Vue.js 입문자가 실무에서 주의해야 할 5가지 특징

FE 컨퍼런스 2019 / 장기효

목차

발표 대상

- Vue.js의 기본 문법을 알고 있는 웹 개발자
- Vue.js로 막 서비스 개발을 시작한 주니어 개발자
- Vue.js로 서비스를 제작중인 웹 개발자
- Vue.js의 특징이 궁금한 프런트엔드 개발자

발표 대상

- Vue.js의 기본 문법을 알고 있는 웹 개발자
- Vue.js로 막 서비스 개발을 시작한 주니어 개발자
- Vue.js로 서비스를 제작중인 웹 개발자
- Vue.js의 특징이 궁금한 프런트엔드 개발자

"Vue.js를 소개하는 발표가 아닙니다"

기대 효과

PR 보냈을 때 사수(시니어)한테 칭찬 받기

발표 자료 및 예제 소스는 아래 링크에서 확인

https://github.com/joshua1988/vue-five-common-mistakes

프로필

- (현) 프런트엔드 개발자
- 인프런, 패스트캠퍼스 Vue.js 강사
- 네이버, 이베이코리아, SK 등 기업 강의
- Do it! Vue.js 입문 저자
- Google 웹 기술 공식 사이트 번역

CAPTAIN P PANGYO

프로필

- (현) 프런트엔드 개발자
- 인프런, 패스트캠퍼스 Vue.js 강사
- 네이버, 이베이코리아, SK 등 기업 강의
- Do it! Vue.js 입문 저자
- Google 웹 기술 공식 사이트 번역

"빨리 배워 공유하는 걸 즐기는 웹 개발자"

1. 반응성

왜 내 화면은 다시 그려지지 않는 걸까?

뷰의 반응성이란

뷰의 반응성이란

- 데이터의 변화에 따라 화면이 다시 그려지는 뷰의 성질

뷰의 반응성이란

- 데이터의 변화에 따라 화면이 **다시 그려지는 뷰의 성질**

```
var vm = new Vue({
  data: {
    count: 0
  }
});
vm.count += 1; // count 값이 증가하면 화면에 표시된 count 값도 증가
```

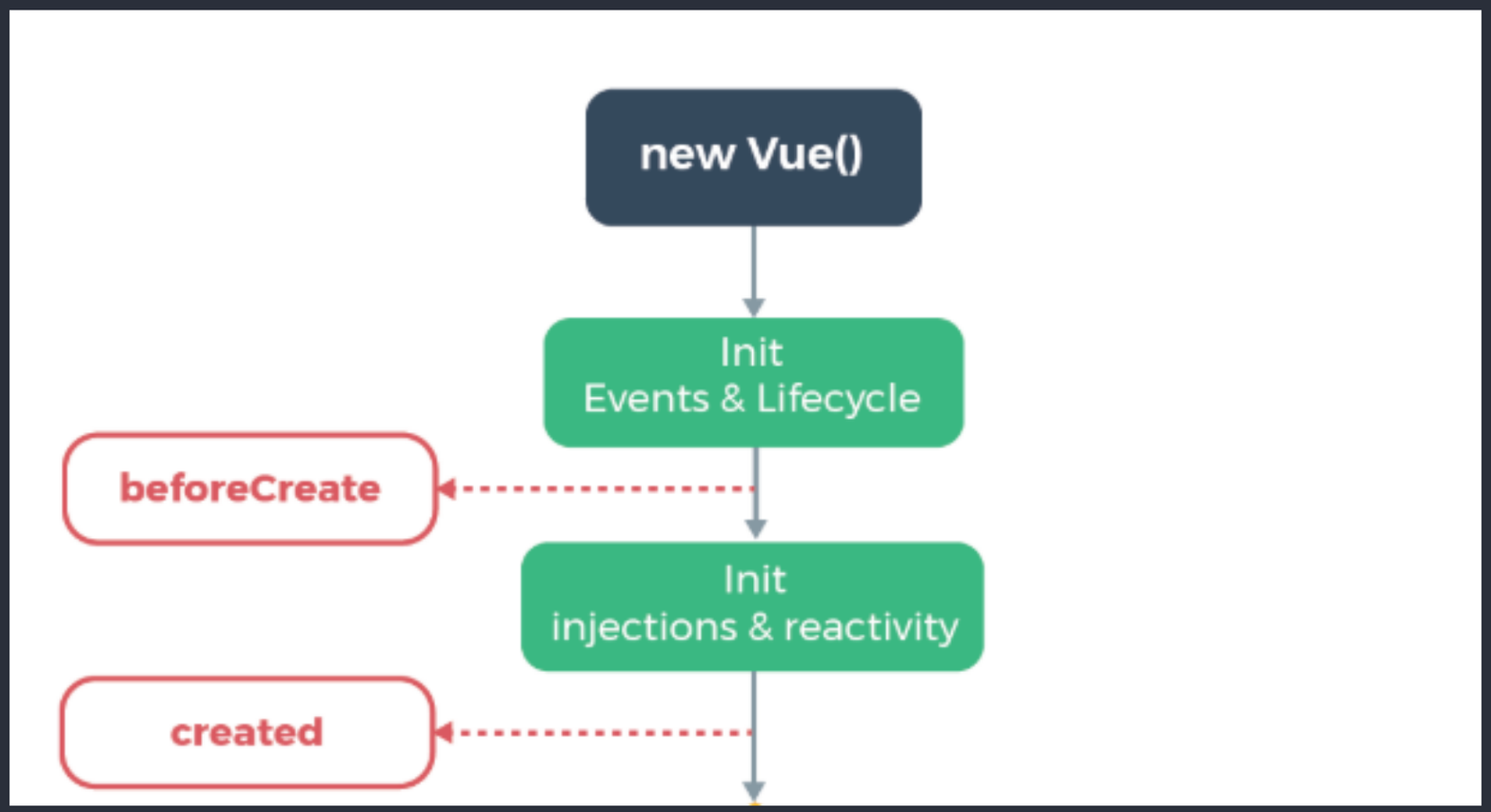
반응성은 언제 설정될까?

반응성은 언제 설정될까?

인스턴스가 **생성될 때** data의 속성들을 초기화

반응성은 언제 설정될까?

인스턴스가 **생성될 때** data의 속성들을 초기화



반응성에 대해 알아야 할 점

반응성에 대해 알아야 할 점

- **생성하는 시점에 없었던** `data`는 반응하지 않는다.

반응성에 대해 알아야 할 점

- **생성하는 시점에 없었던** data**는 반응하지 않는다.**

```
var vm = new Vue({
  data: {
    user: {
      name: 'Captain'
    }
  }
});
vm.user.age += 1; // age 값이 변하더라도 화면은 갱신되지 않음
```

반응성에 대해 알아야 할 점

- **생성하는 시점에 없었던** data**는 반응하지 않는다.**

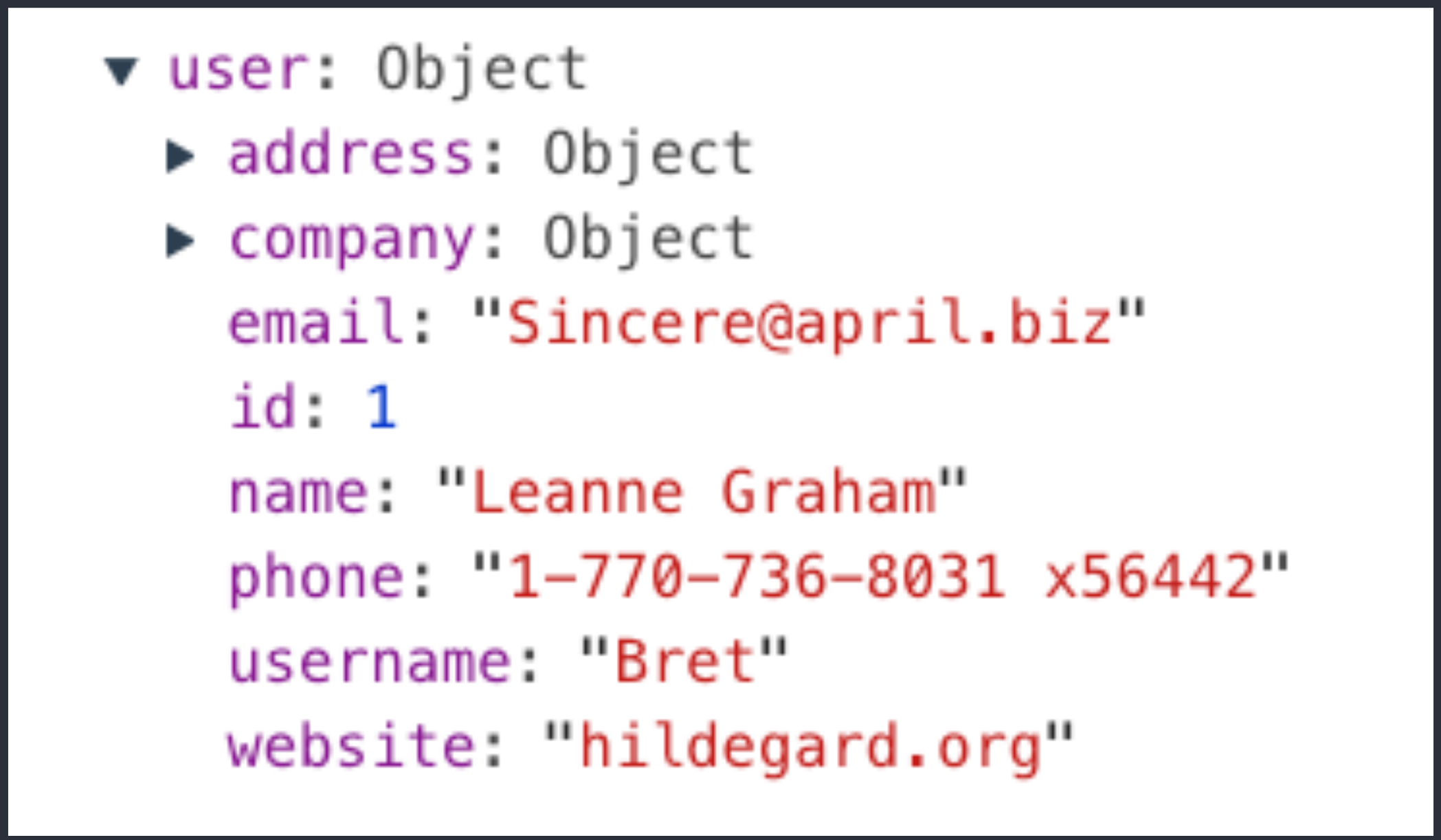
```
var vm = new Vue({
  data: {
    user: {
      name: 'Captain'
    }
  }
});
vm.user.age += 1; // age 값이 변하더라도 화면은 갱신되지 않음
```

반응성을 이해하지 못했을 때의 실수 1

화면에서만 필요한 UI 상태 값을 다룰 때

반응성을 이해하지 못했을 때의 실수 2

백엔드에서 불러온 데이터에 임의 값을 추가하여 사용하는 경우



[참고] 뷰엑스의 state도 data와 동일하게 취급

```
state: {
  user: { name: 'Captain' }
},
mutations: {
  // 생성하는 시점에 없었던 데이터는 반응성이 없음
  setUserAge: function(state) { state.user.age = 23; }, // X
  // 객체 속성을 임의로 추가 또는 삭제하는 경우 뷰에서 감지하지 못함
  deleteName: function(state) { delete state.user.name; } // X
}
```

뷰 3.0에서는 괜찮아요

Object.defineProperty()에서 Proxy 기반으로 변화

```
var obj = {};

// Vue 2.x
Object.defineProperty(obj, 'str', { .. });

// Vue 3.x
new Proxy(obj, { .. });
```

2. DOM 조작

오래된 습관 버리기

(기존) 화면 조작을 위한 DOM 요소 제어 방법

(기존) 화면 조작을 위한 DOM 요소 제어 방법

- 특정 DOM을 검색해서 제어하는 방법

```
// 네이티브 자바스크립트
document.querySelector('#app');
```

```
// 제이쿼리 라이브러리
$('#app');
```

(기존) 화면 조작을 위한 DOM 요소 제어 방법

- 사용자의 입력 **이벤트**를 **기반**으로 한 DOM 요소 제어

```
// 버튼 요소 검색
var btn = document.querySelector('#btn');
```

```
// 사용자의 클릭 이벤트를 기반으로 가장 가까운 태그 요소를 찾아 제거
btn.addEventListener('click', function(event) {
  event.target.closest('.tag1').remove();
});
```

(Vue.js 방식) ref 속성을 활용한 DOM 요소 제어

(Vue.js 방식) ref 속성을 활용한 DOM 요소 제어

- 뷰에서 제공하는 ref 속성

```
<!-- HTML 태그에 ref 속성 추가 -->
<div ref="hello">Hello Ref</div>
```

```
// 인스턴스에서 접근 가능한 ref 속성
this.$refs.hello; // div 엘리먼트 정보
```

(Vue.js 방식) 디렉티브를 활용한 DOM 요소 제어

- 뷰 디렉티브에서 제공되는 정보를 최대한 활용

```
<ul>
  <li v-for="(item, index) in items">
    <span v-bind:id="index">{{ item }}</span>
  </li>
</ul>
```

DOM 제어 사고 전환이 필요한 실제 사례

```
<ul>
  <li @click="removeItem">
    <span>메뉴 1</span>
    <div class="child hide">메뉴 설명</div>
  </li>
  <li @click="removeItem">
    <span>메뉴 2</span>
    <div class="child hide">메뉴 설명</div>
  </li>
  ...
</ul>
```

3. 인스턴스 라이프 사이클

나는 인스턴스를 얼마나 알고 있나?

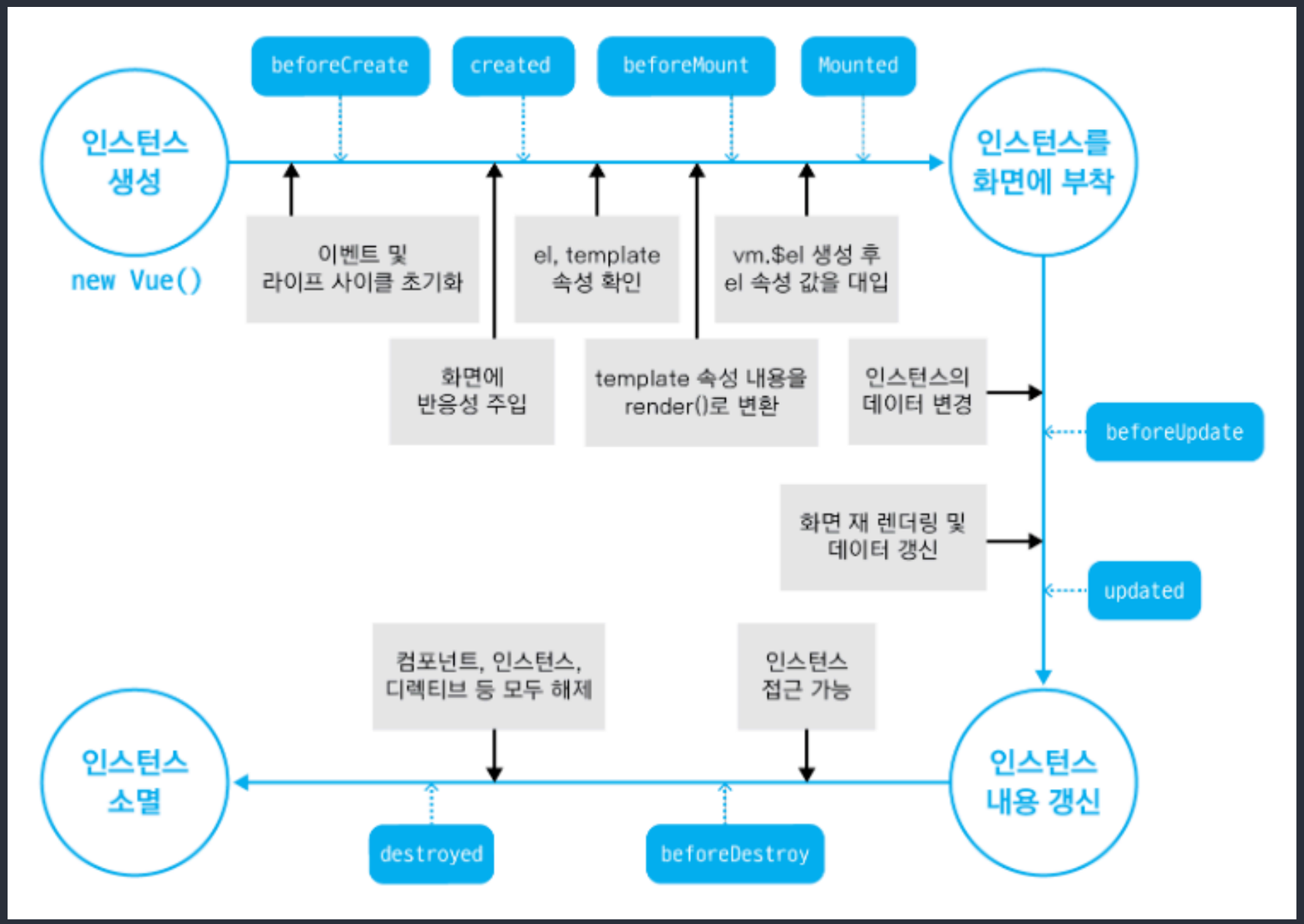
인스턴스 라이프 사이클이란?

인스턴스 라이프 사이클이란?

뷰 인스턴스가 생성되고 소멸되기까지의 생애 주기

인스턴스 라이프 사이클이란?

뷰 인스턴스가 **생성**되고 **소멸**되기까지의 생애 주기



35 — 이미지 출처: Do it! Vue.js 입문(이지스퍼블리싱)

뷰의 템플릿 속성

- 인스턴스, 컴포넌트의 표현부를 정의하는 속성

```
// 인스턴스 옵션 속성
new Vue({
  data: { str: 'Hello World' },
  template: '<div>{{ str }}</div>'
});

<!-- 싱글 파일 컴포넌트 -->
<template>
  <div>{{ str }}</div>
</template>
```

뷰의 템플릿 속성

- 인스턴스, 컴포넌트의 표현부를 정의하는 속성

```
// 인스턴스 옵션 속성
new Vue({
  data: { str: 'Hello World' },
  template: '<div>{{ str }}</div>'
});

<!-- 싱글 파일 컴포넌트 -->
<template>
  <div>{{ str }}</div>
</template>
```

뷰의 템플릿 속성

- 인스턴스, 컴포넌트의 표현부를 정의하는 속성

```
// 인스턴스 옵션 속성
new Vue({
  data: { str: 'Hello World' },
  template: '<div>{{ str }}</div>'
});

<!-- 싱글 파일 컴포넌트 -->
<template>
  <div>{{ str }}</div>
</template>
```

뷰 템플릿 속성의 정체

뷰 템플릿 속성의 정체

실제 DOM 엘리먼트가 아니라 Virtual DOM(자바스크립트 객체)

뷰 템플릿 속성의 정체

실제 DOM 엘리먼트가 아니라 Virtual DOM(자바스크립트 객체)

```
<!-- 사용자가 작성한 코드 -->
<template>
  <div>{{ str }}</div>
</template>
```

뷰 템플릿 속성의 정체

실제 DOM 엘리먼트가 아니라 Virtual DOM(자바스크립트 객체)

```
<!-- 사용자가 작성한 코드 -->
<template>
  <div>{{ str }}</div>
</template>

// 라이브러리 내부적으로 변환한 모습
function render() {
  with(this) {
    return _c('div', [_v(_s(str))]);
  }
}
```

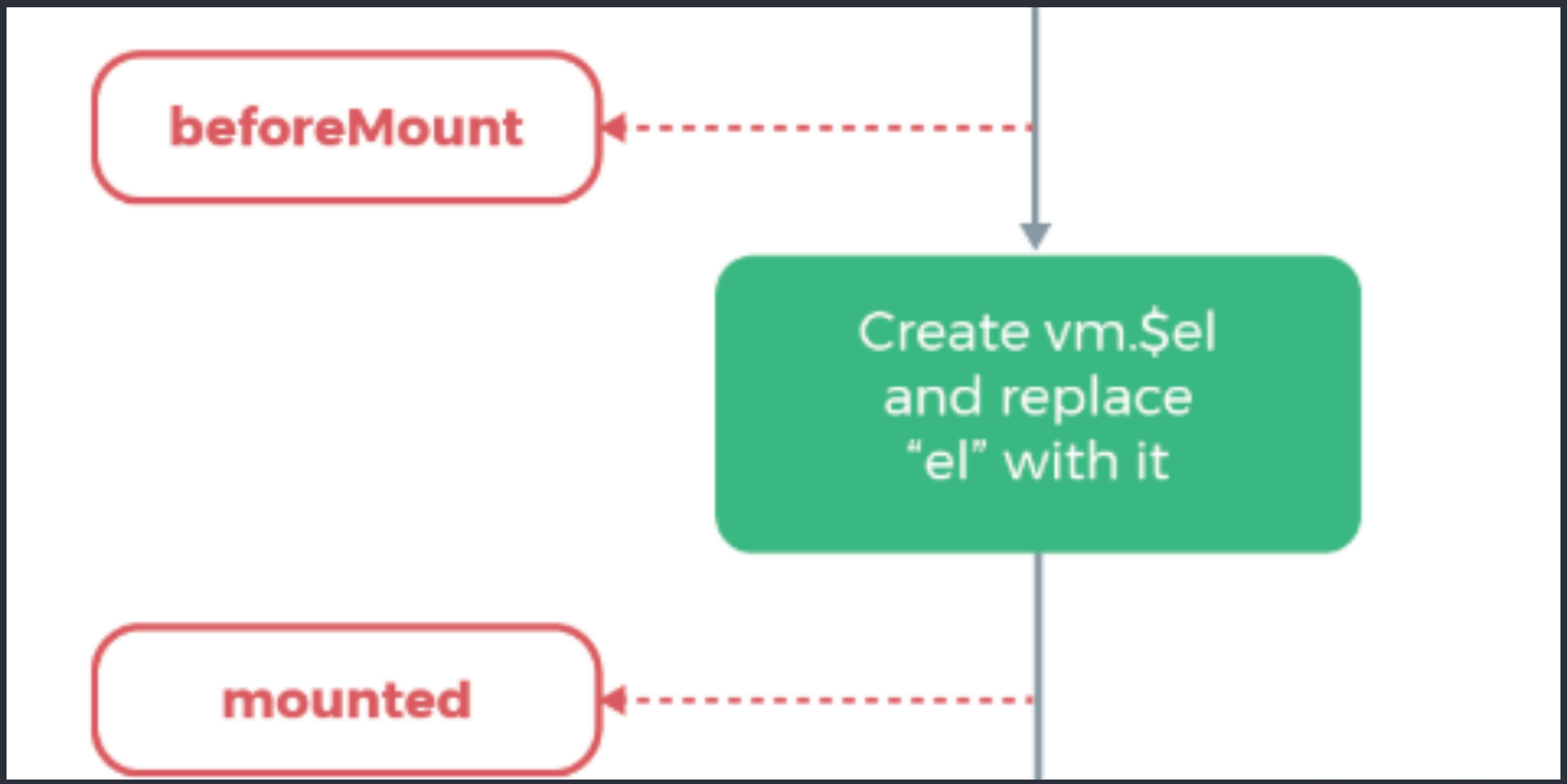
템플릿 속성이 실제로 유효한 시점

템플릿 속성이 실제로 유효한 시점

인스턴스가 화면에 **부착**(mounted)되고 난 후

템플릿 속성이 실제로 유효한 시점

인스턴스가 화면에 **부착**(mounted)되고 난 후



인스턴스 부착 시점을 이해하지 못한 사례 1

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  created: function() {
    var ctx = document.querySelector('#myChart');
    new Chart(ctx, chartOptions);
  }
});
```

인스턴스 부착 시점을 이해하지 못한 사례 1

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  created: function() {
    var ctx = document.querySelector('#myChart'); // null
    new Chart(ctx, chartOptions);
  }
});
```

인스턴스 부착 시점을 이해하지 못한 사례 1

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  created: function() {
    var ctx = document.querySelector('#myChart'); // null
    new Chart(ctx, chartOptions);
  }
});
```

인스턴스 부착 시점을 이해하지 못한 사례 2

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  created: function() {
    this.$nextTick(function() {
      var ctx = document.querySelector('#myChart');
      new Chart(ctx, chartOptions);
    })
  }
});
```

인스턴스 부착 시점을 이해하지 못한 사례 2

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  created: function() {
    this.$nextTick(function() { // 업데이트 시점 혼란 야기 및 코드 복잡도 증가
      var ctx = document.querySelector('#myChart');
      new Chart(ctx, chartOptions);
    })
  }
});
```

인스턴스 부착 시점을 이해하지 못한 사례 2

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  created: function() {
    this.$nextTick(function() { // 업데이트 시점 혼란 야기 및 코드 복잡도 증가
      var ctx = document.querySelector('#myChart');
      new Chart(ctx, chartOptions);
    })
  }
});
```

인스턴스 부착 시점을 이해한 코드

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>
```

```
// 인스턴스 옵션
new Vue({
  mounted: function() {
    var ctx = document.querySelector('#myChart');
    new Chart(ctx, chartOptions);
  }
});
```

인스턴스 부착 시점을 이해한 코드

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  mounted: function() {
    var ctx = document.querySelector('#myChart'); // <canvas>
    new Chart(ctx, chartOptions);
  }
});
```

인스턴스 부착 시점을 이해한 코드

```
<!-- 템플릿 속성 -->
<template>
  <canvas id="myChart"></canvas>
</template>

// 인스턴스 옵션
new Vue({
  mounted: function() {
    var ctx = document.querySelector('#myChart'); // <canvas>
    new Chart(ctx, chartOptions);
  }
});
```

4. ref 속성

만들다가 만 ref 속성

ref 속성이란?

ref 속성이란?

- 특정 DOM 엘리먼트나 하위 컴포넌트를 가리키기 위해 사용

ref 속성이란?

- 특정 DOM 엘리먼트나 하위 컴포넌트를 가리키기 위해 사용
- DOM 엘리먼트에 사용하는 경우 DOM 정보를 접근

ref 속성이란?

- 특정 DOM 엘리먼트나 하위 컴포넌트를 가리키기 위해 사용
- DOM 엘리먼트에 사용하는 경우 DOM 정보를 접근
- 하위 컴포넌트에 지정하는 경우 컴포넌트 인스턴스 정보 접근

ref 속성이란?

- 특정 DOM 엘리먼트나 하위 컴포넌트를 가리키기 위해 사용
- DOM 엘리먼트에 사용하는 경우 DOM 정보를 접근
- 하위 컴포넌트에 지정하는 경우 컴포넌트 인스턴스 정보 접근
- `v-for` 디렉티브에 사용하는 경우 Array 형태로 정보 제공

ref 속성이란?

- 특정 DOM 엘리먼트나 하위 컴포넌트를 가리키기 위해 사용
- DOM 엘리먼트에 사용하는 경우 DOM 정보를 접근
- 하위 컴포넌트에 지정하는 경우 컴포넌트 인스턴스 정보 접근
- `v-for` 디렉티브에 사용하는 경우 Array 형태로 정보 제공

특정 DOM 요소를 조작하고 싶을 때 사용하는 속성

ref 속성 사용할 때 주의할 점 1

- ref 속성은 템플릿 코드를 render 함수로 변환하고 나서 생성

ref 속성 사용할 때 주의할 점 1

- ref 속성은 템플릿 코드를 render 함수로 변환하고 나서 생성
- 접근할 수 있는 최초의 시점은 `mounted` 라이프사이클 훅

ref 속성 사용할 때 주의할 점 1

- ref 속성은 템플릿 코드를 render 함수로 변환하고 나서 생성
- 접근할 수 있는 최초의 시점은 mounted 라이프사이클 훅

```
<p ref="pTag">Hello</p>

created: function() {
  this.$refs.pTag; // undefined
},
mounted: function() {
  this.$refs.pTag; // <p>Hello</p>
}
```

ref 속성 사용할 때 주의할 점 2

- `v-if` 디렉티브와 사용하는 경우 화면에 해당 영역이 그려지기 전까진 DOM 요소 접근 불가

ref 속성 사용할 때 주의할 점 2

- `v-if` 디렉티브와 사용하는 경우 화면에 해당 영역이 그려지기 전까진 DOM 요소 접근 불가

```
<div v-if="isUser">
  <p ref="w3c">W3C</p>
</div>

new Vue({
  data: { isUser: false },
  mounted: function() {
    this.$refs.w3c; // undefined
  },
});
```

ref 속성 사용할 때 주의할 점 2

- `v-if` 디렉티브와 사용하는 경우 화면에 해당 영역이 그려지기 전까진 DOM 요소 접근 불가

```
<div v-if="isUser">
  <p ref="w3c">W3C</p>
</div>

new Vue({
  data: { isUser: false },
  mounted: function() {
    this.$refs.w3c; // undefined
  },
});
```

ref 속성 사용할 때 주의할 점 2

- `v-if` 디렉티브와 사용하는 경우 화면에 해당 영역이 그려지기 전까진 DOM 요소 접근 불가

```
<div v-if="isUser">
  <p ref="w3c">W3C</p>
</div>

new Vue({
  data: { isUser: true },
  mounted: function() {
    this.$refs.w3c; // <p>W3C</p>
  },
});
```

ref 속성 사용할 때 주의할 점 3

- 하위 컴포넌트의 내용을 접근할 순 있지만 남용하면 안된다

```
<div id="app">
  <TodoList ref="list"></TodoList>
</div>

new Vue({
  el: '#app',
  methods: { // 상위 컴포넌트에서 불필요하게 하위 컴포넌트를 제어하는 코드
    fetchItems: function() { this.$refs.list.fetchTodos(); }
  }
});
```

ref 속성 사용할 때 주의할 점 3

- 하위 컴포넌트의 내용을 접근할 순 있지만 남용하면 안된다

```
<div id="app">
  <TodoList></TodoList>
</div>
```

```
// 하위 컴포넌트의 라이프 사이클 훅을 이용
var TodoList = {
  methods: {
    fetchTodos: function() { .. }
  },
  created: function() { this.fetchTodos(); }
};
```

5. computed 속성

간결한 템플릿의 완성

computed 속성이란?

- **간결**하고 **직관적인** 템플릿 표현식을 위해 뷰에서 제공하는 속성

```
<p>{{ 'hello' + str + '!!' }}</p> <!-- 템플릿 표현식만 이용하는 경우 -->
<p>{{ greetingStr }}</p> <!-- computed 속성을 활용하는 경우 -->
```

```
new Vue({
  data: { str: 'world' },
  computed: {
    greetingStr: function() {
      return 'hello' + this.str + '!!';
    }
  }
});
```

computed 속성이란?

- **간결**하고 **직관적인** 템플릿 표현식을 위해 뷰에서 제공하는 속성

```
<p>{{ 'hello' + str + '!!' }}</p> <!-- 템플릿 표현식만 이용하는 경우 -->
<p>{{ greetingStr }}</p> <!-- computed 속성을 활용하는 경우 -->
```

```
new Vue({
  data: { str: 'world' },
  computed: {
    greetingStr: function() {
      return 'hello' + this.str + '!!';
    }
  }
});
```

computed 속성이란?

- **간결**하고 **직관적인** 템플릿 표현식을 위해 뷰에서 제공하는 속성

```
<p>{{ 'hello' + str + '!!' }}</p> <!-- 템플릿 표현식만 이용하는 경우 -->
<p>{{ greetingStr }}</p> <!-- computed 속성을 활용하는 경우 -->
```

```
new Vue({
  data: { str: 'world' },
  computed: {
    greetingStr: function() {
      return 'hello' + this.str + '!!';
    }
  }
});
```

computed 속성이란?

- **간결**하고 **직관적인** 템플릿 표현식을 위해 뷰에서 제공하는 속성

```
<p>{{ 'hello' + str + '!!' }}</p> <!-- 템플릿 표현식만 이용하는 경우 -->
<p>{{ greetingStr }}</p> <!-- computed 속성을 활용하는 경우 -->
```

```
new Vue({
  data: { str: 'world' },
  computed: {
    greetingStr: function() {
      return 'hello' + this.str + '!!';
    }
  }
})
```

computed 속성이란?

- **간결**하고 **직관적인** 템플릿 표현식을 위해 뷰에서 제공하는 속성

```
<p>{{ 'hello' + str + '!!' }}</p> <!-- 템플릿 표현식만 이용하는 경우 -->
<p>{{ greetingStr }}</p> <!-- computed 속성을 활용하는 경우 -->
```

```
new Vue({
  data: { str: 'world' },
  computed: {
    greetingStr: function() {
      return 'hello' + this.str + '!!';
    }
  }
})
```

computed 속성 활용처 1

- 조건에 따라 HTML 클래스를 추가, 변경할 때

```
<li v-bind:class="{ disabled: isLastPage }"></li>
```

```
computed: {
  isLastPage: function() {
    var lastPageCondition =
      this.paginationInfo.current_page >= this.paginationInfo.last_page;
    var nothingFetched = Object.keys(this.paginationInfo).length === 0;
    return lastPageCondition || nothingFetched;
  }
}
```

computed 속성 활용처 1

- 조건에 따라 HTML 클래스를 추가, 변경할 때

```
<li v-bind:class="listItemClass"></li>

computed: {
    listItemClass: function() {
      // ..
    }
}
```

computed 속성 활용처 2

- 스토어(Vuex)의 state 값을 접근할 때

```
<div>
  <p>{{ this.$store.state.module1.str }}</p>
  <p>{{ module1Str }}</p>
</div>

new Vue({
  computed: {
    module1Str: function() {
      return this.$store.state.module1.str;
    }
  }
});
```

computed 속성 활용처 2

- 스토어(Vuex)의 state 값을 접근할 때

```
<div>
  <p>{{ this.$store.state.module1.str }}</p>
  <p>{{ module1Str }}</p>
</div>

new Vue({
  computed: {
    module1Str: function() {
      return this.$store.state.module1.str;
    }
  }
});
```

computed 속성 활용처 2

- 스토어(Vuex)의 state 값을 접근할 때

```
<div>
  <p>{{ this.$store.state.module1.str }}</p>
  <p>{{ module1Str }}</p>
</div>

new Vue({
  computed: {
    module1Str: function() {
      return this.$store.state.module1.str;
    }
  }
});
```

computed 속성 활용처 2

- 스토어(Vuex)의 state 값을 접근할 때

```
<div>
  <p>{{ this.$store.state.module1.str }}</p>
  <p>{{ module1Str }}</p>
</div>

new Vue({
  computed: {
    module1Str: function() {
      return this.$store.state.module1.str;
    }
  }
});
```

computed 속성 활용처 2

- 스토어(Vuex)의 state 값을 접근할 때

```
<div>
  <p>{{ this.$store.state.module1.str }}</p>
  <p>{{ module1Str }}</p>
</div>

new Vue({
  computed: {
    module1Str: function() {
      return this.$store.state.module1.str;
    }
  }
});
```

computed 속성 활용처 3

- Vue i18n과 같은 다국어 라이브러리에도 활용 가능

```
<p>{{ 'userPage.common.filter.input.label' }}</p>
<p>{{ inputLabel }}</p>
```

```
computed: {
  inputLabel: function() {
    return $t('userPage.common.filter.input.label');
  }
}
```

computed 속성 활용처 3

- Vue i18n과 같은 다국어 라이브러리에도 활용 가능

```
<p>{{ 'userPage.common.filter.input.label' }}</p>
<p>{{ inputLabel }}</p>
```

```
computed: {
  inputLabel: function() {
    return $t('userPage.common.filter.input.label');
  }
}
```

computed 속성 활용처 3

- Vue i18n과 같은 다국어 라이브러리에도 활용 가능

```
<p>{{ 'userPage.common.filter.input.label' }}</p>
<p>{{ inputLabel }}</p>
```

```
computed: {
  inputLabel: function() {
    return $t('userPage.common.filter.input.label');
  }
}
```

computed 속성 활용처 3

- Vue i18n과 같은 다국어 라이브러리에도 활용 가능

```
<p>{{ 'userPage.common.filter.input.label' }}</p>
<p>{{ inputLabel }}</p>
```

```
computed: {
  inputLabel: function() {
    return $t('userPage.common.filter.input.label');
  }
}
```

감사합니다.

뷰를 더 알고 싶으신 분들은 인프런과 패스트캠퍼스에서 캡틴판교를 검색해주세요.

jangkeehyo@gmail.com