Options API와 함께 타입스크립트 사용하기

이 페이지에서는 **Vue**와 함께 타입스크립트 사용하기 에 대한 개요를 이미 읽었다고 가정합니다.

> **TIP**
>
> **Vue**는 **Options API**에서 타입스크립트를 지원 합니다. 하지만 보다 간단하고 효율적이며 강력한 타입 추론을 제공하는 **Composition API**를 타입스크립트와 함께 사용하는 것을 추천합니다.

컴포넌트 Props 작성

Options API의 props에 대한 타입 추론은 컴포넌트를 `defineComponent()` 로 래핑해야 합니다. 이를 통해 Vue는 `props` 옵션을 기반으로 props의 타입을 추론할 수 있으며 `required: true` 및 `default` 와 같은 추가 옵션을 사용할 수 있습니다.

ts

```ts
import { defineComponent } from 'vue'

export default defineComponent({
  // type inference enabled
  props: {
    name: String,
    id: [Number, String],
    msg: { type: String, required: true },
    metadata: null
  },
  mounted() {
    this.name // type: string | undefined
    this.id // type: number | string | undefined
    this.msg // type: string
    this.metadata // type: any
  }
})
```

그러나 런타임 `props` 옵션은 생성자 함수를 prop의 타입으로 사용하는 것만 지원합니다. 중첩된 속성이나 함수 호출 시그니처가 있는 객체와 같은 복잡한 타입을 지정할 수 있는 방법은 없습니다.

복잡한 props 유형에 어노테이팅 하기 위해 `PropType` 유틸리티 타입을 사용할 수 있습니다:

ts

```ts
import { defineComponent } from 'vue'
import type { PropType } from 'vue'

interface Book {
  title: string
  author: string
  year: number
}

export default defineComponent({
  props: {
    book: {
      // `Object`에 대한 자세한 타입을 제공합니다
      type: Object as PropType<Book>,
      required: true
    },
    // functions도 어노테이팅 가능합니다
    callback: Function as PropType<(id: number) => void>
  },
```

```
  mounted() {
    this.book.title // string
    this.book.year // number

    // TS Error: argument of type 'string' is not
    // assignable to parameter of type 'number'
    this.callback?.('123')
  }
})
```

주의사항

타입스크립트의 4.7 버전 미만을 사용하신다면 `validator` 및 `default` prop 옵션에 함수 값을 사용할 때 주의해야 합니다. 화살표 함수를 사용해야 합니다:

```ts
import { defineComponent } from 'vue'
import type { PropType } from 'vue'

interface Book {
  title: string
  year?: number
}

export default defineComponent({
  props: {
    bookA: {
      type: Object as PropType<Book>,
      // 타입스크립트 4.7 버전 미만인 경우 반드시 화살표 함수를 사용해야 함.
      default: () => ({
        title: 'Arrow Function Expression'
      }),
      validator: (book: Book) => !!book.title
    }
  }
})
```

이렇게 하면 타입스크립트가 이러한 함수 내에서 `this` 유형 유추에 실패하는 것을 방지할 수 있습니다. 이것은 과거의 설계적 한계 때문에 발생한 현상으로, **4.7**버전에서 개선 되었습니다.

컴포넌트 Emits 작성

`emits` 옵션의 객체 구문을 사용하여 emitted 이벤트의 페이로드 타입을 선언할 수 있습니다. 또한 선언되지 않은 emitted 이벤트는 호출될 때 타입 에러를 발생시킵니다.

```ts
import { defineComponent } from 'vue'

export default defineComponent({
  emits: {
    addBook(payload: { bookName: string }) {
      // perform runtime validation
      // 런타임 유효성 검사 수행
      return payload.bookName.length > 0
    }
  },
  methods: {
    onSubmit() {
      this.$emit('addBook', {
        bookName: 123 // Type error! // 타입 에러!
```

```
      })

      this.$emit('non-declared-event') // Type error! // 타입 에러!
    }
  }
})
```

Computed 속성 작성

Computed 속성은 반환 값을 기반으로 타입을 유추합니다.

ts

```
import { defineComponent } from 'vue'

export default defineComponent({
  data() {
    return {
      message: 'Hello!'
    }
  },
  computed: {
    greeting() {
      return this.message + '!'
    }
  },
  mounted() {
    this.greeting // type: string
  }
})
```

경우에 따라서는 구현이 올바른지 확인하기 위해 computed 속성 타입에 명시적으로 어노테이팅 가능합니다:

ts

```
import { defineComponent } from 'vue'

export default defineComponent({
  data() {
    return {
      message: 'Hello!'
    }
  },
  computed: {
    // 반환 타입에 명시적인 어노테이팅
    greeting(): string {
      return this.message + '!'
    },

    // 쓰기 가능한 computed 속성에 어노테이팅
    greetingUppercased: {
      get(): string {
        return this.greeting.toUpperCase()
      },
      set(newValue: string) {
        this.message = newValue.toUpperCase()
      }
    }
  }
})
```

타입스크립트가 순환 참조 루프로 인해 computed 속성의 타입을 추론하지 못하는 일부의 경우에는 명시적 어노테이팅이 필요 할 수 있습니다.

이벤트 핸들러 작성

네이티브 DOM 이벤트를 처리할 때 핸들러에 올바르게 인자를 전달하는 것이 유용합니다. 다음 예를 살펴보겠습니다:

vue

```vue
<script lang="ts">
import { defineComponent } from 'vue'

export default defineComponent({
  methods: {
    handleChange(event) {
      // `event` implicitly has `any` type
      console.log(event.target.value)
    }
  }
})
</script>

<template>
  <input type="text" @change="handleChange" />
</template>
```

타입 어노테이션이 없으면 event 인자는 암묵적으로 any 타입을 갖습니다. "strict": true 또는 "noImplicitAny": true 가 tsconfig.json 에서 사용되는 경우에 TS 에러가 발생합니다. 따라서 이벤트 핸들러의 전달인자에 어노테이팅 하는 것을 권장합니다. 또한 event 에 대한 속성을 캐스팅해야 할 수도 있습니다:

ts

```ts
import { defineComponent } from 'vue'

export default defineComponent({
  methods: {
    handleChange(event: Event) {
      console.log((event.target as HTMLInputElement).value)
    }
  }
})
```

전역 속성 확장

Some plugins install globally available properties to all component instances via **app.config.globalProperties** . For example, we may install this.$http for data-fetching or this.$translate for internationalization. To make this play well with TypeScript, Vue exposes a ComponentCustomProperties interface designed to be augmented via **TypeScript module augmentation**:

일부 플러그인은 **app.config.globalProperties** 를 통해 모든 컴포넌트 인스턴스에 전역적으로 사용 가능한 프로퍼티를 설치합니다. 예를 들어, 데이터 불러오기를 위해 this.$http 를 설치하거나 국제화를 위해 this.$translate 를 설치할 수 있습니다. 이를 타입스크립트에서 잘 작동하도록 하기 위해 Vue는 타입스크립트 모듈 증강을 통해 증강되도록 설계된 ComponentCustomProperties 인터페이스를 노출합니다:

ts

```ts
import axios from 'axios'

declare module 'vue' {
  interface ComponentCustomProperties {
    $http: typeof axios
    $translate: (key: string) => string
  }
}
```

참고:

컴포넌트 타입 확장을 위한 타입스크립트 유닛 테스트

타입 배치 증강

타입 확장을 .ts 파일이나 프로젝트의 *.d.ts 파일에 넣을 수 있습니다. 어느 쪽이든 tsconfig.json 에 포함되어 있는지 확인하십시오. 라이브러리/플러그인 작성자의 경우 이 파일을 package.json 의 types 속성에 작성해야합니다.

모듈 확장을 활용하려면 타입스크립트 모듈 에 선언 되었는지 확인해야 합니다. 즉, 파일은 export {} 일지라도 최소한 하나의 최상위 import 또는 export 를 포함해야 합니다. 기능 확장이 모듈 외부에 선언되면 원래 타입을 사용하지 않고 덮어씁니다!

ts

```ts
// Does not work, overwrites the original types.
declare module 'vue' {
  interface ComponentCustomProperties {
    $translate: (key: string) => string
  }
}
```

ts

```ts
// Works correctly
export {}

declare module 'vue' {
  interface ComponentCustomProperties {
    $translate: (key: string) => string
  }
}
```

사용자 지정 옵션 증강

vue-router 와 같은 일부 플러그인은 beforeRouteEnter 와 같은 사용자 지정 컴포넌트 옵션을 지원합니다:

ts

```ts
import { defineComponent } from 'vue'

export default defineComponent({
  beforeRouteEnter(to, from, next) {
    // ...
  }
})
```

적절한 타입 보강이 없으면, 이 훅의 인자는 암시적으로 any 타입을 갖습니다. 이러한 사용자 정의 옵션을 지원하기 위해 ComponentCustomOptions 인터페이스를 보강할 수 있습니다:

ts

```ts
import { Route } from 'vue-router'

declare module 'vue' {
  interface ComponentCustomOptions {
    beforeRouteEnter?(to: Route, from: Route, next: () => void): void
  }
}
```

이제 beforeRouteEnter 옵션이 올바르게 입력됩니다. 이것은 단지 예시일 뿐이며, vue-router 와 같이 잘 입력된 라이브러리는 자체 유형 정의에서 이러한 증강을 자동으로 수행해야 합니다.

이 증강의 배치에는 전역 속성 증강과 동일한 제한이 적용됩니다.

참고:

TypeScript unit tests for component type extensions