|  |
| --- |
| **Vue.js** |

|  |  |
| --- | --- |
| 적용 분야 | S/W |
| 소속 부서 | 의료영상SW팀 |
| 작성 일자 | 2020. 06. 12. |
| 작성자 | 김재혁 |
| 보안 등급 |  |
| 기여율 | 100% |

목차

[1 개요 3](#_Toc65737149)

[2 연구(검토) 목적 4](#_Toc65737150)

[3 연구(검토) 세부 내용 4](#_Toc65737151)

[3.1 Vue.js 소개 4](#_Toc65737152)

[3.1.1 개요 4](#_Toc65737153)

[3.1.2 MVVM 패턴 4](#_Toc65737154)

[3.1.3 설치하기 5](#_Toc65737155)

[3.1.4 Vue2 🡪 Vue3 9](#_Toc65737156)

[3.2 Vue CLI 9](#_Toc65737157)

[3.2.1 개요 9](#_Toc65737158)

[3.2.2 CLI Service 10](#_Toc65737159)

[3.2.3 CLI Plugins 10](#_Toc65737160)

[3.2.4 Commands 10](#_Toc65737161)

[3.3 NPM 11](#_Toc65737162)

[3.3.1 개요 11](#_Toc65737163)

[3.3.2 Command 11](#_Toc65737164)

[3.4 개발 12](#_Toc65737165)

[3.4.1 준비 및 환경 12](#_Toc65737166)

[3.4.2 01) Hello World Page 작성 14](#_Toc65737167)

[3.4.3 02) 조건문 15](#_Toc65737168)

[3.4.4 03) 반복문 15](#_Toc65737169)

[3.4.5 04) 사용자 입력 16](#_Toc65737170)

[3.4.1 05) Component 17](#_Toc65737171)

[3.4.2 06) Instance 18](#_Toc65737172)

[3.4.3 07) Template 22](#_Toc65737173)

[3.4.4 08) Computed Property 25](#_Toc65737174)

[3.4.5 09) Class and Binding 28](#_Toc65737175)

[3.4.6 10) Conditional Rendering 29](#_Toc65737176)

[3.4.7 11) List Rendering 31](#_Toc65737177)

[3.4.8 12) Event Handling 34](#_Toc65737178)

[3.4.9 13) Form Input Binding 38](#_Toc65737179)

[4 참고 문헌 및 자료 42](#_Toc65737180)

[4.1 Vue CLI 42](#_Toc65737181)

[4.2 환경 구성 42](#_Toc65737182)

[4.3 HTML 42](#_Toc65737183)

[4.4 환경 구성 42](#_Toc65737184)

# 개요

|  |  |  |  |
| --- | --- | --- | --- |
| 문서 구분 | ■ 기술 보고 □ 선행 조사 □ 경쟁사 분석 □ 기타 | | |
| 개발 구분 | ■ 개발용 □ 양산용 □ 선행 검토용 ■ 교육용 □ 기타 | | |
| 적용 분야 | ■ 공통 □ 의료기기 □ 산업기기 □ 신사업 □ 기타 | | |
| □ HW ■ SW □ FPGA/FW □ 광학 □ 기구 □ 기타 | | |
| 적용 모델 | QXLink | | |
| 보고서 이름 | Vue.js | | |
| 작성일 | 2020. 03. 29. | 작성 부서 | SW팀 |
| 검토 기간 | 2020. 01. 02. ~ 2020. 03. 8. | | |
| 시험 기간 | 2020. 03. 08. ~ 2020. 03. 29. | | |
| 시험 시료 |  | | |
| 시험 장비  프로그램 | Visual Studio Code, Vue.js | | |
| 키워드 | Web Development, Font-End, Vue.js | | |
| 요약 정리 | d | | |

# 연구(검토) 목적

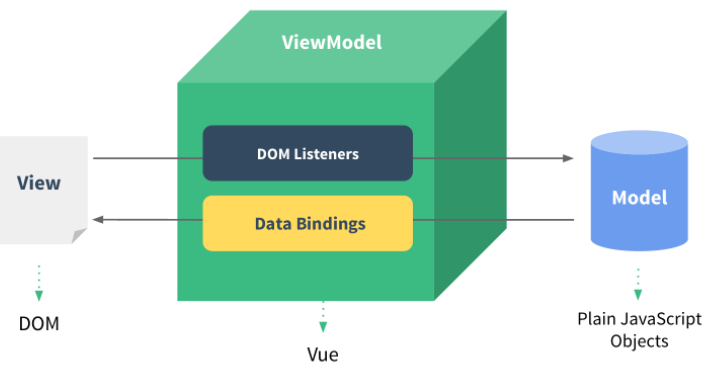
영상의

# 연구(검토) 세부 내용

## Vue.js 소개

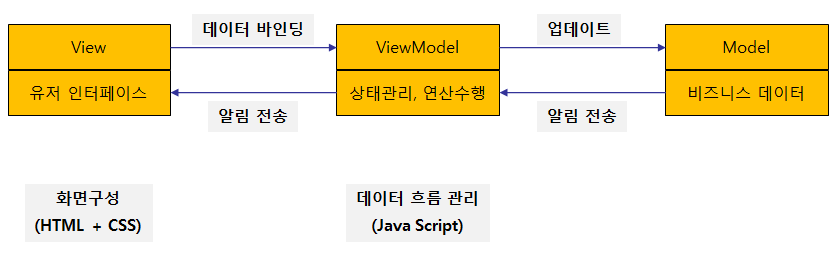
### 개요

* 가장 빠르게 발전하고 확산되고 있는 **Javascript Framework**

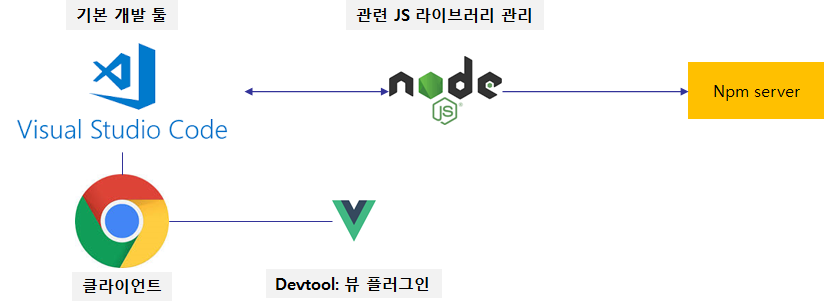


* Data binding과 화면 단위를 Component 형태로 제공하며, 관련 API를 지원
* Angular에서 지원하는 양방향 Data binding을 동일하게 제공
* Component간 통신의 기본 결격은 React의 단방향 데이터 흐름(부모 🡪 자식)을 사용
* 다른 Front-End Framework(Angular, React)와 비교했을 때 상대적으로 **가볍고 빠름**
* 문법이 단순하고 간결하여 **초기 합승 비용이 낮음**

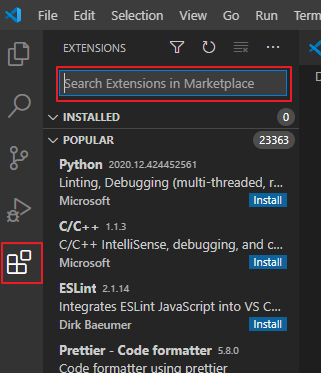
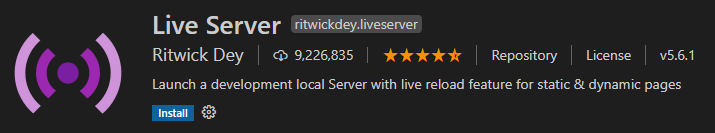
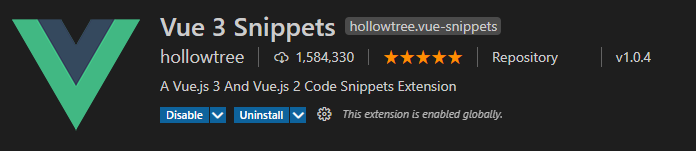
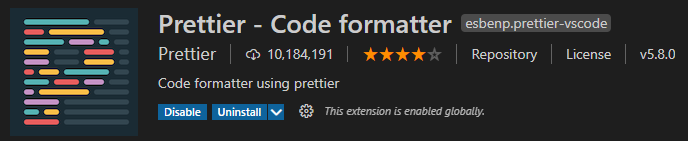
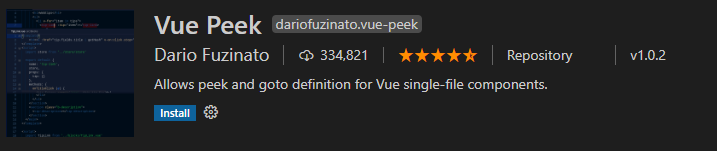
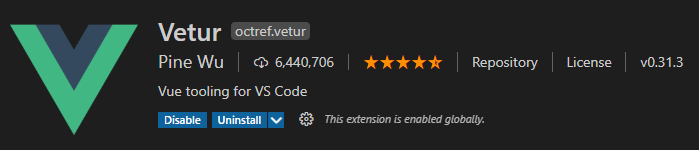
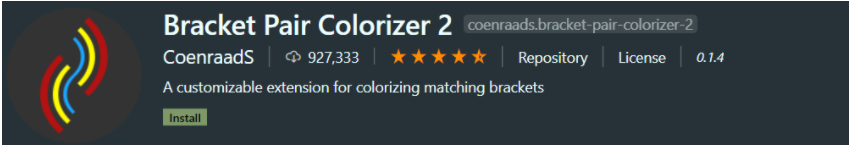
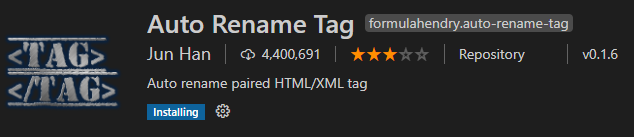
### MVVM 패턴

* Model – View – View Model 패턴
* Back-End 로직과 Client의 Markup & Display 단을 분리하기 위한 구조로 전통적인 MVC 패턴의 방식에서 기인
* Front-End의 표시 동작 로직과 Back-End의 DB Data 처리 및 Server Logic을 분리하고, Back-End에서 넘어온 Data를 Model에 담아 View로 넘겨주는 중간 지점  
  
* Operation
  + View
    - 사용자가 어떤 입력 값을 변경하게 되면 VM에게 Data binding에 대한 요청을 보낸다.
  + Model
    - 비즈니스 데이터를 관리하다가 값이 변경되면 ViewModel에게 변경 사항을 알린다.
  + ViewModel
    - Model의 변경 내용을 화면인 View에게 알려서 반영하여, 사용자가 내용을 확인할 수 있게 한다.
    - ViewModel: 실제 Model의 값을 변경하여 Data의 일관성을 유지한다.
* 구성
  + View: HTML과 CSS로 이뤄진 코드이며, Vue.js에서 제공하는 Directive(지시문)라는 것을 사용하여 내용을 삽입한다.
  + Model은 일반적으로 서버에 전달된 값을 Javascript object 형태로 사용한다.
  + View Model은 Vue.js에서 제공하는 Vue Object 이다.

### 설치하기

* 목표 개발환경  
  
  + 개발 툴로는 Visual Studio Code를 사용할 예정이다.
  + 다양한 3rd party javascript library들을 설치하게되는데 SFC(Single File Component) 방식으로 개발하기 위해서는 CDN보다는 Library를 설치하는 방식을 사용한다.
* 사전 설치 필요 프로그램
  + npm
  + node.js (V 8.9 이상)
    - <https://nodejs.org/ko/download/>
* Vue.js
  + 설치 Guide
    - <https://kr.vuejs.org/v2/guide/installation.html>
    - <https://nm817.tistory.com/35>
  + Build Types
    - NPM 패키지(C:\Users\2017387\AppData\Roaming \npm\node\_modules\vue\)의 dist Directory에는 다양한 빌드가 있으며 그 차이점의 개요는 다음과 같다.

|  | **UMD** | **CommonJS** | **ES Module  (for bundlers)** | **ES Module  (for browsers)** |
| --- | --- | --- | --- | --- |
| **Full** | vue.js | vue.common.js | vue.esm.js | vue.esm.browser.js |
| **Runtime-only** | vue.runtime.js | vue.runtime.common.js | vue.runtime.esm.js | - |
| **Full (production)** | vue.min.js | - | - | vue.esm.browser.min.js |
| **Runtime-only (production)** | vue.runtime.min.js | - | - | - |

* + 경로: C:\Users\2017387\AppData\Roaming\npm\node\_modules\vue
* Visual Studio Code
  + Download: <https://code.visualstudio.com/Download>
  + Extension 설치
    - 
    - 설치되는 경로:   
      (사용자 경로)\.vscode\extensions
    - Live Server: 작성된 Html을 자체 내장 서버에서 실행시켜 준다.  
      
    - Vue 3 Snippets: Snippets을 이용하여 코드 자동 완성을 지원한다.  
      
    - Prettier - Code Forammter: \*.js, \*.html, \*.html 코드 작성 시, Foramt을 일정하게 문서를 수정한다.  
      
    - Vue peek: Vue Component를 쉽게 추적할 수 있게 도와준다.  
      
    - Vetur: Vue.js 코드에 대한 문법 강조, 코드 자동 완성, 디버깅, 린팅 기능들을 제공한다.  
      
    - Bracker Pair Colorizer 2: 괄호의 쌍에 컬러와 경계선을 더해서 잘 구별할 수 있게 도와준다.  
      
    - Auto rename tag: 시작태그 수정 시 닫는 태그까지 자동으로 수정해준다.  
      
  + 환경설정
    - File > Preferences > Settings 의 설정 편집 창은 기본 설정 + 사용자 정의 설정을 보여주는 것으로 실제 Data는 JSON 형태이다.

### Vue2 🡪 Vue3

* 참조 Link
  + <https://velog.io/@bluestragglr/Vue3-%EB%AC%B4%EC%97%87%EC%9D%B4-%EB%B0%94%EB%80%8C%EB%82%98%EC%9A%94>
* Template 생성 방식 변화
* Data, method 작성 방식의 변화
* Lifecycle hook 호출의 변화
* Computed 속성 사용방법의 변화
* Composition API
* Props와 this 바인딩의 분리
* emit과 this 바인딩의 분리
* Suspended Component
* Fragment
* Portal
* 내부적인 변화

## Vue CLI

### 개요

* CLI: Command line interface, 텍스트 터미널을 통해 사용자와 컴퓨터가 상호 작용하는 방식을 뜻함.
* Vue cli는 Vue.js 개발환경을 설정해주는 도구이다.
* Vue cli 가 기본적인 프로젝트 셋팅을 해준다.
  + 폴더 구조, lint, build, 라이브러리 구성, webpack 설정 등에 도움을 준다.
* Vue 3를 이용하기 위해서는 Vue CLI v4.5 이상을 이용해야 한다.
* 설치 Command
  + npm install -g @vue/cli  
    : 사용자 폴더(\AppData\Roaming\npm\node\_modules)에 @vue\cli로 Global하게 설치된다.

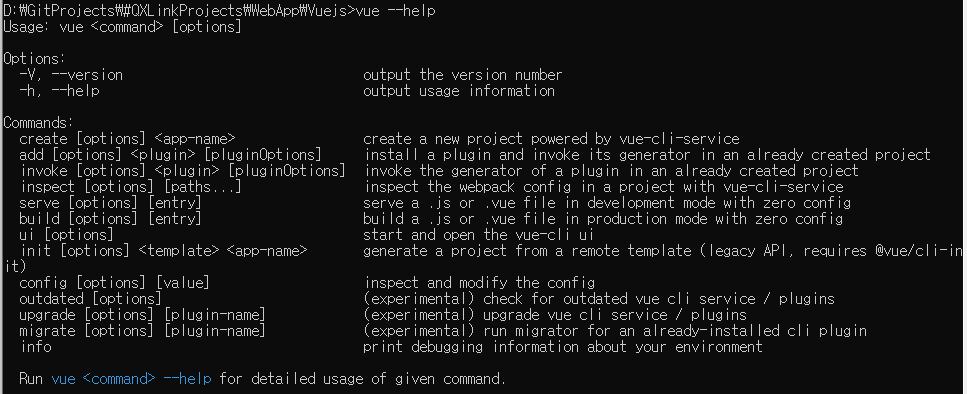
### CLI Service

* @vue/cli를 이용해 만들어진 모든 Project에 설치된다.
* CLI Service는 [webpack](https://webpack.js.org/)과 [webpack-dev-server](https://github.com/webpack/webpack-dev-server)의 위에 빌드 된다.
* CLI PlugIn을 실행하는 핵심 서비스와 webpack에 대한 설정을 포함하고 있어, webpack을 통해 Application의 개발 서버를 실행, 빌드 할 수 있다.

### CLI Plugins

* CLI Plugin들은 선택적인 기능들을 제공하는 npm package 이다.
* 예를 들어 Babel/TypeScript 번역, ESLint 통합, Unit Test, End-to-end Test 등을 쉽게 해준다.
* @vue/cli-plugin- 혹은 vue-cli-plugin-의 이름으로 시작한다.
* vue-cli-service를 project 안에서 Run 할 때, 프로젝트의 package.json에 리스트된 모든 CLI-Plugin들을 자동으로 Load 한다.
* Plugin들은 프로젝트를 생성할 때 포함하거나, 이후에 추가할 수 있다.

### Commands



* create
  + vue create (프로젝트 이름) 을 통해 프로젝트를 생성한다.
* ui
  + vue ui 를 통해 프로젝트를 Web UI에서 관리할 수 있다.

## NPM

### 개요

* Node.js의 기본 패키지 관리자이다.
* NPM을 사용하는 가장 큰 이유는 NPM에 등록되어 있는 다수의 외부 패키지를 설치하기 위해서이다.

### Command

* npm init
  + 현재 Directory에 npm 기반으로 프로젝트를 생성할 수 있다.
* npm install (or npm i) {package}
  + 외부 패키지 설치
  + Git에 npm project를 올릴 때, node\_modules directory는 제외시키는 경우가 일반적이다.
  + package.json 파일을 이용해 관리된다.
  + 옵션
    - -g --global: 전역 설치
* npm ls
  + 설치된 패키지 목록 확인
* npm r {package}
  + 설치된 패키지 제거
* npm run serve
  + NPM에게 package.json에 정의한 serve 를 Run 해라.
* npm run dev / npm run serve

## Basic

### 준비 및 환경

* Script에 직접 추가하기
  + 개발버전: <https://kr.vuejs.org/js/vue.js>  
    : vue.js, 오류 메시지 및 디버그 모드
  + 프로덕션 버전: <https://kr.vuejs.org/js/vue.min.js>  
    : vue.min.js오류 메시지 없음.
* CDN
  + 프로토 타이핑 혹은 학습 목적이라면 아래 script를 추가하여 최신 버전을 사용할 수 있다.
    - Vue.js 2.0  
      :<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    - Vue.js 3.0  
      : <script src="https://unpkg.com/vue@next"></script>
  + Production 환경의 경우, 특정 버전의 빌드 파일을 추가하는 것을 추천한다.
    - <script src="https://cdn.jsdelivr.net/npm/vue@2.6.0"></script>
* Vite
  + [Vite](https://github.com/vitejs/vite)는 빠른 서비스를 위한 web 개발 build tool이다.
  + Native ES Module를 가져오는 방식
  + Vue Project는 Terminal에서 아래 Command를 사용하면 Vite로 인해 빠르게 설치된다.

|  |
| --- |
| $ npm init @vitejs/app <project-name>  $ cd <project-name>  $ npm install  $ npm run dev |

* Build 차이
  + npm package 폴더 (npm\_modules/vue/dist/)에서 볼 수 있다.
    - dist 파일은 use-case에 따라 사용되야 한다.
  + CDN 사용 혹은 Bundler 없는 사용
    - vue(.runtime).global(.prod).js
      1. Browser에서 <script src=”…”>통해 Vue global을 express한다.
      2. Browser에서 template 컴파일
         * vue.global.js: compiler와 runtime을 모두 포함하는 full build 이다.
         * vue.runtime.global.js: 오직 runtime만 포함하고, build step동안 Template이 pre-compile되기를 요구한다.
      3. 모든 Vue core 내부 패키지는 inline 된다.
      4. hard-code된 prod/dev branch를 포함한다.
      5. UMD 빌드가 아니고 IIFEs 빌드이므로 <script src=”…”>를 통해서만 이용할 수 있다.
    - vue(.runtime).esm-browser(.prod).js
      1. native ES module을 통해 import 하여 사용한다.
      2. 같은 runtime compilation, dependency 인라인, hard-coded prod/dev 동작을 global build와 공유한다.
  + Bundler사용
    - vue(.runtime).esm-bundler.js
      1. bundler들 (webpack, rollup, parcel)을 사용하기 위한 방법
      2. prod/dev branch들은 process.env.NODE\_ENV guard와 남겨둔다.
      3. Dependency(@vue/runtime-core, @vue/runtime-compiler)들을 import 한다.
         * Import된 depnedency들도 esm-bundler 빌드이며 차례로 dependency들을 가져온다.
      4. Browser에서 template 컴파일
         * vue.runtime.esm-bundler.js(default)

오직 runtime이며, 모든 template이 pre-compiled 되어 있는 것을 요구한다.

* + - * + vue.esm-bundler.js

runtime compiler를 포함한다.

bundler를 사용하지만 여전히 runtime template compilation을 원하면 이 옵션을 사용해라.

이 파일에 vue라는 alias를 사용하도록 bundler를 구성해야한다.

* + Server-Side Rendering
    - vue.cjs(.prod).js
      1. Node.js server-sider rendering에서 require()를 통해 사용한다.
      2. app을 target:’node’와 webpack을 가지고 bundle하고 적절하게 vue를 외부로 노출시킨다면 이것은 load될 build이다.
      3. dev/prod file들은 pre-build 되지만 적절한 파일은 자동으로 process.env.NODE\_ENV base로 요구한다.
  + Runtime + Compiler vs Runtime-only
    - client에서 template을 compile하는 것이 필요하다(e.g. template option을 string으로 전달하기 혹은 in-DOM HTML사용하는element를 template로써 mounting하는 것)면, compiler와 full build가 필요하다.
    - vue-loader를 사용할 때, \*.vue파일안에 template 은 build 시점에 Javascript에 pre-compiled 된다.
    - 따라서 최종 bundle에서 compiler는 필요하지 않게 되고 runtime-only build만 사용하면 된다.
    - Example)
      1. compiler가 필요한 경우

// this requires the compiler

Vue.createApp({

template: '<div>{{ hi }}</div>'

})

* + - 1. compiler가 필요하지 않은 경우

// this does not

Vue.createApp({

render() {

return Vue.h('div', {}, this.hi)

}

})

### 01) Hello World Page 작성

* App에서 선언한 Data를 HTML에서 다음과 같이 {{Property name}}바인딩하여 표시할 수 있다.
* Vue
* Example
  + index.js

const Counter = {

data() {

return {

counter: 0

}

}

}

Vue.createApp(Counter).mount('#counter')

* + index.html

### 02) 조건문

* 아래와 조건문을 구현할 수 있다.  
  : 예제의 조건은 seen이란 변수값의 true or false를 판별하여 표시한다.
* Example
  + index.html

<div id="conditional-rendering">

<span v-if="seen">Now you see me</span>

</div>

* + index.js

const ConditionalRendering = {

data() {

return {

seen: true

}

}

}

Vue.createApp(ConditionalRendering).mount('#conditional-rendering')

### 03) 반복문

* 아래와 반복문을 구현할 수 있다.
* For문을 아래와 같이 List로 표현할 수 있다.
  + index.html

<div id="list-rendering">

<ol>

<li v-for="todo in todos">

{{ todo.text }}

</li>

</ol>

</div>

* + index.js

const ListRendering = {

data() {

return {

todos: [

{ text: 'Learn JavaScript' },

{ text: 'Learn Vue' },

{ text: 'Build something awesome' }

]

}

}

}

### 04) 사용자 입력

* 사용자 입력을 핸들링 할 수 있다.
* Example)
  + index.html

<div id="event-handling">

<p>{{ message }}</p>

<button v-on:click="reverseMessage">Reverse Message</button>

</div>

* + index.js

const EventHandling = {

data() {

return {

message: 'Hello Vue.js!'

}

},

methods: {

reverseMessage() {

this.message = this.message

.split('')

.reverse()

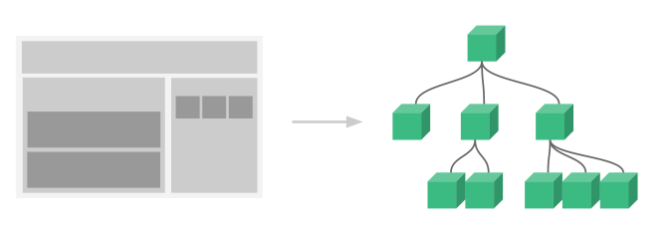
.join('')

}

}

}

### 05) Component

* 독립적으로 재사용할 수 있는 컴포넌트를 이용하여 대규모 어플리케이션을 구축할 수 있게 해준다.
* 모든 유형의 어플리케이션 인터페이스를 Component Tree로 추상화할 수 있다.  
  
* Vue에서 Component는 pre-defined option을 갖는 필수적인 instance 이다.
* Component를 등록하는 방법은 직관 적이며 다음과 같이 Component를 등록할 수 있다.

// Create Vue application

const app = Vue.createApp({})

app.component('my-component-name', {

// ... options ...

})

* 위와 같이 등록한 Component를 Html에서 인스턴스화 할수 있다.

<ol>

<!-- Create an instance of the todo-item component -->

< my-component-name></ my-component-name >

</ol>

* 아래와 같이 위 구성을 Component화 하여 사용할 수 있다.

<div id="app">  
 <app-nav></app-nav>  
 <app-view>  
 <app-sidebar></app-sidebar>  
 <app-content></app-content>  
 </app-view>  
</div>

* Props를 이용하여 Data를 Child component에 전달하기
  + Props는 component에 등록한 사용자 정의 attribute이다.
  + component에 title을 전달하기 위해, props list에 props 옵션을 사용하여 포함시켜라.
    - Example)

const app = Vue.createApp({})

app.component('blog-post', {

props: ['title'],

template: `<h4>{{ title }}</h4>`

})

* + - 값이 prop attribute에 전달될 때, 그것은 component instance에 property가 된다.
    - Component는 많은 props를 갖을 수 있고, 어느 값이든 prop에 전달 될 수 있다.
  + 다음과 같이 data의 배열 형태를 전달할 수 있다.
    - 동적으로 props를 전달하기 위해 v-bind를 이용할 수 있다.
    - props를 알아야 향후 더 구체적인 설계가 가능하다. ([Link](https://v3.vuejs.org/guide/component-props.html))

const App = {

data() {

return {

posts: [

{ id: 1, title: 'My journey with Vue' },

{ id: 2, title: 'Blogging with Vue' },

{ id: 3, title: 'Why Vue is so fun' }

]

}

}

}

const app = Vue.createApp(App)

app.component('blog-post', {

props: ['title'],

template: `<h4>{{ title }}</h4>`

})

app.mount('#blog-posts-demo')

<div id="blog-posts-demo">

<blog-post

v-for="post in posts"

:key="post.id"

:title="post.title"

></blog-post>

</div>

* + Parent와 communication할 때, component instance는 custom event system을 제공한다.
    - parent는 child component instance의 event를 listen 하는 것을 v-on 혹은 @를 사용하여 선택할 수 있다.
      1. emit 한 Event에서 대소문자 조합은 v-on에서 -를 구분자로 구분하여 받게된다. 🡪 kebab-cased (hyphen-delimited) equivalent
      2. <button @click="$emit('enlargeText')">
      3. @enlarge-text="postFontSize += 0.1"

app.component('blog-post', {

props: ['title'],

template: `

<div class="blog-post">

<h4>{{ title }}</h4>

<button @click="$emit(**'enlargeText'**)">

Enlarge text

</button>

</div>

`

})

<div :style="{ fontSize: postFontSize + 'em' }">

<blog-post

v-for="post in posts"

:key="post.id"

:title="post.title"

@**enlarge-text**="postFontSize += 0.1"

></blog-post>

</div>

* + - 다음과 같이 event를 목록화 할 수도 있다.

app.component('blog-post', {

props: ['title'],

emits: ['enlargeText']

})

* + Event와 함께 값 Emit 하기
    - Emit 시에 second parameter를 통해 parameter를 전달 할 수 있다.

<button @click="$emit('enlargeText', 0.1)">

Enlarge text

</button>

* + - Parent에서 listen할 때, Emit된 event의 값을 $event를 통해 접근할 수 있다.

<blog-post ... @enlarge-text="postFontSize += $event"></blog-post>

or

* + - event handler를 method로 이용할 수 있고, 값은 method의 첫번째 parameter로 전달될 것

<blog-post ... @enlarge-text="onEnlargeText"></blog-post>

methods: {

onEnlargeText(enlargeAmount) {

this.postFontSize += enlargeAmount

}

}

* + - Custom Event에 대해 더 자세히 알기 위해선 다음 링크를 확인하자 ([Link](https://v3.vuejs.org/guide/component-custom-events.html#event-names))
  + v-model사용하기
    - custom event들은 v-model을 이용하여 custom input을 생성하는데 사용될 수 있다.
      1. v-model의 일반 적인 동작은 다음과 같다.

<input v-model="searchText" />

or

<input :value="searchText" @input="searchText = $event.target.value" />

1

* + - 1. component에서 사용될 때, v-model은 아래와 같이 사용된다.

<custom-input v-model="searchText"></custom-input>

app.component('custom-input', {

props: ['modelValue'],

emits: ['update:modelValue'],

template: `

<input

:value="modelValue"

@input="$emit('update:modelValue', $event.target.value)"

>

`

})

or

app.component('custom-input', {

props: ['modelValue'],

emits: ['update:modelValue'],

template: `

<input v-model="value">

`,

computed: {

value: {

get() {

return this.modelValue

},

set(value) {

this.$emit('update:modelValue', value)

}

}

}

})

* + slot 사용하기
    - content를 component에 전달할 때 유용하게 사용할 수 있다.
    - 아래와 같을 때, HTML의 content를 render시에 component의 <slot></slot>으로 대체한다.

<alert-box>

Something bad happened.

</alert-box>

app.component('alert-box', {

template: `

<div class="demo-alert-box">

<strong>Error!</strong>

<slot></slot>

</div>

`

})

* Dynamic Components
  + interface tab과 같이 component간 동적으로 변경하는 것은 유용하다.
* dd

### 06) Instance

* 모든 Vue Application은 Vue 함수로 Vue Instance를 생성합니다.
* application instance는 ‘globals’ 등록하는데 사용된다.
  + application안에 있는 component를 사용해서 등록할 수 있다.

const app = Vue.createApp({})

app.component('SearchInput', SearchInputComponent)

app.directive('focus', FocusDirective)

app.use(LocalePlugin)

* createApp은 root component를 구성하는데 사용된다.
* component는 application을 mount할때, rendering을 위한 starting point로써 사용된다.
* application은 DOM element에 mount 될 필요가 있다.
  + Example) html에서 <div id=’app’></div>를 사용하기 위해 #app을 전달해야한다.

const RootComponent = {

/\* options \*/

}

const app = Vue.createApp(RootComponent)

const vm = app.mount('#app')

* + 대부분의application method와 달리 **mount는 application을 return하지 않는다. 대신 root component instance를 리턴**한다.
* 예제에서는 Single component만 필요하지만 실제 Application에서는 nated한 tree 로 구성된다.
  + Example)  
    Root Instance  
    └─ TodoList  
     ├─ TodoItem  
     │ ├─ DeleteTodoButton  
     │ └─ EditTodoButton  
     └─ TodoListFooter  
     ├─ ClearTodosButton  
     └─ TodoListStatistics
  + 각 component들은 component instance를 갖으며, 어떤 component들은 한번에 복수개의 Instance들을 render할 것이다.
* Vue Instance를 생성할 때, Option object를 전달한다.
* Option object 설명: [Link](https://vuejs.org/v2/api/#Options-Data)
  + data
    - Type: Object | Function
      1. Component 정의에서 사용될 때 만 Function이 허용된다.

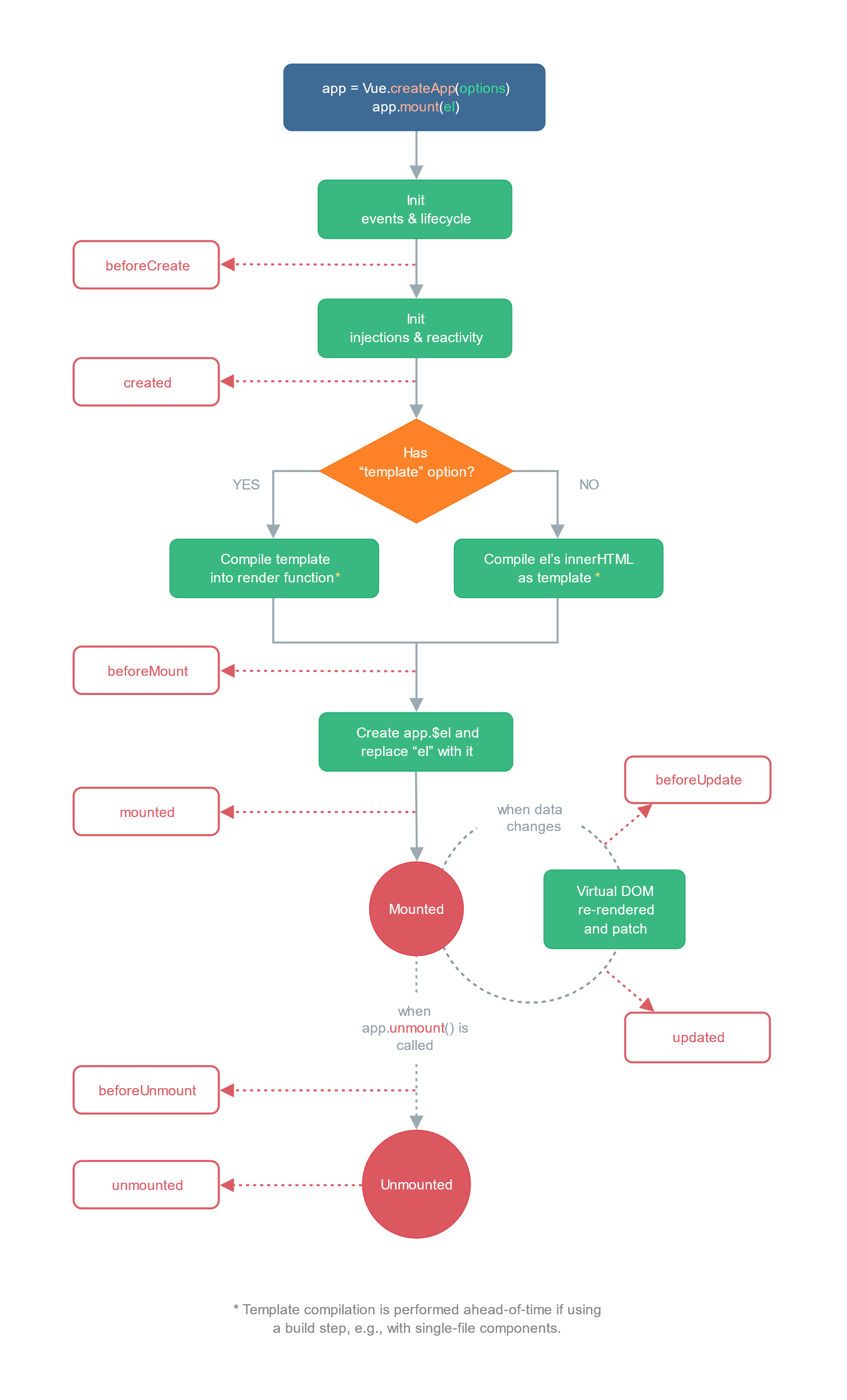
var Component = Vue.extend({

data: function () {

return { a: 1 }

}

})

* + - data object는 Vue Instance를 위한 Object이다.
    - Vue는 재귀적으로 Property들을 getter/setter 들을 이용하여 Reactive(반응형)으로 만든다.
    - Object는 반드시 기본 객체여야 한다.
    - Observe 되면 더 이상 reactive property들을 추가할 수 없다.
    - Instance가 만들어진 후에 Original Data object는 vm.$data로 접근할 수 있다. Vue Instance는 모든 Property를 프록시하므로 **vm.a는 vm.$data.a와 동일하다**.
    - \_ 혹은 $로 시작하는 속성은 충돌할 수 있으므로 Vue Instance에서 프록시 되지 않습니다. 따라서 vm.$data.\_property로 접근해야 한다.
    - Component를 정의할 때, data를 반환하는 함수로 선언해야 한다.
      1. data 함수를 제공함으로써 새로운 Instance가 생성될 때마다 호출하여 초기 데이터의 새 복사본을 반환할 수 있다.
    - Example)  
      var data = { a: 1 }  
      var vm = new Vue({  
       data: data  
      })  
      vm.a // => 1  
      vm.$data === data
* Data와 Method
  + Vue Instance가 생성될 때,
  + Binding
    - 데이터가 변경되면 화면은 다시 렌더링 된다.
    - data에 있는 속성들은 **인스턴스가 생성될 때 존재한 것들만 Reactive(반응형)** 라는 점이다.
    - Object.freeze()를 사용하여 reactivity system이 추적하여 변경하지 않게 할 수 있다.
  + Vue Instance는 유용한 Instance 속성 및 Method를 제공하는데, 사용자 정의 속성과 구분하기 위해 접두어 $를 사용한다.
    - Example)  
      var data = { a: 1 }  
      var vm = new Vue({  
       el: '#example',  
       data: data  
      })  
        
      vm.$data === data // => true  
      vm.$el === document.getElementById('example') // => true  
        
      // $watch 는 인스턴스 메소드 입니다.  
      vm.$watch('a', function (newVal, oldVal) {  
       // `vm.a`가 변경되면 호출 됩니다.  
      })
* Instance Lifecycle
  + Lifecycle Diagram  
     
  + 각 Vue Instance는 생성될 때, 초기화 단계를 거친다
    - Data observation, template Compile하기, DOM에 Instance Mount하기, Data가 변경되어 DOM 업데이트 하기.
    - 그 과정에서 사용자 정의 로직을 실행할 수 있는 Lifecycle hook도 호출된다
    - arrow function을 사용하지 말아라.
    - 종류
      1. created: Instance 생성 시
      2. mounted: Instance를 DOM에 Mount할 때
      3. updated: Update 될 때
      4. destroyed: Instance가 destroy 될 때
    - Example) Created Hook

new Vue({

data: {

a: 1

},

created: function () {

// `this` points to the vm instance

console.log('a is: ' + this.a)

}

})

// => "a is: 1"

### 07) Template

* 개요
  + Vue.js는 HTML 베이스의 Template syntax를 사용한다.
  + 이를 **Render된 DOM을 Vue instance의 Data와 선언적 Binding을 가능하게 한다**.
  + HTML 표준 Tag들과 Vue Instance의 special property(v-bind, v-for)등을 함께 사용하여 브라우저에 Rendering 할 수 있게 한다.
  + 모든 Vue.js template들은 스펙 호환 browser와 HTML Parser에 의해 Parsing될수 있는 유효한 HTML이다.
  + 내부적으로 **Vue는 template을 Virtual DOM Render function으로 컴파일** 한다.
  + Reactive system와 조합된 Vue는 Re-render와 DOM 조작을 최소화한 숫자의 Component들을 파악하여 적용합니다.
* Interpolation
  + Text
    - 가장 일반적인 **data binding의 형태는 “Mustache” syntax(🡪{{}})**를 사용한 text interpolation  
      <span>Message: {{ msg }}</span>
      1. Mustache tag는 msg의 property 값으로 대체될 것인다.
      2. msg property가 바뀔 때 마다 업데이트 될 것이다.
    - v-once directive를 사용하여, 1회 interpolation을 수행할 수 있다. (값이 변경되어도 업데이트 되지 않는다.)  
      <span v-once>This will never change: {{ msg }}</span>
      1. 이는 같은 node의 다른 binding에도 영향을 준다.
  + Raw HTML
    - Double mustache는 HTML이 아닌 plain text로써 data로 해석된다.  
      <p>Using mustaches: {{ rawHtml }}</p>  
      <p>Using v-html directive: <span v-html="rawHtml"></span></p>
    - (Vue instance object의 Property 값이 아닌)Real HTML로 출력하기 위해서 v-html directive가 필요하다.
    - 이때, data binding은 무시된다.
  + Attribute
    - **Mustache는 HTML attribute 안에서 사용될 수 없습니다**. 대신 **v-bind**를 사용합니다.  
      <div v-bind:id="dynamicId"></div>
    - Boolean attribute가 단순 True인 경우, v-bind는 조금 다르게 동작한다.  
      <button v-bind:disabled="isButtonDisabled">Button</button>
      1. isButtonDisabled가 null, undefined 혹은 false일 때, Rendered <button> element에 포함되지 않는다.
  + Javascript Expression
    - Vue.js는 모든 Javascript expression을 data binding안에서 지원한다.

{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>

* + - 한가지 제한 사항은 각 Binding은 오직 한 개의 단일표현식(**Single expression)**만 될 수 있다.
      1. 표현문이 아닌경우

{{ var a = 1 }}

* + - 1. 조건문은 작동하지 않는다.

{{ if (ok) { return message } }}

* + - Template expression들은 sandbox 처리되며, global whitelist에 접근할 수 있다. ex) Math, Data
* Directive
  + 개요
    - v- prefix를 갖는 Special attribute들이다.
    - Directive attribute값은 단일 Javascript expression이다. (v-for 은 예외)
    - Expression의 값이 변경될 때, Reactive하게 DOM에 side effect를 적용하는 것

<p v-if="seen">이제 나를 볼 수 있어요</p>

* + Arguments
    - Directive 중 일부는 directive 뒤에 colon(“:”)을 넣어 Argument를 갖을 수 있다.
      1. Example 1)  
         <a v-bind:href="url"> ... </a>
      2. Example 2)  
         <a v-on:click="doSomething"> ... </a>
  + **Dynamic Arguments**
    - 2.6.0 부터 Directive argument에 대괄호([])를 사용하여 JavaScript expression 사용이 가능해 졌다.
      1. Example 1) Attribute  
         <a v-bind:[attributeName]="url"> ... </a>
      2. Example 2) Event

<a v-on:[eventName]="doSomething"> ... </a>

* + - attributeName은 JavasScript Expression으로써 동적으로 변환된다.
    - 제약사항
      1. Value
         * Argument는 null을 제외하고는 String으로 변환될 것으로 예상된다.
         * null의 경우 binding을 제거하는데 사용될 수 있다.
         * string이 아닌 다른 값은 warning을 trigger할 수 있다.
      2. Expression
         * 특정 character(공백, Quate(‘), 유효하지 않은 이름) 때문에 Syntax 제약사항을 갖는다.

<!-- This will trigger a compiler warning. -->

<a v-bind:['foo' + bar]="value"> ... </a>

* + - * + **공백, Quate(‘)가 없는 Expression**을 사용하거나 복잡한 Expression은 Computed Property로 대체한다.
        + DOM template을 사용할 때, Browser들은 attribute이름을 강제로 소문자로 변환하기 때문에 **대문자를 갖는 name은 피한다**.

<!--  
in-DOM 탬플릿에서는 이 부분이 v-bind:[someattr]로 변환됩니다.  
인스턴스에 "someattr"속성이 없는 경우, 이 코드는 동작하지 않습니다.  
-->  
<a v-bind:[someAttr]="value"> ... </a>

* + 수식어(Modifier)
    - dot(.) 뒤에서 directive가 특별한 방법으로 Bind되야 함을 나타내는 postfix 이다.
    - Example
      1. .prevent: v-on directive는 Event가 트리거 될 때, event.preventDefulat()를 호출하도록 알려준다.

<form v-on:submit.prevent="onSubmit"> ... </form>

* Shorthands(약어)
  + 개요
    - v- prefix는 template에서 Vue-specific attribute를 식별하기 위한 신호로 쓰인다.
    - 동적인 동작을 적용하기 위해 Vue.js를 사용하고 있을 때 유용하다.
    - 모든 Template을 관리하는 Vue에서 SPA building을 할 때는 덜 중요하다.
    - 따라서, v-bind와 v-on에 대해 특별한 약어를 제공한다.
  + v-bind

<!-- 전체 문법 -->  
<a v-bind:href="url"> ... </a>  
  
<!-- 약어 -->  
<a :href="url"> ... </a>  
  
<!-- shorthand with dynamic argument (2.6.0+) -->  
<a :[key]="url"> ... </a>

* + v-on

<!-- 전체 문법 -->  
<a v-on:click="doSomething"> ... </a>  
  
<!-- 약어 -->  
<a @click="doSomething"> ... </a>  
  
<!-- shorthand with dynamic argument (2.6.0+) -->  
<a @[event]="doSomething"> ... </a>

### 08) Computed Property

* 개요
  + Vue.js는 HTML 베이스의 Template syntax를 사용한다.
* Computed property
  + 개요
    - Template에서 표현식은 간단한 동작에서는 매우 편하다.
    - Template에 너무 많은 Logic을 넣는 것은 비대해져서 유지보수가 어렵게된다.

<div id="example">

{{ message.split('').reverse().join('') }}

</div>

* + Basic Example

// html

<p>Computed reversed message: "{{ reversedMessage }}"</p>

// javascript

computed: {

// a computed getter

reversedMessage: function () {

// `this` points to the vm instance

return this.message.split('').reverse().join('')

}

}

* + - 이는 선언적인 Computed property인 reversedMessage를 갖는다.
    - Template에서 Computed property에 데이터 바인딩할 수 있다.
  + vs Methods
    - Method를 이용해서도 같은 결과를 만들어 낼 수 있다.

<p>Reversed message: "{{ reverseMessage() }}"</p>

// in component

methods: {

reverseMessage: function () {

return this.message.split('').reverse().join('')

}

}

* + - 결과적으로는 완벽히 같으나, Computed property는 Reactive dependency들에서 Cached된다.
      1. 따라서**, Computed property는 Reactive dependency들이 변경될 때만 함수를 수행한다**.
  + vs watched Property

|  |
| --- |
| <div id="demo">{{ fullName }}</div> |

|  |
| --- |
| var vm = new Vue({  el: '#demo',  data: {  firstName: 'Foo',  lastName: 'Bar',  fullName: 'Foo Bar'  },  watch: {  firstName: function (val) {  this.fullName = val + ' ' + this.lastName  },  lastName: function (val) {  this.fullName = this.firstName + ' ' + val  }  } }) |

이를 Computed property로 했을 때

var vm = new Vue({  
 el: '#demo',  
 data: {  
 firstName: 'Foo',  
 lastName: 'Bar'  
 },  
 computed: {  
 fullName: function () {  
 return this.firstName + ' ' + this.lastName  
 }  
 }  
})

* + - Vue instance에서 Data 변화를 관찰하고 React하는 더 보편적인 방법은 watch property이다.
    - 다른 data 기반으로 변경할 필요가 있는 경우, watch를 남용하는 경우가 있다. 하지만 watch 보다는 computed property를 사용하는 것이 좋다.
  + vs setter
    - Computed property는 기본적으로 getter 함수만 가지고 있고, 필요한 경우 setter를 만들 수 있다.

computed: {  
 fullName: {  
 // getter  
 get: function () {  
 return this.firstName + ' ' + this.lastName  
 },  
 // setter  
 set: function (newValue) {  
 var names = newValue.split(' ')  
 this.firstName = names[0]  
 this.lastName = names[names.length - 1]  
 }  
 }  
}

* watch property
  + 개요
    - 대부분의 경우 computed property들은 더 적절하지만 watcher가 필요할 때가 있다.
    - Vue는 watch option을 통해서 data 변경의 react를 위한 더 일반적인 방법을 제공한다.
    - data를 변경하는 response를 위해 비동기 적이거나 값비싼 operation을 수행할 때 가장 유용하다.

### 09) Class and Binding

* 개요
  + Element의 Class list와 inline style을 조작하기 위해 Data binding이 필요하다.
  + 둘 다 Attribute이기 때문에 v-bind를 사용한다.
  + expression은 string 이외에 object 혹은 array를 사용할 수 있다.
* Binding HTML Classes
  + Object Syntax
    - 우리는 동적으로 class를 toggle하기 위해, Object를 **v-bind:class(축약 🡪 :class)**에 전달할 수 있다.

<div v-bind:class="{ active: isActive }"></div>

or

<div :class="{ active: isActive }"></div>

* + - 1. 위 Syntax는 active class의 현재 값을 isActive data property의 true or false로 결정한다.
      2. object에 여러 field들을 이용해 복수개의 class를 toggle 할 수 있다.
      3. binding 된 객체는 Inline일 필요는 없다.
      4. Return Object로의 Computed Property도 bind할 수 있다.
  + Array Syntax
    - 배열을 v-bind:class에 전달 할 수 있다.

<div v-bind:class="[activeClass, errorClass]"></div>

data: {

activeClass: 'active',

errorClass: 'text-danger'

}

* + Components
    - Custom Component에서 class attribute를 사용할 떄, 해당 Class들은 component의 root element에 더해진다.
* Binding Inline Style
  + Object Syntax
    - v-bind:style을 사용하는 것은 매우 직관적인 방식으로 Javascript인 것을 제외하면 CSS와 비슷하다.
  + Array Syntax
    - Array를 이용하여 Object를 전달할 수 있다.

<div v-bind:style="[baseStyles, overridingStyles]"></div>

* + - 1. 위의 경우 baseStyles Object 적용 후, overridingStyles를 적용한다.
  + Auto-prefixing
    - v-bind:style에서 vendor prefix들을 요구하는 CSS Property를 사용할 때, ( ex) transform), 적절한 prefix들을 Applied style에 자동으로 detect하고 더한다.
  + Multiple Values
    - multiple array를 더할 수 있다.

<div v-bind:style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>

### 10) Conditional Rendering

* v-if
  + v-if directive는 조건문 block으로 사용된다.
  + 만약 directive의 표현식이 값을 리턴하면 block은 render 될 것이다.

<h1 v-if="awesome">Vue is awesome!</h1>

<h1 v-else>Oh no 😢</h1>

* + <template>에서 v-if를 활용한 group

<template v-if="ok">

<h1>Title</h1>

<p>Paragraph 1</p>

<p>Paragraph 2</p>

</template>

* v-else
  + v-else를 사용하여 else block을 만들 수 있다.

        <div v-if="isShow">

            Now you see me

        </div>

        <div v-else-if="isHide">

            Now you don't

        </div>

        <div v-else>

            Now you don't anything

        </div>

* + v-if 혹은 v-else-if 다음에 바로 나와야한다.
  + v-else-if는 2.1.0 이후 버전에서 사용가능 하다.
* Controlling Reusable Elements with key
  + Vue는 다시 그들을 rendering할 때, 가능한한 효율적으로 시도한다.
  + 이는 빠른 Rendering 이외의 이점이 있다.

<template v-if="loginType === 'username'">  
 <label>사용자 이름</label>  
 <input placeholder="사용자 이름을 입력하세요" key="username-input">  
</template>  
<template v-else>  
 <label>이메일</label>  
 <input placeholder="이메일 주소를 입력하세요" key="email-input">  
</template>

* + 두 element는 별개이므로 재사용하지 마십시오 라는 key속성을 추가해줍니다.
* v-show
  + element를 조건부로 표시하기 위한 다른 Directive이다.

<h1 v-show="ok">Hello!</h1>

* + v-show vs v-if
    - v-show는 항상 DOM에 render되어 남아 있다.
    - v-show는 오직 element의 CSS property가 표시될 때 toggle 한다.
    - v-show는 <template>을 지원하지 않고 v-else 또한 동작하지 않는다.
    - v-show는 더 간단하다: element는 항상 초기 condition에 관하여 CSS-based toggling으로 render 된다.
    - 더 종종 toggle 될 필요가 있으면 v-show를 선호하고, runtime시 조건이 빈번하게 변화하지 않으면 v-if를 선호한다.
* v-for와 v-if 함께 사용
  + v-if와 v-for를 함께 사용하는 것은 권고하지 않는다.
  + v-if와 v-for를 함께 사용할 때 v-for가 더 우선순위가 높다.

### 11) List Rendering

* Mapping Array to Element (with v-for)
  + 개요
    - v-for directive를 이용하여 배열을 이용하여 item의 리스트를 만들 수 있다.
    - v-for directive는 item in items라는 특별한 형태로의 special syntax를 요구한다.  
      items는 source data array이고 item은 array element의 alias이다.

<li v-for="item in items">

{{ item.message }}

</li>

data() {

return {

items: [{ message: 'Foo' }, { message: 'Bar' }]

}

}

* + index를 아래와 같이 활용할 수 있다.

<li v-for="(item, index) in items">

{{ parentMessage }} - {{ index }} - {{ item.message }}

</li>

data() {

return {

parentMessage: 'Parent',

items: [{ message: 'Foo' }, { message: 'Bar' }]

}

}

* + in 대신 delimiter 로써 of를 사용할 수 있다.
* v-for with an Object
  + object의 property들을 iterate 하기 위하여 v-for를 사용할 수 있다.

<li v-for="value in object">

{{ value }}

</li>

data() {

return {

object: {

title: 'How to do lists in Vue',

author: 'Jane Doe',

publishedAt: '2016-04-10'

}

}

}

* + 다음과 같이 Value와 Name을 구분할 수 있다.

<div v-for="(value, name) in object">

{{ name }}: {{ value }}

</div>

* + 기본 정렬은 object.key()의 순서를 이용해 정렬한다.
* Maintaining State
  + v-for를 이용하여 element를 rendering할 때, 기본적으로 “in-place patch”전략을 사용한다.
  + data item의 순서가 바뀌면 DOM element를 옮기는 대신해 Vue는 각 element를 in-place 패치하고 특별한 Index에 render된 것을 반영한다.
  + 기본적인 모드는 효율적이지만 목록 출력 결과가 자식 Component 상태 혹은 임시 DOM 상태에 의존하지 않는 경우 적합하다.
  + Vue에 힌트를 주기 위해 각 node의 ID를 Tracking 할 수 있고 따라서 재사용 및 재정렬할 수 있다.

<div v-for="item in items" v-bind:key="item.id">

<!-- content -->

</div>

* + 가능하면 언제든 v-for에 key를 추가하는 것이 좋습니다.
    - key는 Primitive Value (기본 타입)으로 사용해야한다.
    - key는 Vue가 노드를 식별하는 일반적인 매커니즘이기 때문에 v-for와 연관되지 않는 다른 용도로도 사용할 수 있다.
* Array Change Detection
  + Mutation Method(변이 메소드)
    - Vue는 감시중인 Array의 변이 method들을 감싸고 있어서 view의 갱신을 trigger한다.
    - Wrapped method들은 다음과 같다
      1. push(): 배열 끝에 아이템을 추가한다.
      2. pop(): 배열 마지막에 아이템을 제거한다.
      3. shift(): 첫번째 아이템을 제거한다.
      4. unshift(): 배열 앞에 아이템을 추가한다.
      5. splice(pos, length): pos번째부터 length만큼 아이템을 제거한다.
      6. sort(): 적절한 위치에 정렬 하여 반환한다.
      7. reserve(): 배열의 순서를 반전한다.
  + Replacing an Array
    - Mutation Method는 원본 배열을 변형한다.
    - 변형을 하지 않는 방법은 바로 filter(), concat(), slice()이다.
    - 이들을 이용하면 원본 배열을 변형하지 않고 항상 새 배열을 반환한다.
  + Caveats
    - javascript의 한계 때문에 object와 array의 변화를 인식하지 못하는 타입들이 있다.
      1. index로 배열에 있는 항목을 직접 설정하는 경우
         * vm.items[indexOfItem] = newValue
      2. 배열 길이를 수정하는 경우
         * vm.items.length = newLength
    - 사용자는 vm.$set method를 사용할 수 있다.
      1. vm.$set(vm.items, indexOfItem, newValue)
    - Object.assign() 혹은 \_.extend()를 사용해 기존의 객체에 새 속성을 할당 할 수 있다.

Object.assign(vm.userProfile, {  
 age: 27,  
 favoriteColor: 'Vue Green'  
})

* 결과 표시 및 정렬
  + 원본 데이터를 실제 변경하거나 재설정하지 않고, 필터된 혹은 정렬된 데이터로 표시해야 할 때가 있다.

|  |
| --- |
| <li v-for="n in evenNumbers">{{ n }}</li> |

|  |
| --- |
| data: {  numbers: [ 1, 2, 3, 4, 5 ] }, computed: {  evenNumbers: function () {  return this.numbers.filter(function (number) {  return number % 2 === 0  })  } } |

* + v-for와 v-if 사용

<li v-for="todo in todos" v-if="!todo.isComplete">  
 {{ todo }}  
</li>

or

<ul v-if="todos.length">  
 <li v-for="todo in todos">  
 {{ todo }}  
 </li>  
</ul>  
<p v-else>No todos left!</p>

### 12) Event Handling

* 개요
  + DOM event들을 listen 하기 위해 v-on directive를 사용하거나 @를 사용할 수 있다.
  + Event가 Trigger 될 때, Javascript를 Run 할 수 있다.
  + Example)

<div id="basic-event">

<button @click="counter += 1">Add 1</button>

<p>The button above has been clicked {{ counter }} times.</p>

</div>

Vue.createApp({

data() {

return {

counter: 0

}

}

}).mount('#basic-event')

* Method Event Handler
  + Event Handler를 위한 Logic은 더 복잡할 것이어서 v-on attribute로 보관하는 것은 간단하지 않다. 따라서 method를 사용한다.
* Method in inline Handler
  + method 이름으로 직접 binding하는 대신에 Javascript 구문을 inline으로 사용할 수 있다.

<div id="inline-handler">

<button @click="say('hi')">Say hi</button>

<button @click="say('what')">Say what</button>

</div>

Vue.createApp({

methods: {

say(message) {

alert(message)

}

}

}).mount('#inline-handler')

* + Original DOM event에 접근할 필요가 있을 때, $event variable을 사용하여 전달할 수 있다.
* Multiple Event Handlers
  + event handler에서 복수개의 method를 갖을 수 있다.

<!-- both one() and two() will execute on button click -->

<button @click="one($event), two($event)">

Submit

</button>

1  
2  
3  
4

// ...

methods: {

one(event) {

// first handler logic...

},

two(event) {

// second handler logic...

}

}

* Event Modifiers(수식어)
  + event handler 내부(javascript)에서 event.preventDefault()나 event.stopPropagation()을 빈번히 호출할 필요가 있다. method 내에서 쉽게 할 수 있을지라도 DOM event 상세를 다루기보다 순수하게 data 만 처리할 수 있다면 더 좋을 것이고 이 문제를 해결하기 위해 v-on에 event modifier들을 제공한다.
  + modifier들은 directive 접미어로 dot(.)을 입력한다.
  + 종류
    - .stop
      1. Propagation(전파)가 중단됩니다.
    - .prevent
      1. 해당 이벤트로 페이지를 다시 로드하지 않습니다.
    - .capture
      1. 이벤트 리스너를 추가할 때, 캡쳐모드를 사용하여 element 대상으로 발생되는 이벤트가 해당 element에서 처리되기 전에 여기서 처리된다.
    - .self
      1. event.target이 element 자체인 경우에만 trigger를 처리한다.
    - .once
      1. event가 한번만 trigger 된다.
    - .passive
      1. addEventListener의 passive option, listener에 의해 특정된 function은 취소할 수 없다.(preventDefault()를 호출할 수 없다.)
      2. .prevent와 함께 사용하지 마라. .prevent는 무시되고 브라우저는 오류를 발생 시킬 것이다.
* Key Modifiers
  + keyboard event를 Listening 할 때, 특정 key를 확인할 필요가 있다.
    - Example)

<!-- only call `vm.submit()` when the `key` is `Enter` -->

<input @keyup.enter="submit" />

* + 직접적으로 유효한 key 이름을 사용할 수 있다.
    - Example)

<input @keyup.page-down="onPageDown" />

* Key Aliases
  + 가장 일반적으로 사용되는 key들을 위해 alias를 제공한다.
    - .enter
    - .tab
    - .delete (captures both "Delete" and "Backspace" keys)
    - .esc
    - .space
    - .up
    - .down
    - .left
    - .right
* System Modifier Keys
  + 아래 Modifier들을 사용하여, 다음 modifier key가 눌렸을 때, mouse 혹은 keyboard event 리스너들을 trigger할 수 있다.
    - .ctrl
    - .alt
    - .shift
    - .meta
  + Example)

<!-- Alt + Enter -->

<input @keyup.alt.enter="clear" />

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

* .exact Modifier
  + system modifier들의 정확한 조합을 control 할 때 사용된다.
  + Example)

<!-- this will fire even if Alt or Shift is also pressed -->

<button @click.ctrl="onClick">A</button>

<!-- this will only fire when Ctrl and no other keys are pressed -->

<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- this will only fire when no system modifiers are pressed -->

<button @click.exact="onClick">A</button>

* Mouse Button Modifier
  + 특정 mouse 버튼에 의해 Event가 Trigger 될 때, 반응한다.
    - .left
    - .right
    - .middle
* HTML에 Listener가 있는 이유
  + HTML 템플릿을 봄으로써 JS code 안에 Handler 함수 구현을 더 쉽게 할 수 있다.
  + 수동으로 JS에 event listener를 부착해서는 안되기 때문에, ViewModel code는 순수한 로직과 DOM-free 해질 수 있어 test를 더 쉽게 해준다.
  + ViewModel이 destroy될 때, 모든 event listener들은 자동으로 제거되어 clean 작업을 스스로 할필요가 없다.

### 13) Form Input Binding

* 개요
  + v-model directive를 다음과 같은 목적으로 사용할 수 있다.
    - form input, textarea, selected elements에서 2 way data bindings를 만들기 위해
  + input-type에 기반한 element를 업데이트 하는 정확한 방법을 자동으로 선택한다.
  + v-model은 input event에서 data를 업데이트하기 위한 필수적인 syntax sugar이다.
  + edge case들을 위한 특별한 처리방법이다.
  + v-model은 내부적으로 다른 property들을 사용하고 다른 input element들을 위해 다른 event들을 방출한다.
    - text와 textarea element들은 value property와 input event를 사용한다.
    - checkbox들과 radiobutton들은 checked property와 change event를 사용한다.
    - select field들은 prop으로써 value를 사용하고 change event를 사용한다.
* 주의사항
  + v-model은 어느 form elements에서 발견된 초기 value, checked, selected attribute들을 무시하고, Vue instance data를 실제 source로써 다룰 것이다. 따라서, 초기값은 JS side에서 component data option에서 선언해야한다.
* Text
  + example)

<input v-model="message" placeholder="edit me">

<p>Message is: {{ message }}</p>

* Multiline text
  + example)

<span>Multiline message is:</span>

<p style="white-space: pre-line;">{{ message }}</p>

<br>

<textarea v-model="message" placeholder="add multiple lines"></textarea>

* + 주의사항
    - <textarea>{{text}}</textarea>는 작동하지 않는다. 대신 v-model을 위와 같이 사용해라.
* Checkbox
  + Single Checkbox는 Boolean 값을 갖는다.

<input type="checkbox" id="checkbox" v-model="checked">

<label for="checkbox">{{ checked }}</label>

* + Multiple Checkbox는 같은 배열과 바인딩 된다.

<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">

<label for="jack">Jack</label>

<input type="checkbox" id="john" value="John" v-model="checkedNames">

<label for="john">John</label>

<input type="checkbox" id="mike" value="Mike" v-model="checkedNames">

<label for="mike">Mike</label>

<br>

<span>Checked names: {{ checkedNames }}</span>

new Vue({

el: '...',

data: {

checkedNames: []

}

})

* Radio
  + Example)

<input type="radio" id="one" value="One" v-model="picked">

<label for="one">One</label>

<br>

<input type="radio" id="two" value="Two" v-model="picked">

<label for="two">Two</label>

<br>

<span>Picked: {{ picked }}</span>

* Select
  + Example)

<select v-model="selected">

<option disabled value="">Please select one</option>

<option>A</option>

<option>B</option>

<option>C</option>

</select>

<span>Selected: {{ selected }}</span>

* + Multiple Select option
    - Example)

<select v-model="selected" multiple>

<option>A</option>

<option>B</option>

<option>C</option>

</select>

* + Dynamic options
    - Example)

<select v-model="selected">

<option v-for="option in options" :value="option.value">

{{ option.text }}

</option>

</select>

<span>Selected: {{ selected }}</span>

data() {

return {

selected: 'A',

options: [

{ text: 'One', value: 'A' },

{ text: 'Two', value: 'B' },

{ text: 'Three', value: 'C' }

]

}

}

* Value Binding
  + radio, checkbox, select option들은 v-model의 static string 값으로 바인딩 할 수 있다.
  + Example)

<!-- `picked` is a string "a" when checked -->

<input type="radio" v-model="picked" value="a" />

<!-- `toggle` is either true or false -->

<input type="checkbox" v-model="toggle" />

<!-- `selected` is a string "abc" when the first option is selected -->

<select v-model="selected">

<option value="abc">ABC</option>

</select>

<input type="checkbox" v-model="toggle" true-value="yes" false-value="no" />

<input type="radio" v-model="pick" v-bind:value="a" />

<select v-model="selected">

<!-- inline object literal -->

<option :value="{ number: 123 }">123</option>

</select>

* Modifiers
  + .lazy
    - input event 후에 data를 가지고 input과 동기화 한다.
    - Example)

<!-- synced after "change" instead of "input" -->

<input v-model.lazy="msg" />

* + .number
    - 자동으로 type을 숫자로 변경하길 원할 때 사용한다.
    - Example)

<input v-model.number="age" type="number" />

* + .trim
    - 사용자 입력으로부터 자동으로 공백을 이용하여 trim할 때 사용한다.

<input v-model.trim="msg" />

* v-model with Component
  + HTML의 built-in input 타입은 항상 만족 시킬수는 없다.
  + 다행히 component들은 항상 재사용가능한 input을 사용자 정의된 동작으로 빌드할 수 있다.
  + 이 input들은 v-model과도 잘 동작한다. [[Link](https://v3.vuejs.org/guide/component-basics.html#base-example)]

# 참고 문헌 및 자료

## Vue CLI

* <https://cli.vuejs.org/guide/>
* <https://kdydesign.github.io/2019/04/22/vue-cli3-tutorial/>

## 환경 구성

* <https://goodteacher.tistory.com/192?category=820671>

## HTML

* <http://webberstudy.com/html-css/beginning/introduce/>

## 환경 구성

Vieworks Co., Ltd

41-3, Burim-ro, 170beon-gil,

Dongan-gu, Anyang-si, Gyeonggi-do

431-060 Republic of Korea

<http://www.vieworks.com>