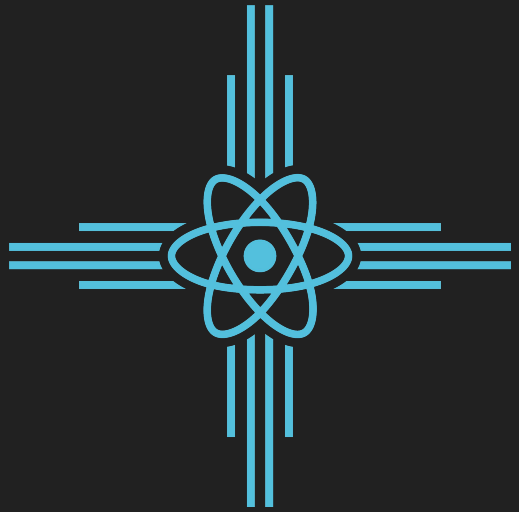


505 React

Meetup 5

Common React Patterns

- Functional components
- Class components
- Higher order components
- Conditional rendering
- Spreading props
- Lifecycle methods

Functional vs. Class Components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Functional vs. Class Components

When should I use a functional component and when should I use a class component?

- Use a class component when you need to setState within the component
- Use a class component when you need to utilize any of React’s lifecycle methods, i.e. componentDidMount, componentWillUnmount, etc.

Functional vs. Class Components

Well then what's the benefit of ever using functional components in the first place??

- Functional component are easier to read and test because they are plain JavaScript functions without state or lifecycle-hooks.
- You end up with less code.
- They help you to use best practices. It will get easier to separate container and presentational components.
- Debate: The React team mentioned that there may be a performance boost for functional component in future React versions.

Activity! Refactor a functional component into a class component

- Fork this Repl.it: https://repl.it/@SamanthaAndrew1/react-meetup-5
- We will start refactoring together, but then work in pairs to execute the instructions found in the index.js file:
  1. Refactor the code from index.js to make Clock an ES6 class
  2. Replace this.props.date with this.state.date in the render() method
  3. Add a class constructor that assigns the initial this.state
  4. Next, we'll make the Clock set up its own timer and update itself every second.

Higher Order Components: An advanced React concept

- A higher-order component (HOC) in React is a pattern used to share common functionality between components without repeating code.
- A HOC function takes a component as an argument and returns a component. It transforms a component into another component and adds additional data or functionality.

```
1  import React from 'react';
2
3  const higherOrderComponent = (WrappedComponent) => {
4    class HOC extends React.Component {
5      render() {
6        return <WrappedComponent />;
7      }
8    }
9
10   return HOC;
11 };
```

```jsx
import React from 'react';

const withSecretToLife = (WrappedComponent) => {
  class HOC extends React.Component {
    render() {
      return (
        <WrappedComponent
          {...this.props}
          secretToLife={42}
        />
      );
    }
  }

  return HOC;
};

export default withSecretToLife;
```

```jsx
import React from 'react';
import withSecretToLife from 'components/withSecretToLife';

const DisplayTheSecret = props => (
  <div>
    The secret to life is {props.secretToLife}.
  </div>
);

const WrappedComponent = withSecretToLife(DisplayTheSecret);

export default WrappedComponent;
```

Higher Order Components

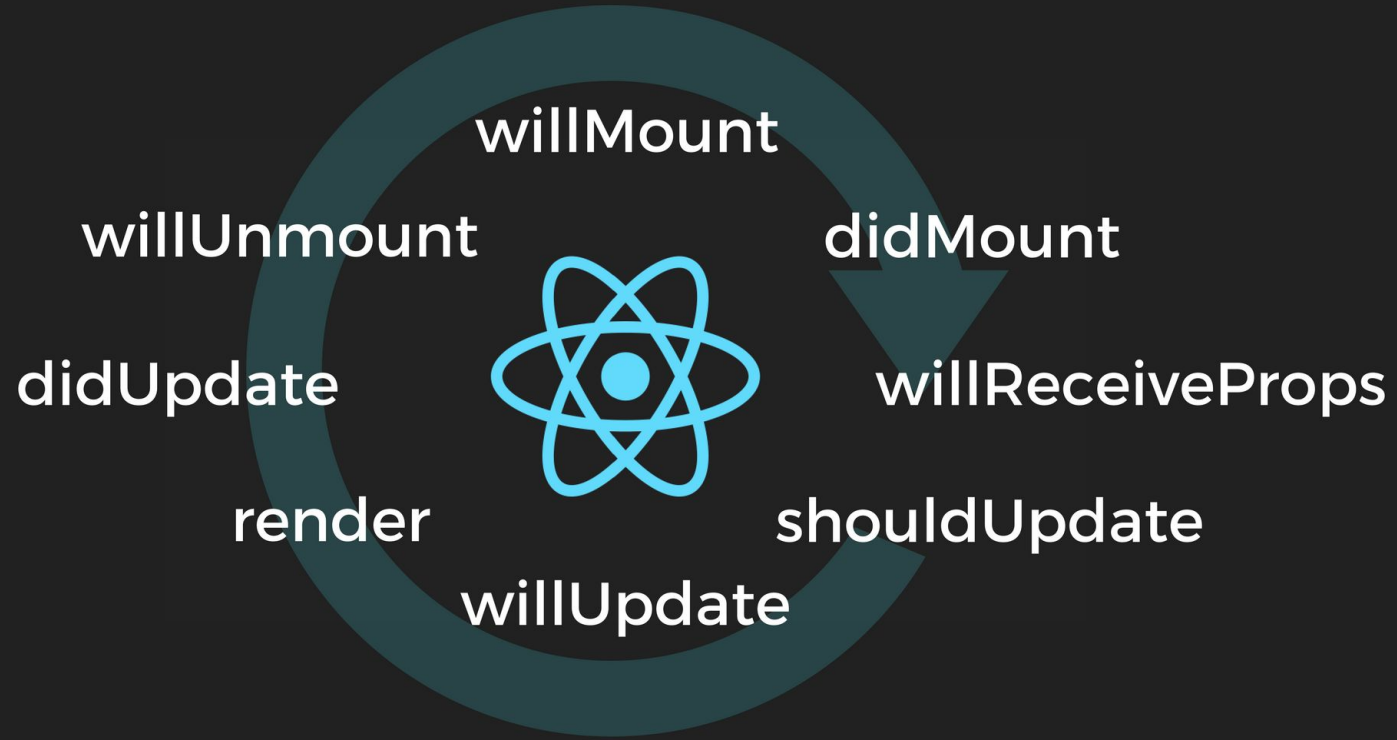
HOC are common in third party React libraries:

- connect from Redux
- withRouter from React Router
- createFragmentContainer from Relay

HOC are pure functions. They do not have any side effects.

Thanks to Trey Huffine of GitConnected.com for writing this blog article that helped me understand HOC. His code was used through this section of the presentation.

React Lifecycle Methods



React Lifecycle Methods

componentWillMount - before anything renders

componentDidMount - best used for calling AJAX

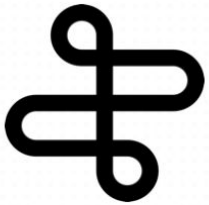
componentWillReceiveProps - gives you access to nextProps and current props

shouldComponentUpdate - you can stop or force re-renders

componentWillUpdate - used in connection with shouldComponentUpdate

componentDidUpdate - you can trigger functions based on change in state or props

componentWillUnmount - component dies



Byteconf JavaScript 2019

March 22 - 23, 2019

- Free!
- Streamed on YouTube and Twitch
- Learn more at byteconf.com