React Event

개요

React에서 JSX에 Event handler를 추가할 수 있다. Event handler는 Click, Mouse hover, Form input, Focus 등 사용자 상호작용에 따라 유발되는 사용자 정의 함수이다.

Add Event handlers

Event handler를 추가하기 위해 함수를 정의하고 JSX 태그에 prop 형태로 전달한다.

```
export default function Button() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

예제에 `handleClick` 함수를 정의하고 `<button>`태그에 prop 형태로 전달한다. 여기서 `handlerClick`은 Event handler이다. Event handler 함수는 다음과 같은 특징이 있다.

- 주로 Component 내부에 정의된다.
- `handle`로 시작하고 그 뒤에 이벤트명을 붙인 함수명을 갖는다.

Event handler를 JSX 내에서 인라인으로 정의할 수도 있다.

```
<button onClick={function handleClick() {
  alert('You clicked me!');
}}>
```

또는 람다식으로 간결하게 정의할 수도 있다.

```
<button onClick={() => {
  alert('You clicked me!');
}}>
```

Event handler로 전달한 함수들은 호출이 아닌 전달

함수 전달	함수 호출
<button onClick={handleClick}>	<button onClick={handleClick()}>
<button onClick={() => alert('…')}>	<button onClick={alert('…')}>

함수에 이벤트 핸들러를 전달하면 이후 React는 이 내용을 기억하고 있다가 사용자가 버튼을 클릭하였을 때만 함수를 호출하도록 한다.

함수에서 이벤트 핸들러를 호출하면 렌더링 과정 중 클릭이 없었음에도 불구하고 즉시 함수를 실행하도록 만든다. 이는 JSX에서 중괄호 사이에 있는 JavaScript가 즉시 실행되기 때문이다.

Event handler를 Prop으로 전달

부모 컴포넌트로부터 자식 컴포넌트의 이벤트 핸들러를 지정하는 경우가 있을 수 있다. 이런 경우 prop으로 핸들러를 전달하면 된다.

```
function Button({ onClick, children }) {
  return (
    <button onClick={onClick}>
      {children}
    </button>
  );
}

function PlayButton({ movieName }) {
  function handlePlayClick() {
    alert(`Playing ${movieName}!`);
  }

  return (
    <Button onClick={handlePlayClick}>
      Play "{movieName}"
    </Button>
  );
}

export default function Toolbar() {
  return (
      <PlayButton movieName="Kiki's Delivery Service" />
  );
}
```

위 예제에서 `<button>` 컴포넌트는 onClick prop을 받는다. `<button>` 의 `onClick` 속성에 함수를 전달한다.

Event handler prop naming

`<button>` 이나 `<div>` 와 같은 built-in 컴포넌트는 `onClick` 과 같이 브라우저 이벤트 이름만 지원한다. 그러나 Custom components는 Event handler prop의 이름을 직접 명명할 수 있다.

```
function Button({ onSmash, children }) {
  return (
    <button onClick={onSmash}>
      {children}
    </button>
  );
}
```

```
export default function App() {
  return (
    <div>
      <Button onSmash={() => alert('Playing!')}>
        Play Movie
      </Button>
      <Button onSmash={() => alert('Uploading!')}>
        Upload Image
      </Button>
    </div>
  );
}
```

이벤트 전파

```
export default function Toolbar() {
  return (
    <div className="Toolbar" onClick={() => {
      alert('You clicked on the toolbar!');
    }}>
      <button onClick={() => alert('Playing!')}>
        Play Movie
      </button>
      <button onClick={() => alert('Uploading!')}>
        Upload Image
      </button>
    </div>
  );
}
```

만약 위와 같은 코드가 있다면 어떻게 동작할까. `<div>`태그만 클릭하면 해당 `onClick`만 호출된다. 그러나 만약 `<div>`태그 안에 `<button>`태그를 클릭한다면 부모, 자식 이벤트가 모두 동작하여 `alert`함수가 두 번 호출된다.

전파 멈추기

Event handler는 event object를 유일한 매개변수로 받는다. 관습에 따라 'event'를 의미하는 `e`로 호출하는 것이 일반적이다. event object는 이벤트 정보를 읽는데 사용할 수 있다. 따라서 event object가 이벤트 전파를 멈출 수 있게 한다. 부모 컴포넌트에 이벤트가 닿지 못하도록 막으려면 `e.stopPropagation()`을 호출한다.

```
function Button({ onClick, children }) {
  return (
    <button onClick={e => {
      e.stopPropagation();
      onClick();
    }}>
      {children}
    </button>
  );
}

export default function Toolbar() {
  return (
    <div className="Toolbar" onClick={() => {
      alert('You clicked on the toolbar!');
    }}>
```

```
      <Button onClick={() => alert('Playing!')}>
        Play Movie
      </Button>
      <Button onClick={() => alert('Uploading!')}>
        Upload Image
      </Button>
    </div>
  );
}
```

1. React가 `<button>`에 전달된 `onClick`을 호출한다.
2. `Button`에 정의된 핸들러는 다음을 수행한다.
   a. `e.stopPropagation()`을 호출하여 이벤트가 더 이상 bubbling 되지 않도록 방지한다.
   b. `Toolbar`컴포넌트가 전달해 준 `onClick`을 호출한다.
3. `Button`에 전달된 함수가 `alert`를 호출한다.
4. 전파가 중단되었기 때문에 부모인 `<div>`의 `onClick`은 실행되지 않는다.

결과적으로 버튼을 클릭하는 행위는 주변 툴바 부분을 클릭하는 것과 다른 행위이기 때문에 이 UI상에서는 이벤트 전파를 멈추게 하는 것이다.

기본 동작 방지

일부 브라우저 이벤트는 기본 동작을 갖고 있다. 예를 들어, `<form>`태그는 기본 동작으로 제출 이벤트를 발생시키는데 버튼을 클릭하면 페이지 전체를 리로드한다. 이런 기본 동작을 방지하기 위해서 `e.preventDefault()`를 이벤트 오브젝트에서 호출할 수 있다.

```
export default function Signup() {
  return (
    <form onSubmit={e => {
      e.preventDefault();
      alert('Submitting!');
    }}>
      <input />
      <button>Send</button>
    </form>
  );
}
```

- `e.stopPropagation()` : 이벤트 핸들러가 상위 태그에서 실행되지 않도록 멈춘다.
- `e.preventDefault()` : 기본 브라우저 동작을 가진 일부 이벤트가 해당 기본 동작을 실행하지 않도록 방지한다.

이벤트 핸들러와 사이드 이펙트

이벤트 핸들러는 사이드 이펙트를 가질 수 있다. 이벤트 핸들러는 함수를 렌더링하는 것과 다르게 순수할 필요가 없기 때문이며 무언가를 변경하기 가장 좋은 위치이다. 물론 일부 정보를 수정하기 위해서는 반드시 정보를 저장하고 사용해야 하기 때문에 이런 수단이 필요한데 React에서는 컴포넌트의 정보 저장에 state를 활용할 수 있다.

참고 자료 :

이벤트에 응답하기 – React