소개

ⓘ 당신은 Vue 3 문서를 읽고 있습니다!

Vue 2 지원은 **2023년 12월 31일**에 종료되었습니다. Vue 2 EOL에 대해 자세히 알아보세요.
Vue 2에서 업그레이드하시나요? 마이그레이션 가이드를 확인하세요.

VueMastery.com에서 동영상 튜토리얼로 Vue를 배워보세요

Vue란 무엇인가요?

Vue(발음: /vju:/, **view**와 비슷함)는 사용자 인터페이스를 구축하기 위한 자바스크립트 프레임워크입니다. 표준 HTML, CSS, JavaScript 위에 구축되며, 선언적이고 컴포넌트 기반의 프로그래밍 모델을 제공하여 복잡도에 상관없이 효율적으로 사용자 인터페이스를 개발할 수 있도록 도와줍니다.

다음은 최소한의 예시입니다:

```js
import { createApp, ref } from 'vue'

createApp({
  setup() {
    return {
      count: ref(0)
    }
  }
}).mount('#app')
```

```template
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
```

```
</div>
```

결과

Count is: 0

위 예시는 Vue의 두 가지 핵심 기능을 보여줍니다:

선언적 렌더링: Vue는 표준 HTML을 확장한 템플릿 문법을 제공하여, JavaScript 상태에 따라 HTML 출력을 선언적으로 기술할 수 있게 해줍니다.

반응성: Vue는 JavaScript 상태 변화를 자동으로 추적하고, 변화가 발생하면 DOM을 효율적으로 업데이트합니다.

이미 궁금한 점이 있을 수도 있지만 걱정하지 마세요. 문서의 나머지 부분에서 모든 세부 사항을 다룰 예정입니다. 지금은 Vue가 제공하는 기능에 대해 높은 수준에서 이해할 수 있도록 계속 읽어주세요.

ⓘ 사전 지식

이후 문서는 HTML, CSS, JavaScript에 대한 기본적인 친숙함을 전제로 합니다. 프론트엔드 개발이 완전히 처음이라면, 프레임워크부터 바로 시작하는 것은 좋은 선택이 아닐 수 있습니다. 기본기를 익힌 후 다시 돌아오세요! 필요하다면 JavaScript, HTML, CSS 개요를 통해 본인의 지식 수준을 확인할 수 있습니다. 다른 프레임워크 경험이 있다면 도움이 되지만, 필수는 아닙니다.

점진적 프레임워크

Vue는 프론트엔드 개발에 필요한 대부분의 일반적인 기능을 포괄하는 프레임워크이자 생태계입니다. 하지만 웹은 매우 다양하며, 우리가 웹에서 만드는 것들은 형태와 규모가 크게 다를 수 있습니다. 이를 염두에 두고, Vue는 유연하고 점진적으로 도입할 수 있도록 설계되었습니다. 사용 사례에 따라 Vue는 다양한 방식으로 사용할 수 있습니다:

빌드 단계 없이 정적 HTML 강화
어떤 페이지에도 웹 컴포넌트로 임베딩
싱글 페이지 애플리케이션(SPA)
풀스택 / 서버 사이드 렌더링(SSR)
Jamstack / 정적 사이트 생성(SSG)
데스크탑, 모바일, WebGL, 심지어 터미널까지 타겟팅

JavaScript 지식만 있으면 따라올 수 있으며, 이 중 어느 것에도 전문가일 필요는 없습니다.

경험이 많은 개발자라면 Vue를 스택에 어떻게 통합하는 것이 최적인지, 또는 이러한 용어들이 무엇을 의미하는지 궁금할 수 있습니다. 이에 대해서는 Vue 사용 방법에서 더 자세히 다룹니다.

이처럼 유연함에도 불구하고, Vue의 동작 방식에 대한 핵심 지식은 모든 사용 사례에 공통적으로 적용됩니다. 지금은 초보자일지라도, 이 과정에서 얻은 지식은 앞으로 더 야심찬 목표에 도전할 때도 유용하게 남을 것입니다. 베테랑이라면, 해결하려는 문제에 따라 Vue를 최적으로 활용하는 방법을 선택하면서도 동일한 생산성을 유지할 수 있습니다. 이것이 Vue를 "점진적 프레임워크"라고 부르는 이유입니다: 여러분과 함께 성장하고, 필요에 맞게 적응할 수 있는 프레임워크입니다.

싱글 파일 컴포넌트

대부분의 빌드 도구 기반 Vue 프로젝트에서는 **싱글 파일 컴포넌트**(일명 `*.vue` 파일, **SFC**로 약칭)라는 HTML 유사 파일 형식을 사용하여 Vue 컴포넌트를 작성합니다. Vue SFC는 이름 그대로 컴포넌트의 로직(JavaScript), 템플릿(HTML), 스타일(CSS)을 하나의 파일에 캡슐화합니다. 아래는 앞서 본 예시를 SFC 형식으로 작성한 것입니다:

```vue
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

SFC는 Vue의 대표적인 기능이며, **빌드 설정이 필요한 경우** Vue 컴포넌트를 작성하는 권장 방식입니다. SFC의 사용 방법과 이유에 대해 더 자세히 알아볼 수 있지만, 지금은 Vue가 모든 빌드 도구 설정을 대신 처리해준다는 것만 알아두세요.

API 스타일

옵션 API

옵션 API에서는 `data`, `methods`, `mounted` 와 같은 옵션 객체를 사용하여 컴포넌트의 로직을 정의합니다. 옵션으로 정의된 속성들은 함수 내부에서 `this` 로 노출되며, 이는 컴포넌트 인스턴스를 가리킵니다:

vue

```vue
<script>
export default {
  // data()에서 반환된 속성들은 반응형 상태가 되며
  // `this`로 노출됩니다.
  data() {
    return {
      count: 0
    }
  },

  // methods는 상태를 변경하고 업데이트를 트리거하는 함수입니다.
  // 템플릿에서 이벤트 핸들러로 바인딩할 수 있습니다.
  methods: {
    increment() {
      this.count++
    }
  },

  // 라이프사이클 훅은 컴포넌트의 생명주기
  // 각 단계에서 호출됩니다.
  // 이 함수는 컴포넌트가 마운트될 때 호출됩니다.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

플레이그라운드에서 실행해보기

컴포지션 API

컴포지션 API에서는 가져온 API 함수들을 사용하여 컴포넌트의 로직을 정의합니다. SFC에서는 컴포지션 API를 `<script setup>` 과 함께 사용하는 것이 일반적입니다. `setup` 속성은 Vue가 컴파일 타임 변환을 수행하도록 힌트를 주어, 컴포지션 API를 더 적은 보일러플레이트로 사용할 수 있게 해줍니다. 예를 들어, `<script setup>` 에서 선언된 import와 최상위 변수/함수는 템플릿에서 바로 사용할 수 있습니다.

트입니다:

```vue
<script setup>
import { ref, onMounted } from 'vue'

// 반응형 상태
const count = ref(0)

// 상태를 변경하고 업데이트를 트리거하는 함수
function increment() {
  count.value++
}

// 라이프사이클 훅
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

플레이그라운드에서 실행해보기

무엇을 선택해야 할까요?

두 API 스타일 모두 일반적인 사용 사례를 완벽하게 지원할 수 있습니다. 이들은 동일한 기반 시스템 위에서 동작하는 서로 다른 인터페이스입니다. 실제로 옵션 API는 컴포지션 API 위에 구현되어 있습니다! Vue에 대한 기본 개념과 지식은 두 스타일 모두에 공통적으로 적용됩니다.

옵션 API는 "컴포넌트 인스턴스"(`this` , 예시에서 볼 수 있듯이) 개념을 중심으로 하며, OOP 언어 배경을 가진 사용자에게는 클래스 기반의 사고방식과 더 잘 맞을 수 있습니다. 또한 반응성의 세부 사항을 추상화하고 옵션 그룹을 통한 코드 구조화를 강제하여 초보자에게 더 친숙합니다.

컴포지션 API는 함수 스코프 내에서 반응형 상태 변수를 직접 선언하고, 여러 함수에서 상태를 조합하여 복잡성을 다루는 데 중점을 둡니다. 더 자유로운 형태이며, 효과적으로 사용하려면 Vue의 반응성 동작에 대한 이해가 필요합니다. 그 대신, 더 강력한 로직 구성 및 재사용 패턴을 가능하게 합니다.

두 스타일의 비교와 컴포지션 API의 잠재적 이점에 대해서는 컴포지션 API FAQ에서 더 알아볼 수 있습니다.

Vue가 처음이라면, 다음과 같이 권장합니다:

념은 두 스타일에 공통적입니다. 나중에 언제든 다른 스타일을 익힐 수 있습니다.

실제 프로젝트에서는:

> 빌드 도구를 사용하지 않거나, Vue를 주로 저복잡도 시나리오(예: 점진적 향상)에 사용할 계획이라면 옵션 API를 사용하세요.
>
> Vue로 전체 애플리케이션을 구축할 계획이라면 컴포지션 API + 싱글 파일 컴포넌트를 사용하세요.

학습 단계에서 한 가지 스타일에만 얽매일 필요는 없습니다. 이후 문서에서는 상황에 따라 두 스타일 모두의 코드 샘플을 제공하며, 왼쪽 사이드바 상단의 **API 선호도 스위치**를 통해 언제든 전환할 수 있습니다.

아직 궁금한 점이 있으신가요?

FAQ를 확인해보세요.

학습 경로 선택하기

개발자마다 학습 스타일이 다릅니다. 본인에게 맞는 학습 경로를 자유롭게 선택하세요. 물론 가능하다면 모든 내용을 한 번씩 훑어보는 것을 추천합니다!

튜토리얼 해보기

직접 실습하며 배우는 것을 선호하는 분들을 위한 경로입니다.

가이드 읽기

가이드는 프레임워크의 모든 측면을 자세히 안내합니다.

예제 살펴보기

핵심 기능과 일반적인 UI 작업 예제를 탐색해 보세요.

GitHub에서 이 페이지 편집