Suspense

실험적인 기능

<Suspense> 는 실험적인 기능입니다. 아직 개발이 완료된 안정된 상태가 아니며 추후 API가 변경 될 수 있습니다.

<Suspense> 는 컴포넌트 트리에서 비동기 의존성을 조정하기 위한 내장 컴포넌트입니다. 컴포넌트 트리 아래에 여러개의 중첩된 비동기 의존성이 해결될 때까지 기다리는 동안 로드 상태를 렌더링할 수 있습니다.

비동기 의존성

<Suspense> 가 해결하려는 문제와 비동기 의존성이 상호 작용하는 방식을 설명하기 위해 다음과 같은 컴포넌트 계층 구조를 그려보겠습니다.

```
<Suspense>
└── <Dashboard>
  ├── <Profile>
  │  └── <FriendStatus> (비동기 setup()과 컴포넌트)
  └── <Content>
    ├── <ActivityFeed> (비동기 컴포넌트)
    └── <Stats> (비동기 컴포넌트)
```

컴포넌트 트리에는 확인할 비동기 리소스에 따라 렌더링이 달라지는 여러 중첩 컴포넌트가 있습니다. <Suspense> 가 없으면 각각 로딩/에러 및 로딩 상태를 처리해야 합니다. 최악의 시나리오에서는 페이지에 3개의 로딩 스피너가 표시되고 컨텐츠가 다른 시간에 표시될 수 있습니다.

<Suspense> 컴포넌트는 이러한 중첩된 비동기 의존성이 해결될 때까지 최상위에서 로드/에러 상태를 표시할 수 있는 기능을 제공합니다.

<Suspense> 가 기다릴 수 있는 두 가지 유형의 비동기 의존성이 있습니다.

비동기 setup() 훅이 있는 컴포넌트. 여기에는 최상위 await 표현식과 함께 <script setup> 을 사용하는 컴포넌트가 포함됩니다.

비동기 컴포넌트.

비동기 setup()

컴포지션 API 컴포넌트의 setup() 훅은 비동기일 수 있습니다.

```js
export default {
  async setup() {
    const res = await fetch(...)
    const posts = await res.json()
    return {
      posts
    }
  }
}
```

<script setup> 을 사용하는 경우 최상위 await 표현식이 있으면 컴포넌트가 자동으로 비동기 의존성을 갖게 됩니다.

```vue
<script setup>
const res = await fetch(...)
```

```
const posts = await res.json()
</script>

<template>
 {{ posts }}
</template>
```

비동기 컴포넌트

비동기 컴포넌트는 기본적으로 **중단** 가능합니다. 즉, 상위 체인에 `<Suspense>` 가 있는 경우 해당 `<Suspense>` 의 비동기 의존성으로 처리됩니다. 이 경우 로딩 상태는 `<Suspense>` 에 의해 제어되며 컴포넌트 자체의 로딩, 에러, 지연 및 시간 초과 옵션은 무시됩니다.

비동기 컴포넌트는 `Suspense` 제어를 선택 해제하고 컴포넌트 옵션에서 `suspensible: false` 를 지정하여 항상 자체 로딩 상태를 제어하도록 할 수 있습니다.

로딩 상태

`<Suspense>` 컴포넌트에는 `#default` 와 `#fallback` 이라는 두 개의 슬롯이 있습니다. 두 슬롯 모두 **하나** 의 직계 자식 노드만 허용합니다. 가능한 경우 기본 슬롯의 노드가 표시됩니다. 그렇지 않은 경우 대체 슬롯의 노드가 대신 표시됩니다.

template

```
<Suspense>
 <!-- 컴포넌트와 중첩된 비동기 의존성 -->
 <Dashboard />

 <!-- #fallback 슬롯을 통한 로딩 상태 -->
 <template #fallback>
  로딩중...
 </template>
</Suspense>
```

초기 렌더링 시 `<Suspense>` 는 기본 슬롯 컨텐츠를 메모리에 렌더링합니다. 프로세스 중에 비동기 의존성이 발생하면 **pending** 상태가 됩니다. pending 상태 동안 대체 컨텐츠가 표시됩니다. 발생한 모든 비동기 의존성이 해결되면 `<Suspense>` 가 **resolve** 된 상태로 들어가고 resolved된 기본 슬롯 컨텐츠가 표시됩니다.

초기 렌더링 중에 비동기 의존성이 발생하지 않은 경우 `<Suspense>` 는 직접 resolved된 상태가 됩니다.

resolved 상태에서 `<Suspense>` 는 `#default` 슬롯의 루트 노드가 교체되는 경우에만 pending 상태로 되돌아갑니다. 트리 내부의 중첩된 새로운 비동기 의존성으로 인해 `<Suspense>` 가 pending 상태로 **되돌아가지 않습니다** .

되돌리기가 발생하면 대체 컨텐츠가 즉시 표시되지 않습니다. 대신 `<Suspense>` 는 새 컨텐츠와 해당 비동기 의존성이 해결될 때까지 기다리는 동안 이전 `#default` 컨텐츠를 표시합니다. 이 동작은 `timeout` prop으로 구성할 수 있습니다. 새로운 기본 컨텐츠를 렌더링하는데 `timeout` 보다 오래 걸리는 경우 `<Suspense>` 가 대체 컨텐츠로 전환됩니다. `timeout` 값이 `0` 이면 기본 컨텐츠가 교체될 때 대체 컨텐츠가 즉시 표시됩니다.

이벤트

`<Suspense>` 컴포넌트는 3가지 이벤트를 내보냅니다: `pending` , `resolve` 및 `fallback` 입니다. `pending` 이벤트는 보류 상태에 들어갈 때 발생합니다. `resolve` 이벤트는 새 컨텐츠가 `default` 슬롯에서 해결되면 발생합니다. `fallback` 슬롯의 컨텐츠가 표시될 때 `fallback` 이벤트가 시작됩니다.

예를 들어 이벤트를 사용하여 새로운 컴포넌트가 로드되는 동안 이전 DOM 앞에 로딩을 표시할 수 있습니다.

에러 처리

<Suspense> 는 현재 컴포넌트 자체를 통해 에러를 처리하지 않습니다. 그러나 **errorCaptured** 옵션 또는 **onErrorCaptured()** 훅을 사용하여 <Suspense> 의 부모 컴포넌트에서 비동기 에러를 캡처하고 처리할 수 있습니다.

다른 컴포넌트와 결합

<Transition> 및 **<KeepAlive>** 컴포넌트와 함께 <Suspense> 를 사용하려는 경우가 일반적입니다. 이러한 컴포넌트의 중첩 순서는 모든 컴포넌트가 올바르게 작동하도록 하는 것에 큰 영향을 줍니다.

또한 이러한 컴포넌트는 **Vue Router** 의 <RouterView> 컴포넌트와 함께 사용되는 경우가 많습니다.

다음 예제에서는 이러한 컴포넌트가 모두 예상한 대로 작동하도록 중첩시키는 방법을 보여줍니다. 더 간단하게 할 경우 필요하지 않은 컴포넌트를 제거할 수 있습니다.

template

```
<RouterView v-slot="{ Component }">
  <template v-if="Component">
    <Transition mode="out-in">
      <KeepAlive>
        <Suspense>
          <!-- 메인 컨텐츠 -->
          <component :is="Component"></component>

          <!-- 로딩 상태 -->
          <template #fallback>
            로딩중...
          </template>
        </Suspense>
      </KeepAlive>
    </Transition>
  </template>
</RouterView>
```

Vue 라우터에는 동적 import를 사용하여 컴포넌트를 **lazy load**하는 기능이 내장되어 있습니다. 이들은 비동기 컴포넌트와 구분되며 현재 <Suspense> 를 트리거하지 않습니다. 그러나 여전히 비동기 컴포넌트를 하위 항목으로 가질 수 있으며 일반적인 방식으로 <Suspense> 를 트리거할 수 있습니다.

중첩 Suspense

여러 비동기 컴포넌트(중첩 또는 레이아웃 기반 라우트에 일반적임)를 다음과 같이 가질 때:

template

```
<Suspense>
  <component :is="DynamicAsyncOuter">
    <component :is="DynamicAsyncInner" />
  </component>
</Suspense>
```

<Suspense> 는 예상대로 트리 아래의 모든 비동기 컴포넌트를 해결하는 경계를 생성합니다. 그러나 DynamicAsyncOuter 를 변경하면 <Suspense> 가 올바르게 대기하지만, DynamicAsyncInner 를 변경하면 중첩된 DynamicAsyncInner 가 해결될 때까지 이전 노드나 폴백 슬롯 대신 빈 노드를 렌더링합니다.

이 문제를 해결하기 위해서는, 중첩된 컴포넌트를 위한 패치를 처리하기 위해 중첩 suspense를 가질 수 있습니다:

template

```
<Suspense>
  <component :is="DynamicAsyncOuter">
    <Suspense suspensible> <!-- 이것 -->
      <component :is="DynamicAsyncInner" />
    </Suspense>
  </component>
</Suspense>
```

`suspensible` 속성을 설정하지 않으면, 내부 `<Suspense>` 는 부모 `<Suspense>` 에 의해 동기 컴포넌트로 취급됩니다. 이는 자체 폴백 슬롯을 가지며, 두 `Dynamic` 컴포넌트가 동시에 변경되면 자식 `<Suspense>` 가 자체 종속성 트리를 로드하는 동안 빈 노드와 여러 패치 사이클이 발생할 수 있음을 의미하며, 이는 바람직하지 않을 수 있습니다. `suspensible` 이 설정되면 모든 비동기 종속성 처리는 부모 `<Suspense>` 에 의해 처리되며(이벤트 발행 포함), 내부 `<Suspense>` 는 종속성 해결 및 패칭을 위한 또 다른 경계 역할만 합니다.

관련 문서

- **`<Suspense>` API** 참고