Vue와 웹 컴포넌트

웹 컴포넌트는 개발자가 재사용 가능한 사용자 정의 요소를 만들 수 있도록 하는 웹 네이티브 API의 집합을 가리키는 용어입니다.

Vue와 웹 컴포넌트는 주로 보완적인 기술로 간주됩니다. Vue는 사용자 정의 요소를 소비하고 생성하는 데 탁월한 지원을 제공합니다. 기존 Vue 애플리케이션에 사용자 정의 요소를 통합하거나 Vue를 사용하여 사용자 정의 요소를 빌드하고 배포하는 경우 모두 좋은 선택입니다.

Vue에서 사용자 정의 요소 사용

Vue는 **Custom Elements Everywhere** 테스트에서 완벽한 100% 점수를 받았습니다. Vue 애플리케이션에서 사용자 정의 요소를 사용하는 것은 네이티브 HTML 요소를 사용하는 것과 거의 동일하게 작동하지만 몇 가지 사항을 유념해야 합니다.

컴포넌트 해결 건너뛰기

기본적으로 Vue는 네이티브 HTML 태그가 아닌 태그를 레지스터된 Vue 컴포넌트로 해결하려고 시도한 후 사용자 정의 요소로 렌더링합니다. 개발 중에 Vue는 "컴포넌트를 해결하지 못했습니다"라는 경고 메시지를 발생시킵니다. 특정 요소를 사용자 정의 요소로 처리하고 컴포넌트 해결을 건너뛰도록 Vue에 알려주려면 **compilerOptions.isCustomElement** 옵션을 지정할 수 있습니다.

빌드 설정을 사용하는 경우 컴파일 시간 옵션인 이 옵션은 빌드 구성을 통해 전달되어야 합니다.

브라우저에서의 예시 구성

```js
// 브라우저에서 컴파일하는 경우에만 작동합니다.
// 빌드 도구를 사용하는 경우 아래 구성 예시를 참조하세요.
app.config.compilerOptions.isCustomElement = (tag) => tag.includes('-')
```

Vite에서의 예시 구성

```js
// vite.config.js
import vue from '@vitejs/plugin-vue'

export default {
  plugins: [
    vue({
      template: {
        compilerOptions: {
          // 하이픈을 포함하는 모든 태그를 사용자 정의 요소로 처리합니다.
          isCustomElement: (tag) => tag.includes('-')
        }
      }
    })
  ]
}
```

Vue CLI에서의 예시 구성

```js
// vue.config.js
module.exports = {
  chainWebpack: config => {
    config.module
      .rule('vue')
      .use('vue-loader')
      .tap(options => ({
```

```
      ...options,
      compilerOptions: {
        // ion-으로 시작하는 모든 태그를 사용자 정의 요소로 처리합니다.
        isCustomElement: tag => tag.startsWith('ion-')
      }
    }))
  }
}
```

DOM 속성 전달

DOM 속성은 항상 문자열만 될 수 있기 때문에 복잡한 데이터를 사용자 정의 요소에 전달하려면 DOM 속성으로 전달해야 합니다. Vue 3에서 사용자 정의 요소의 props를 설정할 때는 `in` 연산자를 사용하여 DOM 속성의 존재를 자동으로 확인하고, 키가 존재하는 경우 DOM 속성으로 값을 설정합니다. 대부분의 경우, 사용자 정의 요소가 권장되는 최상의 방법을 따른다면 이에 대해 생각할 필요가 없습니다.

그러나 드물게 데이터를 DOM 속성으로 전달해야 하지만 사용자 정의 요소가 속성을 제대로 정의하거나 반영하지 않는 경우가 있을 수 있습니다(이로 인해 `in` 검사가 실패함). 이 경우 `.prop` 수정자를 사용하여 `v-bind` 바인딩이 DOM 속성으로 설정되도록 강제할 수 있습니다:

template

```
<my-element :user.prop="{ name: 'jack' }"></my-element>

<!-- 축약형 표현 -->
<my-element .user="{ name: 'jack' }"></my-element>
```

Vue로 사용자 정의 요소 빌드

사용자 정의 요소의 주요 이점은 어떤 프레임워크나 프레임워크 없이 사용할 수 있다는 것입니다. 이는 최종 사용자가 동일한 프론트엔드 스택을 사용하지 않을 때 컴포넌트를 배포하는 데 이상적입니다. 또는 사용하는 컴포넌트의 구현 세부 정보를 최종 애플리케이션으로부터 격리하고 싶을 때 유용합니다.

defineCustomElement

Vue는 **defineCustomElement** 메서드를 통해 동일한 Vue 컴포넌트 API를 사용하여 사용자 정의 요소를 생성하는 것을 지원합니다. 이 메서드는 **defineComponent** 와 동일한 인수를 받지만 `HTMLElement` 를 확장하는 사용자 정의 요소 생성자

를 반환합니다:

template

```
<my-vue-element></my-vue-element>
```

js

```
import { defineCustomElement } from 'vue'

const MyVueElement = defineCustomElement({
  // 여기에 일반적인 Vue 컴포넌트 옵션을 작성합니다.
  props: {},
  emits: {},
  template: `...`,

  // defineCustomElement 전용: 쉐도우 루트에 주입될 CSS
  styles: [`/* 내부 CSS */`]
})

// 사용자 정의 요소를 등록합니다.
// 등록 후 페이지의 모든 `<my-vue-element>` 태그가 업그레이드됩니다.
customElements.define('my-vue-element', MyVueElement)

// 프로그래밍 방식으로 요소를 인스턴스화할 수도 있습니다:
```

```
// (등록 후에만 가능합니다)
document.body.appendChild(
  new MyVueElement({
    // 초기 props (선택 사항)
  })
)
```

라이프사이클

Vue 사용자 정의 요소는 첫 번째로 호출된 `connectedCallback` 에서 요소의 쉐도우 루트 내부에 Vue 컴포넌트 인스턴스를 마운트합니다.

`disconnectedCallback` 이 호출되면 Vue는 요소가 마이크로태스크 틱 이후에 문서에서 분리되었는지 여부를 확인합니다.

요소가 여전히 문서에 있는 경우 이동으로 간주되고 컴포넌트 인스턴스가 보존됩니다.

요소가 문서에서 분리된 경우 제거로 간주되고 컴포넌트 인스턴스가 언마운트됩니다.

Props

`props` 옵션을 사용하여 선언한 모든 props는 사용자 정의 요소의 속성으로 정의됩니다. Vue는 적절한 경우 속성/속성 간의 반영을 자동으로 처리합니다.

속성은 항상 해당하는 속성으로 반영됩니다.

원시 값(`string` , `boolean` 또는 `number`)을 가진 속성은 속성으로 반영됩니다.

Vue는 또한 `Boolean` 또는 `Number` 유형으로 선언된 props를 속성으로 설정할 때(속성은 항상 문자열입니다) 원하는 유형으로 자동 변환합니다. 예를 들어 다음과 같은 props 선언이 있다고 가정해 봅시다:

```js
props: {
  selected: Boolean,
  index: Number
}
```

그리고 다음과 같이 사용자 정의 요소를 사용하는 경우:

```template
<my-element selected index="1"></my-element>
```

컴포넌트에서 `selected` 는 `true` (boolean)로 변환되고 `index` 는 `1` (number)로 변환됩니다.

이벤트

`this.$emit` 또는 설정 `emit` 을 통해 발생한 이벤트는 사용자 정의 요소에서 네이티브 **CustomEvents**로 디스패치됩니다. 추가 이벤트 인수(페이로드)는 `detail` 속성으로 CustomEvent 개체의 배열로 노출됩니다.

슬롯

컴포넌트 내부에서는 슬롯을 평소처럼 `<slot/>` 요소를 사용하여 렌더링할 수 있습니다. 그러나 생성된 요소를 사용할 때는 원시 슬롯 구문만 사용할 수 있습니다:

스코프드 슬롯은 지원되지 않습니다.

이름이 있는 슬롯을 전달할 때는 `v-slot` 지시문 대신 `slot` 속성을 사용하세요:

```template
<my-element>
  <div slot="named">안녕하세요</div>
</my-element>
```

제공 및 주입

제공 및 주입 **API** 및 **Composition API** 버전도 Vue로 정의된 사용자 정의 요소 간에 작동합니다. 그러나 이는 **사용자 정의 요소 간에만 작동**합니다. 즉, Vue로 정의된 사용자 정의 요소는 비사용자 정의 요소 Vue 컴포넌트가 제공하는 속성을 주입할 수 없습니다.

SFC를 사용한 사용자 정의 요소

`defineCustomElement` 은 Vue 단일 파일 컴포넌트(SFC)에서도 작동합니다. 그러나 기본 도구 설정에서 SFC 내부의 `<style>` 태그는 여전히 프로덕션 빌드 중에 단일 CSS 파일로 추출되고 병합됩니다. 사용자 정의 요소로 SFC를 사용할 때는 주로 `<style>` 태그를 사용자 정의 요소의 쉐도우 루트에 주입하는 것이 바람직합니다.

공식적인 SFC 도구는 SFC를 "사용자 정의 요소 모드"로 가져오는 것을 지원합니다(`@vitejs/plugin-vue@^1.4.0` 또는 `vue-loader@^16.5.0` 이 필요합니다). 사용자 정의 요소 모드로 로드된 SFC는 `<style>` 태그를 CSS 문자열로 인라인화하고 컴포넌트의 `styles` 옵션으로 노출합니다. 이는 `defineCustomElement` 에서 가져와 인스턴스화될 때 요소의 쉐도우 루트에 주입됩니다.

이 모드를 사용하려면 컴포넌트 파일 이름을 `.ce.vue` 로 끝내면 됩니다:

```js
import { defineCustomElement } from 'vue'
import Example from './Example.ce.vue'

console.log(Example.styles) // ["/* inlined css */"]

// 사용자 정의 요소 생성자로 변환
const ExampleElement = defineCustomElement(Example)

// 등록
customElements.define('my-example', ExampleElement)
```

사용자 정의 요소 모드로 가져올 파일을 사용자 정의 지정하려면(예: *모든* SFC를 사용자 정의 요소로 처리), 해당 빌드 플러그인에 `customElement` 옵션을 전달할 수 있습니다:

@vitejs/plugin-vue
vue-loader

Vue 사용자 정의 요소 라이브러리 팁

Vue로 사용자 정의 요소를 빌드할 때 요소는 Vue의 런타임에 의존합니다. 사용되는 기능의 수에 따라 약 16KB의 기준 크기 비용이 발생합니다. 이는 단일 사용자 정의 요소를 배포할 때 이상적이지 않으므로 원시 JavaScript, **petite-vue** 또는 작은 런타임 크기에 특화된 프레임워크를 사용하는 것이 좋습니다. 그러나 요소를 복잡한 로직으로 구성한 컬렉션을 배포하는 경우, Vue를 사용하면 각 구성 요소를 훨씬 적은 코드로 작성할 수 있으므로 기준 크기는 충분히 정당화됩니다. 동시에 배포하는 요소가 많을수록 훌륭한 균형을 이룰 수 있습니다.

사용자 정의 요소가 Vue를 사용하는 애플리케이션에서 사용될 경우, Vue를 내장된 번들에서 외부로 분리하여 요소가 호스트 애플리케이션과 동일한 Vue 사본을 사용하도록 할 수 있습니다.

사용자가 필요에 따라 요소를 가져와 원하는 태그 이름으로 등록할 수 있도록 개별 요소 생성자를 내보내는 것이 좋습니다. 또한 모든 요소를 자동으로 등록하는 편리한 함수도 내보낼 수 있습니다. 다음은 Vue 사용자 정의 요소 라이브러리의 예시 진입점입니다:

```js
import { defineCustomElement } from 'vue'
import Foo from './MyFoo.ce.vue'
import Bar from './MyBar.ce.vue'

const MyFoo = defineCustomElement(Foo)
const MyBar = defineCustomElement(Bar)

// 개별 요소 내보내기
export { MyFoo, MyBar }

export function register() {
  customElements.define('my-foo', MyFoo)
  customElements.define('my-bar', MyBar)
}
```

요소가 많은 경우 Vite의 글로브 가져오기 또는 웹팩의 **require.context** 와 같은 빌드 도구 기능을 활용하여 디렉토리에서 모든 구성 요소를 로드할 수도 있습니다.

웹 컴포넌트와 TypeScript

애플리케이션이나 라이브러리를 개발할 때 Vue 컴포넌트를 비롯한 사용자 정의 요소를 타입 체크하려는 경우에는 추가적인 작업이 필요합니다.

사용자 정의 요소는 네이티브 API를 사용하여 전역으로 등록되기 때문에 기본적으로 Vue 템플릿에서는 타입 추론이 제공되지 않습니다. Vue 컴포넌트로 등록된 사용자 정의 요소에 대한 타입 지원을 제공하려면 Vue 템플릿 및/또는 **JSX**에서 전역 컴포넌트 타이핑을 등록할 수 있습니다.

typescript

```
import { defineCustomElement } from 'vue'

// vue SFC
import CounterSFC from './src/components/counter.ce.vue'

// 컴포넌트를 웹 컴포넌트로 변환
export const Counter = defineCustomElement(CounterSFC)

// 전역 타이핑 등록
declare module 'vue' {
  export interface GlobalComponents {
    'Counter': typeof Counter,
  }
}
```

웹 컴포넌트 vs. Vue 컴포넌트

일부 개발자는 특정 프레임워크-독점적인 컴포넌트 모델을 피해야 하며, 오직 웹 컴포넌트만을 사용하여 애플리케이션을 "미래지향적"으로 만들어야 한다고 믿고 있습니다. 이곳에서는 이러한 접근 방식이 문제에 대해 지나치게 단순한 것으로 여겨진다고 생각하는 이유를 설명하고자 합니다.

실제로 웹 컴포넌트와 Vue 컴포넌트 간에는 특정 수준의 기능 중복이 있습니다: 데이터 전달, 이벤트 발생 및 라이프사이클 관리와 같은 기능을 통해 재사용 가능한 컴포넌트를 정의할 수 있습니다. 그러나 웹 컴포넌트 API는 비교적 로우-레벨이며 기본적인 기능만 제공합니다. 실제 애플리케이션을 구축하기 위해서는 플랫폼에서 제공하지 않는 다양한 기능이 필요합니다:

선언적이고 효율적인 템플릿 시스템
상태 관리 시스템으로서의 반응성
서버에서 컴포넌트를 렌더링하고 클라이언트에서 하이드레이션하기 위한 성능 우수한 방법(SSR). 이는 SEO와 LCP와 같은 **Web Vitals** 지표에 중요합니다. 기본 웹 컴포넌트 SSR은 Node.js에서 DOM을 시뮬레이션한 다음 변경된 DOM을 직렬화하는 것을 포함합니다. 반면 Vue SSR은 가능한 경우 문자열 연결로 컴파일되므로 훨씬 효율적입니다.

Vue의 컴포넌트 모델은 이러한 요구 사항을 고려하여 설계되었습니다.

능력 있는 엔지니어링 팀이라면 기본 웹 컴포넌트 위에 동일한 기능을 구축할 수 있을 것입니다. 하지만 이는 독자적인 프레임워크의 장기적인 유지 보수 부담을 갖게되고, 성숙한 Vue와 같은 생태계 및 커뮤니티의 혜택을 잃게 됨을 의미합니다.

또한 웹 컴포넌트를 기반으로 한 프레임워크도 있지만, 이러한 프레임워크는 위에서 언급한 문제에 대한 독자적인 솔루션을 도입해야 합니다. 이러한 프레임워크를 사용하면 이러한 문제를 해결하기 위한 기술적 결정에 따르게 되며, 앞으로 발생할 수 있는 잠재적인 변화로부터 자동으로 보호받을 수 있다는 것은 확신할 수 없습니다.

웹 컴포넌트의 제한적인 측면도 있습니다:

이른 시점 슬롯 평가는 컴포넌트 구성에 제약을 가합니다. Vue의 **scoped slots**는 컴포넌트 구성을 위한 강력한 메커니즘으로, 원시 슬롯의 이른 평가 특성으로 인해 웹 컴포넌트에서는 지원할 수 없습니다. 이른 슬롯은 수신 컴포넌트가 슬롯 콘텐츠를 렌더링할 시기나 여부를 제어할 수 없음을 의미합니다.

현재 shadow DOM 범위 CSS를 사용하여 사용자 정의 요소를 배포하는 것은 JavaScript 내부에 CSS를 포함시켜 런타임에서

shadow root에 주입해야 합니다. 또한 SSR 시나리오에서 마크업에 중복된 스타일이 결과됩니다. 이에 대한 플랫폼 기능들이 작업 중이지만, 현재로서는 보편적으로 지원되지 않으며, 제작 성능 및 SSR에 대한 고려 사항이 아직 처리되어야 합니다. 한편, Vue의 SFC는 **CSS** 스코

핑 메커니즘을 제공하여 스타일을 일반 CSS 파일로 추출하는 기능을 제공합니다.

Vue는 항상 웹 플랫폼의 최신 표준을 따를 것이며, 플랫폼이 작업을 쉽게 할 수 있는 경우에는 가능한한 활용할 것입니다. 그러나 우리의 목표는 현재 잘 작동하는 솔루션을 제공하는 것입니다. 이는 표준이 여전히 부족한 부분을 채우는 것을 의미합니다.