프론트엔드 개발 로드맵

작성자 : 송헌용, 허진, 안미리
소속팀 / 상위부서 : 프론트엔드개발팀 / UIT개발실
대외비

NHN
TECHNOLOGY SERVICES

목차

1.
요즘 프론트엔드를 배우는 기분

요즘 프론트엔드를 배우는 기분

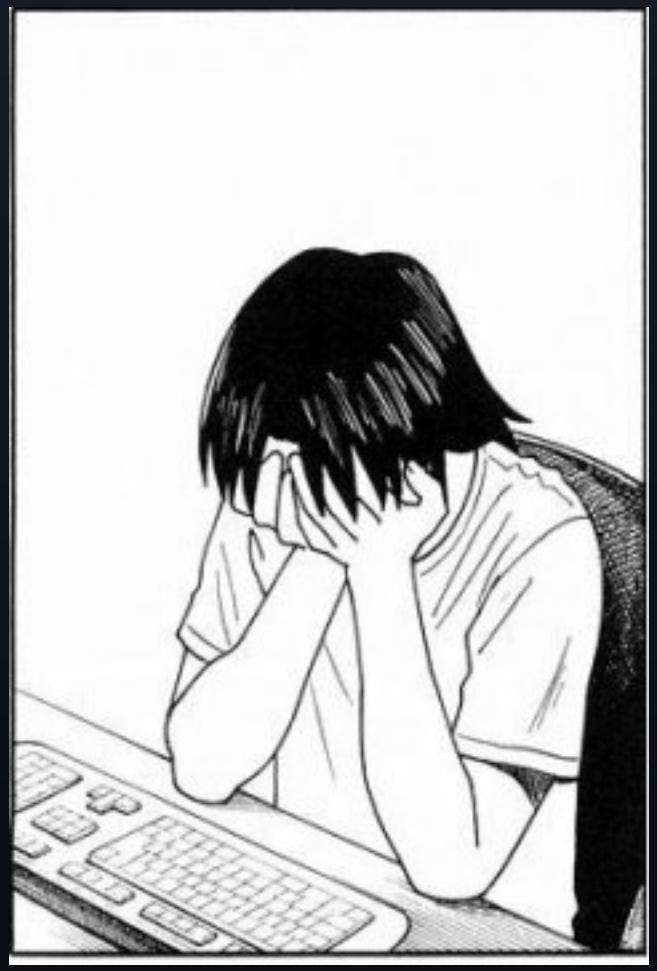
작년에 프론트엔드개발 진영을 뜨겁게 달궜던 글

링크: http://www.looah.com/article/view/2054

대화 속에 나왔던 자바스크립트 개발 관련 키워드

- React, JSX, ECMAScript, AMD, RequireJS, CommonJS, Browserify, NPM, Bower
- Grunt, Gulp, Webpack, TypeScript, Babel, Flow, Fetch, AJAX, Promise, async / await
- Flux, Mustache, underscore, lodash, EJS, Pug, polyfill

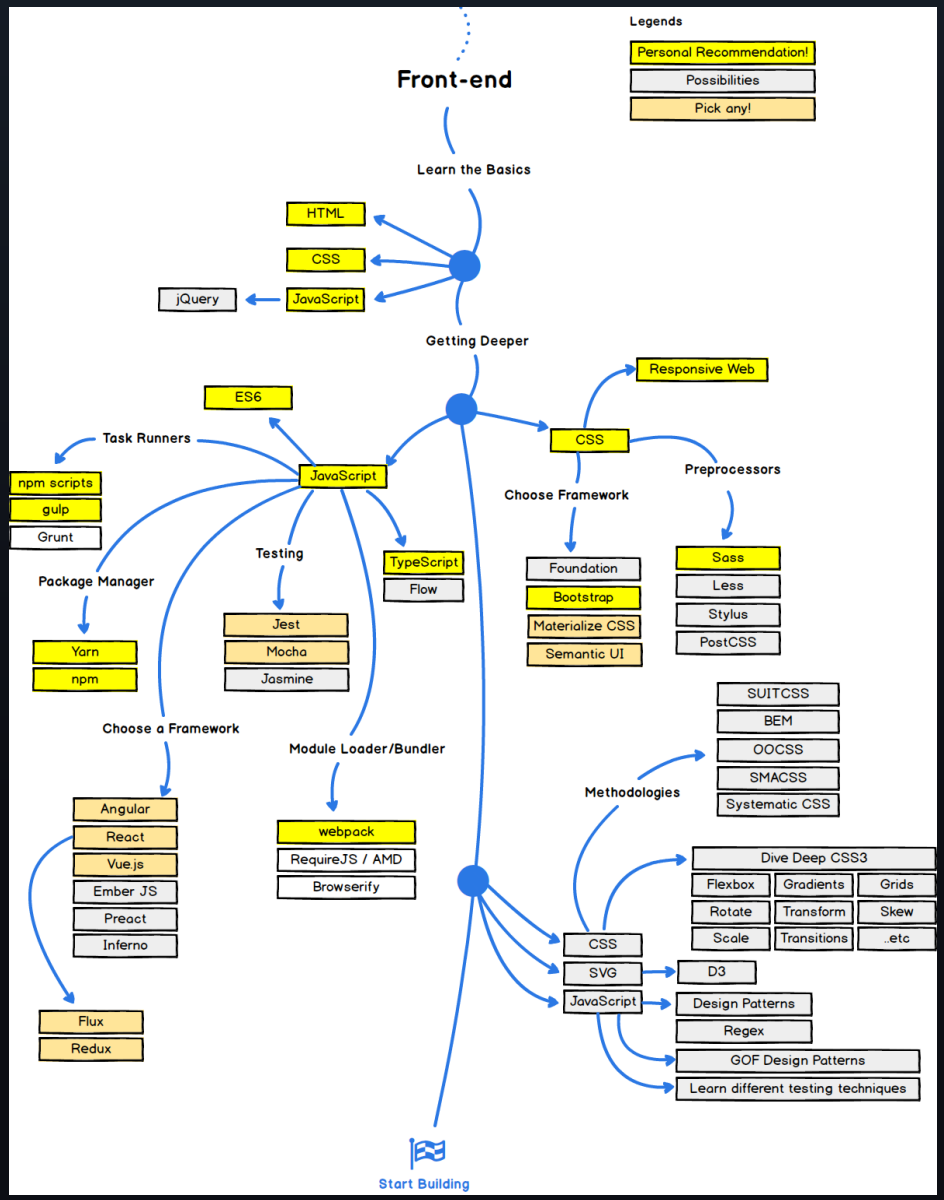
요즘 프론트엔드를 배우는 기분



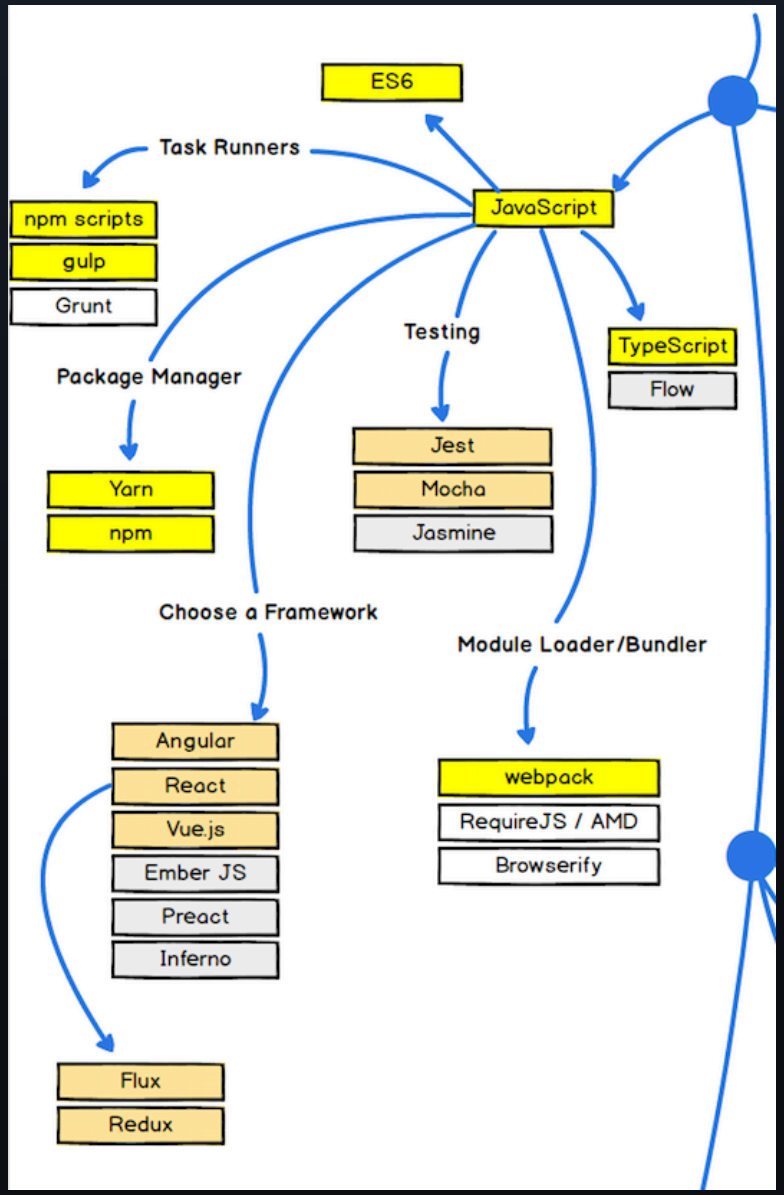
최신 라이브러리만 간단하게 써보고 싶은데 알아야 할 게 너무 많은 현실

2.
개략적인 프론트엔드 개발 변천사

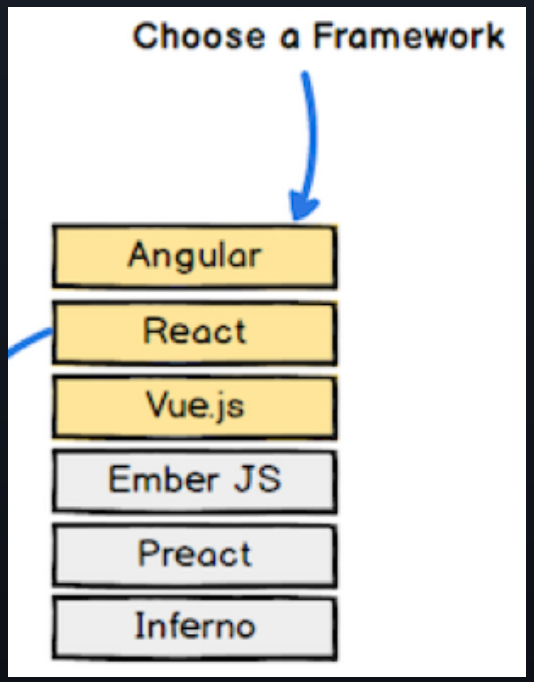
2.1 프론트엔드 개발 로드맵



2.2 자바스크립트 개발 로드맵

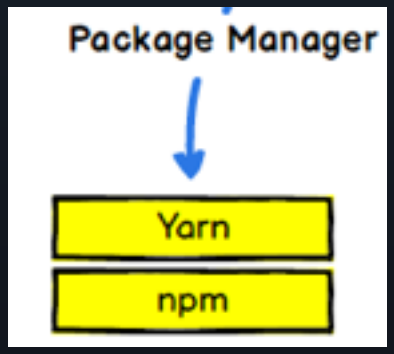


2.3 자바스크립트 프레임워크



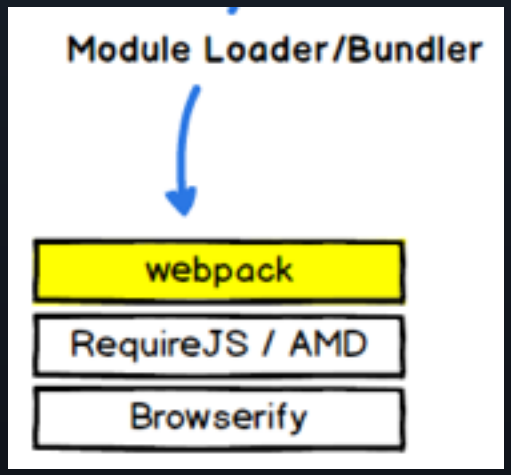
- 구글 덕분에 AJAX가 활성화
- 서버와의 통신이 비동기로 가능
- 개발 방법론이 서버처럼 프레임워크화 됨
- MVC, MVP, MVVM 등 여러 패턴의 라이브러리 & 프레임워크 등장

2.4 패키지 매니저



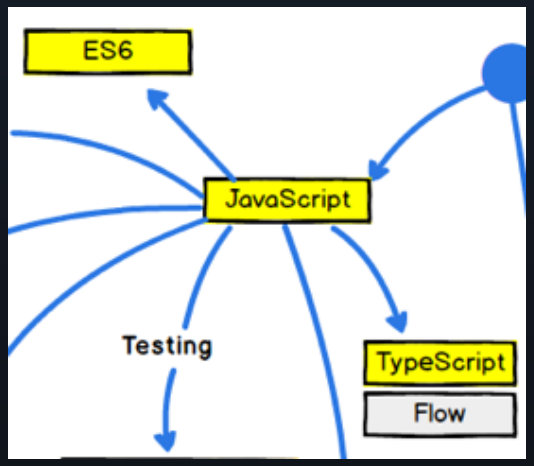
- 동일한 라이브러리 파일이 파편화 되어 한 곳에서 관리할 필요성
- CDN은 버전 관리가 잘 안 되고, 불안정함
- 트위터에서 프론트 소스를 저장하는 Bower가 나옴
- Node.js의 라이브러리 저장소인 NPM 으로 통일되는 분위기

2.5 모듈 로더 / 번들러



- 공통 소스를 다른 곳에서 쓰일 수 있게 라이브러리로 배포
- 모듈의 형태에 따라 CommonJS / AMD / UMD로 분리
- CommonJS 방식은 번들러로 컴파일 과정이 필요함
- AMD는 브라우저에서 실시간으로 동적 로딩
- UMD는 두 가지 환경에 모두 만족하는 방식

2.6 ES6, TypeScript, Flow



- Node.js가 인기를 끌면서 다른 진영의 개발자들이 자바스크립트로 넘어옴
- 언어 자체가 너무 유연하다보니 공통 라이브러리에서 컨벤션이 너무 다름
- ECMAScript 6 Edition으로 구조를 튼튼하게 개발할 수 있도록 지원
- 변수의 데이터 타입을 체크해서 일관성 있는 코드를 지향

3.
ES2016 -> ES2017 -> ES2018

3.1 현재 확정된 스펙

tc39 / proposals

Watch 544 | Star 1,814 | Fork 67

Code | Issues 2 | Pull requests 1 | Projects 0 | Insights

Branch: master ▾ proposals / finished-proposals.md — Find file | Copy path

mathiasbynens Reformat finished/inactive/stage 0 proposal tables — bf8a1c6 on 2 May

3 contributors

18 lines (14 sloc) | 3.19 KB — Raw | Blame | History

Finished Proposals

Finished proposals are proposals that have reached stage 4, and are included in the latest draft of the specification.

Proposal	Champion(s)	TC39 meeting notes	Expected Publication Year
`Array.prototype.includes`	Domenic Denicola, Rick Waldron	November 2015	2016
Exponentiation operator	Rick Waldron	January 2016	2016
`Object.values` / `Object.entries`	Jordan Harband	March 2016	2017
String padding	Jordan Harband & Rick Waldron	May 2016	2017
`Object.getOwnPropertyDescriptors`	Jordan Harband & Andrea Giammarchi	May 2016	2017
Trailing commas in function parameter lists and calls	Jeff Morrison	July 2016	2017
Async functions	Brian Terlson	July 2016	2017
Shared memory and atomics	Lars T Hansen	January 2017	2017
Lifting template literal restriction	Tim Disney	March 2017	2018

See also the stage 0 proposals, active proposals, and inactive proposals documents.

3.2 ES2016 (ES7)

- ES2015의 마이너 패치
- Array.prototype.includes
- ** 거듭 제곱 연산자 추가

3.3 ES2017 (ES8)

- Object.values
- Objec.entries
- String.prototype.padStart
- String.prototype.padEnd
- Object.getOwnPropertyDescriptors
- Trailing commas in function parameter lists and calls

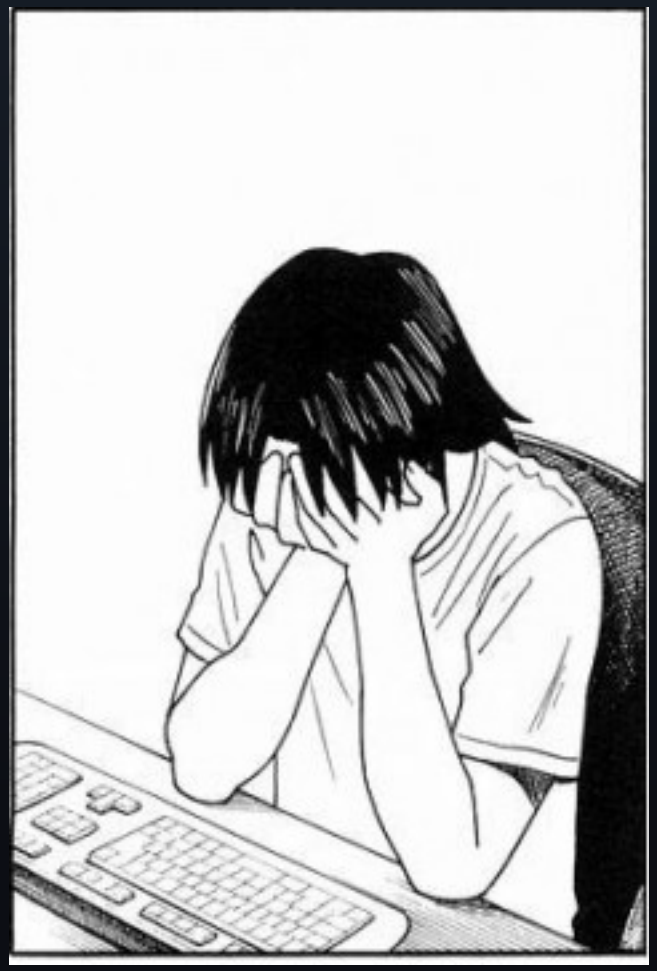
- Async functions: 비동기 함수 처리: Promise와 Generator를 결합한 추상화 형태
- Shared memory : 메인 스레드와 워크 스레드간 작업을 공유해서 병렬 프로그래밍 지원
- atomics: 동시성 문제를 해결하기 위해 지원

3.4 ES2018 (ES9)

- 문자열 템플릿 기능 수정

- 아직은 1개만 확정인데…
  - SIMD.js (Single Instruction, Multiple Data): 하나의 명령으로 여러 개의 데이터를 병렬로 처리

자료조사를 하던 저의 기분



작년에 ES2015를 썼는데 벌써 ES2018이라니…

4.
TypeScript & Flow

4.1 TypeScript

- MicroSoft에서 밀고 있음
- 데이터 타입을 지정하는 형태. 생산성보다 안정성을 중요시
- Abstract Class, Interface 등 OOP 개념을 지원해서 ES의 superset으로 불림
- Angular에서 Typescript를 지원 (마이크로소프트 + 구글)

- 컴파일러 단계가 필요함
- IDE에 컴파일러를 연동하면 실시간 체크 가능

4.2 Flow

- Facebook에서 밀고 있음
- 리액트에만 사용되는 게 아니라 네이티브 자바스크립트에서도 사용 가능
- Typescript는 심화 개념이 많아서 진입장벽이 높음
- 타입만 체크해주는 라이브러리

- 컴파일러 단계가 필요함
- IDE에 컴파일러를 연동하면 실시간 체크 가능

5.
Package Manager

5.1 NPM

- Node Package Manager의 줄임말
- Ruby의 Gem Java의 Maven과 같이 라이브러리 의존성 관리 모듈
- 예전에는 Node.js와 별도였는데 현재는 Node.js를 설치하면 자동 포함
- 현재는 버전5까지 배포된 상태
- 버전이 올라가면서 로컬에 설치되는 라이브러리 최적화

5.2 Yarn

- 페이스북, 구글 등 유명 엔지니어들이 NPM의 핵심 이슈를 해결하기 위해 개발
- 다운로드 과정을 병렬로 처리하기 때문에 빠름
- 한 번 설치한 건 캐쉬로 가지고 있기 때문에 빠름
- 한 번 설치한 모듈은 오프라인 설치 가능
- 요즘은 NPM + Yarn을 함께 사용하는 추세

6.
Task Runner

Task Runner 란?

파일을 수정할 때마다 반복해야하는 일련의 작업들을 자동화해주는 도구 (= 빌드 도구)

(예시)

- LESS 또는 SASS 파일을 CSS로 컴파일하기
- 자바스크립트나 CSS 파일들을 Minify, Uglify하기 (압축)
- 자바스크립트가 코딩 규칙을 따르는지 JSHint 툴로 검사하기

효과 : 반복작업을 하지않아도 되므로 온전히 개발에만 집중 가능 -> 생산성 향상

종류 : Grunt, Gulp, npm script 등

Grunt

특징

- JSON 형식을 통한 선언적 설정 기반으로 Task 정의
- 설정파일에 정의한 Task들을 자동화

```
//Gruntfile.js
module.exports = function(grunt) {
  grunt.initConfig({
    uglify: {
      build: {
        src:  'js/build/app.js',
        dest: 'js/build/app.min.js'
      }
```

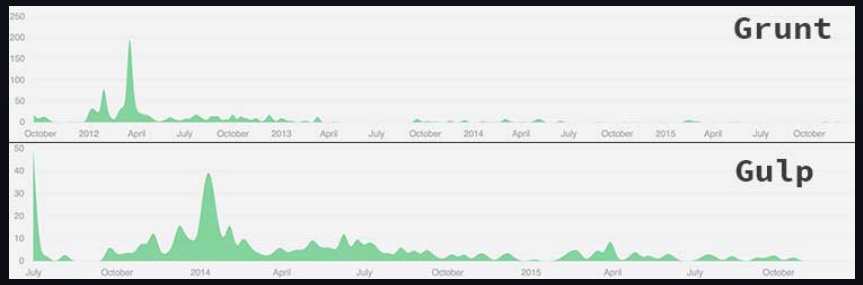
원본 파일 경로 ←

결과 파일 경로 ←

```
grunt.loadNpmTasks('grunt-contrib-uglify');
```

한계

- 유지보수가 제대로 안되면서 유저들이 등을 돌림



Gulp

특징

- Node.js의 스트림(stream)을 기반으로 Task 선언
  - 스트림(stream)?
    작업 요청 후 한번에 결과를 받는 것이 아니라,
    이벤트 중간중간 전달 받는 방식
- 속도가 비교적 빠름
- 직관적이고 친근한 자바스크립트 문법

npm script

특징

- npm만 설치되어 있다면 다른 빌드 도구를 따로 설치하지 않아도 됨
- 간단한 Task를 처리하기에 적합

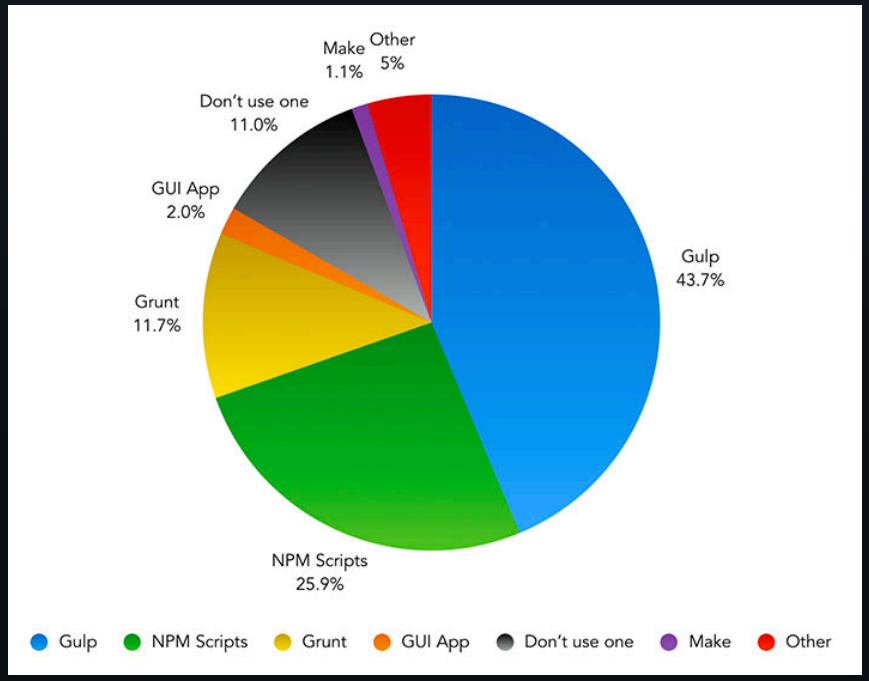
```
➜  ~ npm install uglify-js
```

```
{
  ...
  "script": {
    "uglify": uglifyjs src/app.js -m -c -o dist/app.min.js
  }
}
```

```
➜  ~ npm run uglify
```

뭘 사용할지 고민된다면?

- JSON형식이 익숙하고 다양한 플러그인을 사용하고 싶다면 -> Grunt
- 자바스크립트 코드에 익숙하고 빠르게 여러가지 일을 처리하고 싶다면 -> Gulp
- 따로 빌드 도구를 설치하기 싫고, 간단하게 몇 개 Task만 처리하고 싶다면 -> npm script



2016 Task Runner 선호도

7.
Module Loader / Bundler

모듈?

- 구현 내용을 다른 코드에서 재 사용할 수 있게 만든 코드 조각
- 코드의 재 사용을 편리하게하고, 유지보수를 쉽게 함
- 라이브러리 구성
- C언어의 #include, JAVA의 import, CSS의 import

모듈 포맷

- 모듈을 정의하기 위해 사용할 수 있는 문법

commonJS

- require와 exports로 의존성과 모듈을 정의함.
- Node.js 문법

AMD

- 비동기 모듈 정의. 모듈을 백엔드뿐 아니라 브라우저에서도 사용 가능
- define 함수를 사용해서 모듈 정의
- Node.js가 인기를 얻으며 거의 사용되지않음

모듈 포맷

UMD

- 라이브러리를 배포할 때 사용자가 어떤 방식으로 사용할지 모름
  -> 두가지 방법이 모두 가능하도록 배포
- 조건문으로 구분해서 commonJS와 AMD방식 모두 사용

ES6 내장 모듈

- ES6부터는 자바스크립트에 모듈이 내장됨
- import와 export로 모듈을 내보내고 사용
- 아직 모든 브라우저에서 사용할 수 없으므로, Babel같은 트랜스파일러가 필요

Module Loader / Bundler

- 자바스크립트는 #include, import같이 파일끼리 의존성을 관리해줄 수 있는 방법이 없음

```
<script src="main.js"></script>
<script src="module1.js"></script>
<script src="module2.js"></script>
<script src="module3.js"></script>
<script src="module4.js"></script>
<script src="module5.js"></script>
```

- 여러개의 자바스크립트 파일들 서로 간의 의존성/종속성 관리를 할 수 있음

Module Loader / Bundler

Loader

- 런타임에 실행
- 브라우저를 로드할 때 해당 모듈을 필요한 시점에 로드
- 예) RequireJS, SystemJS

Bundler

- 빌드타임에 실행
- 빌드할때 번들파일을 생성하고 브라우저에서 번들 파일를 로드
- 예) Browserify, webpack

webpack !

- Module Bundler 역할 + 로더로 Task Runner 역할
- 요즘 대세

8.
Framework & Library

8.1 프레임워크 & 라이브러리

- 라이브러리: 공통적인 함수, 기능 등을 모아둔 “도구”
  - 코드 내에서 개발자가 주체적으로 사용
- 프레임워크: 코드 규칙, 진행 방식 등을 모아둔 “틀”
  - 완성된 틀에 내용물(코드)를 채워 넣음
- 연장(라이브러리)으로 가구(코드)를 만들고, 집(프레임워크)에 채워넣는 관계

8.2 jQuery

- Ajax통신과 DOM조작 등의 코드를 단순화시킨 Javascript 라이브러리

- IE로 통합되었던 브라우저가 파이어폭스, 크롬 등이 제작되며 분화
    - 크로스 브라우징 이슈가 발생

- 크로스 브라우징을 해결하기 위한 복잡한 조건문 작업을 jQuery가 대신함

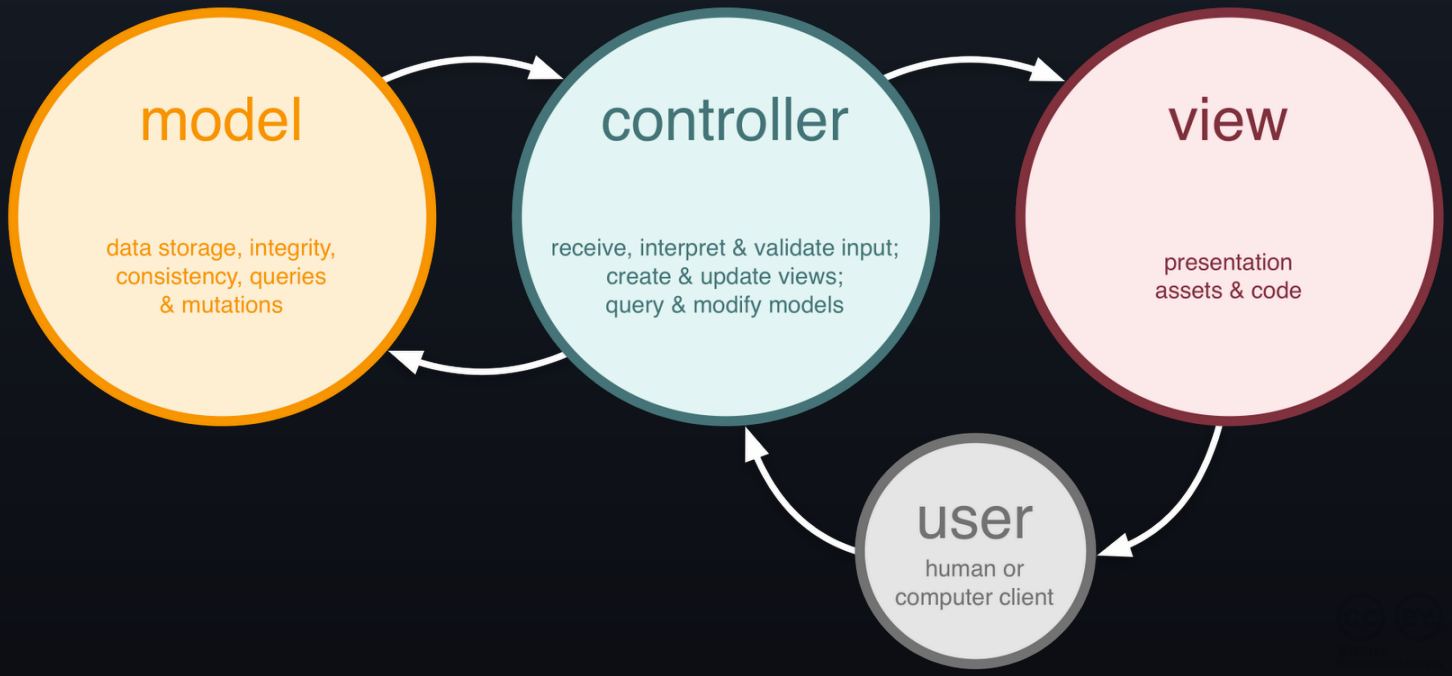
- DOM 조작 과정을 브라우저 관계없이 추상화
- Ajax 사용으로 single-page app이 유행(gmail 등)

8.3 프레임워크 등장배경

- 사용자 클라이언트의 성능이 향상
  - 다양한 기능 요구
  - 로직의 복잡성 증가
- jQuery는 구조를 제공하지 않음
- 유지보수가 어려움

- 구조를 제공하는 프레임워크 등장
- 기존 소프트웨어 공학 이론을 참고하여 MV* 패턴 적용

8.4 MV*?


model
data storage, integrity, consistency, queries & mutations
controller
receive, interpret & validate input; create & update views; query & modify models
view
presentation assets & code
user
human or computer client

8.5 backbone.js

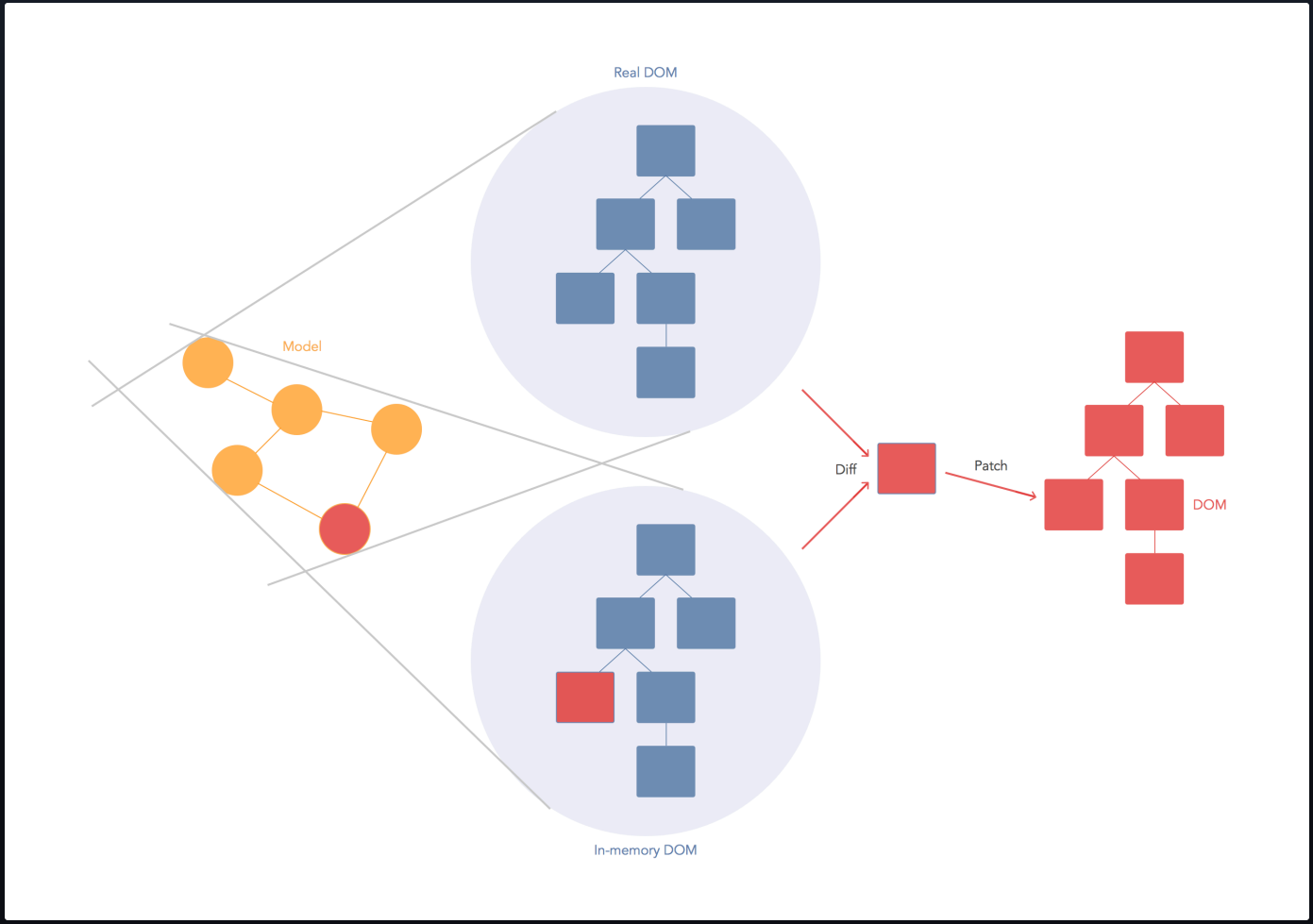
- Model과 View를 분리하는 프레임워크
- Underscore.js와 jQuery 사용
  - Underscore.js: Model에 대한 Utility 함수를 제공하는 라이브러리
    - Lodash: underscore를 개선, 기능을 추가한 라이브러리
  - jQuery를 사용하던 개발자도 구조만 정리하면 되도록
- 둘을 연결하는 코드를 유연성 있게 작성할 수 있음
- 유지보수는 어려운 채로 남음

8.6 Angular.js

- 완성된 MVW(Whatever) 구조를 제공
- 복잡한 앱을 만들 때 적합
- 양방향 데이터 바인딩
    - Model의 Data를 순회하면서 DOM과 다른 부분이 있는지 확인
    - 순회 과정에서 성능 저하가 발생
- 이후 알고리즘과 구조를 개선하여 성능이 좋아진 Angular로 재탄생

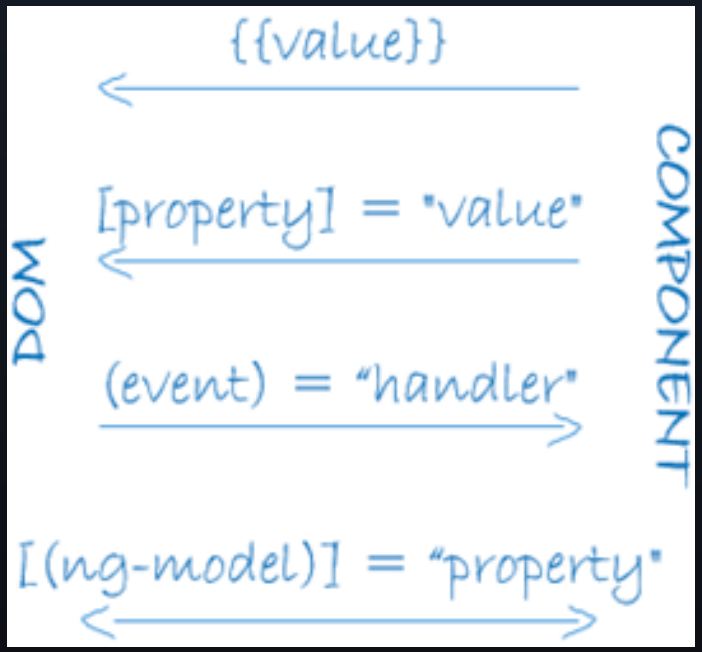
8.7 React.js

- MV* 패턴의 V만을 위해 만들어짐
  - 엄밀하게는 라이브러리로 구분
  - Redux나 Router등의 보조 라이브러리 필요
- 컴포넌트 단위 UI
- 서버 사이드 렌더링으로 렌더링 속도 상승
- 가상 DOM을 사용한 단방향 데이터 바인딩으로 성능 향상


Real DOM
Model
Diff
Patch
DOM
In-memory DOM

8.8 Angular

- Angular.js의 컨셉을 유지하고 단점을 개선한 새로운 프레임워크 (not 업데이트)
- typescript 지원
  - 코드 가독성 증가
  - 리팩토링 및 디버깅 효율 증가
- 단방향 바인딩 지원
  - 프로퍼티 바인딩: Model -> DOM
  - 이벤트 바인딩: DOM -> Model
- 양방향 바인딩 알고리즘 개선: 단방향 바인딩으로 분리
- 서버 사이드 렌더링 도입



-

End of Document

-

Thank You.

-