

Vue.js

Programming Language

Index

1 Vue Instance

2 Computed Properties

3 Watchers

4 Lifecycle Hooks

5 Event Handling



Vue Instance



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

Instance 생성

- ▶ 모든 Vue 앱은 `createApp()` 함수를 사용하여 새로운 앱 인스턴스를 생성

```
const { createApp } = Vue;  
  
const app = createApp({  
  /* 최상위 컴포넌트 옵션 */  
});
```

mount

- ▶ 앱 인스턴스는 `mount()` 메서드가 호출될 때까지 아무 것도 렌더링하지 않음
- ▶ "컨테이너"가 될 실제 DOM 엘리먼트 또는 셀렉터 문자열을 인자로 필요로 함
- ▶ `.mount()` 메서드는 반드시 앱의 환경설정 및 에셋(asset)이 모두 등록 완료된 후에 호출되어야 함

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

```
app.mount("#app");
```

<https://ko.vuejs.org/api/composition-api-setup.html#basic-usage>

setup()

- ▶ Composition API의 핵심 함수
- ▶ 컴포넌트의 생성 및 초기화 시점에 호출
- ▶ 반응형 상태, Life-Cycle hooks, 계산된 속성, 메서드 등을 정의하는 중앙 허브
- ▶ props와 context를 인자로 받을 수 있음
 - props: 컴포넌트로 전달된 속성들
 - context: 컴포넌트 컨텍스트 제공 (context.attrs, context.slots, context.emit 등 접근 가능)

setup() vs <script setup>

- setup(): 명시적이고 전통적인 방식
- <script setup>: 간결하고 현대적인 방식

주의

- this 키워드 사용 불가
- 반드시 **객체를 반환**하거나 렌더 함수 반환
- 컴포넌트 초기화 시 한번만 호출

```
const app = createApp({  
  name: "APP",  
  setup() {  
    const message = ref("Hello Vue !!!");  
    return {  
      message,  
    };  
  },  
});
```

반응형 데이터: ref(), reactive()

<https://ko.vuejs.org/guide/essentials/reactivity-fundamentals.html>

반응형 데이터

- ▶ Option API에서는 data() 함수에서 객체로 관리 했으나 Composition API에서는 ref(), reactive() 함수를 이용
- ▶ setup() 함수 또는 <script setup> 내에서 직접 선언
- ▶ ref()와 reactive()로 반응형 데이터 생성
- ▶ 원시 타입: ref() 사용
- ▶ 객체/배열: reactive() 사용 (ref() 도 사용 가능)
- ▶ 컴포넌트가 자동으로 데이터 변경 감지 및 화면 갱신
- ▶ 사용 시 주의
 - 변수명에 \$,_ 시작 피하기
 - 반응형 데이터는 .value로 접근 (템플릿 제외)



- Composition API
 - 더 명시적인 상태 관리
 - 코드 재사용성 향상
 - 로직의 구조화가 용이

```
<script>
export default {
  data() {
    return {
      message: "Hello",
      count: 0,
    };
  },
</script>
```

Option API

```
<script setup>
import { ref, reactive } from "vue";

// 원시 타입: ref() 사용
const message = ref("Hello");
const count = ref(0);

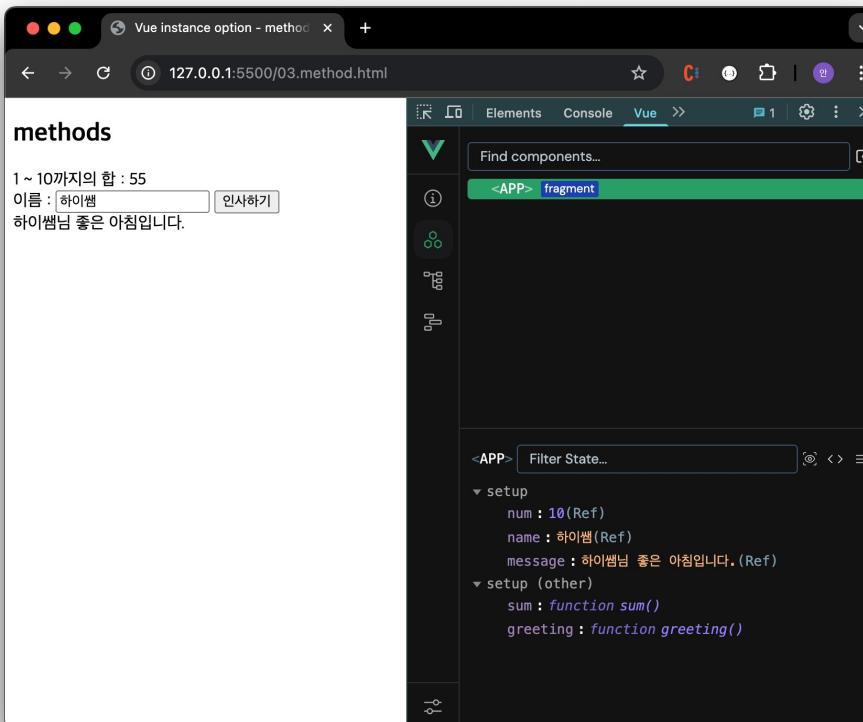
// 복잡한 객체: reactive() 사용
const userInfo = reactive({
  name: "홍길동",
  age: 30,
});
</script>
```

method

- ▶ Option API는 methods 객체 내에 메서드 정의
- ▶ Composition API에서는 직접 함수로 선언
- ▶ Vue Instance를 이용하여 직접 호출하거나 directive 표현식, Mustache 표현식에서 호출 가능
- ▶ 일반적으로 Event Handler로 사용
- ▶ <template>에서 사용하기 위해 **setup()** 함수의 **return**에 포함 시켜야 함
- ▶ 명명 규칙
 - camelCase 사용
 - 동사로 시작(동작을 나타내는 이름)
 - 명확하고 서술적인 이름
- ▶ 메서드 작성 원칙
 - 단일 책임 원칙
 - 재사용 가능하게 작성
 - 간결하고 명확한 로직

method 실습

```
<div id="app">
  <h2>methods</h2>
  1 ~ {{ num }}까지의 합 : {{ sum() }}<br />
  이름 : <input type="text" v-model="name" />
  <button @click="greeting">인사하기</button><br />
  {{ message }}
</div>/#app
```



03.method.html

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

  const app = createApp({
    name: "APP",
    setup() {
      const num = ref(10);
      const name = ref("");
      const message = ref("");

      const sum = () => {
        let sum = 0;
        for (let i = 1; i <= num.value; i++) {
          sum += i;
        }
        return sum;
      };

      const greeting = () => (message.value = name.value + "님 좋은 아침입니다.");
    },
    return {
      num, name, message,
      sum, greeting,
    };
  });
  app.mount("#app");
</script>
```

Computed Properties



계산된 속성: computed

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

computed (계산된 속성)

- ▶ data의 변화를 감지하여 동적으로 계산된 값을 이용할 때 사용
- ▶ computed는 종속된 대상 (data)이 변경될 때만 함수를 실행
- ▶ method는 호출 회수만큼 동작하지만 computed는 호출 회수와 상관없이 한번만 동작한 후 caching해서 사용
- ▶ 즉, **의존하는 값에 변화가 없다면 다시 계산하지 않고 이전 값(caching된 값)을 반환**
- ▶ 인자(argument)를 받을 수 없음
- ▶ return을 갖는 함수의 형태로 작성 (일반적으로 getter[읽기전용]의 역할로 사용, setter도 만들 수는 있음)
- ▶ vue2에서의 filter 역할 수행 (vue3에서 filter는 사라짐)
- ▶ 주의점
 - 순수 계산 로직만 포함 ::> 복잡한 로직의 경우 메서드로 분리
 - 비동기 작업 피해야 함
 - 무거운 연산 지양
 - .value를 사용해 값에 접근

계산된 속성: computed

computed

- ▶ computed는

- CDN: `const { computed } = Vue;`
- SFC: `import { computed } from 'vue';`로 불러와서 사용

```
const needTodos = computed(() => todos.value.filter(todo => !todo.done));
```

- ▶ computed로 정의된 값은 반응형 참조(ref) 형태가 되어 `needTodos.value`를 통해 접근

캐싱 메커니즘

- ▶ computed는 의존하는 reactive 데이터가 변경될 때만 내부 콜백 함수가 재실행됨
- ▶ 만약 의존성이 없다면, 함수가 한 번만 실행되고 이후에는 캐시된 값을 반환하게 됨
- ▶ 이는 불필요한 연산을 줄여 성능을 향상시키는 중요한 기능임

getter와 setter

- ▶ computed는 읽기 전용 (getter)으로 사용되지만, 객체 형태로 정의하면 setter도 구현할 수 있음

계산된 속성: computed

computed: 반응형 (Reactivity)



- computed를 사용하면 특정 값의 변화에 따라 웹페이지를 반응형으로 사용 가능
- 즉, 어떤 값을 기준으로 페이지 또는 컴포넌트를 새롭게 연산 및 출력하도록 설정 가능

```
<div id="app">
  <h2>오늘의 할일 목록</h2>
  <ul v-if="hasOpenedTodos">
    <li v-for="todo in needTodos" :key="todo.id">{{ todo.title }}</li>
  </ul>
  <div v-else>모든 할일 완료!!!</div>
</div>/#app

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

  const app = createApp({
    name: "APP",
    setup() {
      const todos = ref([
        { id: 1, title: "출결 체크", done: true },
        { id: 2, title: "1일 1알고", done: false },
        { id: 3, title: "과제 제출", done: false },
      ]);

      const needTodos = computed(() => todos.value.filter((todo) => !todo.done));
      const hasOpenedTodos = computed(() => needTodos.value.length > 0);

      return {
        todos, needTodos, hasOpenedTodos,
      };
    },
  }).mount("#app");
</script>
```

Vue instance option - comput

127.0.0.1:5500/04.computed.html

오늘의 할일 목록

- 1일 1알고
- 과제 제출

Elements Console Sources Vue

Find components... <APP> fragment

<APP> Filter State...

needTodos : Array[2] (Computed)
 ▼ 0 : Reactive
 id : 2
 title : 1일 1알고
 done : false
 ▼ 1 : Reactive
 id : 3
 title : 과제 제출
 done : false
 hasOpenedTodos : true(Computed)

04.computed.html

계산된 속성: computed

computed: 캐싱 (Caching)

```
<div id="app">
  <h2>문자열 역순 출력</h2>
  <p>원본 메시지: "{{ message }}"</p>
  <p>역순으로 표시한 메시지1: "{{ reversedMsg }}"</p>
  <p>역순으로 표시한 메시지2: "{{ reversedMsg }}"</p>
  <p>역순으로 표시한 메시지3: "{{ reversedMsg }}"</p>
  <p>현재 시간: "{{ nowTime }}"</p>
</div>/#app

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

  const app = createApp({
    name: "APP",
    setup() {
      const message = ref("Hello Vue Computed !!!");
      const reversedMsg = computed(() => {
        console.log("computed >> reverseMsg() call");
        return message.value.split("").reverse().join("");
      });
      // 종속(의존)하지 않는 경우 computed는 없데이트 되지 않음.
      const nowTime = computed(() => {
        console.log("computed >> nowTime() call");
        return Date.now();
      });
      return {
        message, reversedMsg, nowTime,
      };
    },
  }).mount("#app");
</script>
```



- computed는 캐시를 사용하여 연산 된 값을 저장함
- 반복 호출이 있더라도 이 호출이 computed라면 최초 연산 된 값을 캐시에 저장하고 재호출 없이 캐싱되어 있는 값을 불러와 작동
- 호출 시 메소드 호출 형식이 아닌 속성처럼 사용

문자열 역순 출력

원본 메시지: "Hello Vue Computed !!!"

역순으로 표시한 메시지1: "!!! detupmoC euV olleH"

역순으로 표시한 메시지2: "!!! detupmoC euV olleH"

역순으로 표시한 메시지3: "!!! detupmoC euV olleH"

현재 시간: "1743145347559"

한번만 호출됨

05.computed-caching.html

Vue instance option - comput

Elements Console Sources Vue

Find components... <APP> fragment

Filter State... <APP>

message : Hello Vue Computed !!!(Ref)
reversedMsg : !!! detupmoC euV olleH(Computed)
nowTime : 1743145347559(Computed)

Console AI assistance

No Issues

computed >> reverseMsg() call @5.computed-caching.html:27
computed >> nowTime() call @5.computed-caching.html:32

계산된 속성: computed

computed: method와 차이점

```
<div id="app">
  <h2>문자열 역순 출력</h2>
  <p>원본 메시지: "{{ message }}"</p>
  <p>역순으로 표시한 메시지1: "{{ reversedMsg() }}"</p>
  <p>역순으로 표시한 메시지2: "{{ reversedMsg() }}"</p>
  <p>역순으로 표시한 메시지3: "{{ reversedMsg() }}"</p>
</div>/#app

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

  const app = createApp({
    name: "APP",
    setup() {
      const message = ref("Hello Vue Method !!!");

      const reversedMsg = () => {
        console.log("method >> reverseMsg() call");
        return message.value.split("").reverse().join("");
      };

      return {
        message,
        reversedMsg,
      };
    },
  }).mount("#app");
</script>
```



- method의 경우 호출 시마다 매번 실행됨

문자열 역순 출력

원본 메시지: "Hello Vue Method !!!"

역순으로 표시한 메시지1: "!!! dohteM euV olleH"

역순으로 표시한 메시지2: "!!! dohteM euV olleH"

역순으로 표시한 메시지3: "!!! dohteM euV olleH"

매번 호출 됨

06.computedVSmethod.html

Vue instance option - comput

127.0.0.1:5500/06.computedVSmethod.html

Elements Console Sources Vue Find components... <APP> fragment

<APP> Filter State... message : Hello Vue Method !!!(Ref) setup (other) reversedMsg : function reversedMsg()

Console AI assistance Default levels No Issues

method >> reverseMsg() call 06.computedVSmethod.html:27
method >> reverseMsg() call 06.computedVSmethod.html:27
method >> reverseMsg() call 06.computedVSmethod.html:27

계산된 속성: computed

computed: vue2의 filter 기능

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

  const app = createApp({
    name: "APP",
    setup() {
      const money = ref(0);
      const phone = ref("");

      const viewMoney = computed(() =>
        money.value.toString().replace(/\B(?=(\d{3})+(?!d))/g, ",")
      );

      const viewPhone = computed(() => {
        let value = phone.value;
        if (!value || !(value.length === 10 || value.length === 11)) return value;
        return value.replace(/^( \d{3})( \d{3}, \d{4}) (\d{4})/g, "$1-$2-$3");
      });

      return {
        money,
        phone,
        viewMoney,
        viewPhone,
      };
    },
  }).mount("#app");
</script>
```



- vue3의 경우 filter가 없기 때문에 computed로 구현

computed를 이용한 filter 역할

금액입력 : 3자리마다 "," 찍기
입력한 금액 : 1,000,000,000원

전화번호 : 전화번호 형식
입력한 전화번호: 010-1234-5678

<APP> fragment

<APP> Filter State...

setup

- money : 1000000000(Ref)
- phone : 01012345678(Ref)
- viewMoney : 1,000,000,000(Computed)
- viewPhone : 010-1234-5678(Computed)

07.computed-filter.html

Watchers



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

⌚ watch

- ▶ 반응형 데이터 (예: ref, reactive, computed)의 변경을 감지하여 지정한 콜백 함수를 실행할 수 있게 함
- ▶ 주로 부수 효과 (side effect)를 처리하는데 사용됨 (데이터 변화에 따라 API 호출, 로컬 상태 업데이트, 또는 기타 비동기 작업을 수행)

⌚ 주요 특징

- ▶ 반응형 데이터 감시
 - watch는 특정 ref, reactive 객체, 또는 computed 값의 변경을 추적할 수 있음
- ▶ 콜백 함수
 - 기본적으로 두 개의 인자 (**첫 번째: 새 값, 두 번째: 이전 값**)를 받으며, 데이터 변경 시 호출됨
 - 초기 실행 시에는 이전 값이 undefined일 수 있음
- ▶ 즉시 실행 (immediate)
 - 옵션 **{ immediate: true }**를 사용하면 watch 등록 후 바로 콜백을 한 번 실행할 수 있음
- ▶ 실행 시점 조절 (flush): 옵션 flush를 사용해 watch의 실행 시점을 다음 중 하나로 선택할 수 있음
 - **'pre'**: DOM 업데이트 전에 실행 (기본값)
 - **'post'**: DOM 업데이트 후에 실행
 - **'sync'**: 동기적으로 즉시 실행

⌚ 형식

▶ 기본 watch

```
// 기본 Watch
watch(데이터, (newValue, oldValue) => {
  // 로직
});
```

▶ getter & options watch

```
// getter & options Watch
watch(
  () => 계산된_값,
  (newValue, oldValue) => {
    // 로직
  },
  {
    immediate: true, // 즉시 실행
    deep: true, // 객체 깊은 감시
  }
);
```

options	설명
deep	객체나 배열과 같이 중첩된 데이터 변경을 감지
immediate	감시 등록 후 즉시 콜백 실행
flush	콜백 실행 시점을 조정할 수 있으며, 'pre', 'post', 'sync' 옵션이 있음
cleanup 함수	콜백 내에서 반환하는 함수를 통해 이전 효과를 정리(cleanup)할 수 있음 (타이머나 비동기 작업을 취소하는 로직 등)

기본 watch

- ▶ primitive type watch

The screenshot shows a browser window with the URL `127.0.0.1:5500/08.watch.html`. The page content is a simple form with an input field containing "Hello hiss". Below the input is a text area also showing "Hello hiss". The browser's developer tools are open, specifically the Vue tab. In the component tree, under the <APP> fragment, there is a node labeled "message : Hello hiss(Ref)". The Timeline section shows a single entry: "변경 값 : Hello hiss, 이전 값 : Hello hiss" at line 25. The console tab shows two log entries: "변경 값 : Hello hiss, 이전 값 : Hello hi" and "변경 값 : Hello hiss, 이전 값 : Hello hiss", both at line 25.

```
<div id="app">
  <h2>watch</h2>
  <input type="text" v-model="message" /><br />
  {{ message }}
</div>/#app

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

  const app = createApp({
    name: "APP",
    setup() {
      const message = ref("Hello Vue !!!");

      watch(message, (newValue, oldValue) => {
        console.log(`변경 값 : ${newValue}, 이전 값 : ${oldValue}`);
      });

      // watch(message, async (newValue, oldValue) => {
      //   const res = await fetch('https://unpkg.com/vue')
      //   console.log(res)
      // })

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

기본 watch

- ▶ object type watch

09-1.watch-object.html

watch - object

이름 : 하이쌤
점수 : 99

하이쌤학생의 점수 : 99

Find components... <Root> fragment

<Root> Filter State... <Root>

setup

message : 하이쌤학생의 점수 : 99(Ref)
student : Object(Ref)

Console AI assistance

student 속성 값 바뀜!!!
Proxy(Object) {name: '하이쌤', score: 99}

student 속성 값 바뀜!!!
Proxy(Object) {name: '하이쌤', score: ''}

student 속성 값 바뀜!!!
Proxy(Object) {name: '하이쌤', score: 9}

student 속성 값 바뀜!!!
Proxy(Object) {name: '하이쌤', score: 99}

```

<div id="app">
  <h2>watch - object</h2>
  <label for="name">이름 : </label>
  <input type="text" id="name" v-model.lazy="student.name" /><br />
  <label for="score">점수 : </label>
  <input type="number" id="score" v-model.number="student.score" /><br />
  <div>{{ message }}</div>
</div>/#app
<script>
  const { createApp, ref, watch } = Vue;

  const app = createApp({
    setup() {
      const message = ref("");
      const student = ref({ name: "", score: 0 });

      watch(student.value, (val) => {
        console.log("student 속성 값 바뀜!!!", val);
        message.value = `${val.name}학생의 점수 : ${val.score}`;
      });
    },
    return {
      message,
      student,
    };
  },
  ).mount("#app");
</script>

```

⌚ watch - getter 함수

- ▶ 복잡한 표현식 감시: 단순한 ref나 reactive 속성이 아닌 계산된 값을 감시할 때

```
// 사용자의 전체 이름 변화 감시
watch(
  () => `${user.firstName} ${user.lastName}`,
  (newFullName) => console.log(`이름이 ${newFullName}으로 변경됨`)
);
```

- ▶ Reactive 객체의 특정 속성 감시: 전체 객체가 아닌 특정 속성만 감시

```
const user = reactive({ name: "John", age: 30 });

// user.name만 감시
watch(
  () => user.name,
  (newName) => console.log(`이름이 ${newName}으로 변경됨`)
);
```

- ▶ 여러 소스에서 파생된 값 감시

⌚ watch - getter 함수

- ▶ 복잡한 표현식 감시: 단순한 ref나 reactive 속성이 아닌 계산된 값을 감시할 때
- ▶ Reactive 객체의 특정 속성 감시: 전체 객체가 아닌 특정 속성만 감시

- ▶ 여러 소스에서 파생된 값 감시

- 사용자가 상품 수량을 변경할 때
- 사용자가 배송 방법을 변경할 때
- 장바구니에 상품을 추가하거나 제거할 때

```
// 여러 소스(장바구니 상품, 배송비 선택)에서 파생된 '최종 결제 금액' 감시
watch(
  // 상품 가격 * 수량의 합계 + 배송비
  () => {
    const itemsTotal = cartItems.reduce((sum, item) =>
      sum + item.price * item.quantity, 0);

    // 배송 방법에 따라 배송비 계산
    if (shippingMethod.value === "특급") {
      return itemsTotal + 8000;
    } else {
      return itemsTotal + 3000;
    }
  },
  (newTotal) => {
    console.log(`최종 결제 금액이 ${newTotal.toLocaleString()}원으로 변경되었습니다.`);
    // 결제 금액 변경 시 UI 업데이트 또는 서버에 정보 전송 등
  }
);
```

watch - getter 함수

- ▶ student의 score 변경만 감시

09-2.watch-object-getter.html

Vue instance option - watch

watch - deep option

이름 : hissam
점수 : 99
hissam학생의 점수 : 99

Components

<Root> fragment

<Root> Filter State...

setup

- message : hissam학생의 점수 : 99(Ref)
- student : Object(Ref)

Console

1 Issue: 1

```
when deploying for production.
student 속성 값 바뀜!!!
student 속성 값 바뀜!!! 9
student 속성 값 바됨!!! 99
```

```

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

  const app = createApp({
    setup() {
      const message = ref("");
      const student = ref({ name: "", score: 0 });

      // object가 가진 속성을 watch 할 수 없음.
      // watch(student.value.score, (val) => {
      //   console.log("student 속성 값 바뀜!!!", val);
      //   message.value = `${val.name}학생의 점수 : ${val.score}`;
      // });

      // 해결 getter
      watch(
        () => student.value.score,
        (score) => {
          console.log("student 속성 값 바뀜!!!", score);
          message.value = `${student.value.name}학생의 점수 : ${score}`;
        }
      );

      return {
        message,
        student,
      };
    },
  }).mount("#app");
</script>

```

⌚ watch - deep option

▶ 객체나 배열의 중첩된 속성 변경까지 감지

- 복잡한 품 데이터 변경 감지
- JSON 기반 설정 변경 추적
- 중첩된 컬렉션 변경 처리

The screenshot shows a browser window titled "Vue instance option - watch". The address bar shows "127.0.0.1:5500/10-1.watch-option-deep.html". The page content displays a form with fields for name ("이름 : hissam") and score ("점수 : 98"). Below the form, a message says "hissam학생의 점수 : 98". The developer tools are open, showing the component tree with a fragment node and a setup node containing a message ref. The console tab shows several log entries:

```

student 속성 값 바 10-1.watch-option-deep.html:32
됌!!! > Proxy(Object) {name: 'hissam', score: 0}
student 속성 값 바 10-1.watch-option-deep.html:32
됌!!! > Proxy(Object) {name: 'hissam', score: ''}
student 속성 값 바 10-1.watch-option-deep.html:32
m!!! > Proxy(Object) {name: 'hissam', score: 9}
student 속성 값 바 10-1.watch-option-deep.html:32
m!!! > Proxy(Object) {name: 'hissam', score: 98}

```



10-1.watch-option-deep.html

```

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

  const app = createApp({
    setup() {
      const message = ref("");
      const obj = ref({
        student: { name: "", score: 0 },
      });

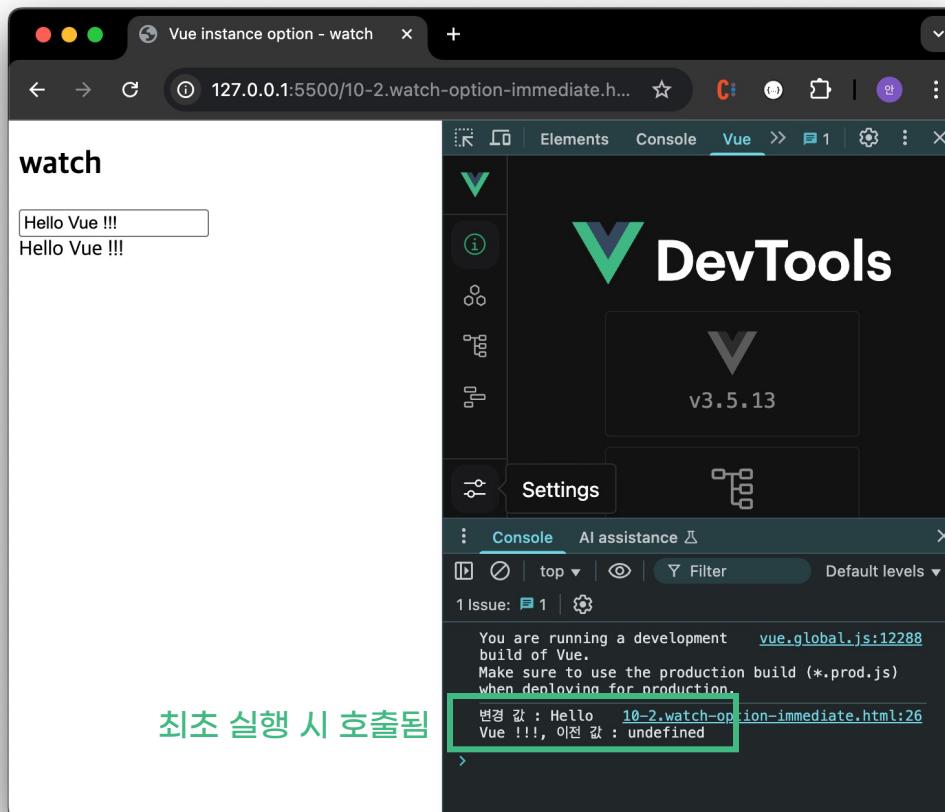
      // 깊은 감시 deep: true 설정
      watch(
        () => obj.value.student,
        (val) => {
          console.log("student 속성 값 바뀜!!!", val);
          message.value = `${val.name}학생의 점수 : ${val.score}`;
        },
        { deep: true } 주석처리 후 확인
      );
    },
    return {
      message,
      obj,
    },
  });
  app.mount("#app");
</script>

```

⌚ watch - immediate option

- ▶ watch가 생성될 때 즉시 콜백을 한 번 실행

- 컴포넌트 마운트 시 초기 데이터 로드
- props 기반 초기화 작업
- 페이지 진입 시 API 호출



10-2.watch-option-immediate.html

```

<div id="app">
  <h2>watch</h2>
  <input type="text" v-model="message" /><br />
  {{ message }}
</div>#app
<script>
  const { createApp, ref, watch } = Vue;

  const app = createApp({
    name: "APP",
    data() {
      const message = ref("Hello Vue !!!");

      watch(
        message,
        (newValue, oldValue) => {
          console.log(`변경 값 : ${newValue}, 이전 값 : ${oldValue}`);
        },
        { immediate: true }
      );

      return {
        message,
      };
    },
  }).mount("#app");
</script>

```

⌚ watch - 비동기(async)

▶ data 변경 한 결과를 이용하여 **비동기 처리**가 필요하거나 **시간이 많이 소요되는 계산과 같은 연산**을 할 경우 watch 사용

```
<h2>전국 주요 도시 날씨</h2>
<label for="area">도시 선택 : </label>
<select id="area" v-model="selArea">
  <option v-for="(ssafy, index) in areas" :key="ssafy.code" :value="ssafy.code">
    {{ ssafy.area }}
  </option>
</select>/#area
선택 도시 코드 : <input type="text" v-model="selArea" /><br /><br />
```

```
const selArea = ref("");
const weathers = ref([]);

watch(selArea, (val) => {
  if (val.length == 10) getWeather(val);
});

const getWeather = async (val) => {
  await fetch(`http://www.kma.go.kr/wid/queryDFSRSS.jsp?zone=${val}`)
    .then((response) => response.text())
    .then((data) => makeList(data));
};
```

시간(시)	아이콘	온도(°C)	날씨	습도(%)
18		15.0	맑음	25
21		9.0	맑음	45
24		6.0	맑음	60
3		5.0	맑음	65
6		4.0	맑음	70

computed VS watch

💡 computed VS watch

- ▶ watch는 "~가 변하면 이것을 해라"라는 명령형 접근방식이고
 - ▶ computed는 "이 값은 항상 ~의 계산 결과이다"라는 선언형 접근방식을 따름
 - ▶ 새로운 값을 계산하는 경우는 computed, 외부 시스템과의 동기화나 사이드 이펙트가 필요한 경우는 watch가 적합
-
- ▶ 성능
 - computed는 캐싱 기능이 있어 의존성이 변경될 때만 재계산됨
 - watch는 감시 대상이 변경될 때마다 함수가 실행됨
 - ▶ 사용 용도
 - computed: 다른 데이터에 기반하여 새 값을 계산할 때 사용
 - watch: API 호출, 로컬 스토리지 업데이트, 로깅 등의 사이드 이펙트 발생시킬 때 사용

computed VS watch



12.computedVSwatch-watch.html

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

  const app = createApp({
    setup() {
      const firstName = ref("");
      const lastName = ref("");
      const fullName = ref("");

      watch(firstName, (val) => {
        fullName.value = val + " " + lastName.value;
      });

      watch(lastName, (val) => {
        fullName.value = firstName.value + " " + val;
      });

      return {
        firstName,
        lastName,
        fullName,
      };
    },
  }).mount("#app");
</script>
```



13.computedVSwatch-computed.html

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

  const app = createApp({
    setup() {
      const firstName = ref("");
      const lastName = ref("");

      const fullName = computed(() => firstName.value + " " + lastName.value);

      return {
        firstName,
        lastName,
        fullName,
      };
    },
  }).mount("#app");
</script>
```

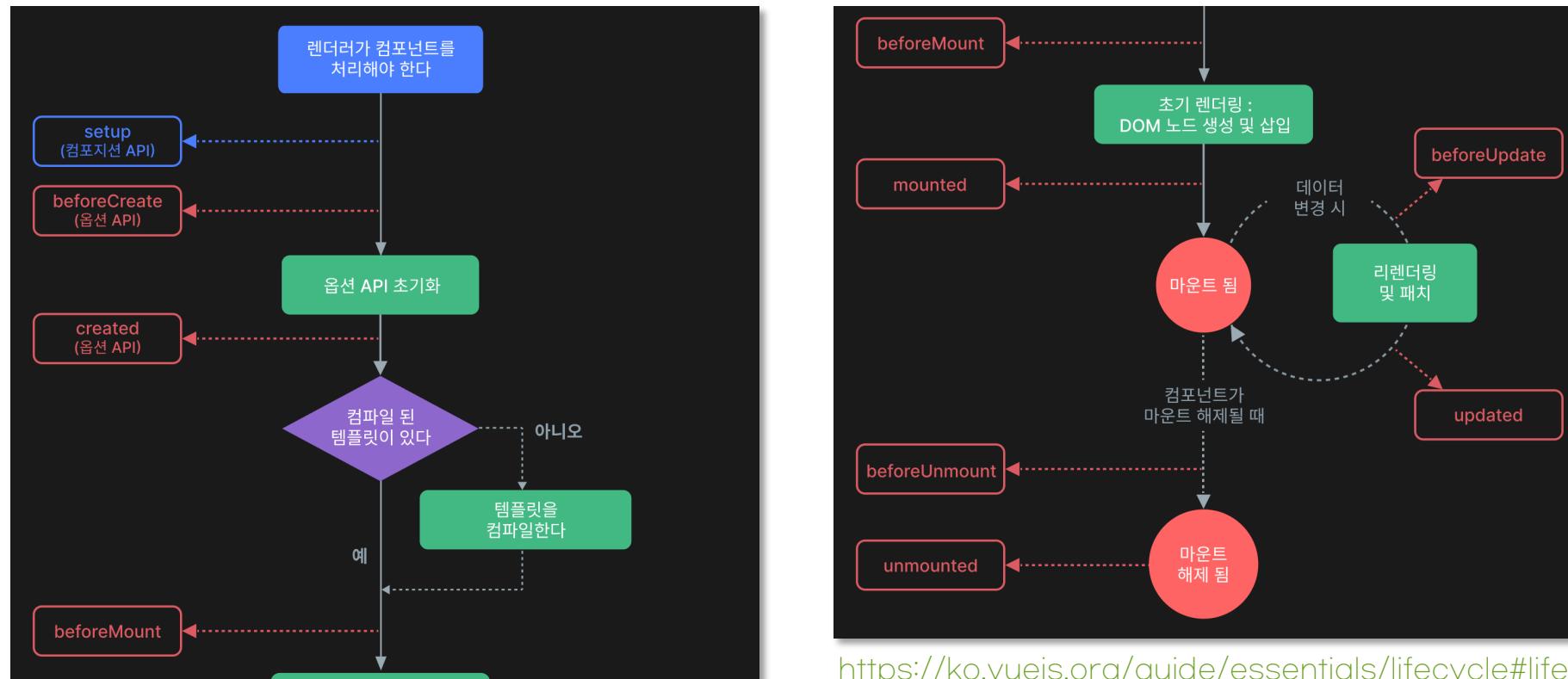
Lifecycle Hooks



<https://ko.vuejs.org/guide/essentials/lifecycle>

생애 주기 (Lifecycle Hooks)

- ▶ 기존 Options API에서 사용하던 라이프사이클 흐름을 함수 형태로 제공
- ▶ 컴포넌트의 로직을 보다 명확하고 재사용 가능하게 구성할 수 있음
- ▶ 라이프사이클 흐름은 **setup()** 함수 내부에서 **호출**되며, **on 접두사**를 사용
- ▶ 컴포넌트의 생애 주기 동안 **특정 시점**에 원하는 작업을 수행할 수 있도록 도와 줌



Lifecycle Hooks

<https://ko.vuejs.org/api/composition-api-lifecycle.html>

☞ 주요 Lifecycle Hooks

- ▶ Vue instance의 Lifecycle은 크게 나누면 **Instance의 생성**, 생성된 Instance를 **화면에 부착**
- ▶ 화면에 부착된 Instance의 **내용이 갱신**, 컴포넌트가 **마운트 해제**될 때의 4단계로 나뉨

Lifecycle Hooks	설명
onBeforeMount	컴포넌트가 DOM에 마운트되기 직전에 호출됨 이 시점에서는 아직 렌더링이 완료되지 않았기 때문에 DOM에 접근하는 것은 권장되지 않음
onMounted	컴포넌트가 DOM에 마운트된 후에 호출됨 이 혹은 DOM 조작이나 외부 라이브러리 초기화, API 호출 등 컴포넌트가 실제로 화면에 렌더링된 이후에 실행해야 하는 로직에 적합함
onBeforeUpdate	컴포넌트의 반응형 데이터가 변경되어 재렌더링되기 전에 호출됨 변경 사항을 반영하기 전에 어떤 처리를 해야 할 때 사용할 수 있음
onUpdated	데이터 변경으로 인한 업데이트가 완료되고 DOM이 새롭게 렌더링된 후에 호출됨 이 혹은 업데이트 후 DOM 상태를 확인하거나 후속 작업을 실행할 수 있음
onBeforeUnmount	컴포넌트가 제거되기 직전에 호출됨 이 단계에서 리소스를 정리하거나 이벤트 리스너를 해제하는 등의 작업을 수행할 수 있음
onUnmounted	컴포넌트가 완전히 제거된 후에 호출됨 주로 컴포넌트가 더 이상 존재하지 않을 때 실행되어야 하는 정리 작업 (clean-up)에 사용됨
onActivated / onDeactivated	<keep-alive>로 감싸진 컴포넌트의 경우, 컴포넌트가 활성화 (보여짐)되거나 비활성화 (숨김)될 때 각각 호출됨 이 기능은 컴포넌트 캐싱 시 상태 관리에 유용합니다.
onErrorCaptured	자식 컴포넌트에서 발생한 에러를 포착할 때 사용함 에러 핸들링 로직을 추가하여 앱의 안정성을 높일 수 있음

Lifecycle Hooks

▶ 초기 호출, 카운트 증가 시 호출 확인

The screenshot shows two browser windows side-by-side. Both windows are titled "Vue instance Lifecycle Hooks" and show the URL "127.0.0.1:5500/14.lifecycle-hooks.html".
The top window displays a component with the title "카운트증가" and a value of "1". Its developer tools console tab shows the following logs:

```
beforeMount call !!! 14.lifecycle-hooks.html:22
mounted call !!! 14.lifecycle-hooks.html:26
```


The bottom window displays a component with the title "카운트증가" and a value of "0". Its developer tools console tab shows the following logs:

```
beforeMount call !!! 14.lifecycle-hooks.html:22
mounted call !!! 14.lifecycle-hooks.html:26
```


Both windows also have their Vue DevTools Elements tabs open, showing the component structure and state. A green arrow points from the "0" value in the bottom window's console to the "beforeUpdate call !!!" log entry in the same window's console.

14.lifecycle-hooks.html

```
<script>
  const { createApp, ref, onBeforeMount, onMounted, onBeforeUpdate, onUpdated } = Vue;

  createApp({
    setup() {
      const count = ref(0);

      onBeforeMount(() => {
        console.log("beforeMount call !!!");
      });

      onMounted(() => {
        console.log("mounted call !!!");
      });

      onBeforeUpdate(() => {
        console.log("beforeUpdate call !!!");
      });

      onUpdated(() => {
        console.log("updated call !!!");
      });

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

Event Handling



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

☞ event

- ▶ DOM 이벤트 처리: 템플릿에서 “**v-on**” directive 또는 “@”(단축 문법)를 사용하여 이벤트를 바인딩
- ▶ setup() 함수 내에서 이벤트 핸들러를 정의함
- ▶ 사용법

```
<button v-on:click="handler">확인</button>
```

- ▶ 또는

```
<button @click="handler">확인</button>
```

- ▶ handler의 값
 - inline handler : event 발생시 실행되는 인라인 JavaScript
 - method handler : 컴포넌트에 정의된 method 이름 또는 method를 가리키는 경로

event handler : inline

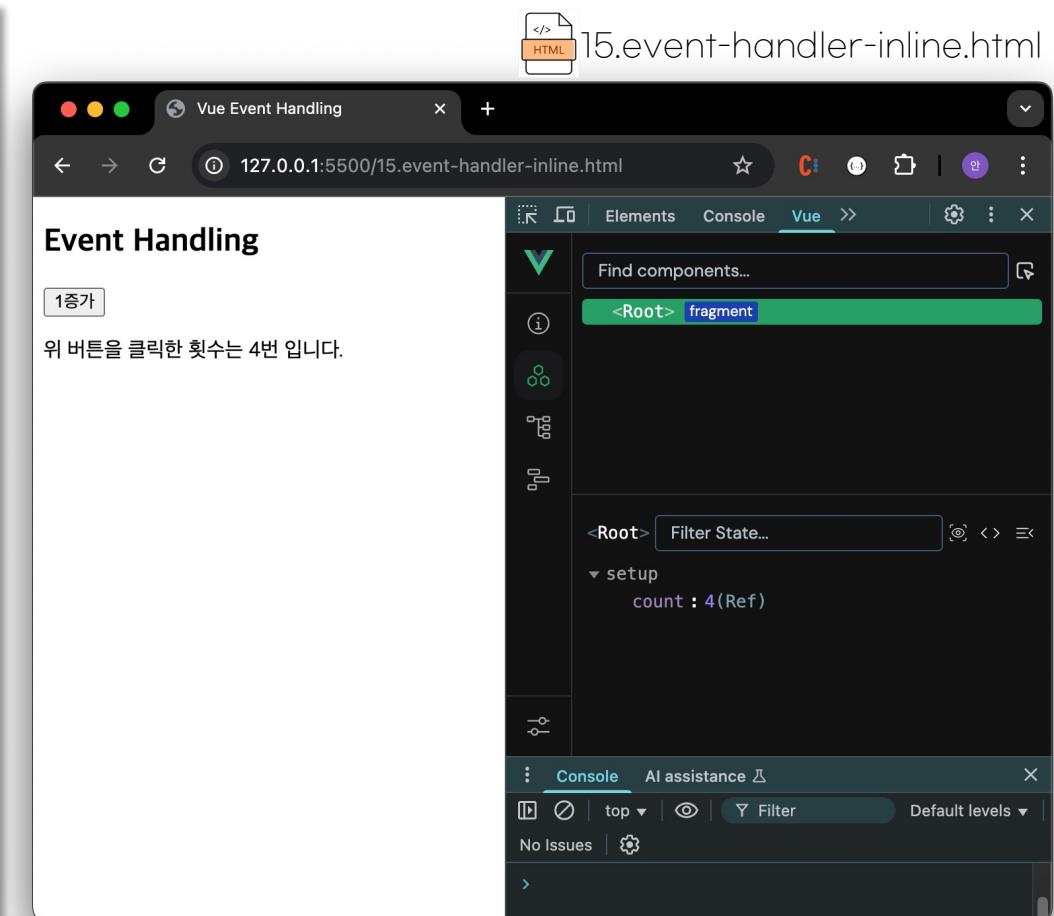
- ▶ v-on directive를 사용하여 DOM Event를 수신하고 트리거 될 때 JavaScript 실행

```
<div id="app">
  <h2>Event Handling</h2>
  <button v-on:click="count++">1증가</button>
  <!-- <button @click="count++">1증가</button> -->
  <p>위 버튼을 클릭한 횟수는 {{ count }}번 입니다.</p>
</div>/#app

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

  const app = createApp({
    setup() {
      const count = ref(0);

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



event handler : method

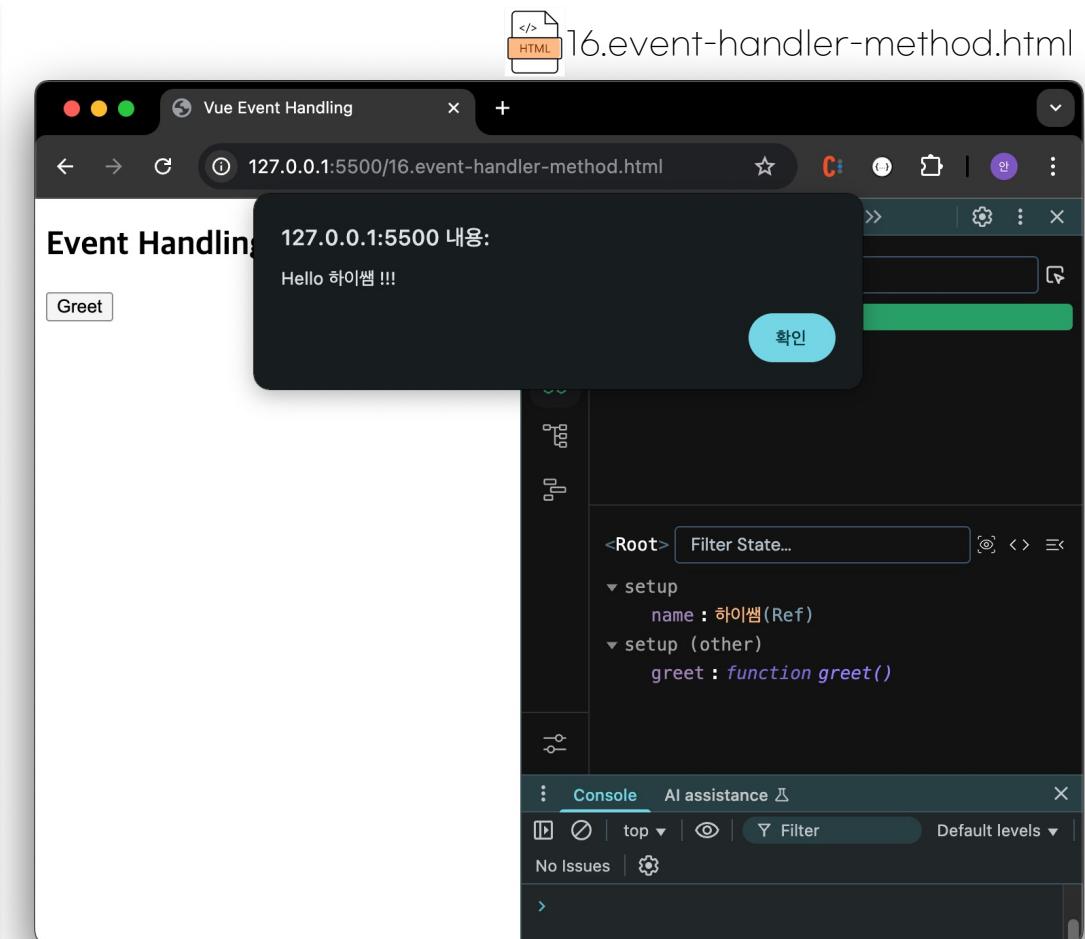
- ▶ event 발생시 처리해야 하는 로직은 대부분 복잡함
- ▶ v-on에서 event 발생시 처리해야 하는 method 이름을 받아 처리

```
<div id="app">
  <h2>Event Handling</h2>
  <button v-on:click="greet">Greet</button>
</div>#app
<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const name = ref("하이쌤");

      const greet = () => {
        alert(`Hello ${name.value} !!!`);
      };

      return {
        name,
        greet,
      };
    },
  }).mount("#app");
</script>
```



Event Handling

event handler : inline handler에서 event 객체 접근

- ▶ method 이름을 직접 바인딩 하는 대신
inline JavaScript 구문에 메소드를 직접 호출 가능
- ▶ 원본 DOM Event에 액세스 해야 하는 경우
특별한 '\$event' 변수를 사용해 method에 전달 가능



17.event-handler-inline-method-call.html

```
<div id="app">
  <h2>Event Handling</h2>
  <form action="">
    <input type="number" v-model.number="num1" />
    +
    <input type="number" v-model.number="num2" />
    <button @click="calculate1">결과보기1</button>
    <button @click="calculate2($event)">결과보기2</button>
  </form>
  <div>{{ result }}</div>
</div>/#app
```

\$event를 통해
event 객체전달

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

  const app = createApp({
    setup() {
      const num1 = ref(0);
      const num2 = ref(0);
      const result = ref(0);

      const calculate1 = () => {
        result.value = num1.value + num2.value;
      };

      const calculate2 = (event) => {
        if (event) {
          console.log(event.target);
          event.preventDefault();
        }
        result.value = num1.value + num2.value;
      };

      return {
        num1, num2, result,
        calculate1, calculate2,
      };
    },
  }).mount("#app");
</script>
```

다음 페이지 이벤트 수식어 참고

<https://ko.vuejs.org/guide/essentials/event-handling#event-modifiers>

☞ event modifiers (이벤트 수식어)

- ▶ 이전 페이지 예제에서의 `event.preventDefault()`와 같이 method 내에서 작업을 할 수도 있지만 method는 DOM의 event를 처리하는 것 보다 data 처리를 위한 로직만 작업하는 것이 좋음
- ▶ 이 문제를 해결하기 위해 v-on은 dot(.)으로 시작하는 접미사인 이벤트 수식어를 제공

```
<!-- 클릭 이벤트 전파가 중지됩니다. -->
<a @click.stop="doThis"></a>

<!-- submit 이벤트가 더 이상 페이지 리로드하지 않습니다. -->
<form @submit.prevent="onSubmit"></form>

<!-- 수식어를 연결할 수 있습니다. -->
<a @click.stop.prevent="doThat"></a>

<!-- 이벤트에 핸들러 없이 수식어만 사용할 수 있습니다. -->
<form @submit.prevent></form>

<!-- event.target이 엘리먼트 자신일 경우에만 핸들러가 실행됩니다. -->
<!-- 예를 들어 자식 엘리먼트에서 클릭 액션이 있으면 핸들러가 실행되지 않습니다. -->
<div @click.self="doThat">...</div>
```

```
<!-- 이벤트 리스너를 추가할 때 캡처 모드 사용 -->
<!-- 내부 엘리먼트에서 클릭 이벤트 핸들러가 실행되기 전에, 여기에서 먼저 핸들러가 실행됩니다. -->
<div @click.capture="doThis">...</div>

<!-- 클릭 이벤트는 단 한 번만 실행됩니다. -->
<a @click.once="doThis"></a>

<!-- 핸들러 내 `event.preventDefault()`가 포함되었더라도 -->
<!-- 스크롤 이벤트의 기본 동작(스크롤)이 발생합니다. -->
<!-- .passive 수식어는 일반적으로 모바일 장치의 성능 향상을 위해 터치 이벤트 리스너와 함께 사용 -->
<div @scroll.passive="onScroll">...</div>
```

Event Modifiers (이벤트 수식어)

기본 이벤트 수식어

modifier (수식어)	설명	예시
.stop	이벤트 전파(propagation)를 중단 (event.stopPropagation())	<button @click.stop="handleClick">클릭</button>
.prevent	이벤트의 기본 동작을 방지 (event.preventDefault())	<form @submit.prevent="handleSubmit">제출</form>
.capture	이벤트 캡처 모드 사용 (외부에서 내부로 이벤트 흐름)	<div @click.capture="handleOuterClick">내용</div>
.self	이벤트가 해당 엘리먼트에서 직접 발생했을 때만 처리	<div @click.self="handleClick">내용</div>
.once	이벤트를 한 번만 처리	<button @click.once="handleClick">한 번만 클릭 가능</button>
.passive	이벤트의 기본 동작을 방해하지 않음을 브라우저에 알림	<div @scroll.passive="handleScroll">...</div>

기본 이벤트 수식어

```
<div id="app">
  <h2>Event Handling - modifiers</h2>
  <a href="https://www.naver.com/" @click="movePage1">네이버이동</a><br />
  <a href="https://www.naver.com/" @click.once='movePage1' target="_blank">네이버이동</a>
  <a href="https://www.naver.com/" @click="movePage2">네이버이동</a><br />
  <a href="https://www.naver.com/" @click.prevent="movePage1">네이버이동</a>
  <div class="test" @click="parent">
    <button type="button" @click="child">클릭클릭</button>
  </div>/.test
  <div class="test" @click="parent">
    <button type="button" @click.stop="child">클릭클릭</button>
  </div>/.test
</div>/#app
```

 18.event-modifiers.html

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

  const app = createApp({
    setup() {
      const message = ref("이동될까요?");

      const movePage1 = () => {
        alert("movePage1 : " + message.value);
      };

      const movePage2 = (e) => {
        e.preventDefault();
        alert("movePage2 : " + message.value);
      };

      const parent = () => {
        alert("div(parent) click !!!");
      };

      const child = () => {
        alert("button(child) click !!!");
      };

      return {
        message,
        movePage1, movePage2, parent, child,
      };
    },
  }).mount("#app");
</script>
```

Event Modifiers (이벤트 수식어)

☞ 키보드 이벤트 수식어

- ▶ 키보드 이벤트를 수신할 때 v-on에 대한 키 수식어를 추가
- ▶ 키 수식어를 사용하지 않을 경우 이벤트 로직 부분에 특정 키가 눌린 것을 확인해야 하는 복잡함 발생

일반적으로 사용되는 키 관련 수식어

- .enter
- .tab
- .delete ("Delete" 및 "Backspace" 키 모두 캡처)
- .esc
- .space
- .up
- .down
- .left
- .right

시스템 입력 관련 수식어

- .ctrl
- .alt
- .shift
- .meta

마우스 관련 수식어

- .left
- .right
- .middle

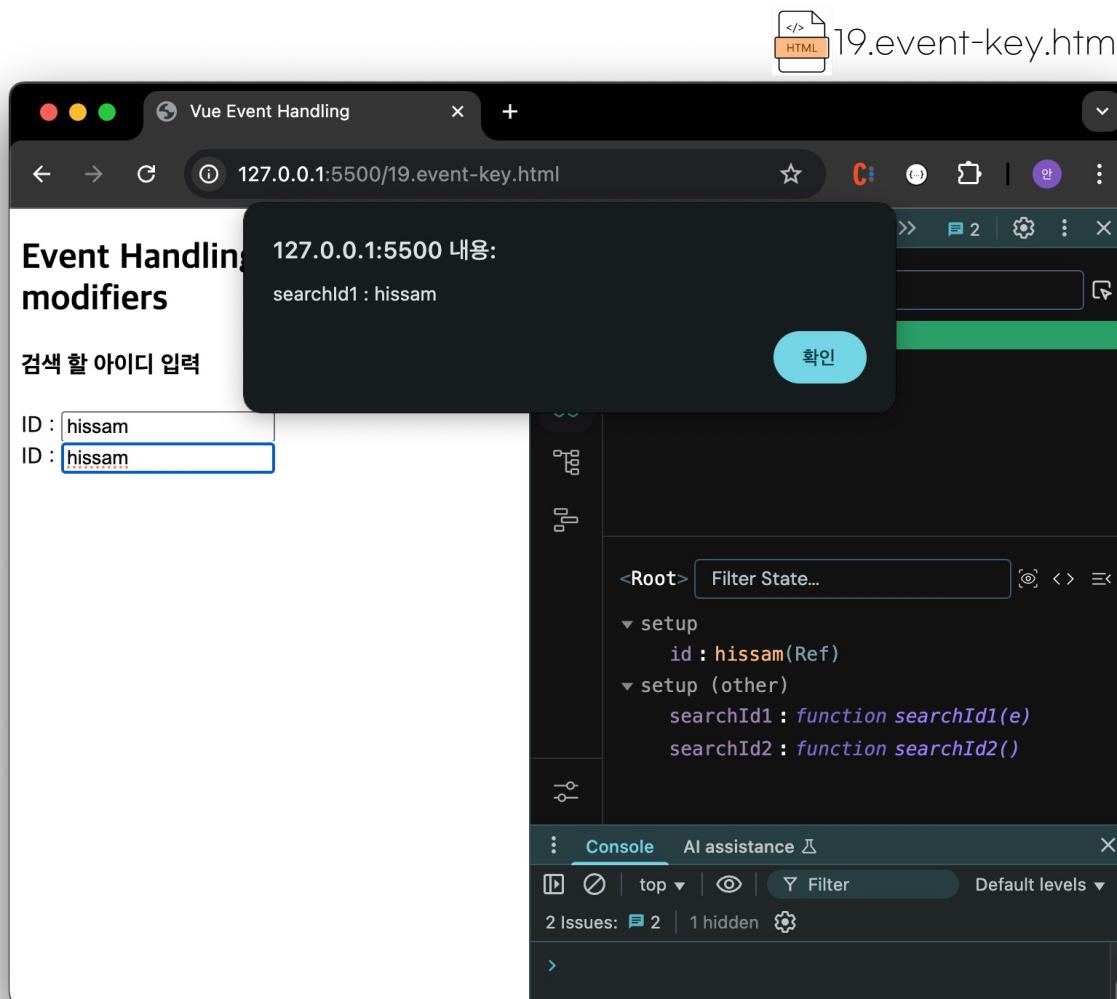
Event Modifiers (이벤트 수식어)

☞ 키보드 이벤트 수식어

modifier (수식어)	설명	예시
.enter	Enter 키	<input @keyup.enter="submit">
.tab	Tab 키	<input @keyup.tab="nextField">
.delete	Delete 및 Backspace 키	<input @keyup.delete="clearInput">
.esc	Escape 키	<input @keyup.esc="cancel">
.space	Space 키	<input @keyup.space="activateItem">
.up	위쪽 화살표 키	<input @keyup.up="increaseValue">
.down	아래쪽 화살표 키	<input @keyup.down="decreaseValue">
.left	왼쪽 화살표 키	<input @keyup.left="movePrev">
.right	오른쪽 화살표 키	<input @keyup.right="moveNext">

☞ 키보드 이벤트 수식어

- ▶ 값 입력 후 엔터



```
<div id="app">
  <h2>Event Handling - input modifiers</h2>
  <h4>검색 할 아이디 입력</h4>
  ID : <input type="text" v-model="id" @keyup="searchId1" /><br />
  ID : <input type="text" v-model="id" @keyup.enter="searchId2" /><br />
</div>/#app

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

  const app = createApp({
    setup() {
      const id = ref("");
      const searchId1 = (e) => {
        if (e.keyCode == 13) {
          alert(`searchId1 : ${id.value}`);
        }
      };

      const searchId2 = () => {
        alert(`searchId1 : ${id.value}`);
      };

      return {
        id,
        searchId1,searchId2,
      };
    },
  }).mount("#app");
</script>
```

Event Modifiers (이벤트 수식어)

⑨ 시스템 키 수식어

modifier (수식어)	설명	예시
.ctrl	Ctrl 키	<div @click.ctrl="doSomething">Ctrl + Click</div>
.shift	Shift 키	<button @click.shift="shiftAction">Shift + Click</button>
.alt	Alt 키	<input @keyup.alt.enter="clear">
.meta	Mac의 Command 키, Windows의 Windows 키	<div @click.meta="metaAction">Meta + Click</div>
.exact	정확한 키 조합만 처리	<button @click.ctrl.exact="onCtrlClick">Ctrl만 누른 상태에서 Click</button>

Event Modifiers (이벤트 수식어)

MouseEvent 수식어

modifier (수식어)	설명	예시
.left	왼쪽 마우스 버튼	<div @mousedown.left="leftClick">왼쪽 클릭</div>
.right	오른쪽 마우스 버튼	<div @mousedown.right="showContextMenu">우클릭 메뉴</div>
.middle	가운데 마우스 버튼	<div @mousedown.middle="middleClick">휠 클릭</div>

Event Modifiers (이벤트 수식어)

☞ v-model 수식어

modifier (수식어)	설명	예시
.lazy	input 이벤트 대신 change 이벤트 후에 동기화	<input v-model.lazy="message">
.number	사용자 입력을 숫자로 변환	<input v-model.number="age">
.trim	입력의 앞뒤 공백 제거	<input v-model.trim="username">

Event Modifiers (이벤트 수식어)

modifier chaining (수식어 체이닝)

조합	설명	예시
.stop.once	이벤트 전파를 중단하고 한 번만 실행	<button @click.stop.once="handleClick">클릭</button>
.prevent.self	엘리먼트 자체에서 발생한 이벤트만 기본 동작 방지	<div @click.prevent.self="handleClick">클릭</div>
.ctrl.enter	Ctrl+Enter를 눌렀을 때만 실행	<input @keyup.ctrl.enter="send">
.right.prevent	우클릭 시 기본 컨텍스트 메뉴를 방지	<div @contextmenu.right.prevent="showCustomMenu">우클릭 영역</div>

Thank you !!!

Programming Language