Chapter 14. 컨텍스트(Context)

교재: 처음만난 리액트 (저자: 이인제, 한빛출판사)

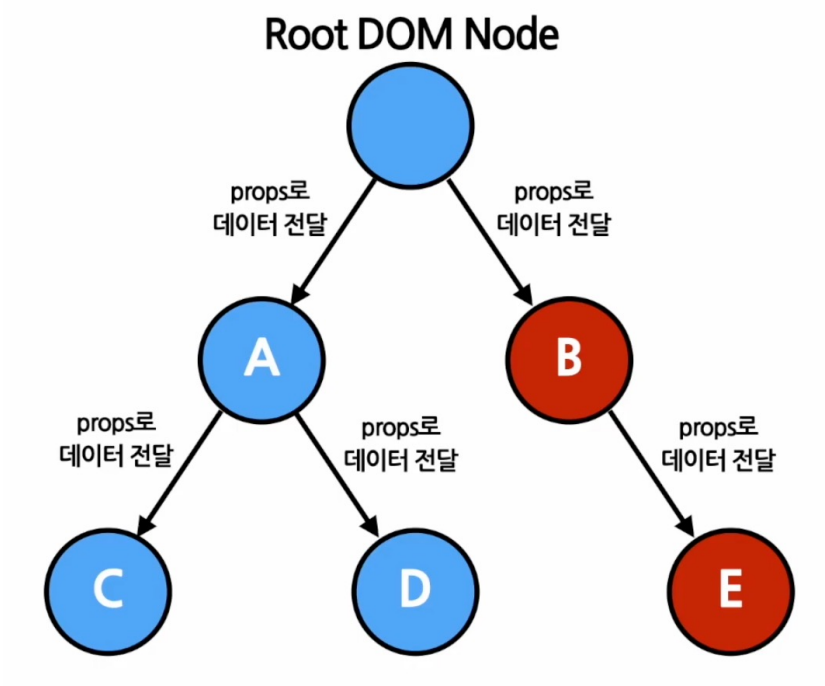
Contents

- CHAPTER 14: 컨텍스트(Context)

SECTION 14.1 컨텍스트란 무엇인가?

◦ 컴포넌트 간 데이터 전달 방식

- 리액트 컴포넌트 사이에서 props를 통해 데이터를 전달하는 방식



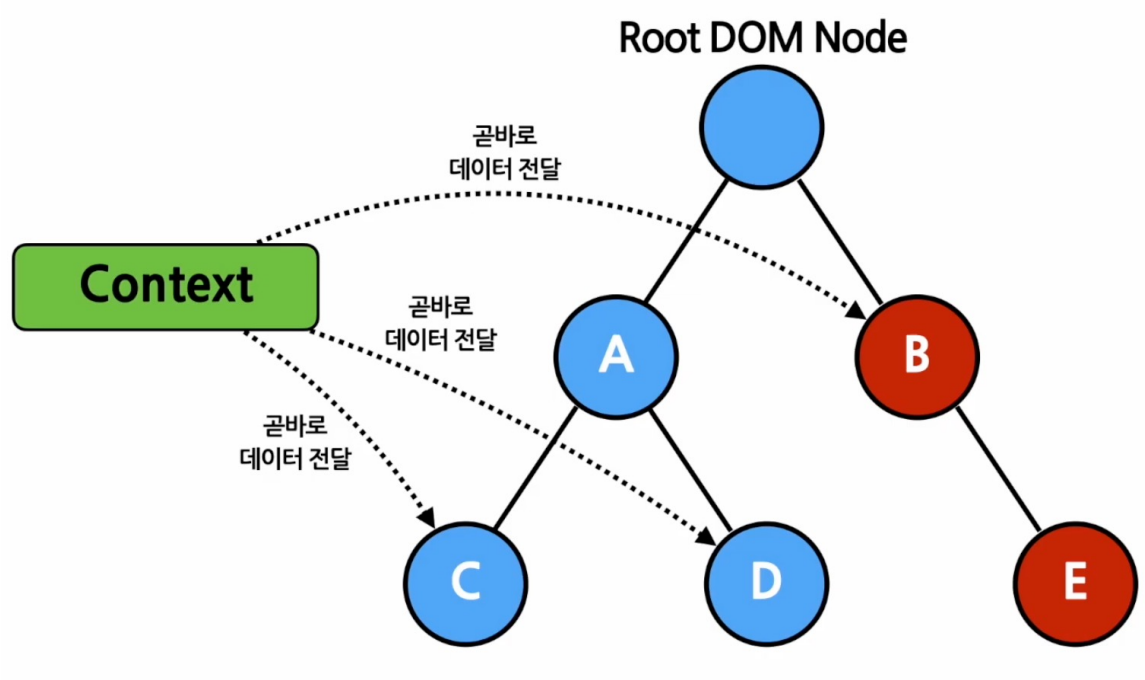
❖ props를 사용한 방식

✓ 부모에서 자식으로 단방향 전달

✓ 여러 컴포넌트에 걸쳐 자주 사용되는 데이터의 경우 depth가 깊어질수록 반복적인 코드 생성, 사용하기 불편

SECTION 14.1 컨텍스트란 무엇인가?

- **Context**
  - React 컴포넌트 트리 안에서 전역적으로 데이터를 공유할 수 있도록 고안된 방법
  - 컴포넌트 트리를 통해 곧바로 컴포넌트에 데이터를 전달하는 방식



❖ context를 사용한 방식

✓ 어떤 컴포넌트든지 데이터에 쉽게 접근할 수 있음

✓ 데이터를 한곳에서 관리
- 코드가 단순해지고 디버깅에 유리

SECTION 14.2 언제 컨텍스트를 사용해야 할까?

- **여러 컴포넌트에서 자주 사용하는 데이터를 사용할 경우**
  - 사용자의 로그인 여부, 로그인 정보, UI 테마, 현재 선택된 언어

    ex. 네비게이션 바에서 사용자의 로그인 여부에 따라 로그인/로그아웃 버튼 표시

```
function App(props) {
  return <Toolbar theme="dark" />;
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton theme={props.theme} />
    </div>
  );
}

function ThemedButton(props) {
  return <Button theme={props.theme} />;
}
```

❖ props로 데이터를 전달하는 코드

✓ 컴포넌트의 깊이가 깊어질수록 복잡

✓ 반복적인 코드를 계속해서 작성
- 비효율적이고 직관적이지 않음

SECTION 14.2 언제 컨텍스트를 사용해야 할까?

- **Context를 사용하는 예제**
  - React.createContext
  - Context.Provider / Context.Consumer

테마를 위한 컨텍스트 생성

```
const ThemeContext = React.createContext('light');

function App(props) {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
  );
}
```

Provider를 사용하여 하위 컴포넌트들에게 테마 데이터 전달

```
function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton(props) {
  return (
    <ThemeContext.Consumer>
      {value => <Button theme={value}/>}
    </ThemeContext.Consumer>
  );
}
```

가장 가까운 상위 테마 Provider를 찾아서 해당되는 값 사용

SECTION 14.3 컨텍스트를 사용하기 전에 고려할 점

- **context의 주된 용도 고려**
  - context는 다양한 레벨에 네스팅된 많은 컴포넌트에게 데이터를 전달하는 용도
  - context 사용 시 컴포넌트 재사용이 어려워질 수 있음
  - 꼭 필요한 경우에만 context 사용하고 컴포넌트 합성 방법 사용

SECTION 14.4 Context API

◦ React.createContext

- Context 객체를 생성하기 위한 함수

```
const MyContext = React.createContext(defaultValue);
```

- Context 객체를 구독하고 있는 컴포넌트를 렌더링할 때,
  트리 상위에서 가장 가까이에 있는 상위 레벨의 Provider로부터 현재값을 읽음
- 상위 레벨에 매칭되는 Provider가 없을 경우 defaultValue 사용
  Provider 없이 컴포넌트를 독립적으로 테스트할 때 유용
  defaultValue를 undefined로 설정하면 구독 컴포넌트에서 기본값을 사용하지 않음

SECTION 14.4 Context API

◦ Context.Provider

- Context 객체에 포함된 React 컴포넌트로 데이터 제공자 역할

```
<MyContext.Provider value={/* 어떤 값 */}>
```

- Context.Provider 하위 컴포넌트들은 해당 context의 데이터에 접근 가능
- value prop을 받아서 이 값을 하위 컴포넌트에게 전달
- context를 구독하는 컴포넌트들(consumer)에게 context의 변화를 알리는 역할
  > consumer 컴포넌트는 provider의 value prop이 변경되면 재렌더링
- 하나의 Provider 컴포넌트는 여러 개의 consumer와 연결될 수 있으며 중첩되어 사용 가능

SECTION 14.4 Context API

◦ Context.Consumer

- context의 데이터를 구독하는 React 컴포넌트

```
<MyContext.Consumer>
  {value => /* context 값을 이용한 렌더링 */}
</MyContext.Consumer>
```

- 함수 컴포넌트에서 context 구독에 사용
- 컴포넌트의 자식은 함수(function as a child)
  현재 context의 value를 받아서 React 노드를 반환
  value는 context의 Provider 중 가장 가까운 Provider의 value prop과 동일
  상위 컴포넌트에 Provider가 없다면 createContext()의 defaultValue 값과 동일

SECTION 14.4 Context API

◦ Context.Consumer

- function as a child
  - 컴포넌트의 자식으로 함수를 사용하는 방법
  - 리엑트에서는 기본적으로 하위 컴포넌트들을 children이라는 prop으로 전달
  - children으로 컴포넌트 대신 함수를 사용할 수 있음

```
// children이라는 prop을 직접 선언하는 방식
<Profile children={name => <p>이름: {name}</p>} />

// 컴포넌트로 감싸서 children으로 만드는 방식
<Profile>{name => <p>이름: {name}</p>}</Profile>
```

SECTION 14.4 Context API

- **Context.displayName**
  - 크롬의 리액트 개발자 도구에서 context를 표시할 때 이 displayName을 표시

```
const MyContext = React.createContext('some-value');
MyContext.displayName = 'MyDisplayName';

// 개발자 도구에 "MyDisplayName.Provider"로 표시됨
<MyContext.Provider />

// 개발자 도구에 "MyDisplayName.Consumer"로 표시됨
<MyContext.Consumer />
```

SECTION 14.4 Context API

- Context를 사용하는 예제

```
import React from 'react';

const ThemeContext = React.createContext('light');

function App(props) {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}
```

```
const styles = {
  dark: {
    background: "gray",
    color: "white"
  },
  light: {
    background: "white",
    color: "black"
  }
}

function ThemedButton(props) {
  return (
    <ThemeContext.Consumer>
      {value => <button style={value === 'dark' ?
        styles.dark : styles.light}>Button</button>}
    </ThemeContext.Consumer>
  )
}

export default App;
```

SECTION 14.5 useContext

◦ 함수 컴포넌트에서 컨텍스트를 쉽게 이용하기 - useContext hook

- useContext() 훅은 context 객체를 인자로 받아서 현재 value 값을 리턴

```
function MyComponent(props) {
  const value = useContext(MyContext);

  return (
    ...
  )
}
```

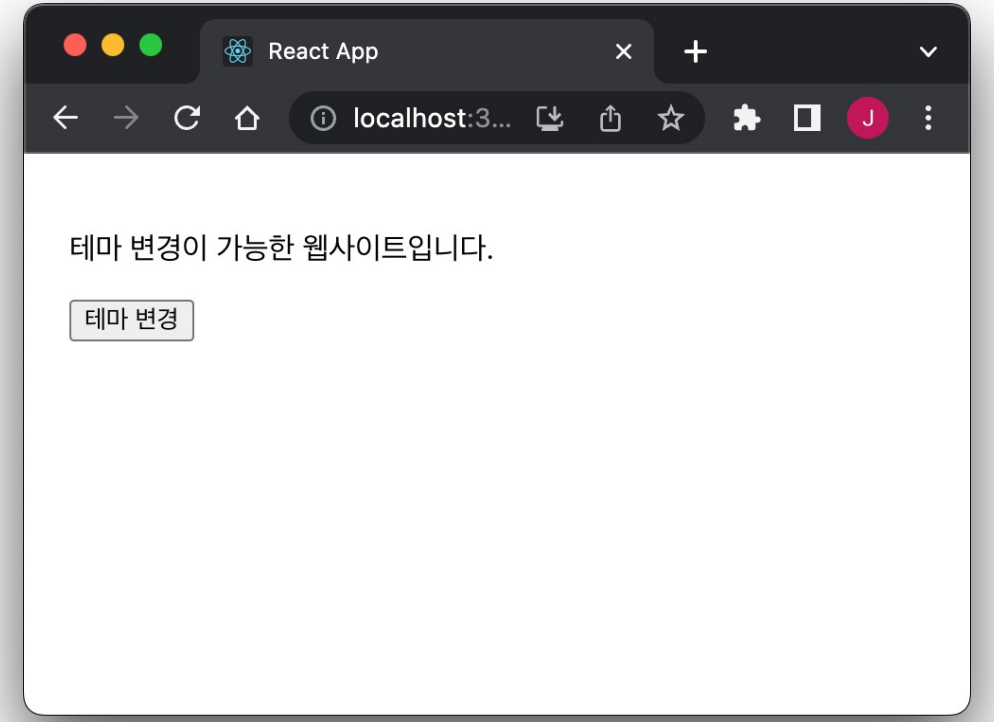
- 컴포넌트 트리상에서 가장 가까운 상위 Provider로부터 context의 value 값을 받아 옴
- context 값이 변경되면 변경된 값과 함께 useContext() 훅을 사용하는 컴포넌트가 재랜더링 됨

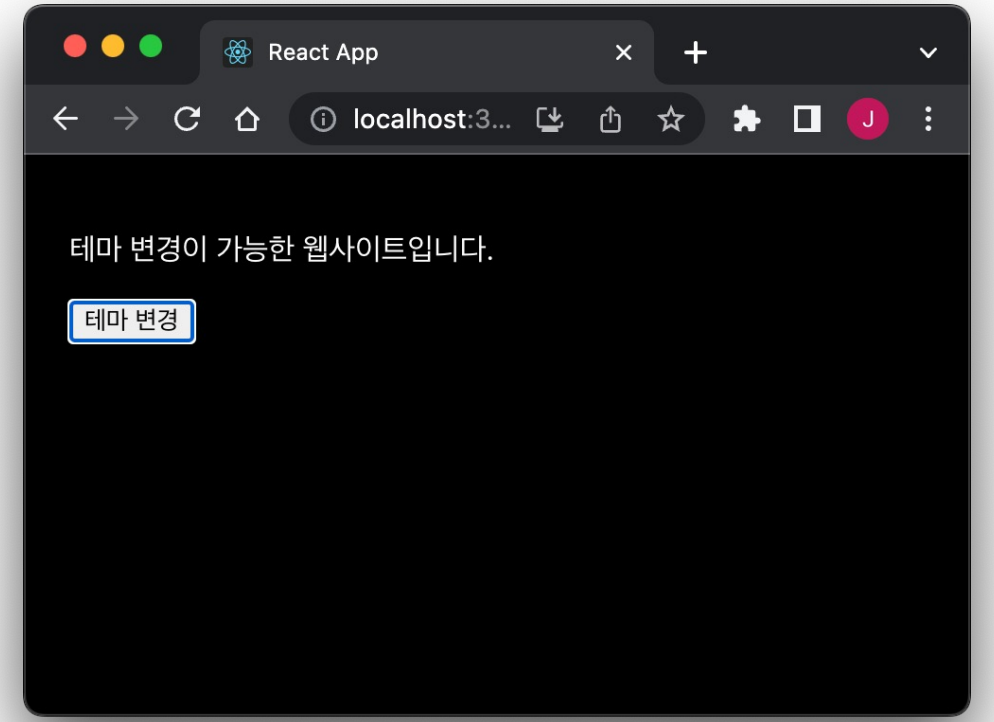
```
useContext(MyContext);

// 잘못된 사용법
useContext(MyContext.Consumer);
useContext(MyContext.Provider);
```

SECTION 14.6 context를 사용하여 테마 변경 기능 만들기

- **context를 사용하는 테마 변경 component**





SECTION 14.6 context를 사용하여 테마 변경 기능 만들기

- context, consumer

```jsx
import React from 'react';

const ThemeContext = React.createContext();

ThemeContext.displayName = 'Theme-Context';

export default ThemeContext;
```

```jsx
import { useContext } from 'react';
import ThemeContext from './ThemeContext';

function MainContent(props) {
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <div
      style={{
        width: '100vw',
        height: '100vh',
        padding: '1.5rem',
        backgroundColor: theme === 'light' ? 'white' : 'black',
        color: theme === 'light' ? 'black' : 'white',
      }}
    >
      <p>테마 변경이 가능한 웹사이트입니다.</p>
      <button onClick={toggleTheme}>테마 변경</button>
    </div>
  );
}

export default MainContent;
```

SECTION 14.6 context를 사용하여 테마 변경 기능 만들기

- **provider**

```jsx
import { useState, useCallback } from 'react';
import ThemeContext from './ThemeContext';
import MainContent from './MainContent';

function DarkOrLight(props) {
  const [theme, setTheme] = useState('light');

  const toggleTheme = useCallback(() => {
    if (theme === 'light') {
      setTheme('dark');
    } else if (theme === 'dark') {
      setTheme('light');
    }
  }, [theme]);

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      <MainContent />
    </ThemeContext.Provider>
  )
}

export default DarkOrLight;
```

useCallback()

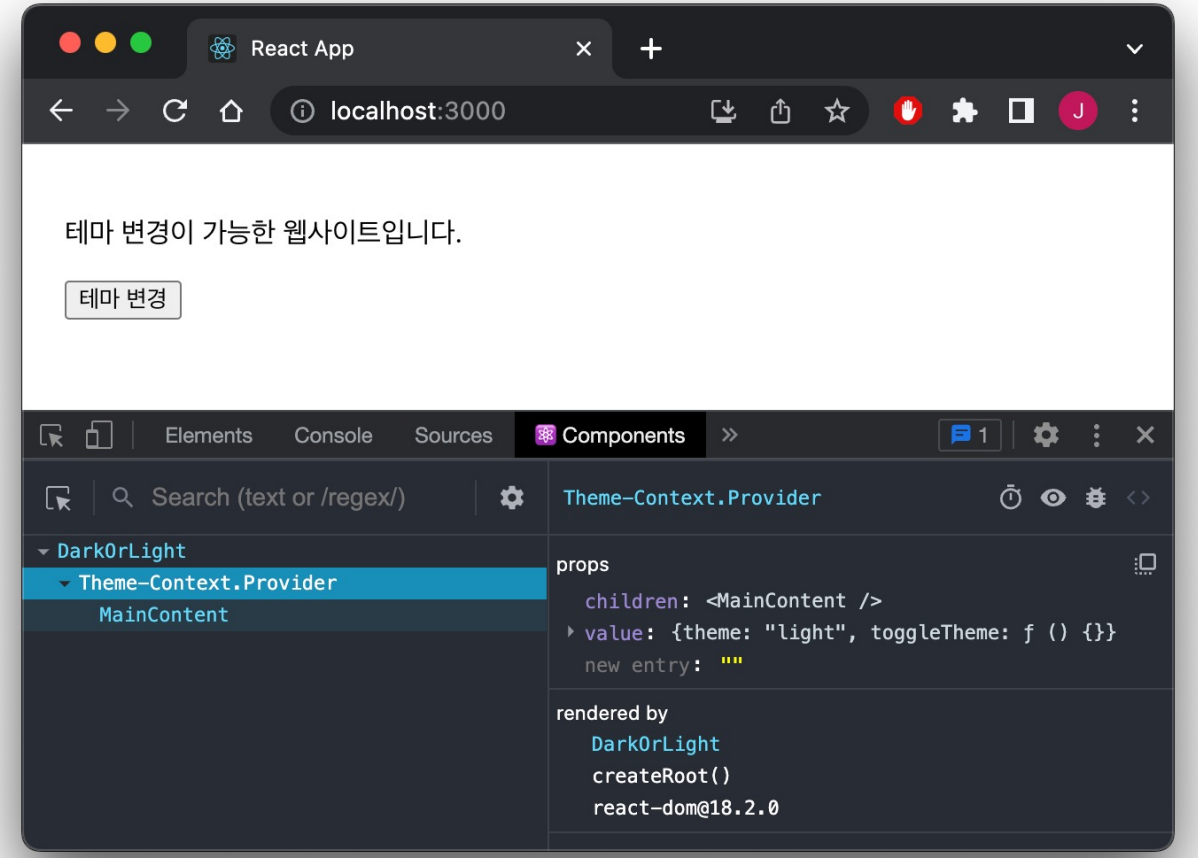
- 함수를 메모이제이션 하기 위한 hook

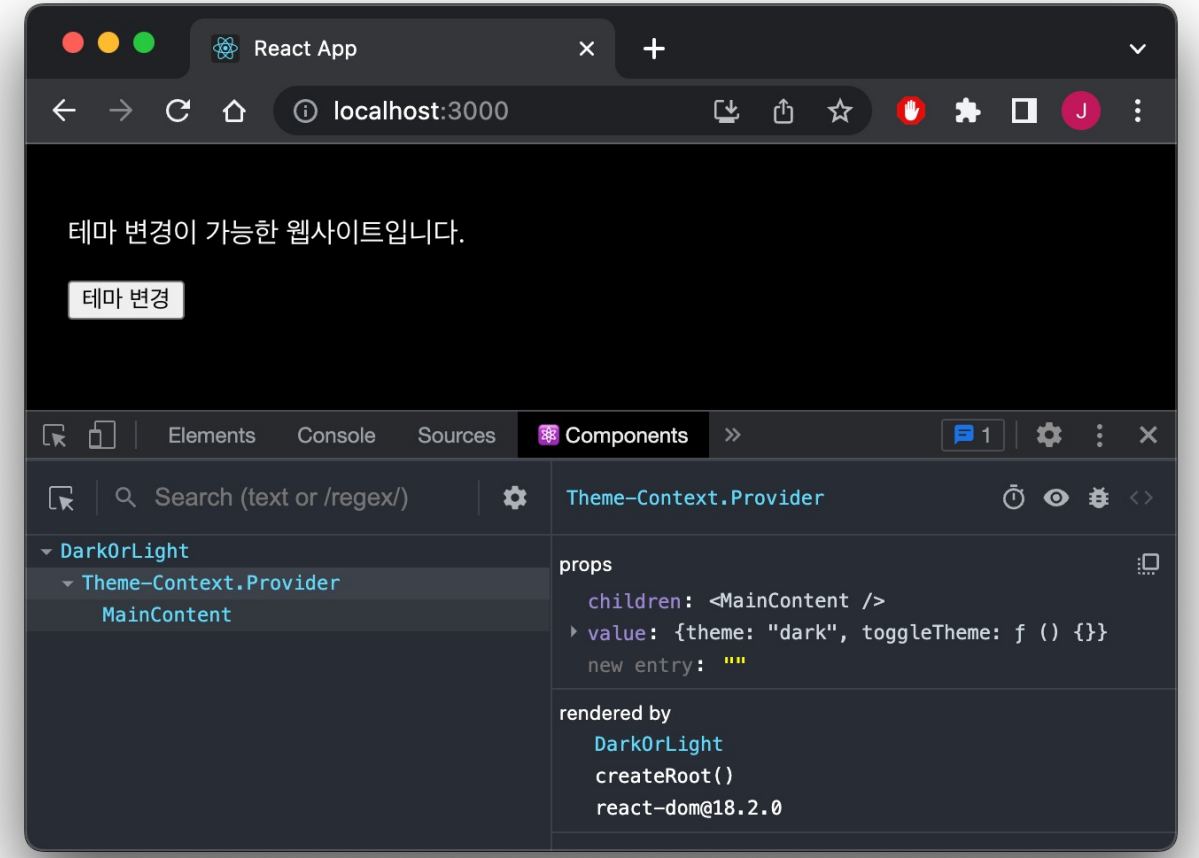
- useCallback() hook을 사용하지 않고
 컴포넌트 내에 함수를 정의할 경우,
 렌더링이 일어날 때마다 함수가 새로 정의됨

- useCallback() hook을 사용하면
 특정 변수가 변경될 때만 함수를 재정의

SECTION 14.6 context를 사용하여 테마 변경 기능 만들기

- 개발자 도구 > 컴포넌트 탭 - context 값 확인





[요약]

- 컨텍스트란?
  - 컴포넌트들 사이에서 데이터를 props를 통해 전달하는 것이 아닌 컴포넌트 트리를 통해 곧바로 데이터를 전달하는 방식
  - 어떤 컴포넌트든지 컨텍스트에 있는 데이터에 쉽게 접근할 수 있음

- 언제 컨텍스트를 사용해야 할까?
  - 여러 컴포넌트에서 계속해서 접근이 일어날 수 있는 데이터들이 있는 경우
  - Provider의 모든 하위 컴포넌트가 얼마나 깊이 위치해 있는지 관계없이 컨텍스트의 데이터를 읽을 수 있음

- 컨텍스트 사용 전 고려할 점
  - 컴포넌트와 컨텍스트가 연동되면 재사용성이 떨어짐
  - 다른 레벨의 많은 컴포넌트가 데이터를 필요로 하는 경우가 아니라면, 기존 방식대로 props를 통해 데이터를 전달하는 것이 더 적합

[요약]

- Context API
  - React.createContext()
    - 컨텍스트를 생성하기 위한 함수
    - 컨텍스트 객체를 리턴함
    - 기본값으로 underined를 넣으면 기본값이 사용되지 않음
  - Context.Provider
    - 모든 컨텍스트 객체는 Provider라는 컴포넌트를 갖고 있음
    - Provider 컴포넌트로 하위 컴포넌트들을 감싸주면 모든 하위 컴포넌트들이 해당 컨텍스트의 데이터에 접근할 수 있게 됨
    - Provider에는 value라는 prop이 있으며, 이것이 데이터로써 하위에 있는 컴포넌트들에게 전달됨
    - 여러 개의 Provider 컴포넌트를 중첩시켜 사용할 수 있음
  - Context.Consumer
    - 컨텍스트의 데이터를 구독하는 컴포넌트
    - 데이터를 소비한다는 뜻에서 consumer 컴포넌트라고도 부름
    - consumer 컴포넌트는 컨텍스트 값의 변화를 지켜보다가 값이 변경되면 재랜더링 됨
    - 하나의 Provider 컴포넌트는 여러 개의 consumer 컴포넌트와 연결될 수 있음
    - 상위 레벨에 매칭되는 Provider가 없을 경우 기본값이 사용됨

[요약]

- Context API
  - Context.displayName
    - 크롬의 리액트 개발자 도구에서 표시되는 컨텍스트 객체의 이름

- useContext()
  - 함수 컴포넌트에서 컨텍스트를 쉽게 사용할 수 있게 해주는 훅
  - React.createContext() 함수 호출로 생성된 컨텍스트 객체를 인자로 받아서 현재 컨텍스트의 값을 리턴
  - 컨텍스트의 값이 변경되면 변경된 값과 함께 useContext() 훅을 사용하는 컴포넌트가 재렌더링 됨