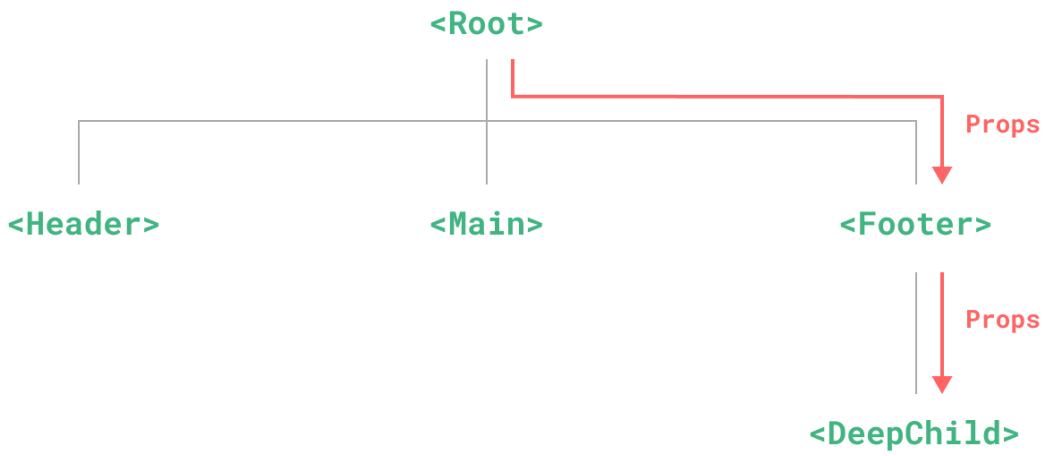
Provide / Inject

> 이 페이지는 이미 컴포넌트 기본을 읽었다고 가정합니다. 컴포넌트가 처음이라면 먼저 해당 내용을 읽어보세요.

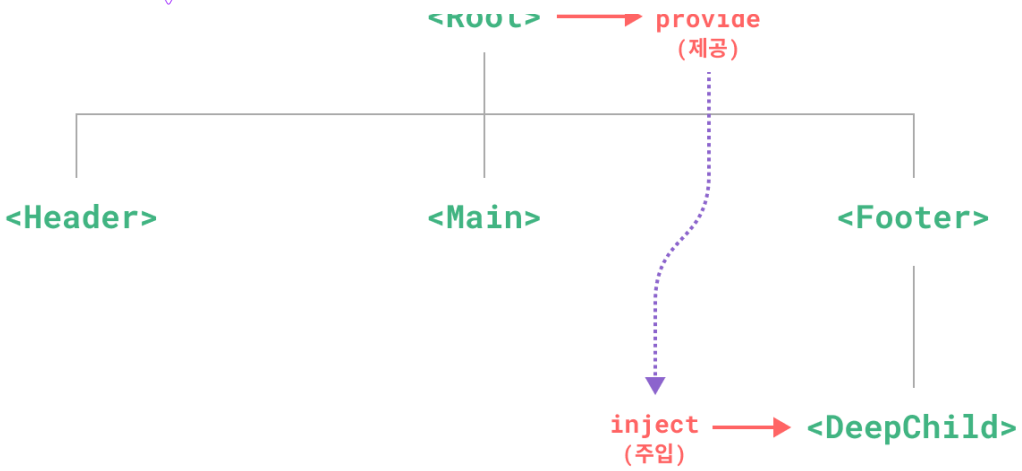
Prop Drilling

일반적으로 부모에서 자식 컴포넌트로 데이터를 전달할 때는 props를 사용합니다. 하지만, 큰 컴포넌트 트리에서 깊이 중첩된 컴포넌트가 먼 조상 컴포넌트의 무언가가 필요하다고 상상해보세요. props만으로는 동일한 prop을 전체 부모 체인에 걸쳐 전달해야 합니다:



`<Footer>` 컴포넌트가 이러한 prop에 전혀 관심이 없더라도, `<DeepChild>` 가 접근할 수 있도록 prop을 선언하고 전달해야 한다는 점에 주목하세요. 부모 체인이 더 길어진다면 더 많은 컴포넌트가 영향을 받게 됩니다. 이를 "props drilling"이라고 하며, 확실히 다루기 번거롭습니다.

`provide` 와 `inject` 를 사용하면 props drilling 문제를 해결할 수 있습니다. 부모 컴포넌트는 모든 자손을 위한 **의존성 제공자** 역할을 할 수 있습니다. 자손 트리 내의 어떤 컴포넌트든, 깊이에 상관없이 부모 체인 상단의 컴포넌트가 제공한 의존성을 **주입**할 수 있습니다.



Provide

컴포넌트의 자손에게 데이터를 제공하려면 `provide()` 함수를 사용하세요:

```vue
<script setup>
import { provide } from 'vue'

provide(/* key */ 'message', /* value */ 'hello!')
</script>
```

`<script setup>` 을 사용하지 않는 경우, `provide()` 는 반드시 `setup()` 내부에서 동기적으로 호출되어야 합니다:

```js
import { provide } from 'vue'

export default {
  setup() {
    provide(/* key */ 'message', /* value */ 'hello!')
  }
}
```

`provide()` 함수는 두 개의 인자를 받습니다. 첫 번째 인자는 **주입 키**(injection key)로, 문자열 또는 `Symbol` 이 될 수 있습니다. 주입 키는 자손 컴포넌트가 주입할 값을 찾는 데 사용됩니다. 하나의 컴포넌트는 서로 다른 주입 키로 여러 번 `provide()` 를 호출하여 다양한 값을 제공할 수 있습니다.

두 번째 인자는 제공할 값입니다. 값은 ref와 같은 반응형 상태를 포함하여 어떤 타입이든 될 수 있습니다:

```
const count = ref(0)
provide('key', count)
```

반응형 값을 제공하면, 제공된 값을 사용하는 자손 컴포넌트가 제공자 컴포넌트와 반응형 연결을 맺을 수 있습니다.

App-level Provide

컴포넌트에서 데이터를 제공하는 것 외에도, 앱 레벨에서 제공할 수도 있습니다:

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

const app = createApp({})

app.provide(/* key */ 'message', /* value */ 'hello!')
```

앱 레벨에서 제공한 값은 앱에서 렌더링되는 모든 컴포넌트에서 사용할 수 있습니다. 이는 플러그인을 작성할 때 특히 유용합니다. 플러그인은 일반적으로 컴포넌트를 통해 값을 제공할 수 없기 때문입니다.

Inject

조상 컴포넌트가 제공한 데이터를 주입하려면 `inject()` 함수를 사용하세요:

```vue
<script setup>
import { inject } from 'vue'

const message = inject('message')
</script>
```

여러 부모가 동일한 키로 데이터를 제공하는 경우, inject는 컴포넌트의 부모 체인에서 가장 가까운 부모의 값을 사용합니다.

제공된 값이 ref인 경우, 해당 값은 그대로 주입되며 **자동으로 언래핑되지 않습니다**. 이를 통해 주입자 컴포넌트가 제공자 컴포넌트와의 반응형 연결을 유지할 수 있습니다.

마찬가지로, `<script setup>` 을 사용하지 않는 경우 `inject()` 는 반드시 `setup()` 내부에서 동기적으로 호출되어야 합니다:

```js
import { inject } from 'vue'

export default {
  setup() {
    const message = inject('message')
    return { message }
  }
}
```

Injection Default Values

기본적으로, `inject` 는 주입된 키가 부모 체인 어딘가에서 제공된다고 가정합니다. 만약 키가 제공되지 않은 경우 런타임 경고가 발생합니다.

주입된 프로퍼티가 선택적 제공자와 함께 동작하도록 하려면, props와 유사하게 기본값을 선언해야 합니다:

```js
// "message"에 해당하는 데이터가 제공되지 않았다면
// `value`는 "default value"가 됩니다
const value = inject('message', 'default value')
```

경우에 따라 기본값을 함수 호출이나 새 클래스 인스턴스화로 생성해야 할 수도 있습니다. 선택적 값이 사용되지 않을 때 불필요한 연산이나 부작용을 피하려면, 기본값 생성에 팩토리 함수를 사용할 수 있습니다:

```js
const value = inject('key', () => new ExpensiveClass(), true)
```

세 번째 인자는 기본값을 팩토리 함수로 처리해야 함을 나타냅니다.

Working with Reactivity

반응형 provide / inject 값을 사용할 때는, **가능하다면 반응형 상태의 모든 변경을 제공자 내부에서만 처리하는 것이 좋습니다**. 이렇게 하면 제공된 상태와 그 변이 로직이 동일 컴포넌트에 위치하게 되어, 향후 유지보수가 쉬워집니다.

는 함수를 함께 제공하는 것을 권장합니다:

```vue
<!-- 제공자 컴포넌트 내부 -->
<script setup>
import { provide, ref } from 'vue'

const location = ref('North Pole')

function updateLocation() {
  location.value = 'South Pole'
}

provide('location', {
  location,
  updateLocation
})
</script>
```

```vue
<!-- 주입자 컴포넌트에서 -->
<script setup>
import { inject } from 'vue'

const { location, updateLocation } = inject('location')
</script>

<template>
  <button @click="updateLocation">{{ location }}</button>
</template>
```

마지막으로, `provide` 를 통해 전달되는 데이터가 주입자 컴포넌트에서 변경되지 않도록 하려면 `readonly()` 로 감쌀 수 있습니다.

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

const count = ref(0)
provide('read-only-count', readonly(count))
</script>
```

Working with Symbol Keys

지금까지 예제에서는 문자열 주입 키를 사용했습니다. 많은 의존성 제공자가 있는 대규모 애플리케이션을 개발하거나, 다른 개발자가 사용할 컴포넌트를 작성하는 경우, 잠재적 충돌을 피하기 위해 Symbol 주입 키를 사용하는 것이 가장 좋습니다.

keys.js

```js
export const myInjectionKey = Symbol()
```

```js
// 제공자 컴포넌트에서
import { provide } from 'vue'
import { myInjectionKey } from './keys.js'

provide(myInjectionKey, {
  /* 제공할 데이터 */
})
```

```js
// 주입자 컴포넌트에서
import { inject } from 'vue'
import { myInjectionKey } from './keys.js'

const injected = inject(myInjectionKey)
```

참고: Provide / Inject 타입 지정 TS

GitHub에서 이 페이지 편집