How to use filter() and reduce() in React

In this article we will see how to use the filter() and reduce() array methods in React.

Using filter()

The `filter()` method in JavaScript is a powerful tool when working with arrays, and is particularly useful in combination with React for manipulating and filtering the data displayed in the user interface.

In React, you can use the `filter()` method to create a filtered copy of an array of elements. For example, if you have an array of objects representing users and you want to view only active users, you can use `filter()` to filter the array based on a specific criteria.

Here is an example of using `filter()` in React:

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

function UserList() {
   const [users, setUsers] = useState([
     { id: 1, name: 'Mario', active: true },
     { id: 2, name: 'Luigi', active: false },
     { id: 3, name: 'Peach', active: true },
     { id: 4, name: 'Toad', active: false }
   ]);

   const activeUsers = users.filter(user =>
user.active);
```

```
    return (
      <div>
        <h1>Active Members List</h1>
        <ul>
          {activeUsers.map(user => (
            <li key={user.id}>{user.name}</li>
          ))}
        </ul>
      </div>
    );
}

export default UserList;
```

In the example above, the `users` array contains a list of users with properties such as `id`, `name` and `active`. The `filter()` method is used to get a fresh copy of the array that includes only active users. This filtered array is then mapped inside a `<ul>` element to display the names of active users in the list.

The `filter()` method offers considerable flexibility, as it allows you to define custom filter criteria to fit your specific application needs. You can filter on different object properties or apply more complex logic. Using `filter()` in combination with React allows you to easily manage data and render UI based on specific criteria.

Using reduce()

In React, the JavaScript reduce() method is a powerful function that allows you to flexibly manipulate and aggregate data. With its ability to reduce an array of values to a single value, it can be used in a variety of scenarios,

such as calculating the total of a list of elements or creating an object from an array.

Let's say we have a React application that manages a shopping cart. Each item in your cart has a name, quantity and price. We want to use the reduce() method to calculate the total amount to pay for the items in the cart.

Here is an example of how the `reduce()` method could be implemented in React:

```
import React from 'react';

const ShoppingCart = ({ items }) => {
   const totalPrice = items.reduce((accumulator,
item) => {
     return accumulator + (item.quantity *
item.price);
   }, 0);

   return (
     <div>
       <h1>Shopping Cart</h1>
       {items.map(item => (
         <div key={item.id}>
           <p>{item.name}</p>
           <p>Quantity: {item.quantity}</p>
           <p>Price: {item.price}</p>
         </div>
       ))}
       <h2>Total: {totalPrice}</h2>
     </div>
   );
};
```

```
export default ShoppingCart;
```

In the example above, we have a `ShoppingCart` component that accepts a `items` prop containing the array of items in the cart. We use the `reduce()` method on the `items` array to calculate the total amount to pay. We start with an initial value of 0 (passed as the second argument to the `reduce()` method) and for each element in the array, we multiply the quantity by the price and add it to the accumulator.

At the end, we get the total amount to pay in the value of the `totalPrice` variable. This value is then displayed within the `ShoppingCart` component.

Besides calculating the total, the `reduce()` method can be used in many other scenarios. For example, we could use it to create an object that groups the items in the cart by category, counting the number of items in each category. The flexibility of the `reduce()` method allows it to be adapted to the specific needs of the application.

In conclusion, the JavaScript `reduce()` method is a very useful tool for manipulating complex data in React. Using this method, it is possible to perform aggregation and calculation operations on data arrays in an elegant and flexible way. By leveraging the power of reduce(), you can simplify and make the process of manipulating data in your React application simpler and more efficient.