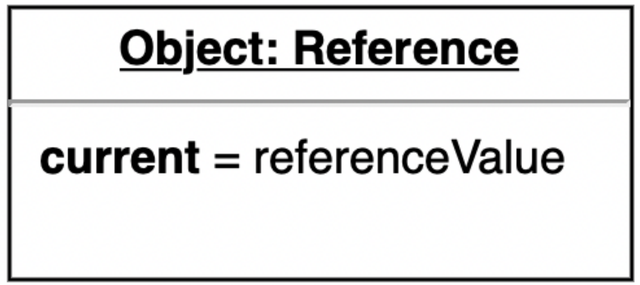
Hook

useRef, useReducer

Hook
useRef



- (useState) State 의변화 -> 렌더링 -> 컴포넌트 내부 변수 초기화 (원하지 않는 렌더링)
  - 컴포넌트의 렌더링 간의 값 유지가 필요한 경우 사용
- (useRef) current 속성을 가지고 있는 객체를 반환, 인자로 넘어온 초기값을 current 속성에 할당
  - 다시 렌더링 되어도 동일한 참조값을 유지
- 차이점
  - State의 변화 -> 렌더링 -> Ref 값은 유지됨
  - Ref 의 변화 -> No 렌더링 -> 변수들의 값이 유지됨

Hook

useRef

```
import { useState, useEffect, useRef } from "react";
import ReactDOM from “react-dom/client";

function App() {
  const [inputValue, setInputValue] = useState("");
  const count = useRef(0);
  useEffect(() => {
    count.current = count.current + 1;
  });
  return (
    <>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h1>Render Count: {count.current}</h1>
    </>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

State의 변화 ➡ 렌더링 ➡ 컴포넌트 내부 변수들 초기화

Ref의 변화 ➡ No 렌더링 ➡ 변수들의 값이 유지됨

State의 변화 ➡ 렌더링 ➡ 그래도 Ref의 값은 유지됨

Hook

Lab. Stopwatch

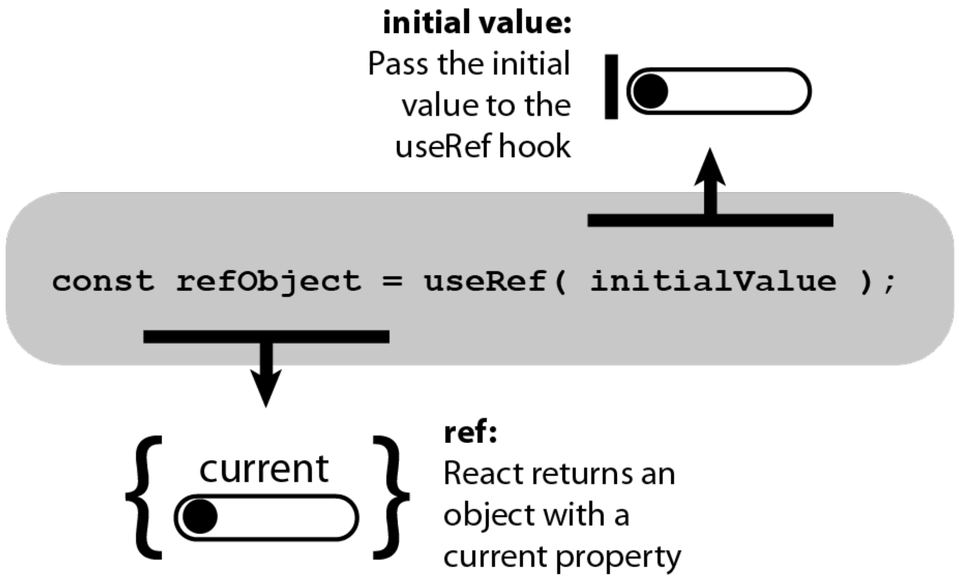
```
import {                                } from 'react';

export default function Stopwatch() {
  const timerIdRef = useRef(0);
  const [count, setCount] = useState(0);
  const startHandler = () => {
                                                                  
  };
  const stopHandler = () => {
    clearInterval(timerIdRef.current);
    timerIdRef.current = 0;
  };
  useEffect(() => {
    return () => clearInterval(timerIdRef.current);
  }, []);
 }
```

```
return (
    <div>
      <div>Timer: {count}s</div>
      <div>
        <button onClick={startHandler}>Start</button>
        <button onClick={stopHandler}>Stop</button>
      </div>
    </div>
  );
```

Hook
useRef

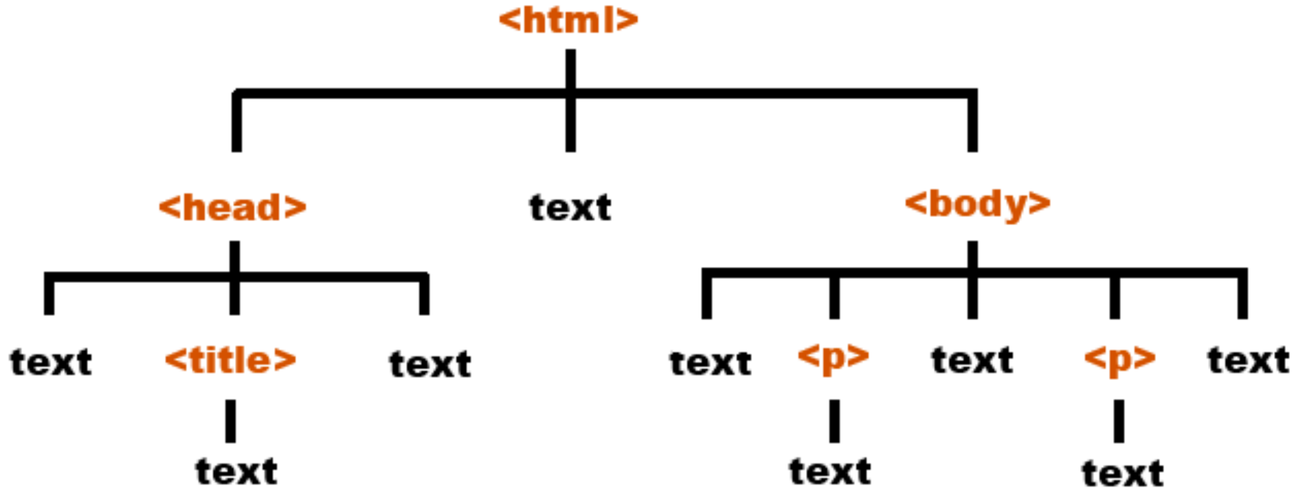
- 원하는 특정 DOM을 직접 선택해서 컨트롤 하기 위해 사용




```jsx
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

Hook

useRef



- 객체 생성하기

```
const nameInput = useRef();
```

- DOM API 사용

```
nameInput.current.focus();
```

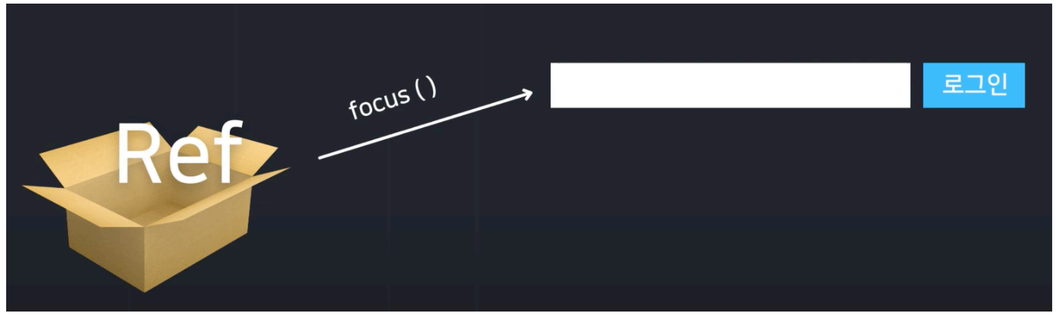
- DOM 설정을 통해 DOM에 직접 접근하기

```
<input
  name="name"
  placeholder="이름"
  onChange={onChange}
  value={name}
  ref={nameInput}
/>
```

Hook

useRef

- (로그인) 편리한 input 요소 접근 제공




```
const ref = useRef (value)
```

```
<input ref= {ref} / >
```

- 자동으로 focus 를 지정해줌

Hook

useRef

```
import { useRef, useEffect } from 'react';

export default function InputFocus() {
  const inputRef = useRef();
  useEffect(() => {
      inputRef.current.focus();
  }, []);
  return (
    <input
      ref={inputRef}
      type="text"
    />
  );
}
```

Hook

useReducer

- useState를 대체할 수 있는 함수
- 더 복잡한 상태 관리가 필요한 React 컴포넌트에서는 setReducer() 함수를 사용

```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```

`useState`의 대체 함수입니다. `(state, action) => newState`의 형태로 reducer를 받고 `dispatch` 메서드와 짝의 형태로 현재 state를 반환합니다. (Redux에 익숙하다면 이것이 어떻게 동작하는지 여러분은 이미 알고 있을 것입니다.)

Hook

useReducer

- 함수 reducer 는 현재 state, action 객체를 인자로 받아, 기존의 state를 대체하여 새로운 state를 반환하는 함수

import React, { useReducer } from "react";

const [state, dispatch] = useReducer(reducer, initialState, init);

- 콜백대신 dispatch를 통해 reducer 함수를 실행

<button onClick={() => dispatch({ type: "INCREMENT" })}>증가</button>

Hook

useReducer

- dispatch 함수에 의해 실행되며, 컴포넌트 외부에서 state를 업데이트 하는 로직을 담당
  - 함수의 인자로는 state와 action을 받아 새로운 state를 반환함

```
function reducer(state, action) {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      throw new Error("unsupported action type: ", action.type);
  }
}
```

Hook

useReducer

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    case 'reset':
      return init(action.payload);
    default:
      throw new Error();
  }
}

function Counter({initialCount}) {
  const [state, dispatch] = useReducer(reducer, initialCount, init);
  return (
    <>
      Count: {state.count}
      <button
        onClick={() => dispatch({type: 'reset', payload: initialCount})}>
        Reset
      </button>
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

Hook