React's setState method with prevState argument

Asked 2 years, 6 months ago Active 9 days ago Viewed 74k times

30

9

I'm new to React, just have a question on setState method. Let's say we have a component:

```
class MyApp extends React.Component {

  state = {
    count: 3
  };

  Increment = () => {
    this.setState((prevState) => ({
      options: prevState.count + 1)
    }));
  }
}
```

so why we have to use `prevState` in the `setState` method? why can't we just do:

```
this.setState(() => ({
  options: this.state.count + 1)
}));
```

javascript reactjs

Share Improve this question Follow

edited Oct 19 at 8:07
Penny Liu
9,538 5 48 74

asked Apr 3 '19 at 12:27
user11224591

Try invoking your setState command multiple times and see the results for yourself. – emix Apr 3 '19 at 12:29

6 Possible duplicate of React - Functional setState (previous state) different from new updated value? – falinsky Apr 3 '19 at 12:29

2 "React may batch multiple setState() calls into a single update for performance. Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state." – UncleDave Apr 3 '19 at 12:29

Usually this doesn't matter as React will always batch multiple `setStates` triggered from the same event listener. That said, it is always a good idea to use the callback approach whenever you want to set a new state that depends on the old state. – Chris Apr 3 '19 at 12:35

1 @clint_milner neither approach mutates the state directly. Both are fine, the one could just be considered a "better practice" than the other. – Chris Apr 3 '19 at 12:42

7 Answers

Active Oldest Votes

27

Both signatures can be used, the only difference is that if you need to change your state based on the previous state you should use `this.setState(function)` which will provide you a snapshot(`prevState`) from the previous state. But if the change does not rely on any other previous value, then a shorter version is recommended `this.setState({prop: newValue})`

```
this.setState(prevState =>{
   return{
        ...prevState,
        counter : prevState.counter +1
   }
})

this.setState({counter : 2})
```

Share Improve this answer Follow edited Oct 20 '20 at 19:44 answered Apr 3 '19 at 13:22
Dupocas
17.3k 5 28 47

3 Don't do `prevState.counter++` as this mutates `prevState` . Do `prevState.counter + 1` instead. – Chris Apr 3 '19 at 13:47

1 It does. From the docs `The first argument is an updater function with the signature: (state, props) => stateChange. state is a reference to the component state at the time the change is being applied. It should not be directly mutated` . Besides why would you want to update `prevState.counter` and the return object counter when you only want to increment the latter? You could in theory also have another key that relied on `prevState.counter` which would not work if you did `++` . Say, `someCountDown: prevState.counter - 1` . – Chris Apr 3 '19 at 13:55

The variable is updated, but I'm generating a new property that merely mirror the actual counter from the previous state – Dupocas Apr 3 '19 at 13:56

Regardless, i'm updating the answer to reflect a more explicit attribution – Dupocas Apr 3 '19 at 13:57

1 I'm just saying it is a bad practice. I have done this exact mistake myself in the past and have had issues. – Chris Apr 3 '19 at 13:58

13

```
class MyApp extends React.Component {

  state = {
    count: 0
  };

  Increment = () => {
    this.setState(prevState => ({
      count: prevState.count + 1
    }));
    this.setState(prevState => ({
      count: prevState.count + 1
    }));
    this.setState(prevState => ({
      count: prevState.count + 1
    }));
    this.setState(prevState => ({
      count: prevState.count + 1
    }));
  };

  IncrementWithoutPrevState = () => {
      this.setState(() => ({
        count: this.state.count + 1
      }));
      this.setState(() => ({
        count: this.state.count + 1
      }));
      this.setState(() => ({
        count: this.state.count + 1
      }));
      this.setState(() => ({
        count: this.state.count + 1
      }));
    };

    render() {
      return (
        <div>
          <button onClick={this.IncrementWithoutPrevState}>
            Increment 4 times without PrevState
          </button>
          <button onClick={this.Increment}>
            Increment 4 times with PrevState
          </button>
          <h1>Count: {this.state.count}</h1>
        </div>
      );
    }
  }
```

I just made an example for you to give an idea what is meant by "React may batch multiple setState()..." and why we should use prevState in the above scenario.

First, try to guess what should the result of `Count` when you click both buttons... If you think the `count` will be incremented by 4 on click of both buttons then it's not right guess ;)

Why? because in `IncrementWithoutPrevState` method since there are multiple `setState` calls, so React batched all those calls and updates the `state` only in the last call of `setState` in this method, so at the time of last call to `setState` in this method `this.state.count` is not yet updated and its value is still the same that was before entering into `IncrementWithoutPrevState` method so the resultant state will contain `count` incremented by 1.

Now on the other hand side if we analyze the `Increment` method: Again there are multiple `setState` calls and React batched them all that means the actual state will be updated in the last call of `setState` but the `prevState` will always contain the modified `state` in the recent `setState` call. As `previousState.count` value has already been incremented 3 times till the last call of `setState` so the resultant `state` will contain the count value incremented by 4.

Share Improve this answer Follow

answered Apr 3 '19 at 14:03
samee
437 2 9

2

here iseveryone know just state:prevstate.counter+1.. we can also do this with state:this.state.counter+1. This is not the way i think to use prevstate. i have problem in array when i push into existing state it allow me but second time it prevent changes

Share Improve this answer Follow

answered Apr 29 '20 at 12:01

arslan
877 8 16

0

If you call a function multiple times to change the value of a state property in a single `render()` function call then the updated value will not go over between the different calls without prevState mechanism.

```
second(){
   this.setState({ // no previous or latest old state passed
       sec:this.state.sec + 1
     }, ()=>{
        console.log("Callback value", this.state.sec)
   })
}

fiveSecJump(){ // all this 5 call will call together
       this.second() // this call found sec = 0 then it will update sec = 0 +1
       this.second() // this call found sec = 0 then it will update sec = 0 +1
       this.second() // this call found sec = 0 then it will update sec = 0 +1
       this.second() // this call found sec = 0 then it will update sec = 0 +1
       this.second() // this call found sec = 0 then it will update sec = 0 +1
   }

  render() {
        return (
            <div>
              <div>  Count - {this.state.sec}</div>
              <button onClick ={()=>this.fiveSecJump()}>Increment</button>
            </div>
        )
    }
```

Finally sec value will be 1, because calls are async, not like high-level programming language like c++/c# or java where there is always a main function which maintains the main thread. Therefore if you want to have `fiveSecJump()` function to work properly you have to help it by passing it an arrow function. prevState. prevState is not a keyword or member function, you can write any word here like oldState, stackTopState, lastState. It will convert to a generic function which will do your desired work.

```
class Counter extends Component {
   constructor(props){
        super(props)
        this.state = {
            sec:0
        }
   }
   second(){
        this.setState(prevState =>({
            sec:prevState.sec + 1
        }))
   }

   fiveSecJump(){
       this.second() // this call found sec = 0 then it will update sec = 0 +1
       this.second() // this call found sec = 1 then it will update sec = 1 +1
       this.second() // this call found sec = 2 then it will update sec = 2 +1
       this.second() // this call found sec = 3 then it will update sec = 3 +1
       this.second() // this call found sec = 4 then it will update sec = 4 +1

   }
```

```
    render() {
        return (
            <div>
              <div>  Count - {this.state.sec}</div>
              <button onClick ={()=>this.fiveSecJump()}>Increment</button>
            </div>
        )
    }
}

export default Counter
```

Share Improve this answer Follow edited Mar 31 '20 at 20:08 answered Mar 31 '20 at 19:22

Lord
4,195 3 20 21

0

```
//Here is the example to explain both concepts:


import React, { Component } from 'react'

export default class Counter extends Component {
    constructor(props) {
        super(props);
        this.state = { counter: 0 }
    }

    increment() {
        this.setState({counter:this.state.counter+1})
    }

    increment3() {
        this.increment();
        this.increment()
        this.increment()
    }
   render() {
        return (
            <div>
               count-{this.state.counter}
                <div>
                    <button onClick={() => this.increment3()}>Increment</button>
                </div>
            </div>

        )
    }
}
```

In this scenario when we click on **Increment** button then the output will be rendered into the UI is **count-0** not **count-3** because react groups all the state call in a single state call and does not carry the updated value over every incremented call. If I want to update the value based on the previous value then use below mentioned code.

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

export default class Counter extends Component {
    constructor(props) {
        super(props);
        this.state = { counter: 0 }
    }
```

```
        increment() {
            this.setState((prevState=>({counter:prevState.counter+1})))
        }

        incremental() {
            this.increment();
            this.increment()
            this.increment()

        }
       render() {
            return (
                <div>
                   count-{this.state.counter}
                    <div>
                        <button onClick={() => this.incremental()}>Increment</button>
                    </div>
                </div>

            )
        }
    }
```

In this scenario the output will be **count-3** not **count-0**

Share Improve this answer Follow

edited Apr 11 at 19:32
marc_s
688k 163 1278 1403

answered Feb 17 at 11:55
Sheo Dayal Singh
1,327 15 9

0

@samee's answer was great and helped me out. I wanted to revive this topic to address React Function Components simply because that is what I'm currently studying.

The following is an example of why **you might want to use prevState if the new state is computed using the previous state**.

```
function App() {
  const [developer, setDeveloper] = useState({
    yearsExp: 0,
  });

  function handleYearsExp() {
    setDeveloper((prevState) => ({
      yearsExp: prevState.yearsExp + 1, //increments by 1
    }));
    setDeveloper((prevState) => ({
      yearsExp: prevState.yearsExp + 1, //increments by another 1
    }));
  }

  return (
    <>
      <button onClick={handleYearsExp}>Increment Years</button> {/* will increment by 2
*/}
      <p>
        I am learning ReactJS and I have {developer.yearsExp} years experience
      </p>
    </>
  );
}
```

Here's the code without the `prevState` function passed into `setState`. Notice that the first `setState` function is essentially ignored.

```
function App() {
  const [developer, setDeveloper] = useState({
    yearsExp: 0,
  });

  function handleYearsExp() {
    setDeveloper({
      yearsExp: developer.yearsExp + 1, // is ignored
    });
    setDeveloper({
      yearsExp: developer.yearsExp + 1, //increments by 1
    });
  }

  return (
    <>
      <button onClick={handleYearsExp}>Increment Years</button> {/* will only increment
by 1 */}
      <p>
        I am learning ReactJS and I have {developer.yearsExp} years experience
      </p>
    </>
  );
}
```

References:

1. ReactJS.org - "Pass a function to setState"
2. ReactJS.org - "React may batch multiple setState() calls into a single update for performance.

Share Improve this answer Follow edited Apr 13 at 5:07 answered Apr 13 at 4:59

Dani Amsalem
734 9 16

0

I look about the question, and also on handling-events (react website). I come to idea to try this way, for me more natural, and they work. I have confusion on about using .bind when this is undefined in callback. I'm correct write handleClick function?

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    // this.handleClick = this.handleClick.bind(this); - i Deleted this because we have
arrow fun. down

  }

  handleClick = (this.setState) => {
      this.isToggleOn: !this.state.isToggleOn
    }));
  }
  render() {
    return (
      <button onClick={handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Share Improve this answer Follow

answered Oct 8 at 12:44

Marko Anastasijevic
13 1 6