React Patterns

tsevdos.me / @tsevdos

Agenda

- stateful and stateless components
- render props
- custom hooks
- compound components

Rules

Feel free to interrupt me for:

- questions
- relevant comments

Stateful and stateless components

- presentational and container components
- smart and dumb component
- seperation of concerns
- easier to reuse
- better structure
- designer friendly

Stateful components

- are concerned with how things work
- provide the data and behavior to stateless or other stateful components
- call actions and provide these as callbacks to the stateless components
- connect with stores (redux, mobx, etc.)

Stateless components

- simple
- are concerned with how things look (UI)
- event handling
- JSX
- performance

Stateful and stateless components

Examples.

Stateful and stateless components exercise

(./src/examples/SSC/exercise/UserCard.js)

1. Create a UserCardContainer stateful component that will keep all the state and logic
2. Extract the UI to a stateless UserCard component

Render props

a “render prop” is simply a prop that takes a function which returns elements that will be used in render()

Render props

solves the problems a HoC solves

Render props

```
const Component = (props) => props.renderProp();
// or
const Component = (props) => props.children();
```

Render props

Examples.

Render props exercise

(./src/examples/RP/exercise/UserCard.js)

1. create a "User" component that will provide the user data using a render prop
2. keep the same functionality
3. make the UserCard component stateless and use the User component to render the data

Render props > HoCs

- simpler (not ES6 classes / hight-order functions)
- clarity (we know exactly from which component our props came)
- no need to create a new component
- no need to rename (avoid naming collisions)

Render props considerations

- callback hell

Custom hooks

Custom hooks are a mechanism to reuse stateful logic (such as setting up a subscription and remembering the current value), but every time you use a custom Hook, all state and effects inside of it are fully isolated.

Custom hooks

- reuse stateful logic between components
- simplify components (easy to understand)
- share logic between different components and lifecycle methods
- easier and more flexible pattern from render props and higher-order components

Custom hooks rules

- only call hooks from React function components
- only call hooks at the top level
- don’t call hooks inside loops, conditions, or nested functions
- custom hooks start with useSomething PascalCase function

Custom hooks

Examples.

Custom hooks

(./src/examples/hooks/exercise/UserCard.js)

1. create a custom "useUser" custom hook
2. keep the same functionality
3. use the custom hook to get the user data on the UserCard component

Custom hooks > Render props

- better abstraction
- easier sharing
- encapsulation
- cleaner logic for side effects

Compound components

Compound components is a pattern where components are used together such that they share an implicit state that lets them communicate with each other in the background. A compound component is composed of a subset of child components that all work in tandem to produce some functionality

Compound components

> Think of compound components like the <select> and <option> elements in HTML. Apart they don't do too much, but together they allow you to create the complete experience. — Kent C. Dodds

HTML elements

```
<select id="programming-languages">
  <option value="javascript">JavaScript</option>
  <option value="typescript">TypeScript</option>
  <option value="rust">Rust</option>
  <option value="go" selected>Go</option>
</select>
```

Compound components

- React.Children.map
- React.cloneElement()
- Context API

Compound components benefits

- flexible
- cleaner API
- customizable
- reusable

Compound components

Examples.

Recap

- stateful and stateless components
- render props
- custom hooks
- compound components

That's all folks

Questions / Discussions?