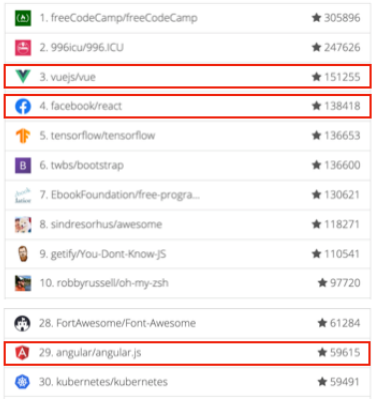
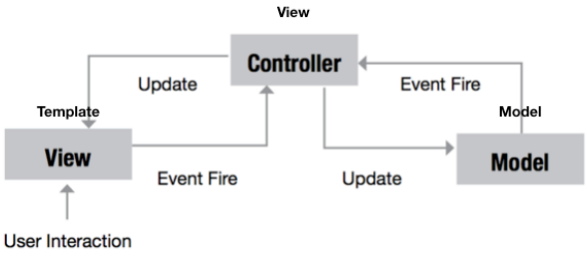
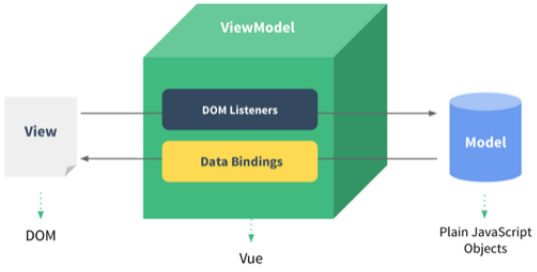
Intro to Vue


1. freeCodeCamp/freeCodeCamp ★ 305896
2. 996icu/996.ICU ★ 247626
3. vuejs/vue ★ 151255
4. facebook/react ★ 138418
5. tensorflow/tensorflow ★ 136653
6. twbs/bootstrap ★ 136600
7. EbookFoundation/free-progra... ★ 130621
8. sindresorhus/awesome ★ 118271
9. getify/You-Dont-Know-JS ★ 110541
10. robbyrussell/oh-my-zsh ★ 97720
28. FortAwesome/Font-Awesome ★ 61284
29. angular/angular.js ★ 59615
30. kubernetes/kubernetes ★ 59491


View
Controller
Update
Event Fire
Template
Model
View
Model
Event Fire
Update
User Interaction


ViewModel
DOM Listeners
Data Bindings
View
Model
DOM
Vue
Plain JavaScript Objects

Vue Instance

개념

모든 Vue App 은 `new Vue()` 함수로 새로운 인스턴스를 만드는 것부터 시작한다.

```
new Vue ({
  // options
})
```

Vue 인스턴스를 생성할 때, `data`, `template`, `el`, `methods`, 라이프 사이클 콜백함수 등을 포함 할 수 있는 options 객체를 전달한다.

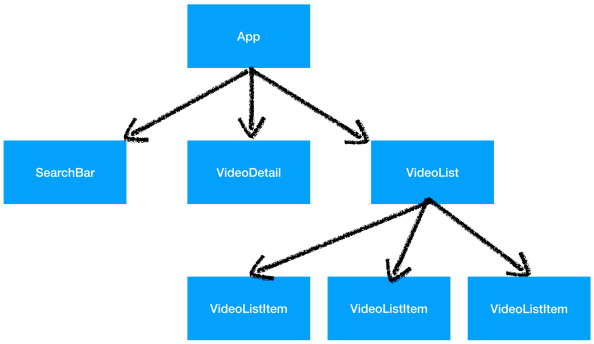
Vue Component

개념

컴포넌트는 Vue 의 가장 강력한 기능 중 하나다. 기본 HTML 요소를 확장하여 **재사용 가능한 코드를 캡슐화**한다.

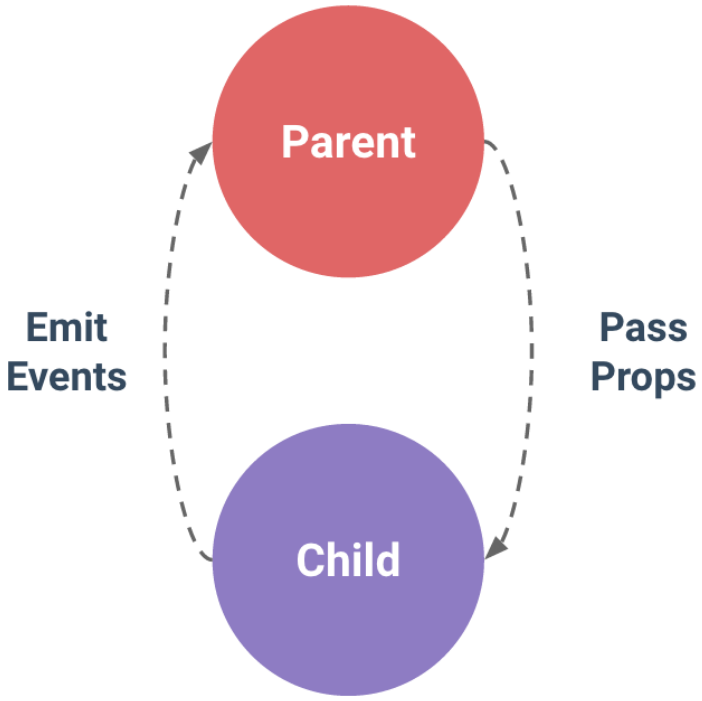
모든 Vue 컴포넌트는 Vue 인스턴스다. 그러므로, 모든 options 객체를 사용할 수 있다(root 에서만 사용할 수 있는 옵션은 제외).

한 Vue app 안에서 각 컴포넌트들은 Root 인스턴스를 시작으로 부모-자식 컴포넌트의 관계를 가진다.



Vue cli 환경에서 사용하는 방법과 일반적인 HTML, JS 에서 사용하는 방법이 다르다.

컴포넌트 간의 통신



컴포넌트는 부모-자식 관계에서 가장 일반적으로 함께 사용하기 위한 것이다.

컴포넌트 A는 자체 템플릿에서 컴포넌트 B를 사용할 수 있다. 그들은 필연적으로 서로 의사 소통이 필요하다. 부모는 자식에게 데이터를 전달해야 할 수도 있으며, 자식은 자신에게 일어난 일을 부모에게 알릴 필요가 있다.

그러나 부모와 자식이 명확하게 정의된 인터페이스를 통해 가능한 분리된 상태로 유지하는 것도 매우 중요하다. 이렇게하면 각 컴포넌트의 코드를 상대적으로 격리 할 수 있도록 작성하고 추론할 수 있으므로 유지 관리가 쉽고 잠재적으로 쉽게 재사용 할 수 있다.

Vue.js에서 부모-자식 컴포넌트 관계는 **props는 아래로, events 위로** 라고 요약 할 수 있다. 부모는 **props**를 통해 자식에게 데이터를 전달하고 자식은 **events**를 통해 부모에게 메시지를 보낸다.

Vue instance 의 속성들

Vue instance 초기화에 사용되는 속성들

`el`

마운트 할 대상 HTML 요소(노드)를 정의한다. Vue 인스턴스 초기화에서 `.$mount()` 로 대체 가능하다.

`data`

Vue 인스턴스가 최초 생성될 때의 속성값들을 정의한다. Vue 인스턴스가 최초 생성되면, `data` 객체 안의 모든 값들을 반응 시스템에 등록한다. 이 `data` 객체에 등록된 key 에 대해서만 value 수정에 대하여 반응하기 때문에, 최초 생성시에 할당할 값이 없다면, `''`, `[]`, `{}`, `0` 과 같은 값으로 초기화 해야 한다. 최소 생성 이후에 추가되는 `data` 는 Vue 가 반응하지 않기 때문이다.

Vue 는 `data` 가 변경되면 기본적으로 DOM 을 다시 렌더링 한다. (`v-once` 같은 디렉티브는 예외)

컴포넌트를 사용할 때에는 반드시 `data` 는 함수여야 하며, 객체를 `return` 해야 한다.

`methods`

사용할 다양한 함수들을 정의한다. 기본적으로 `data` 들을 조작한다.

모든 경우에 자유롭게 사용할 수 있지만, 만약 `methods` 에 정의한 함수가 단순히 `data` 의 속성을 가공해서 `return` 할 것이라면, 아래의 `computed` 를 사용하는 것이 좋다.

`computed`

`data` 를 수정하지 않고, 가공된 `data` 를 활용하고 싶을 때 사용한다. 절대로 화살표 함수를 사용해서 정의해서는 안된다.

반드시 `return` 이 있어야 하며, 함수를 작성하지만 실제 Vue 인스턴스에서는 `return` 된 (`data` 를 가공한) 값을 캐싱한다. 때문에 디렉티브에 연동해서 사용할 때, `()` 를 붙이면 에러가 발생한다. (때문에 이름을 지을 때, `return` 되는 값을 기준으로 명사형으로 짓는 걸 권장한다.)

`watch`

특정 `data` 가 수정되면 자동으로 실행될 watcher 함수들을 정의한다. 함수로 정의하며, key 는 변경을 감지할 `data` 의 key 값으로 작성한다.

다만, 같은 기능을 구현하기 위해 `computed` 속성과 `watch` 속성 두가지 방법을 모두 사용할 수 있는 경우가 많은데, `computed` 사용을 권장한다. Vue 의 기본적인 패러다임인 선언형 프로그래밍에 더 적합하며 코드도 간결해 지기 때문이다.

주로 특정 `data` 의 변경이 시간이 오래걸리는 비동기 작업을 발생시켜야 할 때 사용한다.

`template`

화면에 표시할 요소들(HTML) 을 정의한다.

Component 에서 활용되는 속성들

`name`

컴포넌트 사용 시, 해당 컴포넌트의 이름을 정의한다.

`components`

부모 컴포넌트에서 사용할 자식 컴포넌트의 이름(`name`)들을 작성한다.

`props`

`props` 시스템은 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달 할 때 사용한다.

자식 컴포넌트에 작성하게 되며, `props` 에 등록된 데이터들은 자식 컴포넌트에서 `data` 처럼 접근할 수 있다.

`props` 는 배열과 객체로 작성할 수 있는데, 단순히 전달 받을 `props` 의 이름만 나열하려면 배열로, 전달 받을 `props` 에서 검증하고 싶은 속성들을 명시하여 사전에 잘못된 `props` 전달을 차단하려면 객체를 사용한다. (그리고 객체로 사용하는 것을 권장한다.)

부모 컴포넌트에서 전달할 때, 정적인 값이라면 단순히 `propName="Value"` 와 같이 전달하지만, 일반적으로 동적으로 부모 컴포넌트의 `data` 와 연관된 값을 전달하게 된다. 이 경우에는 `v-bind` 를 활용한다.

Vue 컴포넌트들은 단방향 데이터 바인딩을 형성한다. 물이 위에서 아래로 흐르듯, 부모 컴포넌트의 업데이트는 자동으로 하위 컴포넌트의 업데이트를 일으키지만, 하위 컴포넌트의 변화는 부모 컴포넌트에게 자동 전달 되지 않는다.

때문에 자식 컴포넌트에서 부모 컴포넌트를 움직이게 하려면, 이벤트를 발생(`emit`)시켜야 한다.

모든 Vue 컴포넌트들은 `.$emit` 메서드를 통해 부모 컴포넌트가 들을 수 있는 이벤트를 발생시킬 수 있다. 일반적으로 자식 컴포넌트의 `method`에서 이벤트를 발생시키게 되고, 부모 컴포넌트에서 `v-on` 디렉티브를 통해 해당 이벤트를 탐지한다.

Vue Directives

개념

디렉티브란 **v-** 접두사가 있는 특수 속성이다. Directive(지시) 라는 단어에서 유추할 수 있듯, Vue 에게 지시하는 구문이다.

디렉티브 속성값은 **v-for** 를 제외하고는 모두 **단일 JS 표현식** 이다.

디렉티브의 역할은, 표현식의 값이 변경될 때, 반응적으로 DOM 에 적용하는 것이다.

`v-text`

Vanilla JS 에서 `.innerText` 와 동일한 기능을 수행한다. 태그를 일반 문자 상태로 보여준다.

보간법(`{{ }}`) 을 사용하는 것과 같으며 보간법 사용을 권장한다.

```
<template>
  <div id="app">
    <p v-text="message"></p>
    <p>{{ message }}</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      message: '<h1>hi</h1>',
    }
  }
}
</script>
```

v-html

Vanilla JS 에서 `.innerHTML` 과 동일한 기능을 수행한다. 태그를 파싱하여 화면에 구현한다.

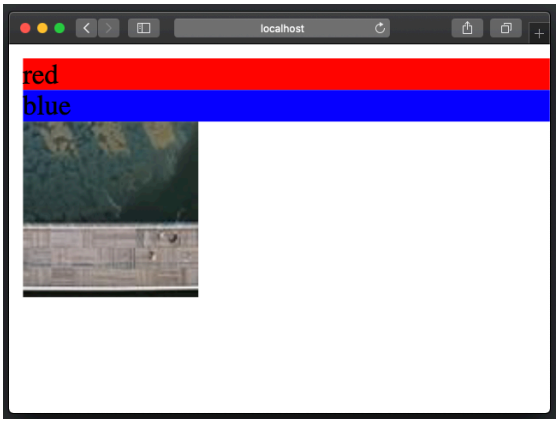
XSS(Cross Site Scripting) 공격에 주의해야 한다.

```
<template>
  <div id="app">
    <p v-html="message"></p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      message: '<h1>hi</h1>',
    }
  }
}
</script>
```

v-bind

HTML 태그의 값을 다루는 것이 아니라 속성을 다룰 때 사용한다. 기존 HTML5 속성 뿐만 아니라, Vue 내부에서 사용하는 속성들도 `v-bind` 를 통해 조작한다. 줄여서 `:` 로 사용 가능하다.

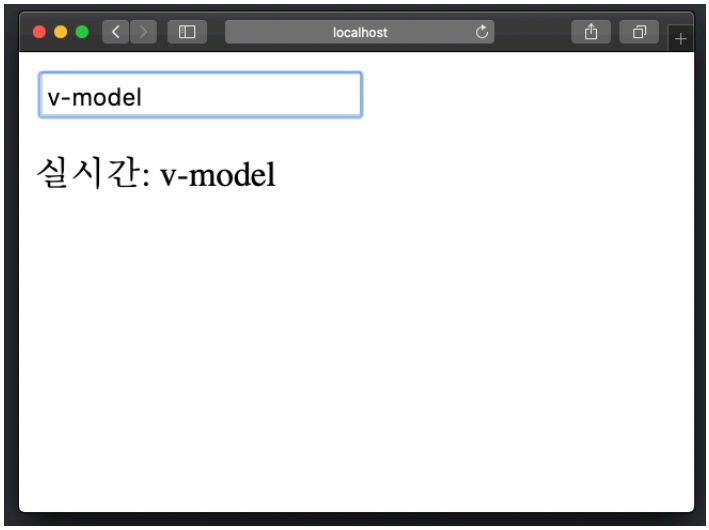
```
<template>
  <div id="app">
    <div v-bind:class="classRed">
      red
    </div>
    <div :class="classBlue">
      blue
    </div>
    <img :src="imageUrl">
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      imageUrl: 'https://picsum.photos/100',
      classRed: 'red',
      classBlue: 'blue',
    }
  }
}
</script>
<style>
  .red {background-color: red;}
  .blue {background-color: blue;}
</style>
```



v-model

`input` / `textarea` 와 같은 요소에서 사용자의 입력과 양방향 데이터 바인딩을 공유할 때 사용한다. 일반적인 HTML 에서의 초기값인 `value`, `checked`, `selected` 등의 속성을 무시한다.

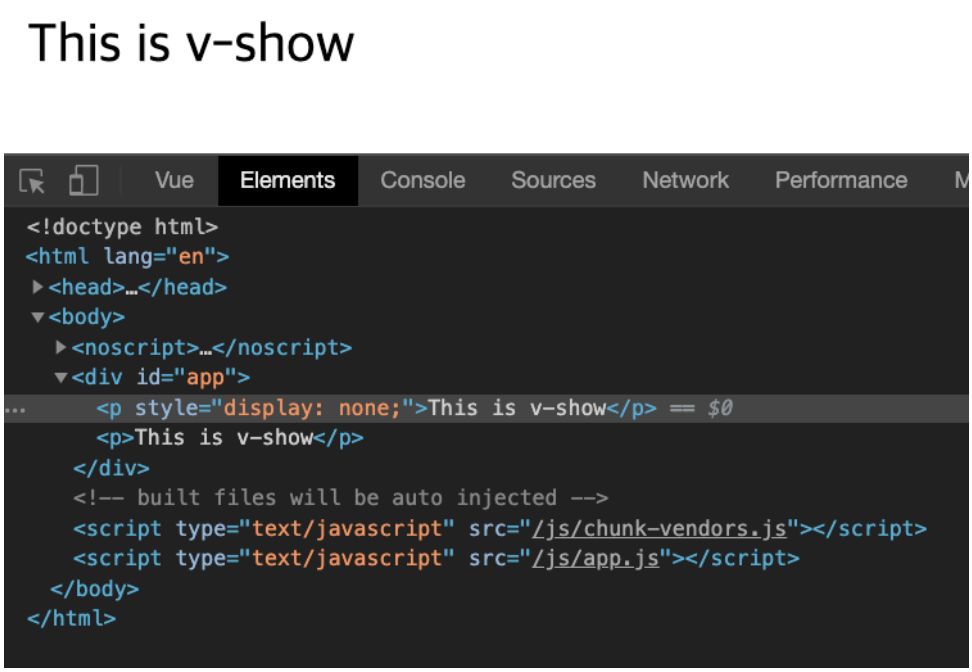
```
<template>
  <div id="app">
    <input type="text" v-model="inputData">
    <p>실시간: {{ inputData }}</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      inputData: ''
    }
  }
}
</script>
```



v-show

CSS `display: none;` 과 동일한 기능

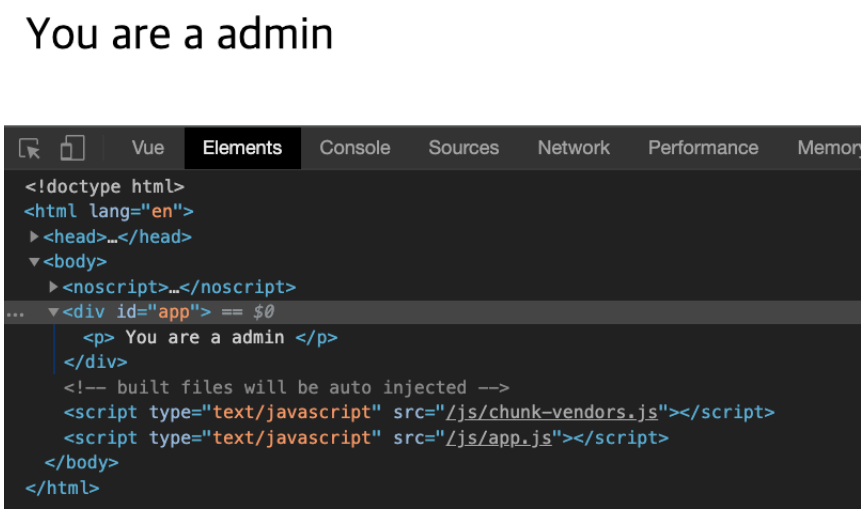
```
<template>
  <div id="app">
    <p v-show="false">{{ data }}</p>
    <p v-show="true">{{ data }}</p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      data: 'This is v-show',
    }
  }
}
</script>
<style></style>
```



`v-if`, `v-else-if`, `v-else`

`if`, `else if`, `else` 조건문을 표현함.

```
<template>
  <div id="app">
    <p v-if="userName === 'admin'">
      You are a admin
    </p>
    <p v-else-if="username === 'bad user'">
      Banned
    </p>
    <p v-else>
      You are normal user
    </p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      userName: 'admin',
    }
  }
}
</script>
<style></style>
```



`v-show` vs `v-if`

특정 요소를 보이게/안보이게 할 목적이라면, `v-if` 와 `v-show` 두가지 선택지가 있다.

해당 요소의 화면 표현 전환(on/off)이 잦다면, `v-show` 를 사용하는 것이 렌더링 비용이 적다.

반면 요소의 전환이 잦지 않고 고정되어 있다면, `v-if` 를 사용하는 것이 컴파일 비용이 적다. 아래는 공식문서 발췌

`v-if` vs `v-show`

`v-if` 는 조건부 블럭 안의 이벤트 리스너와 자식 컴포넌트가 토글하는 동안 적절하게 제거되고 다시 만들어지기 때문에 "진짜" 조건부 렌더링 입니다

`v-if` 는 또한 **게으릅니다** 초기 렌더링에서 조건이 거짓인 경우 아무것도 하지 않습니다. 조건 블록이 처음으로 참이 될 때 까지 렌더링 되지 않습니다.

비교해보면, `v-show` 는 훨씬 단순합니다. CSS 기반 토글만으로 초기 조건에 관계 없이 엘리먼트가 항상 렌더링 됩니다.

일반적으로 `v-if` 는 토글 비용이 높고 `v-show` 는 초기 렌더링 비용이 더 높습니다. 매우 자주 바꾸기를 원한다면 `v-show` 를, 런타임 시 조건이 바뀌지 않으면 `v-if` 를 권장합니다.

v-for

`for` 반복문을 구현한다. `v-bind:key` 를 필요로 한다.

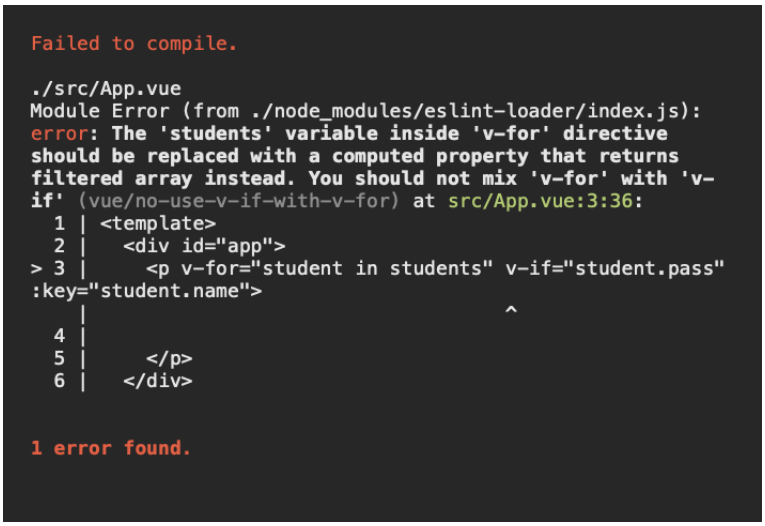
```
<template>
  <div id="app">
    <p>
      <span v-for="number in numbers" :key="`key-${number}`">
        {{ number }}
      </span>
    </p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      numbers: [1, 2, 3, 4],
    }
  }
}
</script>
<style></style>
```

`v-for` & `v-if`

한 HTML 태그에 `v-for` 와 `v-if` 를 같이 사용할 경우, 표기 순서에 상관없이 `v-for` 의 우선순위가 더 높다. 그리고 `v-for` 와 `v-if` 를 같이 사용하는 구문은 애초에 `computed` 속성을 통해 필터링 된 배열을 사용하는게 더 옳다.

Vue-CLI 환경에서는 기본적으로 `v-for` 와 `v-if` 를 함께 사용하면 에러를 발생시키도록 eslint 설정이 되어 있다.

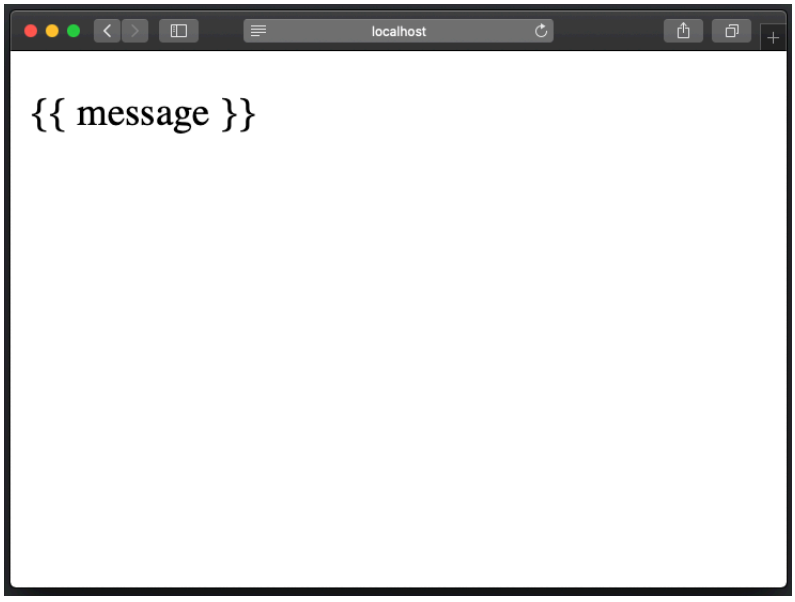
```
<template>
  <div id="app">
    <p v-for="student in students" v-if="student.pass" :key="student.name">

    </p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      testResults: [
        { name: 'ssafy', pass: true },
        { name: 'yfass', pass: false },
      ],
    }
  }
}
</script>
<style></style>
```



v-pre

Vue.js 가 컴파일 하지 않는다. 보간법(`{{ }}`)도 그대로 브라우저에 나타난다.

```
<template>
  <div id="app">
    <p v-pre>
      {{ message }}
    </p>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      message: 'hi'
    }
  }
}
</script>
<style></style>
```



`v-once`

최초 렌더링 단 한번만 수행한다. `data` 가 수정되어도 처음 렌더링 된 값 만을 보여준다.

`v-on` : 이벤트 핸들링

`v-on` 디렉티브를 사용하여 DOM 이벤트 핸들링이 가능하다. 줄여서 `@` 으로 줄여서 사용 가능하다.

활용 가능한 이벤트 명

활용 가능한 이벤트 명에는 다음과 같은 것들이 있다.

이벤트 이름	설명
`@click`	마우스를 클릭했을 때 실행
`@dbclick`	마우스를 더블 클릭했을 때 실행
`@mouseover`	마우스 포인터가 요소 위로 올라왔을 때 실행
`@mousemove`	마우스의 포인터가 요소 안에서 이동했을 때 실행
`@mousedown`	마우스의 버튼을 눌렀을 때 실행
`@mouseup`	마우스의 버튼을 놓았을 때 실행
`@keydown`	키보드의 키를 눌렀을 때 실행함
`@keyup`	키보드의 키를 놓았을 때 실행
`@keypress`	키보드의 키를 눌렀다가 놓았을 때 실행
`@change`	요소가 변경될 때 실행(`checkbox` 및 `radiobutton` 은 `@change` 를 사용)
`@input`	입력값이 변경될 때 실행(`input` 및 `textarea` 는 `@input` 을 사용해야 모든 변경에 반응)
`@submit`	`<form>` 이 제출될 때 실행
`@reset`	`<form>` 이 재 설정될 때 실행
`@select`	`<select>` 의 값이 선택되었을 때 실행
`@focus`	`<input>` 요소에 포커스가 있을 때 실행
`@blur`	`<input>` 요소가 포커스를 잃었을 때 실행함
...	...