09-hook-event

#01. 리액트 이벤트 시스템

1) 이벤트

프로그램이 겪는 어떠한 사건.

- 사용자의 클릭, 마우스 오버
- 페이지 로딩 직후, 페이지 종료 직전

자바스크립트가 제공하는 브라우저 관련 기능 안에 이미 다양한 이벤트를 감지하는 기능이 구현되어 내장 기능으로 개발자에게 제공된다.

2) Javascript에서 일반적으로 이벤트를 처리하는 방법

이벤트 리스너

어떤 이벤트가 발생하기를 기다리고 있다가 이벤트 발생시 연결된 함수를 호출해 주는 기능.

이벤트를 감지하는 것은 Javascript 안에 이미 내장되어 있지만 어떤 이벤트가 발생했을 수행할 동작은 웹 페이지마다 다를 수 밖에 없으므로 미리 구현해 놓을 수 없다. (페이지마다 개별 구현이 필요함)

이벤트를 감지하는 기능이 수행할 동작을 콜백함수로 요구한다.

```
객체.addEventListener('이벤트이름', 콜백함수);
```

이벤트 핸들러

HTML 태그의 속성 형태로 존재하는, 이벤트 발생시 실행할 자바스크립트 코드 영역

Javascript가 고도화 됨에 따라 HTML과 Javascript 코드를 분리하는 경향이 생기면서 이벤트 핸들러는 거의 사용되지 않는 추세.

```
<태그이름 on이벤트이름="...JS코드영역..."></태그이름>
```

3) 리액트에서 이벤트 구현

리액트에서는 JSX 안에서 이벤트 핸들러 방식으로 구현함.

주의사항

1. 이벤트 리스너의 이름은 HTML속성이 아닌 JSX에 의한 자바스크립트 프로퍼티이므로 카멜 표기법으로 작성.
    - onclick (X)
    - onClick (O)
2. 이벤트 리스너에 전달할 이벤트 핸들러는 코드 형태가 아니라 반드시 함수 형태로 전달해야 한다.

3. DOM 요소(=HTML 태그)에만 이벤트 리스너가 존재한다.
    - 직접 구현한 컴포넌트에 대해서는 설정 불가

#02. Hooks

함수형 컴포넌트에서 상태값(state)를 관리하기 위한 기능으로 클래스형 컴포넌트의 LifeCycle에 대응된다.

> React v16.8부터 새로 추가되었음.

쉽게 이야기 하면 아래의 항목들은 특정 상황에서 자동으로 호출되는 함수들을 의미함.

1) 기본 Hook 함수들

a) useState

`useState()` 함수를 import하고 사용하는 경우.

```
import React, {useState} from 'react';
...
const [상태변수, 변수에대한setter함수] = useState(초기값);
```

`useState()` 함수를 import하지 않고 직접 사용하는 경우.

```
const [상태변수, 변수에대한setter함수] = React.useState(초기값);
```

- 가장 기본적인 Hook 함수
- 함수형 컴포넌트에서 state값을 생성한다.
- 하나의 useState 함수는 하나의 상태 값만 관리할 수 있다.
- 컴포넌트에서 관리해야 할 상태가 여러 개라면 useState를 여러번 사용하면 된다.

b) useEffect

useEffect는 기본적으로 렌더링 직후마다 실행되며, 두 번째 파라미터 배열에 무엇을 넣는지에 따라 실행되는 조건이 달라진다.

> 클래스형 컴포넌트의 **componentDidMount**와 **componentDidUpdate**를 합친 형태

렌더링 될 때마다 실행되는 함수 정의

최초 등장하거나 state값이 변경될 때 모두 실행된다.

```
useEffect(() => {
    ... 처리할 코드 ...
});
```

업데이트시에는 생략되는 함수 정의

컴포넌트가 마운트될 때 최초 1회만 실행된다. (state값이 변경될 때는 실행되지 않는다.)

```
useEffect(() => {
    ... 처리할 코드 ...
}, []);
```

특정 state, props값이 변경될 때만 호출되도록 설정하기

```
useEffect(() => {
    ... 처리할 코드 ...
}, [값이름]);
```

컴포넌트가 언마운트(화면에서 사라짐) 될 때만 실행되도록 설정하기

클로저(리턴되는 함수)를 명시한다.

```
useEffect(() => {
    return function() {
        ... 처리할 코드 ...
    };
}, []);
```

```
useEffect(() => {
    return () => {
        ... 처리할 코드 ...
    };
}, []);
```

c) useMemo

함수형 컴포넌트 내에서의 연산 최적화.

숫자, 문자열, 객체처럼 일반 값을 재사용하고자 할 경우 사용한다.

useMemo는 특정 상태변수가 변경되었을 때 그 변경된 값을 가지고 처리할 후속 작업을 구현한다. 변경여부를 감지하기 위한 상태변수는 useMemo()의 두 번째 파라미터 배열에 명시한다. useMemo의 콜백함수에서 리턴하는 값은 useMemo 자체의 리턴값이 된다.

d) useRef

함수형 컴포넌트에서 ref를 쉽게 사용할 수 있도록 처리해 준다.

Vanilla Script에서 `document.getElementById(...)`나 `document.querySelector(...)`로 DOM 객체를 취득하는 과정을 React 스타일로 표현한 것으로 이해할 수 있다.

e) useReducer

useState 보다 더 다양한 컴포넌트 상황에 따라 다양한 상태를 다른 값으로 업데이트 하고자 하는 경우 사용.

useState의 대체 함수로 이해할 수 있다.

state값이 다수의 하위값을 포함하거나 이를 활용하는 복잡한 로직을 만드는 경우에 useState보다 useReducer를 선호한다.

f) useCallback

렌더링 성능 최적화에 사용됨.

이벤트 핸들러 함수를 필요한 경우에만 생성할 수 있다.

> memoized 된 콜백을 반환한다.

g) useContext

거의 사용하지 않음.

2) Hook 사용시 주의사항

1. 반복문, 조건문, 중첩된 함수 내에서 Hook을 실행할 수 없다.
2. React Component 내에서만 호출할 수 있다.