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

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

목차

[1 개요 3](#_Toc64354760)

[2 연구(검토) 목적 4](#_Toc64354761)

[3 연구(검토) 세부 내용 4](#_Toc64354762)

[3.1 Vue.js 소개 4](#_Toc64354763)

[3.1.1 개요 4](#_Toc64354764)

[3.1.2 MVVM 패턴 4](#_Toc64354765)

[3.1.3 설치하기 5](#_Toc64354766)

[3.1.4 Vue2 🡪 Vue3 9](#_Toc64354767)

[3.2 Vue CLI 9](#_Toc64354768)

[3.2.1 개요 9](#_Toc64354769)

[3.2.2 CLI Service 10](#_Toc64354770)

[3.2.3 CLI Plugins 10](#_Toc64354771)

[3.2.4 Commands 10](#_Toc64354772)

[3.3 NPM 12](#_Toc64354773)

[3.3.1 개요 12](#_Toc64354774)

[3.3.2 Command 12](#_Toc64354775)

[3.4 시작하기 13](#_Toc64354776)

[3.4.1 Vue.js 사용하기 13](#_Toc64354777)

[3.4.2 01) Hello World Page 작성 13](#_Toc64354778)

[3.4.3 02) 조건문 13](#_Toc64354779)

[3.4.4 03) 반복문 14](#_Toc64354780)

[3.4.5 04) 사용자 입력 14](#_Toc64354781)

[3.4.1 05) Component 15](#_Toc64354782)

[3.4.2 06) Instance 15](#_Toc64354783)

[3.4.3 07) Template 19](#_Toc64354784)

[3.4.4 08) Computed Property 23](#_Toc64354785)

[3.4.5 09) Class and Binding 25](#_Toc64354786)

[3.4.6 10) Conditional Rendering 26](#_Toc64354787)

[3.4.7 10) List Rendering 28](#_Toc64354788)

[4 참고 문헌 및 자료 31](#_Toc64354789)

[4.1 Vue CLI 31](#_Toc64354790)

[4.2 환경 구성 31](#_Toc64354791)

[4.3 HTML 31](#_Toc64354792)

[4.4 환경 구성 31](#_Toc64354793)

# 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 2015 | | |
| 키워드 | Web Development, Font-End | | |
| 요약 정리 | Gdiplus에서 제공하는 Interpolation을 사용하여 왔으나, 성능과 종류의 부족을 느껴 Interpolation에 대해 알아보고 이를 직접 구현해 보기로 한다. | | |

# 연구(검토) 목적

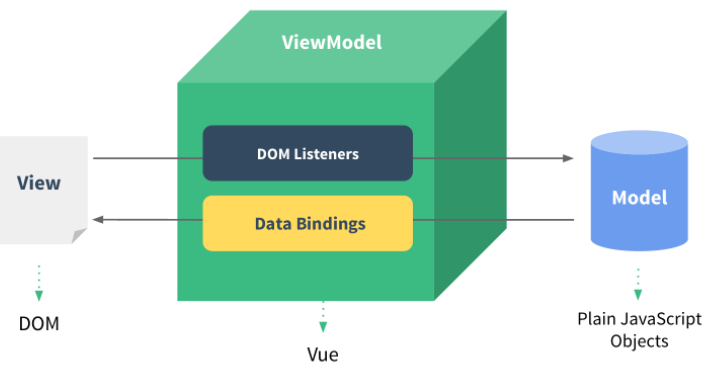
영상의

# 연구(검토) 세부 내용

## Vue.js 소개

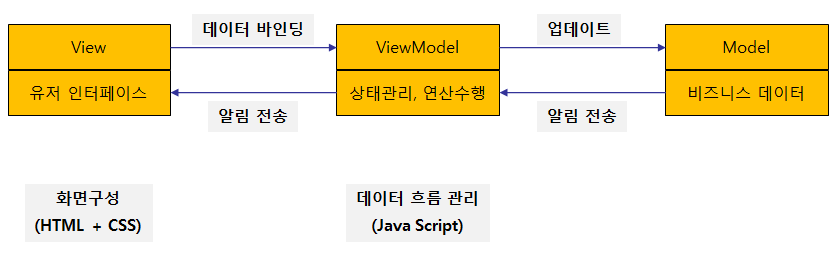
### 개요

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

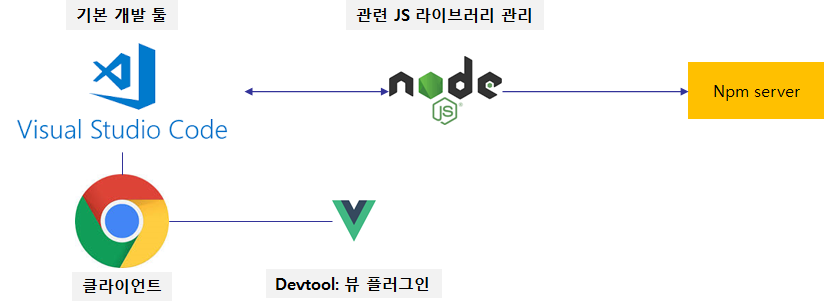


* Data binding과 화면 단위를 Component 형태로 제공하며, 관련 API를 지원
* Angular에서 지원하는 양방향 Data binding을 동일하게 제공
* Component간 통신의 기본 결격은 React의 단방향 데이터 흐름(부모 🡪 자식)을 사용
* 다른 Front-End Framework(Angular, React)와 비교했을 때 상대적으로 **가볍고 빠름**
* 문법이 단순하고 간결하여 **초기 합승 비용이 낮음**

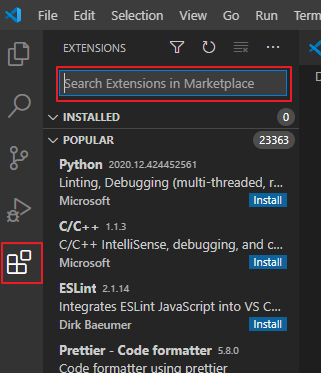
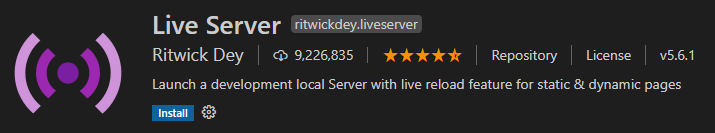
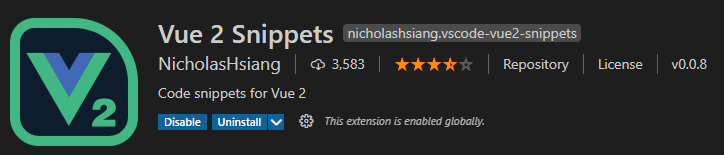
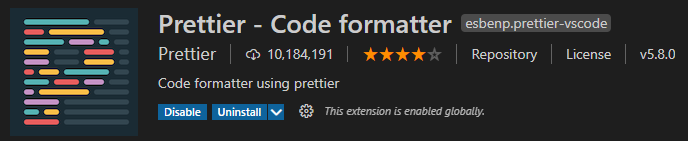
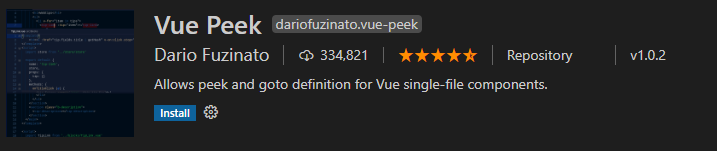
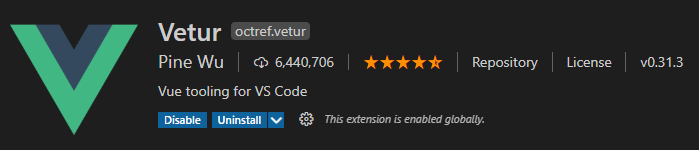
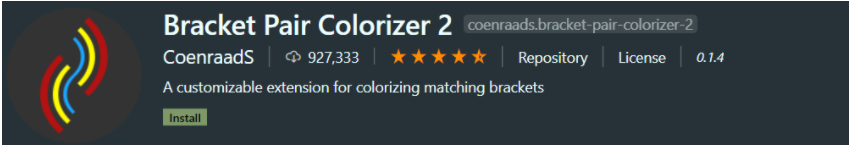
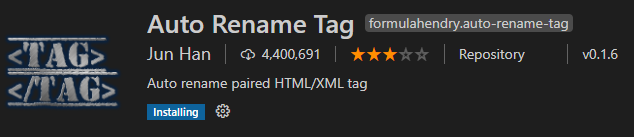
### MVVM 패턴

* 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 2 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.
* 설치 Command
  + npm install -g @vue/cli  
    : 사용자 폴더(\AppData\Roaming\npm\node\_modules)에 @vue\cli로 Global하게 설치된다.

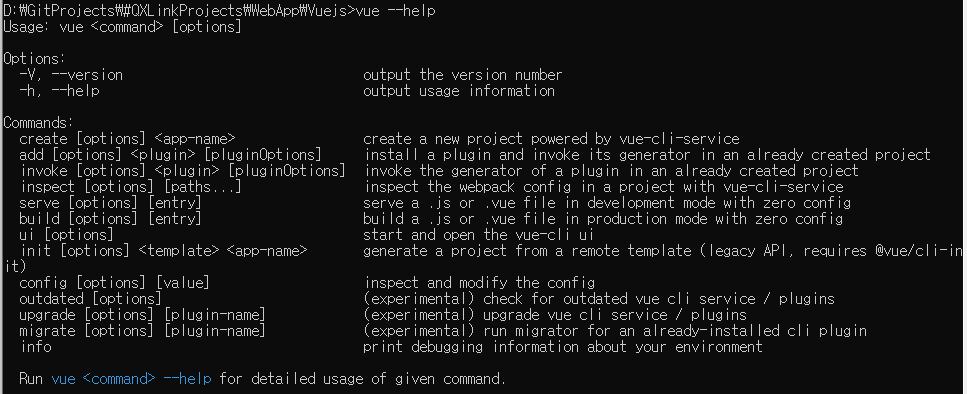
### CLI Service

* @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.
* d

## 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

## 시작하기

### Vue.js 사용하기

* 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.
    - <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  + Production 환경의 경우, 특정 버전의 빌드 파일을 추가하는 것을 추천한다.
    - <script src="https://cdn.jsdelivr.net/npm/vue@2.6.0"></script>

### 01) Hello World Page 작성

* Vetur를 설치하였으면 html 코드 추가 후, **! 🡪 Tab키** 를 이용하여 기본 Page 구성을 만들수 있다.
* 다음의 CDN Script를 추가해준다.

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

* 다음과 같이 index.js 을 html에 추가하여 사용할 수 있게 해준다.

<script src="index.js"></script>

* Body에 Vue.js를 이용하여 생성한 message를 추가해준다.

    <div id="app">

        {{message}}

    </div>

### 02) 조건문

* 아래와 조건문을 구현 할 수 있다.  
  : 예제의 조건은 seen이란 변수값의 true or false를 판별하여 표시한다.

// index.html

    <div id="app">

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

    </div>

// index.js

var app = new Vue({

    el: '#app',

    data: {

      seen: true

    }

  })

* 조건 값 변경

// index.js

 app.seen = true

### 03) 반복문

* 아래와 반복문을 구현할 수 있다.
* For문을 아래와 같이 List로 표현할 수 있다.

       <ol>

          <li v-for="todo in todos">

            {{ todo.text }}

          </li>

        </ol>

* Javascript에는 아래와 같이 배열로 데이터를 넣어주자.

      todos: [

        { text: 'Learn JavaScript' },

        { text: 'Learn Vue' },

        { text: 'Build something awesome' }

      ]

### 04) 사용자 입력

* 사용자 입력을 핸들링 할 수 있다.
* 다음과 같이 입력 컨트롤을 넣어 변수(inputMessage)에 입력을 받을 수 있다.

<input v-model="inputMessage">

var app = new Vue({

    el: '#app',

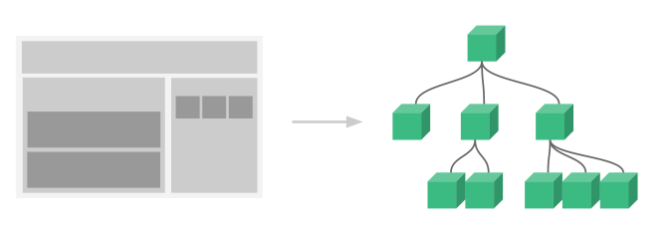
    data: {

      inputMessage: 'Input Your Message'

    },

  });

### 05) Component

* 독립적으로 재사용할 수 있는 컴포넌트를 이용하여 대규모 어플리케이션을 구축할 수 있게 해준다.
* 모든 유형의 어플리케이션 인터페이스를 Component Tree로 추상화할 수 있다.  
  
* 다음과 같이 Component를 등록할 수 있다.

Vue.component('todo-item', {  
 template: '<li>할일 항목 하나입니다.</li>'  
})  
var app = new Vue(...)

* 위와 같이 등록한 Component를 Html에서 인스턴스화 할수 있다.

<ol>  
 <todo-item></todo-item>  
</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>

### 06) Instance

* 모든 Vue Application은 Vue 함수로 Vue Instance를 생성합니다.
* 관례적으로 변수 vm을 Vue Instance로 언급한다.
* 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
* Vue Application은 root Vue Instance로 구성된다. 선택적으로 재사용 가능한 nested tree 로 구성 가능하다.
  + Example)  
    Root Instance  
    └─ TodoList  
     ├─ TodoItem  
     │ ├─ DeleteTodoButton  
     │ └─ EditTodoButton  
     └─ TodoListFooter  
     ├─ ClearTodosButton  
     └─ TodoListStatistics
* 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 Redering 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 사용될수 없습니다. 대신 v-bind를 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**에 전달할 수 있다.  
      <div v-bind: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.

<ul id="example-2">

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

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

</li>

</ul>

var example2 = new Vue({

el: '#example-2',

data: {

parentMessage: 'Parent',

items: [

{ message: 'Foo' },

{ message: 'Bar' }

]

}

})

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

<ul id="v-for-object" class="demo">

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

{{ value }}

</li>

</ul>

new Vue({

el: '#v-for-object',

data: {

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>

* 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>

* + ㅇ
* ㅇㅇ

# 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>