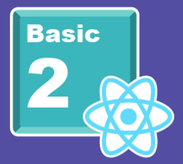
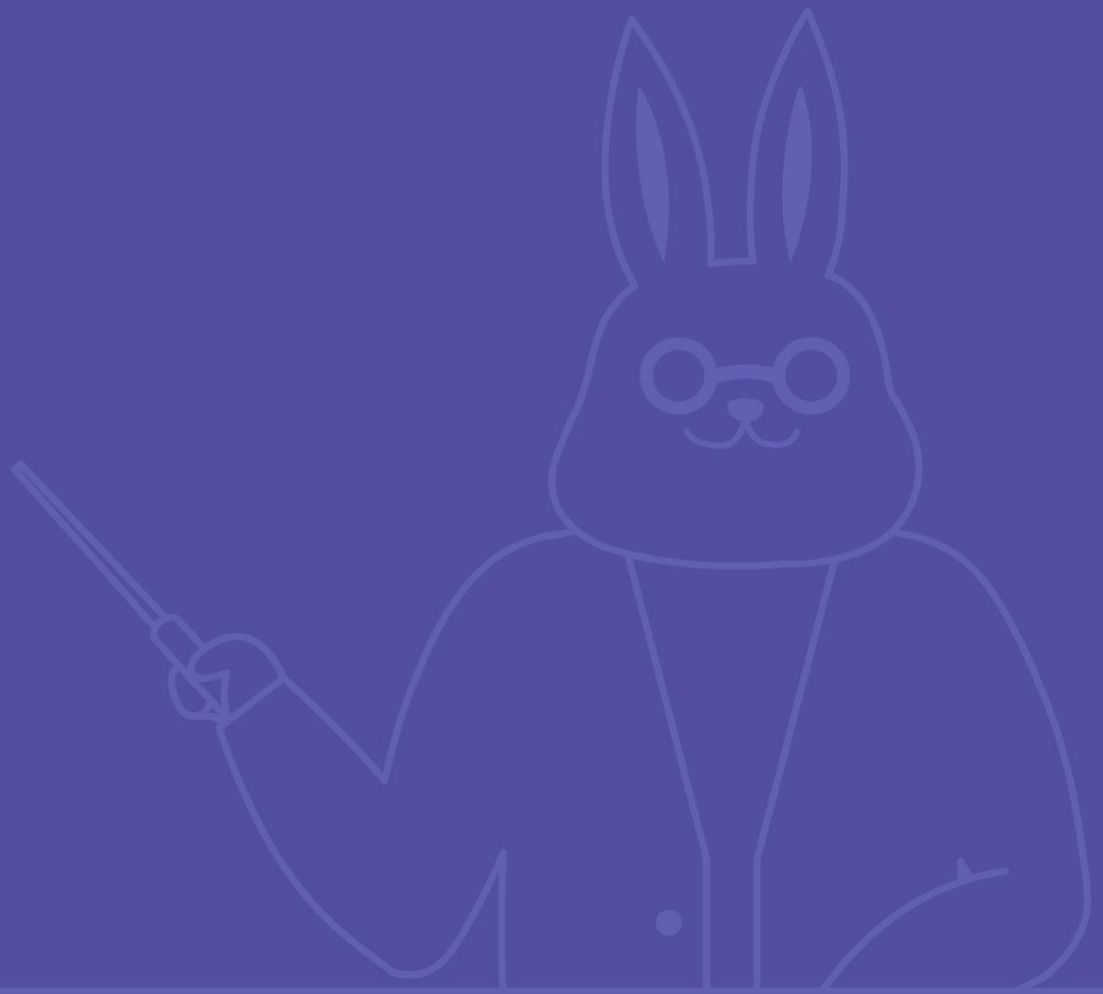


React 기초 II

03 Hooks



/* elice */

목차

01

Hooks 개요

Hook이란?

컴포넌트에서 데이터를 관리(State)하고
데이터가 변경될 때 상호작용(Effect)을 하기 위해 사용합니다.

앞선 강의에서 사용해본 useState가 바로 **State Hook**입니다.

Hook의 등장 배경

기존에는 컴포넌트 내에서 State와 생명주기를 관리하기 위해서
반드시 클래스 컴포넌트(Class Component)를 사용하여야 했습니다.

그러나 개발자가 느끼기에 다소 복잡한 클래스 컴포넌트(Class Component)를 보완하고 함수 컴포넌트에서
클래스 컴포넌트의 기능을 구현하기 위해 React 16.8 버전에 추가된 것이 바로 **Hook**입니다.

유의사항

코드

```
const App = () => {
  const [username, setUsername] = useState('')
  return (
    <div>
      <h1>{username}님 환영합니다.</h1>
    </div>
  )
}
```

Hook은 **React 함수(컴포넌트, Hook)** 내에서만 사용이 가능합니다.

Hook의 이름은 반드시 **‘use’**로 시작해야합니다.

최상위 Level에서만 Hook을 호출할 수 있습니다.

(if문, for문 안 쪽 또는 콜백함수 내에서 호출하지 마세요.)

02

State Hook과 Effect Hook

State Hook

코드

```
const App = () => {
  // 일반적인 useState 사용법
  const [state이름, setState이름] = useState(초기값)
}
```

- useState는 컴포넌트 내 동적인 데이터를 관리할 수 있는 hook입니다.
- 최초에 useState가 호출될 때 초기값으로 설정되며 이후 재 렌더링이 될 경우 무시됩니다.
- state는 읽기 전용이므로 직접 수정하지 마세요.
- state를 변경하기 위해서는 setState를 이용합니다.
- state가 변경되면 자동으로 컴포넌트가 재 렌더링됩니다.

State Hook

```
const App = () => {
 const [title, setTitle] = useState("")

  // State를 변경할 값을 직접 입력
  setTitle("Hello")

  // 또는 현재 값을 매개변수로 받는 함수 선언
  // return 값이 state에 반영됨
  setTitle((current) => {
    return  current + "World";
  })
}
```

- state를 변경하기 위해서는 setState 함수에 직접 값을 입력하거나

- 현재 값을 매개변수로 받는 함수를 전달합니다. 이 때 함수에서 return되는 값이 state에 반영됩니다.

Effect Hook

코드

```
const App = () => {
  useEffect(EffectCallback, Deps?)
}
```

- Effect Hook을 사용하면 함수 컴포넌트에서 side effect를 수행할 수 있습니다.
- 컴포넌트가 최초로 렌더링될 때, 지정한 State나 Prop가 변경될 때마다 이펙트 콜백 함수가 호출됩니다.

- **Deps**: 변경을 감지할 변수들의 집합(배열)
- **EffectCallback**: Deps에 지정된 변수가 변경될 때 실행할 함수

Effect Hook

예시

```
const App = () => {
  const [count, setCount] = useState(0)

  useEffect(() => {
    console.log('버튼을 ${count}회 클릭했습니다.')
  }, [count])

return (
    <div>
      <button onClick={() => setCount(count + 1)}>
        클릭하세요
      </button>
    </div>
  )
}
```

- Effect Hook을 사용하면 함수 컴포넌트에서 side effect를 수행할 수 있습니다.
- 컴포넌트가 최초로 렌더링될 때, 지정한 State나 Prop가 변경될 때마다 이펙트 콜백 함수가 호출됩니다.

Effect Hook

예시

```
const App = () => {
  useEffect(() => {
    // State가 변경될 때, 컴포넌트를 렌더링할 때
    const intervalId = setInterval(()=>{
      console.log("안녕하세요");
    }, 1000);

    // 컴포넌트를 재 렌더링 하기 전에, 컴포넌트가
    // 없어질 때
    return () => {
      clearInterval(intervalId);
    }
  }, [])
...
```

useEffect의 이펙트 함수 내에서 다른 함수를 return할 경우 state가 변경되어 컴포넌트가 다시 렌더링되기 전과 컴포넌트가 없어질 때(Destroy) 호출할 함수를 지정하게 됩니다.

03

이외의 Hooks

다양한 내장 Hook들이 있습니다

useMemo

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b])
```

useCallback

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

useRef

```
const refContainer = useRef(initialValue);
```

useMemo

코드

```
const App = () => {
  const [firstName, setFirstName] = useState('철수')
  const [lastName, setLastName] = useState('김')

  // 이름과 성이 바뀔 때마다 풀네임을 메모이제이션
  const fullName = useMemo(() => {
    return `${firstName} ${lastName}`
  }, [firstName, lastName])
}
```

- 지정한 State나 Props가 변경될 경우 해당 값을 활용해 **계산된 값**을 **메모이제이션**하여 재렌더링 시 불필요한 연산을 줄입니다.
- useMemo의 연산은 렌더링 단계에서 이루어지기 때문에 시간이 **오래 걸리는 로직을 작성하지 않는 것이 권장**됩니다.

useCallback

코드

```
const App = () => {
  const [firstName, setFirstName] = useState('철수')
  const [lastName, setLastName] = useState('김')

  // 이름과 성이 바뀔 때마다
  // 풀네임을 return하는 함수를 메모이제이션
  const getfullName = useCallback(() => {
    return `${firstName} ${lastName}`
  }, [firstName, lastName])

  return <>{fullname}</>
}
```

- **함수**를 **메모이제이션**하기 위해 사용하는 Hook입니다. 컴포넌트가 재렌더링될 때 불필요하게 함수가 다시 생성되는 것을 방지합니다.

- `useMemo(() => fn, deps)`와 `useCallback(fn, deps)`는 같습니다.

useRef

코드

```
const App = () => {
  const inputRef = useRef(null)
  const onButtonClick = () => {
    inputRef.current.focus()
  }
  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={onButtonClick}>
        input으로 포커스
      </button>
    </div>
  )
}
```

- **컴포넌트 생애 주기 내에서 유지**할 ref 객체를 반환합니다.
- ref 객체는 **.current**라는 프로퍼티를 가지며 이 값을 자유롭게 변경할 수 있습니다.
- 일반적으로 React에서 **DOM Element에 접근**할 때 사용합니다(DOM Element의 **ref 속성**을 이용합니다.)
- useRef에 의해 반환된 ref 객체가 **변경되어도** 컴포넌트가 **재렌더링되지 않습니다.**

04

나만의 Hook 만들기

Custom Hook

자신만의 Hook을 만들면 컴포넌트 로직을 함수로 뽑아내어 재사용할 수 있습니다.
UI 요소의 재사용성을 올리기 위해 컴포넌트를 만드는 것 처럼, 로직의 재사용성을 높이기 위해서는 Custom Hook을 제작합니다.

예시

```
function useMyHook(args) {
  const [status, setStatus] = useState(null);
  // ...
  return status;
}
```

- 한 로직이 여러 번 사용될 경우 함수를 분리하는 것 처럼 Hook을 만드는 것일 뿐, 새로운 개념은 아닙니다.
- Hook의 이름은 ‘use’로 시작해야합니다.
- 한 Hook 내의 state는 공유되지 않습니다.

크레딧

/* elice */

코스 매니저
강윤수

콘텐츠 제작자
마로

강사
마로

감수자
장석준

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io