Ask Company

리액트와 함께 장고 시작하기 / 리액트

Context API

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

Context API의 필요성

여러 단계에 걸쳐, 하위 컴포넌트로 속성값을 전달할 때에는, 각 단계별로 속성값을 전달하는 코드를 기계적/반복적으로 작성해야하는 번거로움

```
const App = () => <Level1 message="Hello Context API" />;

const Level1 = ({ message }) => (
  <div>
    Level1
    <Level2 message={message} />
  </div>
);

const Level2 = ({ message }) => (
  <div>
    Level2
    <Level3 message={message} />
  </div>
);

const Level3 = ({ message }) => <div>Level3 :
{message}</div>;
```

객체, Array, state 등을 전달

기계적인 속성값 전달

Context API 활용 예시

Level1/Level2 컴포넌트가 중간에 개입하지 않고서도, 값을 전달 가능

```
// Provider/Consumer 속성을 가집니다.
const MessageContext = React.createContext('default message');

const App = () => (
  <MessageContext.Provider value="Hello Context API">
    <Level1 />
  </MessageContext.Provider>
);
```

Consumer에서 Provider 찾기에 실패했을 경우에 사용될 기본값

해당 value가 변경될 경우,
하위 모든 컴포넌트는 다시 렌더링
(하위 컴포넌트가 shouldComponentUpdate가 false를 반환하더라도)

```
const Level1 = () => (
  <div>
    Level1
    <Level2 />
  </div>
)

const Level2 = () => (
  <div>
    Level2
    <Level3 />
  </div>
)

const Level3 = () => (
  <div>
    <MessageContext.Consumer>
      {message => `Level 3: ${message}`}
    </MessageContext.Consumer>
  </div>
)
```

상위로 올라가며, 가까운 Provider를 찾습니다.
관련 Provider가 없을 경우, 기본값을 사용

Consumer내의 children은
필히 함수로만 적용해야만, 값을 받아올 수 있습니다.

Context 객체를 중첩하기

아래 코드는 render에서만 사용 가능 → 다른 생명주기 메서드에서는 사용 불가

```
const AlertContext = React.createContext(null);
const MessageContext = React.createContext('default message'); // Provider/Consumer를 반환

const Level3 = () => (
  <div>
    <AlertContext.Consumer>
      {alert => (
        <MessageContext.Consumer>
          {message => `Level 3: ${alert}, ${message}`}
        </MessageContext.Consumer>
      )}
    </AlertContext.Consumer>
  </div>
);

// Level1, Level2 생략
```

```
const App = () => (
  <AlertContext.Provider value="Alert Message">
    <MessageContext.Provider value="Hello Context API">
      <Level1 />
    </MessageContext.Provider>
  </AlertContext.Provider>
);
```

다른 컴포넌트 메서드에서 다수의 Context 접근하기 (1/2)

고차 컴포넌트를 활용하여, 속성값(props)으로 전달토록 구성 가능

```
const Level1 = () => (
  <div>
    Level1
    <Level2Wrapper />
  </div>
);
```

```
const Level2Wrapper = (props) => (
  <AlertContext.Consumer>
    {alert => (
      <MessageContext.Consumer>
        {message => (
          <Level2
            {...props}
            alert={alert}
            message={message} />
        )}
      </MessageContext.Consumer>
    )}
  </AlertContext.Consumer>
);
```

```
class Level2 extends React.Component
{
  render() {
    const { alert, message } =
this.props;
    return (
      <div>
        Level 2: {alert}, {message}
      </div>
    );
  }
}
```

```
const App = () => (
  <AlertContext.Provider value="Alert Message">
    <MessageContext.Provider value="Hello Context API">
      <Level1 />
    </MessageContext.Provider>
  </AlertContext.Provider>
);
```

다른 컴포넌트 메서드에서 다수의 Context 접근하기 (2/2)

함수형 컴포넌트에서는 useContext 훅을 통해, Consumer를 보다 간결하게 처리

```
const Level1 = () => (
  <div>
    Level1
    <Level2 />
  </div>
);
```

```
const Level2 = () => {
  const alert = React.useContext(AlertContext);
  const message = React.useContext(MessageContext);
  return (
    <div>
      Level 2: {alert}, {message}
    </div>
  );
};
```

```
const App = () => (
  <AlertContext.Provider value="Alert Message">
    <MessageContext.Provider value="Hello Context API">
      <Level1 />
    </MessageContext.Provider>
  </AlertContext.Provider>
);
```

하위 컴포넌트에서 Context 데이터를 수정하기

Provider측의 state를 수정하는 함수를 Context로 전달하여 호출토록 구현

Redux도 유사한 방식

```
const CounterContext = React.createContext(null);

const Level3 = () => {
  const counter = React.useContext(CounterContext);
  return (
    <div onClick={counter.onIncrement}>
      Level 3: {counter.value}
    </div>
  );
};
```

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 0,
      onIncrement: this.onIncrement
    };
  }

  onIncrement = () => {
    const { value } = this.state;
    this.setState({
      value: value + 1
    });
  };

  render() {
    return (
      <CounterContext.Provider value={this.state}>
        <Level1 />
      </CounterContext.Provider>
    );
  }
}
```

Life is short.
You need Python and Django.

I will be your pacemaker.