React Advanced - Diving Deeper

coding academy

Props Validation

When component accept props from parent, it is possible to set validations and default values

```
MyCmp.propTypes = {
    name: PropTypes.string.isRequired,
    count: PropTypes.number
}
MyCmp.defaultProps = {
    count: 0
}
```

coding academy

Higher-Order Component

a Higher-Order Component (HOC) is a function that takes a component and returns a new component.

We use HOCs to achieve component logic reuse.

Familiar examples:

- The Redux's connect() function
- The withRouter() function

The core idea is:

```
const EnhancedComponent = enhance(WrappedComponent)
```

Higher-Order Component

Here is a sample implementation for the enhance function:

```
function enhance(WrappedComponent) {
    return class extends React.Component {
      render() {
        const extraProp = 'This is an injected prop!'
        return (
          <div className="Wrapper">
            <WrappedComponent
              {...this.props}
              extraProp={extraProp}
            />
          </div>
        )
      }
    }
  }
```

Props Validation

We can also add custom validation:

```
const Title = (props) => <h1>Title: {props.txt}</h1>

Title.propTypes = {
    txt(props, propName, component){
        if(!(propName in props)){
            return new Error(`missing ${propName}`)
        }
        if(props[propName].length < 6){
          return new Error(`${propName} was too short`)
        }
    }
}
```

coding academy

```
class MyComponent extends React.Component {
    render() {
      // ... do things with the props
    }
  }

  MyComponent.propTypes = {
    optionalNumber: PropTypes.number,
    optionalArray: PropTypes.array,
    optionalFunc: PropTypes.func,
    optionalObject: PropTypes.object,
    optionalEnum: PropTypes.oneOf(['News', 'Photos']),
    optionalUnion: PropTypes.oneOfType([
      PropTypes.string,
      PropTypes.number
    ]),
    optionalArrayOf: PropTypes.arrayOf(PropTypes.number),
    requiredAny: PropTypes.any.isRequired,
    requiredFunc: PropTypes.func.isRequired,

    customProp: function(props, propName, componentName) {
      if (!/matchme/.test(props[propName])) {
        return new Error(
          'Invalid prop `' + propName + '` supplied to' +
          ' `' + componentName + '`. Validation failed.'
        );
      }
    }
  };
```

Props Validation

Read about It here

coding academy

Having Children

```
function FancyBox(props) {
    return <div className="fancy-box">
            {props.children}
    </div>
}
```

Having Children

Some components don't know their children ahead of time. This is especially common for components like <Sidebar> or <Dialog> that represent generic "boxes"

```
function FancyBox(props) {
    return <div className="fancy-box">
        <button style={{ float: 'right' }} onClick={props.onClose}>x</button>
        {props.children}
    </div>
}
```

```
<FancyBox onClose={() => console.log('ok, closing')}>
    <h3>{count.toLocaleString()} Followers</h3>
    <button onClick={this.onTellMeMore}>Tell me More</button>
</FancyBox>
```

coding academy

Having Children

```
function SplitPane(props) {
    return (
      <div className="split-pane">
        <div>
          {props.left}
        </div>
        <div>
          {props.right}
        </div>
      </div>
    );
  }

  function Cmp() {
    return (
      <SplitPane
        left={
          <Contacts />
        }
        right={
          <Projects />
        } />
    )
```

Sometimes we might need multiple “holes” in a component.

In such cases we will need to do it in another way (instead of using props.children)

Error Boundaries

Error Boundaries

- React components that catch JavaScript errors in their child component tree
- They will log the error, and display a fallback UI instead of the component tree that crashed.

```
componentDidCatch(error, errorInfo) {
    // Catch errors in children and re-render with error message
    this.setState({
        error,
        errorInfo
    })
}
```

Error Boundaries

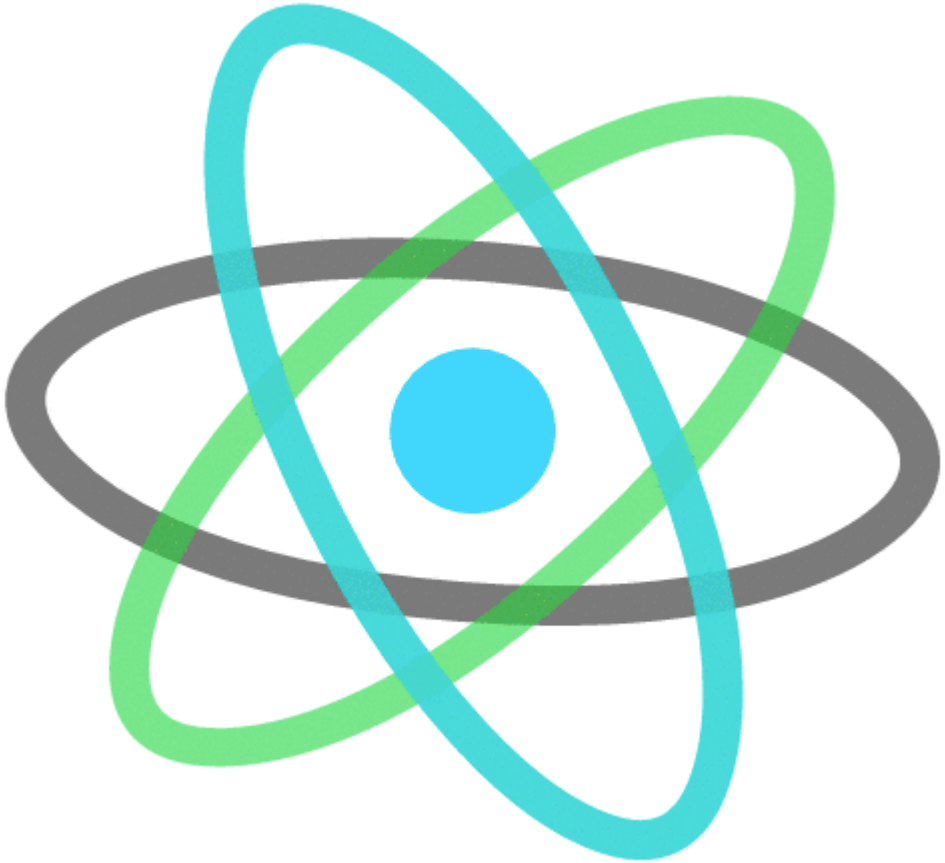
- Error boundaries only catch errors during rendering, and lifecycle methods
- They will not catch any error happened in event handler or any async function, those are still handled with normal try-catch
- Note: in development the error is still presented on screen and you need to ESC to see the fallback UI

```jsx
class ErrorBoundary extends React.Component {
    state = { error: null, errorInfo: null };

    componentDidCatch(error, errorInfo) {
        // Catch errors in children and re-render with error message
        this.setState({
            error,
            errorInfo
        })
        // TODO: Log error to an error reporting service
        // logger.report(error)
    }
    render() {
        if (this.state.errorInfo) {
            return (
                <div>
                    <h2>Something went wrong.</h2>
                    <details style={{ whiteSpace: 'pre-wrap' }}>
                        {this.state.error && this.state.error.toString()}
                        <br />
                        {this.state.errorInfo.componentStack}
                    </details>
                </div>
            );
        }
        // Normally, just render children
        return this.props.children;
    }
}
```

Animations

Animations are great



Animations

- There are many ways to animate stuff
- Simplest way – use CSS transitions in your CSS file
- Also, React suggest using CSSTransitionGroup

Animations

- The CSSTransitionGroup is an add-on component for implementing CSS animations.
  - Still not 100% stable and being extracted to community

```
<CSSTransitionGroup transitionName="example" transitionEnterTimeout={500}
    transitionLeaveTimeout={300}>
    {projects}
</CSSTransitionGroup>
```

- e.g. - When a new item is added to `projects`,
  it will assign the *example-enter* CSS class and the:
  example-enter-active CSS class added in the next tick.
  (Based on the transitionName prop)

Animations

We then use these classes to trigger a CSS animation or transition:

```
.example-enter {
  opacity: 0.01;
}

.example-enter.example-enter-active {
  opacity: 1;
  transition: opacity 500ms ease-in;
}

.example-leave {
  opacity: 1;
}

.example-leave.example-leave-active {
  opacity: 0.01;
  transition: opacity 300ms ease-in;
}
```

React Optimizations

- React shouldComponentUpdate is a performance optimization method, and it may tell React to avoid re-rendering a component, even if state or prop values may have changed.

- Class Component? Use PureComponent
- Functional Components? React Memo

coding academy

React Master!

coding academy