<Suspense>

<Suspense> 는 자식 요소를 로드하기 전까지 화면에 대체 UIFallback를 보여줍니다.

```
<Suspense fallback={<Loading />}>
  <SomeComponent />
</Suspense>
```


레퍼런스

<Suspense>

Props

- `children` : 궁극적으로 렌더링하려는 실제 UI입니다. `children` 의 렌더링이 지연되면, Suspense는 `fallback` 을 대신 렌더링합니다.
- `fallback` : 실제 UI가 로딩되기 전까지 대신 렌더링되는 대체 UI입니다. 올바른 React 노드 형식은 무엇이든 대체 UI로 활용할 수 있지만, 실제로는 보통 로딩 스피너나 스켈레톤처럼 간단한 Placeholder를 활용합니다. Suspense는 `children` 의 렌더링이 지연되면 자동으로 `fallback` 으로 전환하고, 데이터가 준비되면 `children` 으로 다시 전환합니다. 만약 `fallback` 의 렌더링이 지연되면, 가장 가까운 부모 Suspense가 활성화됩니다.

주의 사항

- React는 컴포넌트가 처음으로 마운트 되기 전에 지연된 렌더링을 하는 동안의 어떤 State도 유지하지 않습니다. 컴포넌트가 로딩되면 React는 일시 중지된 트리를 처음부터 다시 렌더링 합니다.
- Suspense가 트리의 콘텐츠를 보여주고 있을 때 또 다시 지연되면 `startTransition` 나 `useDeferredValue` 로 인한 업데이트가 아닌 한, `fallback` 이 다시 보입니다.
- React가 다시 일시 중지되어 보이는 콘텐츠를 숨겨야 하는 경우, 콘텐츠 트리에서 Layout Effect들을 정리합니다. 콘텐츠가 다시 보일 준비가 되면 React는 Layout Effect들을 다시 실행합니다. 이로써 DOM 레이아웃을 측정하는 Effect가 콘텐츠가 숨겨져 있는 동안 동작하지 않도록 보장합니다.
- React는 Suspense와 통합된 *스트리밍 서버 렌더링*과 *선택적 Hydration* 같은 내부 최적화를 포함하고 있습니다. 아키텍처 개요를 읽고 기술 강연을 시청하여 더 자세히 알아보세요.

사용법

콘텐츠를 로딩하는 동안 대체 UI[Fallback] 보여주기

애플리케이션의 모든 곳을 Suspense 경계로 감쌀 수 있습니다.

```
<Suspense fallback={<Loading />}>
  <Albums />
</Suspense>
```

React는 `children` 에 필요한 모든 코드와 데이터를 로딩할 때까지 `loading fallback` 을 보여줍니다.

아래 예시에서는 앨범 목록을 가져오는 동안 `Albums` 컴포넌트가 *지연(Suspend)*됩니다. 렌더링할 준비가 될 때까지 가장 가까운 Suspense는 Fallback, 즉 `Loading` 컴포넌트를 표시합니다. 데이터를 모두 로딩하면 React는 `Loading` Fallback을 숨기고 로딩된 데이터로 `Albums` 컴포넌트를 렌더링합니다.

ArtistPage.js Albums.js 새로고침 Clear 포크

```
import { Suspense } from 'react';
import Albums from './Albums.js';

export default function ArtistPage({ artist }) {
  return (
    <>
      <h1>{artist.name}</h1>
      <Suspense fallback={<Loading />}>
        <Albums artistId={artist.id} />
      </Suspense>
    </>
  );
```

자세히 보기

중요합니다!

Suspense가 가능한 데이터만이 Suspense 컴포넌트를 활성화합니다. 아래와 같은 것들이 해당합니다.

- Relay와 Next.js 같이 Suspense가 가능한 프레임워크를 사용한 데이터 가져오기.
- `lazy` 를 활용한 지연 로딩 컴포넌트.
- `use` 를 사용해서 캐시된 Promise 값 읽기.

Suspense는 Effect 또는 이벤트 핸들러 내부에서 가져오는 데이터를 감지하지 **않습니다**.

위의 `Albums` 컴포넌트에서 데이터를 로딩하는 정확한 방법은 프레임워크마다 다릅니다. Suspense가 가능한 프레임워크를 사용하는 경우, 프레임워크의 데이터 불러오기 관련 문서에서 자세한 내용을 확인할 수 있습니다.

독단적인 프레임워크를 사용하지 않는 Suspense가 가능한 데이터 가져오기 기능은 아직 지원되지 않습니다. Suspense 지원 데이터 소스를 구현하기 위한 요구 사항은 불안정하고 문서화되지 않았습니다. 데이터 소스를 Suspense와 통합하기 위한 공식 API는 향후 React 버전에서 출시될 예정입니다.

콘텐츠를 한꺼번에 함께 보여주기

기본적으로 Suspense 내부의 전체 트리는 하나의 단위로 취급됩니다. 예를 들어, 이러한 구성 요소 중 *하나라도* 어떤 데이터에 의해 지연되더라도 *모든* 구성 요소가 함께 로딩 표시로 대체됩니다.

```
<Suspense fallback={<Loading />}>
  <Biography />
  <Panel>
    <Albums />
  </Panel>
</Suspense>
```

그런 다음 모두 보일 준비가 되면 한꺼번에 모두 함께 보입니다.

아래 예시에서는 `Biography` 와 `Albums` 모두 어떤 데이터를 가져옵니다. 하지만 두 구성 요소는 같은 단일 Suspense 아래에 그룹화되어 있기 때문에 항상 동시에 함께 그려지게 됩니다.

ArtistPage.js Panel.js Biography.js Albums.js 새로고침 Clear 포크

```
import { Suspense } from 'react';
import Albums from './Albums.js';
import Biography from './Biography.js';
import Panel from './Panel.js';

export default function ArtistPage({ artist }) {
  return (
    <>
      <h1>{artist.name}</h1>
      <Suspense fallback={<Loading />}>
        <Biography artistId={artist.id} />
        <Panel>
```

자세히 보기

데이터를 로딩하는 컴포넌트가 Suspense의 직접적인 자식일 필요는 없습니다. 예를 들어, `Biography` 와 `Albums` 를 새로운 `Details` 컴포넌트로 이동할 수 있습니다. 이렇게 해도 동작은

변경되지 않습니다. `Biography` 와 `Albums` 는 가장 가까운 상위 Suspense를 공유하므로 두 컴포넌트의 노출 여부는 함께 조정됩니다.

```
<Suspense fallback={<Loading />}>
  <Details artistId={artist.id} />
</Suspense>

function Details({ artistId }) {
  return (
    <>
      <Biography artistId={artistId} />
      <Panel>
        <Albums artistId={artistId} />
      </Panel>
    </>
  );
}
```

중첩된 콘텐츠가 로딩될 때 보여주기

컴포넌트가 일시 중단되면 가장 가까운 상위 Suspense 컴포넌트가 Fallback을 보여줍니다. 이를 통해 여러 Suspense 컴포넌트를 중첩하여 로딩 순서를 만들 수 있습니다. 각 Suspense의 Fallback은 다음 레벨의 콘텐츠를 사용할 수 있게 되면 채워집니다. 예를 들어 앨범 목록에 자체 Fallback을 지정할 수 있습니다.

```
<Suspense fallback={<BigSpinner />}>
  <Biography />
  <Suspense fallback={<AlbumsGlimmer />}>
    <Panel>
      <Albums />
    </Panel>
  </Suspense>
</Suspense>
```

이 변경으로 `Biography` 를 보여줄 때 `Albums` 가 로딩될 때까지 “기다릴” 필요가 없습니다.

순서는 다음과 같습니다.

1. `Biography` 가 아직 로딩되지 않은 경우, 전체 콘텐츠 영역 대신 `BigSpinner` 가 표시됩니다.
2. `Biography` 의 로딩이 완료되면 `BigSpinner` 가 콘텐츠로 대체됩니다.
3. `Albums` 가 아직 로딩되지 않은 경우, `Albums` 와 그 상위 `Panel` 대신 `AlbumsGlimmer` 가 표시됩니다.
4. 마지막으로 `Albums` 가 로딩을 완료하면 `AlbumsGlimmer` 를 대체합니다.

ArtistPage.js Panel.js Biography.js Albums.js 새로고침 Clear 포크

```
import { Suspense } from 'react';
import Albums from './Albums.js';
import Biography from './Biography.js';
import Panel from './Panel.js';

export default function ArtistPage({ artist }) {
  return (
    <>
      <h1>{artist.name}</h1>
      <Suspense fallback={<BigSpinner />}>
        <Biography artistId={artist.id} />
        <Suspense fallback={<AlbumsGlimmer />}>
```

자세히 보기

Suspense 경계를 사용하면 UI의 어떤 부분이 항상 동시에 그려져야 하는지, 어떤 부분이 로딩 순서에서 점진적으로 더 많은 콘텐츠를 보여줘야 하는지 조정할 수 있습니다. 앱의 나머지 동작에 영향을 주지 않고 트리의 어느 위치에서나 Suspense를 추가, 이동, 삭제할 수 있습니다.

모든 컴포넌트 주위에 Suspense를 두지 마세요. Suspense는 사용자가 경험하기를 원하는 로딩 순서보다 더 세분화되어서는 안 됩니다. 디자이너와 함께 작업하는 경우 로딩 상태를 어디에 배치해야 하는지 디자이너에게 물어보세요. 디자이너가 이미 디자인 와이어 프레임에 포함했을 가능성이 높습니다.

새 콘텐츠가 로딩되는 동안 이전 콘텐츠 보여주기

이 예시에서는 검색 결과를 가져오는 동안 `SearchResults` 컴포넌트가 지연됩니다. `"a"` 를 입력하고 결과를 기다린 다음 `"ab"` 로 바꿔보세요. `"a"` 에 대한 결과는 로딩 Fallback으로 바뀝니다.

App.js SearchResults.js 새로고침 Clear 포크

```
import { Suspense, useState } from 'react';
import SearchResults from './SearchResults.js';

export default function App() {
  const [query, setQuery] = useState('');
  return (
    <>
      <label>
        Search albums:
        <input value={query} onChange={e => setQuery(e.target.value)} />
      </label>
      <Suspense fallback={<h2>Loading...</h2>}>
```

자세히 보기

일반적인 대체 UI 패턴은 목록들에 대한 업데이트를 *지연Defer*하고 새 결과가 준비될 때까지 이전 결과를 계속 보여주는 것입니다. `useDeferredValue` Hook을 사용하면 쿼리의 지연된 버전을 아래로 전달할 수 있습니다.

```
export default function App() {
  const [query, setQuery] = useState('');
  const deferredQuery = useDeferredValue(query);
  return (
    <>
      <label>
        Search albums:
        <input value={query} onChange={e => setQuery(e.target.value)} />
      </label>
      <Suspense fallback={<h2>Loading...</h2>}>
        <SearchResults query={deferredQuery} />
      </Suspense>
    </>
  );
}
```

`query` 는 즉시 업데이트되므로 입력에 새 값이 표시됩니다. 그러나 `deferredQuery` 는 데이터가 로딩될 때까지 이전 값을 유지하므로 `SearchResults` 는 잠시 동안 이전 결과를 보여줍니다.

사용자에게 더 명확하게 알리기 위해 이전 결과 목록이 표시될 때 시각적 표시를 추가할 수 있습니다.

```
<div style={{
  opacity: query !== deferredQuery ? 0.5 : 1
}}>
  <SearchResults query={deferredQuery} />
```

```
</div>
```

아래 예시에서 `"a"` 를 입력하고 결과가 로딩될 때까지 기다린 다음 입력을 `"ab"` 로 편집해보세요. 이제 새 결과가 로드될 때까지 Suspense Fallback 대신 희미한 이전 결과 목록이 표시되는 것을 확인할 수 있습니다.

App.js

새로고침 Clear 포크

```
import { Suspense, useState, useDeferredValue } from 'react';
import SearchResults from './SearchResults.js';

export default function App() {
  const [query, setQuery] = useState('');
  const deferredQuery = useDeferredValue(query);
  const isStale = query !== deferredQuery;
  return (
    <>
      <label>
        Search albums:
        <input value={query} onChange={e => setQuery(e.target.value)} />
```

자세히 보기

중요합니다!

지연된 값Deferred Value과 Transition을 사용하면 Suspense Fallback을 표시하지 않을 수 있습니다. Transition은 전체 업데이트를 긴급하지 않은 것으로 처리하므로 일반적으로 프레임워크와 Router 라이브러리에서 Navigation을 위해 사용합니다. 반면에 지연된 값Deferred Value은 UI의 일부를 긴급하지 않은 것으로 처리하고 나머지 UI보다 "지연"시키려는 목적의 애플리케이션 코드에서 유용합니다.

이미 보인 콘텐츠가 숨겨지지 않도록 방지

컴포넌트가 지연되면 가장 가까운 상위 Suspense가 Fallback을 보여주도록 전환합니다. 이미 일부 콘텐츠가 보이는 경우 사용자 경험이 끊길 수 있습니다. 이 버튼을 눌러 보세요.

App.js 새로고침 Clear 포크

```
import { Suspense, useState } from 'react';
import IndexPage from './IndexPage.js';
import ArtistPage from './ArtistPage.js';
import Layout from './Layout.js';

export default function App() {
  return (
    <Suspense fallback={<BigSpinner />}>
      <Router />
    </Suspense>
  );
}
```

자세히 보기

버튼을 눌렀을 때 `Router` 컴포넌트가 `IndexPage` 대신 `ArtistPage` 를 렌더링했습니다. `ArtistPage` 내부의 컴포넌트가 지연됐기 때문에 가장 가까운 Suspense가 Fallback을 보여주기 시작했습니다. 가장 가까운 Suspense가 Root 근처에 있었기 때문에 전체 사이트 레이아웃이 `BigSpinner` 로 대체되었습니다.

이를 방지하려면 `startTransition` 을 사용하여 Navigation State 업데이트를 *Transition*으로 처리할 수 있습니다.

```
function Router() {
  const [page, setPage] = useState('/');

  function navigate(url) {
    startTransition(() => {
      setPage(url);
    });
  }
  // ...
```

이는 State 전환이 급하지 않으며, 이미 공개된 콘텐츠를 숨기는 대신 이전 페이지를 계속 표시하는 것이 좋다는 것을 React에게 알려줍니다. 이제 버튼을 클릭하면 `Biography` 가 로딩될 때까지 “대기”합니다.

App.js 새로고침 Clear 포크

```
import { Suspense, startTransition, useState } from 'react';
import IndexPage from './IndexPage.js';
import ArtistPage from './ArtistPage.js';
```

```
import Layout from './Layout.js';

export default function App() {
  return (
    <Suspense fallback={<BigSpinner />}>
      <Router />
    </Suspense>
  );
}
```

자세히 보기

Transition은 *모든* 콘텐츠가 로딩될 때까지 기다리지 않습니다. 이미 보여진 콘텐츠가 숨겨지지 않도록 충분히 오래 기다립니다. 예를 들어 웹사이트 `Layout` 은 이미 보이므로 로딩 스피너 뒤에 숨기는 것은 좋지 않을 것입니다. 그러나 `Albums` 주위에 중첩된 `Suspense` 는 새로운 것이므로 Transition이 기다리지 않습니다.

중요합니다!

Suspense를 지원하는 Router는 기본적으로 Navigation 업데이트를 Transition으로 래핑할 것으로 예상합니다.

Transition이 발생하고 있음을 보여주기

위의 예시에서는 버튼을 클릭해도 Navigation이 진행 중이라는 시각적 표시가 없습니다. 표시기를 추가하려면 `startTransition`을 불리언Boolean 값인 `isPending` 값을 제공하는 `useTransition`으로 바꾸면 됩니다. 아래 예시에서는 Transition이 진행되는 동안 웹사이트 헤더 스타일을 변경하는 데 사용됩니다.

App.js

새로고침 Clear 포크

```
import { Suspense, useState, useTransition } from 'react';
import IndexPage from './IndexPage.js';
import ArtistPage from './ArtistPage.js';
import Layout from './Layout.js';

export default function App() {
  return (
    <Suspense fallback={<BigSpinner />}>
      <Router />
    </Suspense>
  );
}
```

자세히 보기

Navigation에서 Suspense 재설정하기

Transition이 진행되는 동안 React는 이미 보인 콘텐츠를 숨기지 않습니다. 하지만 다른 매개변수가 있는 경로로 이동하는 경우 React에 *다른* 콘텐츠라고 알려주고 싶을 수 있습니다. 이를 `key` 로 표현할 수 있습니다.

```
<ProfilePage key={queryParams.id} />
```

사용자의 프로필 페이지 내에서 이동 중인데 무언가가 지연되었다고 가정해 보세요. 해당 업데이트가 Transition으로 감싸져 있으면 이미 표시된 콘텐츠에 대한 Fallback이 트리거되지 않습니다. 이것이 예상되는 동작입니다.

하지만 이제 두 개의 서로 다른 사용자 프로필 사이를 이동한다고 가정해 보겠습니다. 이 경우 Fallback을 표시하는 것이 좋습니다. 예를 들어 한 사용자의 타임라인이 다른 사용자의 타임라인과 *다른 콘텐츠*라고, 가정해 보겠습니다. `key` 를 지정하면 React가 서로 다른 사용자의 프로필을 서로 다른 컴포넌트로 취급하고 탐색하는 동안 Suspense를 재설정하도록 할 수 있습니다. Suspense 통합 라우터는 이 동작을 자동으로 수행해야 합니다.

서버 에러 및 클라이언트 전용 콘텐츠에 대한 Fallback 제공

스트리밍 서버 렌더링 API 중 하나(또는 이에 의존하는 프레임워크)를 사용하는 경우, React는 서버의 에러를 처리하기 위해 `<Suspense>` 경계도 사용할 것입니다. 컴포넌트가 서버에서 에러를 발생시키더라도 React는 서버 렌더링을 중단하지 않습니다. 대신, 그 위에 있는 가장 가까운 `<Suspense>` 컴포넌트를 찾아서 그 Fallback(예: 스피너)을 생성된 서버 HTML에 포함합니다. 사용자는 처음에는 스피너를 보게 됩니다.

클라이언트에서 React는 동일한 컴포넌트를 다시 렌더링하려고 시도합니다. 클라이언트에서도 에러가 발생하면 React는 에러를 던지고 가장 가까운 Error Boundary를 표시합니다. 그러나 클라이언트에서 에러가 발생하지 않으면 콘텐츠가 결국 성공적으로 보였기 때문에 React는 사용자에게 에러를 보여주지 않습니다.

이를 사용하여 일부 컴포넌트를 서버에서 렌더링하지 않도록 선택할 수 있습니다. 이렇게 하려면 서버 환경에서 에러를 발생시킨 다음 `<Suspense>` 경계로 감싸서 해당 HTML을 Fallback으로 대

체합니다.

```
<Suspense fallback={<Loading />}>
  <Chat />
</Suspense>

function Chat() {
  if (typeof window === 'undefined') {
    throw Error('Chat should only render on the client.');
  }
  // ...
}
```

서버 HTML에 로딩 UI가 포함됩니다. 클라이언트에서는 `Chat` 컴포넌트로 대체됩니다.

문제 해결

업데이트 중에 UI가 Fallback으로 대체되는 것을 방지하려면 어떻게 해야 하나요?

표시되는 UI를 Fallback으로 대체하면 사용자 환경이 불안정해집니다. 이는 업데이트로 인해 컴포넌트가 지연되고 가장 가까운 Suspense가 이미 사용자에게 콘텐츠를 보여주고 있을 때 발생할 수 있습니다.

이런 일이 발생하지 않도록 하려면, `startTransition`을 사용하여 업데이트를 긴급하지 않은 것으로 처리하세요. Transition이 진행되는 동안 React는 원치 않는 Fallback이 나타나지 않도록 충분한 데이터가 로딩될 때까지 기다립니다.

```
function handleNextPageClick() {
  // If this update suspends, don't hide the already displayed content
  startTransition(() => {
    setCurrentPage(currentPage + 1);
  });
}
```

이렇게 하면 기존 콘텐츠가 숨겨지지 않습니다. 그러나 새로 렌더링된 `Suspense` 는 여전히 즉시 Fallback을 보여줘서 UI를 차단하지 않고 사용자가 콘텐츠를 이용할 수 있게 합니다.

React는 긴급하지 않은 업데이트 중에만 원치 않는 Fallback을 방지합니다. 긴급한 업데이트의 결과인 경우 렌더링을 지연시키지 않습니다. `startTransition` 또는 `useDeferredValue` 와 같은 API를 사용해야 합니다.

Router가 Suspense와 통합된 경우, Router는 업데이트를 자동으로 `startTransition` 에 래핑해야 합니다.

이전
<StrictMode>

다음
<Activity>