

# Vue.js

---

Programming Language

# **Index**

---

## **1** Template Syntax

- **text interpolation**

## **2** Attribute Bindings

- **v-bind**

## **3** Style & Class Bindings

- **v-bind**

## **4** Form Input Bindings

- **v-model**

## **5** Conditional Rendering

- **v-if, v-else-if, v-else**

## **6** List Rendering

- **v-for**



# Template Syntax



# Template Syntax

Directive API: <https://ko.vuejs.org/api/built-in-directives.html>

## data binding

- ▶ Vue instance의 데이터를 렌더링된 DOM에 바인딩할 수 있는 HTML 기반 템플릿 문법을 사용 ::> Vue는 이 템플릿 문법을 최적화된 JavaScript 코드로 컴파일 함
- ▶ Vue는 양방향 데이터 바인딩 (Two-Way data binding)을 지원
- ▶ 모델 (Model)에서 데이터를 정의한 후 뷰 (View)와 연결하면 모델과 뷰 중 한쪽의 데이터가 변경되면 다른 한쪽의 데이터도 자동으로 변경된 내용이 반영
- ▶ v- 접두사가 있는 특수한 속성인 **directive** 사용

- ▶ 서버에서 받아온 데이터를 뷰에 바인딩하는 경우

- 데이터가 html tag 안에 **텍스트**로 바인딩
- 데이터가 html tag의 **속성(attribute)**로 바인딩
- 데이터가 html의 **Form element**의 **value**에 바인딩
- 다중 데이터가 html의 **다중 element**를 생성하기 위해서 바인딩



<https://ko.vuejs.org/guide/essentials/template-syntax.html>

## 문자열 보간법 (Text Interpolation)

- ▶ 데이터 바인딩의 가장 기본적인 형태
- ▶ “Mustache”(이중 중괄호)를 사용한 텍스트 보간법

```
<h2>메세지 : {{ message }}</h2>
```

- ▶ 이중 중괄호({{ 속성명 }}) 태그 내 message는 해당 컴포넌트 인스턴스가 가진 data의 값으로 대체
- ▶ instance가 가진 data의 message가 변경될 때마다 자동으로 업데이트
- ▶ **v-once** directive : data 변경 시 업데이트 되지 않는 일회성 보간 수행

```
<h2 v-once>메세지 변경? : {{ message }}</h2>
```

## ☞ 문자열 보간법 (Text Interpolation) 실습

```
<div id="app">
  <h2>{{ message }}</h2>
  <h2 v-once>{{ message }}</h2>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const message = ref("Hello Vue!!!");
      return {
        message,
      };
    },
  });
  app.mount("#app");
</script>
```



03.text-interpolation.html

The screenshot shows a browser window titled "Text Interpolation". It displays two identical "Hello Vue!!!". The first message is rendered normally, while the second is rendered with a red underline, indicating it is a once interpolation.

Vue DevTools sidebar:

- Elements: Shows two "Hello Vue!!!"
- Sources: Shows the component tree with "message : Hello Vue!!!(Ref)"
- Network: Shows the request to the server
- Vue: Shows the component tree with "message : Hello Vue!!!(Ref)"

The screenshot shows a browser window titled "Text Interpolation". It displays two messages: "Hello Hissam!!!" and "Hello Vue!!!". The "Hello Hissam!!!" message is underlined in red, while the "Hello Vue!!!".

Vue DevTools sidebar:

- Elements: Shows "Hello Hissam!!!" and "Hello Vue!!!".
- Sources: Shows the component tree with "message : Hello Hissam!!!(Ref)"
- Network: Shows the request to the server
- Vue: Shows the component tree with "message : Hello Hissam!!!(Ref)"

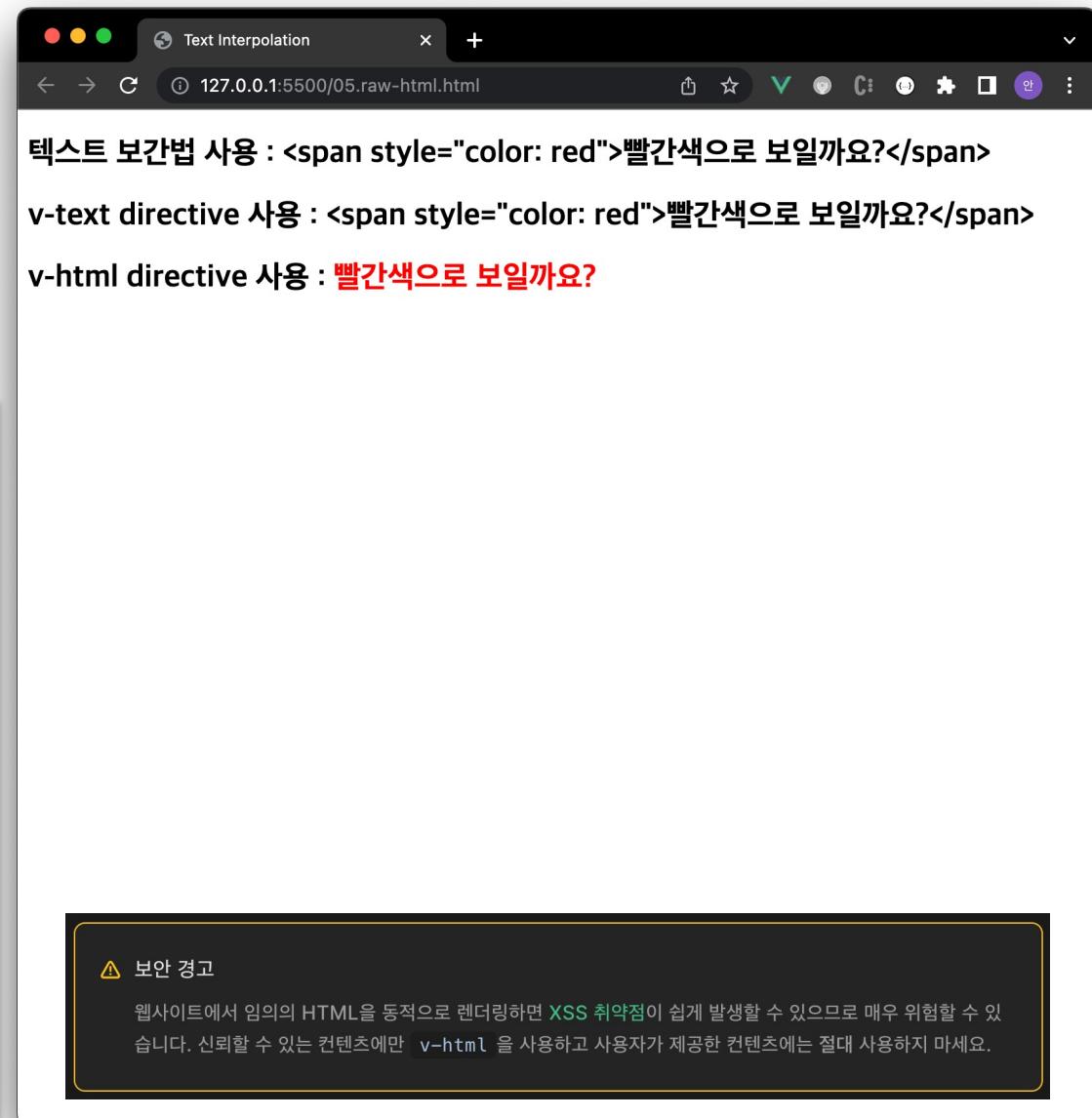
Bottom right corner: 변경 (Change)

## 원시 (Raw) HTML

- ▶ {{ 속성명 }} 는 속성명을 일반 텍스트로 해석 (= v-text)
- ▶ v-html directive는 html로 해석

```
<div id="app">
  <h2>텍스트 보간법 사용 : {{ message }}</h2>
  <h2>v-text directive 사용 : <span v-text="message"></span></h2>
  <h2>v-html directive 사용 : <span v-html="message"></span></h2>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const message = ref('<span style="color: red">빨간색으로 보일까요?</span>');
      return {
        message,
      };
    },
    app.mount("#app");
  });
</script>
```



## JavaScript 표현식 사용

- ▶ 모든 데이터 바인딩 내에서 JavaScript 표현식의 모든 기능을 지원

```
    {{ num + 1 }}  
  
    {{ flag ? '예' : '아니오' }}  
  
    {{ message.split('').reverse().join('') }}  
  
<div :id="`list-${id}`"></div>
```

- ▶ 위 표현식은 현재 component instance의 데이터 범위에서 JavaScript로 평가됨
- ▶ JavaScript 표현식을 사용할 수 있는 위치
  - **이중 중괄호** (text interpolation) 내부
  - **모든 Vue directive 속성** ('v-'로 시작되는 특수 속성) 내부

# Text Interpolation

## JavaScript 표현식 사용

- ▶ 각 바인딩에는 하나의 단일 표현식(값)만 포함 가능.
- ▶ 선언식, 제어문등은 불가능

```
<!-- 선언식은 불가능 -->
{{ var x = 10 }}

<!-- 흐름 제어 불가능. 삼항 연산자 사용 -->
{{ if(flag) { return message } }}
```

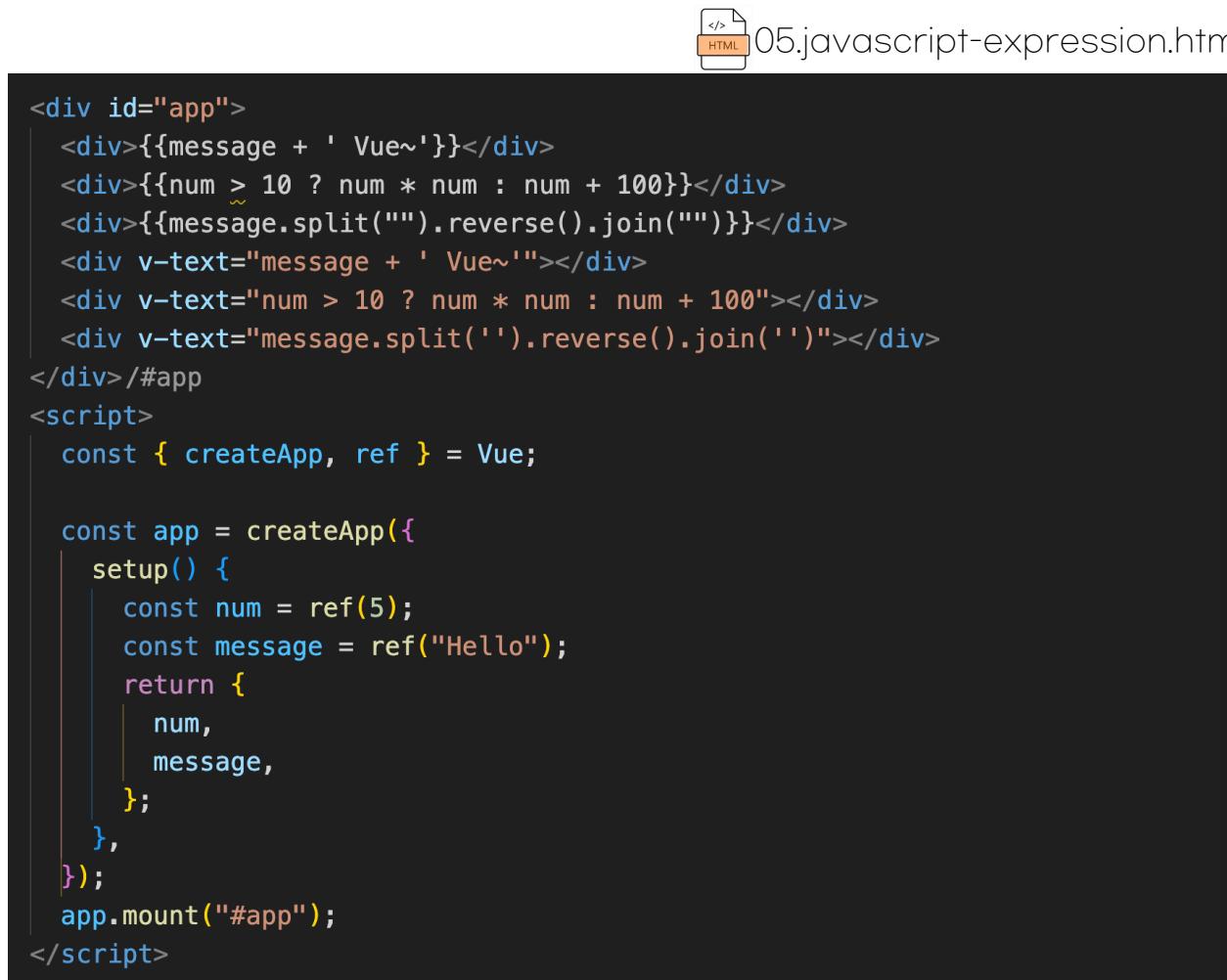
- ▶ 바인딩 표현식 내부에서 method 호출 가능

```
<span :title="currentDate(date)">
  {{ formatDate(date) }}
</span>
```

### TIP

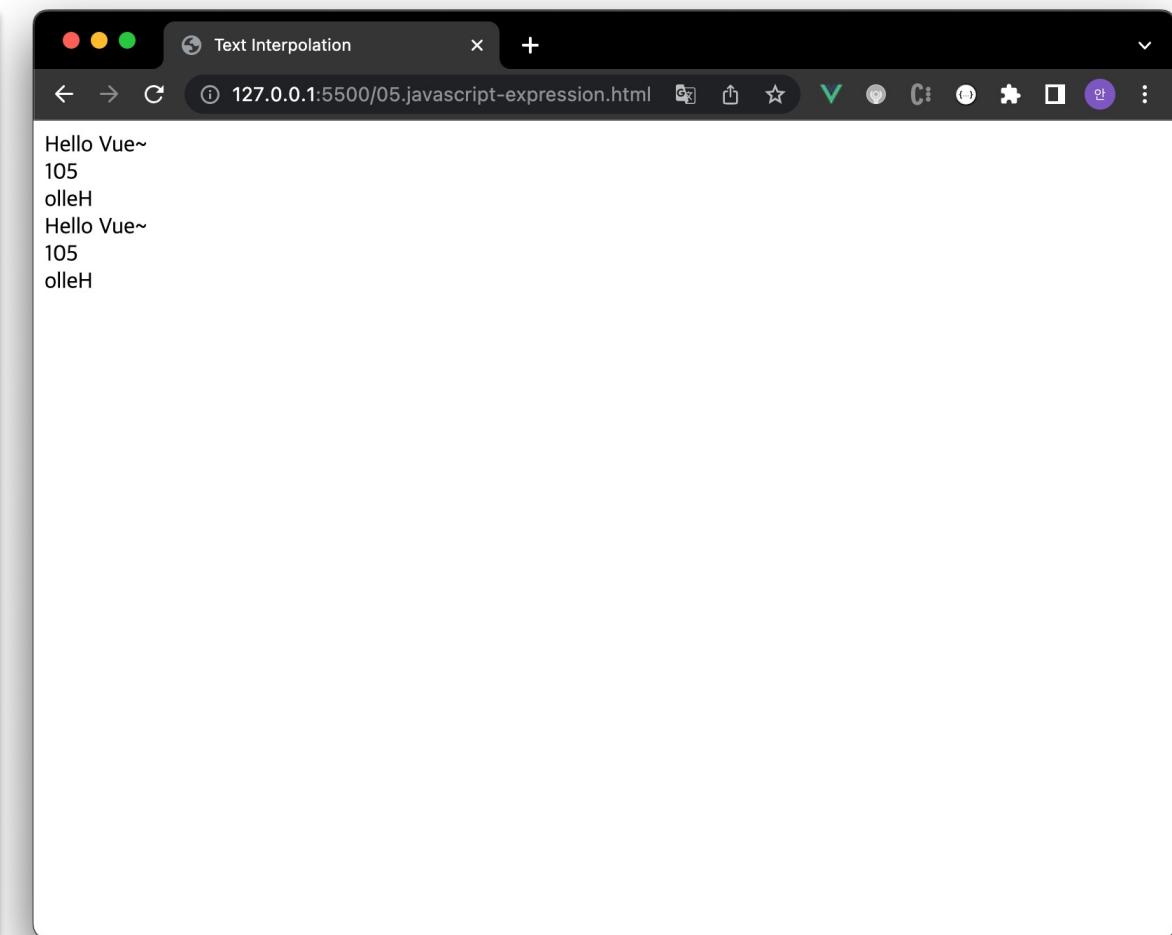
바인딩 표현식 내부에서 호출되는 함수는 컴포넌트가 업데이트될 때마다 호출되므로, 데이터를 변경 또는 비동기 작업을 트리거하는 등의 부작용이 없어야 합니다.

## JavaScript 표현식 사용 실습



```
<div id="app">
  <div>{{message + ' Vue~'}}</div>
  <div>{{num > 10 ? num * num : num + 100}}</div>
  <div>{{message.split("").reverse().join("")}}</div>
  <div v-text="message + ' Vue~'"></div>
  <div v-text="num > 10 ? num * num : num + 100"></div>
  <div v-text="message.split('').reverse().join(''))"></div>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const num = ref(5);
      const message = ref("Hello");
      return {
        num,
        message,
      };
    },
  });
  app.mount("#app");
</script>
```



# Directive (Attribute Bindings: v-bind)



# Directive: Attribute (속성) Bindings

<https://ko.vuejs.org/guide/essentials/template-syntax.html#attribute-bindings>

## 🔗 v-bind directive

- ▶ {{ 속성명 }}는 HTML attribute 내에서 사용 불가능
- ▶ v-bind directive 사용

```
<input type="text" v-bind:id="inputId" />
```

- ▶ 단축 문법 : v-bind는 ':'으로 단축하여 사용 가능

```
<input type="button" :id="btnId" :value="btnVal" />
```

- ▶ Boolean 속성

```
<input type="button" value="취소" :disabled="isBtnDisabled" />
```

# Directive: Attribute (속성) Bindings

## 💡 v-bind 동적 인자

- ▶ directive의 인자를 대괄호([])로 감싸서 JavaScript 표현식으로 사용할 수 있음

```
<input type="button" :[key]="btnId" :value="btnVal" />
```

- ▶ 여러 속성 동적 바인딩

```
<div v-bind="divAttrs">메세지 : {{ message }}</div>
```



- 참고 : v-bind
  - v-bind directive는 단방향으로만 데이터 바인딩을 수행
  - Vue instance의 data나 속성이 변경되면 UI를 갱신
  - 반대로 UI상의 바인딩된 element의 값이 변경되어도 data는 변경되지 않음
  - “단방향 데이터 바인딩” : data ::> 화면 UI로만 단방향으로 binding

## 59 v-bind 실습



06.attribute-bindingexpression.html

```
<div id="app">
  <h2>Attribute Binding</h2>
  <input type="text" v-bind:id="inputId" :value="message" />
  <input type="button" :key="btnId" :value="btnVal" />
  <input type="button" value="취소" :disabled="isBtnDisabled" />
  <div v-bind="divAttrs">메세지 : {{ message }}</div>
  <div><a :href="link">Google</a></div>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const key = ref("id");
      const inputId = ref("message-input");
      const btnId = ref("register-btn");
      const btnVal = ref("확인");
      const message = ref("Hello Hissam!!!");
      const isBtnDisabled = ref(true);
      const divAttrs = ref({ id: "divId", class: "divClass" });
      const link = ref("http://www.google.com");
      return {
        key, inputId, btnId, btnVal, message, isBtnDisabled, divAttrs, link,
      };
    },
  });
  app.mount("#app");
</script>
```

Attribute Binding

Hello Hissam!!!

메세지 . Hello Hissam!!!

Google

input의 값을 바꿀 때와  
message를 바꿀 때 어떻게 될까?

Elements Console Sources Vue

Find components... <Root> fragment

Filter State... setup key : id(Ref) inputId : message-input(Ref) btnId : register-btn(Ref) btnVal : 확인(Ref) message : Hello Hissam!!!(Ref) isBtnDisabled : true(Ref) divAttrs : Object(Ref) link : http://www.google.com(Ref)

# Directive **(Style & Class Bindings: v-bind)**

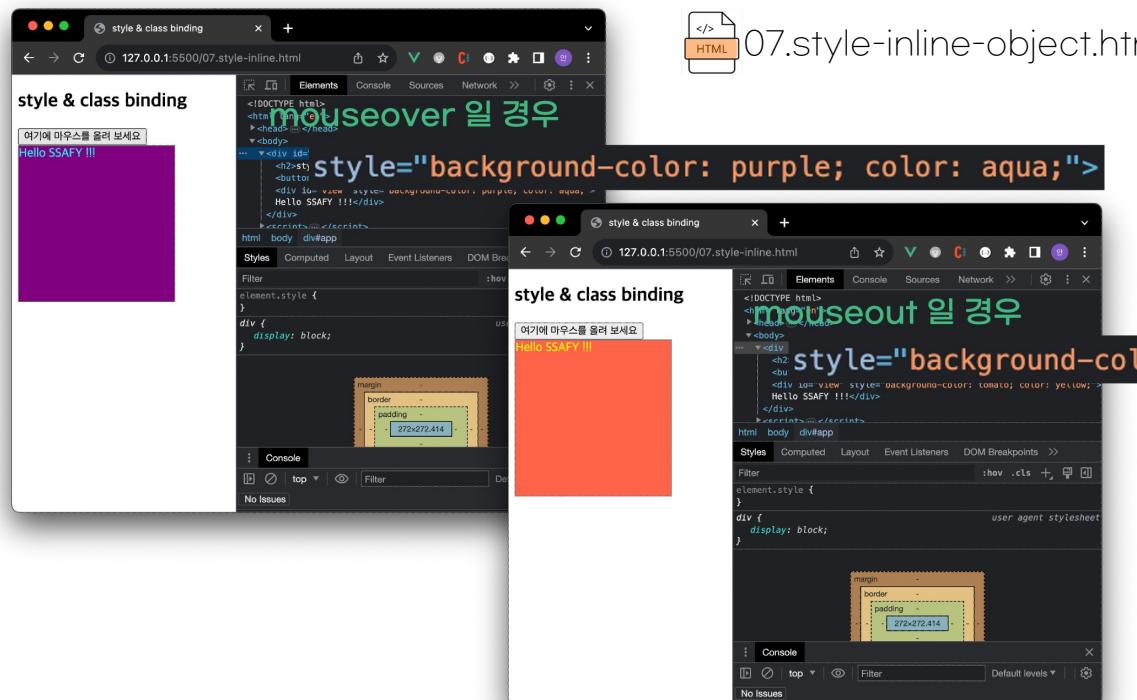


# Directive: style & class Bindings

<https://ko.vuejs.org/guide/essentials/class-and-style.html>

## inline style - 객체 바인딩

- ▶ 형식 `v-bind:style="data 속성 js 객체"`
- ▶ style 속성은 data 속성에 자바스크립트 객체 형태로 설정
- ▶ data 속성 설정 시 스타일 속성은 케밥 표기법 (kebab-cased, font-size)이 아닌 “**카멜 표기법 (camelCase, fontSize)**”으로 사용해야 함



```
const app = createApp({
  setup() {
    const styleVal = ref({ backgroundColor: "pink", color: "gray" });
    const btnOver = () => {
      styleVal.value.backgroundColor = "purple";
      styleVal.value.color = "aqua";
    };
    const btnOut = () => {
      styleVal.value.backgroundColor = "tomato";
      styleVal.value.color = "yellow";
    };
    return {
      styleVal,
      btnOver,
      btnOut,
    };
  },
  mounted() {
    app.mount("#app");
  }
});
```

## inline style - 배열 바인딩

▶ 형식 v-bind:style=" [ data 속성 js 객체, data 속성 js 객체, .. ] "

```
<div id="app">
  <h2>style & class binding</h2>
  <button :style="[colorStyle, layoutStyle, fontStyle]">3개의 style이 적용 된 버튼</button>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const colorStyle = ref({ backgroundColor: "pink", color: "gray" });
      const layoutStyle = ref({ width: "200px", height: "100px", textAlign: "center" });
      const fontStyle = ref({ fontSize: "20px", fontWeight: "bold" });

      return {
        colorStyle,
        layoutStyle,
        fontStyle,
      };
    },
  });
  app.mount("#app");
</script>
```



08.style-inline-array.html

The screenshot shows a browser window titled "style & class binding" displaying the content of "08.style-inline-array.html". The page contains a heading "style & class binding" and a button with the text "3개의 style이 적용 된 버튼". The button's styling is applied via inline JavaScript using the v-bind:style directive. The browser's developer tools are open, showing the element node in the DOM tree under the body. The styles tab in the developer tools reveals the three styles being applied: background-color: pink, color: gray, and font-weight: bold. A detailed zoomed-in view of the button's bounding box in the developer tools highlights its dimensions: width 200px, height 100px, and a bounding box of 318x148.414.

# Directive: style & class Bindings

## HTML class binding

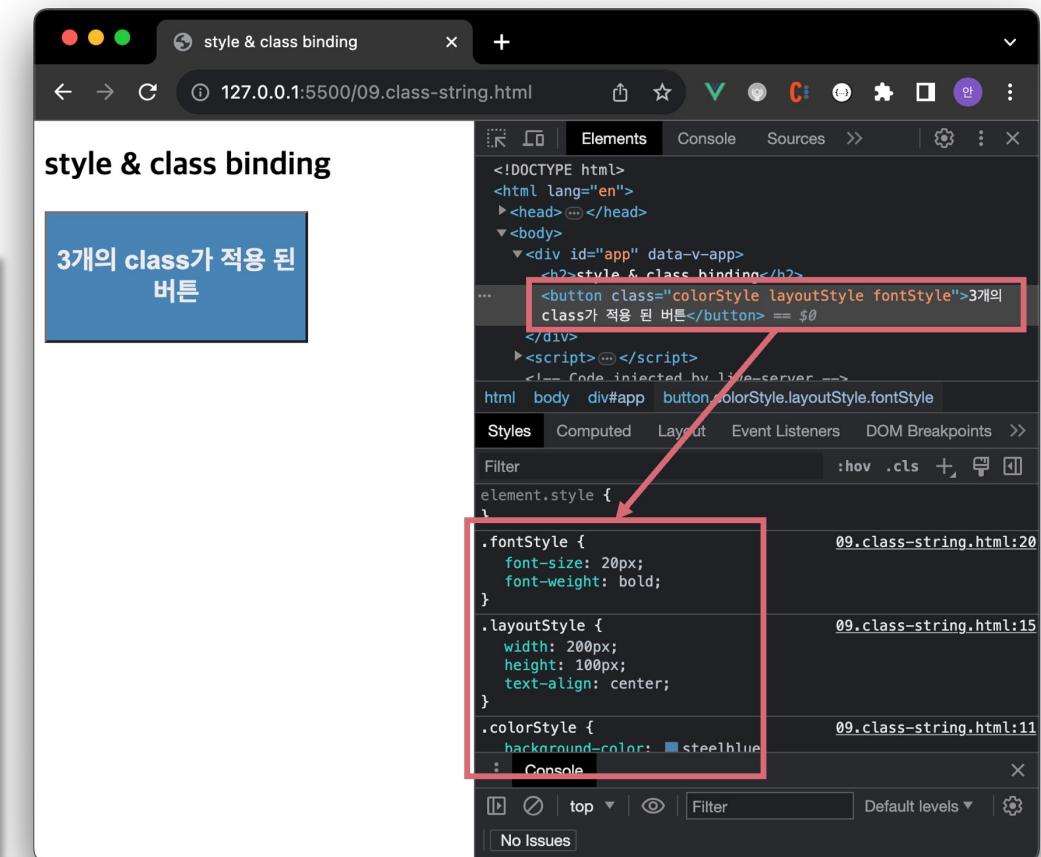
- ▶ 형식 `v-bind:class="..."`
- ▶ css class 문자열을 바인딩하는 방법과 true/false 값을 가진 객체를 바인딩하는 방법이 있음

```
<div id="app">
  <h2>style & class binding</h2>
  <button class="colorStyle" :class="myBtn">3개의 class가 적용 된 버튼</button>
</div> #app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const myBtn = ref("layoutStyle fontStyle");
      return {
        myBtn,
      };
    },
  });
  app.mount("#app");
</script>
```

09.class-string.html

```
<style>
  .colorStyle {
    background-color: steelblue;
    color: rgb(237, 236, 236);
  }
  .layoutStyle {
    width: 200px;
    height: 100px;
    text-align: center;
  }
  .fontStyle {
    font-size: 20px;
    font-weight: bold;
  }
</style>
```



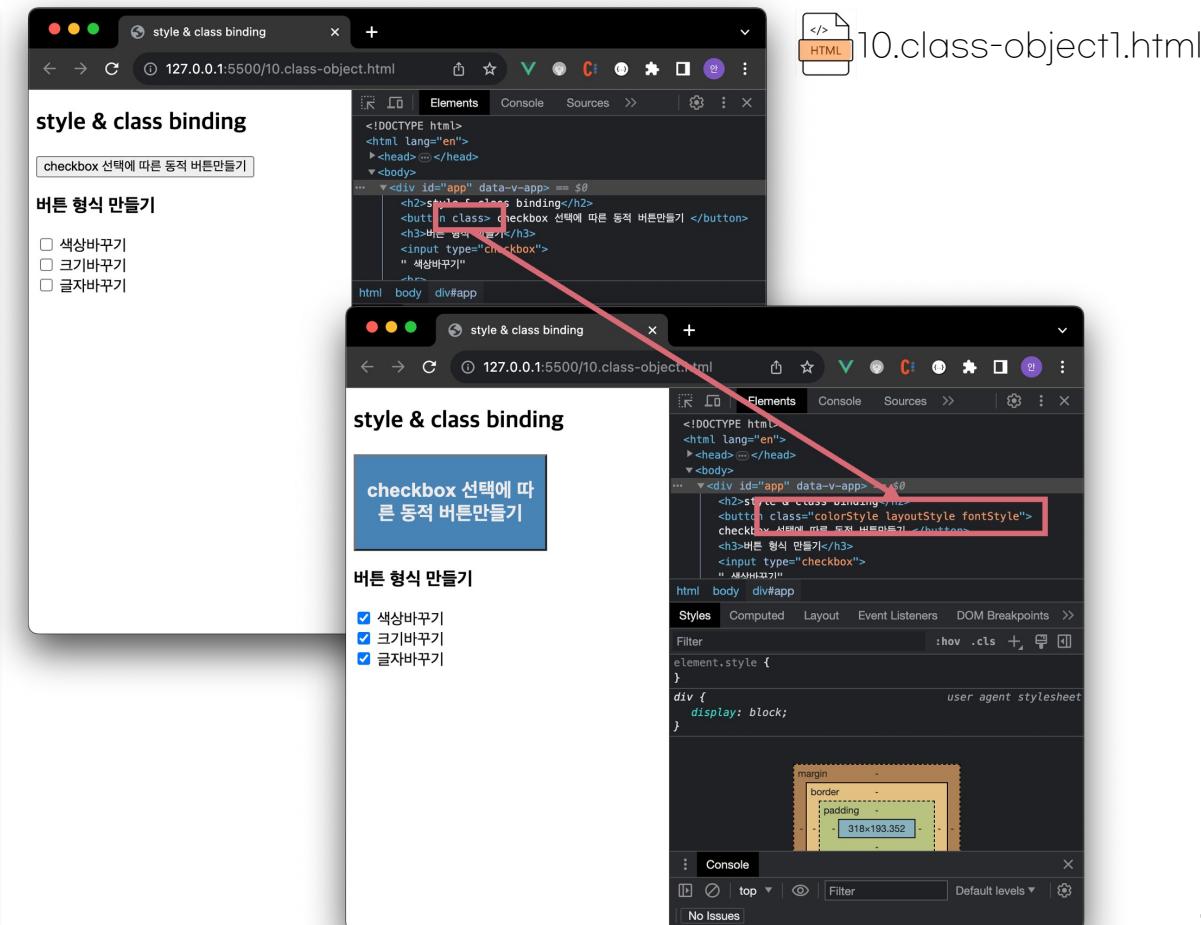
## HTML class binding

- ▶ 형식 `v-bind:class="..."`
- ▶ css class 문자열을 바인딩하는 방법과 `true/false` 값을 가진 객체를 바인딩하는 방법이 있음

```
<div id="app">
  <h2>style & class binding</h2>
  <button :class="{colorStyle: isColor, layoutStyle: isLayout, fontStyle: isFont}">
    checkbox 선택에 따른 동적 버튼만들기
  </button>
  <h3>버튼 형식 만들기</h3>
  <input type="checkbox" v-model="isColor" /> 색상바꾸기<br />
  <input type="checkbox" v-model="isLayout" /> 크기바꾸기<br />
  <input type="checkbox" v-model="isFont" /> 글자바꾸기
</div>#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const isColor = ref(false);
      const isLayout = ref(false);
      const isFont = ref(false);
      return {
        isColor,
        isLayout,
        isFont,
      };
    },
  });
  app.mount("#app");
</script>
```

true, false에 따른 class 적용여부 확인 : 적용 할 class가 많아지면 복잡해짐 > 11.html 참고



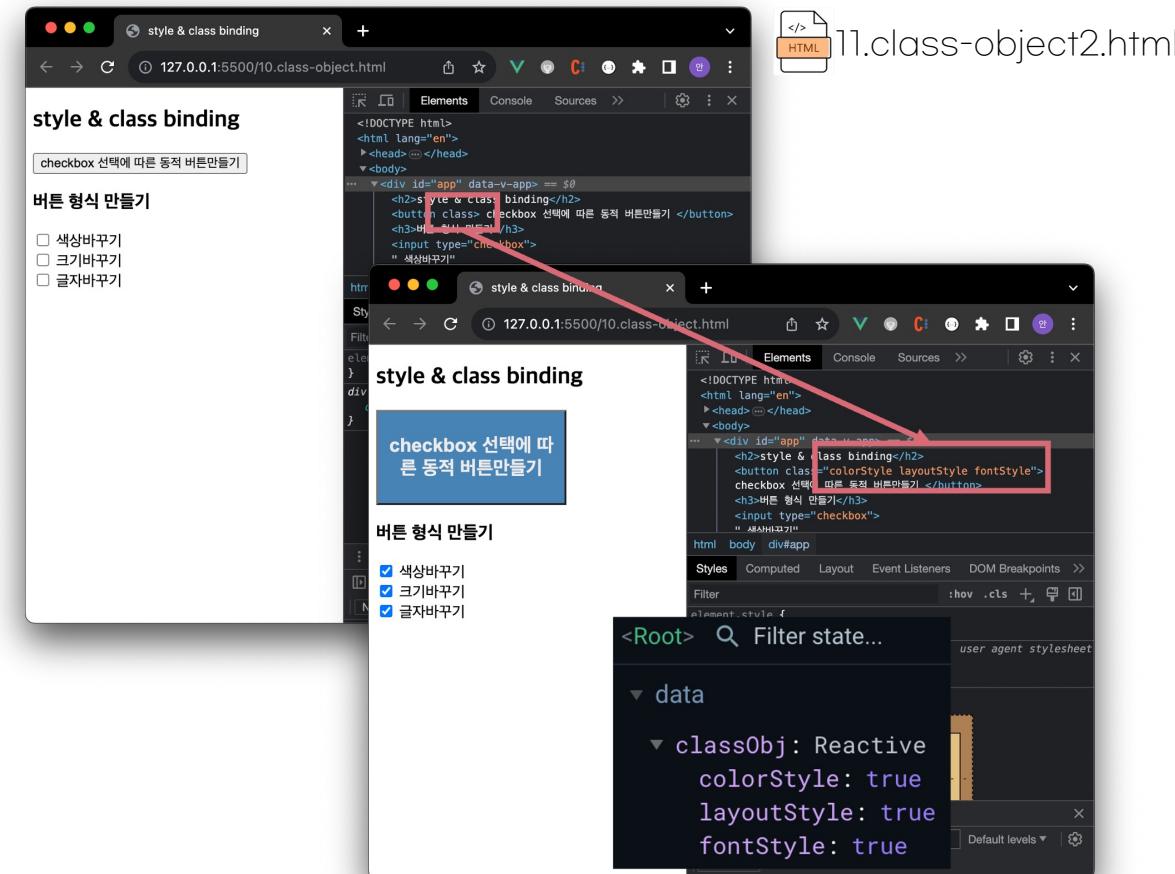
# Directive: style & class Bindings

## HTML class binding

- ▶ CSS class 문자열을 바인딩하는 방법과 true/false 값을 가진 객체를 바인딩하는 방법이 있음
- ▶ 적용할 class가 많아질 경우 inline 형식으로 지정하면 복잡해지는 문제 발생 => true/false 값을 가진 객체로 해결

```
<div id="app">
  <h2>style & class binding</h2>
  <button :class="classObj">checkbox 선택에 따른 동적 버튼만들기</button>
  <h3>버튼 형식 만들기</h3>
  <input type="checkbox" v-model="classObj.colorStyle" /> 색상바꾸기<br />
  <input type="checkbox" v-model="classObj.layoutStyle" /> 크기바꾸기<br />
  <input type="checkbox" v-model="classObj.fontStyle" /> 글자바꾸기
</div>#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const classObj = ref({
        colorStyle: false,
        layoutStyle: false,
        fontStyle: false,
      });
      return {
        classObj,
      };
    },
  });
  app.mount("#app");
</script>
```



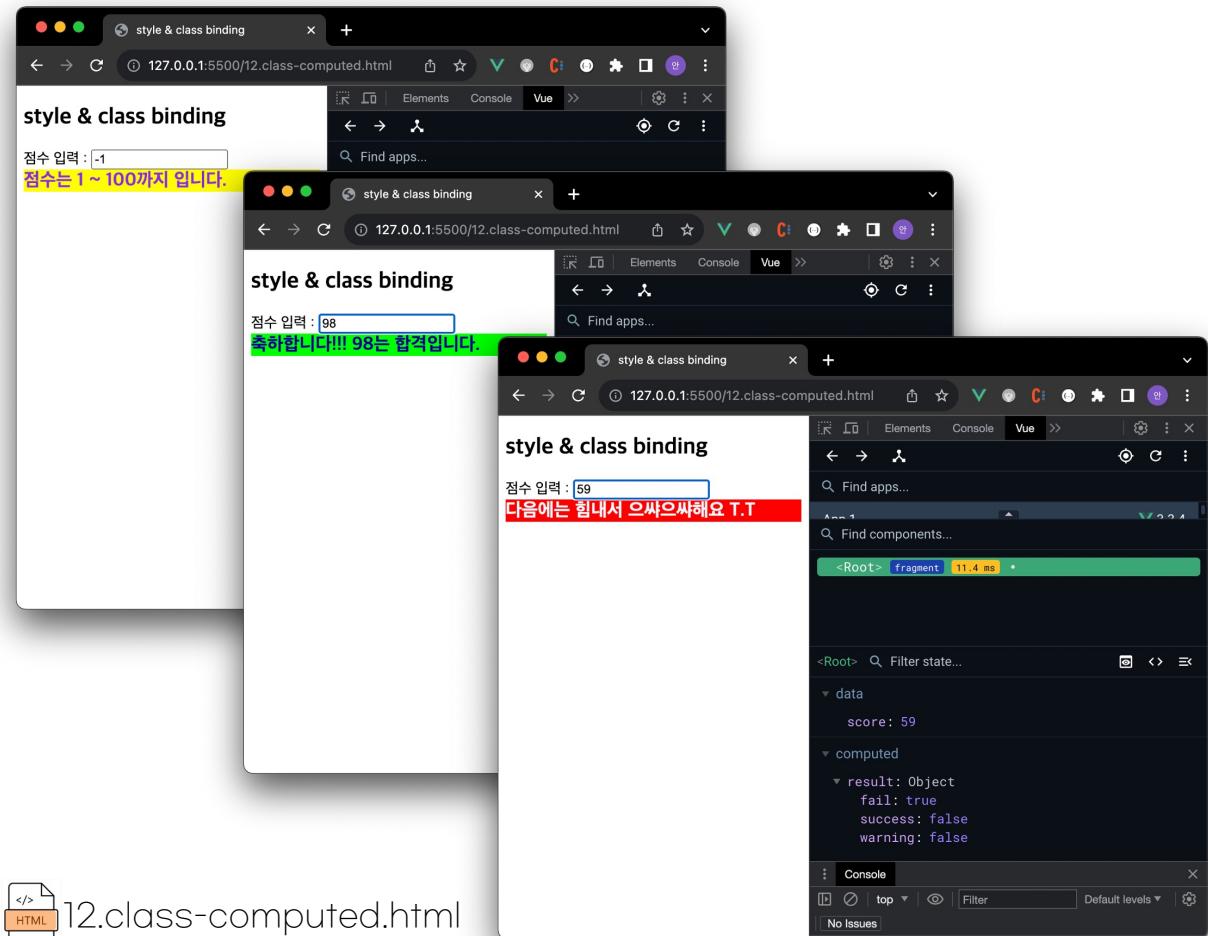
# Directive: style & class Bindings

## 59 동적 스타일 binding

- ▶ computed() method가 리턴 하는 값을 스타일에 적용 가능
- ▶ 리턴값은 css의 **클래스명**이거나 { **클래스명** : true/false } 형태의 객체 타입이어야 함

```
<div id="app">
  <h2>style & class binding</h2>
  <label for="score">점수 입력 : </label>
  <input type="text" id="score" v-model.number="score" />
  <div class="view" :class="result" v-if="result.warning">점수는 1 ~ 100까지입니다.</div>
  <div class="view" :class="result" v-if="result.success">
    축하합니다!!! {{ score }}는 합격입니다.
  </div>/.view
  <div class="view" :class="result" v-if="result.fail">다음에는 힘내서 으쌰으쌰해요 T.T</div>
</div>/#app
<script>
  const { createApp, ref, computed, watch } = Vue;

  const app = createApp({
    setup() {
      const score = ref(0);
      const result = computed(() => {
        return {
          warning: score.value < 0 || score.value > 100,
          success: score.value <= 100 && score.value >= 60,
          fail: score.value > 0 && score.value < 60,
        };
      });
      return {
        score, result,
      };
    },
  });
  app.mount("#app");
</script>
```



# Directive (Form Input Bindings: v-model)



# Directive: Form Input Bindings

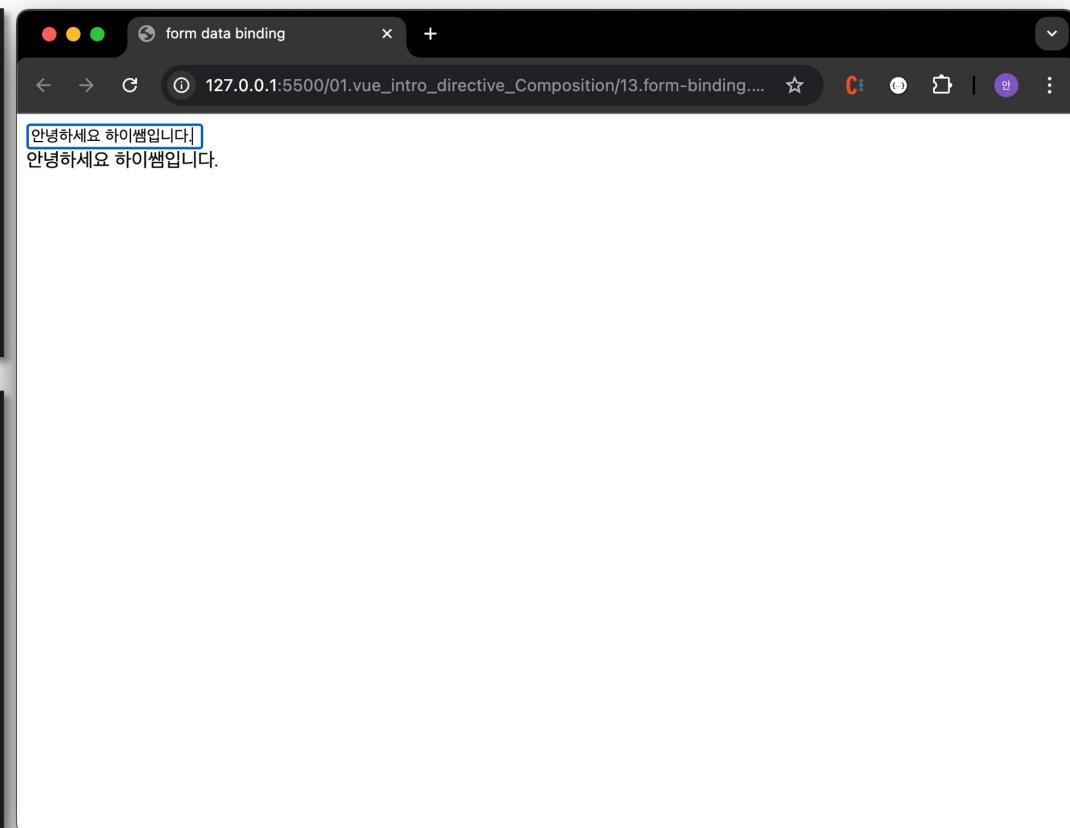
<https://ko.vuejs.org/guide/essentials/forms.html>

## form input binding: v-model

- ▶ {{ text interpolation }}, v-text, v-html, v-bind directive는 단방향 데이터 바인딩 지원
- ▶ v-model directive를 사용하여 “양방향 데이터 바인딩” 지원

```
<div id="app">
  <input type="text" id="message" />
  <div id="views"></div>
</div>#app
<script>
  document.querySelector("#message").addEventListener("input", (e) => {
    let msg = e.target.value;
    document.querySelector("#views").innerHTML = msg;
  });
</script>
```

VanillaJS



```
<div id="app">
  <input type="text" v-model="message" />
  <div>{{ message }}</div>
</div>#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const message = ref("");
      return {
        message,
      };
    },
  });
  app.mount("#app");
</script>
```

Vue.js

# Directive: Form Input Bindings

## ☞ form input binding: v-model

- ▶ v-model directive를 사용하여 양뱡향 데이터 바인딩
- ▶ text와 textarea의 경우 value 속성과 input 이벤트를 사용
- ▶ checkbox, radio의 경우 checked 속성과 change 이벤트를 사용
- ▶ select는 value를 속성으로 change를 이벤트로 사용

ⓘ 참고

v-model 은 모든 폼 엘리먼트에서 감지되는 초기 `value` , `checked` 또는 `selected` 속성 값을 무시합니다. 항상 현재 바인딩된 JavaScript 상태를 유효한 값으로 취급합니다. [reactivity API](#)를 사용하여 JavaScript에서 초기 값을 선언해야 합니다.

# Directive: Form Input Bindings

## ☞ form input binding: v-model

- ▶ form - text, textarea
- ▶ 문자열

```
<input type="text" v-model="message" />
<div>{{ message }}</div>
```

- ▶ 여러 줄을 가진 문장
- ▶ 텍스트 영역의 보간은 불가능 : <textarea>{{ message }}</textarea> ::> v-model 사용

```
<span>여러 줄 메세지:</span>
<p style="white-space: pre-line">{{ message }}</p>
<textarea v-model="message" placeholder="여러 줄을 추가해보세요"></textarea>
```

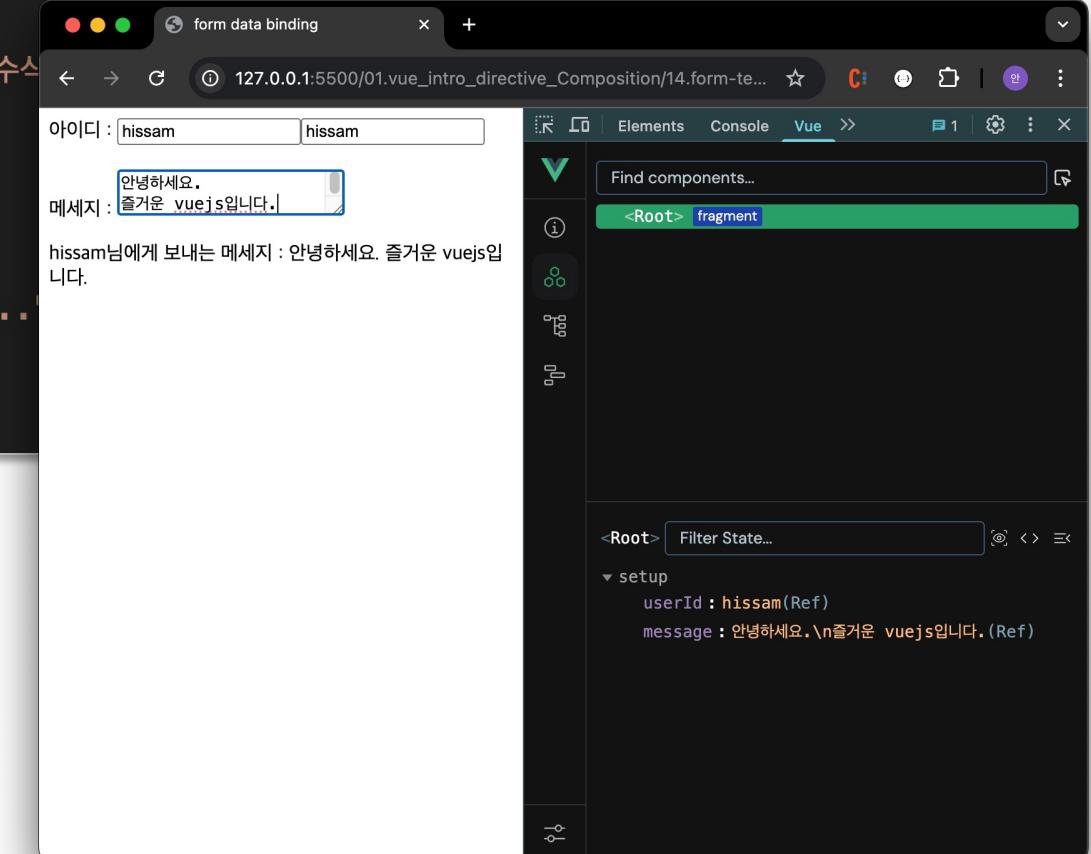
# Directive: Form Input Bindings

## form input binding: v-model

- ▶ form - text, textarea 실습

```
<div>
  <label for="userId">아이디 : </label>
  <input type="text" id="userId" v-model="userId" placeholder="아이디 입력..." />
  <!-- .lazy 수식어를 추가하여 change 이벤트 이후에 동기화 -->
  <input type="text" v-model.lazy="userId" placeholder="lazy 수식어">
</div>
<br />
<div>
  <label for="msg">메세지 : </label>
  <textarea id="msg" v-model="message" placeholder="메세지 입력..."></textarea>
</div>
<p>{{ userId }}님에게 보내는 메세지 : {{ message }}</p>
```

14.form-text-binding.html



# Directive: Form Input Bindings

## form input binding: v-model

- ▶ form - checkbox
- ▶ 하나의 체크박스는 단일 boolean 값을 갖음

```
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

### ▶ 실습

```
<div id="app">
  이메일 수신 :
  <input type="checkbox" id="email" v-model="emailTF" />
  <label for="email">{{ emailTF }}</label><br />
  SMS 수신 :
  <input type="checkbox" id="sms" v-model="smsYN"
    true-value="Y" false-value="N" />
  <label for="sms">{{ smsYN }}</label>
</div>
```

```
const emailTF = ref(false);
const smsYN = ref("Y");
```



15.form-checkbox-binding.html

The screenshot shows a browser window titled "form data binding" displaying a simple form with two checkboxes. The first checkbox is labeled "이메일 수신" and has a checked state of "false". The second checkbox is labeled "SMS 수신" and has a checked state of "Y". To the right of the browser is the Vue DevTools extension for Chrome, which shows the component tree and the state of the reactive variables:

- Root**:
  - emailTF : false(Ref)
  - smsYN : Y(Ref)

# Directive: Form Input Bindings

## ☞ form input binding: v-model

- ▶ form - checkbox
- ▶ 여러 개의 checkbox는 같은 배열(set)을 바인딩 할 수 있음
- ▶ 배열의 값과 checkbox의 value 속성이 같을 경우 체크 처리됨
- ▶ 배열은 체크된 순서대로 값을 저장

```
<div id="app">
  <h3>가고싶은 매장을 순서대로 선택해 주세요</h3>
  <input type="checkbox" id="hn" value="한남" v-model="checkedAreas" />
  <label for="hn">한남</label>
  <input type="checkbox" id="yn" value="연남" v-model="checkedAreas" />
  <label for="yn">연남</label>
  <input type="checkbox" id="hud" value="해운대" v-model="checkedAreas" />
  <label for="hud">해운대</label>
  <input type="checkbox" id="md" value="무등" v-model="checkedAreas" />
  <label for="md">무등</label>
  <input type="checkbox" id="ho" value="한옥마을" v-model="checkedAreas" />
  <label for="ho">한옥마을</label>
  <hr />
  희망 매장 순서 : {{ checkedAreas.join(', ') }}
</div>/#app
```

```
const checkedAreas = ref([]);
```



16.form-checkbox-multi-binding.html

Index	Value
0	연남
1	한옥마을
2	무등

# Directive: Form Input Bindings

## ☞ form input binding: v-model

- ▶ form-radio
- ▶ 선택된 항목의 value 속성 값을 관리

```
<div>선택한 것: {{ picked }}</div>

<input type="radio" id="one" value="하나" v-model="picked" />
<label for="one">하나</label>

<input type="radio" id="two" value="둘" v-model="picked" />
<label for="two">둘</label>
```

# Directive: Form Input Bindings

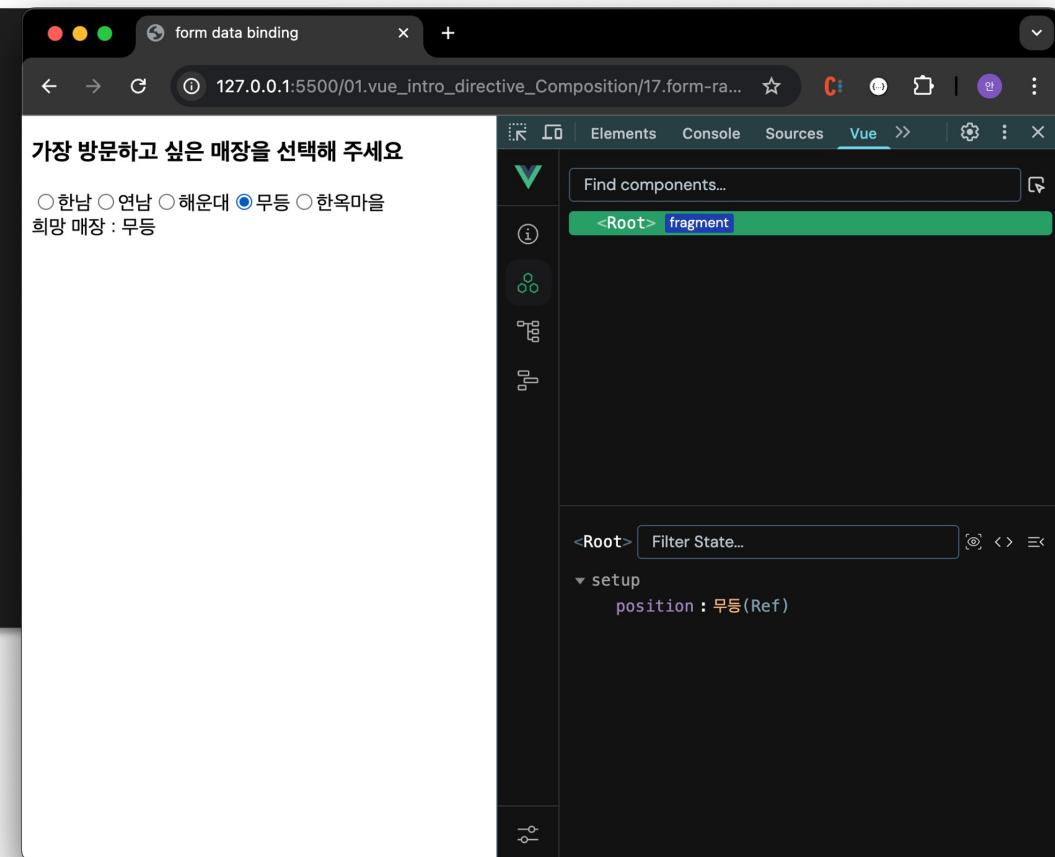
## 5 form input binding: v-model

- ▶ form - radio
- ▶ 초기 화면에서 연남이 체크되어 있음, 라이오 버튼 선택 시 변경 => 양방향

```
<div id="app">
  <h3>가장 방문하고 싶은 매장을 선택해 주세요</h3>
  <input type="radio" id="hn" value="한남" v-model="position" />
  <label for="hn">한남</label>
  <input type="radio" id="yn" value="연남" v-model="position" />
  <label for="yn">연남</label>
  <input type="radio" id="hud" value="해운대" v-model="position" />
  <label for="hud">해운대</label>
  <input type="radio" id="md" value="무등" v-model="position" />
  <label for="md">무등</label>
  <input type="radio" id="ho" value="한옥마을" v-model="position" />
  <label for="ho">한옥마을</label>
  <div>희망 매장 : {{ position }}</div>
</div>/#app
```

```
const position = ref("연남");
```

 17.form-radio-binding.html



# Directive: Form Input Bindings

## 🔗 form input binding: v-model

- ▶ form - select box
- ▶ select box일 경우 선택된 option의 value 속성의 값을 관리
- ▶ v-model 표현식의 초기 값과 일치하는 option의 value가 없으면 빈 상태로 렌더링 됨

```
<div>선택됨: {{ selected }}</div>

<select v-model="selected">
  <option disabled value="">다음 중 하나를 선택하세요</option>
  <option>가</option>
  <option>나</option>
  <option>다</option>
</select>
```

```
data() {
  return {
    selected: '',
  };
},
```

### ⓘ 참고

v-model 표현식의 초기 값이 옵션과 일치하지 않으면 <select> 엘리먼트가 "선택되지 않은" 상태로 렌더링됩니다. iOS에서는 이 경우 변경 이벤트를 발생시키지 않기 때문에 사용자가 첫 번째 항목을 선택할 수 없게 됩니다. 따라서 위의 예에서 설명한 것처럼 비활성화된 옵션에 빈 값을 제공하는 것이 좋습니다.

# Directive: Form Input Bindings

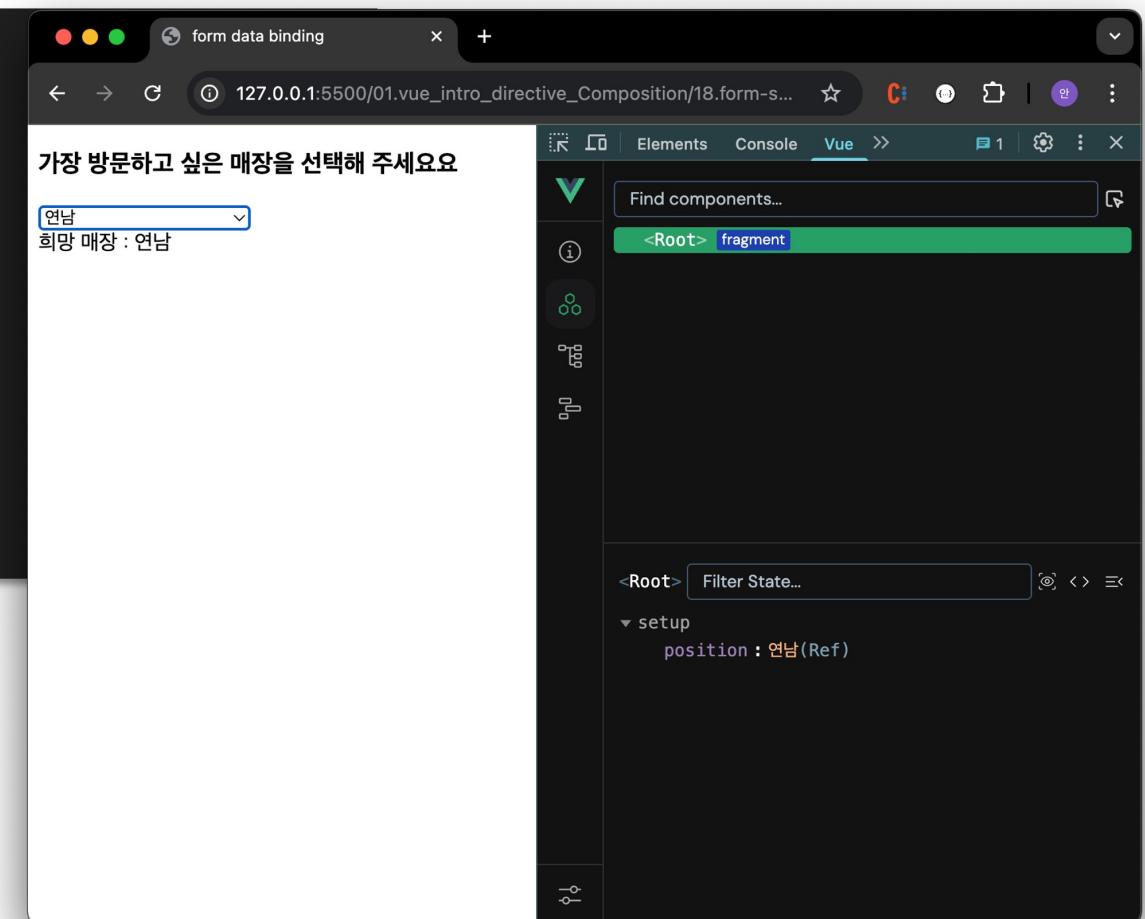
## form input binding: v-model

- ▶ form - select box

```
<div id="app">
  <h3>가장 방문하고 싶은 매장을 선택해 주세요요</h3>
  <select v-model="position">
    <option disabled value="">다음 중 하나를 선택하세요</option>
    <option value="한남">한남</option>
    <option value="연남">연남</option>
    <option value="해운대">해운대</option>
    <option value="무등">무등</option>
    <option value="한옥마을">한옥마을</option>
  </select>
  <div>희망 매장 : {{ position }}</div>
</div>/#app
```

```
const position = ref("");
```

 18.form-select-binding.html



# Directive: Form Input Bindings

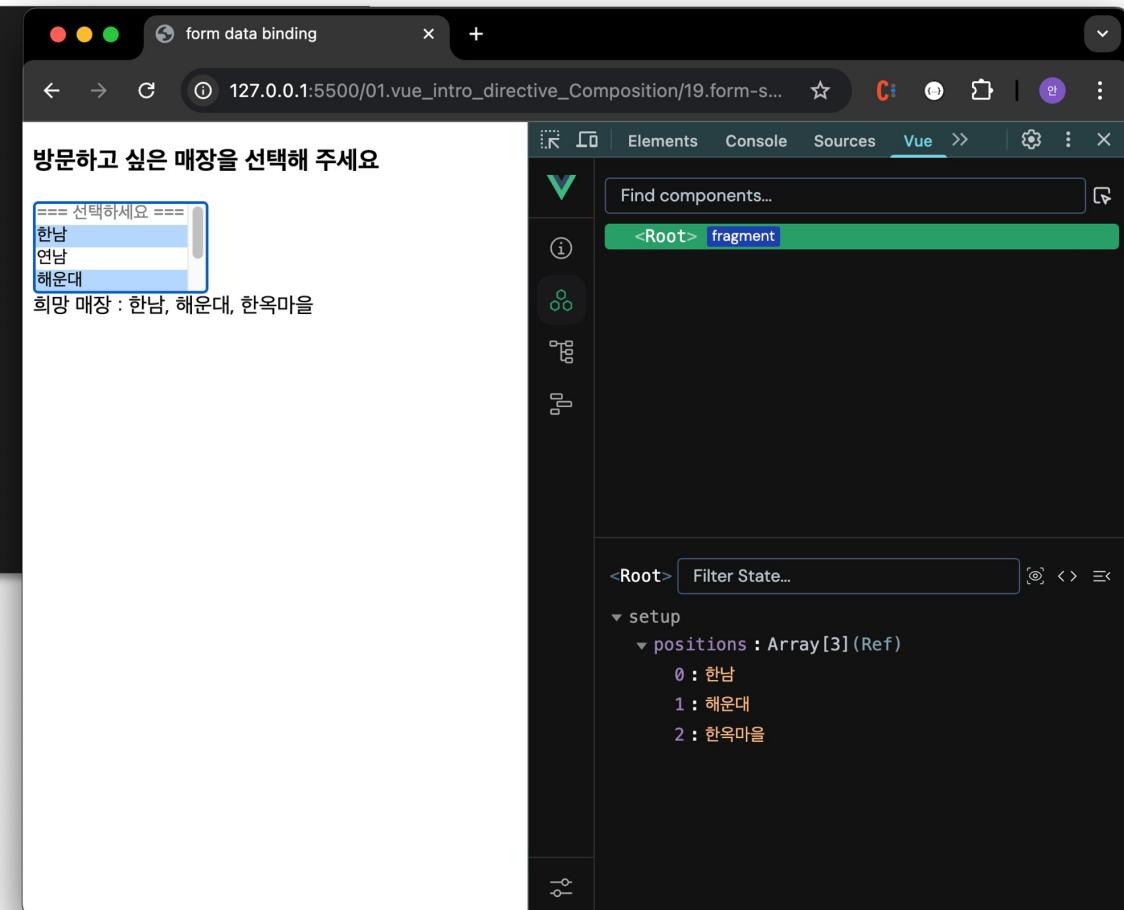
## form input binding: v-model

- ▶ form - select box (multiple)

```
<div id="app">
  <h3>방문하고 싶은 매장을 선택해 주세요</h3>
  <select v-model="positions" multiple>
    <option disabled value="">==== 선택하세요 ====</option>
    <option value="한남">한남</option>
    <option value="연남">연남</option>
    <option value="해운대">해운대</option>
    <option value="무등">무등</option>
    <option value="한옥마을">한옥마을</option>
  </select>
  <div>희망 매장 : {{ positions.join(", ") }}</div>
</div>/#app
```

```
const positions = ref([]);
```

 19.form-select-multi-binding.html



# Directive: Form Input Bindings

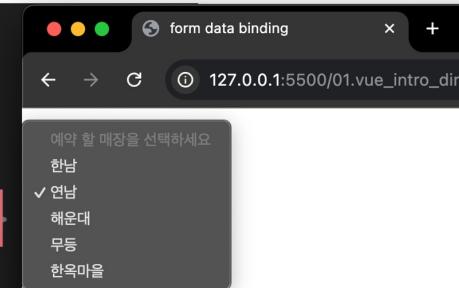
## 5) form input binding: v-model

- ▶ form-select box
- ▶ v-for를 이용한 동적 option 렌더링

```
<div id="app">
  <h3>방문 예약</h3>
  <select v-model="position">
    <option disabled value="">예약 할 매장을 선택하세요</option>
    <option v-for="option in options" :key="option.id" :value="option.value">
      {{option.text}}
    </option>
  </select>
  <div>예약 매장 : {{ position }}</div>
</div>/#app
```

```
const position = ref("");
const options = ref([
  { id: 1, text: "한남", value: "hn" },
  { id: 2, text: "연남", value: "yn" },
  { id: 3, text: "해운대", value: "hud" },
  { id: 4, text: "무등", value: "md" },
  { id: 5, text: "한옥마을", value: "ho" },
]);
```

 20.form-select-for-binding.html



The screenshot shows a browser window with the URL 127.0.0.1:5500/01.vue\_intro\_directive\_Composition/20.form-select-for-binding.html. A dropdown menu is open, displaying five options: "한남", "연남", "해운대", "무등", and "한옥마을". The "연남" option is highlighted with a green checkmark. The browser's developer tools are open, showing the Vue Devtools Network tab. The network panel lists several requests, with the last one being a fragment request.

The Vue Devtools tree view shows the component structure:

- <Root> fragment
  - setup
    - position: yn(Ref)
    - options: Array[5](Ref)
      - 0: Object
        - id: 1
        - text: 한남
        - value: hn
      - 1: Object
      - 2: Object
      - 3: Object

# Directive: Form Input Bindings

## form input binding: 수식어 (Modifiers)

▶ .lazy

- .lazy 수식어를 추가하여 입력폼에서 다른 요소로 포커스가 이동하는 이벤트가 발생할 때 입력한 값을 data와 동기화

```
<input type="text" v-model.lazy="msg1" />
```

▶ .number

- 사용자 입력이 자동으로 숫자로 형 변환 되기를 원하면, v-model이 관리하는 input에 .number 수식어를 추가하면 된다

```
<input type="number" v-model.number="age" />
```

▶ .trim

- v-model이 관리하는 input을 자동으로 trim(앞뒤 공백제거)하기를 원하면 .trim 수식어를 추가하면 된다

```
<input type="text" v-model.trim="msg2" />
```

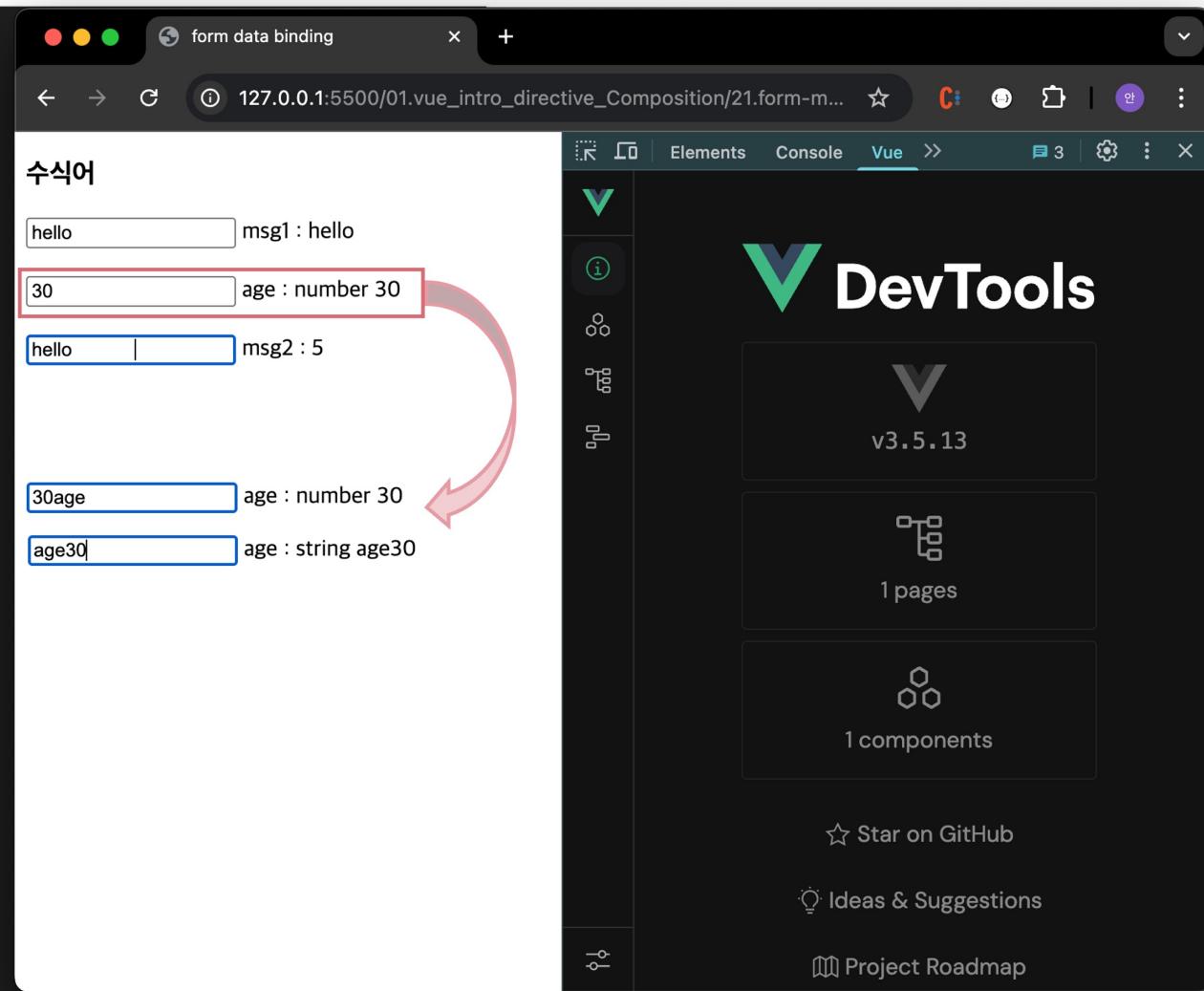


# Directive: Form Input Bindings

## form input binding: 수식어 (Modifiers)

```
<div id="app">
  <h3>수식어</h3>
  <input type="text" v-model.lazy="msg1" />
  msg1 : {{ msg1 }}<br /><br />
  <input type="text" v-model.number="age" />
  age : {{ typeof age}} {{ age }}<br /><br />
  <input type="text" v-model.trim="msg2" />
  msg2 : {{ msg2.length }}<br />
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const msg1 = ref("");
      const age = ref(0);
      const msg2 = ref("");
      return {
        msg1, age, msg2,
      };
    },
  });
  app.mount("#app");
</script>
```



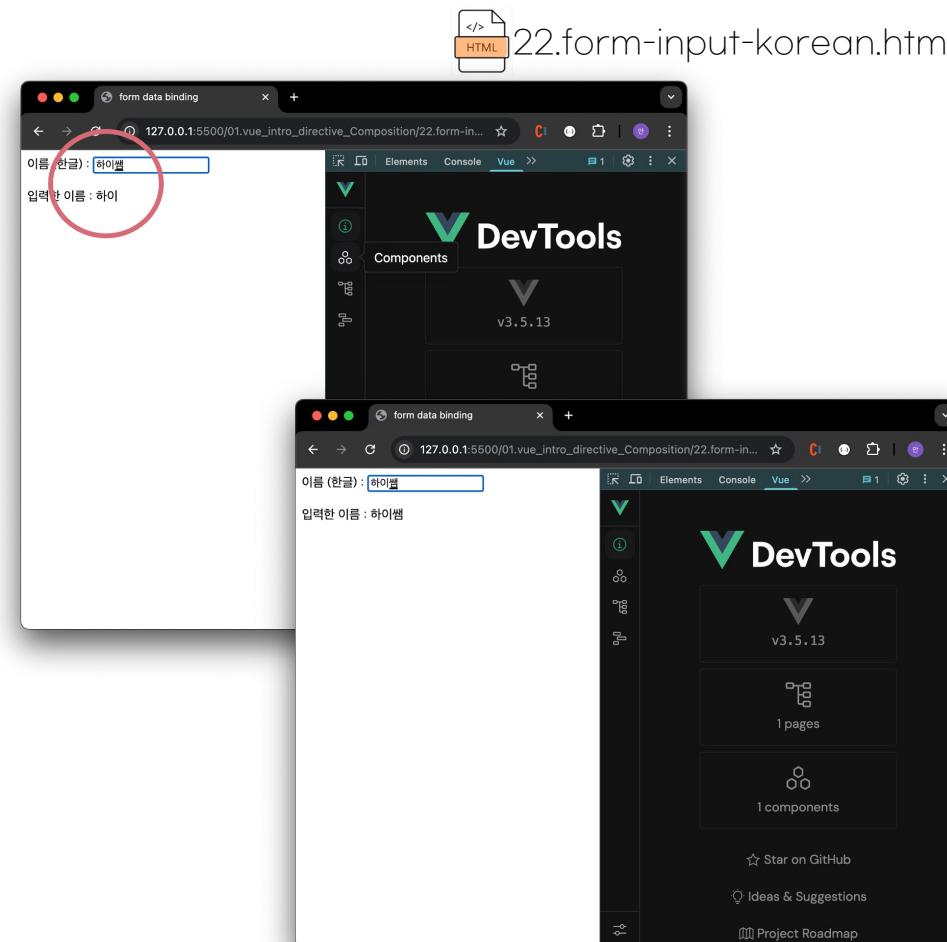
# Directive: Form Input Bindings

## \_form input binding: v-model

- ▶ v-model의 한글처리 문제
- ▶ 한글 값을 받아내는 시점이 한글 한글자가 입력이 완료될 때로 처리하고 있기 때문
- ▶ 이벤트로 해결

```
<div id="app">
  <div>
    <label for="userNmae">이름 (한글) : </label>
    <!-- v-model을 이용하여 한글 입력시 문제 발생 -->
    <!-- <input type="text" id="userNmae" v-model="userNmae" placeholder="이름 입력..." />
    <!-- event로 해결 -->
    <input
      type="text"
      id="userNmae"
      :value="userNmae"
      @input="changeName"
      placeholder="이름 입력...">
    />#userNmae
  </div>
  <br />
  <div>입력한 이름 : {{ userNmae }}</div>
</div>#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const userNmae = ref("");
      const changeName = (event) => {
        userNmae.value = event.target.value;
      };
      return {
        userNmae, changeName,
      };
    },
    app.mount("#app");
</script>
```



# Directive (Conditional Rendering: v-if)



# Directive: Conditional Rendering

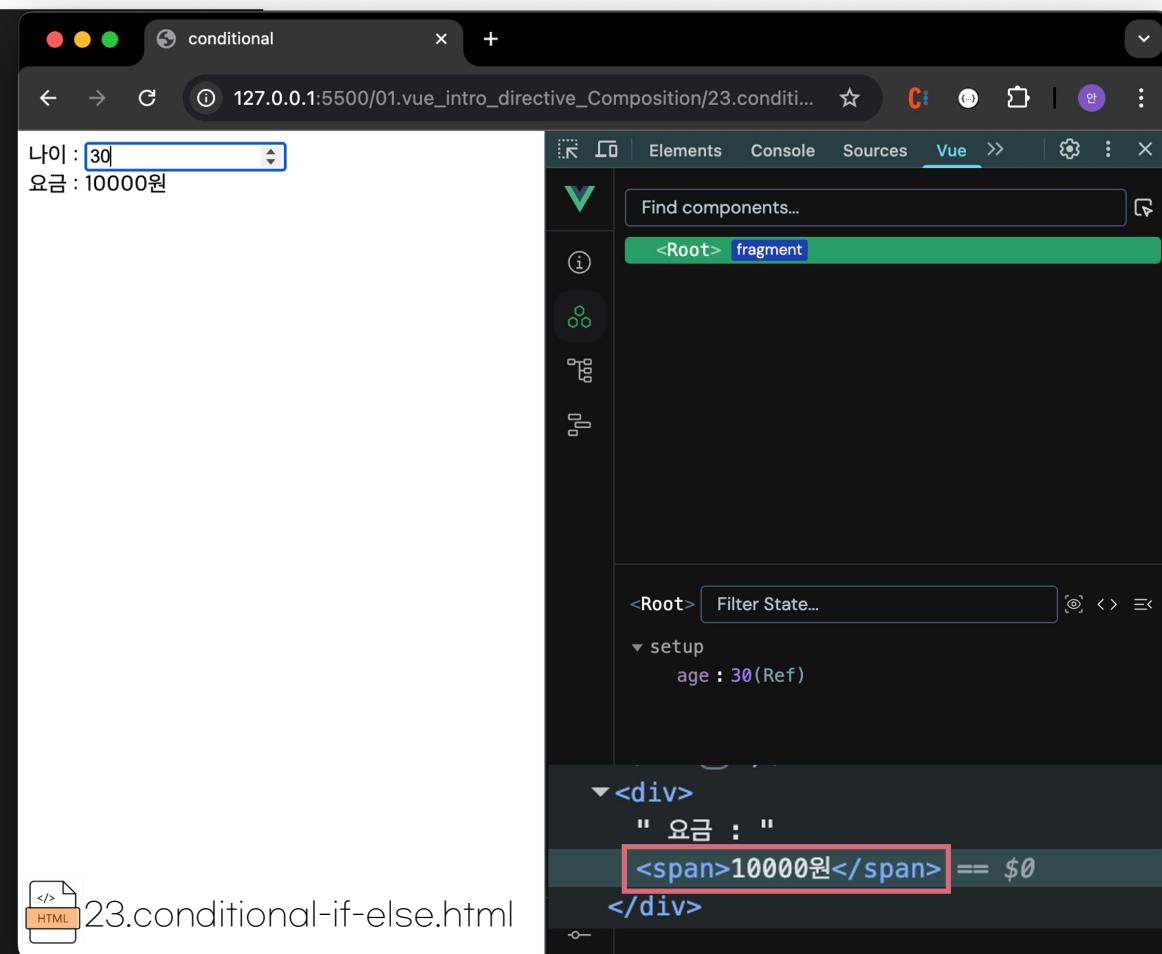
<https://ko.vuejs.org/guide/essentials/conditional.html>

## 조건부 렌더링 (conditional rendering)

- ▶ v-if, v-else-if, v-else
- ▶ 조건에 따라 특정 블록을 렌더링 할지를 결정 (조건이 false일 경우 html 요소를 만들지 않음)

```
<div id="app">
  <div>
    <label for="age">나이 : </label>
    <input type="number" id="age" v-model="age" />
  </div>
  <div>
    요금 :
    <span v-if="age < 10 || age > 80">무료</span>
    <span v-else-if="age < 20">7000원</span>
    <span v-else-if="age < 65">10000원</span>
    <span v-else>3000원</span>
  </div>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const age = ref(0);
      return {
        age,
      };
    },
  });
  app.mount("#app");
</script>
```



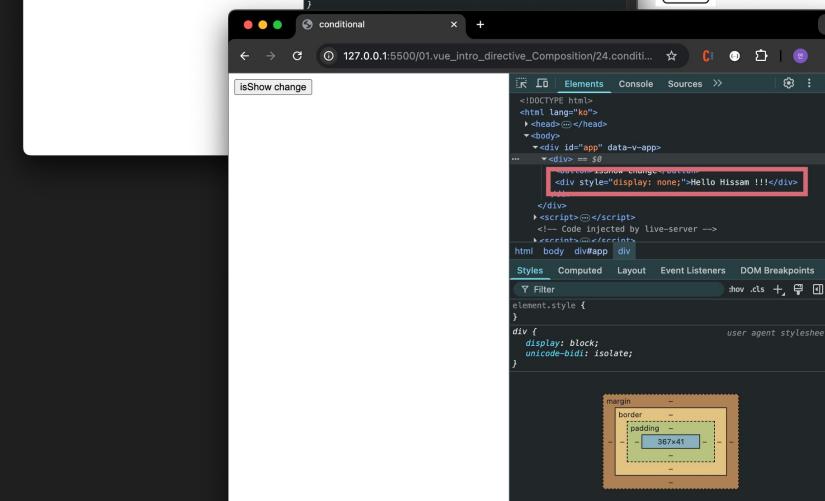
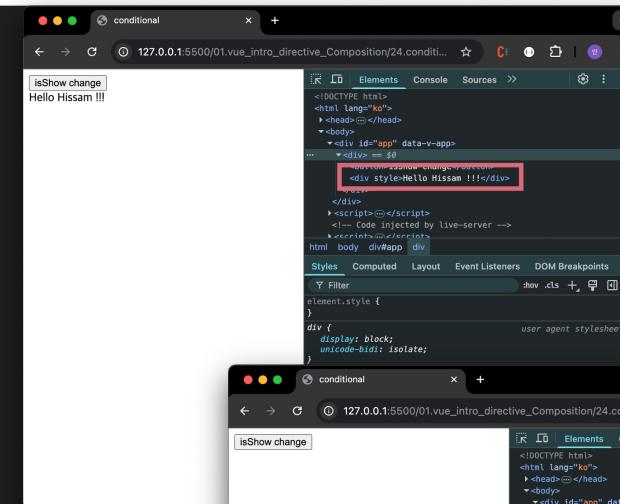
## 조건부 렌더링 (conditional rendering)

### ▶ v-show

- ▶ 조건에 따라 특정 블록을 화면에 보여줄지를 결정
- ▶ 렌더링은 일어나지만 화면 출력여부 결정 ::> **display** css style 속성을 **none**으로 지정

```
<div id="app">
  <div>
    <button @click="changeShow">isShow change</button>
    <div v-show="isShow">{{ message }}</div>
  </div> #app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const isShow = ref(true);
      const message = ref("Hello Hissam !!!");
      const changeShow = () => (isShow.value = !isShow.value);
      return {
        isShow,
        message,
        changeShow,
      };
    },
  });
  app.mount("#app");
</script>
```



24.conditional-show.html

# Directive: Conditional Rendering

## 조건부 렌더링 (conditional rendering)

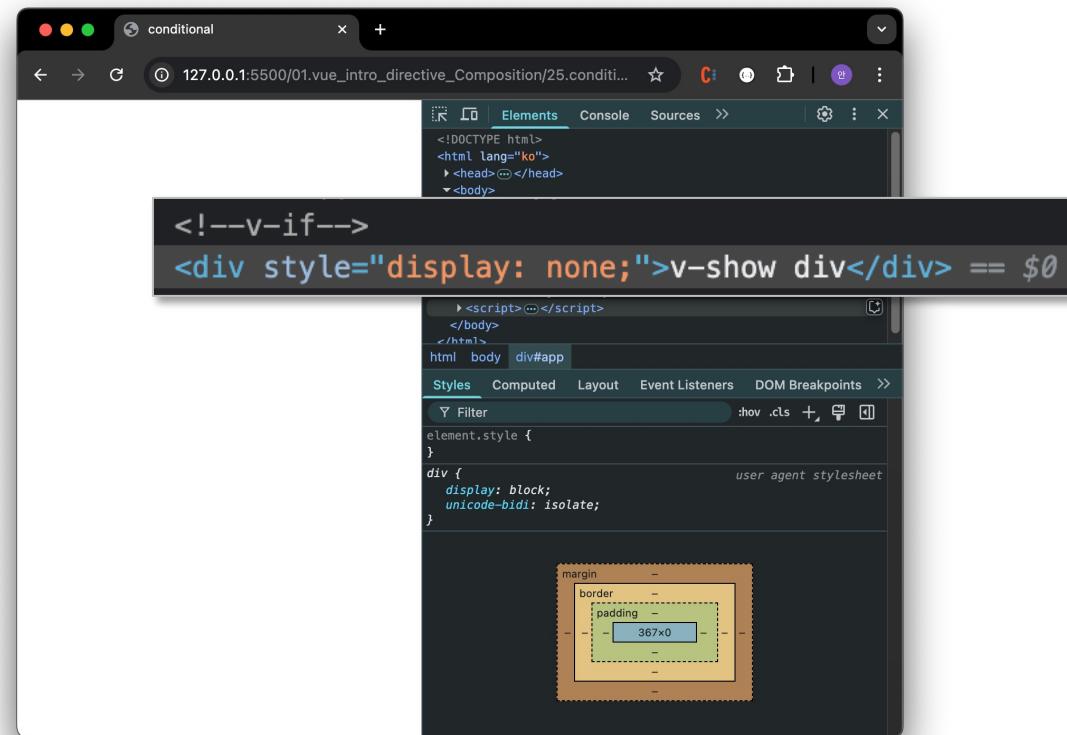
### ▶ v-if VS v-show

	v-if	v-show
렌더링	false일 경우 X	항상 O
false일 경우	element 삭제	display: none 적용
template 지원	O	X
v-else 지원	O	X

25.conditional-if-vs-show.html

```
<div id="app">
  <div v-if="isShow">v-if div</div>
  <div v-show="isShow">v-show div</div>
</div>#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const isShow = ref(false);
      return {
        isShow,
      };
    },
  });
  app.mount("#app");
</script>
```



# Directive (List Rendering: v-for)



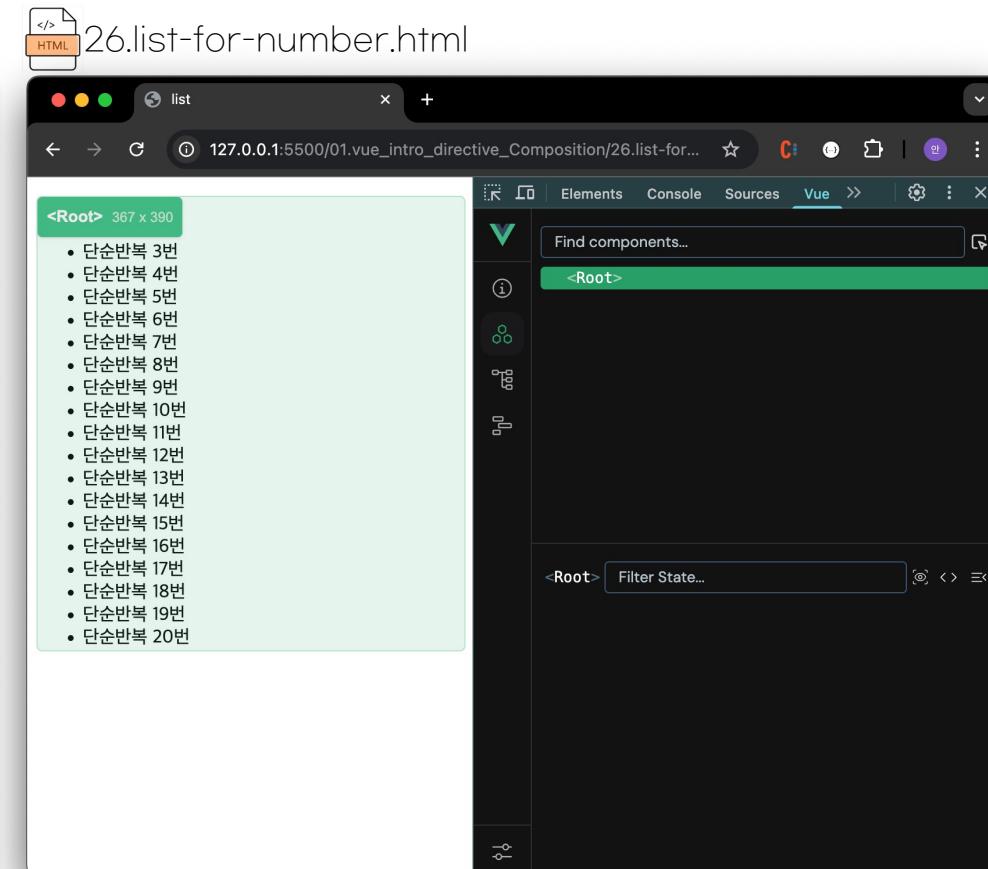
<https://ko.vuejs.org/guide/essentials/list.html>

## 반복 렌더링 (list rendering)

- ▶ v-for
- ▶ 반복적인 데이터를 렌더링 하기 위해 사용
- ▶ 배열이나 객체의 속성 반복에 사용
- ▶ 단순 반복 횟수 지정

```
<div id="app">
  <ul>
    <li v-for="i in 20">단순반복 {{ i }}번</li>
  </> 정수를 이용한 단순 반복 : i는 “0”이 아닌 “1”부터 시작 주의!!!
</div>/#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      return {};
    },
  });
  app.mount("#app");
</script>
```



# Directive: List Rendering

## 5) 반복 렌더링 (list rendering)

- ▶ v-for: array
- ▶ 형식

```
<li v-for="item in items">{{ item.message }}</li>

<li v-for="(item, index) in items" :key="index">{{ item.message }}</li>

<li v-for="({ message }, index) in items">{{ message }} {{ index }}</li>
```

- ▶ items: 배열, item: 배열내 반복되는 element의 “alias”, index: 0부터 시작
- ▶ 세번째 형식과 같이 ES6의 구조 분해 할당 (Destructuring assignment)도 사용 가능
- ▶ 중첩 v-for 가능

```
<li v-for="item in items">
  <span v-for="childItem in item.children">
    {{ item.message }} {{ childItem }}
  </span>
</li>
```

## 반복 렌더링 (list rendering)

### v-for: array

```
<script>
  const { createApp, ref, computed } = Vue;

  const app = createApp({
    setup() {
      const cafes = ref([
        { no: 1, area: "한남", count: 12 },
        { no: 2, area: "연남", count: 14 },
        { no: 3, area: "해운대", count: 11 },
        { no: 4, area: "무등", count: 10 },
        { no: 5, area: "한옥마을", count: 17 },
      ]);
      const menuCount = computed(() => {
        let cnt = 0;
        const arr = [];
        cafes.value.forEach((cafe) => {
          arr.push(cafe.count);
        });
        return arr.reduce((sum, curVal) => sum + curVal);
      });
      return {
        cafes, menuCount,
      };
    },
  });
  app.mount("#app");
</script>
```

```
<div id="app">
  <h3>MM 카페</h3>
  <div id="cnt">총 메뉴 수 : {{ menuCount }}</div>
  <table id="list">
    <thead>
      <tr>
        <th>번호</th>
        <th>지점</th>
        <th>메뉴수</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="(cafe, index) in cafes" :key="cafe.no">
        <td>{{ cafe.no }} ({{ index + 1 }})</td>
        <td>{{ cafe.area }}</td>
        <td>{{ cafe.count }}</td>
      </tr>
    </tbody>
  </table>#/list
</div>#/app
```



27.list-for-array.html

번호	지점	메뉴수
1(1)	한남	12
2(2)	연남	14
3(3)	해운대	11
4(4)	무등	10
5(5)	한옥마을	17

Vue Devtools screenshot showing the component tree and state. The state shows an array of 5 objects, each representing a cafe with properties: area, count, and no.

```

<tr> == $0
<td>1 (1)</td>
<td>한남</td>
<td>12</td>
</tr>
<tr>...</tr>
<tr>...</tr>
<tr>...</tr>
<tr>...</tr>

```

```

<Root> Filter State...
  > setup
    > cafes : Array[5](Ref)
      > 0 : Object
        area : 한남
        count : 12
        no : 1
      > 1 : Object
      > 2 : Object
      > 3 : Object
      > 4 : Object

```

# Directive: List Rendering

## 5) 반복 렌더링 (list rendering)

- ▶ v-for: object
- ▶ 객체의 속성을 반복할 때 사용 가능 (순서는 Object.keys()의 결과에 기반)
- ▶ 형식

```
<li v-for="value in myObject">{{ value }}</li>

<li v-for="(value, key) in myObject">{{ key }}: {{ value }}</li>

<li v-for="(value, key, index) in myObject">{{ index }}. {{ key }}: {{ value }}</li>
```

- ▶ value: 속성의 값, key: 속성명, index: 인덱스

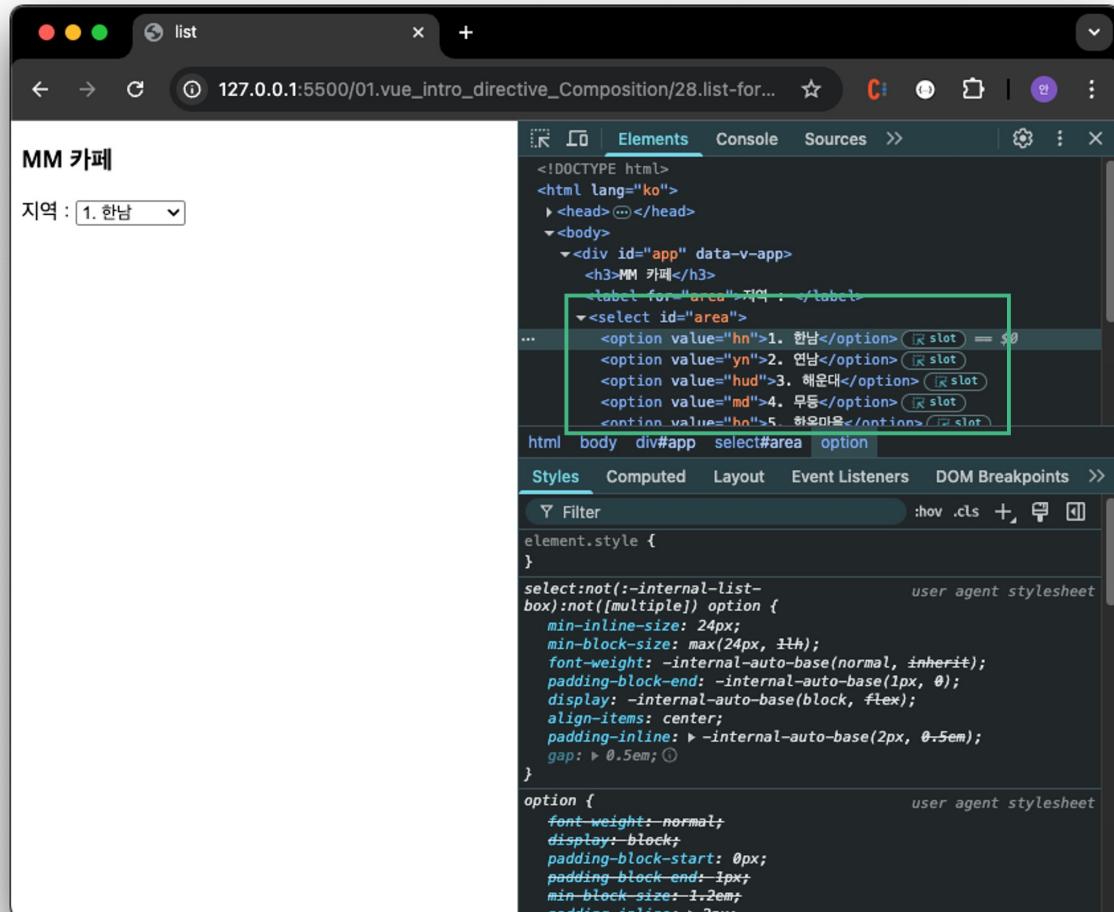
## 59 반복 렌더링 (list rendering)

### ▶ v-for: object

```
<div id="app">
  <h3>MM 카페</h3>
  <label for="area">지역 : </label>
  <select id="area">
    <option v-for="(value, key, index) in areas" :key="key" :value="key">
      {{ index +1 }}. {{ value }}
    </option>
  </select>/#area
</div>/#app

<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const areas = ref({
        hn: "한남",
        yn: "연남",
        hud: "해운대",
        md: "무등",
        ho: "한옥마을",
      });
      return {
        areas,
      };
    },
  });
  app.mount("#app");
</script>
```



28.list-for-object.html

# Directive: List Rendering

## 반복 렌더링 (list rendering)

- ▶ v-for : template
- ▶ 여러 element를 묶어서 반복 렌더링 시 “<template>” 사용
- ▶ <template> 태그는 렌더링 내용에 포함되지 않고 element를 그룹으로 묶어 주기 위한 용도로만 사용

```
<template v-for="(cafe, index) in cafes" :key="cafe.no">
  <tr>
    <td>{{ cafe.no }} ({{ index + 1 }})</td>
    <td>{{ cafe.area }}</td>
    <td>{{ cafe.count }}</td>
  </tr>
  <tr class="divide" v-if="cafe.count == 14">
    <td colspan="3"></td>
  </tr>/.divide
</template>
```

29.list-for-template.html

The screenshot shows a browser window titled "list" displaying a table titled "MM 카페". The table has three columns: 번호 (번호), 지점 (지점), and 메뉴수 (메뉴수). The data is as follows:

번호	지점	메뉴수
1 (1)	한남	12
2 (2)	연남	14
3 (3)	해운대	11
4 (4)	무등	10
5 (5)	한옥마을	17

The browser's developer tools are open, showing the DOM structure. A specific row is highlighted with a gray background, which corresponds to the "divide" row in the table. The developer tools also show the CSS styles applied to the table and its rows.

# Directive: List Rendering

## ⚡ 반복 렌더링 (list rendering)

- ▶ v-for & v-if



v-if 와 v-for 를 함께 사용하는 것은 권장되지 않습니다.

- ▶ v-for와 v-if가 같은 노드에 있을 경우 v-if가 v-for보다 우선순위가 높기 때문에 조건문에서 v-for의 변수에 접근이 불가능
- ▶ 이전 예제에서 아래와 같이 변경할 경우 에러 발생 ::> <template> 활용

```
<tr v-for="(cafe, index) in cafes" v-if="cafe.count == 14" :key="cafe.no">
  <td>{{ cafe.no }} ({{ index + 1 }})</td>
  <td>{{ cafe.area }}</td>
  <td>{{ cafe.count }}</td>
</tr>
```

```
✖ Uncaught TypeError: Cannot read properties of undefined (reading 'count')
  at Proxy.render (eval at compileToFunction (vue.global.js:15199:20), <anonymous>:25:18)
  at renderComponentRoot (vue.global.js:2287:18)
  at ReactiveEffect.componentUpdateFn [as fn] (vue.global.js:7105:48)
  at ReactiveEffect.run (vue.global.js:497:21)
  at instance.update (vue.global.js:7218:53)
  at setupRenderEffect (vue.global.js:7226:7)
  at mountComponent (vue.global.js:7016:7)
  at processComponent (vue.global.js:6969:11)
  at patch (vue.global.js:6455:13)
  at render (vue.global.js:7736:9)
```

# Directive: List Rendering

## ▣ 반복 렌더링 (list rendering)

### ▶ v-for에서의 key 특성

- Vue.js의 가상 DOM은 v-for로 렌더링한 배열 데이터의 순서가 변경되면 HTML의 DOM Element를 이동시키지 않고 기존 DOM Element의 data만 변경
- key에 고유한 변경되지 않는 값을 지정하게 되면 변경되지 않은 data에 대해 가상 DOM이 렌더링을 수행하지 않을 수 있으므로 좋은 성능을 발휘
- DB의 값을 얻어왔을 경우 primary key값을 할당
- 일반적으로 index값을 부여하지 않음 (아래의 문제 발생)

key	0	1	2	3
	A	B	C	D



배열의 특정 index에 데이터가 추가된다면 기존 key에 해당하는 값이 변경  
되므로 모두 렌더링 (reflow & repaint)이 됨  
::> key를 index가 아닌 고유 값 지정으로 해결  
(element의 위치 (reflow)만 변경함)

key	0	1	2	3	4
	Z	A	B	C	D

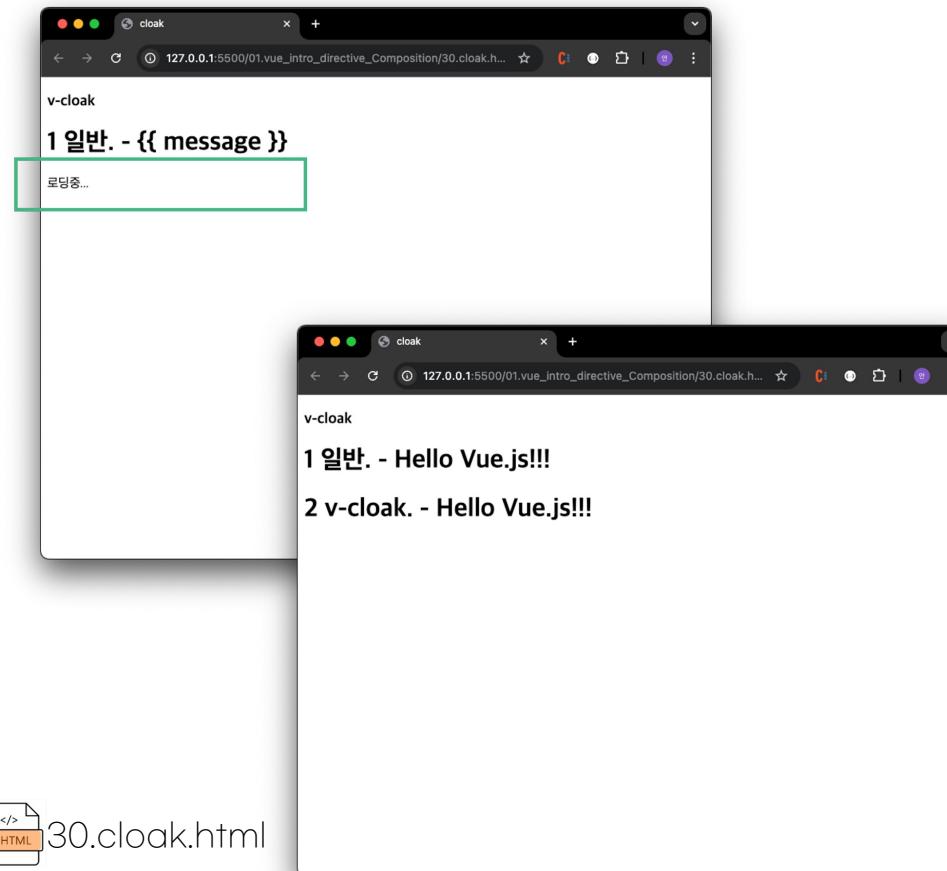
## ⚡ v-cloak

- ▶ Vue Instance가 준비될 때까지 mustache 바인딩을 숨기는데 사용
- ▶ [v-cloak]{display: none}과 같은 CSS 규칙과 함께 사용
- ▶ Vue Instance가 준비되면 v-cloak는 제거됨

```
<div id="app">
  <h3>v-cloak</h3>
  <h1>1 일반. - {{ message }}</h1>
  <div v-cloak>
    <h1>2 v-cloak. - {{ message }}</h1>
  </div>
</div>/#app
<script>
  const { createApp, ref } = Vue;

  setTimeout(function () {
    const app = createApp({
      setup() {
        const message = ref("Hello Vue.js!!!");
        return {
          message,
        };
      },
    });
    app.mount("#app");
  }, 3000);
</script>
```

```
<style>
  [v-cloak]::before {
    content: "로딩중...";
  }
  [v-cloak] > * {
    display: none;
  }
</style>
```



# Thank you !!!

---

Programming Language