useLayoutEffect

> **주의하세요!**
>
> `useLayoutEffect` 를 사용하면 성능이 저하될 수 있습니다. 가능하다면 `useEffect` 를 사용하세요.

`useLayoutEffect` 는 브라우저가 화면을 다시 그리기 전에 실행되는 `useEffect` 입니다.

```
useLayoutEffect(setup, dependencies?)
```


- 레퍼런스
  - `useLayoutEffect(setup, dependencies?)`
- 사용법
  - 브라우저가 화면을 다시 그리기 전에 레이아웃 계산하기
- 문제 해결
  - 오류가 발생했습니다: “ `useLayoutEffect` does nothing on the server”

레퍼런스

`useLayoutEffect(setup, dependencies?)`

`useLayoutEffect` 를 호출하여 브라우저가 화면을 다시 그리기 전에 레이아웃을 계산합니다.

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

function Tooltip() {
  const ref = useRef(null);
  const [tooltipHeight, setTooltipHeight] = useState(0);

  useLayoutEffect(() => {
    const { height } = ref.current.getBoundingClientRect();
    setTooltipHeight(height);
  }, []);
  // ...
```

아래에서 더 많은 예시를 확인하세요.

매개변수

- `setup` : The function with your Effect's logic. Your setup function may also optionally return a *cleanup* function. Before your component is added to the DOM, React will run your setup function. After every re-render with changed dependencies, React will first run the cleanup function (if you provided it) with the old values, and then run your setup function with the new values. Before your component is removed from the DOM, React will run your cleanup function.

- **optional** `dependencies` : The list of all reactive values referenced inside of the `setup` code. Reactive values include props, state, and all the variables and functions declared directly inside your component body. If your linter is configured for React, it will verify that every reactive value is correctly specified as a dependency. The list of dependencies must have a constant number of items and be written inline like `[dep1, dep2, dep3]` . React will compare each dependency with its previous value using the `Object.is` comparison. If you omit this argument, your Effect will re-run after every re-render of the component.

- **선택사항** `dependencies` : `setup` 코드 내에서 참조된 모든 반응형 값의 목록입니다. 반응형 값에는 props, state, 그리고 컴포넌트 본문에 직접 선언된 모든 변수와 함수가 포함됩니다. linter가 React용으로 설정된 경우, 모든 반응형 값이 의존성으로 올바르게 지정되었는지 확인합니다. 의존성 목록에는 일정한 수의 항목이 있어야 하며 `[dep1, dep2, dep3]` 와 같이 작성해야 합니다. React는 `Object.is` 비교 알고리즘을 사용하여 각 의존성을 이전 값과 비교합니다. 의존성을 전혀 지정하지 않으면 컴포넌트를 다시 렌더링할 때마다 Effect가 다시 실행됩니다.

반환값

`useLayoutEffect`는 `undefined`를 반환합니다.

주의 사항

- `useLayoutEffect`는 Hook이므로, **컴포넌트의 최상위 레벨** 또는 커스텀 Hook에서만 호출할 수 있습니다. 반복문이나 조건문 내에서 호출할 수 없습니다. 이 작업이 필요하다면 새로운 컴포넌트로 분리해서 Effect를 새 컴포넌트로 옮기세요.
- Strict Mode가 켜져 있으면, React는 실제 첫 번째 setup 함수가 실행되기 이전에 **개발 모드에만 한정하여 한 번의 추가적인 setup + cleanup 사이클을 실행합니다.** 이는 cleanup 로직이 setup 로직을 완벽히 "반영"하고, setup 로직이 수행하는 작업을 중단하거나 되돌리는 지를 확인하는 스트레스 테스트입니다. 이로 인해 문제가 발생하면 cleanup 함수를 구현하세요.
- 의존성 중에 컴포넌트 내부에서 정의된 객체나 함수가 있는 경우, **Effect 가 필요 이상으로 다시 실행될 위험이 있습니다.** 이를 해결하려면 불필요한 객체 의존성이나 함수 의존성을 제거하세요. State 업데이트나 비 반응형 로직을 effect 밖으로 빼낼 수 도 있습니다.
- Effect는 **클라이언트 환경에서만 동작합니다.** 서버 렌더링 중에는 실행되지 않습니다.
- `useLayoutEffect` 내부의 코드와 이로 인한 모든 state 업데이트는 **브라우저가 화면을 다시 그리는 것을 막습니다.** 과도하게 사용하면 앱이 느려집니다. 가능하면 `useEffect`를 사용하세요.
- `useLayoutEffect` 내부에서 state 업데이트를 실행하면 React는 `useEffect`를 포함한 나머지 모든 Effect를 즉시 실행합니다.

사용법

브라우저가 화면을 다시 그리기 전에 레이아웃 계산하기

대부분의 컴포넌트는 렌더링을 위해 해당 컴포넌트의 화면상 위치와 크기를 알 필요가 없습니다. 컴포넌트가 JSX를 반환하면 브라우저가 컴포넌트의 *레이아웃*(위치와 크기)를 계산하고 화면을 다시 그립니다.

가끔은 이것만으로는 부족한 경우가 있습니다. 마우스 커서를 올리면 툴팁이 요소 옆에 나타나는 경우를 생각해 보세요. 충분한 공간이 있다면 툴팁은 요소 위에 나타나겠지만, 공간이 부족하다면

아래에 나타나야 합니다. 결국 툴팁을 올바른 위치에 렌더링하려면 툴팁의 높이를 알아야 합니다. (위쪽 공간에 들어가는지 판단 해야 함)

이를 위해 두 번의 렌더링을 거쳐야 합니다.

1. 툴팁을 (잘못된 위치라도) 아무 위치에 렌더링합니다
2. 툴팁의 높이를 계산해서 툴팁을 배치할 위치를 결정합니다.
3. 올바른 위치에 툴팁을 *다시* 렌더링합니다.

이 작업은 브라우저가 화면을 다시 그리기 전에 모두 이루어져야 합니다. 툴팁이 움직이는 걸 사용자에게 보이고 싶지 않으니까요. `useLayoutEffect` 를 호출해서 브라우저가 화면을 다시 그리기 전에 레이아웃을 계산하세요.

```
function Tooltip() {
  const ref = useRef(null);
  const [tooltipHeight, setTooltipHeight] = useState(0); // 아직 실제 높이를 모릅니다.

  useLayoutEffect(() => {
    const { height } = ref.current.getBoundingClientRect();
    setTooltipHeight(height); // 실제 높이를 알았으니 다시 렌더링합니다.
  }, []);

  // ...아래에 올 렌더링 로직에서 tooltipHeight를 사용하세요...
}
```

작동 방식을 단계별로 알아봅시다.

1. `Tooltip` 은 초기화된 값인 `tooltipHeight = 0` 으로 렌더링 됩니다 (따라서 툴팁의 위치는 잘못될 수 있습니다).
2. React가 이 툴팁을 DOM에 배치하고 `useLayoutEffect` 안의 코드를 실행합니다.
3. `useLayoutEffect` 가 툴팁의 높이를 계산하고 바로 다시 렌더링시킵니다.
4. `Tooltip` 이 실제 `tooltipHeight` 로 렌더링 됩니다. (따라서 툴팁이 올바른 위치에 배치됩니다.)
5. React가 DOM에서 이를 업데이트하고 마침내 브라우저가 툴팁을 표시합니다.

아래의 버튼들 위로 마우스 커서를 올려서 툴팁이 공간에 들어가는지에 따라 위치를 조정하는 것을 확인하세요.

App.js ButtonWithTooltip.js Tooltip.js TooltipCont 새로고침 Clear 포크

```
import { useRef, useLayoutEffect, useState } from 'react';
import { createPortal } from 'react-dom';
import TooltipContainer from './TooltipContainer.js';

export default function Tooltip({ children, targetRect }) {
  const ref = useRef(null);
  const [tooltipHeight, setTooltipHeight] = useState(0);

  useLayoutEffect(() => {
    const { height } = ref.current.getBoundingClientRect();
    setTooltipHeight(height);
    console.log('Measured tooltip height: ' + height);
```

자세히 보기

`Tooltip` 컴포넌트는 두 번의 렌더링을 거치지만 (처음은 `0` 으로 초기화된 `tooltipHeight` 로 렌더링 되고, 그다음 실제로 계산된 높이로 렌더링 됨), 실제로 보이는 건 최종 결과뿐입니다. 이 예시에서 `useEffect` 대신 `useLayoutEffect` 가 필요한 이유입니다. 아래에서 차이점을 자세하게 살펴봅시다.

useLayoutEffect vs useEffect

예시 1 of 2: useLayoutEffect 는 브라우저가 화면을 다시 그리는 것을 막습니다

React는 useLayoutEffect 내부의 코드와 이로 인한 모든 state 업데이트가 **브라우저가 화면을 다시 그리기 전에** 처리되는 것을 보장합니다. 덕분에 툴팁을 렌더링하고, 위치와 크기를 계산하고 다시 렌더링하면서 첫 번째 렌더링은 사용자가 모르게 할 수 있습니다. 즉, useLayoutEffect 는 브라우저가 화면을 그리는 것을 막습니다.

Tooltip.js

새로고침 Clear 포크

```
import { useRef, useLayoutEffect, useState } from 'react';
import { createPortal } from 'react-dom';
import TooltipContainer from './TooltipContainer.js';

export default function Tooltip({ children, targetRect }) {
  const ref = useRef(null);
  const [tooltipHeight, setTooltipHeight] = useState(0);

  useLayoutEffect(() => {
    const { height } = ref.current.getBoundingClientRect();
    setTooltipHeight(height);
  }, []);
```

자세히 보기

다음 예시

중요합니다!

두 번에 걸쳐서 렌더링하고 브라우저를 막는 것은 성능을 저하합니다. 가능하면 피하세요.

문제 해결

오류가 발생했습니다: "`useLayoutEffect` does nothing on the server"

`useLayoutEffect`의 목적은 레이아웃 정보를 사용해서 컴포넌트를 렌더링하는 것입니다.

1. 초기 콘텐츠를 렌더링합니다.
2. *브라우저가 화면을 다시 그리기 전에* 레이아웃을 계산합니다.
3. 읽은 레이아웃 정보를 사용해서 최종 콘텐츠를 렌더링합니다.

서버 렌더링을 직접 사용하거나 프레임워크에서 사용하는 경우라면, React 앱은 서버에서 초기 렌더링을 해서 HTML을 만듭니다. 이를 통해 JavaScript 코드가 로드되기 전에 초기 HTML을 보여주게 됩니다.

문제는 서버에는 레이아웃 정보가 없다는 것입니다.

앞선 예시에선 `Tooltip` 컴포넌트에서 `useLayoutEffect`를 호출하여 툴팁을 콘텐츠의 높이에 따라 (콘텐츠의 위쪽과 아래쪽 중) 올바른 위치에 배치합니다. 초기 서버 HTML의 일부로 `Tooltip`을 렌더링하려 하면, 이때는 툴팁의 위치를 올바르게 결정할 수 없을 것입니다. 서버에서

는 아직 레이아웃 정보가 없으니까요! 따라서 툴팁을 서버에서 렌더링하기를 원하더라도, 클라이언트로 옮겨서 자바스크립트가 로드되고 실행된 후에 렌더링해야 합니다.

일반적으로 레이아웃 정보에 의존하는 컴포넌트는 어차피 서버에서 렌더링할 필요가 없습니다. 예를 들면, 초기 렌더링 중에 `Tooltip` 이 보이는 것은 말이 안 됩니다. `Tooltip` 은 클라이언트 상호작용에 의해서 보이는 것이니까요.

그럼에도 이런 문제를 마주친다면 몇 가지 다른 옵션이 있습니다.

- `useLayoutEffect` 를 `useEffect` 로 대체 하세요. 화면을 그리는 것을 막지 말고 (초기 HTML이 Effect 실행 전에 보이기 때문에) 초기 렌더링이 보이더라도 괜찮다고 React에게 말해주는 것입니다.
- 또는 해당 컴포넌트를 클라이언트 전용으로 만드세요. React가 가장 가까운 `<Suspense>` 경계 안의 콘텐츠를 서버렌더링 동안 (스피너나 글리머같은) loading fallbck으로 대체 하게 합니다.
- 또는 `useLayoutEffect` 가 있는 컴포넌트를 hydration 이후에만 렌더링할 수도 있습니다. 불리언 타입인 `isMounted` state를 초깃값인 `false` 로 유지하다가, `useEffect` 호출되면 거기서 `true` 로 값을 변경하세요. 그러면 렌더링 로직은 `return isMounted ? <RealContent /> : <FallbackContent />` 처럼 될 수 있습니다. 서버에서 렌더링하는 중이거나 hydration 동안 사용자는 `FallbackContent` 를 볼 것이고 `FallbackContent` 는 `useLayoutEffect` 를 호출하지 않아야 합니다. 그 후에 React가 `FallbackContent` 를 클라이언트 전용이면서 `useLayoutEffect` 를 호출하는 `RealContent` 로 변경할 겁니다.
- 컴포넌트를 외부 데이터 저장소와 동기화하고, 레이아웃 계산 외에 다른 이유로 `useLayoutEffect` 에 의존하는 경우라면, 대신 `useSyncExternalStore` 를 고려해 보세요. 이 Hook은 서버 렌더링을 지원합니다.

이전
useInsertionEffect

다음
useMemo

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