Vue와 TypeScript 함께 사용하기

TypeScript와 같은 타입 시스템은 빌드 시점의 정적 분석을 통해 많은 일반적인 오류를 감지할 수 있습니다. 이는 프로덕션에서의 런타임 오류 가능성을 줄여주고, 대규모 애플리케이션에서 코드를 더 자신 있게 리팩터링할 수 있게 해줍니다. TypeScript는 또한 IDE에서 타입 기반 자동 완성 기능을 제공하여 개발자 경험을 향상시킵니다.

Vue는 자체적으로 TypeScript로 작성되었으며, 일급 TypeScript 지원을 제공합니다. 모든 공식 Vue 패키지는 번들된 타입 선언을 포함하고 있어 별도의 설정 없이 바로 사용할 수 있습니다.

프로젝트 설정

`create-vue` 는 공식 프로젝트 스캐폴딩 도구로, Vite 기반의 TypeScript 준비가 완료된 Vue 프로젝트를 스캐폴딩할 수 있는 옵션을 제공합니다.

개요

Vite 기반 설정에서는 개발 서버와 번들러가 트랜스파일만 수행하며 타입 체크는 하지 않습니다. 이를 통해 TypeScript를 사용할 때에도 Vite 개발 서버의 속도가 매우 빠르게 유지됩니다.

> 개발 중에는 타입 오류에 대한 즉각적인 피드백을 위해 좋은 IDE 설정에 의존하는 것을 권장합니다.
>
> SFC를 사용하는 경우, 커맨드라인 타입 체크 및 타입 선언 생성을 위해 `vue-tsc` 유틸리티를 사용하세요. `vue-tsc` 는 TypeScript의 커맨드라인 인터페이스인 `tsc` 의 래퍼입니다. TypeScript 파일뿐만 아니라 Vue SFC도 지원한다는 점을 제외하면 `tsc` 와 거의 동일하게 동작합니다. `vue-tsc` 를 워치 모드로 Vite 개발 서버와 병렬로 실행하거나, vite-plugin-checker와 같은 Vite 플러그인을 사용해 별도의 워커 스레드에서 체크를 실행할 수 있습니다.
>
> Vue CLI도 TypeScript를 지원하지만, 더 이상 권장되지 않습니다. 아래 참고 사항을 확인하세요.

Visual Studio Code (VS Code)는 TypeScript에 대한 훌륭한 기본 지원을 제공하므로 강력히 추천합니다.

Vue - Official (이전 Volar)은 Vue SFC 내에서 TypeScript 지원을 제공하는 공식 VS Code 확장 프로그램으로, 많은 훌륭한 기능을 함께 제공합니다.

> ⓘ TIP
>
> Vue - Official 확장 프로그램은 Vue 2용 공식 VS Code 확장 프로그램이었던 Vetur를 대체합니다. Vetur가 설치되어 있다면, Vue 3 프로젝트에서는 반드시 비활성화하세요.

WebStorm 역시 TypeScript와 Vue 모두에 대한 기본 지원을 제공합니다. 다른 JetBrains IDE들도 기본적으로 또는 무료 플러그인을 통해 지원합니다. 2023.2 버전부터 WebStorm과 Vue 플러그인은 Vue Language Server에 대한 내장 지원을 제공합니다. 설정 > Languages & Frameworks > TypeScript > Vue에서 모든 TypeScript 버전에 대해 Volar 통합을 사용할 수 있도록 Vue 서비스를 설정할 수 있습니다. 기본적으로 TypeScript 5.0 이상에서는 Volar가 사용됩니다.

`tsconfig.json` 설정하기

`create-vue` 로 스캐폴딩된 프로젝트에는 미리 설정된 `tsconfig.json` 이 포함되어 있습니다. 기본 설정은 `@vue/tsconfig` 패키지에 추상화되어 있습니다. 프로젝트 내에서는 프로젝트 참조를 사용하여 서로 다른 환경에서 실행되는 코드(예: 앱 코드와 테스트 코드)에 대해 올바른 타입을 보장합니다.

`tsconfig.json` 을 수동으로 설정할 때 주목할 만한 옵션은 다음과 같습니다:

- `compilerOptions.isolatedModules` 는 `true` 로 설정되어 있습니다. 이는 Vite가 TypeScript 트랜스파일에 esbuild를 사용하며, 단일 파일 트랜스파일의 한계가 있기 때문입니다. `compilerOptions.verbatimModuleSyntax` 는 `isolatedModules` 의 상위 집합으로, `@vue/tsconfig` 에서 사용하는 좋은 선택지입니다.
- Options API를 사용하는 경우, `compilerOptions.strict` 를 `true` 로(또는 최소한 `strict` 플래그의 일부인 `compilerOptions.noImplicitThis` 를 활성화) 설정해야 컴포넌트 옵션에서 `this` 의 타입 체크를 활용할 수 있습니다. 그렇지 않으면 `this` 가 `any` 로 처리됩니다.
- 빌드 도구에서 리졸버 별칭을 설정한 경우, 예를 들어 `create-vue` 프로젝트에서 기본으로 설정된 `@/*` 별칭 등, TypeScript에서도 `compilerOptions.paths` 를 통해 별칭을 설정해야 합니다.
- Vue에서 TSX를 사용할 계획이라면, `compilerOptions.jsx` 를 `"preserve"` 로, `compilerOptions.jsxImportSource` 를 `"vue"` 로 설정하세요.

공식 TypeScript 컴파일러 옵션 문서
esbuild TypeScript 컴파일 주의사항

Vue CLI 및 `ts-loader` 참고 사항

Vue CLI와 같은 webpack 기반 설정에서는 `ts-loader` 와 같이 타입 체크를 모듈 변환 파이프라인의 일부로 수행하는 것이 일반적입니다. 하지만 타입 시스템은 타입 체크를 위해 전체 모듈 그래프에 대한 지식이 필요하므로, 개별 모듈의 변환 단계는 적합한 위치가 아닙니다. 이로 인해 다음과 같은 문제가 발생합니다:

> `ts-loader` 는 변환 이후의 코드만 타입 체크할 수 있습니다. 이는 IDE나 `vue-tsc` 에서 직접 소스 코드로 매핑되는 오류와 일치하지 않습니다.
>
> 타입 체크가 느릴 수 있습니다. 코드 변환과 동일한 스레드/프로세스에서 수행되면, 전체 애플리케이션의 빌드 속도에 큰 영향을 미칩니다.
>
> 이미 IDE에서 별도의 프로세스로 타입 체크가 실행되고 있으므로, 개발 경험의 속도 저하를 감수할 만한 가치가 없습니다.

현재 Vue CLI를 통해 Vue 3 + TypeScript를 사용 중이라면, Vite로의 마이그레이션을 강력히 권장합니다. 타입 체크를 위해 `vue-tsc` 로 전환할 수 있도록 트랜스파일 전용 TS 지원을 활성화하는 CLI 옵션도 준비 중입니다.

일반 사용 참고 사항

`defineComponent()`

TypeScript가 컴포넌트 옵션 내부의 타입을 올바르게 추론할 수 있도록 하려면, 컴포넌트를 `defineComponent()` 로 정의해야 합니다:

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

export default defineComponent({
  // 타입 추론 활성화
  props: {
    name: String,
    msg: { type: String, required: true }
  },
  data() {
    return {
      count: 1
```

```
  },
  mounted() {
    this.name // 타입: string | undefined
    this.msg // 타입: string
    this.count // 타입: number
  }
})
```

`defineComponent()` 는 `<script setup>` 없이 Composition API를 사용할 때 `setup()` 에 전달되는 props의 타입도 추론할 수 있습니다:

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

export default defineComponent({
  // 타입 추론 활성화
  props: {
    message: String
  },
  setup(props) {
    props.message // 타입: string | undefined
  }
})
```

참고:

- webpack 트리쉐이킹 참고 사항
- `defineComponent` 타입 테스트

> ⓘ TIP
>
> `defineComponent()` 는 일반 JavaScript로 정의된 컴포넌트에도 타입 추론을 활성화합니다.

싱글 파일 컴포넌트에서의 사용

SFC에서 TypeScript를 사용하려면 `<script>` 태그에 `lang="ts"` 속성을 추가하세요. `lang="ts"` 가 있으면 모든 템플릿 표현식에도 더 엄격한 타입 체크가 적용됩니다.

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

export default defineComponent({
  data() {
    return {
      count: 1
    }
```

```
})
</script>

<template>
  <!-- 타입 체크 및 자동 완성 활성화 -->
  {{ count.toFixed(2) }}
</template>
```

`lang="ts"` 는 `<script setup>` 과도 함께 사용할 수 있습니다:

vue

```
<script setup lang="ts">
// TypeScript 활성화
import { ref } from 'vue'

const count = ref(1)
</script>

<template>
  <!-- 타입 체크 및 자동 완성 활성화 -->
  {{ count.toFixed(2) }}
</template>
```

템플릿에서의 TypeScript

`<script lang="ts">` 또는 `<script setup lang="ts">` 를 사용할 때 `<template>` 에서도 바인딩 표현식에 TypeScript를 사용할 수 있습니다. 이는 템플릿 표현식에서 타입 캐스팅이 필요한 경우 유용합니다.

다음은 인위적인 예시입니다:

vue

```
<script setup lang="ts">
let x: string | number = 1
</script>

<template>
  <!-- x가 string일 수 있으므로 오류 발생 -->
  {{ x.toFixed(2) }}
</template>
```

인라인 타입 캐스트로 해결할 수 있습니다:

vue

```
<script setup lang="ts">
let x: string | number = 1
</script>

<template>
```

```
</template>
```

> **TIP**
>
> Vue CLI 또는 webpack 기반 설정을 사용하는 경우, 템플릿 표현식에서 TypeScript를 사용하려면 `vue-loader@^16.8.0` 이 필요합니다.

TSX와 함께 사용하기

Vue는 JSX / TSX로 컴포넌트를 작성하는 것도 지원합니다. 자세한 내용은 렌더 함수 & JSX 가이드에서 다룹니다.

제네릭 컴포넌트

제네릭 컴포넌트는 두 가지 경우에 지원됩니다:

- SFC에서: `<script setup>` 의 `generic` 속성
- 렌더 함수 / JSX 컴포넌트: `defineComponent()` 의 함수 시그니처

API별 레시피

- Composition API에서 TS 사용
- Options API에서 TS 사용