Vue.js 기초문법

- **data**
  - Vue 앱의 상태 데이터를 정의하는 곳
  - Vue template에서 interpolation을 통해 접근할 수 있다.
  - `v-bind`, `v-on`과 같은 디렉티브에서도 사용 가능하다.
  - Vue 객체 내 다른 함수에서 `this` 키워드를 통해 접근할 수 있다.

```
<div id="app">{{ message }}</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hello SSAFY!',
  },
  methods: {
    greeting: function () {
      console.log(this.message)
    },
  },
})
</script>
```

- **v-html**
  - 주어지는 값을 요소의 `innerHTML` 로 할당한다.
  - XSS 공격으로 쉽게 이어질 수 있으므로 매우 위험할 수 있다.
  - 임의의 사용자로부터 입력 받은 내용은 `v-html`에 '절대' 사용하면 안된다.

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

<script>
  const app = new Vue({
    el: '#app',
    data: {
      message: '<script>alert("hi")<\/script>',
    },
  })
</script>
```

Vue.js 기초문법

- **v-text**
  - 주어지는 값을 요소의 innerText로 할당한다.
  - 내부적으로 interpolation 문법이 v-text로 컴파일된다.

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

<script>
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hello SSAFY!',
  },
})
</script>
```

- **v-if, v-else-if, v-else**
  - 조건문에 따라 렌더링 여부를 결정한다.
  - Cheap initial load, expansive toggle.

```
<div id="app">
  <div v-if="error" class="alert alert-danger" role="alert">
    {{ message }}
  </div>
  <div v-else-if="warning" class="alert alert-warning" role="alert">
    {{ message }}
  </div>
  <div v-else class="alert alert-primary" role="alert">
    {{ message }}
  </div>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hello SSAFY!',
    error: false,
    warning: false,
  },
})
</script>
```

Vue.js 기초문법

- **v-show**
  - 항상 렌더링, 조건문에 따라 노출 여부를 결정한다.
  - Expansive initial load, cheap toggle.

```
<div id="app">
  <!-- Modal -->
  <div v-show="modal" class="modal fade" id="exampleModal">
    ...
  </div>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    modal: false,
  },
})
</script>
```

- **v-for**
  - `item in array` 문법으로 배열의 수만큼 요소를 반복 렌더링 한다. 이 때 `item` 위치의 변수를 각 요소에서 사용할 수 있다.
  - `v-for` 사용 시 반드시 `:key` 속성을 각 요소에 작성해야 한다. 참조링크

```
<div id="app">
  <ul class="nav">
    <li v-for="menu in menus" :key="menu" class="nav-item">
      <a class="nav-link">{{ menu }}</a>
    </li>
  </ul>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    menus: ['home', 'login', 'register'],
    current: 'home',
  },
})
</script>
```

Vue.js 기초문법

- **v-bind**
  - HTML 요소의 속성에 Vue의 상태 데이터를 값으로 할당한다.
  - v-bind를 : 문자로 줄여서 사용 가능하다. (v-bind:key === :key)
  - Object의 형태로 사용하면 value가 true인 key가 class 바인딩 값으로 할당된다.

```
<div id="app">
  <ul class="nav">
    <li
      v-for="menu in menus"
      :key="menu"
      class="nav-item"
      :class="{ active: current === menu }"
    >
      <a class="nav-link" :href="`/${menu}`">{{ menu }}</a>
    </li>
  </ul>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    menus: ['home', 'login', 'register'],
    current: 'home',
  },
})
</script>
```

- **v-model**
  - HTML form 요소의 값과 data를 양방향 바인딩한다. 공식문서 예시
  - v-model.lazy
    - 요소에 input 이벤트 대신 change 이벤트 이후 data와 동기화한다.
  - v-model.trim
    - 바인딩 된 값의 앞뒤 공백을 자동으로 제거한다.
  - v-model.number
    - 바인딩 된 값을 숫자 타입으로 자동 형변환한다.

Vue.js 기초문법

- **methods**
  - Vue에서 응용할 함수를 정의하는 곳
  - Vue template에서 interpolation을 통해 접근할 수 있다.
  - v-on과 같은 디렉티브에서도 사용 가능하다.
  - Vue 객체 내 다른 함수에서 this 키워드를 통해 접근할 수 있다.

- **v-on**
  - 특정 이벤트가 발생했을 때, 주어진 코드가 실행된다.
  - v-on을 @ 문자로 줄여서 사용 가능하다. `v-on:click === @click`

```
<div id="app">
  <ul class="nav">
    <li
      v-for="menu in menus"
      :key="menu"
      class="nav-item"
      :class="{ active: current === menu }"
      @click="setMenu(menu)"
    >
      <a class="nav-link" :href="`/${menu}`">{{ menu }}</a>
    </li>
  </ul>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    menus: ['home', 'login', 'register'],
    current: 'home',
  },
  methods: {
    setMenu: function (menu) {
      this.current = menu
    },
  },
})
</script>
```

Vue.js 기초문법

- **computed**
  - 데이터를 기반으로 하는 계산된 속성
  - 함수의 형태로 정의하지만 함수가 아닌 함수의 반환값이 바인딩 된다.
  - 의존성을 띠고 있는 데이터의 값이 바뀌는 순간에만 다시 평가된다. (함수가 다시 실행된다.)

```
<div id="app">
  <p>{{ num }}</p>
  <p>{{ doubleNum }}</p>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    num: 2,
  },
  computed: {
    doubleNum: function () {
      return num * 2
    },
  },
})
</script>
```

- **watch**
  - 데이터를 감시하고, 데이터에 변화가 일어났을 때 실행되는 함수

```
<div id="app">
  <p>{{ num }}</p>
  <button @click="num++">add</button>
</div>

<script>
const app = new Vue({
  el: '#app',
  data: {
    num: 2,
  },
  watch: {
    num: function () {
      console.log(this.num, 'changed')
    },
  },
})
</script>
```

Vue.js 기초문법

- **computed & watch 예시**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>10. Vue computed & watch</title>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>
<body>
  <div id="app">
    <p>a: {{ a }}</p>
    <p>Computed: a의 제곱은 {{ square }} 입니다.</p>
    <p>Watch: a는 {{ increase }} 만큼 증가했습니다.</p>
    <input v-model.number="delta" type="number">
    <button @click="a = a + delta">a 증가</button>
  </div>
  <script>
    const app = new Vue({
      el: '#app',
      data: {
        a: 0,
        delta: 0,
        increase: 0,
      },
      computed: {
        square: function () {
          console.log('Computed!');
          return this.a**2
        },
      },
      watch: {
        a: function (newValue, oldValue) {
          console.log('Watch!')
          this.increase = newValue - oldValue
        },
      },
    })
  </script>
</body>
```

- computed: 특정 데이터를 직접적으로 사용/가공하여 다른 값을 만들 때 주로 사용한다.
- watch: 특정 데이터의 변화 상황에 맞춰 다른 data 등이 바뀌어야 할 때 주로 사용한다.

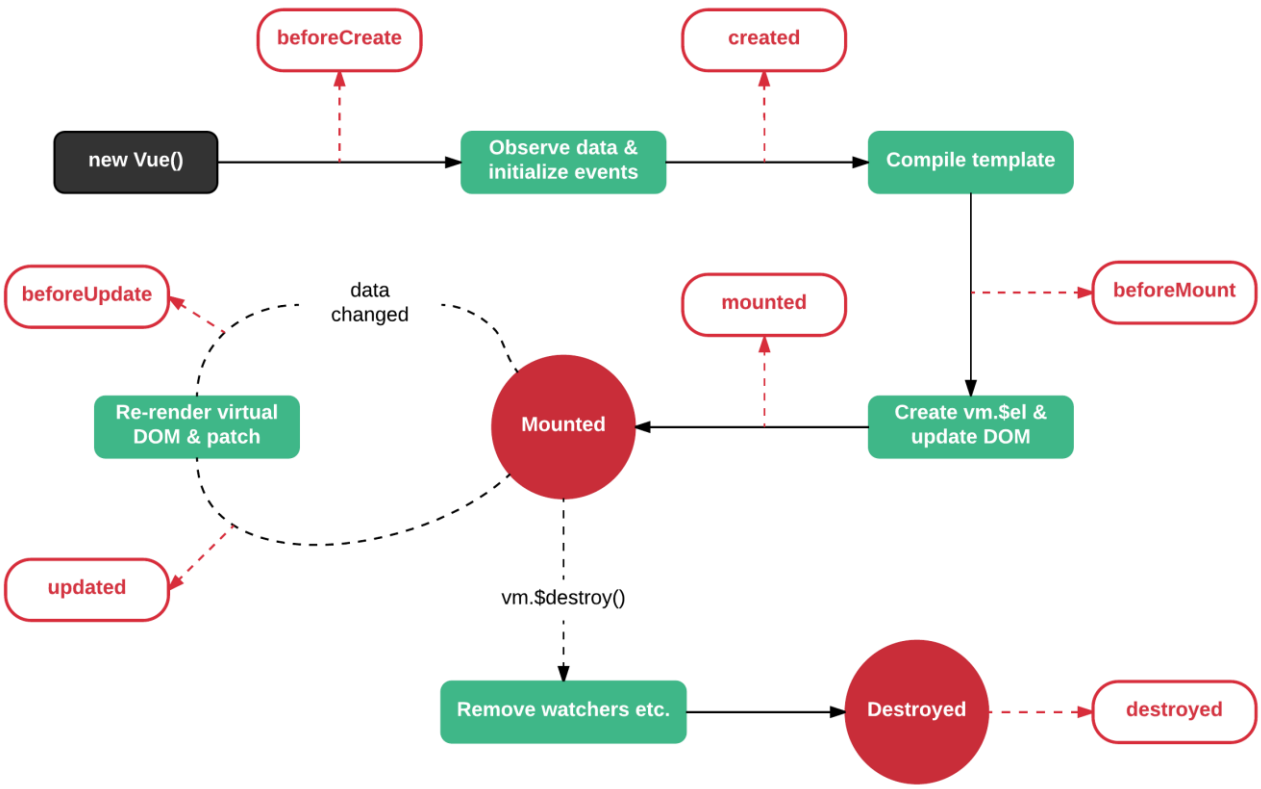
- **computed vs watch**

watch 속성은 감시할 데이터를 지정하고 그 데이터가 바뀌면 이런 함수를 실행하라는 방식으로 소프트웨어 공학에서 이야기하는 **'명령형 프로그래밍'** 방식.

computed 속성은 계산해야 하는 목표 데이터를 정의하는 방식으로 소프트웨어 공학에서 이야기하는 **'선언형 프로그래밍'** 방식.

Lifecycle Hooks

각 Vue 인스턴스는 생성될 때 일련의 초기화 단계를 거친다. 그 과정에서 사용자 정의 로직을 실행할 수 있는 **라이프사이클 훅** 도 호출된다.
공식문서를 통해 각 라이프사이클 훅의 상세 동작을 확인할 수 있다.



출처: codingexplained.com

Vue.js 기초문법

- **Example**

  created를 통해 어플리케이션의 초기 데이터를 API 요청을 통해 불러올 수 있다.

```
export default {
  data: function () {
    return {
      imgSrc: '',
    }
  },
  methods: {
    getImg: function () {
      axios.get(API_URL).then(response => {
        this.imgSrc = response.data.src
      })
    },
  },
  created: function () {     // Vue 인스턴스가 생성되면
    this.getImg()            // 이미지 데이터를 불러온다.
  },
})
```

```
<template>
  <div id="app">
    <img v-if="imgSrc" :src="imgSrc" />
    <p v-else>Image Loading..</p>
  </div>
</template>
```