



startTransition

`startTransition` 을 사용하면 UI의 일부를 백그라운드에서 렌더링할 수 있습니다.

```
startTransition(action)
```

- 레퍼런스

- `startTransition(action)`

- 사용법

- State 업데이트를 Non-Blocking Transition으로 표시

레퍼런스

`startTransition(action)`

`startTransition` 함수는 State 업데이트를 Transition으로 표시할 수 있게 해줍니다.

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

function TabContainer() {
  const [tab, setTab] = useState('about');

  function selectTab(nextTab) {
    startTransition(() => {
      setTab(nextTab);
    });
  }
  // ...
}
```

아래 예시를 참고하세요.

매개변수

- `action`: 하나 이상의 `set` 함수를 호출하여 일부 State를 업데이트하는 함수입니다. React는 매개변수 없이 `action` 을 즉시 호출하고 `action` 함수를 호출하는 동안 동기적으로 예약된 모든 State 업데이트를 Transition으로 표시합니다. `action` 에서 `await` 된 비동기 호출은 Transition에 포함되지만, 현재로서는 `await` 이후의 `set` 함수 호출을 추가적인 `startTransition` 으로 감싸야 합니다([문제 해결 참조](#)). Transitions으로 표시된 상태 업데이트는 **non-blocking** 방식으로 처리되며, **불필요한 로딩 표시가 나타나지 않습니다**.

반환값

`startTransition` 은 아무것도 반환하지 않습니다.

주의 사항

- `startTransition` 은 Transition이 대기(Pending) 중인지 추적할 수 있는 방법을 제공하지 않습니다. 대기 중인 Transition을 표시하려면 `useTransition` 이 필요합니다.
- 해당 State의 `set` 함수에 접근할 수 있는 경우에만 업데이트를 Transition으로 래핑할 수 있습니다. 일부 Props나 Custom Hook 반환 값에 대한 응답으로 Transition을 시작하려면 `useDeferredValue` 를 대신 사용하세요.
- `startTransition` 에 전달하는 함수는 즉시 호출되며, 실행 중 발생하는 모든 상태 업데이트를 Transition으로 표시합니다. 예를 들어 `setTimeout` 내에서 상태를 업데이트하려고 하면, 해당 업데이트는 Transition으로 표시되지 않습니다.
- 비동기 요청 이후의 State 업데이트를 Transition으로 표시하려면, 반드시 또 다른 `startTransition` 으로 감싸야 합니다. 이는 알려진 제한 사항으로 향후 수정될 예정입니다. ([문제 해결 참조](#))
- Transition으로 표시된 State 업데이트는 다른 State 업데이트에 의해 중단됩니다. 예를 들어, Transition 내에서 차트 컴포넌트를 업데이트하지만 차트가 다시 렌더링되는 동안 입력을 시작하면 React는 입력 State 업데이트를 처리한 후 차트 컴포넌트에서 렌더링 작업을 다시 시작합니다.
- Transition 업데이트는 텍스트 입력을 제어하는 데 사용할 수 없습니다.

- 만약 진행 중인 `Transition`이 여러 개 있는 경우, `React`에서는 함께 일괄 처리 합니다. 이는 향후 릴리즈에서 제거될 가능성이 높은 제한 사항입니다.

사용법

State 업데이트를 Non-Blocking Transition으로 표시

`startTransition` 으로 래핑함으로써 State 업데이트를 *Transition*으로 표시할 수 있습니다.

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

function TabContainer() {
  const [tab, setTab] = useState('about');

  function selectTab(nextTab) {
    startTransition(() => {
      setTab(nextTab);
    });
  }
  // ...
}
```

`Transition`을 사용하면 느린 장치에서도 사용자 인터페이스 업데이트의 반응성을 유지할 수 있습니다.

`Transition`을 사용하면 UI가 리렌더링 도중에도 반응성을 유지합니다. 예를 들어 사용자가 탭을 클릭 했다가 마음이 바뀌어 다른 탭을 클릭하면 첫 번째 리렌더링이 완료될 때 까지 기다릴 필요 없이 다른 탭을 클릭할 수 있습니다.

📌 중요합니다!

`startTransition` 은 `useTransition` 과 매우 유사하지만, `Transition`이 대기 중인지 추적하는 `isPending` 플래그를 제공하지 않습니다. `useTransition` 을 사용할 수 없을

때 `startTransition` 을 호출할 수 있습니다. 예를 들어, `startTransition` 은 데이터 라이브러리에서와 같이 컴포넌트 외부에서 작동합니다.

[Transition에 대한 학습 및 예시는 `useTransition` 페이지에서 확인하세요.](#)

< [이전](#)
[memo](#)

[다음](#) >
[use](#)

Meta Open Source

Copyright © Meta Platforms, Inc

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](#)

