Higher Order Component

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Higher Order Component (HOC)
  - React memo
- Decorator design pattern
- Bundling
- Code Splitting and loading components lazily
  - Dynamic import
  - React.lazy and React.Suspense

Higher Order Component

- A higher-order component is a function that takes a component and returns a new component.
- You put some logics inside HOC. That adds features to your component. Hence your base component is enhanced.

- const IronMan = withSuit(TonyStark)

```
const withSomething = (OriginalComponent) => {
  // ... create new component from old one and update
  return EnhancedComponent
}
```

React.memo

- React.memo is a higher order component.
- React.memo is for a performance boost in some cases by memoizing the result. This means that React will skip rendering the component, and reuse the last rendered result. Similar to PureComponent but only for functional components.
- You can’t extend to PureComponent in functional component. You can achieve it using memo for functional components. But React.memo only checks for **prop** changes. That makes sense as functional components are stateless.
- Wrap your component with React.memo()

When to Use memo

- Pure component
- Renders often
- Medium to big sized component
- Re-renders with the same props

```jsx
export function Movie({ title, releaseDate }) {
  return (
    <div>
      <div>Movie title: {title}</div>
      <div>Release date: {releaseDate}</div>
    </div>
  );
}

export const MemoizedMovie = React.memo(Movie);
```

```jsx
// First render - MemoizedMovie IS INVOKED.
<MemoizedMovie title="Heat" releaseDate="December 15, 1995" />

// Second render - MemoizedMovie IS NOT INVOKED.
<MemoizedMovie title="Heat" releaseDate="December 15, 1995" />
```

withAuthenticator from AWS Amplify

- AWS Amplify is a tool to build full-stack web and mobile applications.
- The simplest way to add authentication flows into the AWS Amplify app is to use the withAuthenticator HOC.
- Only signed in users can see the main component such as App.js.
- If user is not signed in, then the default sign in and registration pages provided by AWS Amplify are shown.

Decorator design pattern

- A decorator design pattern that allows **behavior to be added to an individual object**, dynamically.
- Decorator design pattern is popular in the software development world. Many libraries and frameworks takes advantages of it, regardless of the language. You will learn more in the Angular course.
- Annotations are also used in JS and Java to achieve decorator design pattern.

Logical and presentational components

- There is an advanced technique that developers write the code in a way that there are 2 types of components, logical and presentational.
  - Logical – Business logic components (HOC) that inject the computed data after processing to the presentational components via props.
  - Presentational – Components that return JSX and show data on the page.

Bundling

Most React apps will have their files “bundled” using tools like Webpack, Babel. Bundling is the process of following imported files and **merging them into a single file**: a “**bundle**”. This bundle can then be included on a webpage to load an entire app at once.

App.js and Math.js

```
// app.js
import { add } from './math.js';

console.log(add(16, 26)); // 42
```

```
// math.js
export function add(a, b) {
  return a + b;
}
```

After bundling

```
function add(a, b) {
  return a + b;
}
console.log(add(16, 26)); // 42
```

Transpilers and Polyfills

- The bundling tools such as Babel, Webpacks are also transpilers and polyfills.
- **Transpilers** convert modern JS syntax into the old one so that your code is supported by old browsers.
- What if the modern functions don’t even exist in old browsers? That is what **Polyfills** help you with. A script that updates/adds new functions is called “polyfill”. It “fills in” the gap and adds missing implementations.
- Learn more: Polyfills and transpilers on javascript.info

Code-Splitting

Bundling is great, but as your app grows, your bundle will grow too. Especially if you are including large **third-party libraries**. You need to keep an eye on the code you are including in your bundle so that you don't accidentally make it **so large** that your app **takes a long time to load**.

That is when the code-splitting comes into a picture and improves your app performance by reading modules asynchronously.

Code-splitting your app can help you "**lazy-load**" just the things that are currently needed by the user, which can dramatically improve the performance of your app.

Learn more: Code Splitting

Code-Splitting in React

Deciding where in your app to introduce code splitting can be a bit tricky. You want to make sure you choose places that will split bundles evenly, but won’t disrupt the user experience.

There are 2 ways to split code:

1. The dynamic import
2. React.lazy and React.Suspense

The best way to split your code is the **Route-based code splitting**. For now, think of the route as a web page e.g., login, signup, dashboard.

Dynamic import

Before

```
import { add } from './math';

console.log(add(16, 26));
```

After:

```
import("./math").then(math => {
  console.log(math.add(16, 26));
});
```

Dynamic import is supported by the Webpack. If you’re using Create React App, this is already configured for you and you can start using it immediately. When using Babel, you’ll need to make sure that Babel can parse the dynamic import syntax. For that you will need @babel/plugin-syntax-dynamic-import

React.lazy

React.lazy() lets you define a component that is loaded **dynamically**. This helps reduce the bundle size to delay loading components that aren’t used during the initial render.

```
// This component is loaded dynamically
const SomeComponent = React.lazy(() => import('./SomeComponent'));
```

React.Suspense

Lazy components require React.Suspense. It lets you specify the loading indicator in case some components in the tree below it are not yet ready to render.

```
// This component is loaded dynamically
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    // Displays <Spinner> until OtherComponent loads
    <React.Suspense fallback={<Spinner />}>
      <div>
        <OtherComponent />
      </div>
    </React.Suspense>
  );
}
```