



forwardRef

❗ 더 이상 사용되지 않습니다!

React 19부터는 더 이상 `forwardRef` 이 필요하지 않습니다. 이제 `ref` 를 Prop으로 직접 전달하면 됩니다.

`forwardRef` 는 향후 릴리스에서 사용 중단^{Deprecated}될 예정입니다. 자세한 내용은 [여기](#)에서 확인하세요.

`forwardRef` lets your component expose a DOM node to the parent component with a [ref](#).

```
const SomeComponent = forwardRef(render)
```

- 레퍼런스

- `forwardRef(render)`
- `render` 함수

- 사용법

- 부모 컴포넌트에 DOM 노드 노출하기
- 여러 컴포넌트를 통해 Ref 전달하기
- DOM 노드 대신 명령형 핸들 노출하기

- 문제해결

- 컴포넌트가 `forwardRef` 로 감싸져 있지만, 컴포넌트의 `ref` 는 항상 `null` 입니다.

레퍼런스

forwardRef(render)

컴포넌트가 Ref를 받아 하위 컴포넌트로 전달하도록 하려면 `forwardRef()` 를 호출하세요.

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

const MyInput = forwardRef(function MyInput(props, ref) {
  // ...
});
```

아래에서 더 많은 예시를 확인하세요.

매개변수

- `render`: 컴포넌트의 렌더링 함수입니다. React는 컴포넌트가 부모로부터 받은 `props` 와 `ref` 로 이 함수를 호출합니다. 반환하는 JSX는 컴포넌트의 결과가 됩니다.

반환값

`forwardRef` 는 JSX에서 렌더링할 수 있는 React 컴포넌트를 반환합니다. 일반 함수로 정의된 React 컴포넌트와 다르게, `forwardRef` 가 반환하는 컴포넌트는 `ref Prop`도 받을 수 있습니다.

주의 사항

- Strict Mode에서 React는 실수로 발생한 결함을 찾기 위해 렌더링 함수를 두 번 호출합니다. 이는 개발 환경 전용 동작이며 프로덕션 환경에는 영향을 미치지 않습니다. 렌더링 함수가 순수 함수인 경우(그래야만 합니다), 컴포넌트 로직에 영향을 미치지 않습니다. 호출 결과 중 하나의 결과는 무시됩니다.

render 함수

`forwardRef` 는 `render` 함수를 인수로 받습니다. React는 `props` 및 `ref` 와 함께 이 함수를 호출합니다.

```
const MyInput = forwardRef(function MyInput(props, ref) {
  return (
    <label>
      {props.label}
      <input ref={ref} />
    </label>
  );
});
```

매개변수

- `props`: 부모 컴포넌트가 전달한 Props입니다.
- `ref`: 부모 컴포넌트가 전달한 `ref` 어트리뷰트입니다. `ref`는 객체나 함수일 수 있습니다. 부모 컴포넌트가 Ref를 전달하지 않은 경우 `null`이 됩니다. 전달받은 `ref`를 다른 컴포넌트에 전달하거나 `useImperativeHandle`에 전달해야 합니다.

반환값

`forwardRef`는 JSX에서 렌더링할 수 있는 React 컴포넌트를 반환합니다. 일반 함수로 정의된 React 컴포넌트와 다르게 `forwardRef`에 의해 반환되는 컴포넌트는 `ref Prop`를 받을 수 있습니다.

사용법

부모 컴포넌트에 DOM 노드 노출하기

기본적으로 각 컴포넌트의 DOM 노드는 비공개입니다. 그러나 때로는 부모에 DOM 노드를 노출하는 것이 유용할 수 있습니다. 예를 들어 포커스^{Focus}하기 위해 노출할 수 있습니다. 이를 위해 컴포넌트 정의를 `forwardRef()`로 감싸주면 됩니다.

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

const MyInput = forwardRef(function MyInput(props, ref) {
  const { label, ...otherProps } = props;
  return (
```

```

    <label>
      {label}
    <input {...otherProps} />
  </label>
);
});

```

props 다음에 두 번째 인수로 ref 를 받게 됩니다. 노출하려는 DOM 노드에 이를 전달합니다.

```

import { forwardRef } from 'react';

const MyInput = forwardRef(function MyInput(props, ref) {
  const { label, ...otherProps } = props;
  return (
    <label>
      {label}
      <input {...otherProps} ref={ ref } />
    </label>
  );
});

```

이렇게 하면 부모인 Form 컴포넌트가 MyInput 에 의해 노출된 <input> DOM 노드 에 접근할 수 있습니다.

```

function Form() {
  const ref = useRef(null);

  function handleClick() {
    ref.current.focus();
  }

  return (
    <form>
      <MyInput label="Enter your name:" ref={ ref } />
      <button type="button" onClick={handleClick}>
        Edit
      </button>
    </form>
  );
}

```

```
}
```

이 Form 컴포넌트는 MyInput 에게 **Ref를 전달**합니다. MyInput 컴포넌트는 해당 Ref를 `<input>` 태그에 전달합니다. 결과적으로 Form 컴포넌트는 해당 `<input>` DOM 노드에 접근하여 `focus()` 를 호출할 수 있습니다.

컴포넌트 내부 DOM 노드의 Ref를 노출하면 나중에 컴포넌트의 내부를 변경하기가 더 어려워진다는 점에 유의하세요. 일반적으로 버튼이나 텍스트 Input과 같이 재사용할 수 있는 저수준 컴포넌트에서 DOM 노드를 노출하지만, 아바타나 댓글 같은 애플리케이션 레벨의 컴포넌트에서는 노출하고 싶지 않을 것입니다.

예시 살펴보기

1. 텍스트 Input에 초점 맞추기 2. 비디오 재생 및 정지하기



예시 1 of 2: 텍스트 Input에 초점 맞추기

버튼을 클릭하면 Input에 포커스됩니다. Form 컴포넌트는 Ref를 정의하고 이를 MyInput 컴포넌트로 전달합니다. MyInput 컴포넌트는 해당 Ref를 `input` 으로 전달합니다. 이렇게 하면 Form 컴포넌트가 `input` 에 포커스를 줄 수 있습니다.

App.js **MyInput.js**

🔄 새로고침 ✕ Clear 📄 포크

```
import { useRef } from 'react';
import MyInput from './MyInput.js';

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

  function handleClick() {
    ref.current.focus();
  }

  return (
    <form>
```

▼ 자세히 보기

다음 예시

여러 컴포넌트를 통해 Ref 전달하기

ref 를 DOM 노드로 전달하지 않고 MyInput 과 같은 컴포넌트로 전달할 수 있습니다.

```
const FormField = forwardRef(function FormField(props, ref) {  
  // ...  
  return (  
    <>  
      <MyInput ref={ref} />  
      ...  
    </>  
  );  
});
```

MyInput 컴포넌트가 <input> 에 ref 를 전달하면 FormField 의 ref 는 해당 <input> 을 얻을 수 있습니다.

```
function Form() {
  const ref = useRef(null);

  function handleClick() {
    ref.current.focus();
  }

  return (
    <form>
      <FormField label="Enter your name:" ref={ref} isRequired={true} />
      <button type="button" onClick={handleClick}>
        Edit
      </button>
    </form>
  );
}
```

Form 컴포넌트는 Ref를 정의하고 이를 FormField에 전달합니다. FormField 컴포넌트는 해당 Ref를 MyInput으로 전달하고, 이 컴포넌트는 <input> DOM 노드로 전달합니다. 이것이 Form이 <input> DOM 노드에 접근하는 방식입니다.

App.js FormField.js MyInput.js

↻ 새로고침 ✕ Clear 📄 포크

```
import { useRef } from 'react';
import FormField from './FormField.js';

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

  function handleClick() {
    ref.current.focus();
  }

  return (
    <form>
```

▼ 자세히 보기

DOM 노드 대신 명령형 핸들 노출하기

전체 DOM 노드를 노출하는 대신 제한된 메서드 집합과 함께 *명령형 핸들*이라고 하는 사용자 정의 객체를 노출할 수 있습니다. 이를 위해 DOM 노드를 보유할 별도의 Ref를 정의해야 합니다.

```
const MyInput = forwardRef(function MyInput(props, ref) {
  const inputRef = useRef(null);

  // ...

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

전달받은 ref를 `useImperativeHandle`에 전달하고 노출하려는 값을 ref에 지정합니다.

```
import { forwardRef, useRef, useImperativeHandle } from 'react';

const MyInput = forwardRef(function MyInput(props, ref) {
  const inputRef = useRef(null);

  useImperativeHandle(ref, () => {
    return {
      focus() {
```

```

        inputRef.current.focus();
    },
    scrollToView() {
        inputRef.current.scrollToView();
    },
};
}, []);

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

```

일부 컴포넌트가 MyInput 의 Ref를 받으면 DOM 노드 대신 { focus, scrollToView } 객체만 받습니다. 이를 통해 노출하는 DOM 노드의 정보를 최소한으로 제한할 수 있습니다.

App.js MyInput.js

↻ 새로고침 ✕ Clear 📌 포크

```

import { useRef } from 'react';
import MyInput from './MyInput.js';

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

    function handleClick() {
        ref.current.focus();
        // This won't work because the DOM node isn't exposed:
        // ref.current.style.opacity = 0.5;
    }
}

```

▼ 자세히 보기

명령형 핸들 사용에 대해 자세히 알아보세요.

⚠ 주의하세요!

ref를 과도하게 사용하지 마세요. 노드로 스크롤 하기, 노드에 포커스하기, 애니메이션 트리거하기, 텍스트 선택하기 등 prop로 표현할 수 없는 필수적인 동작에만 ref를 사용해야 합니다.

prop로 무언가를 표현할 수 있다면 ref를 사용해서는 안 됩니다. 예를 들어 Modal 컴포넌트에서 { open, close } 와 같은 명령형 핸들을 노출하는 대신 `<Modal isOpen={isOpen} />` 과 같이 prop isOpen 을 사용하는 것이 더 좋습니다. [Effects](#)는 props를 통해 명령형 동작을 노출하는 데 도움이 될 수 있습니다.

문제해결

컴포넌트가 forwardRef 로 감싸져 있지만, 컴포넌트의 ref 는 항상 null 입니다.

일반적으로 ref 를 실제로 사용하는 것을 잊어버렸다는 것입니다.

예를 들어 이 컴포넌트는 ref 로 아무것도 하지 않습니다.

```
const MyInput = forwardRef(function MyInput({ label }, ref) {  
  return (  
    <label>  
      {label}  
    <input />  
  )  
})
```

```

    </label>
  );
});

```

이 문제를 해결하려면 `ref` 를 DOM 노드나 `ref` 를 받을 수 있는 다른 컴포넌트에 전달하세요.

```

const MyInput = forwardRef(function MyInput({ label }, ref) {
  return (
    <label>
      {label}
      <input ref={ref} />
    </label>
  );
});

```

일부 로직이 조건부인 경우 `MyInput` 의 `ref` 가 `null` 일 수 있습니다.

```

const MyInput = forwardRef(function MyInput({ label, showInput }, ref) {
  return (
    <label>
      {label}
      {showInput && <input ref={ref} />}
    </label>
  );
});

```

`showInput` 이 `false` 이면 `ref` 가 어떤 노드로도 전달되지 않으며 `MyInput` 의 `ref` 는 비어 있게 됩니다. 이 예시의 `Panel` 과 같이 조건이 다른 컴포넌트 안에 숨겨져 있는 경우 특히 이 점을 놓치기 쉽습니다.

```

const MyInput = forwardRef(function MyInput({ label, showInput }, ref) {
  return (
    <label>
      {label}
      <Panel isExpanded={showInput}>
        <input ref={ref} />
      </Panel>
    </label>
  );
});

```

```
    </Panel>
  </label>
);
});
```

< 이전
[createRef](#)

다음 >
[isValidElement](#)

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](#)

