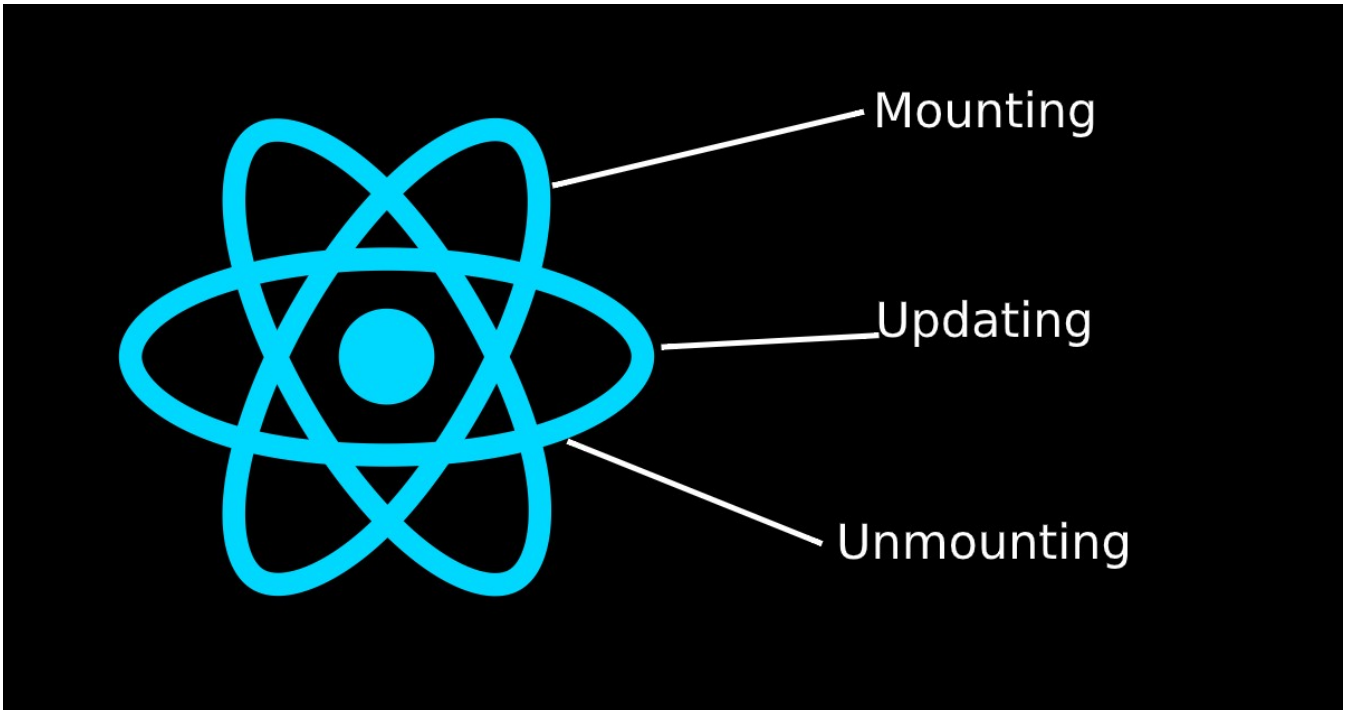
Open in app Get started

Published in How To React · Follow

Manish Mandal · Follow
Sep 29, 2020 · 6 min read

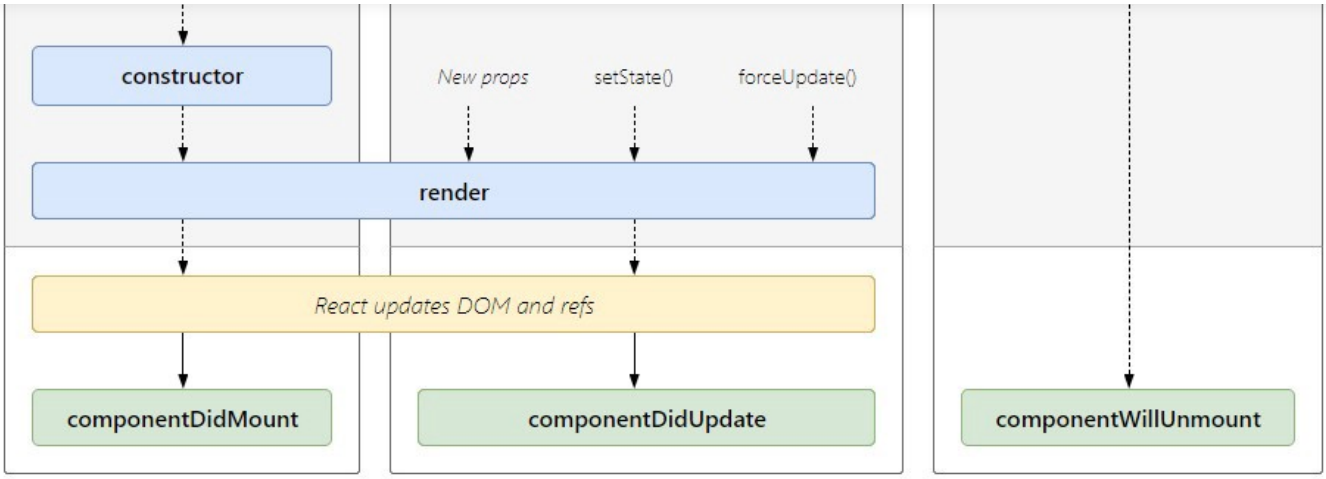
React Lifecycle methods with examples



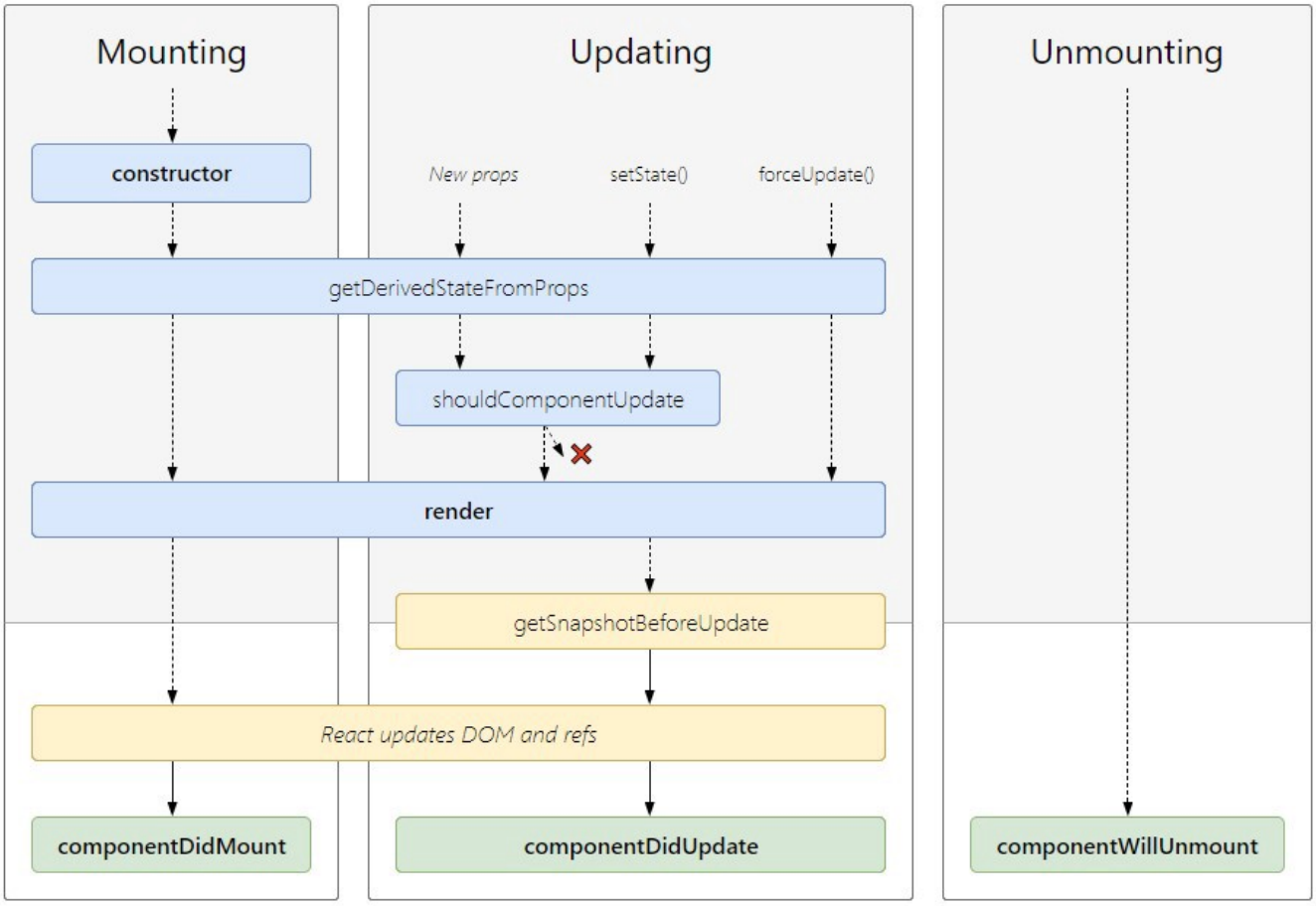
Let's learn about the react life cycle method and how to use them in our react projects.

As everything goes through a cycle of taking **birth**, **growing**, and **death** the same goes with React. Each component in React has a lifecycle that goes through three main phases **Mounting**, **Updating**, and **Unmounting**.

The react lifecycle method is used in the React **class** component. It helps us in creating our state and mutating them. Below are diagrams from wojtekmaj which explains how the React lifecycle works.



Common lifecycles



All lifecycle

These two diagrams explain the flow of the lifecycle method. So let us see how to use them in our project.

Mounting

namely

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

constructor()

constructor() method is called when the component is initiated and it's the best place to initialize our state. The constructor method takes **props** as an argument and starts by calling **super(props)** before anything else.

Example

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

export default class App extends Component {
  constructor(props){
    super(props)
    this.state = {
      name: 'Constructor Method'
    }
  }
  render() {
    return (
      <div>
       <p> This is a {this.state.name}</p>
      </div>
    )
  }
}
```

constructorMethod.is hosted with ❤ by GitHub view raw

Here **name** is our initial state inside our **constructor** method and then we are calling our initial state to **render** method using **this.state.name.** The following code will output `This is a Constructor Method .`

changes to the state.

Example

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

export class ChildComponent extends Component {
    constructor(props){
        super(props)
        this.state = {
          name: 'Constructor Method'
        }
      }

    static getDerivedStateFromProps(props, state) {
        return {name: props.nameFromParent}
      }
    render() {
        return (
            <div>
                This is a {this.state.name}
            </div>
        )
    }
}


export default class getDerivedStateFromPropsMethod extends Component {

    render() {
        return (
            <div>
                <ChildComponent nameFromParent="getDerivedStateFromProps Method"/>
            </div>
        )
    }
}
```

In the above code, I have initialized some state in the **constructor** method inside our **ChildComponent**. After that, I have called that ChildComponent inside



constructor method to props which we are receiving in **getDerivedStateFromProps.** The following code will output `This is a getDerivedStateFromProps Method.`

render()

This is the only compulsory method required by the React. This method is responsible to render our JSX to DOM

Example

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

export default class renderMethod extends Component {
    render() {
        return (
            <>
                <p>This is a render method</p>
            </>
        )
    }
}
```

renderMethod.js hosted with ❤ by GitHub view raw

componentDidMount()

The most common and widely used lifecycle method is **componentDidMount.** This method is called after the component is rendered. You can also use this method to call external data from the API.

Example 1

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

export default class componentDidMountMethod extends Component {
  constructor(props){
    super(props)
    this.state = {
      name: 'This name will change in 5 sec'
    }
```



```
15    }
16    render() {
17      return (
18        <div>
19         <p>{this.state.name}</p>
20        </div>
21      )
22    }
23  }
```

componentDidMountMethod.js hosted with ❤ by GitHub view raw

The above example will print `This is a componentDidMount Method` after 5 sec. This proves that the method is called after the component is rendered.

Example 2 Calling Data from API

```
1   import React, { Component } from 'react'
2
3   export default class componentDidMountMethod extends Component {
4     constructor(props){
5       super(props)
6       this.state = {
7         data: []
8       }
9     }
10
11    componentDidMount() {
12      fetch('https://jsonplaceholder.typicode.com/users').then(
13          (response) => response.json()
14      ).then(data => this.setState({data: data}))
15
16    }
17
18    render() {
19      return (
20        <div>
21         <p>This will print all the name available in API users data</p>
22      {this.state.data.map(d=> <h6 key={d.id}>{d.name}</h6>)}
23        </div>
24      )
25    }
```



componentDidMount method and then changed our **data** state using the React **setState** method from blank array to the data we are receiving from the API. Then we have used the javascript **map** function to iterate our data and print all users' names from the API.

Updating

This is the second phase of the React lifecycle. A component is updated when there is a change in `state` and `props` React basically has five built-in methods that are called while updating the components.

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

We have already discussed `getDerivedStateFromProps()` and `render()` a method so now let us discuss the remaining methods.

shouldComponentUpdate()

This lifecycle method is used when you want your state or props to update or not. This method returns a boolean value that specifies whether rendering should be done or not. The default value is `true` .

Example

```
1  import React, { Component } from 'react'
2
3  export default class shouldComponentUpdateMethod extends Component {
4    constructor(props){
5      super(props)
6      this.state = {
7        name: 'shouldComponentUpdate Method'
```



```
13
14    componentDidMount(){
15      setTimeout(() => {
16        this.setState({name: "componentDidMount Method"})
17      }, 5000)
18    }
19    render() {
20      return (
21        <div>
22         <p>This is a {this.state.name}</p>
23        </div>
24      )
25    }
26  }
```

In the above example, you will notice that first I have initialized the `name` state inside the **constructor** method and after that changed state using **setState** inside **componentDidMount** method. So basically the `name` state should be changed from `"shouldComponentUpdate Method"` to `"componentDidMount Method"` after 5 seconds but it didn't change because of **shouldComponentUpdate** set to false, If you change that true the state will be updated.

getSnapshotBeforeUpdate()

This method is called right before updating the DOM. It has access to **props** and **state** before the update. Here you can check what was the value of your props or state before its update. So let see how it works.

`Note:` **componentDidUpdate()** should be included otherwise you will get an error.

Example

```
1  import React, { Component } from 'react'
2
3  export default class getSnapshotBeforeUpdateMethod extends Component {
4      constructor(props){
5          super(props)
6          this.state = {
```



```
12          setTimeout(() => {
13            this.setState({name: "componentDidMount Method"})
14          }, 5000)
15        }
16        getSnapshotBeforeUpdate(prevProps, prevState) {
17          document.getElementById('previous-state').innerHTML = "The previous state was " + prevS
18        }
19        componentDidUpdate() {
20          document.getElementById('current-state').innerHTML = "The current state is " + this.sta
21        }
22      render() {
23          return (
24              <>
25                 <h5>This is a {this.state.name}</h5>
26                  <p id="current-state"></p>
27                  <p id="previous-state"></p>
28              </>
29          )
30      }
31  }
```

In the above example, we initialized our `name` state as `constructor Method` after that changed that using **setState** in **componentDidMount** method to `componentDidMount Method` . So my previous state was `constructor Method` and the current state is `componentDidMount Method` So now I can get my previous State from the **getSnapshotBeforeUpdate** method. So using the **getSnapshotBeforeUpdate** method I have printed my previous state to our DOM using `document.getElementById('previous-state').innerHTML = "The previous state was " + prevState.name .`

componentDidUpdate()

The **componentDidUpdate** method is called after the component is updated in the DOM. This is the best place in updating the DOM in response to the change of props and state.

Example

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



```
            name: 'from previous state'
        }
    }
    componentDidMount(){
        setTimeout(() => {
            this.setState({name: "to current state"})
          }, 5000)
    }
    componentDidUpdate(prevState){
        if(prevState.name !== this.state.name){
            document.getElementById('statechange').innerHTML = "Yes the state is changed"
        }
    }
    render() {
        return (
            <div>
                State was changed {this.state.name}
                <p id="statechange"></p>
            </div>
        )
    }
}
```

In the above example, our initial **name** state was `from previous state` and after that using **setState**, I have set the **name** state to `to current state` So React will render the **name** state from `State was changed from previous state` to `State was changed to current state` after 5 seconds. Using the conditional checking of the current state with the previous state `prevState.name !== this.state.name` inside the **componentDidUpdate** method, we are updating the value of the id `statechange` to `Yes the state is changed .`

`Note:` You may also call **setState** inside **componentDidUpdate** but you must wrap that in a condition like in the example above or it'll cause an infinite loop.

Unmounting

The final or the end of the react lifecycle is Unmounting. This is used when a component is removed from the DOM. React has only one built-in method that gets

componentWillUnmount()

If there are any cleanup actions like canceling API calls or clearing any caches in storage you can perform that in the componentWillUnmount method. You cannot use **setState** inside this method as the component will never be re-rendered.

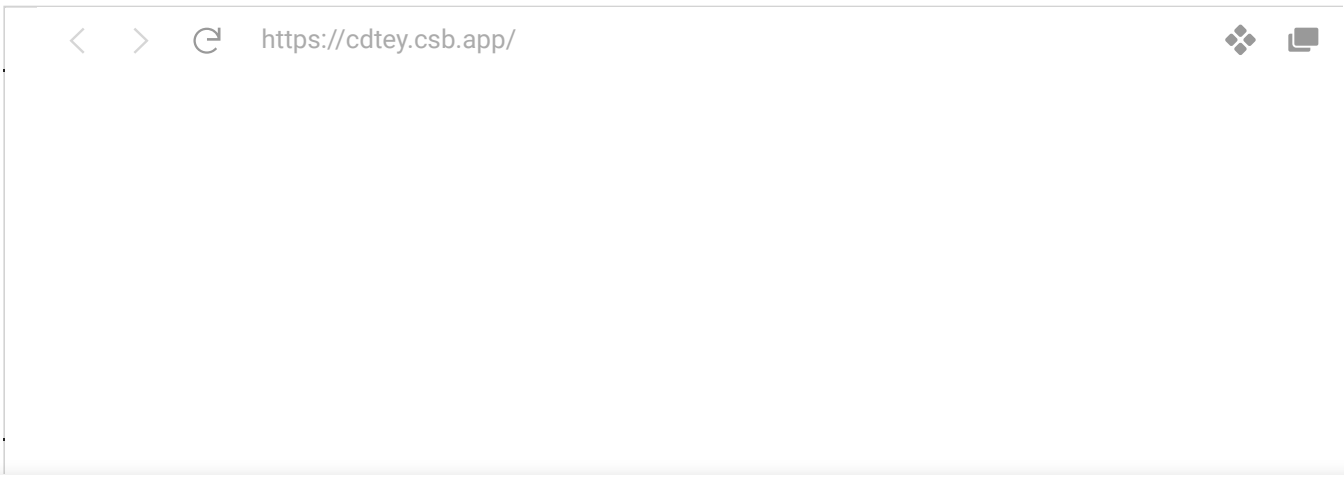
Example

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

export default class componentWillUnmount extends Component {
    constructor(props){
        super(props)
            this.state = {
                show: true,
            }
    }
    render() {
        return (
            <>
              <p>{this.state.show ? <Child/> : null}</p>
               <button onClick={() => {this.setState({show: !this.state.show})}}>Click me to to
            </>
        )
    }
}

export class Child extends Component{
    componentWillUnmount(){
        alert('This will unmount')
    }
    render(){
        return(
            <>
            I am a child component
            </>
        )
    }
}
```

Summary

- The constructor() method is the best place to initialize our state
- The getDerivedStateFromProps() is a rarely used lifecycle method and is the best place to set the state object based on the initial props.
- The shouldComponentUpdate() specifies whether React should continue with the rendering or not.
- The render() method is the most used and compulsory lifecycle method.
- The getSnapshotBeforeUpdate() method has access to the props and state even after the update.
- The componentDidMount() is the most common and widely used lifecycle method and is called after the component is rendered. You can also use this method to call external data from the API.
- The componentDidUpdate() method is called after the component is updated in the DOM and is the best place in updating the DOM in response to the change of props and state.
- The componentWillUnmount() happens just before the component unmounts and is destroyed and is used for cleanup actions like canceling API calls.

Below are the Github repository and live example for reference.





Open Sandbox

Console 3 Problems 0 React DevTools 0

Mediumtutorial/react-lifecycle-medium

This project was bootstrapped with Create React App. In the project directory, you can run: Runs the app in the...

github.com

Sign up for Newsletter

By How To React

Subscribe our newsletter to get all the latest tutorials on How To React. Take a look.

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.