Vue 강의

이병일

라우팅이란?

- 라우팅(Routing)이란 웹 페이 간의 이동 방법을 말함

- 라우팅은 현대 웹 앱 형태 중 하나인 싱글 페이지 애플리케이션(SPA, Single Page Application)에서 주로 사용함

- SPA란?
  페이지를 이동할 때마다 서버에 웹 페이지를 요청하여 새로 갱신하는 것이 아니라 미리 해당 페이지들을 받아 놓고 페이지 이동 시에 클라이언트의 라우팅을 이용하여 화면을 갱신하는 패턴을 적용한 것

- 라우팅을 이용하면 서버에서 응답을 기다리는 동안 깜빡거림 현상 없이 화면을 매끄럽게 전환할 수 있고, 더 빠르게 화면을 조작할 수 있어 사용자 경험이 향상 됨

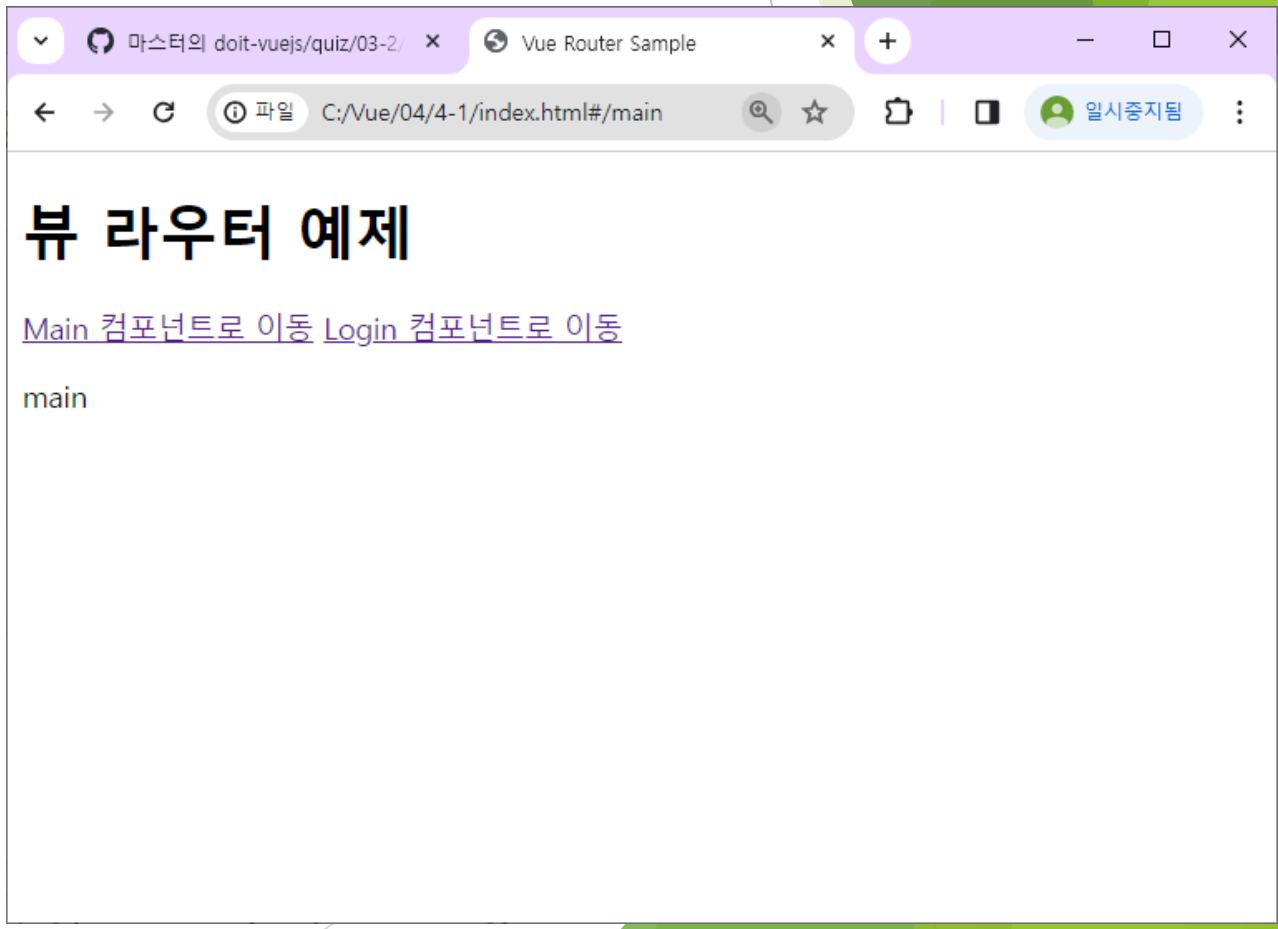
뷰 라우터

- 뷰 라우터는 뷰에서 라우팅 기능을 구현할 수 있도록 지원하는 공식 라이브러리임
- 뷰 라우터를 이용하여 뷰로 만든 페이지 간에 자유롭게 이동할 수 있음
- 뷰 라우터를 구현할 때 필요한 특수 태그와 기능

태 그	설 명
<router-link to="URL 값">	페이지 이동 태그. 화면에서 <a>로 표시되며 클릭하면 to 지정한 URL로 이동함
<router-view>	페이지 표시 태그. 변경되는 URL에 따라 해당 컴포넌트를 뿌려주는 영역

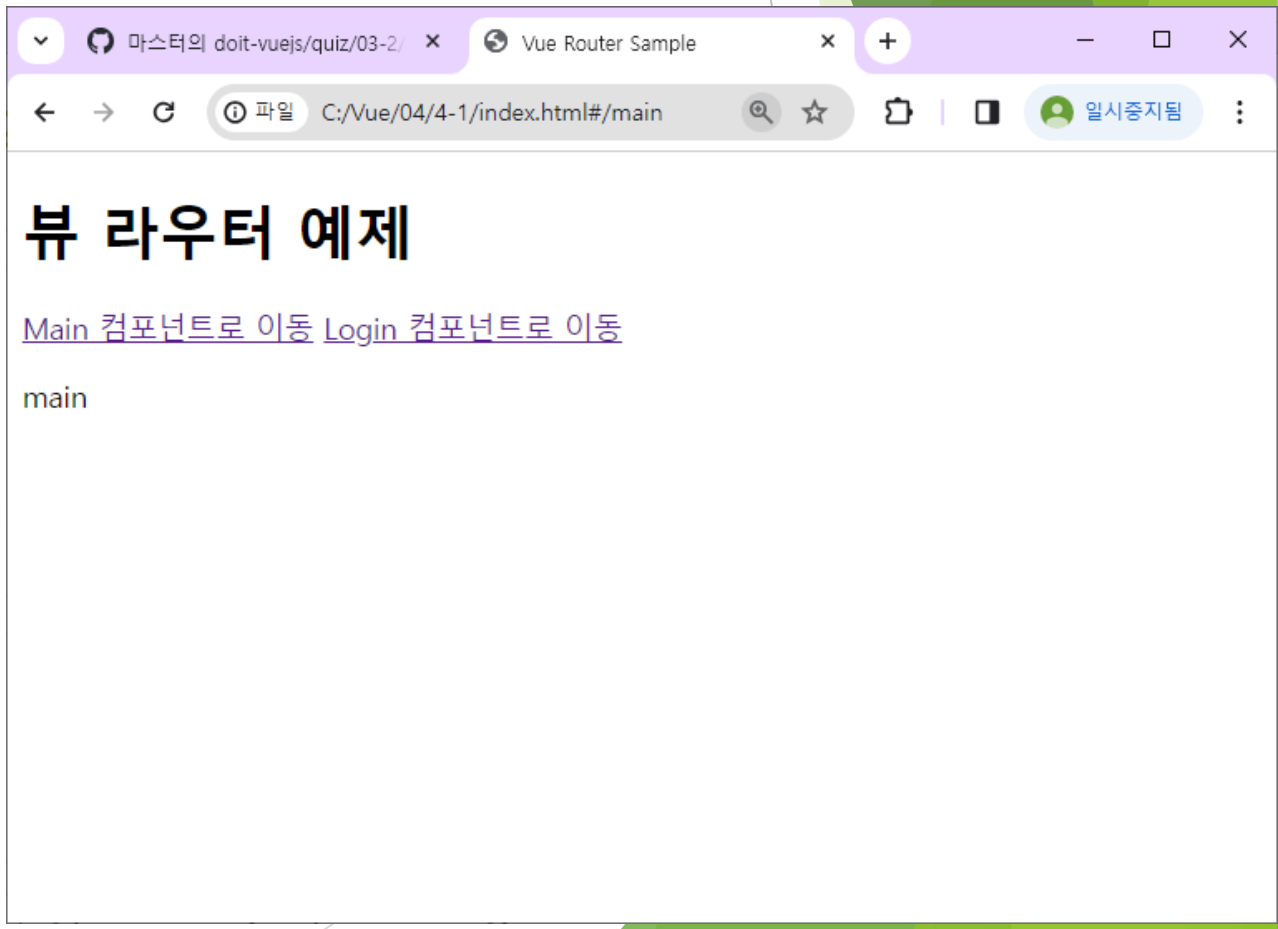
뷰 라우터 실습 예제

```
8   <body>
9     <div id="app">
10      <h1>뷰 라우터 예제</h1>
11      <p>
12        <router-link to="/main">Main 컴포넌트로 이동</router-link>
13        <router-link to="/login">Login 컴포넌트로 이동</router-link>
14      </p>
15      <router-view></router-view>
16    </div>
17
18    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
19    <script src="https://unpkg.com/vue-router@3.0.1/dist/vue-router.js"></script>
20    <script>
21      // 3. Main. Login 컴포넌트 내용 정의
22      var Main = { template: '<div>main</div>' };
23      var Login = { template: '<div>login</div>' };
24
25      // 4. 각 url에 해당하는 컴포넌트 등록
26      var routes = [
27        { path: '/main', component: Main },
28        { path: '/login', component: Login }
29      ];
30
31      // 5. 뷰 라우터 정의
32      var router = new VueRouter({
33        routes
34      });
35
36      // 6. 뷰 라우터를 인스턴스에 등록
37      var app = new Vue({
38        router
39      }).$mount('#app');
40    </script>
41  </body>
42  </html>
```



뷰 라우터 실습 예제

```
8   <body>
9     <div id="app">
10      <h1>뷰 라우터 예제</h1>
11      <p>
12        <router-link to="/main">Main 컴포넌트로 이동</router-link>
13        <router-link to="/login">Login 컴포넌트로 이동</router-link>
14      </p>
15      <router-view></router-view>
16    </div>
17
18    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
19    <script src="https://unpkg.com/vue-router@3.0.1/dist/vue-router.js"></script>
20    <script>
21      // 3. Main. Login 컴포넌트 내용 정의
22      var Main = { template: '<div>main</div>' };
23      var Login = { template: '<div>login</div>' };
24
25      // 4. 각 url에 해당하는 컴포넌트 등록
26      var routes = [
27        { path: '/main', component: Main },
28        { path: '/login', component: Login }
29      ];
30
31      // 5. 뷰 라우터 정의
32      var router = new VueRouter({
33        routes
34      });
35
36      // 6. 뷰 라우터를 인스턴스에 등록
37      var app = new Vue({
38        router
39      }).$mount('#app');
40    </script>
41  </body>
42  </html>
```

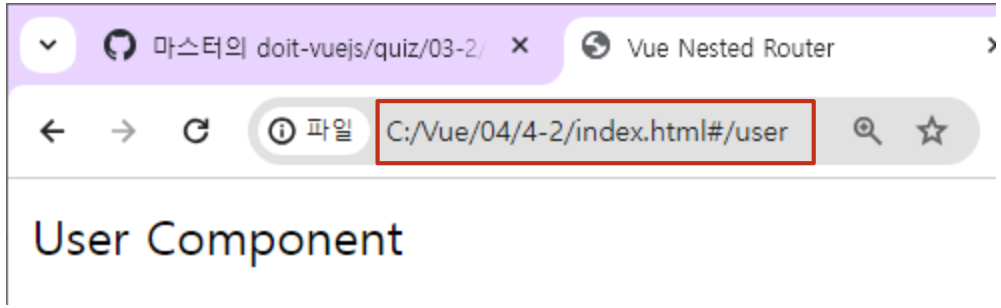


네스티드 라우터

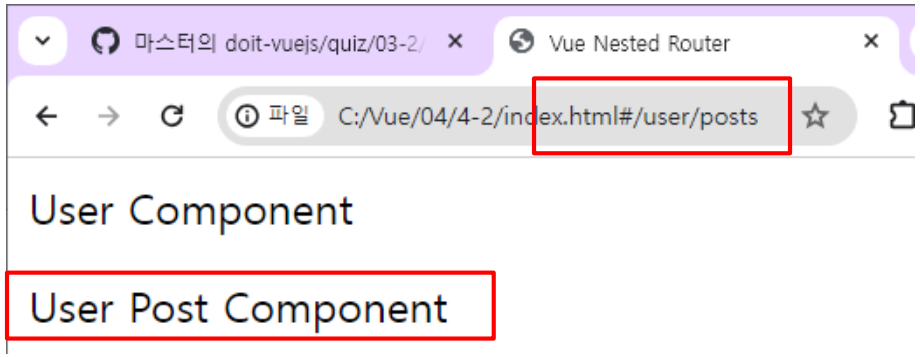
```
<body>
  <div id="app">
    <router-view></router-view>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
  <script src="https://unpkg.com/vue-router@3.0.1/dist/vue-router.js"></script>
  <script>
    var User = {
      template: `
        <div>
          User Component
          <router-view></router-view>
        </div>
      `
    };
    var UserProfile = { template: '<p>User Profile Component</p>' };
    var UserPost = { template: '<p>User Post Component</p>' };
    var routes = [
      {
        path: '/user',
        component: User,
        children: [
          {
            path: 'posts',
            component: UserPost
          },
          {
            path: 'profile',
            component: UserProfile
          },
        ]
      }
    ];
    var router = new VueRouter({
      routes
    });
    var app = new Vue({
      router
    }).$mount('#app');
  </script>
</body>
```

- 네스티드 라우터(Nested Router)는 라우터로 페이지를 이동할 때 최소 2개 이상의 컴포넌트를 화면에 나타날 수 있음
- 네스티드 라우터를 이용하면 URL에 따라 컴포넌트의 하위 컴포넌트가 다르게 표시됨

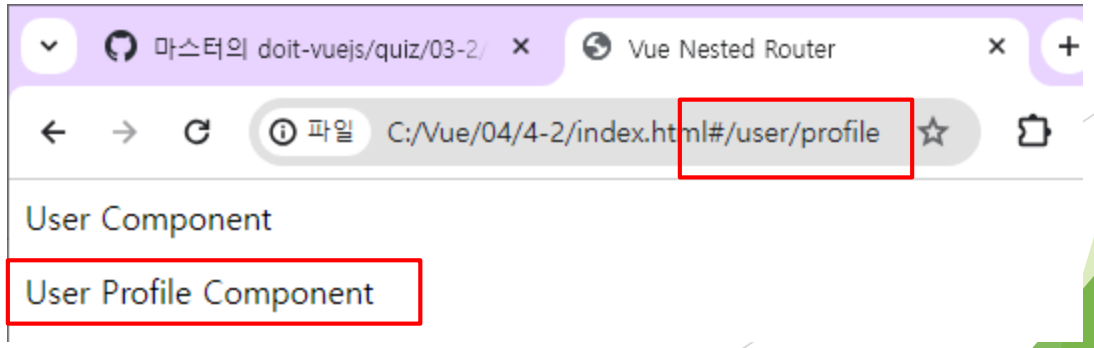
- 코드를 실행하면 첫 화면에는 아무것도 보이지 않음
- 브라우저를 실행하고 URL 값의 끝에 user를 입력



- URL 값의 끝에 '/posts'를 추가



- URL 값의 끝에 'profile'를 추가

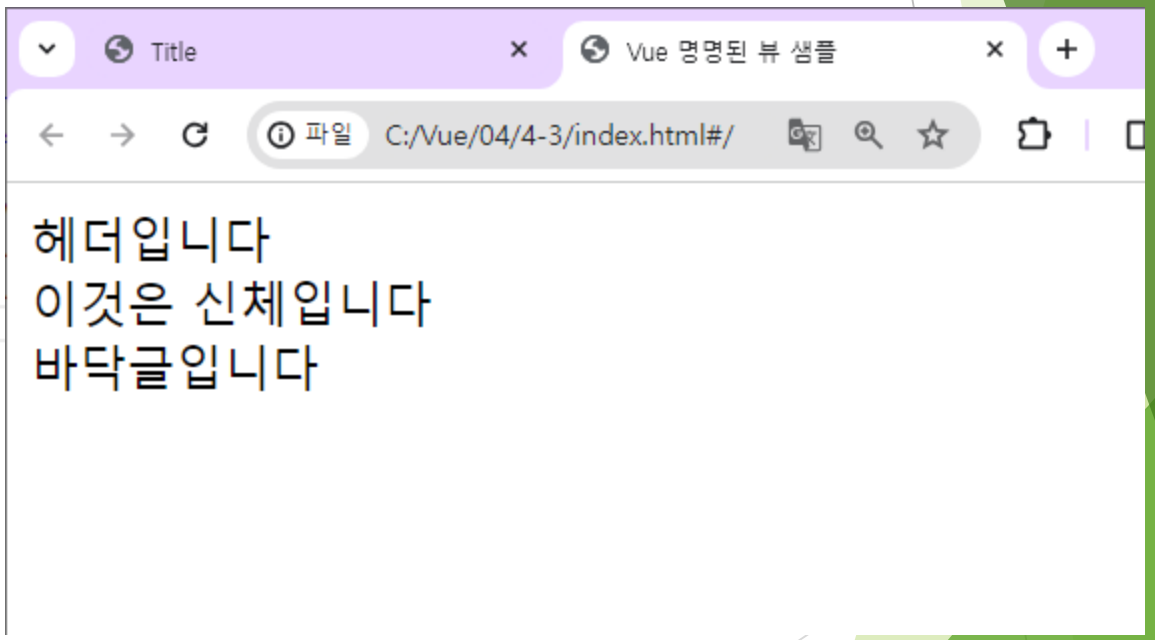


네임드 뷰(Named View)

- 네스티드 라우터는 화면을 구성하는 컴포넌트 수가 적을 때는 유용하지만 한 번에 더 많은 컴포넌트를 표시하는 데는 한계가 있음
- 이 문제를 해결할 수 있는 방안으로 네임드 뷰가 있음
- 네임드 뷰는 특정 페이지로 이동했을 때 여러 개의 컴포넌트를 동시에 표시하는 라우팅 방식
- 네임드 뷰는 같은 레벨에서 여러 개의 컴포넌트를 한 번에 표시함

네임드 뷰 구현 예제

```
8   <body>
9     <div id="app">
10      <router-view name="header"></router-view>
11      <router-view></router-view>
12      <router-view name="footer"></router-view>
13    </div>
14
15    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
16    <script src="https://unpkg.com/vue-router@3.0.1/dist/vue-router.js"></script>
17    <script>
18      var Body = { template: '<div>This is Body</div>' };
19      var Header = { template: '<div>This is Header</div>' };
20      var Footer = { template: '<div>This is Footer</div>' };
21
22      var router = new VueRouter({
23        routes: [
24          {
25            path: '/',
26            components: {
27              default: Body,
28              header: Header,
29              footer: Footer
30            }
31          }
32        ]
33      })
34
35      var app = new Vue({
36        router
37      }).$mount('#app');
38    </script>
39  </body>
40  </html>
```



이렇게 네임드 뷰를 활용하면 특정 페이지로 이동했을 때
해당 **URL**에 맞추어 여러 개의 컴포넌트를 한 번에 표시할 수 있음

뷰 HTTP 통신

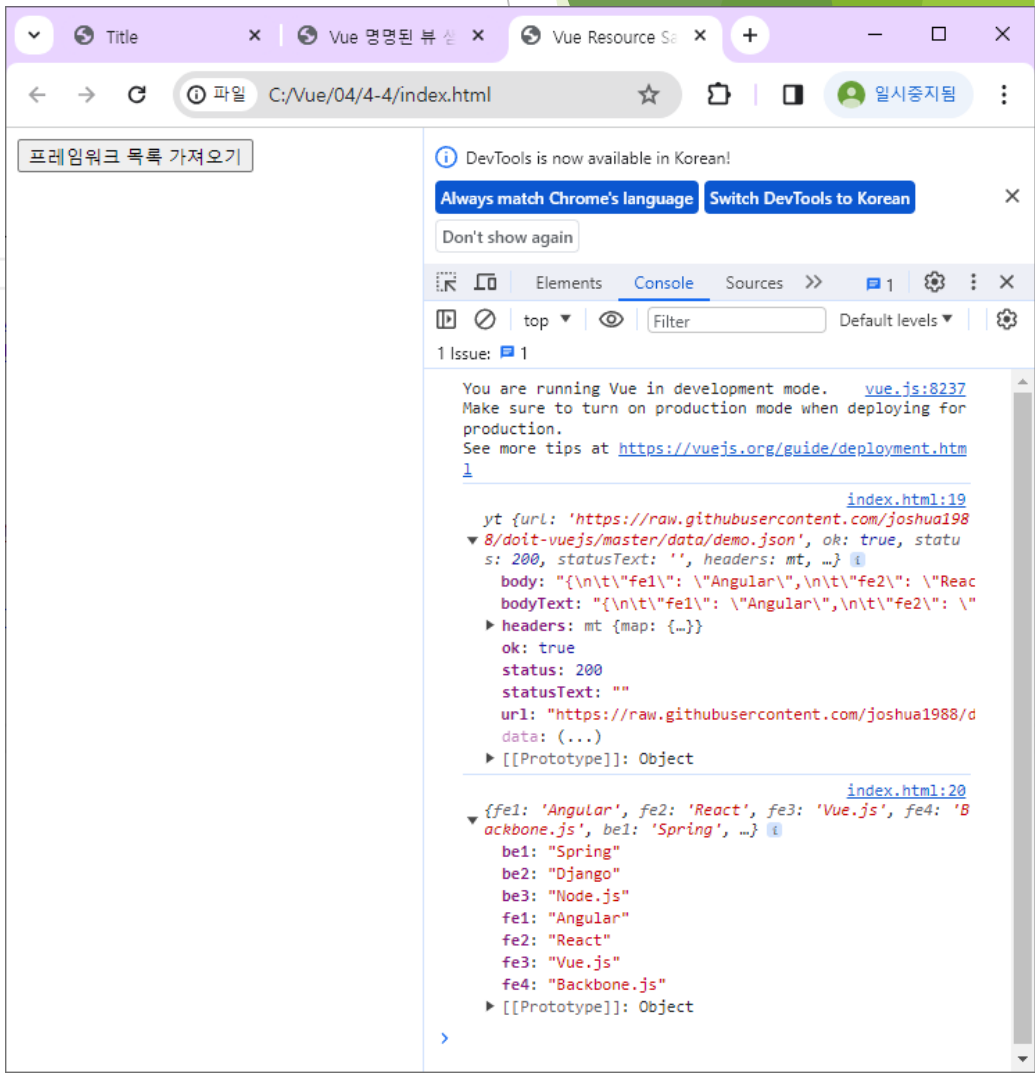
- 브라우저에서 특정 데이터를 보내달라고 요청(request)를 보내면 서버에서 응답(response)으로 해당 데이터를 보내주는 방식
- Ajax는 서버에서 받아온 데이터를 표시할 때 화면 전체를 갱신하지 않고도 화면의 일부분만 변경할 수 있게 하는 자바스크립트 기법
- 뷰도 마찬가지로 ajax 를 지원하기 위한 라이브러리를 제공
- 뷰 프레임워크의 필수 라이브러리로 관리하던 뷰 리소스와 요즘 가장 많이 사용하는 엑시오스(axios)

뷰 리소스

- 뷰 리소스는 초기에 코어 팀에서 공식적으로 권하는 라이브러리였으나 2016년 말에 에반이 공식적인 지원을 중단하기로 결정하면서 다시 기존에 관리했던 PageKit 팀의 라이브러리로 돌아갔음
- 버튼 하나를 추가하고 클릭하면 지정한 URL의 데이터를 가져오는 예제

4-4 > index.html > html > body > div#app

```
<html>
  <head>
    <title>Vue Resource Sample</title>
  </head>
  <body>
    <div id="app">
      <button v-on:click="getData">프레임워크 목록 가져오기</button>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/vue-resource@1.3.4"></script>
    <script>
      new Vue({
        el: '#app',
        methods: {
          getData: function() {
            this.$http.get(`https://raw.githubusercontent.com/joshua1988/doit-vuejs/master/data/demo.json`)
                .then(function(response) {
                  console.log(response);
                  console.log(JSON.parse(response.data));
                });
          }
        }
      });
    </script>
  </body>
</html>
```



엑시오스(Axios)

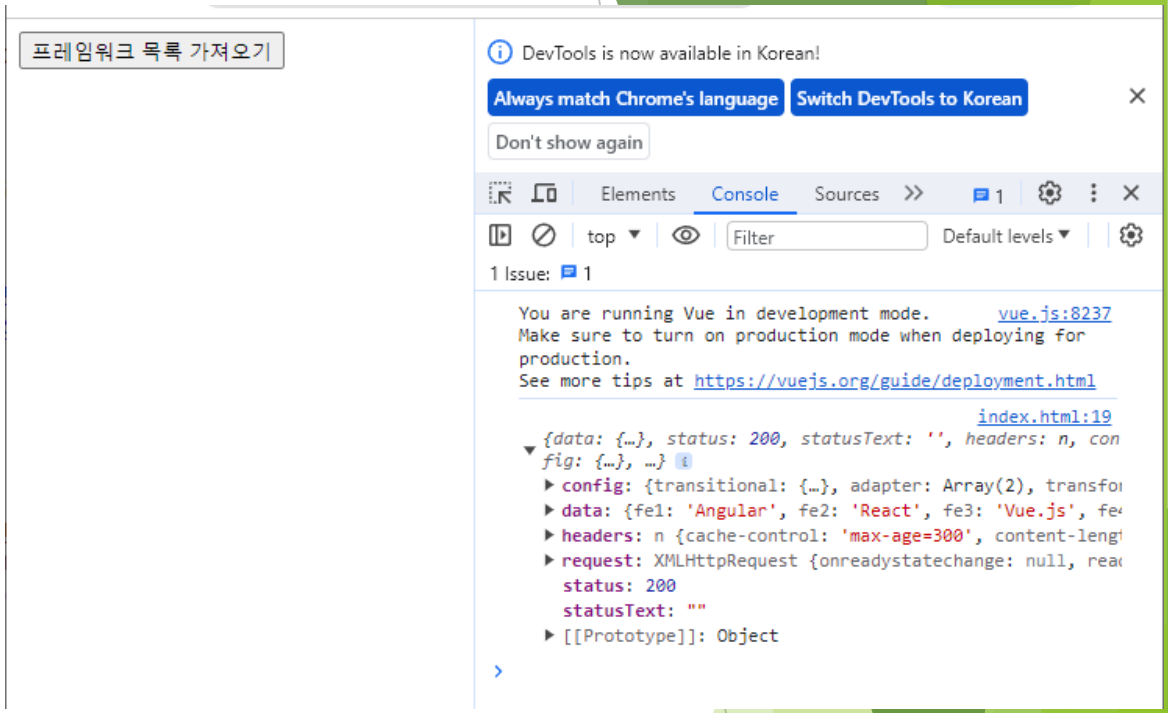
- 현재 뷰 커뮤니티에서 가장 많이 사용되는 HTTP 통신 라이브러리

```
// HTTP GET 요청
axios.get('URL 주소').then().catch();

//HTTP POST 요청
axios.post('URL 주소').then().catch();

//HTTP 요청에 대한 옵션 속성 정의
axios({
    method: 'get',
    url: 'url주소',
    …
});
```

엑시오스(Axios)로 데이터 받아오기 예제

- Response 객체를 확인해 보면 data 속성이 일반 문자열 형식이 아니라 객체 형태이기 때문에 별도로 JSON.parse()를 사용하여 객체로 변환할 필요가 없기 때문에 뷰 리소스 보다 사용이 편이함

```
<html>
    <head>
        <title>Vue with Axios Sample</title>
    </head>
    <body>
        <div id="app">
            <button v-on:click="getData">프레임워크 목록 가져오기</button>
        </div>

        <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
        <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
        <script>
            new Vue({
                el: '#app',
                methods: {
                    getData: function() {
                        axios.get('https://raw.githubusercontent.com/joshua1988/doit-vuejs/master/data/demo.json')
                            .then(function(response) {
                                console.log(response);
                            });
                    }
                }
            });
        </script>
    </body>
</html>
```



뷰 템플릿

- 뷰의 템플릿은 **HTML, CSS** 등의 마크업 속성과 뷰 인스턴스에서 정의한 데이터 및 로직들을 연결하여 사용자가 브라우저에서 볼 수 있는 형태의 **HTML**로 변환해 주는 속성임
- 템플릿에서 사용하는 뷰의 속성

  데이터 바인딩

  자바 스크립트 표현식

  디렉티브

  이벤트 처리

  고급 템플릿 기법

데이터 바인딩

- HTML 화면 요소를 뷰 인스턴스의 데이터와 연결하는 것을 의미함

- {{ }} - 콧수염 괄호
  - 뷰 인스턴스의 데이터를 HTML 태그에 연결하는 가장 기본적인 텍스트 삽입 방식

- 오른쪽 코드는 data 속성의 message 속성 값인 Hello Vue.JS!를 태그 안의 {{message}}에 연결하여 화면에 나타내는 코드로 만약 data 속성의 message 값이 바뀌면 뷰 반응성에 의해 화면이 자동의 갱신 됨

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

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

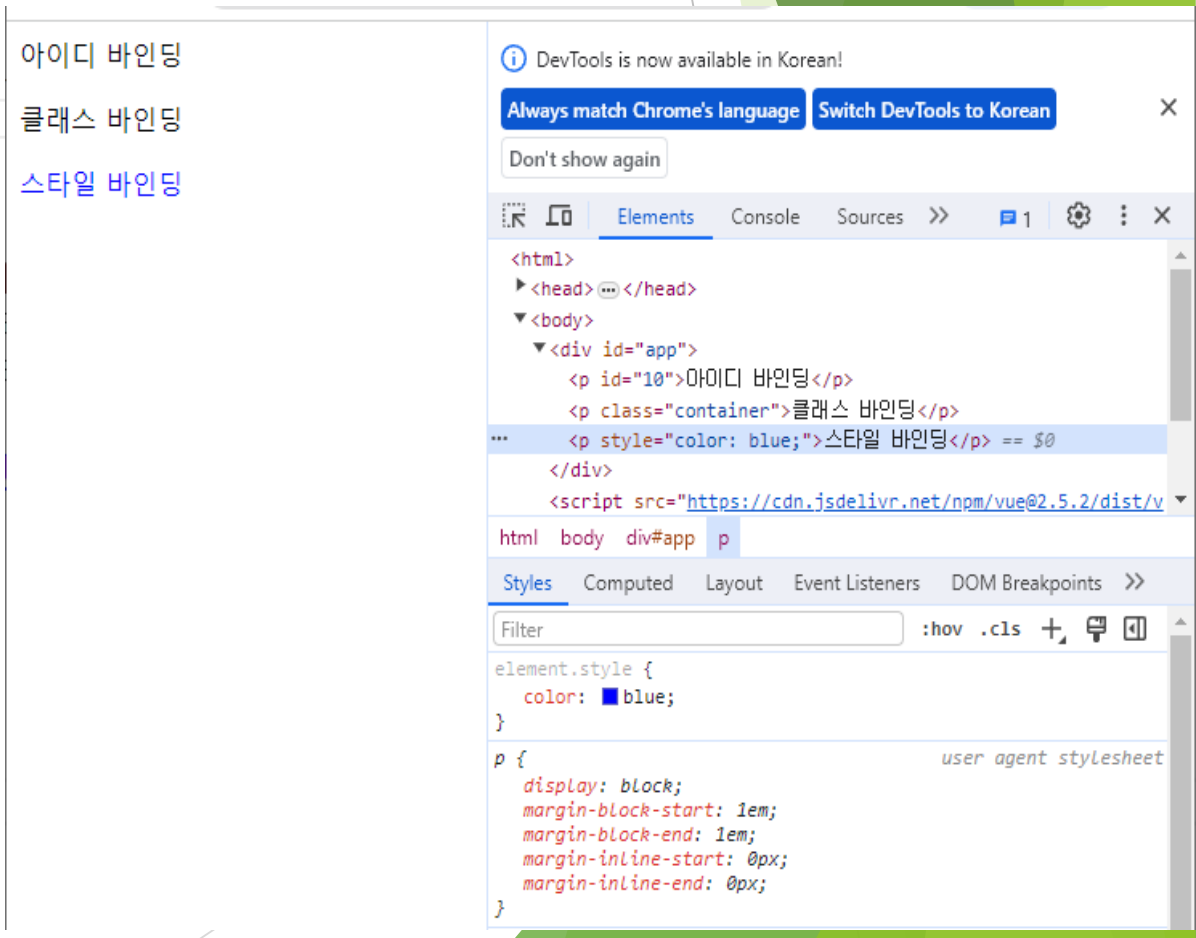
- 만약 뷰 테이처가 변경되어도 값을 바꾸고 싶지 않다면 오른쪽과 같이 v-once 속성을 사용함

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

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

v-bind

- 아이디, 클래스, 스타일 등의 HTML 속성 값에 뷰 데이터 값을 연결할 때 사용하는 데이터 연결 방식
- v-bind 속성으로 지정할 HTML 속성이나 props 속성 앞에 접두사로 붙여줌

```
1  <html>
2    <head>
3      <title>Vue Template - Data Binding</title>
4    </head>
5    <body>
6      <div id="app">
7        <p v-bind:id="idA">아이디 바인딩</p>
8        <p v-bind:class="classA">클래스 바인딩</p>
9        <p v-bind:style="styleA">스타일 바인딩</p>
10     </div>
11     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
12     <script>
13       new Vue({
14         el: '#app',
15         data: {
16           idA: 10,
17           classA: 'container',
18           styleA: 'color: blue'
19         }
20       });
21     </script>
22   </body>
23 </html>
```

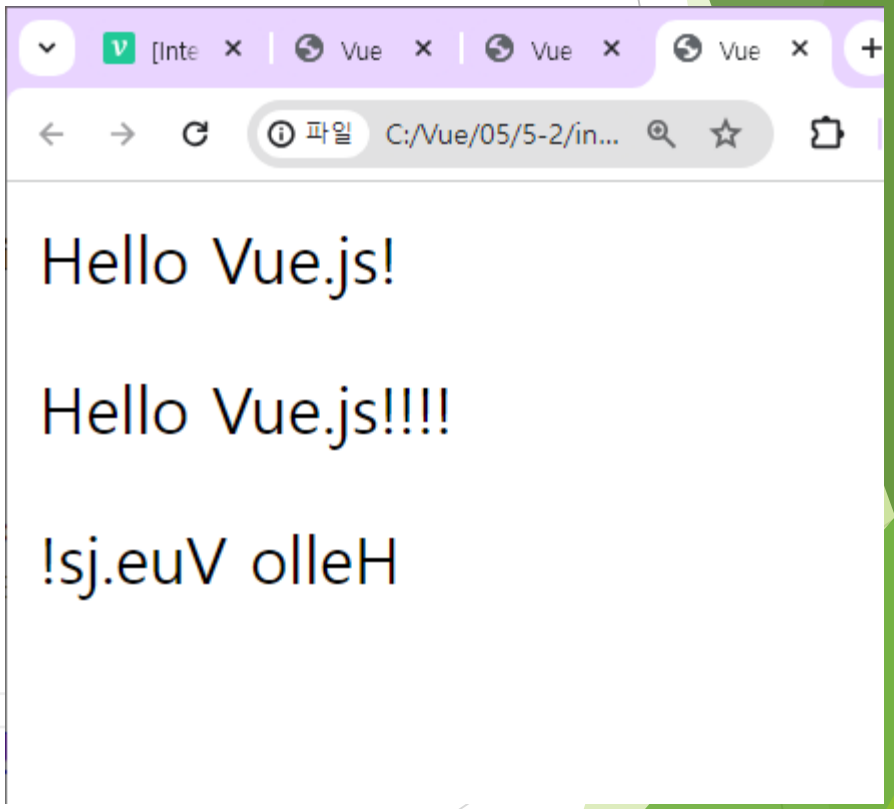


자바스크립트 표현식

- 데이터 바인딩 방법 중 하나인 {{ }} 안에 자바스크립트 표현식을 넣으면 됨

5-2 > index.html > html

```
1  <html>
2    <head>
3      <title>Vue Template - Javascript Expression</title>
4    </head>
5    <body>
6      <div id="app">
7        <p>{{ message }}</p>
8        <p>{{ message + "!!!" }}</p>
9        <p>{{ message.split('').reverse().join('') }}</p>
10     </div>
11
12     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
13     <script>
14       new Vue({
15         el: '#app',
16         data: {
17           message: 'Hello Vue.js!'
18         }
19       });
20     </script>
21   </body>
22 </html>
```



자바스크립트 표현식에서 주의할 점

- 첫째 : 자바 스트립트의 선언문과 분기 구문은 사용할 수 없음
- 둘째 : 복잡한 연산은 인스턴스 안에서 처리하고 화면에는 간단한 연산 결과만 표시해야 함

```
5   <body>
6     <div id="app">
7       <!-- 1. -->
8       {{ var a = 10; }} <!-- X, 선언문은 사용 불가능 -->
9       {{ if (true) {return 100} }} <!-- X, 분기 구문은 사용 불가능 -->
10      {{ true ? 100 : 0 }} <!-- O, 삼항 연산자로 표현 가능 -->
11
12      <!-- 2. -->
13      {{ message.split('').reverse().join('') }} <!-- X, 복잡한 연산은 인스턴스 안에서 수행 -->
14      {{ reversedMessage }} <!-- O, 스크립트에서 computed 속성으로 계산 후 최종 값만 표현 -->
15    </div>
16
17    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
18    <script>
19      new Vue({
20        el: '#app',
21        data: {
22          message: 'Hello Vue.js!'
23        },
24        computed: {
25          reversedMessage: function() {
26            return this.message.split('').reverse().join('');
27          }
28        }
29      });
30    </script>
31  </body>
32 </html>
```

디렉티브(Directive)

- HTML 안에 v-접두사를 가지는 모든 속성들을 의미함

  <a v-if="flag"> Vue.js

  위 <a> 태그는 뷰 인스턴스 데이터 속성에 정의된 flag 값에 따라 보이기도 하고 안보이기도 함
  flag 값이 참(true)이면 텍스트가 보이고, 값이 거짓(false)이면 보이지 않음

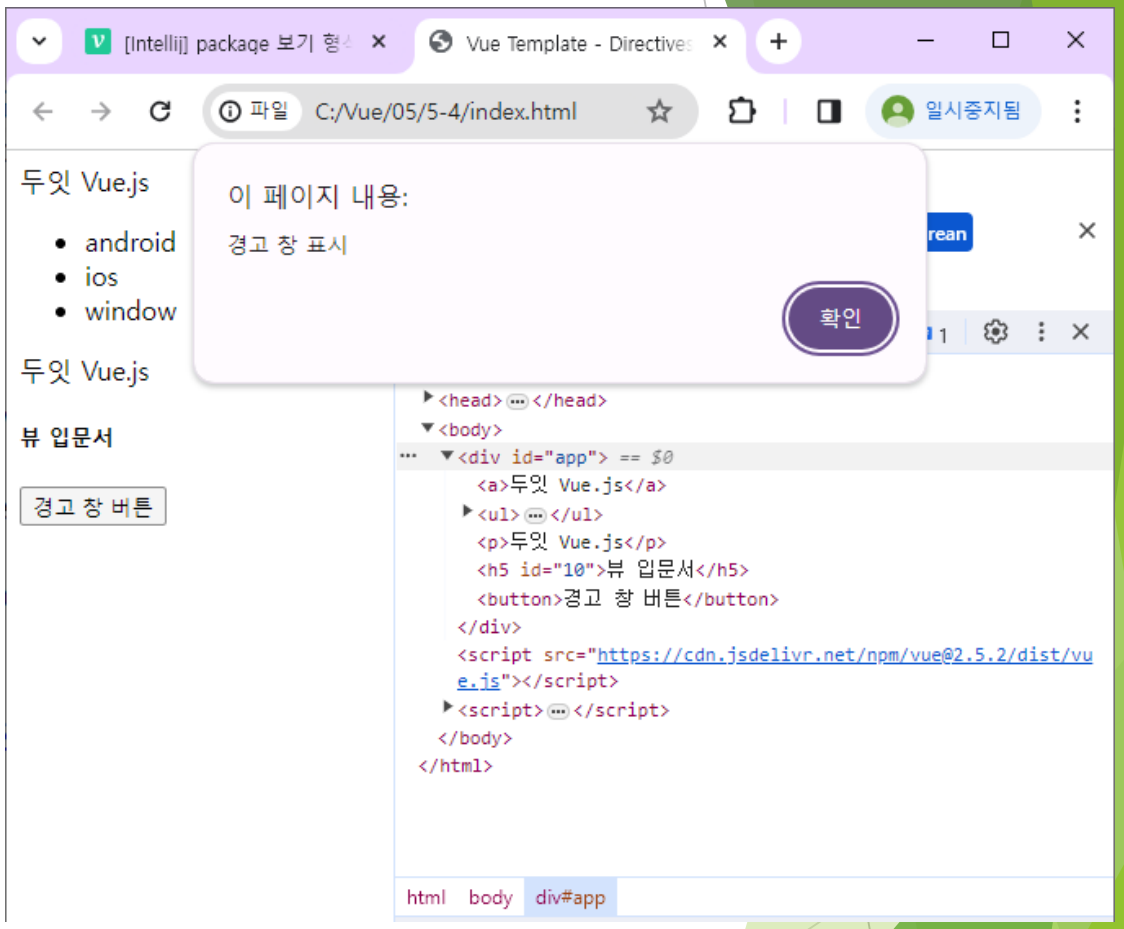
- 디렉티브는 화면의 요소를 더 쉽게 조작하기 위해 사용하는 기능

- 뷰의 데이터 값이 변경되었을 때 화면의 요소들이 리액티브(Reactive) 하게 반응하여 변경된 데이터 값에 따라 갱신됨

- 이런식으로 요소를 직접 제어할 필요 없이 뷰의 디렉티브를 활용하여 화면 요소들을 조작할 수 있음

많이 사용되는 디렉티브 예제

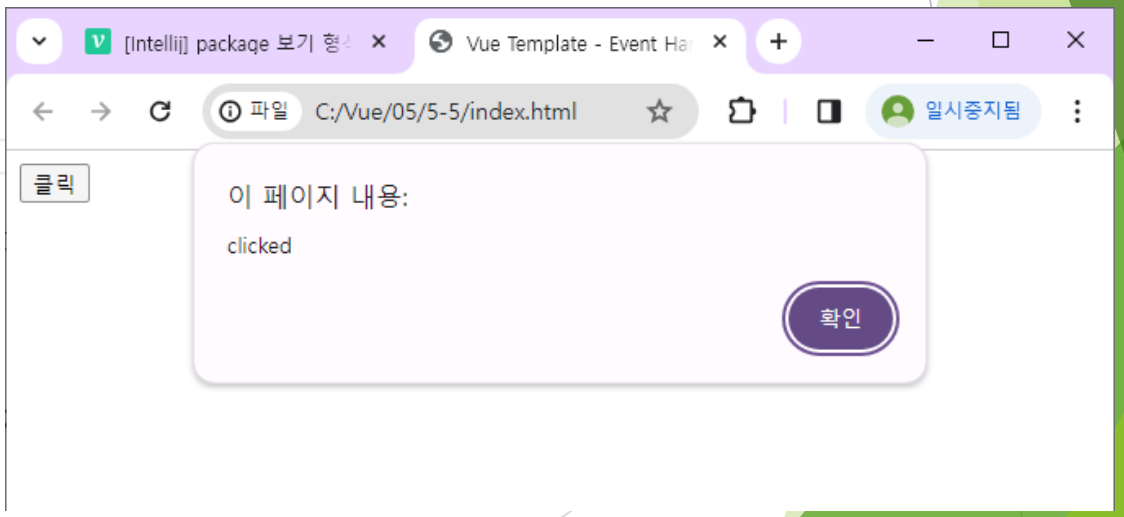
```
5   <body>
6     <div id="app">
7       <a v-if="flag">두잇 Vue.js</a>
8       <ul>
9         <li v-for="system in systems">{{ system }}</li>
10      </ul>
11      <p v-show="flag">두잇 Vue.js</p>
12      <h5 v-bind:id="uid">뷰 입문서</h5>
13      <button v-on:click="popupAlert">경고 창 버튼</button>
14    </div>
15
16    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
17    <script>
18      new Vue({
19        el: '#app',
20        data: {
21          flag: true,
22          systems: ['android', 'ios', 'window'],
23          uid: 10
24        },
25        methods: {
26          popupAlert: function() {
27            return alert('경고 창 표시');
28          }
29        }
30      });
31    </script>
32  </body>
33 </html>
```



이벤트 처리

- 뷰 역시 화면에서 발생하는 이벤트를 처리하기 위해 v-on 디렉티브와 methods 속성을 활용함

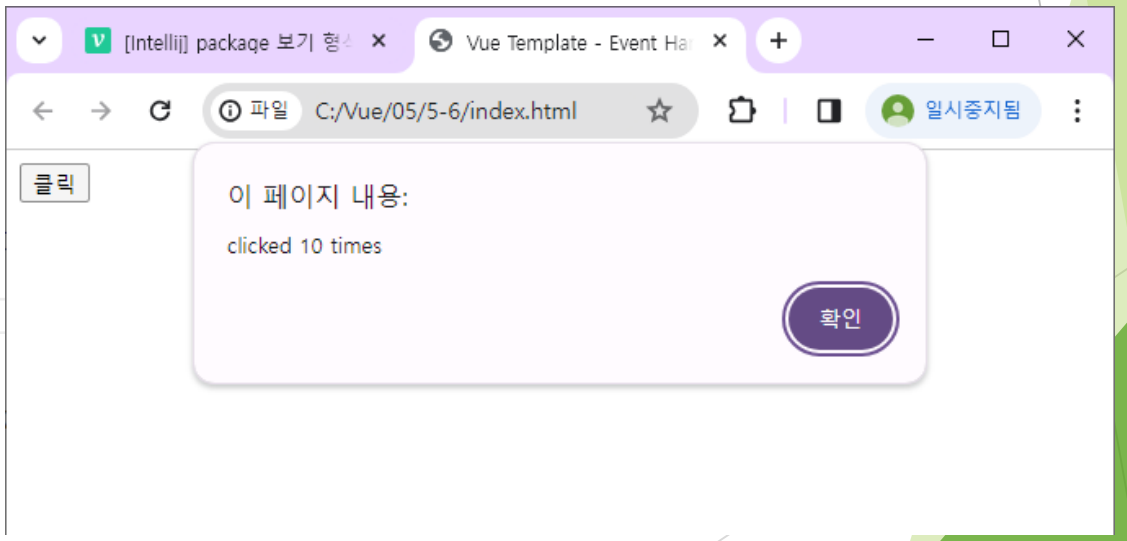
```
1  <html>
2    <head>
3      <title>Vue Template - Event Handling</title>
4    </head>
5    <body>
6      <div id="app">
7        <button v-on:click="clickBtn">클릭</button>
8      </div>
9
10     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
11     <script>
12       new Vue({
13         el: '#app',
14         methods: {
15           clickBtn: function() {
16             alert('clicked');
17           }
18         }
19       });
20     </script>
21   </body>
22 </html>
```



v-on 디렉티브로 메서드 호출할 때 인자 값 넘기기

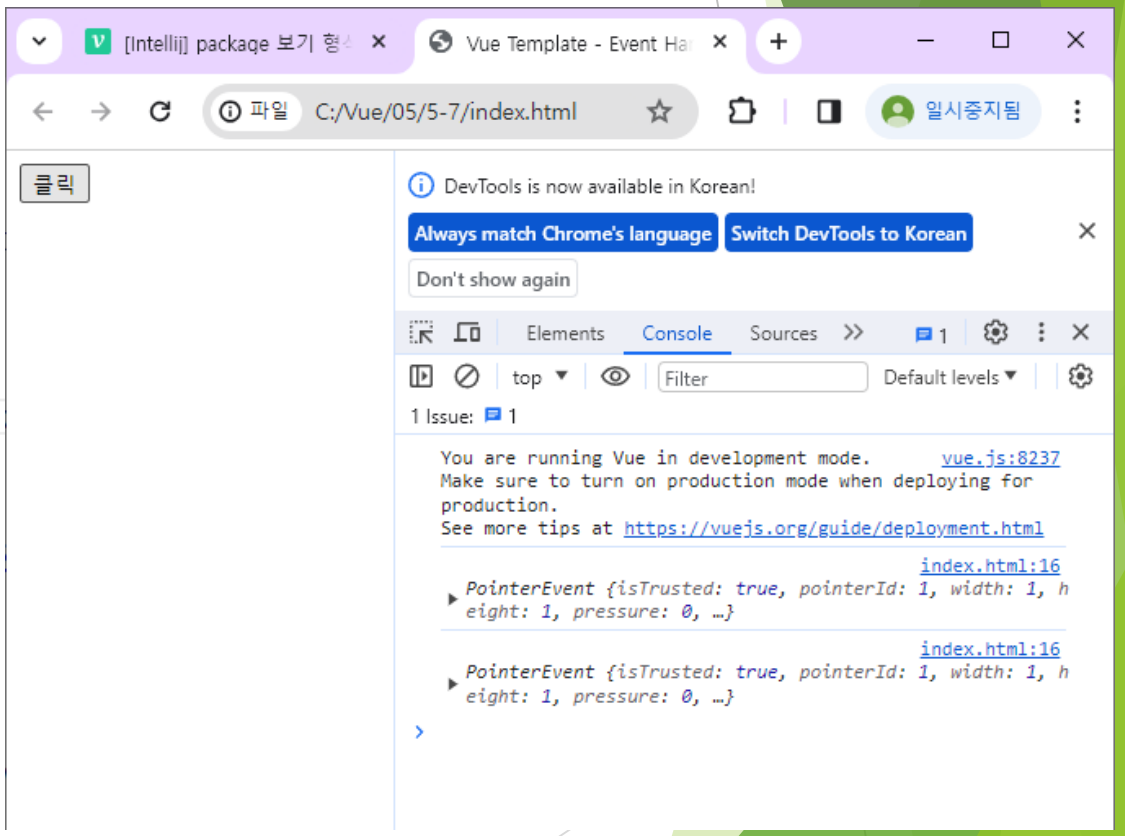
5-6 > index.html > html

```
1  <html>
2    <head>
3      <title>Vue Template - Event Handling</title>
4    </head>
5    <body>
6      <div id="app">
7        <button v-on:click="clickBtn(10)">클릭</button>
8      </div>
9
10     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
11     <script>
12       new Vue({
13         el: '#app',
14         methods: {
15           clickBtn: function(num) {
16             alert('clicked ' + num + ' times');
17           }
18         }
19       });
20     </script>
21   </body>
22 </html>
```



event 인자를 이용해 화면 요소의 돔 이벤트에 접근하는 예제

```html
<html>
  <head>
    <title>Vue Template - Event Handling</title>
  </head>
  <body>
    <div id="app">
      <button v-on:click="clickBtn">클릭</button>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
    <script>
      new Vue({
        el: '#app',
        methods: {
          clickBtn: function(event) {
            console.log(event);
          }
        }
      });
    </script>
  </body>
</html>
```



고급 템플릿 기법

- 고급 템플릿 기법은 실제 어플리케이션을 개발할 때 유용한 속성으로, 앞에서 배운 데이터 바인딩, 디렉티브와 같은 기본적인 문법과 함께 사용함
- computed 속성
  - 데이터를 가공하는 등의 복잡한 연산은 뷰 인스턴스 안에서 하고 최종적으로 HTML에는 데이터를 표현해야 하는데 이러한 데이터 연산들을 정의하는 영역

```
1  <div id="app">
2      <p>{{reversedMessage}}</p>
3  </div>
4  ...
5  <script>
6      new Vue({
7          el: '#app',
8          data: {
9              message: 'Hello Vue.js!'
10         },
11         computed: {
12             reversedMessage: function(){
13                 return this.message.split('').reverse().join('');
14             }
15         }
16     })
17 </script>
```

- HTML에 바로 {{ message.split(‘’).reverse().join(‘’)}}를 정의할 수도 있지만 위 코드처럼 computed를 이용하면 코드가 더 깔끔해 짐

고급 템플릿 기법

- computed 속성의 첫 번째 장점은 data 속성 값의 변화에 따라 자동으로 다시 연산한다는 점
- 예를 들어, computed 속성에서 사용하고 있는 data 속성 값이 변경되면 전체 값을 다시 한번 계산함

- 두번째 장점은 캐싱
- 캐싱은 동일한 연산을 반복해서 하지 않기 위해 연상의 결과 값을 미리 저장하고 있다가 필요할 때 불러오는 동작

- computed 속성과 methods 속성의 차이점
  methods 속성은 호출할 때만 해당 로직이 수행되고, computed 속성은 대상 데이터의 값이 변경되면 자동적으로 수행된다는 점

  다시말해 수동적으로 데이터를 갱신하느냐, 능동적으로 데이터를 갱신하느냐의 차이점

computed 속성과 methods 속성 차이점 예제

```html
<html>
  <head>
    <title>Vue Template - Computed vs Methods</title>
  </head>
  <body>
    <div id="app">
      <p>{{ message }}</p>
      <button v-on:click="reverseMsg">문자열 역순</button>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
    <script>
      new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue.js!'
        },
        methods: {
          reverseMsg: function() {
            this.message = this.message.split('').reverse().join('');
            return this.message;
          }
        }
      });
    </script>
  </body>
</html>
```

앞에서 살펴본 **computed** 속성으로 문자열 순서를 바꾼 코드와 동일한 결과를 보이지만 차이점은 버튼을 클릭 했을때만 변환된다는 것

methods 속성은 수행할 때마다 연산을 하기 때문에 별도로 캐싱를 하지 않지만, **computed** 속성은 데이터가 변경되지 않는 한 이전의 계산값을 가지고 있다가 필요할 때 바로 반환해 줌.
따라서 복잡한 연산을 반복 수행해서 화면에 나타내야 한다면 **computed** 속성을 이용하는 것이 **methods** 속성을 이용하는 것보다 성능면에서 효율적임

watch 속성

- 데이터 변화를 감지하여 자동으로 특정 로직을 수행함
- computed 속성과 유사하지만 computed 속성은 내장 API를 활용한 간단한 연산 정도로 적합한 반면 watch 속성은 데이터 호출과 같이 시간이 상대적으로 더 많이 소모되는 비동기 처리에 적합
- 인풋 박스의 입력값을 v-model 디렉티브로 연결하여 입력 값에 변화가 있을때마다 watch 속성에서 변화된 로그를 출력함

```
5    <body>
6      <div id="app">
7        <input v-model="message">
8      </div>
9
10     <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
11     <script>
12       new Vue({
13         el: '#app',
14         data: {
15           message: 'Hello Vue.js!'
16         },
17         watch: {
18           message: function(data) {
19             console.log("message의 값이 바뀝니다 : ", data);
20           }
21         }
22       });
23     </script>
24   </body>
25 </html>
```