

# React Functional Components VS Class Components



bikash bhandari

Follow

Feb 12, 2020 · 4 min read

From this article, we will learn the differences between functional and class components in React and also i will explain which one you should choose.

## Introduction to component

- >a component represent the part of user interface
- >component are reusable and can be use in anywhere in user interface

There are mainly two components in React:

1. Functional Components also known as Stateless component
2. Class Component also known as Stateful component

## Functional Component or Stateless Component:

- It is simple javascript functions that simply returns html UI
- It is also called “stateless” components because they simply accept data and display them in some form that is they are mainly responsible for rendering UI.
- It accept properties(props) in function and return html(JSX)
- It gives solution without using state
- There is no render method used in functional components.

- These can be typically defined using arrow functions but can also be created with the regular function keyword.

```
example:
//with arrow function
import React from "react";

const Person = (props) => (
  <div>
    <h1>Hello, {props.name}</h1>
  </div>
);

export default Person;

//without arrow function

import React from "react";

function Person(props) {

  return (
    <div>
      <h1>Hello, {props.name}</h1>
    </div>
  )

};

export default Person;
```

## Lifecycle method or Lifecycle hooks in functional component

Component lifecycle method do not exist in functional component, because a functional component is just a plain JavaScript function, we cannot use `setState()` method inside component. That's why they also get called functional stateless components.

We can use React Hooks in functional component, `useEffect()` hook can be used to replicate lifecycle behaviour, and `useState` can be used to store state in a functional component.

```
example:
const User = (props) => {
```

```
const [values, setValues] = useState({
  email: "",
  password: ""
});
```

```
useEffect(() => {
  if(!props.fetched) {
    props.fetchData();
  }
  console.log('mount it!');
}, []);
```

NOTE:By passing an empty array as second argument triggers the callback in useEffect only after the initial render thus replicating `componentDidMount` lifecycle behaviour

```
return(
  )
}
```

useEffect can also return a function that will be run when the component is unmounted. This can be used to unsubscribe to listeners. An can be used to replicate the componentWillUnmount behaviour  
Eg: componentWillUnmount

```
useEffect(() => {
  window.addEventListener('unhandledRejection', handler);
  return () => {
    window.removeEventListener('unhandledRejection', handler);
  }
}, [])
```

## Class Component or Stateful Component

- It is regular ES6 classes that extends component class form react library
- Also known as “stateful” components because they implement logic and state.
- It must have render() method returning html
- It has complex UI Logic
- You pass props to class components and access them with this.props

example:  
import React, { Component } from "react";

```
class Person extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "bikash";
    }
  }

  render() {
    return (
      <div>
        <h1>Hello {this.state.name}</h1>
      </div>
    );
  }
}

export default Person;
```

## Lifecycle method or Lifecycle hooks in Class Component

The lifecycle hooks of class component can be classified into 4 phases.

### 1. Mounting():

- when an instance of a component is being created and inserted into the DOM
- it has 4 methods: constructor(), static getDerivedStateFromProps(), render() and componentDidMount()

#### a. constructor(props):

- > a special function that will be called whenever a new component is created
- > initializing the state, binding the event handler
- > super(props) that directly overrides this.state

#### b. static getDerivedStateFromProps(props, state)

- > when the state of the component depends on props

#### c. render():

- > we read props and state and return jsx

#### d. componentDidMount()

- > invoked immediately after a component and all its children components have been rendered to the DOM

## 2.Updating():

- when a component is being re-rendered as a result of change to either its props or state
- it has 5 lifecycle static methods: `getDerivedStateFromProps()`, `shouldComponentUpdate()`, `render()`, `getSnapshotUpdate()` and `componentDidUpdate()`

### a.`getDerivedStateFromProps()`

->This is the first method that is called when a component gets updated.

->This is still the natural place to set the state object based on the initial props.

### b.`shouldComponentUpdate()`

->In this method you can return a Boolean value that specifies whether React should continue with the rendering or not.

->The default value is true.

### c.`getSnapshotBeforeUpdate()`

->the `getSnapshotBeforeUpdate()` method you have access to the props and state before the update, meaning that even after the update, you can check what the values were before the update.

->If the `getSnapshotBeforeUpdate()` method is present, you should also include the `componentDidUpdate()` method, otherwise you will get an error.

### d.`componentDidUpdate()`

->the `componentDidUpdate()` method is called after the component is updated in the DOM.

## 3.UnMounting:

- used when component is being removed from the DOM.
- it has 1 lifecycle method `componentWillUnmount()`

## 4.Error Handling:

- when there is an error during rendering
- it has 2 lifecycle static methods: `getDerivedStateFromProps()` and `componentDidCatch()`

## So why to use functional components insted of class component?

Benefits you get by using functional components in React:

- 1.Functional component are much easier to read and test because they are plain JavaScript functions without state or lifecycle-hooks
- 2.It has less code which makes it more readable
- 3.It will get easier to separate container and presentational components because you need to think more about your component's state if you don't have access to `setState()` in your component

### Conclusion:

If you are writing a presentational component which doesn't have its own state or needs to access a lifecycle hook,use functional component as much as possible. For state management you can use class component.

[React](#)   [Components](#)   [React Component Lifecycle](#)

[About](#)   [Help](#)   [Legal](#)

Get the Medium app

