<Fragment> (<>...</>)

`<Fragment>`, often used via `<>...</>` syntax, lets you group elements without a wrapper node.

Canary

Fragments can also accept refs, which enable interacting with underlying DOM nodes without adding wrapper elements. See reference and usage below.

```
<>
  <OneChild />
  <AnotherChild />
</>
```

레퍼런스

`<Fragment>`

하나의 엘리먼트가 필요한 상황에서 엘리먼트를 `<Fragment>` 로 감싸서 그룹화하세요. `Fragment` 안에서 그룹화된 엘리먼트는 DOM 결과물에 영향을 주지 않습니다. 즉, 엘리먼트가 그룹화되지 않은 것과 같습니다. 대부분의 경우 빈 JSX 태그인 `<></>` 는 `<Fragment></Fragment>` 의 줄임말입니다.

Props

- **optional** `key` : 명시적 `<Fragment>` 로 선언된 Fragment에는 `key` 를 사용할 수 있습니다.
- **optional** `key` : Fragments declared with the explicit `<Fragment>` syntax may have keys.
- Canary only **optional** `ref` : A ref object (e.g. from `useRef`) or callback function. React provides a `FragmentInstance` as the ref value that implements methods for interacting with the DOM nodes wrapped by the Fragment.

Canary only FragmentInstance

When you pass a ref to a fragment, React provides a `FragmentInstance` object with methods for interacting with the DOM nodes wrapped by the fragment:

Event handling methods:

- `addEventListener(type, listener, options?)` : Adds an event listener to all first-level DOM children of the Fragment.
- `removeEventListener(type, listener, options?)` : Removes an event listener from all first-level DOM children of the Fragment.
- `dispatchEvent(event)` : Dispatches an event to a virtual child of the Fragment to call any added listeners and can bubble to the DOM parent.

Layout methods:

- `compareDocumentPosition(otherNode)` : Compares the document position of the Fragment with another node.

  - If the Fragment has children, the native `compareDocumentPosition` value is returned.
  - Empty Fragments will attempt to compare positioning within the React tree and include `Node.DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC`.
  - Elements that have a different relationship in the React tree and DOM tree due to portaling or other insertions are `Node.DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC`.
- `getClientRects()`: Returns a flat array of `DOMRect` objects representing the bounding rectangles of all children.
- `getRootNode()`: Returns the root node containing the Fragment's parent DOM node.

Focus management methods:

- `focus(options?)`: Focuses the first focusable DOM node in the Fragment. Focus is attempted on nested children depth-first.
- `focusLast(options?)`: Focuses the last focusable DOM node in the Fragment. Focus is attempted on nested children depth-first.
- `blur()`: Removes focus if `document.activeElement` is within the Fragment.

Observer methods:

- `observeUsing(observer)`: Starts observing the Fragment's DOM children with an IntersectionObserver or ResizeObserver.
- `unobserveUsing(observer)`: Stops observing the Fragment's DOM children with the specified observer.
- Fragment에 `key`를 사용하려면 `<>...</>` 구문을 사용할 수 없습니다. 명시적으로 `react`에서 `Fragment`를 불러오고Import `<Fragment key={yourKey}>...</Fragment>`를 렌더링해야 합니다.
- React는 `<><Child /></>`에서 `[<Child />]`로 렌더링하거나 (또는 반대의 경우), 혹은 `<><Child /></>`에서 `<Child />` 렌더링하거나 (또는 반대의 경우) State를 초기화하지 않습니다. 이는 오직 한 단계 깊이Single Level Deep까지만 적용됩니다. 예를 들어 `<><><Child /></></>`에서 `<Child />`로 렌더링하는 것은 State가 초기화됩니다. 정확한 의미는 여기서 확인할 수 있습니다.
- **Canary only** If you want to pass `ref` to a Fragment, you can't use the `<>...</>` syntax. You have to explicitly import `Fragment` from `'react'` and render `<Fragment ref={yourRef}>...</Fragment>`.

사용법

여러 엘리먼트 반환하기

여러 엘리먼트를 함께 그룹화하기 위해 `Fragment` 나 `<>...</>` 문법을 사용하세요. 한 개의 엘리먼트가 존재할 수 있는 곳에 여러 엘리먼트를 넣을 수 있습니다. 예를 들어 컴포넌트는 한 개의 엘리먼트만 반환할 수 있지만 `Fragment` 를 사용하여 여러 엘리먼트를 함께 그룹화하여 반환할 수 있습니다.

```
function Post() {
  return (
    <>
      <PostTitle />
      <PostBody />
    </>
  );
}
```

`Fragment` 로 엘리먼트를 그룹화하면 DOM 엘리먼트와 같은 다른 컨테이너로 엘리먼트를 감싸는 경우와는 달리, 레이아웃이나 스타일에 영향을 주지 않기 때문에 `Fragment` 는 효과적입니다. 브라우저로 아래 예시를 검사하면 모든 `<h1>` , `<article>` DOM 노드가 래퍼 없이 형제 노드로 나타나는 것을 볼 수 있습니다.

App.js

다운로드 새로고침 Clear 포크

```
export default function Blog() {
  return (
    <>
      <Post title="An update" body="It's been a while since I posted..." />
      <Post title="My new blog" body="I am starting a new blog!" />
    </>
  )
}

function Post({ title, body }) {
  return (
```

자세히 보기

자세히 살펴보기

특별한 문법 없이 `Fragment` 를 작성하는 방법은 무엇인가요?

자세히 보기

변수에 여러 엘리먼트 할당

다른 엘리먼트와 마찬가지로 `Fragment` 를 변수에 할당하고 Props로 전달하는 등의 작업을 할 수 있습니다.

```
function CloseDialog() {
  const buttons = (
    <>
```

```
      <OKButton />
      <CancelButton />
    </>
  );
  return (
    <AlertDialog buttons={buttons}>
      Are you sure you want to leave this page?
    </AlertDialog>
  );
}
```

텍스트와 함께 엘리먼트 그룹화

`Fragment` 를 사용하여 텍스트를 컴포넌트와 함께 그룹화할 수 있습니다.

```
function DateRangePicker({ start, end }) {
  return (
    <>
      From
      <DatePicker date={start} />
      to
      <DatePicker date={end} />
    </>
  );
}
```

`Fragment` 리스트 렌더링

`<></>` 문법을 사용하는 대신 명시적으로 `Fragment` 를 작성해야 하는 상황이 있습니다. 반복을 통해 여러 엘리먼트를 렌더링할 때 각 요소에 `key` 를 할당해야 합니다. 반복 안에 엘리먼트가 `Fragment` 인 경우 `key` 속성을 제공하기 위해 일반 JSX 엘리먼트 문법을 사용해야 합니다.

```
function Blog() {
  return posts.map(post =>
    <Fragment key={post.id}>
```

```
      <PostTitle title={post.title} />
      <PostBody body={post.body} />
    </Fragment>
  );
}
```

DOM을 검사하여 `Fragment` 자식 주위에 래퍼 엘리먼트가 없는 것을 확인할 수 있습니다.

App.js

다운로드 새로고침 Clear 포크

```
import { Fragment } from 'react';

const posts = [
  { id: 1, title: 'An update', body: "It's been a while since I posted..." },
  { id: 2, title: 'My new blog', body: 'I am starting a new blog!' }
];

export default function Blog() {
  return posts.map(post =>
    <Fragment key={post.id}>
      <PostTitle title={post.title} />
      <PostBody body={post.body} />
```

자세히 보기

Canary only Using Fragment refs for DOM interaction

Fragment refs allow you to interact with the DOM nodes wrapped by a Fragment without adding extra wrapper elements. This is useful for event handling, visibility tracking, focus management, and replacing deprecated patterns like `ReactDOM.findDOMNode()`.

```
import { Fragment } from 'react';

function ClickableFragment({ children, onClick }) {
  return (
    <Fragment ref={fragmentInstance => {
      fragmentInstance.addEventListener('click', handleClick);
      return () => fragmentInstance.removeEventListener('click', handleClick);
    }}>
      {children}
    </Fragment>
  );
}
```

Canary only Tracking visibility with Fragment refs

Fragment refs are useful for visibility tracking and intersection observation. This enables you to monitor when content becomes visible without requiring the child Components to expose refs:

```
import { Fragment, useRef, useLayoutEffect } from 'react';

function VisibilityObserverFragment({ threshold = 0.5, onVisibilityChange, children }) {
  const fragmentRef = useRef(null);

  useLayoutEffect(() => {
    const observer = new IntersectionObserver(
      (entries) => {
        onVisibilityChange(entries.some(entry => entry.isIntersecting))
      },
      { threshold }
```

```
    );

    fragmentRef.current.observeUsing(observer);
    return () => fragmentRef.current.unobserveUsing(observer);
  }, [threshold, onVisibilityChange]);

  return (
    <Fragment ref={fragmentRef}>
      {children}
    </Fragment>
  );
}

function MyComponent() {
  const handleVisibilityChange = (isVisible) => {
    console.log('Component is', isVisible ? 'visible' : 'hidden');
  };

  return (
    <VisibilityObserverFragment onVisibilityChange={handleVisibilityChange}>
      <SomeThirdPartyComponent />
      <AnotherComponent />
    </VisibilityObserverFragment>
  );
}
```

This pattern is an alternative to Effect-based visibility logging, which is an anti-pattern in most cases. Relying on Effects alone does not guarantee that the rendered Component is observable by the user.

Canary only Focus management with Fragment refs

Fragment refs provide focus management methods that work across all DOM nodes within the Fragment:

```
import { Fragment, useRef } from 'react';

function FocusFragment({ children }) {
  return (
```

```
      <Fragment ref={(fragmentInstance) => fragmentInstance?.focus()}>
        {children}
      </Fragment>
    );
  }
```

The `focus()` method focuses the first focusable element within the Fragment, while `focusLast()` focuses the last focusable element.

이전
컴포넌트

다음
<Profiler>

Meta Open Source

uwu?

React 학습하기

- 빠르게 시작하기
- 설치하기
- UI 표현하기
- 상호작용성 더하기
- State 관리하기
- 탈출구

API 참고서

- React APIs
- React DOM APIs

커뮤니티

- 행동 강령
- 팀 소개
- 문서 기여자
- 감사의 말

더 보기

- 블로그
- React Native
- 개인 정보 보호
- 약관