useRef

useRef 는 렌더링에 필요하지 않은 값을 참조할 수 있는 React Hook입니다.

```
const ref = useRef(initialValue)
```

레퍼런스

useRef(initialValue)

컴포넌트의 최상위 레벨에서 useRef 를 호출하여 ref를 선언합니다.

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

function MyComponent() {
  const intervalRef = useRef(0);
  const inputRef = useRef(null);
  // ...
```

아래에서 더 많은 예시를 확인하세요.

매개변수

- `initialValue` : ref 객체의 `current` 프로퍼티 초기 설정값입니다. 여기에는 어떤 유형의 값이든 지정할 수 있습니다. 이 인자는 초기 렌더링 이후부터는 무시됩니다.

반환값

`useRef` 는 단일 프로퍼티를 가진 객체를 반환합니다:

- `current` : 처음에는 전달한 `initialValue` 로 설정됩니다. 나중에 다른 값으로 바꿀 수 있습니다. ref 객체를 JSX 노드의 `ref` 어트리뷰트로 React에 전달하면 React는 `current` 프로퍼티를 설정합니다.

다음 렌더링에서 `useRef` 는 동일한 객체를 반환합니다.

주의 사항

- `ref.current` 프로퍼티는 state와 달리 변이할 수 있습니다. 그러나 렌더링에 사용되는 객체(예: state의 일부)를 포함하는 경우 해당 객체를 변이해서는 안 됩니다.
- `ref.current` 프로퍼티를 변경해도 React는 컴포넌트를 다시 렌더링하지 않습니다. ref는 일반 JavaScript 객체이기 때문에 React는 사용자가 언제 변경했는지 알지 못합니다.
- 초기화를 제외하고는 렌더링 중에 `ref.current` 를 쓰거나 *읽지* 마세요. 이렇게 하면 컴포넌트의 동작을 예측할 수 없게 됩니다.
- Strict Mode에서 React는 **컴포넌트 함수를 두 번 호출하여** 의도하지 않은 변경을 찾을 수 있도록 돕습니다. 이는 개발 환경 전용 동작이며 Production 환경에는 영향을 미치지 않습니다. 각 ref 객체는 두 번 생성되고 그중 하나는 버려집니다. 컴포넌트 함수가 순수하다면(그래야만 합니다), 컴포넌트의 로직에 영향을 미치지 않습니다.

사용법

ref로 값 참조하기

컴포넌트의 최상위 레벨에서 `useRef` 를 호출하여 하나 이상의 ref를 선언합니다.

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

```
function Stopwatch() {
  const intervalRef = useRef(0);
  // ...
```

`useRef`는 처음에 제공한 `초기값`으로 설정된 단일 `current 프로퍼티`가 있는 `ref 객체`를 반환합니다.

다음 렌더링에서 `useRef`는 동일한 객체를 반환합니다. 정보를 저장하고 나중에 읽을 수 있도록 `current` 속성을 변경할 수 있습니다. state가 떠오를 수 있지만, 둘 사이에는 중요한 차이점이 있습니다.

ref를 변경해도 리렌더링을 촉발하지 않습니다. 즉 ref는 컴포넌트의 시각적 출력에 영향을 미치지 않는 정보를 저장하는 데 적합합니다. 예를 들어 interval ID를 저장했다가 나중에 불러와야 하는 경우 ref에 넣을 수 있습니다. ref 내부의 값을 업데이트하려면 `current 프로퍼티`를 수동으로 변경해야 합니다:

```
function handleStartClick() {
  const intervalId = setInterval(() => {
    // ...
  }, 1000);
  intervalRef.current = intervalId;
}
```

나중에 ref에서 해당 interval ID를 읽어 해당 interval을 취소할 수 있습니다:

```
function handleStopClick() {
  const intervalId = intervalRef.current;
  clearInterval(intervalId);
}
```

ref를 사용하면 다음을 보장합니다:

- (렌더링할 때마다 재설정되는 일반 변수와 달리) 리렌더링 사이에 **정보를 저장**할 수 있습니다.
- (리렌더링을 촉발하는 state 변수와 달리) 변경해도 **리렌더링을 촉발하지 않습니다.**
- (정보가 공유되는 외부 변수와 달리) 각각의 컴포넌트에 **로컬로 저장됩니다.**

ref를 변경해도 다시 렌더링되지 않으므로 화면에 표시되는 정보를 저장하는 데는 ref가 적합하지 않습니다. 대신 state를 사용하세요. 더 자세한 내용은 `useRef`와 `useState` 중 선택하기에서 확인하세요.

useRef로 값을 참조하는 예시

1. counter 클릭하기
2. 스톱워치

예시 1 of 2: counter 클릭하기

이 컴포넌트는 ref를 사용하여 버튼이 클릭된 횟수를 추적합니다. 클릭 횟수는 이벤트 핸들러에서만 읽고 쓰기 때문에 여기서는 state 대신 ref를 사용해도 괜찮습니다.

App.js

다운로드 새로고침 Clear 포크

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

export default function Counter() {
  let ref = useRef(0);

  function handleClick() {
    ref.current = ref.current + 1;
    alert('You clicked ' + ref.current + ' times!');
  }

  return (
    <button onClick={handleClick}>
```

자세히 보기

JSX에 `{ref.current}`를 표시하면 클릭 시 번호가 업데이트되지 않습니다. `ref.current`를 설정해도 리렌더링을 촉발하지 않기 때문입니다. 렌더링에 사용하는 정보는 ref가 아닌 state여야 합니다.

다음 예시

주의하세요!

렌더링 중에는 `ref.current`를 쓰거나 *읽지* 마세요.

React는 컴포넌트의 본문이 순수 함수처럼 동작하기를 기대합니다:

- 입력값들(props, state, context)이 동일하면 완전히 동일한 JSX를 반환해야 합니다.
- 다른 순서나 다른 인수를 사용하여 호출해도 다른 호출의 결과에 영향을 미치지 않아야 합니다.

렌더링 중에 ref를 읽거나 쓰면 이러한 기대가 깨집니다.

```
function MyComponent() {
  // ...
  // 🚩 Don't write a ref during rendering
  myRef.current = 123;
  // ...
  // 🚩 Don't read a ref during rendering
  return <h1>{myOtherRef.current}</h1>;
}
```

대신 이벤트 핸들러나 Effect에서 ref를 읽거나 쓸 수 있습니다.

```
function MyComponent() {
  // ...
  useEffect(() => {
    // ✅ You can read or write refs in effects
    myRef.current = 123;
  });
  // ...
  function handleClick() {
    // ✅ You can read or write refs in event handlers
    doSomething(myOtherRef.current);
  }
  // ...
}
```

렌더링 중에 무언가를 읽거나 써야*만* 하는 경우, 대신 state를 사용하세요.

컴포넌트는 이러한 규칙을 어기더라도 여전히 작동할 수도 있지만, React에 추가되는 대부분의 새로운 기능들은 이러한 기대에 의존합니다. 자세한 내용은 컴포넌트를 순수하게 유지하기에서 확인하세요.

---

ref로 DOM 조작하기

ref를 사용하여 DOM을 조작하는 것은 특히 일반적입니다. React에는 이를 위한 기본 지원이 있습니다.

먼저 초기값 이 `null` 인 ref 객체 를 선언하세요:

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

function MyComponent() {
  const inputRef = useRef(null);
  // ...
```

그런 다음 ref 객체를 `ref` 속성으로 조작하려는 DOM 노드의 JSX에 전달하세요:

```
// ...
return <input ref={inputRef} />;
```

React가 DOM 노드를 생성하고 화면에 그린 후, React는 ref 객체의 `current` 프로퍼티를 DOM 노드로 설정합니다. 이제 DOM 노드 `<input>` 접근해 `focus()` 와 같은 메서드를 호출할 수 있습니다.

```
function handleClick() {
  inputRef.current.focus();
}
```

노드가 화면에서 제거되면 React는 `current` 프로퍼티를 다시 `null` 로 설정합니다.

자세한 내용은 ref로 DOM 조작하기에서 알아보세요.

useRef로 DOM을 조작하는 예시

1. 텍스트 input에 초점 맞추기 2. 이미지 스크롤하기 3. 비디오 재생 및 정지하기 4.

예시 1 of 4: 텍스트 input에 초점 맞추기

이 예시에서는 버튼을 클릭하면 입력에 초점이 맞춰집니다.

App.js 다운로드 새로고침 Clear 포크

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

export default function Form() {
  const inputRef = useRef(null);

  function handleClick() {
    inputRef.current.focus();
  }
```

```
    return (
      <>
        <input ref={inputRef} />
```

자세히 보기

다음 예시

ref 콘텐츠 재생성 피하기

React는 초기에 ref 값을 한 번 저장하고, 다음 렌더링부터는 이를 무시합니다.

```
function Video() {
  const playerRef = useRef(new VideoPlayer());
  // ...
```

`new VideoPlayer()` 의 결과는 초기 렌더링에만 사용되지만, 호출 자체는 이후의 모든 렌더링에서도 여전히 계속 이뤄집니다. 이는 값비싼 객체를 생성하는 경우 낭비일 수 있습니다.

이 문제를 해결하려면 대신 다음과 같이 ref를 초기화할 수 있습니다:

```
function Video() {
  const playerRef = useRef(null);
  if (playerRef.current === null) {
    playerRef.current = new VideoPlayer();
  }
  // ...
```

일반적으로 렌더링 중에 `ref.current`를 쓰거나 읽는 것은 허용되지 않습니다. 하지만 이 경우에는 결과가 항상 동일하고 초기화 중에만 조건이 실행되므로 충분히 예측할 수 있으므로 괜찮습니다.

자세히 살펴보기

useRef를 초기화할 때 null 검사를 피하는 방법

자세히 보기

문제 해결

커스텀 컴포넌트에 대한 ref를 얻을 수 없습니다

컴포넌트에 `ref`를 전달하고자 다음과 같이 하면:

```
const inputRef = useRef(null);

return <MyInput ref={inputRef} />;
```

다음과 같은 오류가 발생할 것입니다:

Console

TypeError: Cannot read properties of null

기본적으로 컴포넌트는 내부의 DOM 노드에 대한 ref를 외부로 노출하지 않습니다.

이 문제를 해결하려면 ref를 가져오고자 하는 컴포넌트를 찾으세요:

```
export default function MyInput({ value, onChange }) {
  return (
    <input
      value={value}
      onChange={onChange}
    />
  );
}
```

그리고 `ref` 를 컴포넌트가 받는 Props 목록에 추가한 뒤, 아래처럼 해당 자식 내장 컴포넌트에 Prop으로 `ref` 를 전달하세요.

```
function MyInput({ value, onChange, ref }) {
  return (
    <input
      value={value}
      onChange={onChange}
      ref={ref}
    />
  );
};

export default MyInput;
```

그러면 부모 컴포넌트가 ref를 가져올 수 있습니다.

자세한 내용은 다른 컴포넌트의 DOM 노드에 접근하기에서 확인하세요.

이전
useReducer

다음
useState

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