useRef 훅

**학습목표**

useRef 훅

스크롤박스 자동 스크롤하기

**목차**

[1) useRef 훅 2](#_Toc150179337)

[2) 용도 #1 2](#_Toc150179338)

[3) 용도 #2 4](#_Toc150179339)

[4) 오류 7](#_Toc150179340)

## useRef 훅

useRef 훅의 용도는 두 개 이다.

용도 #1: Physical DOM 객체의 속성이나 메소드를 사용하기. (웹브라우저의 DOM 객체)

용도 #2: 상태값이 변경될 경우에 즉시 렌더링 될 필요는 없는 상태값 만들기.

위 두 용도에 대한 예제를 구현해 보자.

## 용도 #1

Physical DOM 객체의 속성이나 메소드를 사용하기. (웹브라우저의 DOM 객체)

useRef 훅을 이용해서 Pysical DOM 객체에 대한 참조 변수를 만듦어서,

그 DOM 객체의 메소드, 속성 등을 사용함.

### ScrollBox.tsx

|  |
| --- |
| import React, { useEffect, useRef } from 'react'  import styled from 'styled-components';  const Div = styled.div<{w: string, h: string}>`    border: 1px solid #ccc;    padding: 5px;    overflow-y: scroll;    width: ${(props) => props.w};    height: ${props => props.h};  `;  type Props = {    width: string,    height: string,    children: JSX.Element | JSX.Element[]  }  function ScrollBox({width, height, children}: Props) {    const divRef = useRef<HTMLDivElement>(null)    useEffect(() => divRef.current?.scrollIntoView());    return (      <Div w={width} h={height}>        {children}        <div ref={divRef}></div>      </Div>    )  }  export default ScrollBox |

const Div = styled.div<{w: string, h: string}>`

  border: 1px solid #ccc;

  padding: 5px;

  overflow-y: scroll;

  width: ${props => props.w};

  height: ${props => props.h};

`;

Div 컴포넌트의 props는 w: string, h: string 이다.

이 props 값들이 각각 ${props => props.w} 부분과 ${props => props.h} 부분에 출력된다.

<Div w={width} h={height}>

Div 컴포넌트의 w, h props 값 전달하기

const divRef = useRef<HTMLDivElement>(null)

useEffect(() => divRef.current?.scrollIntoView());

divRef 변수는 Pysical DOM 객체(웹브라우저의 DOM 객체)를 참조하기 위한 변수이다.

scrollIntoView() 메소드는 Pysical DOM 객체의 메소드이다.

이 메소드를 호출하면 그 태그가 화면에 보이도록 자동으로 스크롤된다.

useEffect 훅의 호출형태 #1 이므로,

() => divRef.current?.scrollIntoView() 이 콜백 함수는 화면에 렌더링 될 때 마다 호출된다.

정확히 말하면, 화면이 렌더링 된 직후에 호출된다.

<div ref={divRef}></div>

이 div 태그에 해당하는 Pysical DOM 객체를 divRef 변수가 참조하게 된다.

따라서 화면이 렌더링 된 직후에, 이 div 태그가 화면에 보이도록 자종으로 스크롤된다.

이 div 태그는 {children} 목록의 끝에 위치하기 때문에,

화면이 렌더링 될 때 마다 스크롤바는 목록의 끝으로 자동으로 스크롤된다.

### App.tsx

|  |
| --- |
| import React, { useState } from 'react';  import ScrollBox from './ScrollBox';  function App() {    const [list, setList] = useState<string[]>([]);    const addTime = () => {      const time = new Date().toLocaleTimeString();      setList([...list, time]);    }    return (      <div style={{padding: 20}}>        <button onClick={addTime}>Add Time</button>        <ScrollBox width="200px" height="500px">          {list.map(s => <p>{s}</p>)}        </ScrollBox>      </div>    );  }  export default App; |

## 용도 #2

상태값이 변경될 경우에 즉시 렌더링 될 필요는 없는 상태값 만들기.

useRef 훅을 이용해서, 그 상태값을 저장하기 위한 상태 변수를 만든다.

### App.tsx

|  |
| --- |
| import React, { useRef, useState } from 'react';  import styled from 'styled-components';  const Div = styled.div`    padding: 20px;    & select { padding: 5px; width: 100px; }    & button { padding: 5px 20px; margin-left: 10px; }  `  type Item = {    time: string,    color: string  }  function App() {    const colorRef = useRef<string>("#ffa");    const [list, setList] = useState<Item[]>([]);    const addTime = () => {      const time = new Date().toLocaleTimeString();      setList([...list, {time, color: colorRef.current}]);    }    return (      <Div>        <select onChange={e => colorRef.current = e.target.value}>          <option value="#ffa">yellow</option>          <option value="#dfd">green</option>          <option value="#ccf">blue</option>          <option value="#fdd">red</option>        </select>        <button onClick={addTime}>Add Time</button>        <div>          {list.map(item =>                      <p style={{backgroundColor: item.color}} key={item.time}>                        {item.time}                      </p>)}        </div>      </Div>    );  }  export default App; |

const colorRef = useRef<string>("#ffa");

colorRef 상태 변수를 만든다. 이 변수의 최초 값은 "#ffa"

이 변수의 값은 앞으로 추가될 항목의 배경 색이다.

따라서 이 변수의 값이 변경되어도 화면이 다시 렌더링 될 필요가 없다.

type Item = {

  time: string,

  color: string

}

데이터 항목은, 현재 시각 문자열과 배경색 문자열 객체이다.

const [list, setList] = useState<Item[]>([]);

데이터 목록 상태 변수

const addTime = () => {

    const time = new Date().toLocaleTimeString();

    setList([...list, {time, color: colorRef.current}]);

}

add 버튼이 클릭되면, 이 메소드가 호출된다.

현재 시각 문자열과, colorRef 상태 변수의 현재 값으로 데이터 항목 객체를 만들어서

데이터 목록에 추가한다. 이때 화면이 다시 렌더링 된다.

<select onChange={e => colorRef.current = e.target.value}>

select 태그에서 선택된 항목이 변경되면, 그 항목의 value 값을 colorRef 상태 변수에 대입된다.

이때는 화면이 다시 렌더링 되지 않는다.

### App.tsx 비효율적인 구현

|  |
| --- |
| import React, { useRef, useState } from 'react';  import styled from 'styled-components';  const Div = styled.div`    padding: 20px;    & select { padding: 5px; width: 100px; }    & button { padding: 5px 20px; margin-left: 10px; }  `  type Item = {    time: string,    color: string  }  function App() {    const [color, setColor] = useState<string>("#ffa");    const [list, setList] = useState<Item[]>([]);    const addTime = () => {      const time = new Date().toLocaleTimeString();      setList([...list, {time, color: color}]);    }    return (      <Div>        <select onChange={e => setColor(e.target.value)}>          <option value="#ffa">yellow</option>          <option value="#dfd">green</option>          <option value="#ccf">blue</option>          <option value="#fdd">red</option>        </select>        <button onClick={addTime}>Add Time</button>        <div>          {list.map(item =>                      <p style={{backgroundColor: item.color}} key={item.time}>                        {item.time}                      </p>)}        </div>      </Div>    );  }  export default App; |

useState 훅으로. color 상태를 구현하면,

select 태그의 값이 변경될 때마다 다시 렌더링되는 비효율이 있다.

## 오류

### 상태를 지역 변수로 구현하기

|  |
| --- |
| import React, { useRef, useState } from 'react';  import styled from 'styled-components';  const Div = styled.div`  padding: 20px;  & select { padding: 5px; width: 100px; }  & button { padding: 5px 20px; margin-left: 10px; }  `  type Item = {  time: string,  color: string  }  function App() {  let color = "#ffa";  const [list, setList] = useState<Item[]>([]);  const addTime = () => {  const time = new Date().toLocaleTimeString();  setList([...list, {time, color}]);  }  return (  <Div>  <select onChange={e => color = e.target.value}>  <option value="#ffa">yellow</option>  <option value="#dfd">green</option>  <option value="#ccf">blue</option>  <option value="#fdd">red</option>  </select>  <button onClick={addTime}>Add Time</button>  <div>  {list.map(item =>  <p style={{backgroundColor: item.color}} key={item.time}>  {item.time}  </p>)}  </div>  </Div>  );  }  export default App; |

컴포넌트의 상태를 지역 변수로 구현할 수 없는 이유는 다음과 같다.

App 컴포넌트가 렌더링 될 때 마다, App() 함수가 호출된다.

App 함수가 호출 될 때 마다, color 지역 변수가 다시 생성되고 초기화 된다.

즉 color 지역 변수의 값이 렌더링될 때 마다 초기화 된다.

### 상태를 전역 변수로 구현하기

|  |
| --- |
| import React, { useRef, useState } from 'react';  import styled from 'styled-components';  const Div = styled.div`  padding: 20px;  & select { padding: 5px; width: 100px; }  & button { padding: 5px 20px; margin-left: 10px; }  `  type Item = {  time: string,  color: string  }  let color = "#ffa";  function App() {  const [list, setList] = useState<Item[]>([]);  const addTime = () => {  const time = new Date().toLocaleTimeString();  setList([...list, {time, color}]);  }  return (  <Div>  <select onChange={e => color = e.target.value}>  <option value="#ffa">yellow</option>  <option value="#dfd">green</option>  <option value="#ccf">blue</option>  <option value="#fdd">red</option>  </select>  <button onClick={addTime}>Add Time</button>  <div>  {list.map(item =>  <p style={{backgroundColor: item.color}} key={item.time}>  {item.time}  </p>)}  </div>  </Div>  );  }  export default App; |

예를 들어 TodoItem 컴포넌트는 todo 항목의 갯수 만큼 컴포넌트 객체가 생성되어야 한다.

TodoItem 컴포넌트의 상태를 전역 변수로 구현하면,

생성된 TodoItem 컴포넌트 객체들이 그 전역 변수 한 개를 공유하게 된다.

### 전역변수 부작용 실행 예제

#### MyList.tsx

|  |
| --- |
| import React, { useRef, useState } from 'react';  import styled from 'styled-components';  const Div = styled.div`  padding: 20px;  & select { padding: 5px; width: 100px; }  & button { padding: 5px 20px; margin-left: 10px; }  `  type Item = {  time: string,  color: string  }  let color = "#ffa";  function MyList() {  const [list, setList] = useState<Item[]>([]);  const addTime = () => {  const time = new Date().toLocaleTimeString();  setList([...list, {time, color}]);  }  return (  <Div>  <select onChange={e => color = e.target.value}>  <option value="#ffa">yellow</option>  <option value="#dfd">green</option>  <option value="#ccf">blue</option>  <option value="#fdd">red</option>  </select>  <button onClick={addTime}>Add Time</button>  <div>  {list.map(item =>  <p style={{backgroundColor: item.color}} key={item.time}>  {item.time}  </p>)}  </div>  </Div>  );  }  export default MyList; |

#### App.tsx

|  |
| --- |
| import React from 'react';  import MyList from './MyList'  function App() {    return (      <div>        <MyList />        <MyList />      </div>    );  }  export default App; |