

Ref와 DOM

Ref는 render 메서드에서 생성된 DOM 노드나 React 엘리먼트에 접근하는 방법을 제공합니다.

일반적인 React의 데이터 플로우에서 props는 부모 컴포넌트가 자식과 상호작용할 수 있는 유일한 수단입니다. 자식을 수정하려면 새로운 props를 전달하여 자식을 다시 렌더링해야 합니다. 그러나, 일반적인 데이터 플로우에서 벗어나 직접적으로 자식을 수정해야 하는 경우도 가끔씩 있습니다. 수정할 자식은 React 컴포넌트의 인스턴스일 수도 있고, DOM 엘리먼트일 수도 있습니다. React는 두 경우 모두를 위한 해결책을 제공합니다.

Ref를 사용해야 할 때

Ref의 바람직한 사용 사례는 다음과 같습니다.

- 포커스, 텍스트 선택영역, 혹은 미디어의 재생을 관리할 때.
- 애니메이션을 직접적으로 실행시킬 때.
- 서드 파티 DOM 라이브러리를 React와 같이 사용할 때.

선언적으로 해결될 수 있는 문제에서는 ref 사용을 지양하세요.

예를 들어, Dialog 컴포넌트에서 `open()` 과 `close()` 메서드를 두는 대신, `isOpen` 이라는 prop을 넘겨주세요.

Ref를 남용하지 마세요

ref는 애플리케이션에 “어떤 일이 일어나게” 할 때 사용될 수도 있습니다. 그럴 때는 잠시 멈추고 어느 컴포넌트 계층에서 상태를 소유해야 하는지 신중하게 생각해 보세요. 대부분의 경우, 상태를 소유해야 하는 적절한 장소가 더 높은 계층이라는 결론이 날 겁니다. 상태를 상위 계층으로 올리는 것에 대한 예제는 상태 끌어올리기 가이드에서 확인하실 수 있습니다.



주의

아래에 있는 예제는 React 16.3 에서 추가된 `React.createRef()` API를 사용하도록 수정되었습니다. 이전 버전의 React를 사용하고 계신다면, 콜백 ref를 대신 사용하시는 편이 좋습니다.

Ref 생성하기

Ref는 `React.createRef()` 를 통해 생성되고 `ref` 어트리뷰트를 통해 React 엘리먼트에 부착됩니다. 보통, 컴포넌트의 인스턴스가 생성될 때 Ref를 프로퍼티로서 추가하고, 그럼으로서 컴포넌트의 인스턴스의 어느 곳에서도 Ref에 접근할 수 있게 합니다.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

Ref에 접근하기

`render` 메서드 안에서 `ref`가 엘리먼트에게 전달되었을 때, 그 노드를 향한 참조는 `ref.current` 어트리뷰트에 담기게 됩니다.

```
const node = this.myRef.current;
```

`ref`의 값은 노드의 유형에 따라 다릅니다.

- `ref` 어트리뷰트가 HTML 엘리먼트에 쓰였다면, 생성자에서 `React.createRef()` 로 생성된 `ref`는 자신을 전달받은 DOM 엘리먼트를 `current` 프로퍼티의 값으로서 받습니다.
- `ref` 어트리뷰트가 커스텀 클래스 컴포넌트에 쓰였다면, `ref` 객체는 마운트된 컴포넌트 인스턴스를 `current` 프로퍼티의 값으로서 받습니다.
- **함수 컴포넌트는 인스턴스가 없기 때문에 함수 컴포넌트에 `ref` 어트리뷰트를 사용할 수 없습니다.**

니다.

아래의 예제들은 위에서 언급한 차이점들을 보여줍니다.

DOM 엘리먼트에 Ref 사용하기

아래의 코드는 DOM 노드에 대한 참조를 저장하기 위해 `ref` 를 사용합니다.

```
class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);
    // textInput DOM 엘리먼트를 저장하기 위한 ref를 생성합니다.
    this.textInput = React.createRef();
    this.focusTextInput = this.focusTextInput.bind(this);
  }

  focusTextInput() {
    // DOM API를 사용하여 명시적으로 text 타입의 input 엘리먼트를 포커스합니다.
    // 주의: 우리는 지금 DOM 노드를 얻기 위해 "current" 프로퍼티에 접근하고 있습니다.
    this.textInput.current.focus();
  }

  render() {
    // React에게 우리가 text 타입의 input 엘리먼트를
    // 우리가 생성자에서 생성한 `textInput` ref와 연결하고 싶다고 이야기합니다.
    return (
      <div>
        <input
          type="text"
          ref={this.textInput} />
        <input
          type="button"
          value="Focus the text input"
          onClick={this.focusTextInput}
        />
      </div>
    );
  }
}
```

컴포넌트가 마운트될 때 React는 `current` 프로퍼티에 DOM 엘리먼트를 대입하고, 컴포넌트의 마운트가 해제될 때 `current` 프로퍼티를 다시 `null`로 돌려 놓습니다. `ref`를 수정하는 작업은 `componentDidMount` 또는 `componentDidUpdate` 생명주기 메서드가 호출되기 전에 이루어집니다.

클래스 컴포넌트에 ref 사용하기

아래에 있는 CustomTextInput 컴포넌트의 인스턴스가 마운트 된 이후에 즉시 클릭되는 걸 흉내내기 위해 CustomTextInput 컴포넌트를 감싸는 걸 원한다면, ref를 사용하여 CustomTextInput 컴포넌트의 인스턴스에 접근하고 직접 focusTextInput 메서드를 호출할 수 있습니다.

```
class AutoFocusTextInput extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }

  componentDidMount() {
    this.textInput.current.focusTextInput();
  }

  render() {
    return (
      <CustomTextInput ref={this.textInput} />
    );
  }
}
```

위 코드는 CustomTextInput 가 클래스 컴포넌트일 때에만 작동한다는 점을 기억하세요.

```
class CustomTextInput extends React.Component {
  // ...
}
```

Ref와 함수 컴포넌트

- 함수 컴포넌트는 인스턴스가 없기 때문에 함수 컴포넌트에 ref 어트리뷰트를 사용할 수 없습니다.

```
function MyFunctionComponent() {
  return <input />;
}

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }
```



```

}
render() {
  // 이 코드는 동작하지 않습니다.
  return (
    <MyFunctionComponent ref={this.textInput} />
  );
}
}

```

함수 컴포넌트에 `ref`를 사용할 수 있도록 하려면, `forwardRef` (높은 확률로 `useImperativeHandle`와 함께)를 사용하거나 클래스 컴포넌트로 변경할 수 있습니다.

다만, DOM 엘리먼트나 클래스 컴포넌트의 인스턴스에 접근하기 위해 **`ref` 어트리뷰트를 함수 컴포넌트에서 사용하는 것은 됩니다.**

```

function CustomTextInput(props) {
  // textInput은 ref 어트리뷰트를 통해 전달되기 위해서
  // 이곳에서 정의되어야만 합니다.
  const textInput = useRef(null);

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

  return (
    <div>
      <input
        type="text"
        ref={textInput} />
      <input
        type="button"
        value="Focus the text input"
        onClick={handleClick}
      />
    </div>
  );
}

```

부모 컴포넌트에게 DOM ref를 공개하기

보기 드문 경우지만, 부모 컴포넌트에서 자식 컴포넌트의 DOM 노드에 접근하려 하는 경우가 있습니다. 자식 컴포넌트의 DOM 노드에 접근하는 것은 컴포넌트의 캡슐화를 파괴하기 때문에 권장되지 않습니다. 그렇지만 가끔가다 자식 컴포넌트의 DOM 노드를 포커스하는 일이나, 크기 또는 위치를 계산하는 이 드문 하 때에는 추가적이 바뀌어 될 수 있습니다.

이 기사를 개인적인 블로그에 올리는 것에 동의하지 않습니다. 이 글은 저작권에 의해 보호되는 저작물입니다.

자식 컴포넌트에 ref를 사용할 수 있지만, 이 방법은 자식 컴포넌트의 인스턴스의 DOM 노드가 아닌 자식 컴포넌트의 인스턴스를 가져온다는 점에서, 자식 컴포넌트가 함수 컴포넌트인 경우에는 동작하지 않는다는 점에서, 좋은 방법이 아닙니다.

React 16.3 이후 버전의 React를 사용하신다면 위와 같은 경우에서 ref 전달하기(ref forwarding)을 사용하는 것이 권장됩니다. **Ref 전달하기는 컴포넌트가 자식 컴포넌트의 ref를 자신의 ref로서 외부에 노출시키게 합니다.** 자식 컴포넌트의 DOM 노드를 부모 컴포넌트에게 공개하는 방법에 대한 자세한 예시는 ref 넘겨주기 문서에서 볼 수 있습니다..

React 16.2 이전 버전을 사용하시거나 ref 전달하기보다 더 유연한 방법을 원한다면 이런 대안을 사용할 수 있습니다.

가능하다면 DOM 노드를 외부에 공개하는 일을 지양해야 합니다만 DOM 노드를 외부에 공개하는 일은 유용한 해결책이 될 수 있습니다. 또한 이 방법들은 자식 컴포넌트의 코드 수정을 요한다는 점을 기억하세요. 만약 자식 컴포넌트의 코드를 수정할 수 없다면 최후의 방법인 findDOMNode()를 사용하는 방법이 있지만 findDOMNode()는 좋지 못한 방법일 뿐더러 StrictMode에서 사용할 수 없습니다.

콜백 ref

React는 ref가 설정되고 해제되는 상황을 세세하게 다룰 수 있는 “콜백 ref” 이라 불리는 ref를 설정하기 위한 또 다른 방법을 제공합니다.

콜백 ref를 사용할 때에는 ref 어트리뷰트에 `React.createRef()`를 통해 생성된 ref를 전달하는 대신, 함수를 전달합니다. 전달된 함수는 다른 곳에 저장되고 접근될 수 있는 React 컴포넌트의 인스턴스나 DOM 엘리먼트를 인자로서 받습니다.

아래의 예시는 DOM 노드의 참조를 인스턴스의 프로퍼티에 저장하기 위해 ref 콜백을 사용하는 흔한 패턴을 보여줍니다.

```
class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);

    this.textInput = null;

    this.setTextInputRef = element => {
      this.textInput = element;
    };
  }
}
```



```

    this.focusTextInput = () => {
      // DOM API를 사용하여 text 타입의 input 엘리먼트를 포커스합니다.
      if (this.textInput) this.textInput.focus();
    };
  }

  componentDidMount() {
    // 마운트 되었을 때 자동으로 text 타입의 input 엘리먼트를 포커스합니다.
    this.focusTextInput();
  }

  render() {
    // text 타입의 input 엘리먼트의 참조를 인스턴스의 프로퍼티
    // (예를 들어 `this.textInput`)에 저장하기 위해 `ref` 콜백을 사용합니다.
    return (
      <div>
        <input
          type="text"
          ref={this.setTextInputRef}
        />
        <input
          type="button"
          value="Focus the text input"
          onClick={this.focusTextInput}
        />
      </div>
    );
  }
}

```

컴포넌트의 인스턴스가 마운트 될 때 React는 ref 콜백을 DOM 엘리먼트와 함께 호출합니다. 그리고 컴포넌트의 인스턴스의 마운트가 해제될 때, ref 콜백을 null 과 함께 호출합니다. ref 콜백들은 componentDidMount 또는 componentDidUpdate 가 호출되기 전에 호출됩니다.

콜백 ref 또한 React.createRef() 를 통해 생성했었던 객체 ref와 같이 다른 컴포넌트에게 전달 할 수 있습니다.

```

function CustomTextInput(props) {
  return (
    <div>
      <input ref={props.inputRef} />
    </div>
  );
}

```

```

class Parent extends React.Component {

```

```
class Parent extends React.Component {  
  render() {  
    return (  
      <CustomTextInput  
        inputRef={el => this.inputElement = el}  
      />  
    );  
  }  
}
```

위의 예시에서 Parent 는 자신의 콜백 ref를 inputRef prop으로서 CustomTextInput 에게 전달합니다. 그리고 CustomTextInput 은 전달받은 함수를 <input>에게 ref 어트리뷰트로서 전달합니다. 결과적으로 Parent 에 있는 this.inputElement 는 CustomTextInput 의 <input> 엘리먼트에 대응하는 DOM 노드가 됩니다.

레거시 API: 문자열 ref

리엑트를 이전에 사용해 보셨다면 ref 어트리뷰트의 값이 "textInput" 처럼 문자열이고, DOM 노드를 this.refs.textInput 와 같이 접근하는 구식 API를 아시고 계실지도 모릅니다. 문자열 ref는 몇몇 문제를 가지고 있고, 레거시로 여겨지며, **차후 배포에서 삭제될 것으로 예상**되기 때문에 권장되지 않습니다.



주의

ref에 접근하기 위해 this.refs.textInput 를 사용하고 계신다면, 콜백 ref이나 createRef API를 대신 사용하는 것을 권해 드립니다.

콜백 ref에 관한 주의사항

ref 콜백이 인라인 함수로 선언되었다면 ref 콜백은 업데이트 과정 중에 처음에는 null 로, 그 다음에는 DOM 엘리먼트로, 총 두 번 호출됩니다. 이러한 현상은 매 렌더링마다 ref 콜백의 새 인스턴스가 생성되므로 React가 이전에 사용된 ref를 제거하고 새 ref를 설정해야 하기 때문에 일어납니다. 이러한 현상은 ref 콜백을 클래스에 바인딩된 메서드로 선언함으로써 해결할 수 있습니다. 하지만 많은 경우 이러한 현상은 문제가 되지 않는다는 점을 기억하세요.



Is this page useful?  

[Edit this page](#)

