

# Vue.js

DalSoft 개발팀

## 목차

1. Vue.js 소개
2. 개발환경
   1. Web Browser 선택 및 설치
   2. Visual Studio Code 설치하기 (IDE)
   3. Node.js (NPM) 설치
   4. Vue CLI를 통한 프로젝트 구성 (Vue CLI 2.x 기준)
   5. 개발 환경 세팅및 확장도구 설치
3. Vue.js 사용하기
   1. 템플릿 문법
   2. 계산된 속성과 감시자
   3. 클래스와 스타일 바인딩
   4. 조건부 렌더링
   5. 리스트 렌더링
   6. 이벤트 핸들링
   7. 폼 입력 바인딩
4. 컴포넌트
   1. 컴포넌트 사용법
   2. Props
   3. Events
   4. Slot
   5. Ref
5. Vue 라우터
   1. 라우터 사용방법
   2. 동적 라우트 매칭
   3. 중첩된 라우트
   4. 프로그래밍 방식 네비게이션
   5. 라우트 컴포넌트에 속성 전달
   6. HTML5 히스토리 모드
6. Vue와 관련된 프로젝트
   1. Vuex

## Vue.js 소개

Vue(/vjuː/ 로 발음하며 view 와 발음 동일).js는 Evan You가 만들었으며, 2014년 릴리즈를 시작했다.

Vue.js는 발음대로 철저히 뷰(View)에 최적화된 프레임워크이다. **컨트롤러 대신 뷰 모델**을 가지는

**MVVM(Model-View-ViewModel)패턴**을 기반으로 디자인 되었으며, **컴포넌트(Components)**의 **재사용**이 가능하다.

## 개발환경

Vue.js를 이용해 프론트엔드를 개발하고 테스트후 배포를 하기 위해서는 아래와 같은

환경이 구축되어 있어야 합니다.

[ Web Browser, IDE, Node.js(NPM), Vue.js ]

### Web Browser 선택 및 설치

Web Browser는 웹 표준을 따르고 있는 Chrome 브라우저를 설치해 개발한 Vue 코드를 테스트 합니다.

다운로드 링크 : <http://www.google.co.kr/chrome>

### Visual Studio Code 설치하기 (IDE)

통합 개발 환경( IDE )은 가볍고 빠르며 스마트한 IntelliSense 와 HTML, 자바 스크립트 및 CSS언어를 지원하는 무료 툴인 Visual Studio Code를 사용 합니다.

다운로드 링크 : <https://code.visualstudio.com/>

### Node.js (NPM) 설치

Node.js 와 NPM 설치하기, NPM의 경우는 Node Package Manager 이기 때문에 Node.js를 설치 하면 같이 설치 됩니다.

다운로드 링크 : <https://nodejs.org/en/>

※ Node.js 와 NPM 이란?

Node.js 공식 사이트에서 Node.js를 이렇게 정의 하고 있습니다.

Node.js 란 Chrome V8 Javascript 엔진으로 빌드된 JavaScript 런타임이고, Node.js는 이벤트 기반, Non 블로킹 I/O 모델을 사용해 가볍고 효율적입니다.

NPM은 Node Package Manager 또는 Modules로 Node.js 기반의 모듈을 모아둔 집합 저장소라 생각할 수 있습니다.

Node.js가 정상적으로 설치 되었는지 확인을 원하면 아래 명령어를 실행해 보면 됩니다.

$ node -v

NPM이 정상적으로 설치 되었는지 확인을 원하면 아래 명령어를 실행해 보면 됩니다.

$ npm -v

### Vue CLI를 통한 프로젝트 구성 (Vue CLI 2.x 기준)

※ Vue.js를 사용하는 방법은 Script에 CDN을 추가하여 사용하거나 NPM으로 Vue를 인스톨 해서 사용하는 방법등 다양한 방법이 있지만 단일 페이지 응용프로그램을 빠르게 스캐폴딩하기 위한 공식 CLI를 제공하므로 Vue-CLI를 통한 프로젝트 구성을 추천합니다.

* 설치

Vue CLI의 설치 최소 요구사항은 Node.js 4.x이상(6.x 버전 추천)과 Git이 설치 되어야

한다.

1. 명령 프롬프트(CMD) 혹은 PowerShell을 실행

Visual Studio Code에서 실행 후 폴더 선택 후 터미널 실행 ( Ctrl + ` )으로 가능

1. ‘npm install --global vue-cli ’ 를 실행

* 프로젝트 만들기

프로젝트 생성은 원하는 경로에 다음과 같은 명령어를 사용한다.

$ vue init <template> <project-name>vue

* 템플릿 설정

Vue-CLI 에서는 미리 세팅된 몇가지 템플릿이 존재한다. 해당 템플릿은 “vue list”를

통해 확인 할 수 있다.

기본적인 템플릿은 4가지가 있다.

* browserify
* browserify-simple
* webpack
* webpack-simple

ex) “my-project”를 “webpack” 템플릿으로 하는 경우

$ vue init webpack my-project

* ESLint preset

해당 부분에서는 javacript 코딩스타일을 정하는 부분이다.

* + Standard
  + AirBNB
  + none(사용자정의)
* Unit Test 설정

유닛 테스트 여부 설정

* E2E Test 설정

전체적인 프로젝트(End - to - End)의 테스트 여부 설정

* 프로젝트 실행

1. 생성한 vue 프로젝트로 이동
2. 터미널에 다음 명령어 실행
   1. $ npm run dev

### 개발 환경 세팅 및 확장도구 설치

개발을 조금 더 빠르고 편리하게 하기 위해 회사에서 지향하는 개발 환경 세팅 및

개발에 도움이 되는 확장도구 설치 방법 입니다.

우선, Visual Studio Code의 확장 플러그인 입니다.

* + Auto Close Tag

HTML/XML 코드의 닫는 태그를 자동으로 생성해 주는 확장 도구

* + Bracket Pair Colorizer

괄호가 여러 개 겹쳐 있을 시 색상으로 표시해 주는 확장 도구

* + Debugger for Chrome

Visual Studio Code디버그 모드와 크롬 브라우저를 통한 디버깅을 제공

* + ESLint

자바스크립트 문법 중 에러가 있는 곳이나 잠재적인 문제를 표시해주는 확장 도구

**사용방법**

1. ESLint를 적용할 프로젝트를 연다.
2. F1 키를 누른다.
3. “ESLint : Create ‘.eslintrc.json’ Files”을 명령어 입력후 엔터키를 입력하여 .eslintrc.json파일을 생성한다.
4. ‘.eslintrc.json’파일이 생성된 프로젝트는 작업시 자동으로 ESLint가 적용된다.
   * vscode-icons

파일 및 폴더에 대한 아이콘 확장 도구

* + HTML Boilerplate

HTML5 방식의 기본틀을 작성하는 Snippet 확장 도구

* + IntelliSense for CSS class names

HTML 작성시 자동으로 CSS의 Class 명을 찾아주는 확장도구

* + JavaScript(ES6) code snippet:

ES6 문법에 대한 자바스크립트 Snippet

**지원 대상**

1. 자바스크립트
2. 타입스크립트
3. 자바스크립트 리액트
4. 타입스크립트 리액트
5. HTML
6. Vue
   * Live Server

로컬 서버에서 HTML 페이지가 수정되면 즉시 반영 해주는 확장 도구

**사용법**

1. VSCode의 프로젝트에서 작업할 HTML 파일을 우클릭을 한다.
2. Open with Live Server 메뉴를 클릭한다.
   * View In Browser

HTML 파일을 시스템에 기본으로 설정된 웹브라우저를 바로 띄우는 확장도구

**사용 방법**

1. HTML 파일을 선택한다.
2. “Ctrl” + “F1”를 누른다.
   * Path Intellisense

파일경로를 자동으로 찾아주는 확장도구

* + Npm Intellisense

프로젝트에 설치된 npm 모듈 중 원하는 모듈을 자동으로 추가하는 확장도구

**사용법**

1. F1 키를 누른다.
2. “Import Module”를 검색 후 엔터를 누른다.
3. 설치된 npm 모듈중 하나를 선택 후 엔터를 누른다.
   * Setting Sync

여러 컴퓨터에서 자신이 사용한 Snippet, 테마, 파일 아이콘 런처, 단축키, 작업 목록, 확장도구 등을 동기화 하는 확장 도구

**업로드 방법**

1. Shift + Alt + U를 눌러 GitHub 페이지를 연다.
2. GitHub 페이지에 로그인 한다.
3. 자신의 프로필을 클릭하고 Settings를 클릭한다.
4. Developer settings를 클릭한다.
5. Personal access tokens를 클릭한다.
6. Generate new token을 클릭한다.
7. Token description에 해당 Token의 이름을 작성한다.
8. 체크 박스중에 “gits”만 체크한다.
9. Generate token을 클릭한다.
10. 화면에 나타난 코드를 복사한다.
11. VSCode로 돌와 와서 1번에서 나타난 창에 복사한 코드를 붙이고 엔터키를 누른다.
12. 출력창에 GitHub Gist에 나온 코드를 복사한다.
13. 이후 1번에 나온 단축키만 입력 시 셋팅 업로드가 진행 된다.

**동기화 다운로드 방법**

1. Shift + Alt + D를 눌러 동기화 다운로드 창을 연다.
2. 업로드시 GitHub Gist에 나온 코드를 입력후 엔터를 한다.

## Vue.js 사용하기

Vue.js는 공식 홈페이지에 한국어로 친절하게 설명되어 있지만 기본적으로 Vue.js 프로젝트를 시작하기 위해서 다음과 같은 내용을 알고있어야 한다.

### 템플릿 문법

Vue.js는 렌더링 된 DOM을 기본 Vue인스턴스의 데이터에 선언적으로 바인딩 할 수있는 HTML 기반 템플릿 구문을 사용한다. 데이터 바인딩의 가장 기본 형태는 “Mustache”구문(이중 중괄호)을 사용한 텍스트 보간이다.

<span>메시지: {{ msg }}</span>

Mustache태그는 해당 데이터 객체의 msg속성 값으로 대체된다. 또한 msg속성이 변경될 때마다 갱신된다. Mustache태그는 HTML이 아닌 일반 텍스트로 데이터를 해석한다. 실제 HTML을 출력하려면 v-html 디렉티브를 사용해야 한다.

<p>Using v-html directive: <span v-html="rawHtml"></span></p>

span의 내용은 rawHtml로 대체된다. 이때 데이터 바인딩은 무시된다.

Mustaches는 HTML 속성에서 사용할 수 없다. 이때는 v-bind디렉티브를 사용한다.

<div v-bind:id="dynamicId"></div>

Vue.js는 데이터 바인딩 내에서 JavaScript 표현식의 모든 기능을 지원한다.

{{ number + 1 }}

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

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

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

이 표현식은 Vue 인스턴스 범위내에서 JavaScript로 계산되며 각 바인딩에서 하나의 단일 표현식만 포함될수 있으므로 아래처럼 작성하면 안된다.

<!-- 아래는 구문이므로 표현식은 안된다. -->

{{ var a = 1 }}

<!-- 조건문은 작동하지 않으므로 삼항 연산자를 사용해야 한다. -->

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

v-접두사가 있는 특수속성을 디렉티브라고 한다. 디렉티브 속성값은 단일 JavaScript표현식이 된다.(v-for은 예외) 디렉티브의 역할은 표현식의 값이 변경될때 사이드 이팩트를 반응적으로 DOM에 적용하는 것이다.

<p v-if="seen">이제 나를 볼 수 있어요</p>

여기서, v-if 디렉티브는 seen의 Boolean값에 따라 <p>엘리먼트를 제거 또는 삽입한다.

일부 디렉티브는 콜론으로 표시되는 “전달인자”를 사용할 수 있다. 예를 들어, v-bind 디렉티브는 반응적으로 HTML속성을 갱신하는데 사용된다.

<a v-bind:href="url"> ... </a>

여기서 href는 전달인자로, 엘리먼트의 href 속성을 표현식 url 값에 바인드 하는 v-bind 디렉티브에게 알려준다. DOM이벤트를 수신하는 v-on디렉티브에서의 전달인자는 이벤트를 받을 이름이다.

<a v-on:click="doSomething"> ... </a>

Vue.js에서 가장 자주 사용되는 v-bind와 v-on에 대해서는 특별한 약어가 제공된다.

v-bind

<!-- 전체 문법 →

<a v-bind:href="url"> ... </a>

<!-- 약어 →

<a :href="url"> ... </a>

v-on

<!-- 전체 문법 →

<a v-on:click="doSomething"> ... </a>

<!-- 약어 →

<a @click="doSomething"> ... </a>

### 계산된 속성과 감시자

* 계산된 속성

템플릿 내에서 사용하는 표현식은 매우 편리하지만 단순한 연산에만 사용해야 한다. 너무 많은 로직을 템플릿에 넣으면 유지보수가 어려워 질 수 있다.

<div id="example">

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

</div>

위와 같이 데이터를 여러번 호출하는 복잡한 로직의 경우 반드시 계산된 속성을 사용해야 한다.

<div id="example">

<p>원본 메시지: "{{ message }}"</p>

<p>뒤집히도록 계산된 메시지: "{{ reversedMessage }}"</p>

</div>

var vm = new Vue({

el: '#example',

data: {

message: '안녕하세요'

},

computed: {

// 계산된 getter

reversedMessage: function () {

// `this` 는 vm 인스턴스를 가리킵니다.

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

}

}

})

--------------------------------------------------------------

결과:

원본 메시지: “안녕하세요”

뒤집히도록 계산된 메시지: “요세하녕안”

----------------------------------------------------------------------

여기서 우리는 계산된 속성인 reversedMessage를 선언했다. 우리가 제공하는 함수는 vm.reversedMessage 속성에 대한 getter함수로 사용된다.

console.log(vm.reversedMessage) // => '요세하녕안'

vm.message = 'Goodbye'

console.log(vm.reversedMessage) // => 'eybdooG'

vm.reversedMessage의 값은 항상 vm.message의 값에 의존한다.

일반 속성처럼 템플릿의 계산된 속성에 데이터 바인딩을 할 수 있다. Vue는 vm.reversedMessage가 vm.message에 의존하는 것을 알고 있기 때문에 vm.message가 바뀔때 vm.reversedMessage에 의존하는 바인딩을 모두 업데이트 한다. 선언적으로 의존 관계를 만들었기 때문에 계산된 getter함수는 사이드 이펙트가 없어 테스트와 추론하기에 쉬워진다.

표현식에서 메소드를 호출하여도 같은 결과를 얻을수 있다.

<p>뒤집힌 메시지: "{{ reversedMessage() }}"</p>

// 컴포넌트 내부

methods: {

reversedMessage: function () {

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

}

}

최종결과에 대해 두가지 접근 방식은 서로 동일하다. 하지만 차이점은 계산된 속성은 종속성에 따라 캐시된다는 것이다. 계산된 속성은 종속성 중 일부가 변경된 경우에만 다시 계산된다. 이것은 message가 변경되지 않는 한, 계산된 속성인 reversedMessage에 대한 다중 접근은 함수를 다시 수행할 필요없이 이전에 계산된 결과를 즉시 반환 한다는 것을 의미한다. 하지만 메소드는 재 렌더링 할 때마다 항상 메소드를 호출한다.

Vue는 Vue인스턴스의 데이터 변경을 관찰하고 이에 반응하는 속성 감시 방법을 제공한다. 하지만 아래와 같은 경우 watch콜백보다 계산된 속성을 사용하는 것이 좋다.

<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

}

}

})

위의 코드는 반복이 필수적이다. 계산된 속성을 사용하면 더 간단하다.

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

var vm = new Vue({

el: '#demo',

data: {

firstName: 'Foo',

lastName: 'Bar'

},

computed: {

fullName: function () {

return this.firstName + ' ' + this.lastName

}

}

})

계산된 속성은 기본적으로 getter만 가지고 있지만 필요한 경우 setter를 제공할 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]

}

}

}

// ...

이제 vm.fullName = ‘John Doe’를 실행하면 설정자가 호출되고 vm.firstName과 vm.lastName이 그에 따라 업데이트 된다.

* 감시자

대부분의 경우 계산된 속성이 더 적합하지만 사용자 정의 감시자가 필요한 경우가 있다. 그래서 Vue는 watch옵션을 통해 데이터 변경에 반응 하는 보다 일반적인 방법을 제공한다. 이는 데이터 변경에 대한 응답으로 [비동기식 또는 시간이 많이 소요되는 조작](https://kr.vuejs.org/v2/guide/computed.html#%EA%B0%90%EC%8B%9C%EC%9E%90)을 수행하려는 경우에 유용하다.

### 클래스와 스타일 바인딩

* HTML 클래스 바인딩

클래스를 동적으로 토글하기 위해 v-bind:class에 객체를 전달할수 있다.

<div class="static"

v-bind:class="{ active: isActive, 'text-danger': hasError }">

</div>

data: {

isActive: true,

hasError: false

}

는 아래와 같이 렌더링 된다.

<div class="static active"></div>

isActive또는 hasError가 변경되면 클래스 목록도 그에 따라 업데이트 된다. 예를들어 hasError가 true가 되면 클래스 목록은 “static active text-danger”가 된다.

* 인라인 스타일 바인딩

v-bind:style 객체 구문은 거의 CSS처럼 보이지만 JavaScript객체이다. 속성이름에 camelCase와 kebab-case(따옴표를 함께 사용)를 사용할 수 있다.

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

data: {

activeColor: 'red',

fontSize: 30

}

스타일 객체에 직접 바인딩 하여 템플릿을 더 간결하게 만드는 것이 좋다.

<div v-bind:style="styleObject"></div>

data: {

styleObject: {

color: 'red',

fontSize: '13px'

}

}

### 조건부 렌더링

탬플릿에서 v-if 나 v-show를 이용해 다음과 같은 조건부 블록을 작성할 수 있다.

v-if

{{#if ok}}

<h1>Yes</h1>

{{/if}}

디렉티브를 사용하여 같은 결과를 얻을 수 있다.

<h1 v-if="ok">Yes</h1>

v-else와 함께 “else 블록”을 추가하는 것도 가능하다.

<h1 v-if="ok">Yes</h1>

<h1 v-else>No</h1>

Vue 2.1.0 부터는 v-else-if을 사용하여 “else if”블록을 만들수 있다.

<div v-if="type === 'A'">

A

</div>

<div v-else-if="type === 'B'">

B

</div>

<div v-else-if="type === 'C'">

C

</div>

<div v-else>

Not A/B/C

</div>

v-show

엘리먼트를 조건부로 표시하기 위한 또 다른 옵션은 v-show 디렉티브입니다. 사용법은

거의 동일하다.

<h1 v-show="ok">안녕하세요!</h1>

차이점은 v-show가 있는 엘리먼트는 항상 렌더링 되고 DOM에 남아있다는 점이다. v-show는 단순히 엘리먼트에 display CSS 속성을 토글한다.

### 리스트 렌더링

**v-for**

v-for 디렉티브를 사용하여 배열을 기반으로 리스트를 렌더링 할 수 있다. v-for

디렉티브는 item in items 형태로 특별한 문법이 필요하다. 여기서 items는 원본 데이터

배열이고 item은 반복되는 배열 엘리먼트의 별칭 이다.

기본 사용방법은 아래와 같다.

|  |
| --- |
| <ul id="example-1">  <li v-for="item in items">  {{ item.message }}  </li>  </ul> |

|  |
| --- |
| var example1 = new Vue({  el: '#example-1',  data: {  items: [  { message: 'Foo' },  { message: 'Bar' }  ]  }  }) |

### 이벤트 핸들링

Vue에서 태그의 이벤트를 핸들링을 하기 위해서는 **“v-on”** 지시자를 사용한다.

v-on 뒤에 DOM에서 사용하는 이벤트를 명시한다. 다음 예제를 보자

<div id="example-1">

<button v-on:click="counter += 1">Add 1</button>

<p>위 버튼을 클릭한 횟수는 {{ counter }} 번 입니다.</p>

</div>

var example1 = new Vue({

el: '#example-1',

data: {

counter: 0

}

})

현재 버튼에 “click” 이벤트를 받으면 counter를 1 증가하는 동작을 하게 만들었다.

간단한 로직은 위와 같은 방식으로 하면 되지만 복잡한 로직이 있는경우 위와 같은 방법을

한다면 복잡하다.

이런 경우 “methods”에 객체로 정의하면 된다.

<div id="example-2">

<!-- `greet`는 메소드 이름으로 아래에 정의되어 있습니다 -->

<button v-on:click="greet">Greet</button>

</div>

var example2 = new Vue({

el: '#example-2',

data: {

name: 'Vue.js'

},

// 메소드는 `methods` 객체 안에 정의합니다

methods: {

greet: function (event) {

// 메소드 안에서 사용하는 `this` 는 Vue 인스턴스를 가리킵니다

alert('Hello ' + this.name + '!')

// `event` 는 네이티브 DOM 이벤트입니다

if (event) {

alert(event.target.tagName)

}

}

}

})

// 또한 JavaScript를 이용해서 메소드를 호출할 수 있습니다.

example2.greet() // => 'Hello Vue.js!

메소드 객체를 바인딩하여 파라메터를 넣을 수도 있다.

<div id="example-3">

<button v-on:click="say('hi')">Say hi</button>

<button v-on:click="say('what')">Say what</button>

</div>

new Vue({

el: '#example-3',

methods: {

say: function (message) {

alert(message)

}

}

})

### 폼 입력 바인딩

<form /> 태그와 같이 입력이 가능한 태그에는 “양방향 데이터 바인딩”이라는 것이 가능하다.

Vue에서는 **‘v-model’** 이라는 디렉티브를 사용하여 양방향 데이터 바인딩을 사용 할 수 있다.

사용 방법은 다음과 같다.

<input v-model=”msg” />

※ input에서 v-model을 사용할 때 중국어, 일본어, 한국어가 필요한 경우 v-model 대신 input 이벤트를 대신 사용해야한다.

또한 체크박스와 같은 여러개를 체크가능한 곳에서 배열로 v-model를 통해 바인딩을 할 수 있다.

v-model 뒤에 **‘.\*’** 수식어를 사용하여 다양한 방법으로 이용 가능하다.

1. .lazy(기본 값)
   1. 입력 이벤트 후 데이터와 동기화를 한다.
2. .number
   1. 입력 된 문자열(String)이 자동으로 숫자(Number)로 변환 된다.
3. .trim
   1. 입력 된 문자열에 trim을 하여 데이터를 동기화 한다.

## 컴포넌트

### 컴포넌트 사용법

컴포넌트는 Vue의 가장 강력한 기능중 하나이다. 기본 HTML엘리먼트를 확장하여 재사용 가능한 코드를 캡슐화 하는데 도움이 된다.

전역 컴포넌트를 등록하려면, Vue.component(tagName, options)를 사용한다.

Vue.component('my-component', {

// 옵션

})

! Vue는 사용자 지정 태그 이름에 대해 W3C 규칙을 적용하지 않는다.(모두 소문자 이어야 하고 하이픈을 포함해야 한다.)그러나 이규칙을 따르는 것이 좋다.

일단 등록되면, 컴포넌트는 인스턴스의 템플릿에서 커스텀 엘리먼트, <my-component></my-component>로 사용할 수 있다.

<div id="example">

<my-component></my-component>

</div>

// 등록

Vue.component('my-component', {

template: '<div>사용자 정의 컴포넌트 입니다!</div>'

})

// 루트 인스턴스 생성

new Vue({

el: '#example'

})

아래와 같이 렌더링 된다.

<div id="example">

<div>사용자 정의 컴포넌트 입니다!</div>

</div>

지역 컴포넌트는 컴포넌트를 components인스턴스 옵션으로 등록함으로써 다른 인스턴스/컴포넌트의 범위에서만 사용할 수있는 컴포넌트를 만들수 있다.

var Child = {

template: '<div>사용자 정의 컴포넌트 입니다!</div>'

}

new Vue({

// ...

components: {

// <my-component> 는 상위 템플릿에서만 사용할 수 있습니다.

'my-component': Child

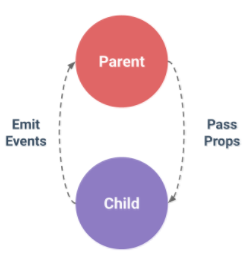
}

})

동일한 캡슐화는 디렉티브와 같은 다른 등록 가능한 Vue 기능에도 적용된다.

### Props

컴포넌트는 부모-자식 관계에서 가장 일반적으로 함께 사용하기 위한 것이다. 그들은 서로 의사소통이 필요하다. 부모는 자식에게 데이터를 전달해야 할 수도 있으며, 자식은 자신에게 일어난 일을 부모에게 알릴 필요가 있다.

Vue.js에서 부모-자식 컴포넌트 관계는 props는 아래로, events위로 라고 요약할 수 있다. 부모는 props를 통해 자식에게 데이터를 전달하고 자식은 events를 통해 부모에게 메시지를 보낸다.

props는 상위 컴포넌트의 정보를 전달하기위한 사용자 지정 특성이다. 하위 컴포넌트는 props옵션을 사용하여 수신 할 것으로 기대되는 props를 명시적으로 선언해야 한다.

Vue.component('child', {

// props 정의

props: ['message'],

// 데이터와 마찬가지로 prop은 템플릿 내부에서 사용할 수 있으며

// vm의 this.message로 사용할 수 있다.

template: '<span>{{ message }}</span>'

})

그런 다음 일반 문자열을 다음과 같이 전달할 수 있다.

<child message="안녕하세요!"></child>

--------------------------------------------------------------

결과:

안녕하세요!

----------------------------------------------------------------------

동적으로 props를 사용하려면 정규 속성을 표현식에 바인딩 하는 것과 비슷하게, v-bind를 사용하여 부모의 데이터에 바인딩 하면 된다. 데이터가 상위에서 업데이트 될 때마다 하위 데이터로도 전달된다.

<div>

<input v-model="parentMsg">

<br>

<child v-bind:my-message="parentMsg"></child>

</div>

v-bind에 대한 약어를 사용하는 것이 더 간단하다.

<child :my-message="parentMsg"></child>

### Events

모든 Vue인스턴스는 다음과 같은 이벤트 인터페이스를 구현한다.

* $on(eventName)을 사용하여 이벤트를 감지한다.
* $emit(eventName)을 하용하여 이벤트를 트리거 한다.

$on은 자식에서 호출한 이벤트는 감지하지 않는다. v-on을 템플릿에 반드시 지정해야 한다.

<div id="counter-event-example">

<p>{{ total }}</p>

<button-counter v-on:increment="incrementTotal"></button-counter>

<button-counter v-on:increment="incrementTotal"></button-counter>

</div>

Vue.component('button-counter', {

template: '<button v-on:click="incrementCounter">{{ counter }}</button>',

data: function () {

return {

counter: 0

}

},

methods: {

incrementCounter: function () {

this.counter += 1

this.$emit('increment')

}

},

})

new Vue({

el: '#counter-event-example',

data: {

total: 0

},

methods: {

incrementTotal: function () {

this.total += 1

}

}

})

이 예제에서 하위 컴포넌트가 외부에서 발생 하는 것과 완전히 분리 된다는 점에 유의해야 한다. 부모 컴포넌트가 신경 쓸 수 있는 경우를 대비하여 자체 활동에 대한 정보를 보고 하는 것뿐이다.

### Slot

부모 컴포넌트와 자식 컴포넌트를 섞을때 <slot>엘리먼트를 사용한다.

하위 컴포넌트 탬플릿에 최소한 하나의 <slot>콘텐츠가 포함되어 있지 않으면 부모 콘텐츠가 삭제된다. 속성이 없는 슬롯이 하나 뿐인 경우 전체 내용 조각이 DOM의 해당 위치에 삽입되어 슬롯 자체를 대체한다.

원래 <slot>태그 안에 있는 내용은 대체 콘텐츠로 간주된다. 대체 콘텐츠는 하위 범위에서 컴파일 되며 호스팅 엘리먼트가 비어있고 삽입할 콘텐츠가 없는 경우에만 표시된다.

다음은 템플릿으로 my-component라는 컴포넌트가 있다고 가정한다.

<div>

<h2>나는 자식 컴포넌트의 제목입니다</h2>

<slot>

제공된 컨텐츠가 없는 경우에만 보실 수 있습니다.

</slot>

</div>

그리고 그 컴포넌트를 사용하는 부모는

<div>

<h1>나는 부모 컴포넌트의 제목입니다</h1>

<my-component>

<p>이것은 원본 컨텐츠 입니다.</p>

<p>이것은 원본 중 추가 컨텐츠 입니다</p>

</my-component>

</div>

아래처럼 렌더링 된다.

<div>

<h1>나는 부모 컴포넌트의 제목입니다</h1>

<div>

<h2>나는 자식 컴포넌트의 제목 입니다</h2>

<p>이것은 원본 컨텐츠 입니다.</p>

<p>이것은 원본 중 추가 컨텐츠 입니다</p>

</div>

</div>

### Ref

props나 이벤트가 있었음에도 불구하고 때때로 JavaScript로 하위 컴포넌트에 직접 액세스 해야 할 수도 있다. 이런경우 ref를 이용하여 참조 컴포넌트 ID를 자식 컴포넌트에 할당해야 한다.

<div id="parent">

<user-profile ref="profile"></user-profile>

</div>

var parent = new Vue({ el: '#parent' })

// 자식 컴포넌트 인스턴스에 접근합니다.

var child = parent.$refs.profile

ref가 v-for와 함께 사용될 때, 얻을수 있는 ref는 데이터 소스를 미러링 하는 자식 컴포넌트를 포함하는 배열이 될 것이다.

$refs는 컴포넌트가 렌더링 된 후에만 채워지며 반응적이지 않다. 그것은 직접 자식 조작을 위한 escape해치를 의미한다. - 템플릿이나 계산된 속성에서 $refs를 사용하지 말아야 한다.

## Vue 라우터

Vue-Router

SPA(Single Page Application)을 만드는데 필요한 프로젝트로 컴포넌트를 route에 매핑하고

vue-router가 어디서 렌더링 할지 지정할 수 있다.

npm install vue-router 로 설치하며 모듈시스템과 함께 사용하는 경우 명시적으로 Vue.use()를 통해 라우터를 설치해야 한다. Vue-Cli를 사용하는 경우 옵션으로 자동 설치가 가능하다. <router-link

to="/foo">Go to Foo</router-link>를 이용해 원하는 컴포넌트로 이동이 가능하며 <router-view></router-view>에 라우트와 일치하는 컴포넌트가 렌더링된다.

[관련된 예제](http://jsfiddle.net/yyx990803/xgrjzsup/)

### 라우터 사용방법

라우터를 사용하기 위해서는 라우터를 설치 해야하는데 설치하는 방법은 크게 직접 다운로드 받거나 CDN으로 Script 링크로 사용할 수 있다.

[CDN]

<https://unpkg.com/vue-router/dist/vue-router.js>

[NPM install]

$ npm install vue-router

모듈 시스템에서 사용하려면 Vue.ues()를 통해 명시적으로 라우터를 추가해야 합니다.

[예시 코드]

import Vue from 'vue'  
import VueRouter from 'vue-router'  
  
Vue.use(VueRouter)

전역 스크립트 태그를 사용할 때는 이 작업을 하지 않아도 된다.

### 동적 라우트 매칭

주어진 패턴을 가진 라우트를 동일한 컴포넌트에 매핑해야하는 경우가 자주 있다. 예를 들어 모든 사용자에 대해 동일한 레이아웃을 가지지만 다른 사용자 ID로 렌더링되어야 하는 User 컴포넌트가 있을 수 있다. Vue-router에서 우리는 경로에서 동적 세그먼트를 사용할 수 있다.

const User = {  
 template: '<div>User</div>'  
}  
  
const router = new VueRouter({  
 routes: [  
 // 동적 세그먼트는 콜론으로 시작합니다.  
 { path: '/user/:id', component: User }  
 ]  
})

이제 /user/foo와 /user/bar 같은 URL은 모두 같은 경로에 매핑된다.

동적 세그먼트는 콜론 : 으로 표시된다. 라우트가 일치하면 동적 세그먼트 값은 모든 컴포넌트에서 this.$rout.params로 표시된다. 그러므로 User의 템플릿을 다음과 같이 갱신하여 현재 사용자 ID를 표현할 수 있다.

const User = {  
 template: '<div>User {{ $route.params.id }}</div>'  
}

\*Tip

매개 변수와 함께 라우트를 사용할 때 주의 해야할 점은 사용자가 /user/foo에서 /user/bar로 이동할 때 동일한 컴포넌트 인스턴스가 재사용된다는 것이다. 두 라우트 모두 동일한 컴포넌트를 렌더링하므로 이전 인스턴스를 삭제한 다음 새 인스턴스를 만드는 것보다 효울적 이다. **그러나 이는 컴포넌트의 라이프 사이클 훅이 호출되지 않음을 의미한다.**

이때, 동일한 컴포넌트의 params 변경 사항에 반응하려면 watch에서 $route 객체를 보면된다.

const User = {  
 template: '...',  
 watch: {  
 '$route' (to, from) {  
 // 경로 변경에 반응하여...  
 }  
 }  
}

### 중첩된 라우트

실제 앱 UI는 일반적으로 여러 단계로 중첩 된 컴포넌트로 이루어져 있다. URL의 세그먼트가 중첩 된 컴포넌트의 특정 구조와 일치한다는 것은 매우 일반적이다

vue-router를 사용하면 중첩 된 라우트 구성을 사용하여 이 관계를 표현하는 것이 매우 간단하다.

위에서 설명한 앱을 예를 들어 설명하면,

<div id="app">

<router-view></router-view>

</div>  
 const User = {  
 template: '<div>User {{ $route.params.id }}</div>'

}

const router = new VueRouter({  
 routes: [  
 { path: '/user/:id', component: User }  
 ]  
})

여기에있는 <router-view>는 최상위 outlet 이다. 최상위 경로와 일치하는 컴포넌트를 렌더링 한다. 비슷하게 렌더링 된 컴포넌트는 자신의 중첩된 <router-view>를 포함 할 수도 있다.

다음은 User 컴포넌트의 템플릿 안에 하나를 추가하는 예 이다.

const User = {  
 template: `  
 <div class="user">  
 <h2>User {{ $route.params.id }}</h2>  
 <router-view></router-view>  
 </div>  
 `  
}

이 중첩 outlet에 컴포넌트를 렌더링하려면 children을 사용해야 한다. VueRouter 생성자 옵션 config에서 아래와 같이 사용하면 된다.

const router = new VueRouter({  
 routes: [

{ path: '/user/:id', component: User,

children: [

{

// /user/:id/profile 과 일치 할 때

// UserProfile은 User의 <router-view> 내에 렌더링 됩니다.

path: 'profile',

component: UserProfile

},

{

// /user/:id/posts 과 일치 할 때

// UserPosts가 User의 <router-view> 내에 렌더링 됩니다.

path: 'posts',

component: UserPosts

}

]

}

]

})

/ 로 시작하는 중첩 된 라우트는 루트 경로로 취급된다. 이렇게하면 중첩 된 URL을 사용하지 않고도 컴포넌트 중첩을 활용할 수 있다.

children 옵션은 routes와 같은 라우트 설정 객체의 또 다른 배열이다. 따라서 필요한 만큼 중첩 된 뷰를 유지할 수 있다.

이 시점에서, 위의 설정으로, /user/foo를 방문했을 때 하위 라우트가 매치 되지 않았기 때문에 User의 outlet에 아무 것도 출력되지 않는다. 따라서 그곳에 무언가를 렌더링 하고 싶을때는 서브 루트 경로를 제공 할 수도 있다.

const router = new VueRouter({  
 routes: [  
 {  
 path: '/user/:id', component: User,  
 children: [  
 // UserHome은 /user/:id 가 일치 할 때  
 // User의 <router-view> 안에 렌더링됩니다.  
 { path: '', component: UserHome },  
  
 // ...또 다른 서브 라우트  
 ]  
 }  
 ]  
})

### 프로그래밍 방식 네비게이션

<router-link>를 사용하여 선언적 네비게이션용 anchor 태그를 만드는 것 외에도 라우터의 인스턴스 메소드를 사용하여 프로그래밍으로 이를 수행할 수 있다.

router.push(location, onComplete?, onAbort?)

|  |  |
| --- | --- |
| **선언적 방식** | **프로그래밍 방식** |
| <router-link :to="..."> | router.push(...) |

[사용 예시]

// 리터럴 string  
router.push('home')  
  
// object  
router.push({ path: 'home' })  
  
// 이름을 가지는 라우트  
router.push({ name: 'user', params: { userId: 123 }})  
  
// 쿼리와 함께 사용, 결과는 /register?plan=private 입니다.  
router.push({ path: 'register', query: { plan: 'private' }})

이외의 경로이동 메소드

router.push 이외에도 경로를 이동하는 다양한 메소드들이 존재하는데 간략하게 살펴본다면,

router.replace(location)

router.push와 같은 역할을 하지만 유일한 차이는 새로운 히스토리 항목에 추가하지 않고 탐색한다는 것이다. 이름에서 알 수 있듯이 현재 항목을 대체한다.

router.go

이 메소드는 window.history.go(n)와 비슷하게 히스토리 스택에서 앞으로 또는 뒤로 이동하는 단계를 나타내는 하나의 정수를 매개 변수로 사용한다.

[사용예시]

// 한 단계 앞으로 갑니다. history.forward()와 같습니다. history.forward()와 같습니다.  
router.go(1)  
  
// 한 단계 뒤로 갑니다. history.back()와 같습니다.  
router.go(-1)  
  
// 3 단계 앞으로 갑니다.  
router.go(3)  
  
// 지정한 만큼의 기록이 없으면 자동으로 실패 합니다.  
router.go(-100)  
router.go(100)

### 라우트 컴포넌트에 속성 전달

컴포넌트에서 $route를 사용하면 특정 URL에서만 사용할 수 있는 컴포넌트의 유연성을 제한하는 라우트와 강한 결합을 만든다.

컴포넌트와 라우터 속성을 분리하려면 아래 예시를 확인하자

[$Route에 의존성 추가]

const User = {  
 template: '<div>User {{ $route.params.id }}</div>'  
}  
const router = new VueRouter({  
 routes: [  
 { path: '/user/:id', component: User }  
 ]  
})

[속성에 의존성 해제]

const User = {  
 props: ['id'],  
 template: '<div>User {{ id }}</div>'  
}  
const router = new VueRouter({  
 routes: [  
 { path: '/user/:id', component: User, props: true },  
 ]  
})

props를 true로 설정하면 route.params가 컴포넌트 props로 설정됩니다.

### HTML5 히스토리 모드

vue-router의 기본 모드는 hash mode이다. URL 해시를 사용하여 전체 URL을 시뮬레이트 하므로 URL이 변경될 때 페이지가 다시 로드 되지 않는다.

해시를 제거하기 위해 라우터의 history 모드를 사용할 수 있다. history.pushState API를 활용하여 페이지를 다시 로드하지 않고도 URL탐색을 할 수 있다.

const router = new VueRouter({  
 mode: 'history',  
 routes: [...]  
})

히스토리 모드를 사용하면 URL이 “정상” 으로 보인다.

그러나 적절한 서버 설정이 없는 단일 페이지 클라이언트 앱이기 때문에 사용자가 직접 URL주소로 접근하면 404 오류가 발생한다.

이를 해결하려면 서버에 간단하게 포괄적인 대체 경로를 추가하기만 하면 된다. URL이 정적 에셋과 일치하지 않으면 앱이 있는 동일한 index.html 페이지를 제공해야 한다.

서버 설정에 관한 설명은 링크로 대체한다.

<https://router.vuejs.org/kr/essentials/history-mode.html>

## Vue와 관련된 프로젝트

### Vuex

Vuex는 Vue.js 애플리케이션에 대한 상태 관리 패턴 + 라이브러리이다. 애플리케이션의 모든 컴포넌트에 대한 중앙 집중식 저장소 역할을 하며 예측 가능한 방식으로 상태를 변경할 수 있다.

[참고자료]

1. Vue.js소개

* Vue.js 홈페이지 [https://kr.vuejs.or/](https://kr.vuejs.org/)
* vue.js소개 <http://meetup.toast.com/posts/99>

1. 개발환경

* Node.js의 개념

<https://www.infoworld.com/article/3210589/node-js/what-is-nodejs-javascript-runtime-explained.html>

* NPM의 개념 - <https://docs.npmjs.com/getting-started/what-is-npm>
* NPM Document - <https://docs.npmjs.com/>

1. Vue.js 기능
2. 컴포넌트
3. Vue 라우터

* Vue-router <https://router.vuejs.org/kr/>

1. Vue와 관련된 프로젝트

* Vuex <https://vuex.vuejs.org/kr/>
* Vuex 한국어로 설명된 사이트 <https://joshua1988.github.io/web-development/vuejs/vuex-start/>