

조건부 렌더링

조건부 렌더링 - 1

• 조건부 렌더링(v-if)

조건에 따라 블록을 렌더링하고 화면에 표시함

1. true일 경우 해당 블록을 렌더링
2. false일 경우에는 else if를 확인하여 true인 조건이 있으면 렌더링
3. 모두 다 아닐 경우 else 렌더링

조건부 렌더링 - 2

- v-if 예시

```
var vm = new Vue({  
  el: '#if-example',  
  data: {  
    season: "Summer"  
  }  
})
```

```
<div id="if-example">  
  <h1 v-if="season === 'spring'">  
    봄  
  </h1>  
  <h1 v-else-if="season === 'Summer'">  
    여름  
  </h1>  
  <h1 v-else-if="season === 'autumn'">  
    가을  
  </h1>  
  <h1 v-else-if="season === 'Winter'">  
    겨울  
  </h1>  
  <h1 v-else>  
    해당하는 계절이 아닙니다.  
  </h1>  
</div>
```

조건부 렌더링 - 3

- 조건부 렌더링(v-show)

v-if와 마찬가지로 조건이 true일 때 화면에 표시함
단, v-show는 조건에 상관없이 항상 렌더링되고 DOM에 남아있음

```
var vm = new Vue({  
  el: '#show-example',  
  data: {  
    ok: true  
  }  
})
```

```
<div id="show-example">  
  <h1 v-show="ok">안녕하세요!</h1>  
</div>
```

조건부 렌더링 - 4

• v-show 예시

버튼 클릭 시 shouldShowGreeting 데이터가 논리 부정에 의해 h1 텍스트가 사라지거나 생김
→ 논리부정 : false일 경우 true, true일 경우 false

* 참고 : @click == v-on:click

```
var vm = new Vue({
  el: '#show-example',
  data: {
    shouldShowGreeting: true
  },
  methods: {
    toggleGreeting() {
      this.shouldShowGreeting = !this.shouldShowGreeting;
    }
  }
})
```

```
<div id="show-example">
  <h1 v-show="shouldShowGreeting">안녕하세요!</h1>
  <button @click="toggleGreeting">인사 토글</button>
</div>
```

조건부 렌더링 - 5

- v-if vs v-show

조건에 따라 해당되는 태그의 존재 여부 차이가 있음

v-fi

```
▼ <div id="if-example"> == $0  
  <!-->  
</div>
```

v-show

```
<hr>  
▼ <div id="show-example">  
  <h1 style="display: none;">안녕하세요!</h1>  
</div>  
▶ <script>...</script>
```


리스트 렌더링

리스트 렌더링 - 1

• 리스트 렌더링(v-for)

v-for 디렉티브는 배열을 기반으로 리스트를 렌더링할 수 있음
자바의 for each문과 유사하게 동작

```
var example = new Vue({  
  el: "#list-example",  
  data: {  
    items: [  
      { message : '아이템1'},  
      { message : '아이템2'},  
      { message : '아이템3'},  
      { message : '아이템4'}  
    ]  
  }  
})
```



```
<ul id="list-example">  
  <li v-for="item in items" :key="item.id">  
    {{ item.message }}  
  </li>  
</ul>
```


리스트 렌더링 - 1

- 참고

v-for를 사용할 때는 key를 같이 사용해야 함

* 사용 안해도 에러는 없지만 각 아이템을 고유하게 만들어 추후 업데이트를 수행하거나 DOM 요소를 재사용할 때 필요하기 때문

리스트 렌더링 - 2

• 참고

v-for와 v-if를 같이 사용할 수 있으나, vuejs 공식 문서에서는 사용하지 않을것을 권장
→ v-for이 우선순위가 높음
→ 이후 배울 Vuejs3에서는 v-if가 우선순위가 높음

```
var example = new Vue({  
  el: "#list-example",  
  data: {  
    items: [  
      { message : '아이템1', isComplete: false},  
      { message : '아이템2', isComplete: true},  
      { message : '아이템3', isComplete: true},  
      { message : '아이템4', isComplete: false}  
    ],  
  },  
})
```

```
<div id="list-example">  
  <li v-for="item in items" v-if="!item.isComplete">  
    {{ item.message }}  
  </li>  
</div>
```

리스트 렌더링 - 3

- 참고

우선순위와 key에 대한 내용은 Vuejs 스타일 가이드 문서 참고

<https://v2.ko.vuejs.org/v2/style-guide/#v-if%EC%99%80-v-for%EB%A5%BC-%EB%8F%99%EC%8B%9C%EC%97%90-%EC%82%AC%EC%9A%A9%ED%95%98%EC%A7%80-%EB%A7%88%EC%84%B8%EC%9A%94-%ED%95%84%EC%88%98>

리스트 렌더링 - 4

• 컴포넌트와 사용하기(예시 코드)

```
<div id="todo-list-example">
  <form v-on:submit.prevent="addNewTodo">
    <label for="new-todo">해야 할 작업 추가</label>
    <input
      v-model="newTodoText"
      id="new-todo"
      placeholder=""
    >
    <button>추가</button>
  </form>
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo.id"
      v-bind:title="todo.title"
      v-on:remove="todos.splice(index, 1)"
    > </li>
  </ul>
</div>
```

```
Vue.component('todo-item', {
  template: 'W
    <li>W
      {{ title }}W
      <button v-on:click="$emit(W\'removeW\')">삭제 </button>W
    </li>W
  ',
  props: ['title']
})

new Vue({
  el: '#todo-list-example',
  data: {
    newTodoText: '',
    todos: [
      {
        id: 1,
        title: '밥먹기',
      },
      {
        id: 2,
        title: '잠자기',
      },
      {
        id: 3,
        title: '놀이'
      }
    ],
    nextTodold: 4
  },
  methods: {
    addNewTodo: function () {
      this.todos.push({
        id: this.nextTodold++,
        title: this.newTodoText
      })
      this.newTodoText = ''
    }
  }
})
```

리스트 렌더링 - 5

• 컴포넌트와 사용하기(예시 코드)

```
<div id="todo-list-example">  
  <form v-on:submit.prevent="addNewTodo">  
    <label for="new-todo">해야 할 작업 추가</label>  
    <input  
      v-model="newTodoText" —————> 입력한 값을 Vue 데이터에 연결  
      id="new-todo" —————> label 태그와 연결(for)하기 위한 id (vue 아님)  
      placeholder=""  
    >  
    <button>추가</button>  
  </form>  
  <ul>  
    <li  
      is="todo-item" —————> li요소를 todo-item 컴포넌트로 취급하도록 지시  
      v-for="(todo, index) in todos"  
      v-bind:key="todo.id"  
      v-bind:title="todo.title"  
      v-on:remove="todos.splice(index, 1)"  
    > </li>  
  </ul>  
</div>
```

```
Vue.component('todo-item', {  
  template: 'W  
    <li>W  
    {{ title }}W  
    <button v-on:click="$emit(W\'removeW\')">삭제 </button>W  
  </li>W  
,  
  props: ['title']  
)  
  
new Vue({  
  el: '#todo-list-example',  
  data: {  
    newTodoText: '',  
    todos: [  
      {  
        id: 1,  
        title: '밥먹기',  
      },  
      {  
        id: 2,  
        title: '잠자기',  
      },  
      {  
        id: 3,  
        title: '놀이',  
      },  
    ],  
    nextTodoId: 4  
  },  
  methods: {  
    addNewTodo: function () {  
      this.todos.push({  
        id: this.nextTodoId++,  
        title: this.newTodoText  
      })  
      this.newTodoText = ''  
    }  
  }  
)
```

리스트 렌더링 - 6

• 컴포넌트와 사용하기(예시 코드)

```
<div id="todo-list-example">
  <form v-on:submit.prevent="addNewTodo">
    <label for="new-todo">해야 할 작업 추가</label>
    <input
      v-model="newTodoText"
      id="new-todo"
      placeholder=""
    >
    <button>추가</button>
  </form>
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo.id"
      v-bind:title="todo.title"
      v-on:remove="todos.splice(index, 1)"
    ></li>
  </ul>
</div>
```

index에 위치한 배열 요소 1개를 제거
= 삭제 버튼을 클릭했을 때 해당 요소 제거

컴포넌트 호출

```
Vue.component('todo-item', {
  template: 'W'
  <li>W
    {{ title }}W
    <button v-on:click="$emit('remove', W)">삭제</button>W
  </li>W
},
  props: ['title']
})

new Vue({
  el: '#todo-list-example',
  data: {
    newTodoText: '',
    todos: [
      {
        id: 1,
        title: '밥먹기',
      },
      {
        id: 2,
        title: '잠자기',
      },
      {
        id: 3,
        title: '놀이'
      }
    ],
    nextTodoId: 4
  },
  methods: {
    addNewTodo: function () {
      this.todos.push({
        id: this.nextTodoId++,
        title: this.newTodoText
      })
      this.newTodoText = ''
    }
  }
})
```

컴포넌트가 호출된곳에 삽입할 HTML

remove 이벤트 전달
→ \$emit : 다른 컴포넌트에게 이벤트를 전달

리스트 렌더링 - 7

• 컴포넌트와 사용하기(예시 코드)

```
<div id="todo-list-example">
  <form v-on:submit.prevent="addNewTodo">
    <label for="new-todo">해야 할 작업 추가</label>
    <input
      v-model="newTodoText"
      id="new-todo"
      placeholder=""
    >
    <button>추가</button>
  </form>
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo.id"
      v-bind:title="todo.title"
      v-on:remove="todos.splice(index, 1)"
    > </li>
  </ul>
</div>
```

```
Vue.component('todo-item', {
  template: 'W
    <li>W
      {{ title }}W
      <button v-on:click="$emit(W\'removeW\')">삭제</button>W
    </li>W
  ',
  props: ['title']
})

new Vue({
  el: '#todo-list-example',
  data: {
    newTodoText: '',
    todos: [
      {
        id: 1,
        title: '밥먹기',
      },
      {
        id: 2,
        title: '잠자기',
      },
      {
        id: 3,
        title: '놀이'
      }
    ],
    nextTodoId: 4
  },
  methods: {
    addNewTodo: function () {
      this.todos.push({
        id: this.nextTodoId++,
        title: this.newTodoText
      })
      this.newTodoText = ''
    }
  }
})
```

→ 컴포넌트의 템플릿 안에 데이터를 전달

리스트 렌더링 - 8

• 컴포넌트와 사용하기(예시 코드)

```
<div id="todo-list-example">
  <form v-on:submit.prevent="addNewTodo">
    <label for="new-todo">해야 할 작업 추가</label>
    <input
      v-model="newTodoText"
      id="new-todo"
      placeholder=""
    >
    <button>추가</button>
  </form>
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo.id"
      v-bind:title="todo.title"
      v-on:remove="todos.splice(index, 1)"
    ></li>
  </ul>
</div>
```

1. todos 데이터를 가져와서 하나씩 꺼냄
2. 데이터의 id와 title을 각각 key, title로 넣음

삭제할 때 사용하기 위한 인덱스

```
Vue.component('todo-item', {
  template: 'W
    <li>W
      {{ title }}W
      <button v-on:click="$emit(W\'removeW\')">삭제</button>W
    </li>W
  ',
  props: ['title']
})
```

```
new Vue({
  el: '#todo-list-example',
  data: {
    newTodoText: '',
    todos: [
      {
        id: 1,
        title: '밥먹기',
      },
      {
        id: 2,
        title: '잠자기',
      },
      {
        id: 3,
        title: '놀이'
      }
    ],
    nextTodoId: 4
  },
  methods: {
    addNewTodo: function () {
      this.todos.push({
        id: this.nextTodoId++,
        title: this.newTodoText
      })
      this.newTodoText = ''
    }
  }
})
```

새로운 해야할 작업의 텍스트를 저장하는 변수

다음 해야할 작업의 id를 넘길 값

리스트 렌더링 - 9

• 컴포넌트와 사용하기(예시 코드)

```
<div id="todo-list-example">
  <form v-on:submit.prevent="addNewTodo">
    <label for="new-todo">해야 할 작업 추가</label>
    <input
      v-model="newTodoText"
      id="new-todo"
      placeholder=""
    >
    <button>추가</button>
  </form>
  <ul>
    <li
      is="todo-item"
      v-for="(todo, index) in todos"
      v-bind:key="todo.id"
      v-bind:title="todo.title"
      v-on:remove="todos.splice(index, 1)"
    ></li>
  </ul>
</div>
```

추가 버튼을 클릭했을 때
실행할 함수(기능)

```
Vue.component('todo-item', {
  template: 'W
    <li>W
      {{ title }}W
      <button v-on:click="$emit(W\'removeW\')">삭제 </button>W
    </li>W
  ',
  props: ['title']
})
```

```
let vm = new Vue({
  el: '#todo-list-example',
  data: {
    newTodoText: '',
    todos: [
      {
        id: 1,
        title: '밥먹기',
      },
      {
        id: 2,
        title: '잠자기',
      },
      {
        id: 3,
        title: '놀이'
      }
    ],
    nextTodoId: 4
  },
  methods: {
    addNewTodo: function () {
      this.todos.push({
        id: this.nextTodoId++,
        title: this.newTodoText
      })
      this.newTodoText = ''
    }
  }
})
```

push : todos 데이터에 추가

데이터의 nextTodoId의 값을 1 증가
사용자가 입력한 값을 데이터의 newTodoText에 담음

다음 입력을 받기 위한 텍스트 초기화

이벤트 핸들링

이벤트 핸들링

• 이벤트 청취

v-on 디렉티브를 사용하여 DOM 이벤트를 듣고 js를 실행할 수 있음

```
<div id="event-1">
  <button @click="counter += 1"> Add 1</button>
  <p>버튼을 클릭한 횟수는 {{ counter }}번 입니다.</p>
</div>
```

```
var event1 = new Vue({
  el: "#event-1",
  data: {
    counter: 0
  }
})
```

이벤트 핸들링

• 이벤트 청취

조건부 렌더링을 사용하여 10번을 클릭하면 텍스트가 변경되는 웹 만들어보기

```
<div id="event-1">
  <button @click="counter += 1"> Add 1</button>
  <p v-if="counter < 10">버튼을 클릭한 횟수는 {{ counter }}번 입니다.</p>
  <p v-else-if="counter >= 10">10번을 모두 클릭하셨습니다!</p>
</div>
```

```
var event1 = new Vue({
  el: "#event-1",
  data: {
    counter: 0
  }
})
```

이벤트 핸들링

• 메소드 이벤트 핸들러

메소드를 활용한 이벤트 핸들러 구현

```
<div id="method-handler">
  <button @click="greet">버튼</button>
</div>
```

```
var vm = new Vue({
  el: '#method-handler',
  data : {
    name: 'Vue.js'
  },
  methods : {
    greet: function(event) {
      alert("Hello, " + this.name);
      if(event) {
        alert(event.target.tagName);
      }
    }
  }
})
```

클릭 이벤트를 가져옴

이벤트가 발생한 태그를 출력

크롬 개발자 도구

```
pressure: 0
relatedTarget: null
returnValue: true
screenX: -1862
screenY: 128
shiftKey: false
sourceCapabilities: InputDeviceCapabilities
srcElement: button
tangentialPressure: 0
target: button
accessKey: ""
ariaAtomic: null
ariaAutoComplete: null
ariaBrailleLabel: null
ariaBrailleRoleDescription: null
ariaBusy: null
ariaChecked: null
ariaColCount: null
ariaColIndex: null
```

이벤트 핸들링

• 인라인 메소드 핸들러

인라인 방식으로 함수를 호출하고 매개변수를 전달하는 방법

```
<div id="inline-handler">
  <button v-on:click="say('첫번째 버튼')">버튼1</button>
  <button v-on:click="say('두번째 버튼')">버튼2</button>
</div>
```

```
var vm = new Vue({
  el: '#inline-handler',
  methods: {
    say: function (message) {
      alert(message)
    }
  }
})
```



watch

watch

- watch

데이터를 감시하여, 값이 변경될 때 실행되는 속성

```
<div id="Count">
  <input type="text" v-model="message" >
  <p>글자 수 : {{ charCount }}</p>
</div>
```

양방향 데이터 바인딩



```
var vm = new Vue({
  el: "#Count",
  data: {
    message: '',
    charCount: 0
  },
  watch: {
    message(newMessage) {
      this.charCount = newMessage.length;
    }
  }
})
```

초기값

watch가 변경된 데이터를 감지

watch

- watch

아래와 같이 변경되기 전의 데이터도 감지할 수 있음

```
<div id="Count">
  <input type="text" v-model="message" >
  <p>글자 수 : {{ charCount }}</p>
  <p>이전 글자 수 : {{ oldCount }}</p>
</div>
```

```
var vm = new Vue({
  el: "#Count",
  data: {
    message: '',
    charCount: 0,
    oldCount: 0
  },
  watch: {
    message(newMessage, oldMessage) {
      this.charCount = newMessage.length;
      this.oldCount = oldMessage.length;
    }
  }
})
```

watch가 변경되기 전 데이터를 감지