1. React Hook 개요

Hook?

- 함수형 컴포넌트에서 상태(State)와 생명주기 등과 관련된 기능을 사용할 수 있도록 하는 새로운 기능
  - props, state, context, refs, and lifecycle
- 16.8 ~ 에서 지원
- 필수는 아님. 클래스 컴포넌트는 계속해서 지원함.

왜 Hook을 사용하는가?

- 성능 : 함수형 컴포넌트 > 클래스 컴포넌트
- 기능 : 함수형 컴포넌트(only render!!) + Hook(다양한 기능)
- 하지만 클래스 컴포넌트를 완벽히 대체하지는 못함.
  - 예) 생명주기 메서드 중에서 componentDidMount(), componentDidUpdate(), componentWillUnmount 만을 처리함.

2. useState

기능

- 함수형 컴포넌트에서 상태를 사용할 수 있도록 함
- 예제 : react-hook-test 프로젝트
  - App01.js 를 참조. 실행을 위해서 index.js의 import 구문을 변경한다.

```
import React, { useState } from "react";

function App() {
  const [msg, setMsg] = useState("React!!");

  return (
    <div>
      <input type="text" value={msg} onChange={e => setMsg(e.target.value)} />
      <span>Hello {msg}</span>
    </div>
  );
}
export default App;
```

2

■ useState([상태에 지정할 기본값]) 으로 호출하면 배열을 리턴하는데 그 값은 다음과 같음

- 첫번째 배열 값 : 읽기 전용의 속성(getter)
- 두번째 배열 값 : setState()를 대신하는 상태 값을 설정하는 메서드(setter)

3. useEffect(1)

기능

- 생명주기 메서드 중 componentDidUpdate, componentDidMount, componentWillUnmount 기능을 수행함.

간단한 카운터 예제(App02.js 참조)

- componentDidMount, componentDidUpdate 시에 함수가 호출됨.

```
import React, { useState, useEffect } from "react";

function App() {
  const [count, setCount] = useState(0);
  useEffect(() => {
    console.log(`You clicked ${count} times`);
  });
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

3. useEffect(2)

특정한 state, props가 바뀌었을 때만 함수가 호출되도록...

- 배열값으로 state나 props를 지정함.
- App03.js 참조

```
useEffect(() => {
    console.log(`You clicked ${count} times`);
}, [count]);
```

빈 배열을 지정하면

- componentDidMount 경우에만 함수가 호출됨.
- App04.js 참조

```
useEffect(() => {
  console.log(`You clicked ${count} times`);
}, []);
```

4

■ useEffect() 함수의 두번째 인자에 배열로 의존객체를 지정할 수 있음.

- 의존 객체로 등록된 것이 변경될 때만 함수가 실행됨.

■ App04와 같이 의존 객체를 등록하지 않으면?

- 처음 마운트될 때만 실행됨

■ App04.js 예제를 실행해보면 서버를 실행시킨 콘솔에 다음과 같은 경고 메시지가 나타남. 이유는 무엇인가?

WARNING in src₩App04.js

Line 8:6: React Hook useEffect has a missing dependency: 'count'. Either include it or remove the dependency array react-hooks/exhaustive-deps

- useEffect()에 인자로 전달한 실행 함수가 상태 데이터를 사용하고 있지만 의존 객체를 등록하지 않았기 때문에 경고하는 것임
- 실행함수내에서 상태 데이터를 사용하지 않도록 변경하면 경고는 사라짐.

3. useEffect(3)

componentWillUnmount 처리

- 컴포넌트가 unmount될 때의 처리를 수행함
- 예1) 간단한 시계를 만든다면?
  - componentDidMount() 에서 setInterval()을 이용해 주기적으로 시간정보를 갱신하고
  - componentWillUnmount() 에서 clearInterval()에서 주기적인 실행을 중단시킴.
- 예2) websocket으로 네트워크 연결이 필요하다면?
  - componentDidMount() 에서 소켓을 연결하고
  - componentWillUnmount() 에서 소켓 연결을 해제한다.

- component가 unmount되었을 때 소켓 연결을 유지하거나 주기적인 실행을 하고 있을 필요가 없음.

3. useEffect(4)

componentWillUnmount 처리(이어서)

- 5장에서 작성한 digital-clock-app 프로젝트 코드를 변경
  - App.js에서는 Clock 컴포넌트를 Toggle함
    - 그 결과로써 Clock 컴포넌트는 mount -> unmount -> mount -> unmount -> .....
  - Clock.js에서는...
    - componentDidMount()에서 setInterval() 로 주기적인 시간 갱신 수행.
    - componentWillUnmount()에서 clearInterval()로 주기적인 갱신 작업 중단.
  - Clock2.js 작성(Clock.js의 UI를 참조해서 작성하세요)

```
const [currentTime, setCurrentTime] = useState(new Date());

useEffect(() => {
  console.log("### Clock component is mounted!!");
  let handle = setInterval(() => {
    setCurrentTime(new Date());
    console.log("### Time is updated");
  }, 1000);

  return () => {
    console.log("### Clock component will be unmounted!!");
    clearInterval(handle);
  };
}, []);
```

Cleanup 메서드

6

■ componentWillUnmount 생명주기 메서드의 기능을 useEffect로 처리하려면? --> Cleanup 메서드

■ App.js 코드 변경

```
import React, { Component } from 'react';

//import Clock from './Clock';

import Clock from './Clock2';
```

3. useEffect(5)

useEffect의 장점

- 여러개의 useEffect()를 작성할 수 있음.
- 상태 데이터를 중심으로 관련된 작업을 배치할 수 있음
- App05.js

```
import React, { useEffect, useState } from 'react';

const App = () => {
  const [count, setCount] = useState(0);
  const [name, setName] = useState("hong");

  useEffect(() => {
    console.log(`You clicked ${count} times`);
  }, [count]);

  useEffect(() => {
    console.log(`Name is '${name}' `);
  }, [name]);
```

```
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
          Click me</button>
      <br />
      <input type="text" value={name}
          onChange={e => setName(e.target.value)} />
    </div>
  );
};

export default App;
```


7

4. 간단한 예제(1)

useState와 useEffect를 사용한 간단한 예제(App06.js)

- componentDidMount 시에
  - name 상태의 초기값을 이용해 axios로 원격 서버의 데이터를 가져와서 contacts 상태를 초기화한다.

```
import React, { useState, useEffect } from "react";
import axios from "axios";

const App = () => {
  const [name, setName] = useState('');
  const [contacts, setContacts] = useState([]);

  useEffect(() => {
    setName('ja');    //초기값 설정 예)현재시간, 게시판이라면 1페이지
  }, []);
```

(다음 페이지로 이어짐)

8

■ 이 예제를 작성하기 전에 다음과 같이 axios 의존 패키지를 설치합니다.

- yarn add axios
- axios 는 HTTP 통신 기능을 제공하는 라이브러리입니다.

4. 간단한 예제(2)

이전 페이지에 이어서

```
const searchContacts = () => {
    const url = `https://contactsvc.herokuapp.com/contacts/search/${name}`;
    axios.get(url).then(response => {
      if (response.data.status && response.data.status === 'fail') {
        alert(`조회 실패 : ${response.data.message}`);
      } else {
        setContacts(response.data);
      }
    });
  };

  return (
    <div  style={{ padding: '20px' }}>
      이름 : <input type="text" value={name} onChange={e => setName(e.target.value)}
        onKeyUp={e => { if (e.code === 'Enter') searchContacts() }} /><hr />
      <table className="list">
        <thead>
          <tr>th>이름</th><th>모바일</th><th>이메일</th></tr>
        </thead>
```

(다음 페이지로 이어짐)

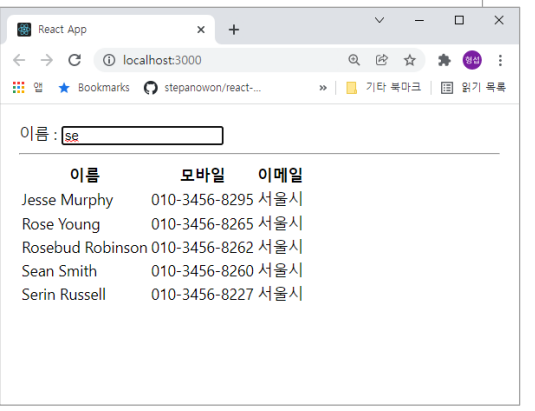
4. 간단한 예제(3)

이전 페이지에 이어서

```
      <tbody>
        {
          contacts.map(c => (
            <tr key={c.no}><td>{c.name}</td><td>{c.tel}</td><td>{c.address}</td></tr>
          ))
        }
      </tbody>
    </table>
  </div>
 );
}

export default App;
```

실행

- 영문 두글자를 입력하고 엔터!!



5. useReducer(1)

reducer와 순수함수

- 순수함수(pure function) : 다음의 조건을 만족해야 함.
  - 입력인자가 동일하면 리턴값도 동일해야 함
  - 부수효과(side effect)가 없어야 함
  - 함수에 전달된 인자는 불변성으로 여겨짐. 인자는 변경할 수 없음
- reducer란?
  - 배열(Array)의 메서드 중 reduce()에 인자로 전달하는 함수가 reducer임.
    - reduce() 메서드는 합계 값을 구할 때 사용할 수 있음
  - 두개의 인자를 이용해 연산한 값을 리턴하면 리턴값이 새로운 상태값이 됨.

```
const familyMembers = [
  { name:"홍길동", point: 10000, rel:"본인" },
  { name:"성춘향", point: 20000, rel:"처"  },
  { name:"홍예지", point: 15000, rel:"딸" },
  { name:"홍철수", point: 5000, rel:"아들" },
  { name:"홍희수", point: 10000, rel:"아들" },
];
```

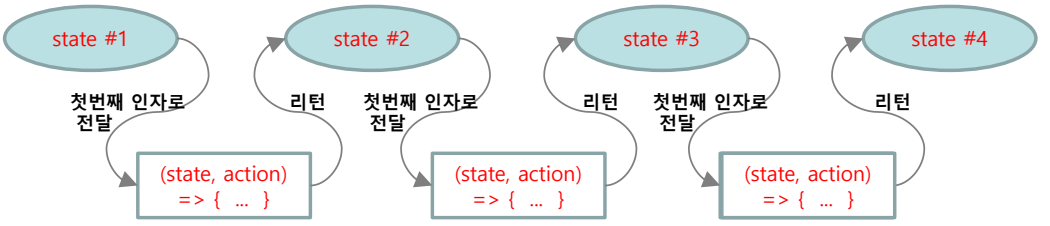
```
const initialPoint = 10000;
const reducer = (totalPoint, member) => {
   totalPoint += member.point;
   return totalPoint;
}

const totalPoint =
     familyMembers.reduce(reducer, initialPoint);
console.log(`가족 합계 포인트 : ${totalPoint}`)
```


11

5. useReducer(2)

reducer 함수 구조구조

- 인자 : (state, action) => { }
- 리턴값 : action(type, payload)와 기존 상태(state)를 이용해 새로운 상태를 만들어서 리턴함. --> 불변성(immutability)
- 불변성을 사용하는 이유?
  - 렌더링 최적화 : 5장에서 이미 학습한 내용
  - 상태 데이터에 대한 변경 추적과 효과적인 디버깅
    - 상태 변경 추적에 대한 자세한 내용은 Redux를 학습할 때 살펴봄



5. useReducer(3)

useReducer() Hook

- 함수 컴포넌트에서 reducer를 이용해 상태를 변경할 수 있도록 함
- 사용 방법
  - const [state, dispatch] = useReducer(reducer, initialState);
  - 입력값
    - initialState : 초기 상태 값
    - reducer : reducer 함수
  - 리턴값 : 배열
    - 첫번째 배열값 : 읽기 전용의 상태 값(state)
    - 두번째 배열값 : 상태 변경을 위해 액션(action: type+payload)을 인자로 전달하여 호출할 함수(dispatch)
  - 상태 변경은 반드시 dispatch 함수를 이용해야만 함
  - action의 형식 : type(액션 유형) + payload(액션을 수행할 때 필요한 데이터)
    - 예) { type:"addTodo", payload : { id: 1001, todo:"강아지 산책" } }
  - dispatch() 호출 형식
    - 예) dispatch({ type:"addTodo", payload : { id: 1001, todo:"강아지 산책" } })

5. useReducer(4)

useReducer를 이용한 todolist 기능 예제 (App07.js)

```
import React, { useReducer, useState } from "react";
import produce from "immer";

let ts = new Date().getTime();
const initialTodoList = [
  { id: ts + 1, todo: "운동" },
  { id: ts + 2, todo: "독서" },
  { id: ts + 3, todo: "음악감상" },
]

const reducer = (todoList, action) => {
  switch (action.type) {
    case "addTodo":
      return produce(todoList, (draft) => {
        draft.push({ id: new Date().getTime(),todo: action.payload.todo });
      });
    case "deleteTodo":
      let index = todoList.findIndex(item => item.id === action.payload.id);
      return produce(todoList, (draft) => {
        draft.splice(index, 1);
      });
    default:
      return todoList;
  }
};
```


14

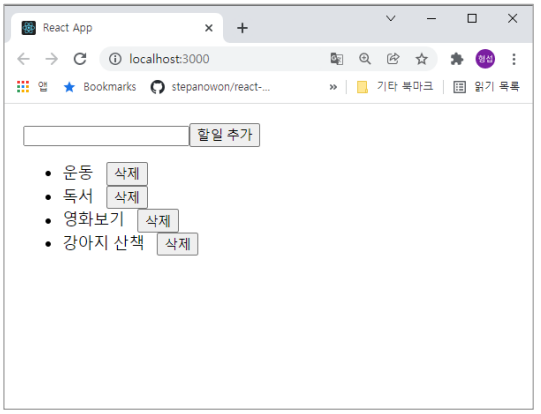
5. useReducer(5)

이전 페이지에 이어서(App07.js)

```
const App = () => {
  const [todoList, dispatchTodoList] = useReducer(reducer, initialTodoList);
  const [todo, setTodo] = useState("");
  const addTodo = () => {
    dispatchTodoList({ type: "addTodo", payload: { todo: todo } });
    setTodo("");
  }
  const deleteTodo = (id) => {
    dispatchTodoList({ type: "deleteTodo", payload: { id:id  }})
  }

  return (
    <div style={{ padding:'20px' }}>
      <input type="text" onChange={e => setTodo(e.target.value)} value={todo} />
      <button onClick={addTodo}>할일 추가</button>
      <ul>
        {todoList.map(item => (
          <li key={item.id}>{item.todo}   
            <button onClick={()=>deleteTodo(item.id)}>삭제</button>
          </li>
        ))}
      </ul>
    </div>
  );
};
export default App;
```

5. useReducer(6)

useReducer 예제 실행 결과



16

6. useRef(1)

useRef Hook의 용도

- Browser DOM에 대한 직접적인 접근을 허용
- 상태가 아닌 데이터 관리
  - useRef에 의해 생성된 데이터는 상태가 아니므로 변경해도 re-render가 일어나지 않음

Browser DOM 접근 예제

```
import React, { useRef } from "react";
const App = () => {
  const elName = useRef(null);
  const goFirstInputElement = () => {
    elName.current.focus();
  };
  return (
    <div>
      이름 : <input ref={elName} type="text" defaultValue="홍길동" /><br />
      전화 : <input type="text" defaultValue="010-2222-3333" /><br />
      주소 : <input type="text" defaultValue="서울" /><br />
      <button onClick={goFirstInputElement}>첫번째 필드로 포커스 이동</button>
    </div>
  );
};
export default App;
```

17

■ Browser DOM에 직접적으로 접근하기

- 다음과 같이 참조 객체를 생성함
  - const elName = useRef(null);
- 다음과 같이 Element에서 ref Attribute를 이용해 ref 객체를 참조함
  - `<input ref={elName} type="text" defaultValue="홍길동" />`
- Browser DOM에 접근할 때는 다음과 같이 참조 객체를 이용함
  - `elName.current.focus();`
  - elName.curent와 같이 항상 current 속성을 통해 Browser DOM 요소에 접근함

6. useRef(2)

상태가 아닌 데이터 관리(App09.js)

- 참조객체의 데이터가 변경되어도 re-rende하지 않음.
- 다른 상태가 변경되어 re-rende되었을 때 참조객체 값의 변경을 확인할 수 있음

```
import React, { useRef, useState } from 'react';

const App = () => {
  const [name, setName] = useState('홍길동');
  const refTel = useRef('010-2222-2222');
  return (
    <div>
      <h2>상태 데이터</h2>
      <input type='text' value={name} onChange={ (e) => setName(e.target.value) } /><br />
      <div> 상태(name) : {name}</div>
      <hr />
      <input type="text" onChange={ (e)=>refTel.current = e.target.value } /><br />
      <div> refTel 값 : {refTel.current}</div>
    </div>
  );
};

export default App;
```


18

7. useMemo(1)

함수의 실행 결과를 캐싱함.

- 기존 state, props 값에 대한 실행된 함수의 캐싱값을 저장
  - 렌더링될 때마다 함수가 실행되지 않도록 함.
- useMemo 적용 전 코드(App10-before.js)

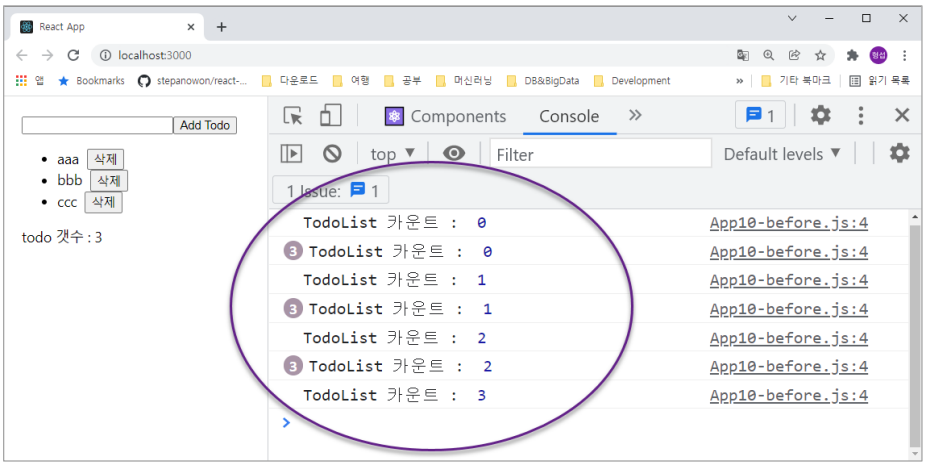
```
import React, { useState } from "react";

const getTodoListCount = todoList => {
  console.log("TodoList 카운트 : ", todoList.length);
  return todoList.length;
};

const App = () => {
  const [todoList, setTodoList] = useState([]);
  const [todo, setTodo] = useState("");
  ……

  return (
    <div>
      <input type="text" value={todo}
          onChange={e => setTodo(e.target.value)} />
      <button onClick={() => addTodo(todo)}>
           Add Todo</button>
      <br />
      <ul>
        {todoList.map(item => (
          <li key={item.id}>
            {item.todo}  
            <button onClick={() => deleteTodo(item.id)}>
                삭제</button>
          </li>
        ))}
      </ul>
      <div>todo 갯수 : { getTodoListCount(todolist) }</div>
    </div>
  );
};

export default App;
```


■ todolist는 변경된 것이 없어도 이 컴포넌트가 re-render가 일어나면 getTodoListCount() 함수는 매번 실행되므로 개선이 필요함.

7. useMemo(2)

App10-before.js 실행 결과

- TodoList 카운트가 변경되지 않았음에도 getTodoListCount() 함수를 매번 호출하고 있음(re-render할 때마다)



20

7. useMemo(3)

useMemo 적용 코드(App10.js)

- useMemo(func, array)
  - 두번째 인자 배열에 등록된 의존 객체(state, props)가 변경될 때 함수가 실행하여 값을 캐싱함

```
import React, { useState, useMemo } from "react";

const getTodoListCount = todoList => {
  console.log("TodoList 카운트 : ", todoList.length);
  return todoList.length;
};

const App = () => {
  ……
  const todolistCount =
     useMemo(() => getTodoListCount(todoList), [todoList]);

  return (
    <div>
       ……(생략)
       <div>todo 갯수 : {todolistCount}</div>
    </div>
  );
};
export default App;
```

21

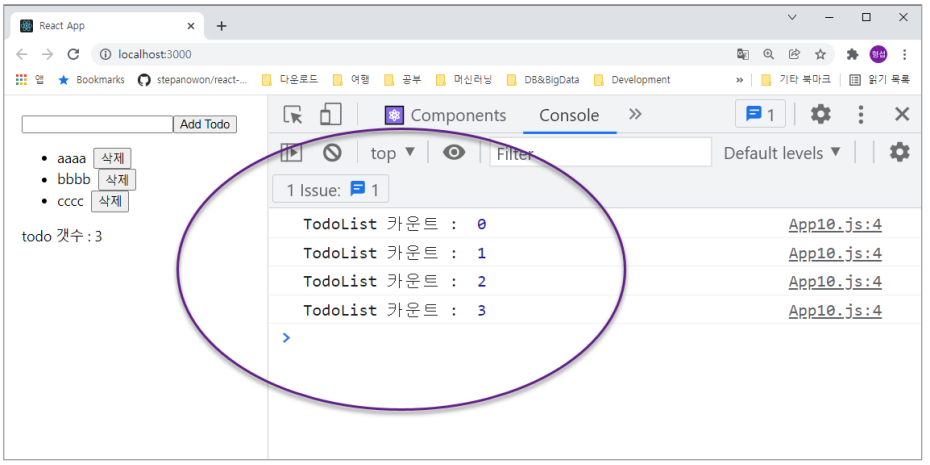
■ useMemo() 함수의 두번째 인자로 전달한 todoList 배열 객체가 바뀔 때만 useMemo()에 인자로 전달한 함수가 실행됨

- 함수 실행 후 리턴된 값이 todoListCount로 캐싱됨.

7. useMemo(4)

App10.js 실행 결과

- todoList 배열에 변화가 있을 때만 함수가 실행됨



22

8. useCallback(1)

함수의 중복 선언을 방지함

- 함수 컴포넌트 내부의 함수는 렌더링할 때마다 재생성됨.
  - 이것을 막기 위해 useCallback() 사용할 수 있음.
- useCallback 적용 전 코드(App11-before.js)

```
import React, { useState, useMemo } from "react";

const App = () => {
  const [todoList, setTodoList] = useState([]);
  const [todo, setTodo] = useState("");

  const addTodo = (todo) => {
    let newTodoList = [...todoList, { id: new Date().getTime(), todo: todo }];
    setTodoList(newTodoList);
  };

  const deleteTodo = (id) => {
    let index = todoList.findIndex(item => item.id === id);
    let newTodoList = [...todoList];
    newTodoList.splice(index, 1);
    setTodoList(newTodoList);
  };
  ……(생략)
}
```

8. useCallback(2)

useCallback을 적용한 코드(App11.js)

- useCallback의 두번째 인자는 함수 실행시에 의존하는 참조형 값을 기재한다.

```
import React, { useState, useMemo, useCallback } from "react";
……
const App = () => {
  const [todoList, setTodoList] = useState([]);
  const [todo, setTodo] = useState("");

  const addTodo = useCallback( (todo) => {
      let newTodoList = [...todoList, { id: new Date().getTime(), todo: todo }];
      setTodoList(newTodoList);
      setTodo("");
  },[todoList] );

  const deleteTodo = useCallback( (id) => {
      let index = todoList.findIndex(item => item.id === id);
      let newTodoList = [...todoList];
      newTodoList.splice(index, 1);
      setTodoList(newTodoList);
  }, [todoList] );
  ……
};

export default App;
```

24

■ useCallback에 두번째 인자로 전달하는 의존 객체를 반드시 지정해야 함

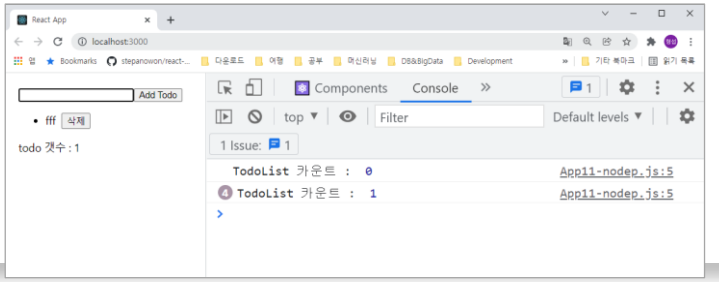
- 의존 객체에 변화가 생길 때만 함수를 생성함

8. useCallback(3)

의존 객체를 지정하지 않으면?(App11-nodep.js)

```
const addTodo = useCallback(
  (todo) => {
    let newTodoList = [...todoList, { id: new Date().getTime(), todo: todo }];
    setTodoList(newTodoList);
    setTodo("");
  },
  []
);
```

- 항상 초기 상태값을 참조함.
  - 여러 할일을 추가해도 항상 마지막에 추가한 할일만 남음
  - 초기 상태값([]) + 마지막 추가한 할일



25

8. useCallback(4)

useMemo, useCallback을 반드시 써야 하는가?

- 모든 컴포넌트에 사용할 필요는 없다.
- 성능 최적화가 필요하다고 판단되는 컴포넌트만.....
  - useMemo나 useCallback 은 캐싱으로 인해 추가로 메모리를 소비함.

9. Custom Hook(1)

기존의 Hook들을 조합해 새로운 사용자 정의 훅을 생성

- 예제 : 마우스 위치를 리턴하는 Custom Hook

Hook 적용 전 예제(App12.js)

```
import React, { useState, useEffect } from "react";

const App = () => {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    const onMove = (e) => setPosition({ x: e.pageX, y: e.pageY });
    window.addEventListener("mousemove", onMove);
    return () => {
      window.removeEventListener("mousemove", onMove);
    };
  }, []);

  return (
    <h2>{position.x}:{position.y}</div>
  );
};
export default App;
```

27

■ 관심사의 분리가 주목적

- 컴포넌트를 작성할 때 마우스 위치는 신경쓰고 싶지 않음
  - 마우스 위치를 알아내는 기능을 훅으로 작성하여 재사용하도록 함.

■ 이 예제는 다음 문서를 기반으로 작성하였음

- https://codedaily.io/tutorials/Create-a-useMousePosition-Hook-with-useEffect-and-useState-in-React

9. Custom Hook(2)

Hook 적용

- src/hooks/useMousePosition.js

```
import { useEffect, useState } from "react";

const useMousePosition = () => {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    const onMove = (e) => setPosition({ x: e.pageX, y: e.pageY });
    window.addEventListener("mousemove", onMove);

    return () => {
      window.removeEventListener("mousemove", onMove);
    };
  }, []);

  return position;
};

export { useMousePosition };
```

9. Custom Hook(3)

Hook 적용(이어서)

- App13.js

```
import React from "react";
import { useMousePosition } from "./hooks/useMousePosition";

//Custom Hook 적용후
const App = () => {
  const position = useMousePosition();

  return (
    <h2>
      커스텀 훅 적용 [ {position.x}, {position.y} ]
    </h2>
  );
};

export default App;
```