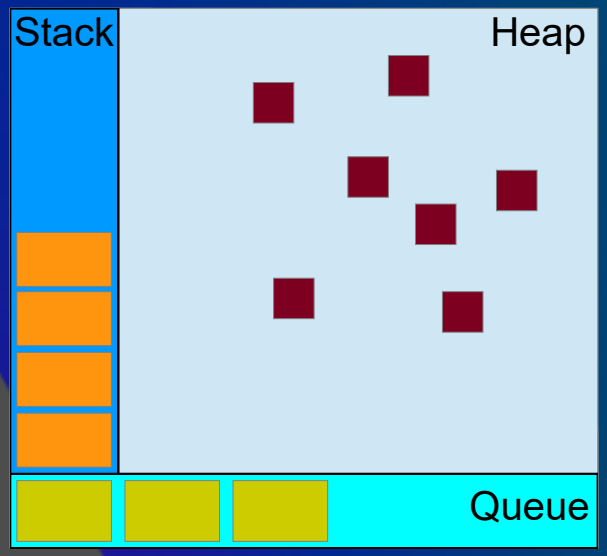
개념적으로 컴포넌트는 함수에
데이터를 전달하고 그 결과를 받는 과정

$$v = f(d)$$

중지 한다는 의미는?
결국, 함수 호출을 일시 중지 한다는 의미

Virtual Stack Frame

React는 v16(코드명 Fiber)에서 가상의 stack 프레임을
통해 stack을 재구현해 동시성 문제를 해결했다.

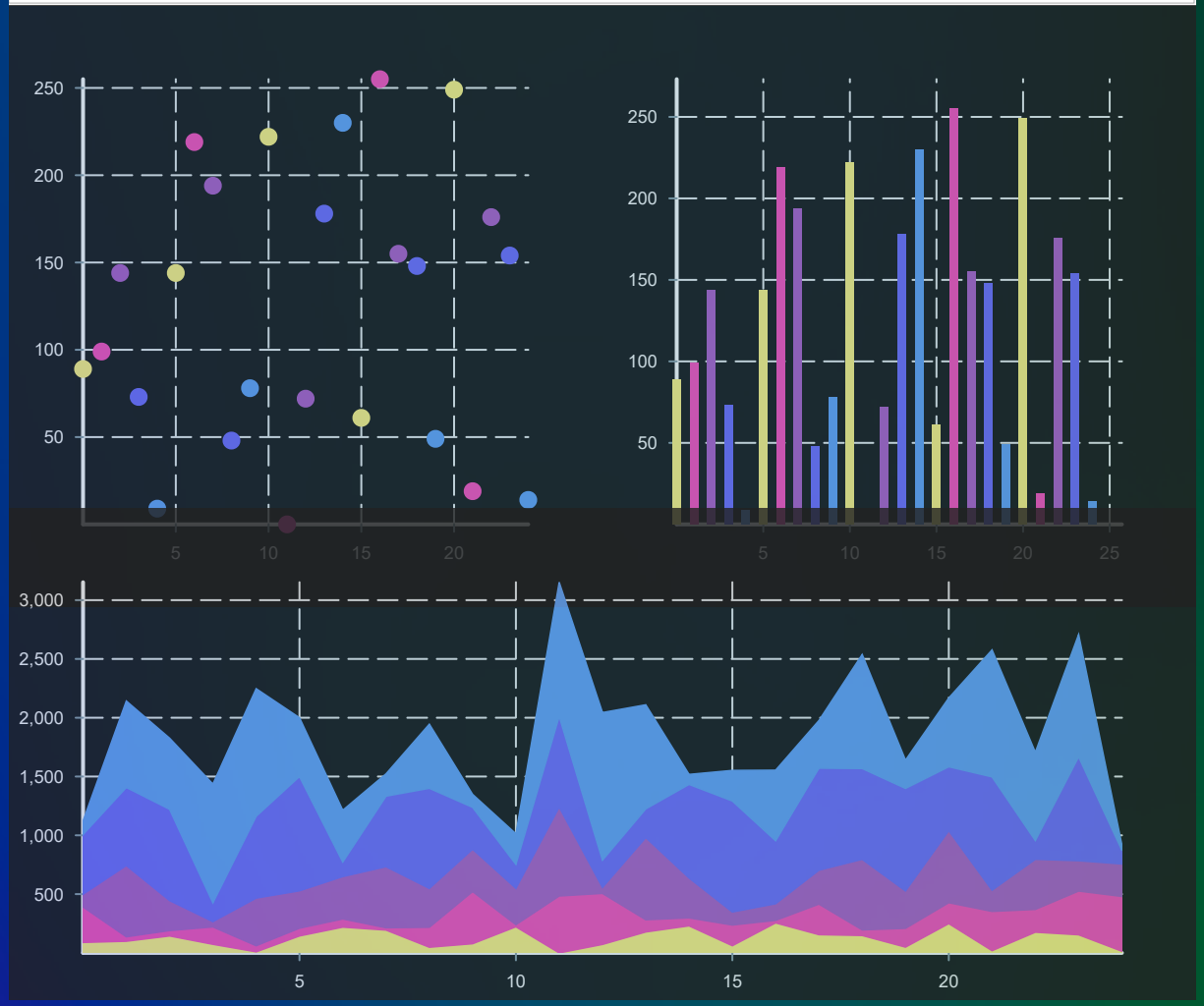


Stack Frame	Fiber
subroutine(function)	component type
body (nested function calls)	children
return address	parent component
arguments	props
return value	DOM elements

[참고] Andrew Clark: What's Next for React — ReactNext 2016
React Fiber Architecture
Eevent loop: Runtime concepts
Jake Archibald: In The Loop - JSConf.Asia
Loupe: visualisation of JavaScript's call stack/event loop/callback queue

● Synchronous ○ Asynchronous

longer input → more components and DOM



[참고] CPU async rendering demo (Source code)

```
// Synchronous
this.setState({value});

// Asynchronous
unstable_scheduleCallback(() => {
  this.setState({value});
});
```

```
// scheduler/src/Scheduler.js
function unstable_scheduleCallback(priorityLevel, callback, options) {
  timeout = timeoutForPriorityLevel(priorityLevel);
  ...
  requestHostCallback(flushWork);
}

// scheduler/src/forks/SchedulerHostConfig.default.js
requestHostCallback = function(cb) {
  ...
  // in a non-DOM environment
  if (typeof window === 'undefined' || typeof MessageChannel !== 'function') {
     setTimeout(requestHostCallback, 0, cb);
  } else {
    requestAnimationFrame(rAFTime => {
       onAnimationFrame(rAFTime);
    };
};
```

[참고] SchedulerHostConfig.default.js#L58, #342


```
const sleep = (ms, job = 1) => {
    const taskMs = job === 1 ? ms : ms / job;
    const cb = () => {
        const start = +new Date();

        while (true) {
            if (+new Date() - start >= taskMs) break;
        }
    };

    for (let i = 0; i < job; i++) {
        job.length === 1 ? cb() : setTimeout(cb, 0);
    }
}

document.querySelector("input[type=text]").addEventListener("keydown", e => {
    sleep(300, type === "sync" ? 1 : 50);
});
```

Template Compilation

스케줄링과 같은 코드가 포함될 필요 없으므로,
가벼운 런타임을 제공할 수 있다.

SVELTE

컴파일러가 '훌륭한 사용자 경험 코드를 생성'할 수 있다는 가설 증명을 위해 시작
효율적 명령(imperative) 코드로 변환, DOM을 업데이트

```
<!-- 작성 코드 -->
<script>
let count = 0;
function handleClick() {
  count += 1;
}
</script>
<button on:click={handleClick}>
  Clicks: {count}
</button>
```

```
// 컴파일된 코드
function handleClick() {
  count += 1;
  // 이전 값 무효화
  $$invalidate('count', count);
}

if (changed.count) {
  // t6 => <text> node
  set_data(t6, ctx.count);
}
```

VDOM의 diffing과 인지적 성능개선 처리가 필요없다.

[참고] A simple Svelte component
Compilers are the New Frameworks

효율적 코드 작성

Hooks

[Source] https://tenor.com/view/sakanaction-fishing-gif-5557678

특정 위치의 코드(place in code) 실행을 낚아채(Hooking)
기본 동작(또는 특정 조건에만 반응)을 다른 동작을 하도록 만든다.

jQuery's Special event hook

```
$("#myButton").on("click", function(e) { ... });  // click 이벤트 바인딩
```

```
// Special event hook
jQuery.event.special["click"] = {
    add: function(handleObj) { ... }, // 이벤트를 .on()으로 바인딩할 때마다 실행
    remove: function(handleObj) { ... }, // 이벤트를 .off()로 제거할 때마다 실행
    ...
}
```

```
jQuery.event = {
    add: function( elem, types, handler, data, selector ) {
        ...
        special = jQuery.event.special[ type ] || {};

        while ( t-- ) {
            ...
            if ( special.add ) {
                special.add.call( elem, handleObj );
            }
            ...
            handlers.push( handleObj );
        }
```

[참고] jQuery는 이벤트를 어떻게 처리하는가?
https://github.com/jquery/jquery/blob/2.2.0/src/event.js#L189

React Hooks

"hook into" React state and lifecycle features from function components.

컴포넌트의 기존 구조를 변경하지 않고
상태를 갖는 로직의 재사용과 작은 함수로 분리해
단순화 한다.

[참고] Vue.js는?
vue-hooks → Composition API RFC

Class vs. Function Component

```
// Class component (stackblitz.com/edit/react-t8mb19)
import React, {Component} from "react";

export class Greeting extends Component {
  constructor(props) {
    super(props);
    this.state = {name: "Mary"};
    this.handler = this.handler.bind(this);
  }

  handler(e) {
    this.setState({name: e.target.value});
  }

  render() {
    return <section>
      <input value={this.state.name} onChange={this.han
      <h1>{this.state.name}</h1>
    </section>;
  }
}
```

```
// Function component w/useState Hook
// stackblitz.com/edit/react-nbkstu
import React, {useState} from "react";

export function Greeting(props) {
  const [name, setName] = useState("Mary");

  function handler(e) {
    setName(e.target.value);
  }

  return <section>
      <input value={name} onChange={handler} />
      <h1>{name}</h1>
  </section>;
}
```

[참고] React: Introducing Hooks

useState() vs. setState()

```
const [name, setName] = useState("Mary"); // service call

function mountState<S>(initialState:(() => S) | S):[S, Dispatch<BasicStateAction<S>>] {
    ...
  return [hook.memoizedState, dispatch];
}

// dispatch: setName()
function dispatchAction<S, A>(fiber: Fiber, queue: UpdateQueue<S, A>,action: A) {
    ...
    scheduleWork(fiber, expirationTime);
}
```

react-reconciler/src/ReactFiberHooks.js#L823, ReactFiberHooks.js#L1121

```
this.setState({name: "Mary"}); // service call

const classComponentUpdater = {
    enqueueSetState(inst, payload, callback) {
        ...
        enqueueUpdate(fiber, update);
        scheduleWork(fiber, expirationTime);
    }
}
```

react-reconciler/src/ReactFiberClassComponent.js#L182

CLI는 이제 기본

create-xxx-app

CLI도 JavaScript로 작성된 코드 (ex. crate-react-app)

```
$ npx create-react-app my-app
```

```
// createReactApp.js
const commander = require('commander');
...
function createApp(name, verbose, version, ...) {
  const root = path.resolve(name);
  const appName = path.basename(root);

  checkAppName(appName);
  ...
```

또는 직접 만들수 있는 도구들도 존재: Commander.js, yargs

Create React App

Set up a modern web app by running one command.

ng new (Angular CLI)

The Angular CLI makes it easy to create an application that already works, right out of the box.

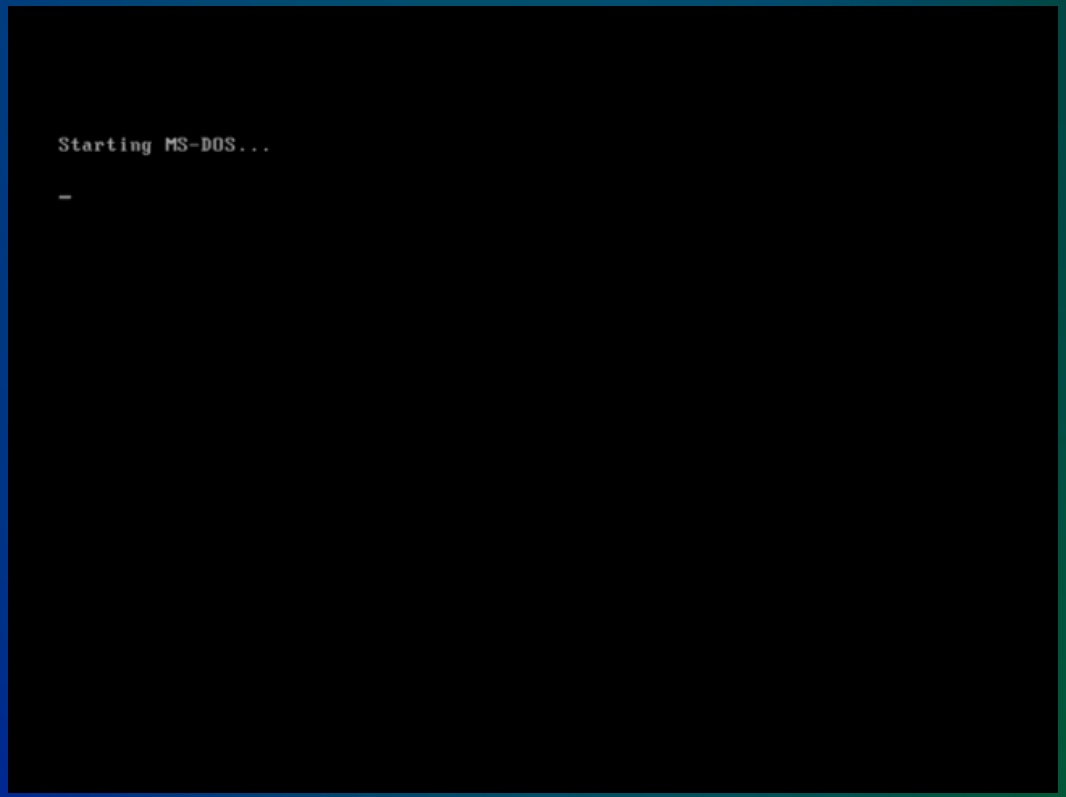
Vue CLI

Vue CLI is a full system for rapid Vue.js development.

그런데, CLI가 불편해서

GUI가 등장한 것 아니었나?

아무리 좋은 것이라도 끝이 있게 마련이다. 이제 명령 프롬프트는 과거의 것이다.
"MS-DOS여,... 너는 분명 PC 혁명의 시작에 한몫을 했다. 결코 잊혀지지 않을 것이다."
- 굿바이 프롬프트 · · · 마침내 떠나는 DOS에게 (2016.12.07)



[Source] Windows 3.1 - Startup

컴포넌트와 타입

컴포넌트 구조 기반의 설계

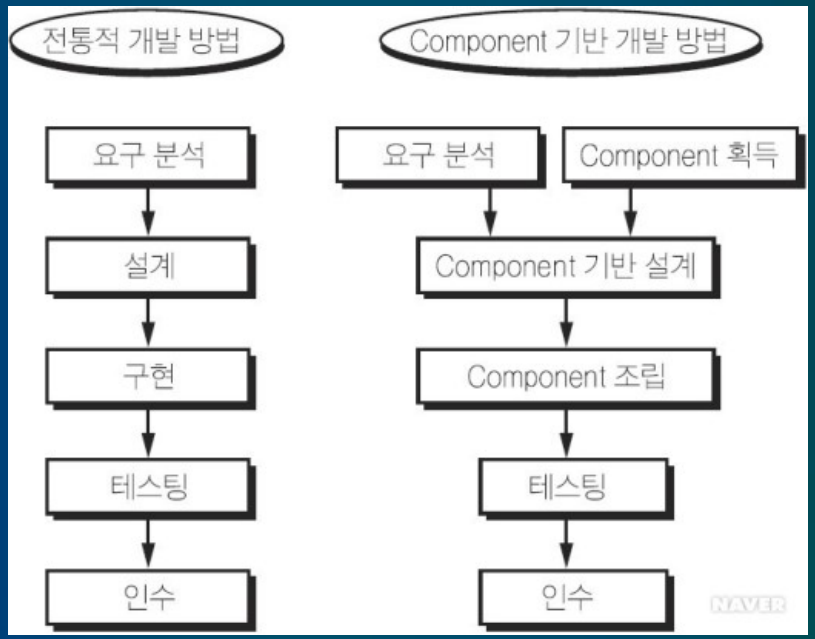
모던 프레임워크의 '성공'은 컴포넌트 기반에 기인

```
const Compo01 = <h1>Title</h1>;
const Compo02 = <div><h2>Sub Title</h2></div>;

ReactDOM.render(<>
    <Compo01 />
    <Compo02 />
    ...
</>, document.getElementById("root"));
```

- React.Component
- Vue.js Component
- Polymer: lit-html, litElement

CBD(Component Based Development)

기존의 시스템 및 소프트웨어를 구성하고 있는 컴포넌트를 조립해서 하나의 새로운 애플리케이션을 만드는 소프트웨어 개발 방법론 - IT용어사전 (한국정보통신기술협회)



[Source] 컴퓨터인터넷IT용어대사전 (2011)

WebComponents 표준화를 통해 시도되었지만,
낮은 브라우저 커버리지로 주도권을
모던 프레임워크들의 '컴포넌트'에 넘겨준 상황

보편화된 정적 타입 시스템

JavaScript에서 타입은 이제 보편화 되었다.

```
const add = (x: number, y: number) =>  x + y;
```

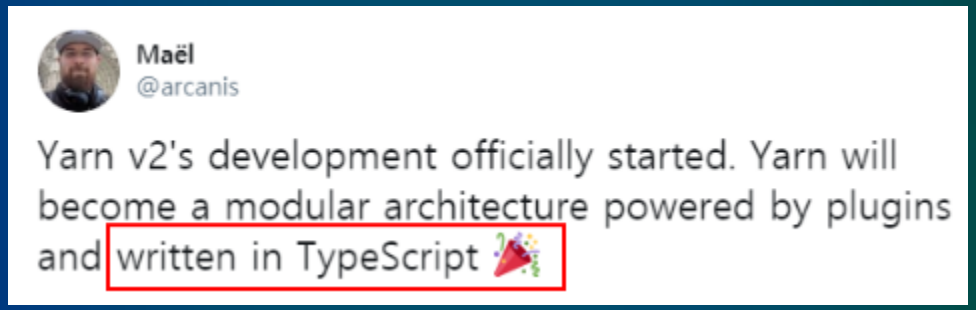
TypeScript

Vue.js, Angular, create-react-app TS지원,
lit-html, LitElement, etc.

flow

React, ~~Yarn~~

FB 프로젝트인 Yarn도 이제 TS 진영으로...



Flow는 문제 없지만, 컨트리뷰션 장벽을 낮추기 위해 TS로 전환

[참고] https://twitter.com/arcanis/status/1088450974404927491

성장비결

CoffeeScript와 Dart처럼 JavaScript로 컴파일되는
정적 타입 언어가 존재했지만 성공하지 못했다.

몇가지 이유?

- 학습비용: TS는 JS 슈퍼셋으로 문법적 차이가 거의 없음
- 타입의 옵트인 정의: 사용자가 타입 사용을 결정
- 강력한 도구와 생태계: VS Code, DefinitelyTyped 등의 지원

[참고] List of languages that compile to JS

혼자서는 어렵다.

강력한 생태계 기반 성장

오늘날의
모던 프레임워크 구성셋

프레임워크라면 기본적으로 갖춰야할 생태계

- CLI (Command Line Interface)
- 상태 관리자
- 라우터
- SSR/정적 사이트 개발 도구
- Universal Framework
- 네이티브앱(desktop/mobile) 개발 도구

Universal Framework

/n[eu][sx]t.js/i

'종합선물 세트'

- SSR
- Routing
- Static file serving
- Automatic code splitting
  각 페이지들의 import 항목들은 페이지별 번들링 된다.
- Built-in CSS
  - CSS-in-JS
  - styled-jsx: isolated scoped CSS

RFC

(Request For Comments)

프로세스 도입

자유롭고 투명한 기능 제안과 토론

향후 프레임워크의 개발 방향성을 미리 엿볼 수 있다.

- React RFCs
- Vue.js RFCs
- Node.js package-maintenance

개선 노력

RN의 경우, 불만을 듣고 개선에 대한 계획을 공개

What do you dislike about React Native? #64

Closed cpojer opened this issue on 6 Dec 2018 · 145 comments

cpojer commented on 6 Dec 2018 • edited — Contributor

Edit: See our reply here: #104.

We, the React Native team at Facebook, would like to get a current list of all the things that people in the community are having problems with when using React Native. There are many discussions out there, and this has been done in the past, but let's have a fresh start and make a collection of things that we can then consider looking at in 2019. For now, we'll just make a list. Please don't expect any of these things to be prioritized and fixed right away. For me personally, this is helpful to learn more as I just joined the React Native team.

Built in SVG Images

I'd like to be able to use SVG images without having to convert it to a font or other hacks that have been attempted (webviews, etc).
When using the same code between my web react apps and native, I have to convert images especially ones that are dynamic.

- **Votes:** 115
- **Estimated time to fix:** done
- **Point of contact:** React Native Community
- **Comment by Facebook:** The community has built some well maintained packages to natively support rendering SVGs. We want to enable these packages to iterate and improve and thus have no plans on building it into React Native. We recommend using one of these packages or building your own and sharing it with others.

[참고] Reply from Facebook about "What do you dislike about React Native?

There is nothing new under the sun.

(하늘 아래 새로운 것은 없다.)

SSR?

웹의 등장때 부터 계속해 오던 것.

Response는 기본적으로 초기 화면을 구성하는 마크업 및 자원들로 구성


```
<div>
  <?php
     $name = "<h1 style='color:cyan'>Pengsoo</h1>";
  ?>
  Hello, <?=$name>!.
</div>
```

성능 향상 기법

Minimize HTTP Requests

Combined files are a way to reduce the number of HTTP requests by combining all scripts into a single script, and similarly combining all CSS into a single stylesheet.

현재는? Code-Splitting

[참고] Yahoo Best Practices for Speeding Up Your Web Site

인라인 이벤트 바인딩

```
<button onclick="bgChange()">Press me</button>
```

인라인 이벤트 바인딩은 DOM과의 커플링으로 인한 비유연성
(Presentation과 logic의 결합)은 좋지 않은 패턴

Inline event handlers — don't use these.
- MDN: Introduction to events

현재는?

물론, 프레임워크 내의 이벤트 바인딩 처리를 통한
효율적 처리 적용으로 이전의 처리 방식과는 다르다.

```
<button onClick={handleClick}> // React
<button @click.prevent="follow()"> // Vue.js
<button (click)="toggleFollowing()"> // Angular
```

하지만, 패턴으로만 본다면 DOM과 JS(또한 CSS: CCS-in-JS)는
보다 결합된 형태로 작성되고 있다.

서로 닮아가는 (Inspired by)

XHP: A New Way to Write PHP

... just extract the part that I really liked about Angular
and build something really lightweight.
Between the Wires: An interview with Vue.js creator Evan You

PREACT

... is an attempt to recreate
the core value proposition of React
https://www.npmjs.com/package/preact/v/1.2.0

사실은 항상 하던 것.

새로운 '정의', '용어', '기술' 들이 계속해서 등장하지만,
그러나 알고보면...

Every 10 years we give

new name to what we always do

and ***get excited*** *about it.*

[참고] Anonymous speaker: Java 9 Functional and Reactive programming paradigms

미래 언젠가는 다시금 이런 말을 하게 될수도 있겠죠?

Oh my god no, no one uses React anymore.

You should try learning OOO, it's 20xx.

고맙습니다.
Thank You.
Gracias.