Vue.js 기초

2020-04-10

이승진

**학습목표**

v-model 속성

v-on:click 속성

v-show 속성

v-bind 속성

checkbox 구현 방법

radio 버튼 구현 방법

**목차**

[1. hello2 2](#_Toc37784678)

[1) hello2.html 2](#_Toc37784679)

[2) v-model 3](#_Toc37784680)

[3) v-on:click 5](#_Toc37784681)

[4) v-show 10](#_Toc37784682)

[2. toggle 17](#_Toc37784683)

[1) v-model, checkbox 17](#_Toc37784684)

[2) v-show 19](#_Toc37784685)

[3) v-if 21](#_Toc37784686)

[4) radio 버튼 23](#_Toc37784687)

[5) v-bind 28](#_Toc37784688)

[3. 과제 30](#_Toc37784689)

[1) empty1.html 30](#_Toc37784690)

[2) empty2.html 31](#_Toc37784691)

# hello2

이 강의노트의 예제 파일들을 public/ex01 폴더에 만들자. (필수는 아님)

## hello2.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>  </head>  <body>    <div id="app">      <h1>{{ title }}</h1>      <p>{{ body }}</p>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: 'hello world',          body: 'good morning'        }      })      window.onload = function() {        let helloRef = firebase.database().ref("hello");        helloRef.on('value', (snapshot) => {          let hello = snapshot.val();          app.title = hello.title;          app.body = hello.body;        });      };     </script>  </body>  </html> |

지난 시간의 예제 hello2.html

## v-model

### hello2a.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>  </head>  <body>    <div id="app">      <h1>{{ title }}</h1>      <p>{{ body }}</p>      <hr />      <input type="text" v-model="title" /> <br />      <input type="text" v-model="body" />    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: 'hello world',          body: 'good morning'        }      })      window.onload = function() {        let helloRef = firebase.database().ref("hello");        helloRef.on('value', (snapshot) => {          let hello = snapshot.val();          app.title = hello.title;          app.body = hello.body;        });      };     </script>  </body>  </html> |

#### v-model

data: {

  title: 'hello world',

body: 'good morning'

}

Vue 앱에 title 속성과 body 속성을 만든다.

{{ title }}

Vue 앱의 title 속성 값을 출력한다.

title 속성값이 변경되면, 그 변경된 값이 즉시 다시 출력된다.

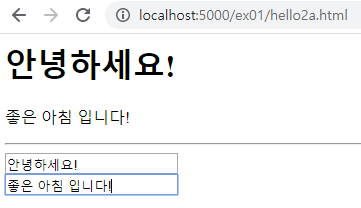
v-model="title"

이 입력 태그에 입력된 값을 Vue 앱의 title 속성에 즉시 대입한다.

문자 하나가 입력되자 마자 즉시 title 속성에 대입된다.

title 속성값이 변경되자마자 그 변경된 값이 {{ title }} 부분에 즉시 출력된다.

그런데 hello2a.html 소스 코드에, 변경된 title 값을 DB에 저장하는 코드는 없다.



input 태그의 내용을 수정하면, 화면 위쪽에 출력된 내용도 즉시 수정된다.

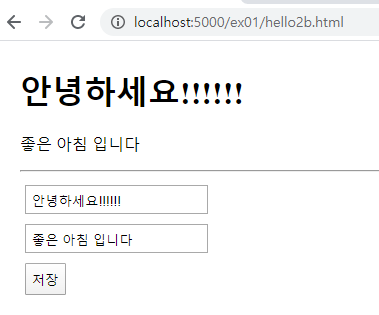
수정된 내용은 DB에 저장되지 않기 때문에, 새로고침 하면 원래 내용으로 되돌아 간다.

DB의 데이터가 다시 전달되어 출력되기 때문이다.

## v-on:click

### hello2b.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <style>      body { padding: 20px; }      input, button { padding: 5px; margin: 5px; }    </style>  </head>  <body>    <div id="app">      <h1>{{ title }}</h1>      <p>{{ body }}</p>      <hr />      <input type="text" v-model="title" /> <br />      <input type="text" v-model="body" /> <br />      <button type="button" v-on:click="save">저장</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: 'hello world',          body: 'good morning'        },        methods: {          save: function () {            let helloRef = firebase.database().ref("hello");            helloRef.set({ title: this.title, body: this.body });          }        }      })      window.onload = function() {        let helloRef = firebase.database().ref("hello");        helloRef.on('value', (snapshot) => {          let hello = snapshot.val();          app.title = hello.title;          app.body = hello.body;        });      };     </script>  </body>  </html> |



input 태그의 내용을 수정하고 저장 버튼을 클릭하면, 그 내용이 DB에 저장된다.

var app = new Vue({ 생략 })

Vue 앱 객체를 생성하여 전역 변수 app에 대입한다.

Vue 생성자 파라미터는 javascript 객체이다. (노란색으로 칠한 부분)

이 객체에, Vue 앱의 기능을 구현해야 한다.

Vue 앱의 기능을 구현한 이 객체를 요약하면 다음과 같다.

{

  el: '#app',

  data: { 생략 },

  methods: { 생략 }

}

이 객체에는 el, data, methods 세 속성이 있다.

el 속성의 값은 문자열이다.

data 속성과 methods 속성의 값은 javascript 객체이다.

data 속성의 값은 다음과 같다.

{

title: 'hello world',

  body: 'good morning'

}

data 속성의 값은 Vue 앱 객체의 속성을 정의한다.

methods 속성의 값은 다음과 같다.

{

save: function () {

   let helloRef = firebase.database().ref("hello");

    helloRef.set({ title: this.title, body: this.body });

  }

}

methods 속성의 값은 Vue 앱 객체의 메소드를 정의한다.

save 메소드의 본문을 살펴보자

let helloRef = firebase.database().ref("hello");

firebase realtime database의 hello 항목에 대한 원격 참조 객체를 구하여

helloRef 지역 변수에 대입한다.

helloRef.set({ title: this.title, body: this.body });

메소드에서 this는 자기 자신을 참조한다.

this.title은 Vue 앱 객체의 title 속성이다.

{ title: this.title, body: this.body }

javascript 객체를 새로 생성한다.

이 객체에는 title, 속성과 body 속성이 만들어진다.

이 속성들의 값은 각각 this.title, this.body 값이다.

this.title, this.body는 Vue 앱의 속성이다.

즉 Vue 앱의 title, body 속성값으로 새 객체를 생성한다.

helloRef.set({ title: this.title, body: this.body });

그렇게 생성된 객체를, firebase DB의 hello 항목에 저장한다.

즉 Vue 앱 객체의 title 속성과 body 속성값이, firebase DB의 hello 항목에 저장된다.

#### v-on:click

<button type="button" v-on:click="save">

이 버튼을 클릭하면, Vue 앱의 save 메소드가 즉시 호출된다.

### hello2c.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <style>      body { padding: 20px; }      input, button { padding: 5px; margin: 5px; }    </style>  </head>  <body>    <div id="app">      <h1>{{ title }}</h1>      <p>{{ body }}</p>      <hr />      <input type="text" v-model="title\_edit" /> <br />      <input type="text" v-model="body\_edit" /> <br />      <button type="button" v-on:click="save">저장</button>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: 'hello world',          body: 'good morning',          title\_edit: '',          body\_edit: ''        },        methods: {          save: function () {            let helloRef = firebase.database().ref("hello");            helloRef.set({ title: this.title\_edit, body: this.body\_edit });          }        }      })      window.onload = function() {        let helloRef = firebase.database().ref("hello");        helloRef.on('value', (snapshot) => {          let hello = snapshot.val();          app.title = hello.title;          app.body = hello.body;        });      };     </script>  </body>  </html> |

data: {

title: 'hello world',

    body: 'good morning',

    title\_edit: '',

    body\_edit: ''

}

Vue 앱 객체에, title\_edit 속성과 body\_edit 속성을 추가한다.

<input type="text" v-model="title\_edit" /> <br />

<input type="text" v-model="body\_edit" /> <br />

input 태그에 입력되는 값은 title\_edit, body\_edit 속성에 저장된다.

title 속성과 body 속성값은 변하지 않는다.

helloRef.set({ title: this.title\_edit, body: this.body\_edit });

저장 버튼을 누르면, title\_edit 속성과 body\_edit 속성의 값이 DB의 hello 항목에 저장된다.

helloRef.on('value', (snapshot) => {

let hello = snapshot.val();

app.title = hello.title;

  app.body = hello.body;

});

DB의 hello 항목의 값이 변경되면, (노란색으로 칠한) 콜백 함수가 즉시 호출된다.

이 콜백 함수의 파라미터 snapshot에, DB의 hello 항목의 새 값이 담겨져 전달된다.

이렇게 전달된 hello 항목의 title 속성과 body 속성을 각각 Vue 앱의 title 속성과 body 속성에 대입한다.

<h1>{{ title }}</h1>

<p>{{ body }}</p>

Vue 앱의 title 속성과 body 속성의 값이 변경되면, 그 값이 즉시 출력된다.

helloRef.on('value', (snapshot) => {

let hello = snapshot.val();

  app.title = hello.title;

  app.body = hello.body;

});

이 콜백함수는 Vue 앱 객체의 메소드가 아니다.

따라서 Vue 앱 객체를 참조하려면, 전역 변수 app을 사용해야 한다.

save: function () {

let helloRef = firebase.database().ref("hello");

  helloRef.set({ title: this.title\_edit, body: this.body\_edit });

}

save 메소드는 Vue 앱 객체의 메소드이다.

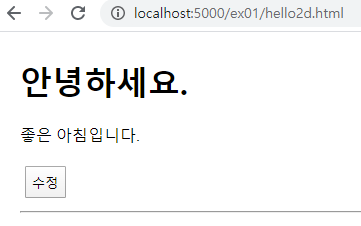
이 메소드 본문에서 this는 Vue 앱 객체이다.

javascript 언어에서는 this를 생략할 수 없다는 점에 주의해야 한다.

## v-show

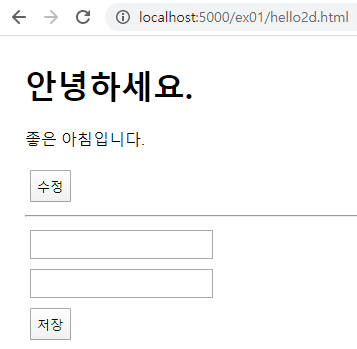
### hello2d.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <style>      body { padding: 20px; }      input, button { padding: 5px; margin: 5px; }    </style>  </head>  <body>    <div id="app">      <h1>{{ title }}</h1>      <p>{{ body }}</p>      <button type="button" v-on:click="edit">수정</button>      <hr />      <div v-show="editing">        <input type="text" v-model="title\_edit" /> <br />        <input type="text" v-model="body\_edit" /> <br />        <button type="button" v-on:click="save">저장</button>      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: 'hello world',          body: 'good morning',          title\_edit: '',          body\_edit: '',          editing: false        },        methods: {          edit: function() {            this.editing = true;          },          save: function () {            let helloRef = firebase.database().ref("hello");            helloRef.set({ title: this.title\_edit, body: this.body\_edit });            this.editing = false;          }        }      })      window.onload = function() {        let helloRef = firebase.database().ref("hello");        helloRef.on('value', (snapshot) => {          let hello = snapshot.val();          app.title = hello.title;          app.body = hello.body;        });      };     </script>  </body>  </html> |



첫 화면에서 input 태그들을 보이지 않는다.

수정 버튼을 클릭하면



input 태그들과 저장 버튼이 보인다.

data: {

  title: 'hello world',

  body: 'good morning',

  title\_edit: '',

  body\_edit: '',

editing: false

}

Vue 앱 객체에 editing 속성이 추가되었다.

methods: {

edit: function() {

   this.editing = true;

},

  save: function () {

   let helloRef = firebase.database().ref("hello");

    helloRef.set({ title: this.title\_edit, body: this.body\_edit });

    this.editing = false;

  }

}

javascript 문법에 주목하자.

methods 속성 값은 객체이다. 이 객체는 다음과 같은 형태이다.

methods: {

edit: function() { 생략 },

save: function () { 생략 }

}

methods 속성 값은 객체이고, 이 객체에는 edit 속성과 save 속성이 있다.

이 두 속성의 값은 함수이다. 이 두 함수는 Vue 앱 객체의 메소드가 된다.

edit 속성값 끝에 콤마(,)가 있어야 함에 주목하자.

<div v-show="editing">

... 생략 ...

</div>

v-show 속성의 값은 javascript 표현식이다.

이 표현식의 값이 false 이면, 태그는 화면에 보이지 않게 되고,

true이면 화면에 보이게 된다.

editing은 Vue 앱의 속성이다. 이 속성의 초기 값은 false 이다.

따라서 이 div 태그와 그 내용은 처음에 화면에 보이지 않는다.

<button type="button" v-on:click="edit">수정</button>

수정 버튼을 클릭하면 edit 메소드가 호출된다.

edit: function() {

this.editing = true;

}

edit 메소드는 editing 속성값에 true를 대입한다.

<div v-show="editing">

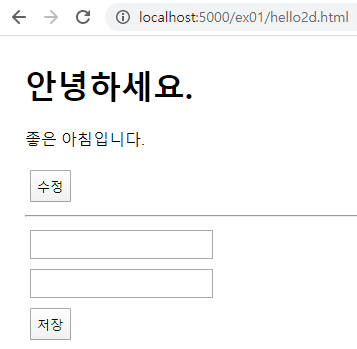
  <input type="text" v-model="title\_edit" /> <br />

  <input type="text" v-model="body\_edit" /> <br />

<button type="button" v-on:click="save">저장</button>

</div>

editing 속성값이 true가 되자 마자 이 태그는 화면에 보여진다.



화면에 보여진 input 태그에 값을 입력하고 저장 버튼을 누르면

save: function () {

let helloRef = firebase.database().ref("hello");

  helloRef.set({ title: this.title\_edit, body: this.body\_edit });

  this.editing = false;

}

save 메소드가 호출된다.

input 태그에 입력된 값들이 DB의 hello 항목에 저장된다.

앱의 editing 속성에 false가 대입되면,

입력 태그와 버튼을 포함하는 <div v-show="editing"> 태그는 다시 보이지 않게 된다.

DB의 hello 항목에 새로 저장된 내용은, 앱의 콜백 함수의 파라미터로 전달된다.

      helloRef.on('value', (snapshot) => {

        let hello = snapshot.val();

        app.title = hello.title;

        app.body = hello.body;

      });

DB의 hello 항목의 값이 바뀌면,

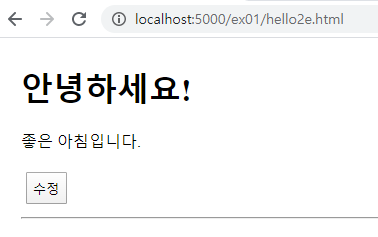
firebase DB 원격 참조 객체(helloRef)의 value 이벤트 핸들러 콜백 함수가 호출된다.

바뀐 새 값이, 콜백 함수의 파라미터에 담겨서 전달된다.

이 새 값은, Vue 앱 객체의 title 속성과 body 속성에 대입되어 화면에 출력된다.

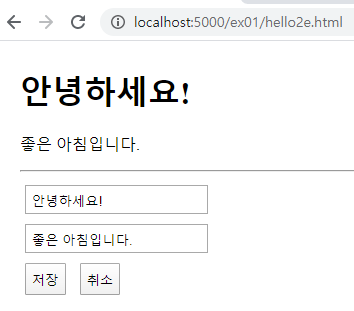
### hello2e.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-app.js"></script>    <script defer src="/\_\_/firebase/7.13.1/firebase-database.js"></script>    <script defer src="/\_\_/firebase/init.js"></script>    <style>      body { padding: 20px; }      input, button { padding: 5px; margin: 5px; }    </style>  </head>  <body>    <div id="app">      <h1>{{ title }}</h1>      <p>{{ body }}</p>      <button type="button" v-on:click="edit" v-show="!editing">수정</button>      <hr />      <div v-show="editing">        <input type="text" v-model="title\_edit" /> <br />        <input type="text" v-model="body\_edit" /> <br />        <button type="button" v-on:click="save">저장</button>        <button type="button" v-on:click="cancel">취소</button>      </div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: 'hello world',          body: 'good morning',          title\_edit: '',          body\_edit: '',          editing: false        },        methods: {          edit: function() {            this.title\_edit = this.title;            this.body\_edit = this.body;            this.editing = true;          },          save: function () {            let helloRef = firebase.database().ref("hello");            helloRef.set({ title: this.title\_edit, body: this.body\_edit });            this.editing = false;          },          cancel: function() {            this.editing = false;          }        }      })      window.onload = function() {        let helloRef = firebase.database().ref("hello");        helloRef.on('value', (snapshot) => {          let hello = snapshot.val();          app.title = hello.title;          app.body = hello.body;        });      };     </script>  </body>  </html> |



첫 화면에서 input 태그들을 보이지 않고, 수정 버튼은 보인다.

수정 버튼을 클릭하면



input 태그들과 저장, 취소 버튼이 보이고, 수정 버튼은 보이지 않는다.

input 태그들에는 각각 앱의 title 속성과 body 값이 채워져 있다.

<button type="button" v-on:click="edit" v-show="!editing">수정</button>

editing 속성 값이 false 이면 수정 버튼은 보이고, input 태그들은 보이지 않는다.

editing 속성 값이 true 이면, 수정 버튼은 보이지 않고, input 태그들은 보인다.

edit: function() {

this.title\_edit = this.title;

  this.body\_edit = this.body;

  this.editing = true;

}

수정 버튼을 클릭하면 edit 메소드가 호출된다.

앱의 title 속성과 body 속성 값이 각각 title\_edit 속성과 body\_edit 속성에 대입된다.

<input type="text" v-model="title\_edit" /> <br />

<input type="text" v-model="body\_edit" /> <br />

수정 버튼을 클릭했을 때 화면에 보이는 input 태그들은, title\_edit 속성과 body\_edit 속성과 연결되어 있다.

<button type="button" v-on:click="cancel">취소</button>

취소 버튼을 클릭하면 cancel 메소드가 호출된다.

cancel: function() {

this.editing = false;

}

editing 속성 값이 false 이면 수정 버튼은 보이고, input 태그들은 보이지 않는다.

# toggle

## v-model, checkbox

### toggle1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=checkbox] { width: 20px; height: 20px; }    </style>  </head>  <body>    <div id="app">      <h1>        <input type="checkbox" v-model="visible" />        visible: {{ visible }}      </h1>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          visible: true        }      })     </script>  </body>  </html> |

#### v-model

data: {

visible: true

}

앱에 visible 속성을 생성한다. 초기 값은 true 이다.

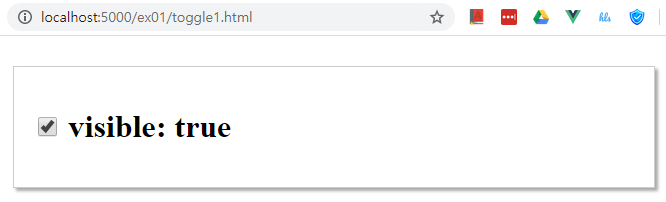
#### checkbox

<input type="checkbox" v-model="visible" />

이 체크박스의 상태와, 앱의 visible 속성을 양방향으로 연결한다.

앱의 visible 속성의 값이 바뀌면, 이 체크박스의 체크 상태도 실시간으로 바뀐다.

이 체크박스의 체크 상태가 바뀌면, 앱의 visible 속성값도 실시간으로 바뀐다.



#### CSS 서식

|  |
| --- |
| div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; } |

div#app

<div id="app> 태그에 서식을 적용한다.

padding: 20px;

태그 경계선 내부 여백을 지정함.

위 아래 왼쪽 오른쪽 여백 모두 20px

margin: 30px auto;

태그 경계선 외부 여백을 지정함.

위쪽과 아래쪽 여백 30px, 왼쪽과 오른쪽 여백 auto

왼쪽과 오른쪽 margin 값이 auto 이면, 부모 영역의 가운데 위치하게 된다.

<div id="app> 태그의 부모는 <body> 태그이다. 즉 웹브라우저 창이다.

max-width: 600px;

태그의 최대 폭이 600px 이다. 부모 영역에 여유가 있다면 태그의 폭은 600px 폭이 된다.

부모 폭에 여유가 없다면, 600px 보다 작아질 수 있다.

border: 1px solid #ccc;

태그의 경계선은 1px 굵기의 #ccc 색 실선

box-shadow: 3px 3px 3px #aaa;

태그의 위치보다 약간 오른쪽으로 3px, 아래쪽으로 3px 치우진 위치에,

3px 크기로 흩어진 #aaa 색 그림자를 그린다.

|  |
| --- |
| input[type=checkbox] { width: 20px; height: 20px; } |

input[type=checkbox]

<input type="checkbox" /> 태그에 서식을 적용한다.

width: 20px;

가로 방향 크기 20px

height: 20px;

세로 방향 크기 20px

## v-show

### toggle2.html

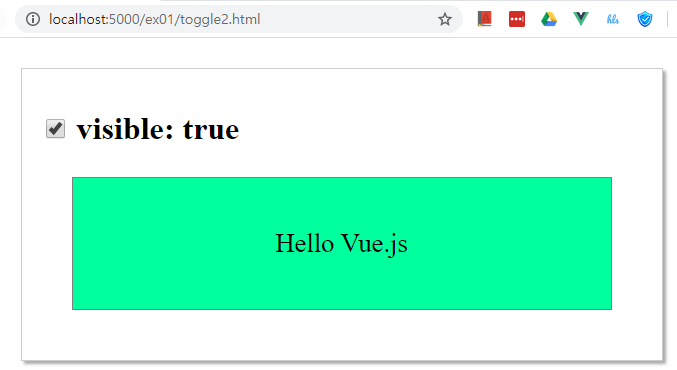
|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=checkbox] { width: 20px; height: 20px; }      div#hello { margin: 30px; padding: 50px;        background-color: #00ff9d; border: 1px solid gray;        text-align: center; font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <h1>        <input type="checkbox" v-model="visible" />        visible: {{ visible }}      </h1>      <div id="hello" v-show="visible">Hello Vue.js</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          visible: true        }      })     </script>  </body>  </html> |

#### v-show

<div id="hello" v-show="visible">Hello Vue.js</div>

v-show 속성의 값은 javascript 표현식으로 해석된다.

이 표현식의 값이 true 이면 이 태그가 화면에 보여지고, false 이면 보이지 않는다.



#### CSS 서식

|  |
| --- |
| div#hello { margin: 30px; padding: 50px;  background-color: #00ff9d; border: 1px solid gray;    text-align: center; font-size: 20pt; } |

div#hello

<div id="hello"> 태그에 서식이 적용된다.

margin: 30px;

경계석 외부 여백을 지정함.

위 아래 왼쪽 오랜쪽 여백이 모두 30px

padding: 50px;

경계석 내부 여백을 지정함.

위 아래 왼쪽 오랜쪽 여백이 모두 30px

background-color: #00ff9d;

태그 내부 배경색을 지정함

border: 1px solid gray;

경계선은 1px 굵기의 gray색 실선

text-align: center;

태그 내부 내용은 가로 방향으로 중앙 정렬.

세로 방향 정렬과는 무관함.

font-size: 20pt;

태그 내부 글자 크기는 20pt

## v-if

### toggle3.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      input[type=checkbox] { width: 20px; height: 20px; }      div#hello { margin: 30px; padding: 50px;        background-color: #00ff9d; border: 1px solid gray;        text-align: center; font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <h1>        <input type="checkbox" v-model="visible" />        visible: {{ visible }}      </h1>      <div id="hello" v-if="visible">Hello Vue.js</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          visible: true        }      })     </script>  </body>  </html> |

#### v-if

<div id="hello" v-if="visible">Hello Vue.js</div>

v-if 속성의 값은 javascript 표현식으로 해석된다.

이 표현식의 값이 true 이면 이 태그가 존재하고, false 이면 없어진다.

#### v-show="false" 과 v-if="false" 의 차이는

v-show="false"는 태그가 존재하지만 보이지 않는 상태

v-if="false"는 태그가 존재하지 않는 상태

v-show 속성의 값이 바뀌면, 태그의 보이는 상태만 변화한다.

v-if 속성의 값이 바뀌면, 태그가 만들어졌다 없어졌다 해야 한다.

아주 가끔 화면에 나타나는 에러 메시지를 표시하기 위한 태그의 경우,

대부분의 경우 필요 없는 그 태그가 메모리를 차지하지 않는 편이 좋으니까

v-if 로 구현하는 것이 좋다.

v-if 값이 false 이면, 태그가 존재하지 않는다.

보이는 상태가 자주 바뀌는 태그의 경우는

v-if 로 구현하면, 태그를 만들고 제거하는 작업을 자주 반복해야 하니 바람직하지 않고,

보이는 상태만 바꾸는 v-show 가 바람직하다.

## radio 버튼

### toggle4.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      label { font-size: 20pt; }      input[type=radio] { width: 20px; height: 20px; }      div#hello { margin: 30px; padding: 50px;        background-color: #00ff9d; border: 1px solid gray;        text-align: center; font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <label>        <input type="radio" value="true" v-model="visible" />        show      </label>      <label>        <input type="radio" value="false" v-model="visible" />        hide      </label>      <h1>visible: {{ visible }}</h1>      <h2>typeof visible: {{ typeof visible }}</h2>      <div id="hello" v-show="visible">Hello Vue.js</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          visible: true        }      })     </script>  </body>  </html> |

<input type="radio" value="true" v-model="visible" />



위와 같은 radio 버튼을 출력하는 태그이다.

<label>

<input type="radio" value="true" v-model="visible" />

  show

</label>

radio 버튼이나 checkbox를 위와같이 <label> 태그로 묶어 주는 것이 바람직하다.

label 태그로 묶은 경우에는, label 태그 내부 어디를 클릭해도, 체크 상태가 변하지만,

묶지 않은 경우에는, 정확하게 radio 버튼을 클릭해야 한다.

<input type="radio" value="true" v-model="visible" />

이 radio 버튼을 체크하면, 이 태그의 value 속성값이 앱의 visible 속성에 대입된다.

즉 "true" 값이 대입된다.

<input type="radio" value="false" v-model="visible" />

이 radio 버튼을 체크하면, 이 태그의 value 속성값이 앱의 visible 속성에 대입된다.

즉 "false" 값이 대입된다.

value="true" 속성과 value="false" 속성은 평범한 html 속성(attribute)이다.

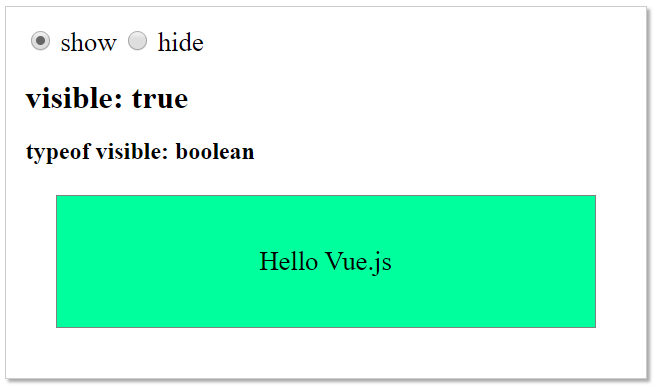
Vue.js 속성이 아니다.

평범한 html 속성의 값은 문자열(string)으로 해석된다.

따라서 위 radio 버튼을 체크할 때, 앱의 visiable 속석에 대입되는 값은

"true" 문자열이거나 "false" 문자열이다.

#### boolean 타입



data: {

visible: true

}

앱이 처음 시작되었을 때, 앱의 visible 속성값은 true 이다. 이 값의 타입은 boolean 이다.

<h2>typeof visible: {{ typeof visible }}</h2>

따라서 앱을 처음에 실행했을 때, 이 부분에 출력되는 값은 boolean 이다.

#### string 타입

<input type="radio" value="false" v-model="visible" />

이 버튼을 클릭하면, 앱의 visible 속성에 대입되는 값은 "false" 문자열이다.

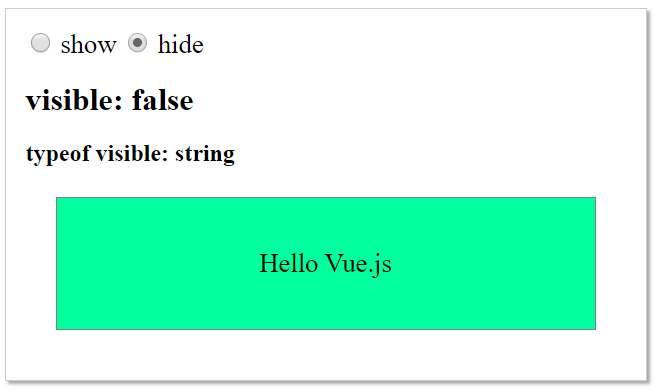
<div id="hello" v-show="visible">Hello Vue.js</div>

visible 속성의 값이 "false" 문자열이면, 이 값은 true로 취급된다.

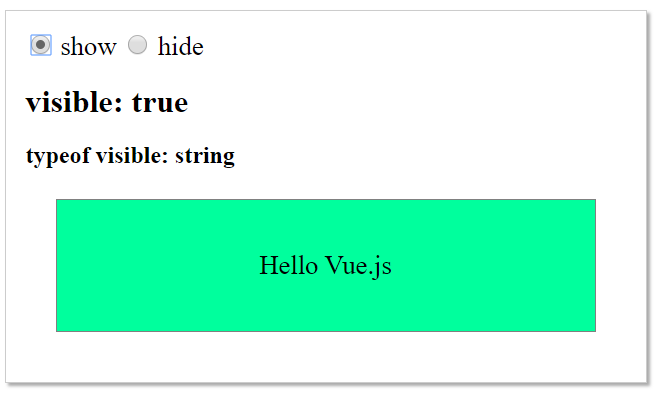
javascript에서 문자열의 값이 null 이거나 "" 인 경우에만 false로 취급되고,

그 외의 문자열은 true로 취급된다.

따라서 visible 속성 값이 "false" 일 때, 이 태그는 화면에 보여진다.



visible 속성의 값이 "false" 문자열인 상태이다.



visible 속성의 값이 "true" 문자열인 상태이다.

### toggle5.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      label { font-size: 20pt; }      input[type=radio] { width: 20px; height: 20px; }      div#hello { margin: 30px; padding: 50px;        background-color: #00ff9d; border: 1px solid gray;        text-align: center; font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <label>        <input type="radio" value="true" v-model="visible" />        show      </label>      <label>        <input type="radio" value="false" v-model="visible" />        hide      </label>      <h1>visible: {{ visible }}</h1>      <h2>typeof visible: {{ typeof visible }}</h2>      <div id="hello" v-show="visible == 'true'">Hello Vue.js</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          visible: true        }      })     </script>  </body>  </html> |

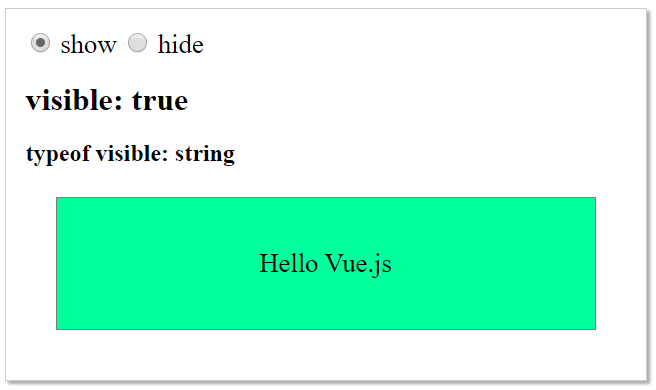
v-show="visible == 'true'"

v-show 속성은 Vue.js 속성이므로 이 속성값은 javascript 표현식으로 해석된다.

visible == 'true'

앱의 visible 속성 값이 'true' 문자열이면, 이 표현식의 값은 true 이다.

앱의 visible 속성 값이 'true' 문자열이 아니면, 이 표현식의 값은 false 이다.





## v-bind

### toggle6.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 20px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }      label { font-size: 20pt; }      input[type=radio] { width: 20px; height: 20px; }      div#hello { margin: 30px; padding: 50px;        background-color: #00ff9d; border: 1px solid gray;        text-align: center; font-size: 20pt; }    </style>  </head>  <body>    <div id="app">      <label>        <input type="radio" v-bind:value="true" v-model="visible" />        show      </label>      <label>        <input type="radio" v-bind:value="false" v-model="visible" />        hide      </label>      <h1>visible: {{ visible }}</h1>      <h2>typeof visible: {{ typeof visible }}</h2>      <div id="hello" v-show="visible">Hello Vue.js</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          visible: true        }      })     </script>  </body>  </html> |

#### v-bind

<input type="radio" v-bind:value="true" v-model="visible" />

html 속성 앞에 v-bind: 접두어를 붙이면, 그 속성의 값은 javascript 표현식으로 해석된다.

v-bind:value="true"

이 속성값은 true 이다. 이 값의 타입은 boolean 이다.

v-bind:value="'true'"

이 속성값은 'true' 이다. 이 값의 타입은 string 이다.

v-bind:value='"true"'

이 속성값은 "true" 이다. 이 값의 타입은 string 이다.

v-bind:value="false"

이 속성값은 false 이다. 이 값의 타입은 boolean 이다.

value="true"

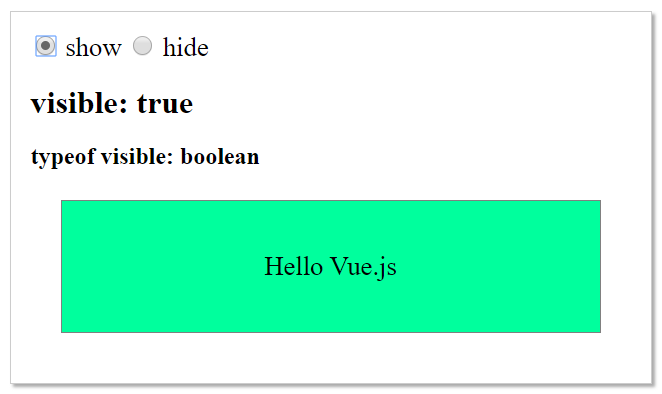
이 속성값은 "true" 이다. 이 값의 타입은 string 이다.

<input type="radio" v-bind:value="true" v-model="visible" />

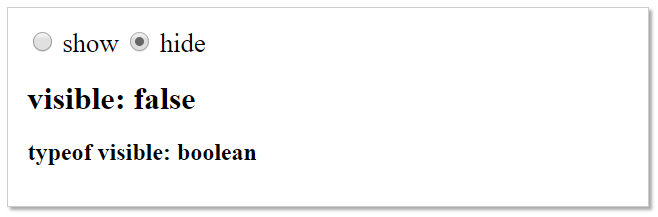
이 radio 버튼이 체크되면, 앱의 visible 속성에 true 값이 대입된다. (boolean 타입)

<input type="radio" v-bind:value="false" v-model="visible" />

이 radio 버튼이 체크되면, 앱의 visible 속성에 false 값이 대입된다. (boolean 타입)



visible 속성 값이 true 인 상태 (boolean 타입)



visible 속성 값이 false 인 상태 (boolean 타입)

# 과제

## empty1.html

|  |
| --- |
| <html>  <head>    <meta charset="utf-8" />    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>    <style>      div#app { padding: 50px; margin: 30px auto; max-width: 600px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; font-size: 11pt; }      input { padding: 6px; width: 500px; }      div.error { background-color: #fcc; border: 1px solid red;        padding: 10px; max-width: 500; margin-top: 30px; }    </style>  </head>  <body>    <div id="app">      제목:      <input type="text" v-model="title" />      <div class="error">제목 입력은 필수입니다.</div>    </div>    <script type="text/javascript">      var app = new Vue({        el: '#app',        data: {          title: ''        }      })     </script>  </body>  </html> |



제목을 입력 하지 않았을 때만, 에러 메시지가 보이고,

제목을 입력하면 에러 메시지가 보이지 않도록 소스코드를 수정하시오.

## empty2.html



성별을 선택하지 않은 화면

radio 버튼과 연결(v-model)된 속성의 값이 null 이면, 위와 같이 아무것도 선택되지 않은 상태이다.



성별을 선택한 화면

성별을 선택하면 에러 메시지가 보이지 않는다.

위와 같이 실행되도록 구현하시오.

(안 해도 되지만) 화면 형태도 위 그림과 비슷하도록 구현해 보라.