

## SFC

**SFC**

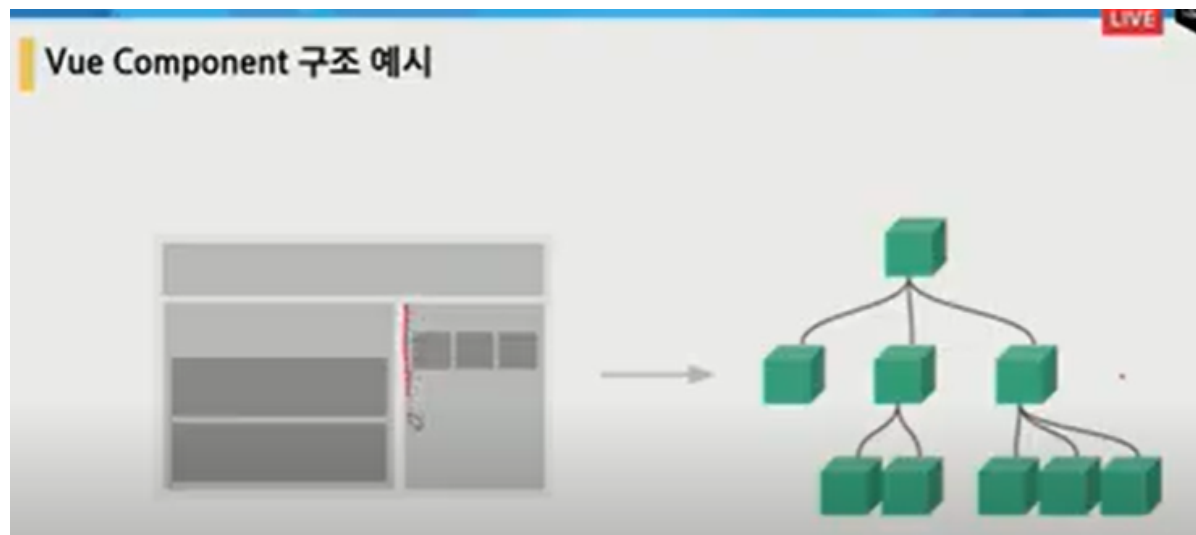
### Component (컴포넌트)

- 기본 HTML 엘리먼트를 확장하여 **재사용** 가능한 코드를 캡슐화 하는데 도움을 줌
- CS에서는 **다시 사용할 수 있는** 범용성을 위해 개발된 소프트웨어 구성 요소를 의미
- 즉, 컴포넌트는 개발을 함에 있어 유지보수를 쉽게 만들어 줄 뿐만 아니라, 재사용성의 측면에서도 매우 강력한 기능을 제공
- Vue 컴포넌트 === Vue 인스턴스

```
const app = new Vue()
```

이렇게 했을 때 app이 component

## 구조



쪼개면 쪼갤수록 유지 보수에 용이

상하관계가 있음(큰 네모 안에 작은 네모 여러개)

## SFC

하나의 파일(vue)이 하나의 컴포넌트

## SFC (Single File Component)

- Vue의 컴포넌트 기반 개발의 핵심 특징
- 하나의 컴포넌트는 .vue라는 하나의 파일 안에서 작성되는 코드의 결과물
- 화면의 특정 영역에 대한 HTML, CSS, JavaScript 코드를 하나의 파일(.vue)에서 관리
- 즉, .vue 확장자를 가진 싱글 파일 컴포넌트를 통해 개발하는 방식
- Vue 컴포넌트 === Vue 인스턴스 === .vue 파일

## Vue CLI

command line, 명령어

### Vue CLI

- Vue.js 개발을 위한 표준 도구
- 프로젝트의 구성을 도와주는 역할을 하며 Vue 개발 생태계에서 표준 tool 기준을 목표로 함
- 확장 플러그인, GUI, ES2015 구성 요소 제공 등 다양한 tool 제공

## Node.js

Vue cli를 사용하려면 node.js를 설치해야함

자바는 브라우저 안에서만 동작한다는 한계가 있었음

노드.js가 탄생하면서 serversiderendering이 가능해지고 지금같은 인기를 갖게된 계기

### Node.js

- 자바스크립트를 브라우저가 아닌 환경에서도 구동할 수 있도록 하는 자바스크립트 런타임 환경
  - 브라우저 밖을 벗어 날 수 없던 자바스크립트 언어의 태생적 한계를 해결
- Chrome V8 엔진을 제공하여 여러 OS 환경에서 실행할 수 있는 환경을 제공
- 즉, 단순히 브라우저만 조작할 수 있던 자바스크립트를 SSR에서도 사용 가능하도록 함
- 발표
  - 2009년 Ryan Dahl

## NPM (Node Package Manage)

- 자바스크립트 언어를 위한 패키지 관리자
  - Python의 pip가 있다면 Node.js 에는 NPM
  - pip와 마찬가지로 다양한 의존성 패키지를 관리
- Node.js의 기본 패키지 관리자
- Node.js와 함께 자동으로 설치 됨

### Vue CLI 설치

#### Vue CLI 설치

- vue-cli 설치
  - `$ npm install -g @vue/cli`
- 버전 확인
  - `$ vue --version`
- 프로젝트 생성
  - `$ vue create my-first-vue-app`
- run server
  - `$ npm run serve`

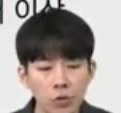
### 설치

vue 3이 릴리즈 되긴 했지만 vue 2로 설치

## Babel

### Babel

- JavaScript Transcomiler
- 자바스크립트의 신버전 코드를 구버전으로 번역/변환 해주는 도구
- 자바스크립트 역사에 있어서 파편화와 표준화의 영향으로 작성된 코드의 스펙트럼이 매우 다양
  - 최신 문법을 사용해도 브라우저의 버전별로 동작하지 않는 상황이 발생
  - 같은 의미의 다른 코드를 작성하는 등의 대응이 필요해졌고 이러한 문제를 해결하기 위한 도구
- 원시 코드(최신 버전)를 목적 코드(구 버전)으로 옮기는 번역기가 등장하면서 개발자는 더 이상 내 코드가 특정 브라우저에서 동작하지 않는 상황에 대해 크게 고민하지 않을 수 있음



### Babel 동작 예시

JavaScript

Copy

```
// Babel Input: ES2015 arrow function
[1, 2, 3].map((n) => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

원시 코드

목적 코드

compile

## Webpack

### Webpack

- static module bundler
- 모듈 간의 의존성 문제를 해결하기 위한 도구

## Module

- 모듈은 단지 파일 하나를 의미 (ex. 스크립트 하나 === 모듈 하나)
- 배경
  - 브라우저만 조작할 수 있었던 시기의 자바스크립트는 모듈 관련 문법 없이 사용 되어짐
  - 하지만 자바스크립트와 애플리케이션이 복잡해지고 크기가 커지자 전역 스코프를 공유하는 형태의 기존 개발 방식의 한계점이 드러남
  - 그래서 라이브러리를 만들어 필요한 모듈을 언제든지 불러오거나 코드를 모듈 단위로 작성하는 등의 다양한 시도가 이루어짐
- 과거 모듈 시스템
  - AMD, CommonJS, UMD
- 모듈 시스템 2015년 표준으로 등재 되었으며 현재는 대부분의 브라우저와 Node.js가 모듈 시스템을 지

## Module 의존성 문제

- 모듈의 수가 많아지고 라이브러리 혹은 모듈 간의 의존성(연결성)이 깊어지면서 특정한 곳에서 발생한 문제가 어떤 모듈 간의 문제인지 파악하기 어려워짐 (의존성 문제)
- Webpack은 모듈 간의 의존성 문제를 해결하기 위해 존재하는 도구

## Bundler

- 모듈 의존성 문제를 해결해주는 작업이 Bundling이고 이러한 일을 해주는 도구가 Bundler이고, Webpack은 다양한 Bundler 중 하나
- 모듈들을 하나로 묶어주고 묶인 파일은 하나(혹은 여러 개)로 만들어짐
- Bundling된 결과물은 더 이상 서-순에 영향을 받지 않고 동작하게 됨
- Bundling 과정에서 문제가 해결되지 않으면 최종 결과물을 만들어 낼 수 없기 때문에 유지 & 보수의 측면에서도 매우 편리해짐
  - snowpack, parcel, rollup.js 등의 webpack 이외에도 다양한 모듈 번들러 존재
- Vue CLI는 이러한 Babel, Webpack에 대한 초기 설정이 자동으로 되어 있음

## 의존성 문제 예시

```
<!-- index.html -->
```

```
<body>
```

```
...
```

```
<script src="a.js"></script>
```

```
<script src="b.js"></script>
```

```
</body>
```

```
// a.js
```

```
const a = 1  
console.log(a)
```

```
// b.js
```

```
a = 2
```

const는 재할당, 재선언이 불가능하므로 오류 발생

src:webpack에 빌드되는 정적 파일들

components: 하위 컴포넌트가 들어감

app.vue: 최상위 컴포넌트

main.js: webpack이 빌드를 시작할때 가장 먼저 불러오는 시작점, 뷰의 전역에서 사용하는 모듈들을 등록

gitignore은 기본적으로 만들어줌

gitinit을 안했는데 자동으로 master가 생김(git 초기화가 저절로 생김)

package.json: requirements.txt와 비슷한 역할, 우리가 건들일 x

package-lock.json: 팀원들과 동일한 환경(종속성)을 유지하기 위한 파일, 우리가 건들일 x

### app.vue

template(html) - script(js) - style(css)

.vue가 하나의 컴포넌트



```
3   
4   <HelloWorld msg="Welcome to Your Vue.js App"/>
5 </div>
6 </template>
7
8 <script>
9   import HelloWorld from '../components/HelloWorld.vue'
10
11   export default {
12     name: 'App',
13     components: {
14       HelloWorld
15     }
16   }
```

app.vue와 하위 컴포넌트의 연결 확인

불러와서 - 등록하고 - 사용

```
1 <template>
2   <div id="app">
3     
4     <HelloWorld msg="Welcome to Your Vue.js App"/>
5   </div>
6 </template>
7
8 <script>
9   import HelloWorld from '../components/HelloWorld.vue'
10
11   export default {
12     name: 'App',
13     components: {
14       HelloWorld
15     }
16   }
17 </script>
```

style scope 뜻

이 스타일은 딱 이 .vue에서 한정되게 동작한다는 뜻

```
36   props: {
37     msg: String
38   }
39 }
40 </script>
41
42 <!-- Add "scoped" attribute to limit CSS to this component only -->
43 <style scoped>
44   h3 {
45     margin: 40px 0 0;
46   }
47   ul {
48     list-style-type: none;
49     padding: 0;
50   }
51 </style>
```

## component 만들기

new component 만들

vue 하고 엔터하면 3등분 자동완성

template 안에는 하나의 최상위 태그가 있다

```
1 <template>
2   <div>
3     <h2>New Component!</h2>
4   </div>
5 </template>
6
```

app 이름과 동일하게 name 설정

꼭 그럴 필요는 없지만 그러지 않을 이유도 없음

```
6
7 <script>
8   export default {
9     name: 'NewComponent',
10  }
11 </script>
12
```

### 1. 불러오기

```
7
8 <script>
9   import HelloWorld from './components/HelloWorld.vue'
10  import NewComponent from './components/NewComponent.vue'
11
```

### 2. 등록하기

```
8 <script>
9   import HelloWorld from './components/HelloWorld.vue'
10  // 1. 불러오기
11  import NewComponent from './components/NewComponent.vue'
12
13  export default {
14    name: 'App',
15    components: {
16      HelloWorld,
17      NewComponent
18    }
19  }
```

### 3. 사용하기(달는 태그 작성)

```
App.vue
1 <template>
2   <div id="app">
3     
4     <HelloWorld msg="Welcome to Your Vue.js App"/>
5     <!-- 3. 보여주기 -->
6     <NewComponent/>
7   </div>
8 </template>
```



