Props

이 페이지에서는 컴포넌트 기초를 이미 읽었다고 가정합니다. 컴포넌트를 처음 사용하는 경우, 그 문서를 먼저 읽으십시오.

Props 선언

Vue 컴포넌트는 명시적인 props 선언을 요구하는데, 이렇게 함으로써 외부에서 컴포넌트에 props를 넘길 때 어떤 속성이 폴스루 속성으로 처리되어야 하는지 알 수 있습니다(전용 섹션에서 설명함).

<script setup> 스타일의 SFC에서는 defineProps() 매크로를 사용하여 props를 선언할 수 있습니다:

vue
```
<script setup>
const props = defineProps(['foo'])

console.log(props.foo)
</script>
```

<script setup> 를 사용하지 않는 컴포넌트에서 props는 **props** 옵션을 사용하여 선언합니다:

js
```
export default {
  props: ['foo'],
  setup(props) {
    // setup()은 첫 번째 인자로 props를 받습니다.
    console.log(props.foo)
  }
}
```

defineProps() 에 전달하는 인자는 props 옵션에 제공하는 값과 동일합니다. 두 선언 스타일은 동일한 props 옵션을 사용합니다.

문자열 배열을 사용하여 props를 선언하는 것 외에도 객체 선언 문법을 사용할 수도 있습니다:

js
```
// <script setup>에서
defineProps({
  title: String,
  likes: Number
})
```

js
```
// <script setup>가 아닐 때
export default {
  props: {
    title: String,
    likes: Number
  }
}
```

객체 선언 문법의 각 객체 속성의 키는 props의 이름이 되며, 객체 속성의 값은 값이 될 데이터의 타입에 해당하는 생성자 함수 (Number , String 같은)여야 합니다.

타입을 지정하는 것은 컴포넌트를 가독성이 좋게 문서화하는데 도움이 되며, 컴포넌트를 사용하는 다른 개발자가 잘못된 유형을 전달할 때에 브라우저 콘솔에 경고를 출력합니다. **prop** 유효성 검사에 대한 자세한 내용은 이 페이지 아래에서 더 자세히 설명하겠습니다.

TypeScript를 <script setup> 과 함께 사용하는 경우, 타입스크립트의 타입 표기법을 사용하여 props를 선언할 수도 있습니다:

```vue
<script setup lang="ts">
defineProps<{
  title?: string
  likes?: number
}>()
</script>
```

참고: 컴포넌트 **props**에 타입 지정하기

Props 전달에 관한 심화

Props 이름 케이싱

긴 속성명을 선언할 때 obj['kebab-case'] 와 같이 키에 따옴표를 사용하는 번거로움을 줄이기 위해, obj.camelCase 와 같이 camelCase를 사용합니다. 이렇게 선언된 속성명은 유효한 JavaScript 식별자이므로 템플릿 표현식에서 바로 참조해서 사용 할 수 있습니다:

```js
defineProps({
  greetingMessage: String
})
```

```template
<span>{{ greetingMessage }}</span>
```

기술적으로 props를 자식 컴포넌트에 전달할 때 camelCase를 사용할 수도 있습니다(**in-DOM** 템플릿 제외). 그러나 camelCase로 선언된 props 속성일지라도 관례적으로 HTML 속성 표기법과 동일하게 kebab-case로 표기해서 사용하도록 해야 합니다:

```template
<MyComponent greeting-message="안녕!" />
```

기본 엘리먼트와 Vue 컴포넌트를 쉽게 구별하여 템플릿 가독성을 향상하기 위해 되도록 컴포넌트는 **PascalCase**를 사용합니다. 그러나 props를 전달할 때 camelCase를 사용하면 실질적인 이점이 많지 않으므로 각 언어의 규칙을 따르기로 했습니다.

정적 vs. 동적 Props

지금까지는 정적인 값으로 전달된 props 예제들을 보았습니다:

```template
<BlogPost title="Vue와 함께한 나의 여행" />
```

그리고 v-bind (: : 축약어)를 사용하여 동적으로 할당된 props도 보았습니다:

```template
<!-- 변수 값을 동적으로 할당 -->
<BlogPost :title="post.title" />

<!-- 복잡한 표현식의 값을 동적으로 할당 -->
<BlogPost :title="post.title + ' by ' + post.author.name" />
```

다양한 타입의 값 전달

위의 두 가지 예제에서 문자열 값을 전달했지만, 사실 어떠한 타입의 값도 prop로 전달할 수 있습니다.

숫자

template

```
<!-- `42`는 정적이지만 Vue에 이것이 문자열이 아닌 -->
<!-- JavaScript 표현식임을 알려주려면 v-bind가 필요합니다. -->
<BlogPost :likes="42" />

<!-- 변수 값을 동적으로 할당 -->
<BlogPost :likes="post.likes" />
```

불리언

template

```
<!-- 값이 없는 prop는 `true`가 전달됩니다. -->
<BlogPost is-published />

<!-- `false`는 정적이지만 Vue에 이것이 문자열이 아닌 -->
<!-- JavaScript 표현식임을 알려주려면 v-bind가 필요합니다. -->
<BlogPost :is-published="false" />

<!-- 변수 값을 동적으로 할당 -->
<BlogPost :is-published="post.isPublished" />
```

배열

template

```
<!-- 배열이 정적이더라도 Vue에 이것이 문자열이 아닌 -->
<!-- JavaScript 표현식임을 알려주려면 v-bind가 필요합니다. -->
<BlogPost :comment-ids="[234, 266, 273]" />

<!-- 변수 값을 동적으로 할당 -->
<BlogPost :comment-ids="post.commentIds" />
```

객체

template

```
<!-- 객체가 정적이더라도 Vue에 이것이 문자열이 아닌 -->
<!-- JavaScript 표현식임을 알려주려면 v-bind가 필요합니다. -->
<BlogPost
  :author="{
    name: '신형만',
    company: '떡잎 상사'
  }"
/>

<!-- 변수 값을 동적으로 할당 -->
<BlogPost :author="post.author" />
```

객체로 여러 속성 바인딩하기

객체의 모든 속성을 props로 전달하려면 인자 없이 **v-bind** 를 사용할 수 있습니다. 예를 들어, post 객체가 주어지면:

js

```
const post = {
  id: 1,
  title: 'Vue와 함께하는 나의 여정'
}
```

다음 템플릿은:

template

```
<BlogPost v-bind="post" />
```

다음과 동일합니다:

```template
<BlogPost :id="post.id" :title="post.title" />
```

단방향 데이터 흐름

모든 props는 자식 속성과 부모 속성 사이에 **하향식 단방향 바인딩**을 형성합니다. 부모 속성이 업데이트되면 자식으로 흐르지만 그 반대는 안됩니다. 이렇게 하면 자식 컴포넌트가 실수로 부모의 상태를 변경하여 앱의 데이터 흐름을 이해하기 어렵게 만드는 것을 방지할 수 있습니다.

또한 부모 컴포넌트가 업데이트될 때마다 자식 컴포넌트의 모든 props가 최신 값으로 업데이트 됩니다. 따라서 자식 컴포넌트 내부에서 props를 변경하려 하면 **안 됩니다**. 그렇지 않을 경우, Vue는 콘솔에서 다음과 같이 경고합니다.

```js
const props = defineProps(['foo'])

// ❌ 경고, props는 읽기 전용입니다!
props.foo = 'bar'
```

일반적으로 prop을 변경하고 싶은 두 가지 경우가 있습니다:

prop은 초기 값을 전달하는 데 사용되며, 자식 컴포넌트는 나중에 이를 로컬 데이터 속성으로 사용하려고 합니다. 이 경우 prop을 초기 값으로 사용하는 로컬 데이터 속성을 정의하는 것이 가장 좋습니다:

```js
const props = defineProps(['initialCounter'])

// props.initialCounter는 counter의 초기 값으로 사용됩니다.
// 추후 props가 갱신되어도 counter 값이 업데이트 되지 않습니다.
const counter = ref(props.initialCounter)
```

prop은 변환이 필요한 원시 값으로 전달됩니다. 이 경우 prop의 값을 사용하여 계산된 속성을 정의하는 것이 가장 좋습니다:

```js
const props = defineProps(['size'])

// prop이 변경될 때, 계산된 속성은 자동으로 업데이트 됩니다.
const normalizedSize = computed(() => props.size.trim().toLowerCase())
```

객체/배열 props 변경에 관하여

객체와 배열이 props로 전달되면, 자식 컴포넌트는 바인딩된 prop을 변경할 수는 없지만, **객체 또는 배열의 중첩 속성을 변경할 수는 있습니다.** 이것은 JavaScript에서 객체와 배열이 참조로 전달되고, Vue가 이런 변경까지 방지하는 것은 너무 큰 비용이 들기 때문에 수행 하지 않습니다.

이러한 구현의 주요 단점은 자식 컴포넌트가 명확하지 않은 방식으로 부모 컴포넌트의 상태에 영향을 미쳐 잠재적으로 향후 데이터 흐름에 대한 추론을 어렵게 만든다는 것입니다. 가장 좋은 방법은 부모와 자식이 의도적으로 밀접하게 연결되어 있지 않는 한 이러한 변경을 피하는 것이며, 필요 시 자식은 부모가 변경을 수행할 수 있도록 **emit** 이벤트를 호출하는 방식으로 구현해야 합니다.

Prop 유효성 검사

앞서 봤던 것처럼 컴포넌트는 props에 타입을 지정할 수 있습니다. 지정한 요구 사항이 충족되지 않으면 Vue는 브라우저의 JavaScript 콘솔에서 경고합니다. 이것은 다른 사람들이 사용하도록 컴포넌트를 개발할 때 특히 유용합니다.

Props에 유효성 검사를 지정하려면, `defineProps()` 매크로에 문자열로 구성된 배열 대신, 유효성 검사 요구 사항이 있는 객체를 제공하면 됩니다. 예를 들어:

```js
defineProps({
  // 기본 타입 검사
  // (`null` 및 `undefined` 값은 모든 타입을 허용)
  propA: Number,
  // 여러 가지 가능한 타입
  propB: [String, Number],
  // 필수 문자열
  propC: {
    type: String,
    required: true
  },
  // 필수이지만 널 허용 문자열
  propD: {
    type: [String, null],
    required: true
  },
  // 기본값이 있는 숫자
  propE: {
    type: Number,
    default: 100
  },
  // 기본값이 있는 객체
  propF: {
    type: Object,
    // 객체 또는 배열 기본값은
    // 팩토리 함수에서 반환되어야 합니다. 이 함수는
    // 컴포넌트가 받은 원시 props를 인수로 받습니다.
    default(rawProps) {
      return { message: '안녕!' }
    }
  },
  // 사용자 정의 검증 함수
  // 3.4+ 버전에서는 전체 props가 두 번째 인수로 전달됨
  propG: {
    validator(value, props) {
      // 값은 이 문자열 중 하나와 일치해야 합니다
      return ['성공', '경고', '위험'].includes(value)
    }
  },
  // 기본값이 있는 함수
  propH: {
    type: Function,
    // 객체 또는 배열 기본값과 달리, 이는 팩토리 함수가 아닌
    // 기본값으로 사용할 함수입니다.
    default() {
      return 'Default function'
    }
  }
})
```

TIP

`defineProps()` 인자 내부의 코드는 `<script setup>` **에서 선언된 다른 변수에 접근할 수 없습니다.** 컴파일할 때 전체 표현식이 외부 함수 범위로 이동되기 때문입니다.

추가 세부 사항:

- `required: true` 가 지정되지 않은 모든 prop은 기본적으로 선택 사항(optional)입니다.
- prop의 타입이 `Boolean` 이 아니고 선택사항인 경우, 누락되면 `undefined` 값을 가집니다.
- prop의 타입이 `Boolean` 이고 누락된 경우, `false` 가 기본값이 됩니다. 의도에 따라서 `default` 값 정의가 필요할 수 있습니다.

prop이 누락되었거나 명시적으로 선언된 값이 undefined 이고 default 값이 정의되어 있다면, default 값이 사용됩니다.

prop 유효성 검사에 실패하면 Vue는 콘솔에 경고를 출력합니다. (개발 빌드를 사용하는 경우).

타입 기반 **props** 선언 시, Vue는 타입 문법을 적절한 prop 선언으로 컴파일하기 위해 최선을 다할 것입니다. 예를 들어 defineProps<{ msg: string }> 는 { msg: { type: String, required: true }} 로 컴파일됩니다.

실행 간 타입 체크

type 은 다음 기본 생성자 중 하나일 수 있습니다:

- String
- Number
- Boolean
- Array
- Object
- Date
- Function
- Symbol
- Error

또한 type 은 사용자 정의 클래스 또는 생성자 함수일 수도 있으며, instanceof 를 사용하여 검사를 실시합니다. 예를 들어, 다음 클래스가 주어졌을 때:

```js
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }
}
```

props 타입으로 사용할 수 있습니다:

```js
defineProps({
  author: Person
})
```

Vue는 ``instanceof Person 를 사용하여 author prop의 값이 실제로 Person` 클래스의 인스턴스인지 확인합니다.

Nullable Type

타입이 필수이지만 널 허용인 경우, null 을 포함한 배열 구문을 사용할 수 있습니다:

```js
defineProps({
  id: {
    type: [String, null],
    required: true
  }
})
```

type 이 배열 구문 없이 null 만인 경우, 모든 타입을 허용한다는 점에 유의하세요.

불리언 캐스팅

Boolean 타입의 props는 네이티브 불리언 속성의 동작을 모방하는 특별한 캐스팅 규칙이 있습니다. 다음 선언과 함께 <MyComponent> 가 제공됩니다:

```js
defineProps({
  disabled: Boolean
})
```

컴포넌트를 다음처럼 사용할 수 있습니다:

```template
<!-- :disabled="true" 같음 -->
<MyComponent disabled />

<!-- :disabled="false" 같음 -->
<MyComponent />
```

여러 타입을 허용하도록 프롭(prop)을 선언할 때, Boolean 에 대한 캐스팅 규칙도 적용됩니다. 그러나 String 과 Boolean 이 모두 허용되는 경우에는 주의해야 합니다 - Boolean 캐스팅 규칙은 String보다 앞에 위치해야만 적용됩니다:

```js
// disabled는 true로 캐스팅됩니다
defineProps({
  disabled: [Boolean, Number]
})

// disabled는 true로 캐스팅됩니다
defineProps({
  disabled: [Boolean, String]
})

// disabled는 true로 캐스팅됩니다
defineProps({
  disabled: [Number, Boolean]
})

// disabled는 빈 문자열로 파싱됩니다 (disabled="")
defineProps({
  disabled: [String, Boolean]
})
```