# 15. Context API

- Context API는 리액트 프로젝트에서 전역적으로 사용할 데이터가 있을 때 융ㅇ한 기능이다.
- 예를 들어 사용자로그인정보, 애플리케이션 환경설정정보, 테마등...
- redux, react-router, styled-components등이 Context API를 기반으로 구현되어 있다.
- 리액트 V16.3 이후부터는 별도의 라이브러리 설치 없이도 사용할 수 있게 되었다.
- Context API를 사요하면 Context를 만들어 단 한 번에 원하는 값을 받아 와서 사용할 수 있다.

## 15.1 Ccontext API 사용법

### 15.1.1 새 Context만들기

- src/contexts/color.js파일 생성
- 새 Context를 만들 때는 createContext()함수를 사용한다.
- 파라미터에는 해당 Context의 기본상태를 지정한다.

src/contexts/color.js

```
import { createContext } from 'react';

const ColorContext = createContext({color: 'black'});

export default ColorContext;
```

### 15.1.2 Consumer사용하기

- ColorBox라는 컴퍼넌트를 만들어서 ColorContext안에 있는 색상을 사용
- props로 전달 받는 것이 아니라 ColorContext에 저장되어 있는 Concumer라는 컴퍼넌트를 통해 색상을 조회

Render Props의 예

```
const RenderPropsSample = ({ children }) => {
  return <div>결과 = {children(5)}</div>
}

export default RenderPropsSample;
```

상기와 같은 컴퍼넌트가 있다면

```
<RenderPropsSample>{value => value * 2}</RenderPropsSample>
```

처럼 RenderPropsSample에 cildren에게 함수를 전달하면 해당 컴퍼넌트는 children(5)를 실행하게 된다.

components/ColorBox.js

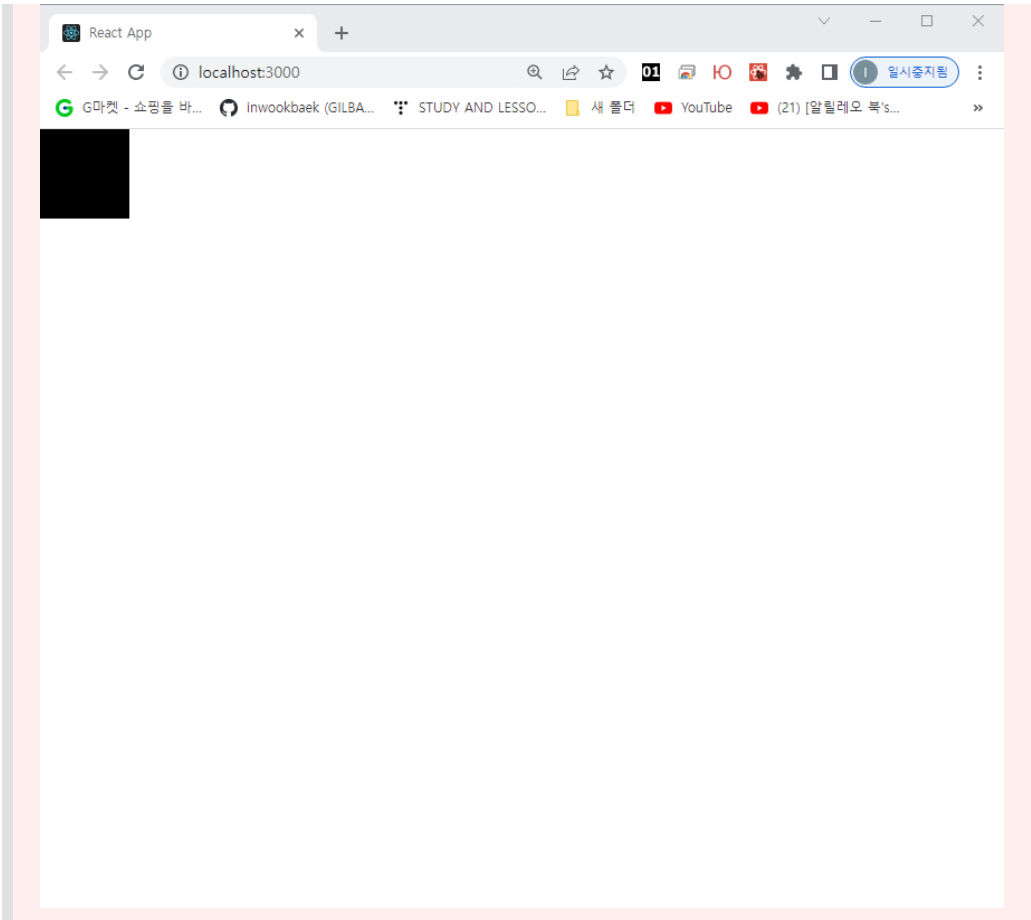
- Consumer사이에 중괄호를 열어서 그 안에 함수를 정의

- 이러한 패턴을 `Function as child 또는 Rebder Props`라고 한다.
- `컴퍼넌트의 children에 있어야할 자리에 일반 JSX 혹은 문자열이 아닌 함수를 전달` 하는 것이다.

```
import ColorContext from '../contexts/color';

const ColorBox = () => {
  return (
    <ColorContext.Consumer>
      {value => (
        <div style={{
          width: '64px',
          height: '64px',
          background: value.color
        }}>

        </div>
      )}
    </ColorContext.Consumer>
  )
}

export default ColorBox
```

src/App.js

```
import ColorBox from "./components/ColorBox";

function App() {
  return (
    <div>
      <ColorBox />
    </div>
  );
}

export default App;
```

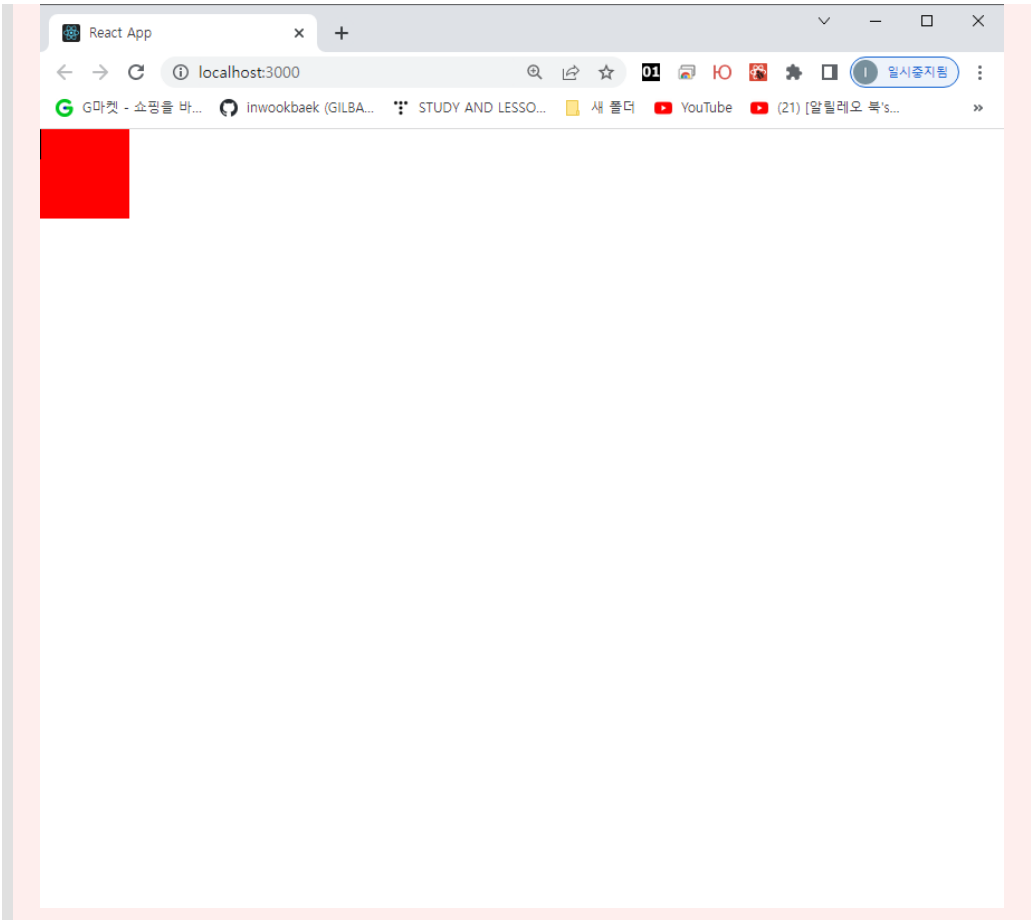



## 15.1.3 Provider

- `Provider를 사용하면 Context의 value를 변경할 수 있다.`
- Prover에 파라미터 즉, value를 전달하지 않으면 에러가 발생한다.

src/App.js 수정

```
import ColorContext from "./contexts/color";
import ColorBox from "./components/ColorBox";

function App() {
  return (
    <ColorContext.Provider value={{ color: 'red'}}>
      <div>
        <ColorBox />
      </div>
    </ColorContext.Provider>
  );
}

export default App;
```




## 15.2 동적 Context 사용하기

- Context에는 value뿐만 아니라 함수도 전달할 수 있다.

src/contexts/color.js

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

const ColorContext = createContext({
  state: { color: 'black', subcolor: 'red' },
  actions: {
    setColor: () => {},
    setSubcolor: () => {}
  }
});

const ColorProvider = ({ children }) => {
  const [color, setColor] = useState('black');
  const [subcolor, setSubcolor] = useState('red');

  const value = {
    state: { color, subcolor },
    actions: { setColor, setSubcolor }
  };
  return (
    <ColorContext.Provider value={value}>{children}</ColorContext.Provider>
  );
};

// const ColorConsumer = ColorContext.Consumer과 같은 의미
```

```
const { Consumer: ColorConsumer } = ColorContext;

// ColorProvider와 ColorConsumer 내보내기
export { ColorProvider, ColorConsumer };

export default ColorContext;
```

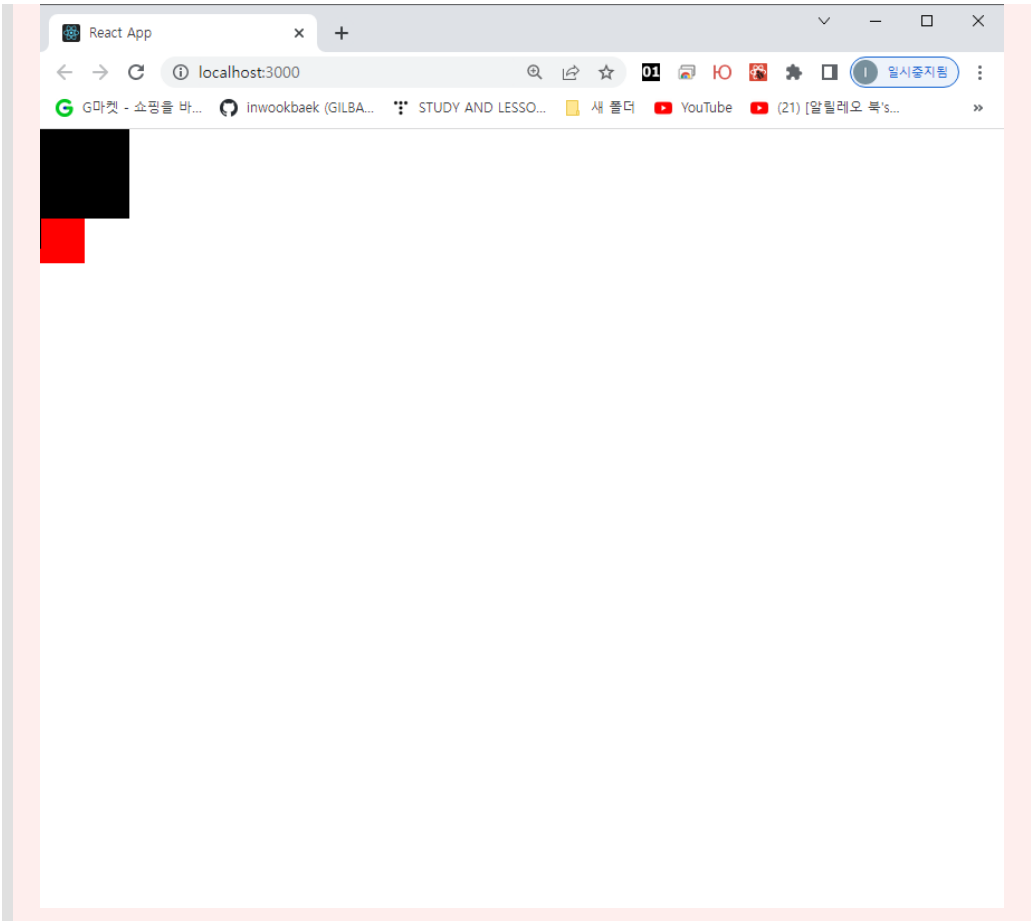
## 15.2.1 새로운 Context 프로젝트에 반영하기

src/App.js 수정

```
import ColorContext from "./contexts/color";
import ColorBox from "./components/ColorBox";

function App() {
  return (
    <ColorProvider>
      <div>
        <ColorBox />
      </div>
    </ColorProvider>
  );
}

export default App;
```

components/ColorBox.js

```
import { ColorConsumer } from '../contexts/color';

const ColorBox = () => {
  return (
    <ColorConsumer>
      {value => (
        <>
          <div style={{
            width: '64px',
            height: '64px',
            background: value.state.color
          }}/>

          <div style={{
            width: '32px',
            height: '32px',
            background: value.state.subcolor
          }}/>
        </>
      )}
    </ColorConsumer>
  )
}

export default ColorBox
```




객체 비구조화 할당 문법을 사용하면 다음과 같이 value를 조회하는 것을 생략할 수도 있다.

components/ColorBox.js - 객체 비구조화 할당 문법 적용

```
import { ColorConsumer } from '../contexts/color';

const ColorBox = () => {
  return (
    <ColorConsumer>
      {({ state }) => (
        <>
          <div style={{
            width: '64px',
            height: '64px',
            background: state.color
          }}/>

          <div style={{
            width: '32px',
            height: '32px',
            background: state.subcolor
          }}/>
        </>
      )}
    </ColorConsumer>
  )
}

export default ColorBox
```

## 15.2.2 색상 컴퍼넌트 만들기

- Context의 action에 정의한 함수를 호출하는 컴퍼넌트 SelectColors.js를 작성

components/SelectColors.js

```
const colors = ['red','orange','yellow','green','blue','indigo','violet'];

const SelectColors = () => {
  return (
    <div>
      <h2>색상을 선택하세요!</h2>
      <div style={{ display: 'flex'}}>
        {colors.map(color => (
          <div
            key={color}
            style={{
              background: color,
              width: '24px',
              height: '24px',
              cursor: 'pointer'
            }}
          />
        ))}
      </div>
      <hr />
    </div>
  )
}

export default SelectColors
```

src/App.js

```
import ColorBox from "./components/ColorBox";
import SelectColors from "./components/SelectColors";
import { ColorProvider } from "./contexts/color";

function App() {
  return (
    <ColorProvider>
      <div>
        <SelectColors />
        <ColorBox />
      </div>
    </ColorProvider>
  );
}

export default App;
```

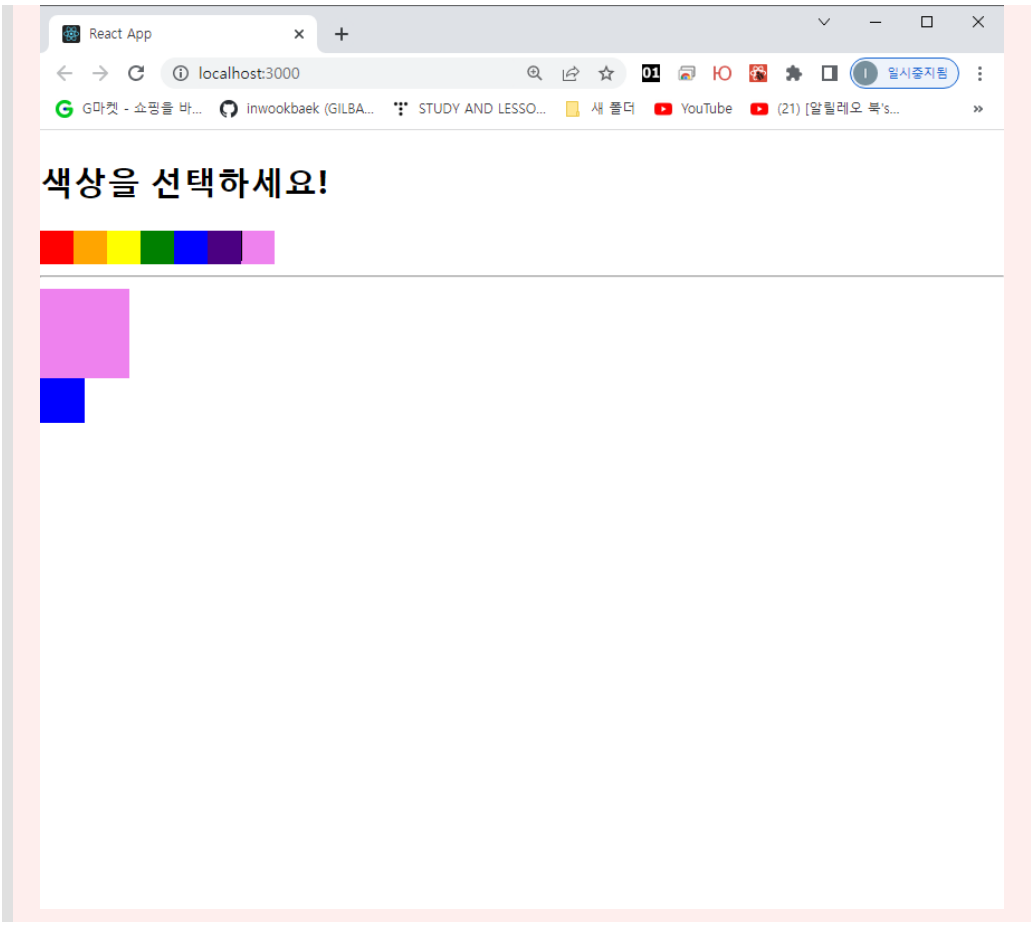
### SelectColors.js에 마우스 이벤트 적용하기

- 마우스 왼쪽버튼 - 큰 상자색상 변경, 마우스 오른쪽 버튼 - 작은 상자색상 변경

components/SelectColors.js

```
import { ColorConsumer } from "../contexts/color";
```

```
const colors = ['red','orange','yellow','green','blue','indigo','violet'];

const SelectColors = () => {
  return (
    <div>
      <h2>색상을 선택하세요!</h2>
      <ColorConsumer>
        {({ actions }) => (
          <div style={{ display: 'flex'}}>
          {colors.map(color => (
            <div
              key={color}
              style={{ background: color, width: '24px', height: '24px',
cursor: 'pointer' }}
              onClick={() => actions.setColor(color)}
              onContextMenu={e => { // onContextMenu 이벤트는 오른쪽 버튼 클릭
이벤트
                e.preventDefault();
                actions.setSubcolor(color);
              }}
            />
          ))}
          </div>
          )}
      </ColorConsumer>
      <hr />
    </div>
  )
}

export default SelectColors
```



# 15.3 Consumer대신 HOOK 또는 static contextType 사용하기

## 15.3.1 `useContext Hook` 사용하기

components/ColorBox.js - useContext 적용

- children에 함수를 전달하는 Render Props패턴이 불편하다면 useContext hook을 사용하면 간단하게 Context값을 조회할 수 있다.
- 다만 Hook은 함수형 컴퍼넌트에서만 사용할 수 있다

```
import { useContext } from 'react';
import ColorContext, { ColorConsumer } from '../contexts/color';

const ColorBox = () => {

  const { state } = useContext(ColorContext);
  return (
    <>
      <div style={{
        width: '64px',
        height: '64px',
        background: state.color
      }}/>

      <div style={{
        width: '32px',
        height: '32px',
        background: state.subcolor
      }}/>
    </>
  )
}

export default ColorBox
```

## 15.3.2 static contextType사용하기

- 클래스형 컴퍼넌트에서 Context를 쉽게 사용하려면 static contextType을 정의하는 방법이 있다.
- static contextType을 정의하면 클래스메서드에서도 Context에 넣어둔 함수를 호출할 수 있다.
- 하지만, 한 클래스에서 하나의 Context밖에 사용하지 못한다는 단점이 있다.
- 따라서 useContext hook을 사용하는 것을 권장

components/SelectColors_staticContextType.js

```
import { Component } from "react";
import ColorContext from "../contexts/color";

const colors = ['red','orange','yellow','green','blue','indigo','violet'];

class SelectColors_staticContextType extends Component {
  static contextType = ColorContext;
```

```
  handleSetColor = color => {
    this.context.actions.setColor(color);
  }

  handleSetSubColor = color => {
    this.context.actions.setSubcolor(color);
  }

  render() {
    return (
      <div>
        <h2>색상을 선택하세요!</h2>
        <div style={{ display: 'flex'}}>
          {colors.map(color => (
            <div
              key={color}
              style={{ background: color, width: '24px', height: '24px',
cursor: 'pointer' }}
              onClick={() => this.handleSetColor(color)}
              onContextMenu={e => {
                e.preventDefault();
                this.handleSetSubColor(color);
              }}
            />
          ))}
        </div>
        <hr />
      </div>
    );
  }
}

export default SelectColors_staticContextType
```

src/App.js

```
import ColorBox from "./components/ColorBox";
import SelectColors from "./components/SelectColors";
import SelectColors_staticContextType from
"./components/SelectColors_staticContextType";
import { ColorProvider } from "./contexts/color";

function App() {
  return (
    <ColorProvider>
      <div>
        <SelectColors />
        <ColorBox />
        <hr />
        <h4>static contextType</h4>
        <SelectColors_staticContextType />
        <ColorBox />
      </div>
    </ColorProvider>
  );
}
```

```
export default App;
```