타입스크립트와 함께 Vue 사용하기

타입스크립트와 같은 타입 시스템은 빌드 시 정적 분석을 통해 많은 일반에러를 감지할 수 있습니다. 이렇게 하면 프로덕션에서 런타임 에러가 발생할 가능성이 줄어들고 대규모 앱에서 안정적으로 코드를 리팩토링할 수 있습니다. 또한 타입스크립트는 여러 IDE에서 제공하는 타입기반 자동완성 기능을 통해 개발 환경을 개선 할 수 있습니다.

Vue는 타입스크립트로 작성 되었으며 최고 수준의 타입스크립트 지원을 제공합니다. 모든 공식 Vue 패키지에는 타입 선언파일을 기본적으로 포함하여 제공합니다.

프로젝트 설정

공식 프로젝트 스캐폴딩 도구인 **create-vue** 는 **Vite** 기반으로 타입스크립트를 지원하는 Vue 프로젝트를 생성할 수 있습니다.

개요

Vite 기반 설정을 사용하면 개발 서버와 번들러가 트랜스파일 만 수행하며, 타입 검사를 하지 않습니다. 이렇게 하면 타입스크립트를 사용하는 경우에도 Vite dev 서버가 빠른 속도로 유지됩니다.

개발하는 동안 타입 에러에 대한 즉각적인 피드백을 위해 좋은 **IDE** 설정 에 의존하는 것을 추천합니다.

SFC를 사용하는 경우 command line 타입 확인 및 타입 선언 생성을 위해 **vue-tsc** 유틸리티를 사용하십시오. vue-tsc 는 타입스크립트의 자체 command line 인터페이스인 tsc 를 포함하고 있습니다. 타입스크립트 파일 외에도 Vue SFC를 지원한다는 점을 제외하면 tsc 와 거의 동일하게 작동합니다.

Vue CLI는 타입스크립트를 지원 하지만 더 이상 권장되지 않습니다. 아래 사항을 참고 하십시오.

IDE 지원

Visual Studio Code (VS Code)는 기본적으로 타입스크립트를 훌륭하게 지원하고 있기에 강력하게 권장합니다.

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

> **TIP**
>
> **Vue** - 공식 확장은 이전에 **Vue 2**용 공식 **VS Code** 확장 기능이었던 **Vetur**을 대체합니다. 현재 **Vetur**를 설치한 경우, **Vue 3** 프로젝트에서는 비활성화해야 합니다.

WebStorm 은 타입스크립트와 Vue 모두 기본적으로 지원합니다. 다른 JetBrains IDE도 무료 플러그인 설치를 통해 바로 사용 가능합니다. 2023.2 버전을 기준으로, WebStorm과 Vue 플러그인은 Vue Language Server에 대한 내장 지원을 제공합니다. 세팅 > 언어 및 프레임워크 > TypeScript > Vue 에서 모든 TypeScript 버전에 대해 Volar 통합을 사용하도록 Vue 서비스를 설정할 수 있습니다. 기본적으로 TypeScript 5.0 이상의 버전에 대해 Volar가 사용됩니다.

tsconfig.json 설정

create-vue 를 통해 생성된 프로젝트에는 사전 설정된 tsconfig.json 가 포함됩니다. 기본 설정은 **@vue/tsconfig** 패키지에서 추상화됩니다. 프로젝트 내에서 프로젝트 참조 를 사용하여 다른 환경(예: 앱 코드와 테스트 코드에는 서로 다른 전역 변수가 있어야 합니다.)에서 실행되는 코드에 대한 올바른 타입을 확인합니다.

tsconfig.json 을 수동으로 구성할 때 몇 가지 알고 있으면 좋은 옵션들은 다음과 같습니다.

compilerOptions.isolatedModules 옵션은 true 로 설정됩니다. 이는 Vite가 TypeScript를 변환하기 위해 **esbuild**를 사용하며, 단일 파일 변환 제한에 영향을 받기 때문입니다. **compilerOptions.verbatimModuleSyntax** 는 **isolatedModules** 의 상위 집합이며 좋은 선택이 될 수 있습니다 - 이것은 **@vue/tsconfig** 에서 사용하는 것입니다.

옵션 API를 사용하는 경우 컴포넌트 옵션에서 this 타입 검사를 사용 하려면 **compilerOptions.strict** 를 true 로 설정해야 합니다 (또는 strict 플래그의 일부인 **compilerOptions.noImplicitThis** 활성화). 그렇지 않으면 this 는 any 로 취급 됩니다.

빌드 도구에서 resolver aliases를 설정한 경우(예: create-vue 프로젝트에서 기본적으로 설정된 @/*) 타입스크립트 **compilerOptions.paths** 설정도 필요합니다.

Vue와 함께 TSX를 사용하려는 경우, **compilerOptions.jsx** 를 "preserve" 로 설정하고, **compilerOptions.jsxImportSource** 를 "vue" 로 설정하세요.

참고항목

공식 타입스크립트 컴파일러 옵션 문서
esbuild 타입스크립트 컴파일 주의 사항

Vue CLI 및 ts-loader 에 대한 참고 사항

Vue CLI와 같은 웹팩 기반 설정에서는 ts-loader 와 같이 모듈 변환 파이프라인의 일부로 타입 검사를 수행하는 것이 일반적입니다. 그러나 이것은 타입 시스템이 타입 검사를 수행하기 위해 전체 모듈 그래프에 대한 지식을 필요로 하기 때문에 깔끔한 방법이 아닙니다. 개별 모듈의 변환 단계는 단순히 작업에 적합한 위치가 아닙니다. 다음과 같은 문제가 발생합니다.

ts-loader 는 변환 후 코드의 타입만 확인 할 수 있습니다. 이것은 소스 코드에 다시 매핑되는 IDE 또는 vue-tsc 에서 볼 수 있는 에러와 일치하지 않습니다.

타입 검사가 느릴 수 있습니다. 코드 변환이 있는 동일한 스레드/프로세스에서 수행될 경우 전체 앱의 빌드 속도에 큰 영향을 미칩니다.

우리는 이미 별도의 프로세스로 IDE에서 바로 실행 중인 타입 검사가 가능 합니다. 따라서 개발이 느려지는 경험은 좋지는 않습니다.

현재 Vue CLI를 통해 Vue 3 + TypeScript를 사용하는 경우 Vite로 마이그레이션하는 것이 좋습니다. 또한 타입 검사를 위해 vue-tsc 로 전환할 수 있도록 변환 전용 TS 지원을 활성화하는 CLI 옵션에 대해 작업하고 있습니다.

일반 사용 참고 사항

defineComponent()

타입스크립트가 컴포넌트 옵션 내에서 타입을 올바르게 추론할 수 있도록 하려면 **defineComponent()** 를 사용하여 컴포넌트를 정의해야 합니다.

ts

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

export default defineComponent({
  // 타입추론 활성화
  props: {
    name: String,
    msg: { type: String, required: true }
  },
  data() {
    return {
      count: 1
    }
  },
  mounted() {
    this.name // type: string | undefined
    this.msg // type: string
    this.count // type: number
  }
})
```

또한 defineComponent() 는 <script setup> 없이 Composition API를 사용할 때 setup() 에 전달된 props의 추론을 지원합니다.

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

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

참고사항:

- 웹팩에서 트리쉐이킹
- **defineComponent** 를 위한 타입 테스트

> **TIP**
>
> **defineComponent()** 는 일반 **JavaScript**로 정의된 컴포넌트에 대한 타입 추론을 가능하게 합니다.

싱글 파일 컴포넌트에서 사용법

SFC에서 타입스크립트를 사용하려면 <script> 태그에 lang="ts" 속성을 추가하세요. lang="ts" 가 있으면 모든 템플릿 표현식도 더 엄격한 타입 검사를 받게 됩니다.

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

export default defineComponent({
  data() {
    return {
      count: 1
    }
  }
})
</script>

<template>
  <!-- 타입 확인 및 자동완성 활성화 -->
  {{ count.toFixed(2) }}
</template>
```

lang="ts" 는 <script setup> 과 함께 사용할 수도 있습니다.

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

const count = ref(1)
</script>

<template>
  <!-- 타입 확인 및 자동완성 활성화 -->
  {{ count.toFixed(2) }}
</template>
```

템플릿에서의 타입스크립트

<template> 은 <script lang="ts"> 또는 <script setup lang="ts"> 가 사용되는 경우 바인딩 표현식에서 타입스크립트를 지원합니다. 이는 템플릿 표현식에서 타입 캐스팅을 수행해야 하는 경우에 유용합니다.

다음은 부자연스러운 예입니다.

vue

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

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

인라인 타입 캐스트로 이 문제를 해결할 수 있습니다.

vue

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

<template>
  {{ (x as number).toFixed(2) }}
</template>
```

TIP

Vue CLI 또는 웹팩 기반 설정을 사용하는 경우 타입스크립트 템플릿 표현식은 vue-loader@^16.8.0 이 필요합니다.

TSX와 함께 사용하기

Vue는 JSX/TSX로 컴포넌트 제작을 지원합니다. 자세한 내용은 **Render Function & JSX** 가이드에서 다룹니다.

Generic Components

Generic components are supported in two cases:

- In SFCs: **<script setup> with the generic attribute**
- Render function / JSX components: **defineComponent() 's function signature**

API-Specific Recipes

- **TS와 Composition API**
- **TS와 Options API**