

MORE ON REACTIS

ESTR2106 2022-23 Term 1

Building Web Applications

Dr. Chuck-jee Chau chuckjee@cse.cuhk.edu.hk

OUTLINE

- Functional vs. class components
- States in functional components
- Lifecycle methods and useEffect()
- Typescript with React

TWO KINDS OF REACT COMPONENTS

Class components

- Class is supported since ES2015
 - Before that, long JS code is required
- Props are found in this.props
- Events are class methods called with this
- States are read with this.state and changed with setState()

Functional components

- Functions have been around for long with easy syntax
 - More intuitive with arrow functions
- Props are found in **props**
- Events are simple functions inside the component
- State support is new, done using useState()

See: https://reactjs.org/docs/state-and-lifecycle.html

TWO KINDS OF REACT COMPONENTS

```
const {useState} = React;
                                                                                                  https://codepen.io/chuckjee/pen/YzEEa
// import {useState} from 'react';
 class Home extends React.Component {
                                                                                                      Class component
   constructor(props) {
     super(props);
    this.state = {time: "morning"};
   handleMouseOver = () => this.setState({time: this.state.time=="morning"?"evening":"morning"});
   render() {
    return <h1 onMouseOver={this.handleMouseOver}>This is a {this.props.name} home. Good {this.state.time}!</h1>;
                                                                                                      Functional component
function Home2(props) {
   const [time, setTime] = useState("morning");
   const handleMouseOver = () => setTime(time=="morning"?"evening":"morning");
   return <h1 onMouseOver={handleMouseOver}>This is a {props.name} home! Good {time}...</h1>;
                                                                                                      Functional component
const EasyHome = (props) => <h1>This is an {props.name} short function component!</h1>;
                                                                                                      as arrow function
//ReactDOM.render(<Home name="nice" />, document.querySelector("#app"));
//ReactDOM.render(<Home2 name="nice" />, document.querySelector("#app"));
 //ReactDOM.render(<EasyHome name="nice" />, document.querySelector("#app"));
```

STATES IN FUNCTIONAL COMPONENTS

- In particular, useState() is a function in recent React versions (since v16.8) to set up a hook for a variable and a function
 - A destructured array [var, func] is returned from useState()
 - E.g., *time* to be changed by *setTime()* in this example, and used directly later

```
function Home2(props) {
  const [time, setTime] = useState("morning");
  return <h1 onMouseOver={()=>setTime(time=="morning"?"evening":"morning")}>This
  is a {props.name} home! Good {time}...</h1>;
}
```

LIFECYCLE METHODS

- The lifecycle methods are useful to insert your own functionalities in the component's lifecycle
 - componentWillMount() / componentDidMount()
 - componentWillUpdate() / componentDidUpdate()
 - componentWillReceiveProps()
 - componentWillUnmount()
- See: https://www.newline.co/fullstack-react/30-days-of-react/day-7/

EFFECT HOOK IN FUNCTIONAL COMPONENTS

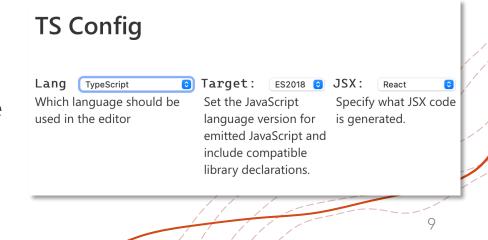
- Together with useState(), useEffect() is another hook in new versions of React
 - To achieve a result of lifecycle in functional components
- useEffect() is called
 - when the component is rendered, and
 - whenever the component is updated
- Further return can be done in useEffect() to mimic componentWillUnmount()
- See: https://reactjs.org/docs/hooks-effect.html

UPDATING THE COMPONENTS

```
const {useState, useEffect} = React;
                                                              render() {
                                                                                     https://codepen.io/chuckjee/pen/PoOE
                                                                  return <h1>Counting down:
class App extends React.Component {
                                                                         {this.state.count}</h1>;
  constructor() {
                                   Class component
    super();
    this.state = {count: 10};
                                                                                        Functional component
                                                            function App2 () {
  componentDidMount() {
                                                              const [count, setCount] = useState(10);
    console.log("Mounted...");
                                                              useEffect(() => {
    setTimeout(()=>this.setState(
                                                                console.log("Updating with count="+count);
      {count:this.state.count-1}), 1000);
                                                                if (count>0)
                                                                  setInterval(()=>setCount(count-1), 1000);
  componentDidUpdate() {
                                                              });
    console.log("Updated with count="
                                                              return <h1>Counting down: {count}</h1>;
                 +this.state.count);
    if (this.state.count > 0)
      setTimeout(()=>this.setState(
                                                            //ReactDOM.render(<App/>,
        {count:this.state.count-1}), 1000);
                                                            document.querySelector("#app"));
                                                            //ReactDOM.render(<App2/>,
                                                            document.guerySelector("#app"));
```

TYPESCRIPT WITH REACT

- It is possible to further enforce type support in React
- TypeScript can now handle JSX well
 - See: https://www.typescriptlang.org/docs/handbook/jsx.html
- Usually, TypeScript support can easily be added with toolchains
 - E.g., npx create-react-app my-app --template typescript
- To test out, simply set under TS Config in the TypeScript Playground
 - Choose "React" for JSX to generate JS code
 - Choose an appropriate "Target"

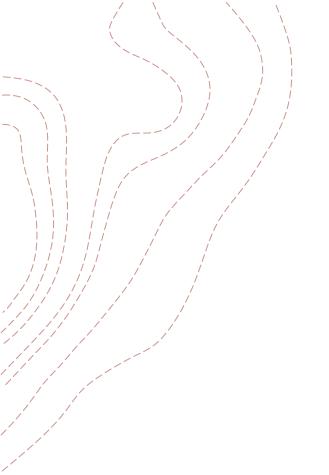


REACT COMPONENTS WITH TYPESCRIPT

```
import React from 'react';
                                                                                                 Click here for TypeScript playground
import {useState} from 'react';
 type MyProps = { name: string; }
type MyState = { time: string; }
                                                                                                               Class component
 class Home extends React.Component<MyProps, MyState> {
   constructor(props :MyProps) {
     super(props);
    this.state = {time: "morning"};
   handleMouseOver = () => this.setState({time: this.state.time=="morning"?"evening":"morning"});
   render() {
     return <h1 onMouseOver={this.handleMouseOver}>This is a {this.props.name} home. Good {this.state.time}!</h1>;
function Home2 (props:MyProps) :JSX.Element {
                                                                                                        Functional component
   const [time, setTime] = React.useState<string>("morning");
   return <h1 onMouseOver={()=>setTime(time=="morning"?"evening":"morning")}>This is a {props.name} home! Good {time}...</h1>;
 const EasyHome: React.FC (MyProps) = (props: MyProps) => <h1>This is an {props.name} short function component!</h1>; Arrow function
```

COMBINING TOOLSETS FOR GOOD WORK

- JSX is optional
 - It helps you to write simpler syntax for elements
- TypeScript is optional
 - It enables better type checks during the coding stage
- React+TypeScript Cheatsheets
 - https://github.com/typescript-cheatsheets/react#reacttypescript-cheatsheets
- Different development communities have different views
- In your own development, find the best way for good work!



READ FURTHER...

ReactJS: Introducing Hooks

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