styled components와 Context API

**학습목표**

children props

styled components

useContext

**목차**

[1. children props 2](#_Toc133848883)

[1) 개요 2](#_Toc133848884)

[2) 예제 2](#_Toc133848885)

[2. styled-components 4](#_Toc133848886)

[1) 개요 4](#_Toc133848887)

[2) 예제 #1 4](#_Toc133848888)

[3) 예제 #2 6](#_Toc133848889)

[4) 예제 #3 7](#_Toc133848890)

[3. Context API 9](#_Toc133848891)

[1) 개요 9](#_Toc133848892)

[2) 예제 #1 10](#_Toc133848893)

[3) 렌더링 최적화와 Context API 14](#_Toc133848894)

# children props

## 개요

|  |
| --- |
| <ScrollBox width="50px" height="200px">     <p>one</p>     <p>two</p>     <p>three</p>     <p>four </p>  </ScrollBox> |

ScrollBox 컴포넌트에 전달되는 props는 3 개이다.

width: "50px" 문자열

height: "200px" 문자열

children: [<p>one</p>, <p>two</p>, <p>three</p>, <p>four </p>] 배열

## 예제

### ScrollBox.tsx

|  |
| --- |
| import React, { CSSProperties } from 'react'  type Props = {    width: string,    height: string,    children: JSX.Element | JSX.Element[]  }  const style: CSSProperties = {    border: '1px solid #ccc',    padding: '5px',    overflowY: 'scroll'  }  function ScrollBox({width, height, children}: Props) {    return (      <div style={{...style, width, height}}>        {children}      </div>    )  }  export default ScrollBox |

JSX.Element 타입은 Virtual DOM 태그 객체의 타입이다.

children: JSX.Element | JSX.Element[]

chilren props의 타입은 JSX.Element 한 개이거나, JSX.Element 배열 이다.

const style: CSSProperties = {

  border: '1px solid #ccc',

  padding: '5px',

  overflowY: 'scroll'

}

JSX.Element 태그의 style 속성의 타입이 CSSProperties 이다.

이 속성값은 Javascript 객체이고, 이 속성값으로 인라인 서식을 지정한다.

### App.tsx

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

### App.css

|  |
| --- |
| div.App { padding: 50px; }  button { padding: 0.5em 1em; } |

# styled-components

## 개요

별도의 .css 파일에 서식을 구현하는 것이 약간 번거롭다.

styled-components 라이브러리를 사용하면, tsx 파일에 CSS 서식을 구현할 수 있다.

### 라이브러리 설치

|  |
| --- |
| npm install styled-components @types/styled-components |

프로젝트 디렉토리에서 위 명령을 실행하자.

## 예제 #1

### App.tsx

|  |
| --- |
| import React, { useState } from 'react';  import styled from 'styled-components';  function App() {    const [color, setColor] = useState("#ffa");    const Div = styled.div`      border: 1px solid gray;      box-shadow: 5px 5px 5px gray;      padding: 40px;      margin: 20px;      display: inline-block;      background-color: ${color};    `;    const H1 = styled.h1`      margin: 10px;    `;    const Select = styled.select`      padding: 4px;      width: 150px;      font-size: 12pt;    `;      return (      <Div>        <H1>Color</H1>        <Select onChange={e=>setColor(e.target.value)} value={color}>          <option value="#ffa">yellow</option>          <option value="#dfd">green</option>          <option value="#ccf">blue</option>          <option value="#fdd">red</option>        </Select>      </Div>    );  }  export default App; |

import styled from 'styled-components';

라이브러리 import

const Div = styled.div**`**

    border: 1px solid gray;

    box-shadow: 5px 5px 5px gray;

    padding: 40px;

    margin: 20px;

    display: inline-block;

    background-color: ${color};

**`**;

분홍색으로 칠한 부분이 역따옴표 문자임에 주의하자. (탭 키 바로위)

위 코드에 의해, div 태그에 CSS 서식이 적용되어 Div 태그 컴포넌트가 생성된다.

${ expression } 부분은 expression의 값으로 교체된다.

expression = 자바스크립트 표현식

즉 color 변수의 값이 '#ffe' 이므로, background-color: '#ffe'; 가 된다.

## 예제 #2

App.tsx

|  |
| --- |
| import React, { useState } from 'react';  import styled from 'styled-components';  function App() {    const [color, setColor] = useState("#ffa");    const Div = styled.div`      border: 1px solid gray;      box-shadow: 5px 5px 5px gray;      padding: 40px;      margin: 20px;      display: inline-block;      background-color: ${color};      & h1 { margin: 10px; }      & select { padding: 4px;          width: 150px;          font-size: 12pt;        }    `;      return (      <Div>        <h1>Color</h1>        <select onChange={e=>setColor(e.target.value)} value={color}>          <option value="#ffa">yellow</option>          <option value="#dfd">green</option>          <option value="#ccf">blue</option>          <option value="#fdd">red</option>        </select>      </Div>    );  }  export default App; |

  const Div = styled.div`

    border: 1px solid gray;

    box-shadow: 5px 5px 5px gray;

    padding: 40px;

    margin: 20px;

    display: inline-block;

    background-color: ${color};

   & h1 { margin: 10px; }

    & select { padding: 4px;

        width: 150px;

        font-size: 12pt;

      }

  `;

Div 내부에 있는 & h1 태그, & select 태그에 서식을 적용한다.

const Div 변수를 App 함수 지역 변수가 아니고, 전역 변수로 구현할 수도 있지만,

${color} 부분은 App 함수 밖으로 나갈 수 없다. color는 App 함수의 지역 변수이기 때문이다.

## 예제 #3

App.tsx

|  |
| --- |
| import React, { useState } from 'react';  import styled from 'styled-components';  const Div = styled.div`    border: 1px solid gray;    box-shadow: 5px 5px 5px gray;    padding: 40px;    margin: 20px;    display: inline-block;    background-color: ${props => props.color};    & h1 { margin: 10px; }    & select { padding: 4px;      width: 150px;      font-size: 12pt;    }  `;  function App() {    const [color, setColor] = useState("#ffa");    return (      <Div color={color}>        <h1>Color</h1>        <select onChange={e=>setColor(e.target.value)} value={color}>          <option value="#ffa">yellow</option>          <option value="#dfd">green</option>          <option value="#ccf">blue</option>          <option value="#fdd">red</option>        </select>      </Div>    );  }  export default App; |

위와 같이 color 값을, styled component의 props 값으로 전달할 수 있다.

소스코드에서 서식 부분과 태그 부분을 위와 같이 분리되어 있으면 코드를 읽기 편하다.

### App.tsx 수정 - props 타입 선언

|  |
| --- |
| import React, { useState } from 'react';  import styled from 'styled-components';  const Div = styled.div<{color: string}>`    border: 1px solid gray;    box-shadow: 5px 5px 5px gray;    padding: 40px;    margin: 20px;    display: inline-block;    background-color: ${props => props.color};    & h1 { margin: 10px; }    & select { padding: 4px;      width: 150px;      font-size: 12pt;    }  `;  function App() {    const [color, setColor] = useState("#ffa");    return (      <Div color={color}>        <h1>Color</h1>        <select onChange={e=>setColor(e.target.value)} value={color}>          <option value="#ffa">yellow</option>          <option value="#dfd">green</option>          <option value="#ccf">blue</option>          <option value="#fdd">red</option>        </select>      </Div>    );  }  export default App; |

Div styled component의 color props의 이름과 타입을 다음과 같이 선언할 수 있다.

const Div = styled.div<{color: string}>`

이 Div 컴포넌트의 props 타입은 {color: string} 이다.

# Context API

## 개요

예를 들어 학사정보시스템을 생각해 보자.

학사 정보시스템의 대부분의 페이지에서, 현재 사용자가 누구인지,

이 페이지를 사용할 권한이 있는지 검사해야 한다.

현재 사용자 정보를 모든 페이지에 props 값으로 전달하는 것은 매우 불편하다.

마치 전역 변수처럼, 학사 정보 시스템의 모든 페이지에서 현재 사용자 정부에 접근할 수 있으면 편리하다.

이런 기능을 제공해주는 것이 Context API 이다.

Context API를 이용하면, 마치 전역 변수처럼 아무 곳에서나 쉽게 사용할 수 있는, 전역 상태를 만들 수 있다.

### UserContext

현재 사용자 정보를 담은 UserContext 객체를 만들자.

현재 사용자 정보를 전역에 제공하기 위한,

UserProvider 컴포넌트도 구현해야 한다.

UserContext: 전역 상태

UserProvider 컴포넌트: 전역 상태를 전역에 제공해주는 컴포넌트

## 예제 #1

### UserContext.tsx

|  |
| --- |
| import React, { createContext, useState } from 'react'  export type User = {    name: string,    role: string  }  export type UserContextValue = {    state: { user: User },    actions: {      setUser: (user: User) => void    }  }  const UserContext = createContext<UserContextValue|null>(null)  type Props = {    children: JSX.Element | JSX.Element[]  }  function UserProvider({ children }: Props) {    const [user, setUser] = useState<User>({name: 'anonymous', role: 'guest'})    const value: UserContextValue = {      state: { user },      actions: { setUser }    };    return (      <UserContext.Provider value={value}>        {children}      </UserContext.Provider>    );  }  export { UserProvider }  export default UserContext; |

### UserContext 객체

export type User = {

  name: string,

  role: string

}

현재 사용자 정보 타입 선언

export type UserContextValue = {

  state: { user: User },

  actions: {

    setUser: (user: User) => void

  }

}

전역 상태 타입 선언

state 속성: 전역 상태 값

actions 속성: 전역 상태에 대한 액션 함수들

const UserContext = createContext<UserContextValue|null>(null)

전역 상태 객체를 생성한다.

전역 상태 객체의 타입은 UserContextValue 이거나 null 이다.

### UserProvider 컴포넌트

// UserProvider 컴포넌트의 props 타입 선언

type Props = {

  children: JSX.Element | JSX.Element[]

}

function UserProvider({ children }: Props) {

// 현재 사용자 정보를 관리하기 위한 상태 생성

  const [user, setUser] = useState<User>({name: 'anonymous', role: 'guest'})

// UserContext 전역 상태 값

  const value: UserContextValue = {

    state: { user },

    actions: { setUser }

  };

  return (

    <UserContext.Provider value={value}>

      {children}

    </UserContext.Provider>

  );

}

### index.tsx

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import './index.css';  import App from './App';  import reportWebVitals from './reportWebVitals';  import { UserProvider } from './UserContext';  const root = ReactDOM.createRoot(    document.getElementById('root') as HTMLElement  );  root.render(    <React.StrictMode>      <UserProvider>        <App />      </UserProvider>    </React.StrictMode>  );  // If you want to start measuring performance in your app, pass a function  // to log results (for example: reportWebVitals(console.log))  // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals  reportWebVitals(); |

App 컴포넌트와 그 자식 컴포넌트들에게, 현재 사용자 정보 전역 상태가 제공된다.

### CounterA.tsx

|  |
| --- |
| import React, { useContext, useState } from 'react'  import UserContext from './UserContext'  function CounterA() {    console.log("CounterA 호출됨");    const context = useContext(UserContext)    const [count, setCount] = useState<number>(0);    return (      <div>        <h1>CounterA</h1>        { context?.state.user.name }        <button onClick={() => setCount(count + 1)}>          { count }        </button>      </div>    )  }  export default React.memo(CounterA); |

현재 사용자 정보 전역 상태 값 사용

context?.state.user.name

context 값이 null 일 수도 있기 때문에, ?. 연산자를 사용해야 한다.

만약 context 값이 null 이면, ?. 연산자를 사용한 표현식 전체 값도 null 이 된다.

### CounterB.tsx

|  |
| --- |
| import React, { useState } from 'react'  function CounterB() {    console.log("CounterB 호출됨");    const [count, setCount] = useState<number>(0);    return (      <div>        <h1>CounterB</h1>        <button onClick={() => setCount(count + 1)}>          { count }        </button>      </div>    )  }  export default React.memo(CounterB); |

### GroupComponent.tsx

|  |
| --- |
| import React, { useContext, useState } from 'react'  import UserContext from './UserContext'  import CounterA from './CounterA'  import CounterB from './CounterB'  function GroupComponent() {    console.log('GroupComponent 호출됨');    const context = useContext(UserContext)    const [count, setCount] = useState<number>(0);    return (      <div>        <h1>GroupComponent</h1>        { context?.state.user.name }        <button onClick={() => setCount(count + 1)}>          { count }        </button>        <CounterA />        <CounterB />      </div>    )  }  export default React.memo(GroupComponent); |

### Login.tsx

|  |
| --- |
| import React, { useContext, useState } from 'react'  import UserContext from './UserContext'  function Login() {    console.log('Login 호출됨');    const [name, setName] = useState('')    const [role, setRole] = useState('')    const context = useContext(UserContext)    return (      <div>        <input type='text' placeholder='name' onChange={e=>setName(e.target.value)} />        <input type='text' placeholder='role' onChange={e=>setRole(e.target.value)} />        <button onClick={()=> context?.actions.setUser({name, role})}>로그인</button>      </div>    )  }  export default React.memo(Login) |

현재 사용자를 변경하는 기능 구현

### App.tsx

|  |
| --- |
| import React, { useState } from 'react'  import styled from 'styled-components';  import GroupComponent from './GroupComponent';  import Login from './Login';  const Div = styled.div`    padding: 30px;    & div { display: inline-block; margin: 10px; padding: 10px; border: 1px solid gray; }    & h1 { font-size: 11pt; margin: 0; }    & button { display: block; padding: 0.2em 2em; margin: 0.5em; }    & input { display: block; padding: 5px; margin: 5px 0; }  `;  function App() {    console.log('App 호출됨');    const [value, setValue] = useState<number>(0);    return (      <Div>        <h1>App</h1>        <button onClick={() => setValue(value + 1)}>          { value }        </button>        <GroupComponent />        <Login />      </Div>    );  }  export default React.memo(App); |

## 렌더링 최적화와 Context API

위 예제 코드에는 React.memo를 이용한 렌더링 최적화도 구현되어 있다.

현재 사용자 정보가 변경되면,

현재 사용자 정보 전역 상태에 접근하고 있는 컴포넌트들만 다시 렌더링 되는 것을 확인해 보자.

전역 상태가 변경될 때 마다, 전역 상태에 접근하는 모든 컴포넌트들이 다시 렌더링 되기 때문에,

Todo 예제의 TodoList 상태를 Context API 전역 상태로 구현하는 것은 별로 바람직하지 않다.