How to manage React State with Arrays

MAY 17, 2020 BY ROBIN WIERUCH - EDIT THIS POST

After learning how to pass props in React, the next thing you will learn is often state in React. Managing JavaScript primitives such as strings, booleans, and integers in React State are the basics for state management in React. But what about more complex data structures such as JavaScript arrays? There are many questions popping up for React beginners on how to manage arrays in React state. Often the answer is to grab a fundamental JavaScript function that does the job for you. It's not always about using React for the task at hand.

This tutorial walks you through the most common scenarios for managing arrays in React state. For each I want to show you a array example in React state, such as how to push an item to an array or how to update an item in an array, when React state is used to store it.

The following tutorial shows you array manipulation in React with class components. If you are interested in seeing this with function components and React Hooks, check out the following guides:

- Add an Item to a List in React

- Update an Item in a List in React
- Remove an Item from a List in React

ARRAYS IN REACT STATE

Before we are going to manipulate a JavaScript array in React state, let's recap state in React shortly. State in React can be initialized in the constructor of a React component and afterward used by accessing it via the React component's class instance with the `this` object.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [1, 2, 3],
    };
  }

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

As mentioned, the initial array state is done in the React component's constructor and afterward used within the `render()` method to display it. Every time the state changes, the render method will run again and display the correct state in your browser. However, the state shouldn't be changed by mutating `this.state` directly. Instead, there exists the `this.setState()` method on a React component to update React's state. A quote from the official React documentation says:

"Never mutate this.state directly, as calling setState() afterwards may replace the mutation you made. Treat this.state as if it were immutable."

The `this.setState()` method on the component instance is used to update the React state

The this.setState() method on the component instance is used to update the React state. It does a shallow merge, meaning that when you update one property in the state (e.g. list), the other properties in the state stay intact.

Now, let's see what kind of state manipulations can be done with arrays in React by extending the previous example for different React state array examples. Note that in this case, the array is only a list of integers. You can substitute the following examples for other arrays of primitives and arrays of objects.

REACT STATE: EMPTY ARRAYS AND INITIAL ARRAY STATE

The following three questions often come along when speaking about initial/empty state in React:

- How to initialize an empty array in React state?
- How to push an empty array in React state?
- How to create an initial array React state?

The first question can be answered with React's component constructor by simply initializing the state as as empty array:

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [],
    };
  }

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>
      </div>
    );
  }
```

```
  }
}

export default App;
```

The `map()` method in the `render()` method, that is used to display the items of the array, does work, because it iterates over an empty array and thus returns no item for it. The iterator function in the map method isn't called. Let's say you would have a null state for your array instead, then you would have to have a fallback for your map method in your render method:

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: null,
    };
  }

  render() {
    return (
      <div>
        <ul>
          {(this.state.list || []).map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

That's only one way of providing a fallback for an array being null. As alternative, you can also use React's conditional rendering for it.

The second question, which asks how to push an empty array in React state, involves manipulating the state with `this.setState()`. Let's say you want to empty the array on a button click. Then you can do it the following way:

```
import React, { Component } from 'react';

class App extends Component {
```

```
class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [1, 2, 3],
    };
  }

  onClearArray = () => {
    this.setState({ list: [] });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>

        <button type="button" onClick={this.onClearArray}>
          Clear Array
        </button>
      </div>
    );
  }
}

export default App;
```

You set an empty array as React state for the component by having a clear method for it. Building on top of this example, the third question, which asks how to create an initial array state, can be answered too. You have already seen how an initial array is set to state in the component's constructor. How would you reinitialize the initial state again, basically a reset of the state, after you have manipulated the state already? You can extract the initial state and then set it again whenever you want.

```
import React, { Component } from 'react';

const list = [1, 2, 3];

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list,
    };
```

```
  }

  onClearArray = () => {
    this.setState({ list: [] });
  };

  onResetArray = () => {
    this.setState({ list });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>

        <button type="button" onClick={this.onClearArray}>
          Clear Array
        </button>

        <button type="button" onClick={this.onResetArray}>
          Reset Array
        </button>
      </div>
    );
  }
}

export default App;
```

The state reset was only applied to the array. But you can apply it to your entire state too by extracting the whole initial state object.

```
import React, { Component } from 'react';

const INITIAL_STATE = {
  list: [1, 2, 3],
};

class App extends Component {
  constructor(props) {
    super(props);

    this.state = INITIAL_STATE;
  }

  onClearArray = () => {
    this.setState({ list: [] });
  };
```

```
  };

  onResetArray = () => {
    this.setState({ ...INITIAL_STATE });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>

        <button type="button" onClick={this.onClearArray}>
          Clear Array
        </button>

        <button type="button" onClick={this.onResetArray}>
          Reset Array
        </button>
      </div>
    );
  }
}

export default App;
```

By using the JavaScript spread operator, you can spread all key value pairs of the initial state object to the initial state of the component. That works with multiple properties in the state object too.

REACT STATE: ADD ITEM TO ARRAY

One of the most common questions is how to add an item to an array in React state. Since you are not allowed to mutate the state directly, you cannot simply push an item to an array.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      value: '',
      list: ['a', 'b', 'c'],
```

```
    };
  }

  onChangeValue = event => {
    this.setState({ value: event.target.value });
  };

  onAddItem = () => {
    // not allowed AND not working
    this.setState(state => {
      const list = state.list.push(state.value);

      return {
        list,
        value: '',
      };
    });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>{item}</li>
          ))}
        </ul>

        <input
          type="text"
          value={this.state.value}
          onChange={this.onChangeValue}
        />
        <button
          type="button"
          onClick={this.onAddItem}
          disabled={!this.state.value}
        >
          Add
        </button>
      </div>
    );
  }
}

export default App;
```

First, it is not allowed to use the array push method, because it mutates the array. It doesn't leave the array intact but changes it. Instead, there should be a new array created which is used to update the state.

Second, assumed for a moment the array push method would be allowed, the `onAddItem`

method still would not work. Take a moment to think about why it wouldn't work this way. In my workshops, I have seen many people stumbling into this problem.

Here comes the answer: The push to array method wouldn't work, because it doesn't return the updated array. Instead, it returns the length of the updated array (which is 4 in this case).

In conclusion, the array push method doesn't work for us. Fortunately there exists a great substitute for it which overcomes both drawbacks of the array push method. It's the array concat method. It creates a new array, leaving the old array intact, but also returning a new array from it.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      value: '',
      list: ['a', 'b', 'c'],
    };
  }

  ...

  onAddItem = () => {
    this.setState(state => {
      const list = state.list.concat(state.value);

      return {
        list,
        value: '',
      };
    });
  };

  ...
}

export default App;
```

After all, when having immutable data structures (or treating them as immutable as for React state), concat is our friend and push our foe when it comes to arrays.

One question remains though: how to add an item at the beginning of an array in React state? Same as before, is hasn't to do with React state, but simply with executing the correct JavaScript functionalities. One way would be to exchange places for the concat method from before (e.g. `const list = [state.value].concat(state.list);`). Another way of

doing it would be using the array spread operator. Let's see how it works as alternative to the array concat method.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      value: '',
      list: ['a', 'b', 'c'],
    };
  }

  ...

  onAddItem = () => {
    this.setState(state => {
      const list = [...state.list, state.value];

      return {
        list,
        value: '',
      };
    });
  };

  ...

}

export default App;
```

While the array from the previous state is spread into a new array, so the previous arrays doesn't get mutated, the new item is added at the end of the array. Now, you can add the item at the start of the array by using `const list = [state.value, ...state.list];` instead.

REACT STATE: UPDATE ITEM IN ARRAY

In this part of the walkthrough, we will go through two cases to update items in an array:

- How to update the entire array in React state?
- How to update a specific item in array in React state?

In both cases, the array map method is our friend. Whereas the array concat is used to add an item to an array, the array map method is useful to update item(s) in an array. It returns a new array too and thus doesn't mutate the previous array. Let's see how we can update an entire array by using the array map method.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [42, 33, 68],
    };
  }

  onUpdateItems = () => {
    this.setState(state => {
      const list = state.list.map(item => item + 1);

      return {
        list,
      };
    });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>The person is {item} years old.</li>
          ))}
        </ul>

        <button type="button" onClick={this.onUpdateItems}>
          Make everyone one year older
        </button>
      </div>
    );
  }
}

export default App;
```

The array map method takes as argument a function. The function is applied on each item of the array and thus has access to each item as argument. In this case, a shorthand arrow function is used to increase the integer in the array by one. Each item in the array is affected by this update once you click the button. As you can see, the array map method works

wonders here. Basically it's a array replace for each item in the array.

How would you implement the same scenario when only updating the integer (age) of a single item (person)? Instead of updating the entire array, you only want to update a specific item in it. Let's see how this goes. First, you can give each item (person) a button to increase its integer (age).

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [42, 33, 68],
    };
  }

  onUpdateItem = i => {
    ...
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map((item, index) => (
            <li key={item}>
              The person is {item} years old.
              <button
                type="button"
                onClick={() => this.onUpdateItem(index)}
              >
                Make me one year older
              </button>
            </li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

In this case, you take the index of the item in the array to identify it later in the `onUpdateItem()` class method. By using a wrapping arrow function in the `onClick` handler, you can sneak in this identifier for being used in the class method. How to update a specific item in the array, no? The simplest way I have found to do it is using the array

specific item in the array now? The simplest way I have found out to do it is using the array map method again.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [42, 33, 68],
    };
  }

  onUpdateItem = i => {
    this.setState(state => {
      const list = state.list.map((item, j) => {
        if (j === i) {
          return item + 1;
        } else {
          return item;
        }
      });

      return {
        list,
      };
    });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map((item, index) => (
            <li key={item}>
              The person is {item} years old.
              <button
                type="button"
                onClick={() => this.onUpdateItem(index)}
              >
                Make me one year older
              </button>
            </li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

By adding conditional logic in the map method's iterating function, you can modify the single item, but leave every other item as before. Basically it's a array replace for a specific item in the array.

As you have seen, the array map method is a perfect fit to change a single item, multiple items or all items in an array. You don't need to extract an item by index before, adjust it, and create a new array with it. Instead, you simply iterate over all items in the array and only adjust the needed items by using conditional logic.

REACT STATE: REMOVE ITEM FROM ARRAY

When it comes to removing an item from an array, the array filter method is your friend.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [42, 33, 68],
    };
  }

  onRemoveItem = i => {
    this.setState(state => {
      const list = state.list.filter((item, j) => i !== j);

      return {
        list,
      };
    });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map((item, index) => (
            <li key={item}>
              The person is {item} years old.
              <button
                type="button"
                onClick={() => this.onRemoveItem(index)}
```

```
              >
                Remove
              </button>
            </li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

Similar to the other array methods, the filter method uses a function as argument that determines whether an item stays in the array or gets removed.

There is another neat little trick for one case: If you want to remove the first item in an array, you can do it with the array destructuring operator. Let's see how you can remove the first item of an array on a button click.

```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [42, 33, 68],
    };
  }

  onRemoveFirstItem = () => {
    this.setState(state => {
      const [first, ...rest] = state.list;

      return {
        list: rest,
      };
    });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item}>The person is {item} years old.</li>
          ))}

          <button type="button" onClick={this.onRemoveFirstItem}>
            Remove First Item
```

```
            Remove first item
          </button>
        </ul>

      </div>
    );
  }
}

export default App;
```

That's it. You destructure the first item and all the remaining items from the array. Then you use all remaining items to store them in React's state. The first item isn't used anymore.

All previous array examples worked on an array of integers. Let's see in a more complex scenario how to remove an object from a React state array instead. That example is better suited for a robust application, because we can work on identifiers instead of indexes.



```
import React, { Component } from 'react';

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      list: [
        { id: '1', age: 42 },
        { id: '2', age: 33 },
        { id: '3', age: 68 },
      ],
    };
  }

  onRemoveItem = id => {
    this.setState(state => {
      const list = state.list.filter(item => item.id !== id);

      return {
        list,
      };
    });
  };

  render() {
    return (
      <div>
        <ul>
          {this.state.list.map(item => (
            <li key={item.id}>
              The person is {item.age} years old.
              <button
                type="button"
```

```
                type="button"
                onClick={() => this.onRemoveItem(item.id)}
              >

                Remove
              </button>
            </li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

The example shows you that it doesn't make any difference for whether you are working with primitives or objects when using the previously applied JavaScript array methods. You would still use the array concat method to add an item, the array map method to update item(s), and the array filter method to remove an item. However, this time having an identifier for each item gives you more control over the array than having only indexes as before for the array manipulations.

. . .

You have learned about different ways on how to manage React state with arrays. You can add items to an array, update an item in the array or update the whole array, and remove items from an array. Everything is possible with only JavaScript without using React for it. React is only used to set the state in the end. It boils down to knowing the essential JavaScript array methods such as concat, map, filter and reduce.

KEEP READING ABOUT REACT›

REACT GLOBAL STATE WITHOUT REDUX

The article is a short tutorial on how to achieve global state in React without Redux. Creating a global state in React is one of the first signs that you may need Redux (or another state management...

REACT STATE WITHOUT CONSTRUCTOR

The article is a short tutorial on how to have state in React without a constructor in a class component and how to have state in React without a class component at all. It may be a great refresher on...

TAKE PART

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✔ Join 50.000+ Developers
- ✔ Learn Web Development with JavaScript
- ✔ Tips and Tricks
- ✔ Access Tutorials, eBooks and Courses
- ✔ Personal Development as a Software Engineer

Your email address SUBSCRIBE

View our Privacy Policy.

PORTFOLIO

Online Courses

Open Source

Tutorials

ABOUT

About me

What I use

How to work with me

How to support me

Contact Me Privacy & Terms