Ask Company

리액트와 함께 장고 시작하기 / 리액트

함수형 컴포넌트와 Hook

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

클래스형 컴포넌트의 한계

class가 코드 재사용성과 코드 구성을 더 어렵게 만든다.

자바스크립트의 this는 다른 언어와 다르게 작동하며, 개발자는 이벤트 핸들러가 등록되는 방법을 기억해야만 한다.

부수적으로 작성해야하는 코드가 많다.

서로 연관성이 없는 다수 로직을 하나의 생명주기 메서드에서 구현하는 경우가 많다.

코드 압축이 잘 안 되는 경우가 있고, 컴파일 단계에서 코드 최적화를 어렵게 만든다.

componentDidMount에서 등록하고 componentWillUnmount에서 해제를 깜빡할 수 있다.

... 등등.

https://ko.reactjs.org/docs/hooks-intro.html

함수형 컴포넌트

클래스형 컴포넌트에 적용했던 것들을 대부분 적용 가능 (Hook을 활용)

현재 리액트 팀에서도 함수형 컴포넌트와 Hook 개발에 집중

새로이 작성하는 컴포넌트는 함수형을 쓰기를 권장

클래스 컴포넌트에 대한 호환성 보장

```
class Message1 extends React.Component {
  render() {
    return (
      <div>{this.props.message}</div>
    );
  }
}
```

```
const Message2 = (props) => (
  <div>{props.message}</div>
);
```

```
const Message3 = ({ message }) => (
  <div>{message}</div>
);
```

함수형 컴포넌트로 재현할 수 없는 메서드

리액트 버전 16.8 기준

getSnapshotBeforeUpdate

getDerivedStateFromError

componentDidCatch

"관심사의 분리"로 컨테이너 설계하기

비슷한 기능을 하는 코드끼리 모아서 별도로 관리하는 것

하나의 컴포넌트에서 모든 기능을 구현할 수 없다.

비지니스 로직과 상탯값의 유무로 컨테이너를 분리하기 (문법차이X, 역할차이)

프레이젠테이션 컴포넌트

데이터를 SET/GET하는 방법에 대해서 관여하지 않습니다. 속성값을 통해 callback 함수와 데이터를 받습니다. → 재사용성이 좋습니다.

상태를 거의 가지지 않습니다. (UI 상태에 대한 것들을 가지기는 합니다.)

컨테이너 컴포넌트

상탯값들을 직접 제어, Redux로부터 데이터를 받고 action을 실행(dispatch)

데이터와 함수들을 Presentation 컴포넌트 등에 제공

함수형 컴포넌트로 구현하기 용이

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

필수 Hooks

(useState, useEffect, useCallback)

Hook

리액트 버전 16.8에 새로이 추가

Hook을 통해 함수형 컴포넌트에서도 상태값과 여러 React의 기능을 활용 가능

props, state, context, refs, life-cycle에 대한 보다 직관적인 API를 제공

같은 로직을 한 곳으로 모을 수 있어서 가독성에 좋다.

클래스형 컴포넌트에서 사용할려면, 커스텀 Wrapper 컴포넌트가 필요하다.

https://ko.reactjs.org/docs/hooks-overview.html

useState 훅

컴포넌트 내에서 상탯값을 유지/변경하고 싶어요.

```
const Counter1 = () => {
  const [age, setAge] = React.useState(0);
  return (
    <div>
      age : {age}
      <button onClick={() => setAge(age => age+1)}>Inc</button>
    </div>
  );
};
```

클래스 컴포넌트와 다르게, 필요한 갯수만큼 상탯값을 정의할 수 있습니다.

```
const Counter2 = () => {
  const [state, setState] = React.useState({ age: 0 });
  return (
    <div>
      age : {state.age}
      <button onClick={() => setState(state => ({age: state.age+1}))}>Inc</button>
    </div>
  );
};
```

클래스 컴포넌트의 상탯값처럼 object 형태로도 가능하지만, 주의하세요. (다음 페이지에 계속)

useState 훅은 이전 상태값을 항상 지웁니다. (1/2)

useState 단위로 object를 저장하고, object 통채로 변경합니다.
클래스 컴포넌트 상탯값의 setState와는 달라요.

```
const Person = () => {
  const [state, setState] = React.useState({ age: 0, name: 'Tom' });
  return (
    <div>
      name: {state.name}, age : {state.age}
      <button onClick={() => setState(state => ({age: state.age+1}))}>Inc</button>
      <br/>
      <input type="text" value={state.name}
             onChange={(e) => setState({name: e.target.value})} />
    </div>
  );
};
```

```
Warning: A component is changing a controlled input of type text to be uncontrolled. Input elements
should not switch from controlled to uncontrolled (or vice versa). Decide between using a controlled
or uncontrolled input element for the lifetime of the component. More info: https://fb.me/react-
controlled-components
    in input (at App.js:12)
    in div (at App.js:8)
    in Person (at App.js:20)
    in div (at App.js:19)
    in App (at src/index.js:7)
```

useState 훅은 이전 상태값을 항상 지웁니다. (2/2)

클래스 컴포넌트의 setState와는 다르게,

useState의 setter 함수에서는 매번 전체값을 지정해줘야 합니다.

```
const Person = () => {
  const [state, setState] = React.useState({age: 0, name: 'Tom'});
  return (
    <div>
      name: {state.name}, age : {state.age}
      <button onClick={() => setState(state => ({ ...state, age: state.age+1 }))}>
        Inc
      </button>
      <br/>
      <input type="text"
             value={state.name}
             onChange={(e) => setState(state => ({ ...state, name: e.target.value }))} />
    </div>
  );
};
```

Tip) Immer 라이브러리를 쓰면,
코드가 좀 더 직관적일 때가 있어요.

useEffect 훅 (1/4) 로직 단위로 반복해서 사용

생명주기의 componentDidMount/componentDidUpdate에 대응

컴포넌트 마운트 뒤에 수행할 코드가 있을 때

특정 속성값/상탯값이 변경되었을 때, 수행할 코드가 있을 때

```
const Counter = () => {
  const [count, setCount] = React.useState(0);

  React.useEffect(() => {
    document.title = `count: ${count}`;
  });

  return (
    <div>
      count : {count}
      <button onClick={() => setCount(count => count+1)}>Inc</button>
    </div>
  );
};
```

클래스형 컴포넌트에서는 본 로직을 componentDidMount와 componentDidUpdate에 중복해서 적용해야 합니다.
그런데, 2번째 인자를 지정하지 않았기에 매 render시마다 호출이 됩니다. 2번째 인자를 꼭 지정해주세요. ☹

useEffect 훅 (2/4)

```
const PostDetail = ({ postId }) => {
  const [post, setPost] = React.useState(null);
  React.useEffect(
    () => {
      // requestPostDetail(postId).then(post => setPost(post));
      setPost({ title: 'hello title', content: '내용' });
    }
  , [postId]);

  return (
    <div>
      {!post && `포스팅 정보 가져오는 중 ...`}
      {post && (
        <div>
          <h2>포스팅: #{postId} - {post.title}</h2>
          {post.content}
        </div>
      )}
    </div>
  );
};
```

2번째 인자로 Array를 지정하면
mount/update 시마다 호출되지 않고,
참조되는 상탯값/속성값이 변경될 때에만 호출

useEffect 훅 (3/4)

```
const TraceWidth = () => {
  const [width, setWidth] = React.useState(window.innerWidth);

  React.useEffect(
    () => {
      const onResize = () => setWidth(window.innerWidth);
      window.addEventListener('resize', onResize);
      return () => {
        window.removeEventListener('resize', onResize);
      };
    },
    []
  );

  return <div>window.innerWidth: {width}</div>;
};
```

1번째 인자의 함수에서 반환된 함수는 컴포넌트가 언마운트되거나 1번째 인자의 함수가 호출되기 직전에 호출

2번째 인자로 빈 Array를 지정하면, 컴포넌트가 마운트될 때에만 호출

componentDidUpdate에서는 호출되지 않아요.

useEffect 훅 (4/4) - Clock 샘플

```
const Clock = () => {
  const [date, setDate] = React.useState(new Date());
  React.useEffect(
    () => {
      const interval = setInterval(() => setDate(new Date()), 1000);
      return () => clearInterval(interval);
    },
    []);
  return (
    <div>{date.toISOString().slice(11, 19)}</div>
  );
}
```

useCallback 훅

컴포넌트가 렌더링될 때마다, 함수를 생성해서 속성값으로 지정하면, 성능 저하

- 요즘 브라우저 성능이 좋아져서, 성능에 미치는 영향이 적다고는 합니다.
- 함수를 재활용하여, 불필요한 컴포넌트 재렌더링을 제거하기

아래 2개 코드는 동일한 동작을 합니다.

```
useCallback(() => {
  console.log("hello");
}, []);
```

```
useMemo(() => {
  const fn = () => {
    console.log("hello");
  };
  return fn;
}, []);
```

Hook, 유의사항

1. 하나의 컴포넌트에서 훅을 호출하는 순서는 일정해야 합니다.

   리액트에서 각 훅을 구별하는 유일한 기준이 훅이 정의된 순서이기 때문

   → 내부적으로 각 훅은 Array에 담겨져 처리됩니다.

2. 함수형 컴포넌트 또는 커스텀 훅 안에서만 호출되어야만 합니다.

   클래스형 컴포넌트나 일반 함수에서는 사용할 수 없습니다.

3. 최상위 수준 (Top Level)에서 훅을 호출해야만 합니다.

   반복문이나 조건문식, 중첩된 함수에서 훅을 호출해서는 안 됩니다.

https://reactjs.org/docs/hooks-rules.html

렌더링 퍼포먼스 최적화

최적화의 개념에 우선하여, 발적화를 방지합시다.

- 오용하지 않기에 집중
- 요즘은 프레임워크/라이브러리 차원에서 자동으로 처리되는 것들이 많습니다.

"제대로 이해하고 사용하고 있는 가?" 에 포커스.

Life is short.
You need Python and Django.

I will be your pacemaker.