WMNWA 2210

WEB APP DEVELOPMENT

React 4 | Managing State

SAE
CREATIVE
MEDIA
EDUCATION

ΠΕΡΙΕΧΟΜΕΝΑ

Περιεχόμενα

- Hooks
  - **useState**
  - **useReducer**
- Controlled Components
  - **useRef**

HOOKS

Hooks

Stateful Components

Στο παρελθόν, τα **Function Components** χρησιμοποιούνταν για απλά στοιχεία, που δεν είχαν τρόπο να "διατηρούν" την κατάστασή στην οποία βρίσκονται (*stateless*).

Με την εμφάνιση των **React Hooks** στην έκδοση *16.8*, αυτό άλλαξε και πλέον τα στοιχεία αυτά μπορούν να έχουν τη λειτουργικότητα που έχουν και και τα **Class Component**.

Hooks

Class Component issues

Λόγοι που οδήγησαν σε αυτή την αλλαγή:

- *It's hard to reuse stateful logic between components*
- *Complex components become hard to understand*
- *Classes confuse both people and machines*

Hooks

Hook advantages

Τι προσφέρουν;

- *Hooks allow you to reuse stateful logic without changing your component hierarchy.*
- *Hooks let you split one component into smaller functions based on what pieces are related (such as setting up a subscription or fetching data).*
- *Hooks let you use state without classes.*

Hooks

useState hook

Η μέθοδος **useState** δέχεται ως όρισμα μια αρχική τιμή/κατάσταση και επιστρέφει έναν πίνακα με δύο στοιχεία (θυμίζει **tuple**).

Το πρώτο στοιχείο είναι η τρέχουσα τιμή/κατάσταση και το δεύτερο μια συνάρτηση που θα πρέπει να χρησιμοποιηθεί για να αλλάζει η τιμή αυτή.

```
const [state, setState] = useState(initialState);
```

Και εδώ, η τεχνική του *destructuring* είναι πολύ συνηθισμένη.

Hooks

useState hook

- Το πρώτο στοιχείο (ίδιου τύπου με την αρχική τιμή), θα χρησιμοποιηθεί μέσα στην **JSX** για να αναπαραστήσει τη συγκεκριμένη τιμή.
- Το δεύτερο στοιχείο (συνάρτηση), θα χρησιμοποιηθεί για την αλλαγή της τιμής, καθώς "απαγορεύεται" (*immutability*) να επέμβουμε απευθείας σε αυτή.
  - Αν δοθεί, ως όρισμα, μια τιμή, αυτή η τιμή θα αντικαταστήσει την παλιά.
  - Αν η νέα τιμή εξαρτάται από την προηγούμενη, τότε πρέπει να δοθεί, ως όρισμα, μια συνάρτηση που να περιγράφει την αλλαγή, βάσει της προηγούμενης τιμής.

Hooks

Παράδειγμα #1

```
const Counter = ({ initial }) => {
  const [count, setCount] = useState(initial);
  return (
    <>
      Count: {count}
      <button onClick={() => setCount(initial)}>Reset</button>
      <button onClick={() => setCount(prev => prev - 1)}>-</button>
      <button onClick={() => setCount(prev => prev + 1)}>+</button>
    </>
  );
}
```

Hooks

Παράδειγμα #2

```
const ClickCounter = () => {
  const [count, setCount] = React.useState(0);

  const incrementCounter = () => { setCount((prev) => prev + 1); };
  const resetCounter = () => { setCount(0); };

  return (
    <div>
      <button onClick={incrementCounter}>Click me!</button>
      <button onClick={resetCounter}>Reset</button>
      <span>Clicks: {count}</span>
    </div>
  );
};

const App = () => (
  <ClickCounter />
);

ReactDOM.createRoot(document.getElementById('root')).render(<App />);
```

Hooks

useReducer hook

Η μέθοδος **useReducer** δέχεται ως όρισμα μια μία *reducer function* και μια αρχική τιμή/κατάσταση και επιστρέφει έναν πίνακα με δύο στοιχεία (θυμίζει **tuple**).

Το πρώτο στοιχείο είναι η τρέχουσα τιμή/κατάσταση και το δεύτερο μια συνάρτηση που θα πρέπει να χρησιμοποιηθεί για να αλλάζει η τιμή αυτή.

```
const [state, dispatch] = useReducer(reducer, initialState);
```

Hooks

useReducer hook

- Πρόκειται για εναλλακτική προσέγγιση, αντί της **useState**.
- Εκτός από την αρχική τιμή/κατάσταση δέχεται και μία *reducer function*, πάνω στην οποία θα βασιστεί η *function* που επιστρέφεται.
- Είναι, συνήθως, προτιμότερη όταν:
  - πρέπει να διαχειριστούμε πολύπλοκη λογική (*complex state logic*).
  - κάθε αλλαγή βασίζεται στην προηγούμενη κατάσταση.

Hooks

reducer function

Η *reducer function* έχει συνήθως μια μορφή όπως η παρακάτω, χωρίς αυτό να είναι δεσμευτικό.

```
const reducer = (state, action) => {
  switch (action.type) {
    case 'ACTION_TYPE_1':
      return ...;
    case 'ACTION_TYPE_2':
      return ...;
    ...
    default:
      throw new Error(`Invalid action type: ${action.type}`);
  }
};
```

Hooks

The *reducer* pattern

Μια απλοποιημένη υλοποίηση της **useReducer** είναι η παρακάτω.

Στην ουσία δεν προσφέρει κάτι νέο, απλά βοηθάει στην εφαρμογή μιας διαδομένης τεχνικής.

```
function useReducer(reducer, initialState) {
  const [state, setState] = useState(initialState);

  function dispatch(action) {
    const nextState = reducer(state, action);
    setState(nextState);
  }

  return [state, dispatch];
}
```

Hooks

Παράδειγμα #1

```
const reducer = (state, action) => {
  switch (action.type) {
    case 'SET':
      return action.payload;
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
    default:
      throw new Error(`Invalid action type: ${action.type}`);
  }
};

const Counter = ({ initial }) => {
  const [count, dispatch] = useReducer(reducer, initial);
  return (
    <>
      Count: {count}
      <button onClick={() => dispatch({ action: 'SET', payload: initial })}>Reset</button>
      <button onClick={() => dispatch({ action: 'DECREMENT' })}>-</button>
      <button onClick={() => dispatch({ action: 'INCREMENT' })}>+</button>
    </>
  );
}
```

Hooks

Παράδειγμα #2

```
const reducer = (state, action) => {
  switch (action.type) {
    case 'SET':
      return action.payload;
    case 'INCREMENT':
      return state + 1;
    default:
      throw new Error(`Invalid action type: ${action.type}`);
  }
};

const ClickCounter = () => {
  const [count, dispatch] = useReducer(reducer, 0);
  const incrementCounter = () => { dispatch({ action: 'INCREMENT' }) };
  const resetCounter = () => { dispatch({ action: 'SET', payload: 0 }) };

  return (
    <div>
      <button onClick={incrementCounter}>Click me!</button>
      <button onClick={resetCounter}>Reset</button>
      <span>Clicks: {count}</span>
    </div>
  );
};
```

CONTROLLED COMPONENTS

Controlled Components

Single source of truth

Τα στοιχεία σε μια **HTML** φόρμα κρατούν εσωτερικά την κατάσταση (*state*) στην οποία βρίσκονται.

Στη **React** ιδανικά θέλουμε να υπάρχει ένα σημείο αναφοράς για την κατάσταση της φόρμας και, καθώς δεν υπάρχει η δυνατότητα για *2-way binding*, φροντίσουμε τα στοιχεία της φόρμας να ενημερώνουν το *state* του **component** σε κάθε αλλαγή.

Ένα στοιχείο του οποίου η τιμή ελέγχεται από τη **React** ονομάζεται *Controlled Component*.

Αν σε ένα τέτοιο στοιχείο δεν ορίσουμε *event handler*, για να διαχειριστεί τις αλλαγές, τότε δεν θα μπορεί ο χρήστης να αλλάξει την τιμή του.

Controlled Components

Παράδειγμα #1

```
const NameForm = () => {
  const [name, setName] = useState('');

  const handleChange = (event) => {
    setName(event.currentTarget.value);
  }

  const handleSubmit = (event) => {
    console.log(`A name was submitted: ${name}`);
    event.preventDefault();
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
};
```

Controlled Components

Παράδειγμα #2

```
const FavoriteFlavorForm = () => {
  const [favoriteFlavor, setFavoriteFlavor] = useState('coconut');

  const handleChange = (event) => {
    setFavoriteFlavor(event.currentTarget.value);
  }

  const handleSubmit = (event) => {
    console.log(`Your favorite flavor is: ${favoriteFlavor}`);
    event.preventDefault();
  }

  return (
    <form onSubmit={handleSubmit}>
      <label> Pick your favorite flavor:
        <select value={value} onChange={handleChange}>
          <option value="lime">Lime</option>
          <option value="coconut">Coconut</option>
          <option value="mango">Mango</option>
        </select>
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
};
```

Uncontrolled Components

useRef

Τα **Controlled Components** είναι ο ενδεδειγμένος τρόπος για να υλοποιούμε στοιχεία σε φόρμες.

Παρόλα αυτά, υπάρχει η δυνατότητα για **Uncontrolled Components**, όπου, αντί να ενημερώνεται το *state* σε κάθε αλλαγή του στοιχείου, παίρνουμε την τιμή του όταν τη χρειαζόμαστε.

Για να αναφερθούμε σε ένα στοιχείο, είτε για να πάρουμε την τιμή του, είτε για οποιοδήποτε άλλο λόγο, μπορούμε να κάνουμε χρήση του **useRef hook**.

Uncontrolled Components

Παράδειγμα

```
const NameForm = () => {
  const [name, setName] = useState('');

  const inputEl = useRef(null);

  const handleSubmit = (event) => {
    console.log(`A name was submitted: ${inputEl.current.value}`);
    inputEl.current.value = '';
    inputEl.current.focus();

    event.preventDefault();
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" ref={inputEl} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
};
```

HOMEWORK

Homework

Image Gallery

Τροποποιήστε το *Image Gallery* από το προηγούμενο *homework*, ώστε να κάνει χρήση του **useReducer** στη θέση του **useState**.

Σημείωση: Χρησιμοποιήστε το **starting-point** από τη διάλεξη.

Χρήσιμα links

- Introducing Hooks – React
  https://reactjs.org/docs/hooks-intro.html
- Hooks at a Glance – React
  https://reactjs.org/docs/hooks-overview.html
- React Hooks - React docs
  https://reactjs.org/docs/hooks-state.html
- Hooks API Reference – React
  https://reactjs.org/docs/hooks-reference.html

Extra info

- Building Your Own Hooks – React
  https://reactjs.org/docs/hooks-custom.html
- Rules of Hooks – React
  https://reactjs.org/docs/hooks-rules.html
- Forms - Controlled Components – React
  https://reactjs.org/docs/forms.html#controlled-comp…
- Uncontrolled Components – React
  https://reactjs.org/docs/uncontrolled-components.html
- Hooks FAQ – React
  https://reactjs.org/docs/hooks-faq.html

THANK YOU!