함수형 vs 클래스형 컴포넌트, Lifecycle API

함수형 vs 클래스형 컴포넌트

React의 컴포넌트 선언 방식은 함수형, 클래스형 2가지로 나뉩니다. 초기엔 클래스형 컴포넌트가 자주 사용되었지만 현재는 자주 사용되지 않는데 이를 Lifecycle API 와 연결지어 얘기해보려고 합니다.

선언 방식

클래스형 컴포넌트

```
import React, { Component } from 'react'

class App extends Component {
  render() {
    const name = 'react'

    return <div className="react">{name}</div>
  }
}

export default App;
```

클래스형 컴포넌트의 경우 class 키워드 뒤에 함수 명을 입력하고, Component를 상속받아야 합니다. 또한, render() 함수가 반드시 필요하며, 사용하지 않을 시 컴포넌트 렌더링이 되지 않습니다.

함수형 컴포넌트

```
import React from 'react';

const App = () => {
  const name = 'react'

  return <div className="react">{name}</div>
}

export default App;
```

함수형 컴포넌트의 경우 말 그대로 함수처럼 선언이 가능하며, render() 함수 또한 필요 없습니다.

차이

초기엔 클래스형 컴포넌트를 주로 사용했다고 얘기했습니다. 그 이유는 클래스형에서 state, lifeCycle 관련 기능 사용이 가능했기 때문인데요. 여기서 state란 컴포넌트 내부에서 바뀔 수 있는 값이고, lifeCycle API란 컴포넌트가 DOM 위에 생성되기 전 후 및 데이터가 변경되어 상태를 업데이트 하기 전 후로 실행되는 함수들입니다. 즉, 렌더링 과정에서 사용할 수 있는 함수들을 의미합니다.

추가적으로 클래스형이 함수형보다 메모리 자원을 조금 더 사용합니다.

함수형 컴포넌트는 위 state, lifeCycle API 사용이 불가능하여 주로 쓰이지 못했으나 이후 React Hooks가 나타나면서 함수형 컴포넌트에서도 사용이 가능해졌습니다. 예를 들어 useState 함수를 이용해 state 사용이 가능해졌고, useEffect, useCallback 등의 Hook 등장으로 lifeCycle API 또한 사용할 수 있게 되었습니다.

state 사용 방법

클래스형

```
class App extends Component {
  render() {
    state = {
      count: 1
    }

    onClick = {() => this.setState({ count: count + 1 })}

    return <div onClick={onClick}>{count}</div>
  }
}
```

함수형

```
const App = () => {
  const [count, setCount] = useState(0)

  onClick = () => setCount(count + 1)

  return <div onClick={onClick}>{count}</div>
}
```

props 사용 방법

React에서 props란 부모 요소가 자식 요소에게 주는 속성 값입니다. 모든 React 컴포넌트는 props를 다룰 때 반드시 순수함수처럼 동작해야 하기 때문에 props 자체를 수정해서는 안됩니다.

클래스형

```
class App extends Component {
  render() {
    const { name } = this.props;

    return <div>Hi, my name is {name}</div>
  }
}
```

함수형

```
const App = ({ name }) => {
  return <div>Hi, my name is {name}</div>
}
```

LifeCycle API

클래스형 컴포넌트엔 정말 다양한 LifeCycleAPI 가 있습니다. 이를 살펴보면…

- 컴포넌트 생성 시 : constructor → componentWillMount → render → componentDidMount 순서로 실행됩니다.
- 컴포넌트 제거시 : componentWillUnmount 메서드만 실행됩니다.
- 컴포넌트의 props 변경 시 : componentWillReceiveProps → shouldComponentUpdate → componentWillUpdate → render → componentDidUpdate 순으로 실행됩니다.
- 컴포넌트의 state 변경 시 : shouldComponentUpdate → componentWillUpdate → render → componentDidUpdate 순으로 실행됩니다.

각각의 메서드 들을 살펴보면…

constructor

```
constructor(props) {
  super(props);
  console.log("constructor");
}
```

생성자 메서드로 컴포넌트가 처음 만들어 질 때 실행됩니다.

componentWillMount

```
componentWillMount() {
  console.log("componentWillMount");
}
```

컴포넌트가 DOM 위에 만들어지기 전에 실행됩니다.

render

```
render () {
  ...
}
```

컴포넌트 렌더링을 담당합니다.

componentDidMount

```
componentDidMount() {
  console.log("componentDidMount")
}
```

컴포넌트가 렌더링을 마친 후 실행되는 메서드입니다.

componentWillReceiveProps

```
componentWillReceiveProps(nextProps){
  console.log("componentWillReceiveProps: " + JSON.stringify(nextProps));
}
```

컴포넌트가 props를 새로 받았을 때 실행됩니다. props의 변경에 따라 state를 업데이트 해야 할 때 사용하면 유리합니다.

shouldComponentUpdate

```
shouldComponentUpdate(nextProps, nextState){
  console.log("shouldComponentUpdate: " + JSON.stringify(nextProps) + " "+JSON.stringify(nextState));
```

```
  return nextProps.id !== this.props.id;
}
```

props 혹은 state가 변경되었을 때, 리렌더링을 할지 말지 정하는 메서드입니다. return 값이 true이면 리렌더링이 진행됩니다.

componentWillUpdate

```
componentWillUpdate(nextProps, nextState){
  console.log("componentWillUpdate: " + JSON.stringify(nextProps) + " " +           JSON.stringify(nextState));
}
```

컴포넌트가 업데이트 되기 전에 실행됩니다.

componentDidUpdate

```
componentDidUpdate(prevProps, prevState){
   console.log("componentDidUpdate: " + JSON.stringify(prevProps) + " " +             JSON.stringify(prevState));
}
```

컴포넌트가 리렌더링을 마친 후 실행됩니다.

```
componentWillUnmount(){
   console.log("componentWillUnmount");
}
```

컴포넌트가 DOM 에서 사라진 후 실행됩니다.

이렇게 중요한 함수들이 많은데...

위에서 말했다시피 클래스형 컴포넌트는 요새 거의 사장되었다고 봐도 무방합니다. 물론 기존에 클래스형으로 개발된 프로젝트의 경우 유지보수를 하는 과정에서 계속 클래스형 컴포넌트를 사용할 수 있겠지만 대부분의 프로젝트는 함수형으로 만들어집니다.

그리고 그렇게 된 가장 큰 이유는 React Hooks의 등장입니다. Hooks는 간단하게 공통 함수 또는 util 함수라고 봐도 무방합니다. hooks는 React 16.8 버전에 새로 추가된 기능으로 클래스 컴포넌트에서는 사용할 수 없습니다.

가장 유명한 Hook으로는 useState, useEffect 가 있습니다. useState의 경우 state 선언 및 활용에 사용할 수 있으며, useEffect는 componentDidMount, componentDidUpdate, componentWillUnmount가 합쳐진 것으로 생각할 수 있습니다.

예시

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

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useState의 경우 아래와 같이 사용합니다.

> const [변수명, 변수를 바꿔줄 함수] = useState(초기화변수)

JavaScript는 기본적으로 불변성을 유지해야 하기 때문에 변수 자체를 바꾸는 것은 추천드리지 않습니다. 하지만 리액트에는 state 라는 컴포넌트 내부에서 변경이 가능한 변수를 선언할 수 있고, 이를 위해 useState Hook이 등장했습니다.

useEffect의 경우 아래와 같이 사용됩니다.

```
useEffect(() => {
  doigndoingdoingdoing
}, [변수]}
```

useEffect(함수, []) 이런 느낌으로, 대괄호 안에는 함수 안에서 사용한 변수를 모두 넣어줘야 합니다. 넣어주지 않으면 에러가 발생하며, 넣어주는 순간 해당 변수가 업데이트 되면 useEffect 내의 함수를 다시 실행합니다.