React hooks: What/Why `useEffect`?

Asked 3 years, 4 months ago Modified 8 months ago Viewed 11k times

27

8

Concerning the newly proposed React Effect Hook;

1. What are the advantages and use cases of the `Effect` hook (`useEffect()`)?
2. Why would it be preferable & how does it differ over `componentDidMount/componentDidUpdate/componentWillUnmount` (performance/readability)?

The documentation states that:

> Mutations, subscriptions, timers, logging, and other side effects are not allowed inside the main body of a function component (referred to as React's render phase).

but I think it was already common knowledge to have these behaviors in lifecycle methods like componentDidUpdate, etc. instead of the render method.

There's also the mention that:

> The function passed to useEffect will run after the render is committed to the screen.

but isn't that what `componentDidMount` & `componentDidUpdate` do anyways?

javascript reactjs react-hooks

Share Follow

edited Oct 30, 2018 at 8:09
skyboyer
18.9k 7 48 61

asked Oct 29, 2018 at 18:12
Mirodinho
959 1 12 24

3 Can you compare your question to the parts in the documentation that you don't understand? – Mark C. Oct 29, 2018 at 18:23

@MarkC. Added more info to make question less broad. – Mirodinho Oct 29, 2018 at 19:18

You could ask in the official repo your questions if the current documentation is not clear. The official doc asked also to raise any questions readers have to the github as a feedback. This doesn't fit here very well. All answers you may get a here of a digest of all available blog posts. You already got such an answer. But if you do ask in the official repo, and if they think the documentation needs an update, it will happen. By doing that many people will be helpful. – Arup Rakshit Oct 29, 2018 at 19:28

3 Answers

Highest score (default)

19

1. What are the advantages and use cases of the `Effect` hook (`useEffect()`)?

Advantages

Primarily, hooks in general enable the extraction and reuse of stateful logic that is common across multiple components without the burden of higher order components or render props.

A secondary benefit (of Effect hooks in particular) is the avoidance of bugs that might otherwise arise if state-dependent side effects are not properly handled within `componentDidUpdate` (since Effect hooks ensure that such side effects are setup and torn-down on every render).

See also the peformance and readability benefits detailed below.

Use cases

Any component that implements stateful logic using lifecycle methods—the Effect hook is a "Better Way".

2. Why would it be preferable & how does it differ over `componentDidMount` / `componentDidUpdate` / `componentWillUnmount` (performance/readability)?

Why it's preferable

Because of the advantages detailed above and below.

How it differs from lifecycle methods

Performance

Effect hooks—

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up

- will however setup and tear-down side effects on every render, which could be expensive...
- ...so can be optimised to be skipped entirely unless specific state has been updated.

Readability

Effect hooks result in:

- simpler and more maintainable components, owing to an ability to split unrelated behaviour that previously had to be expressed across the same set of lifecycle methods into a single hook for each such behaviour—for example:

  ```
  componentDidMount() {
    prepareBehaviourOne();
    prepareBehaviourTwo();
  }

  componentDidUnmount() {
    releaseBehaviourOne();
    releaseBehaviourTwo();
  }
  ```

  becomes:

  ```
  useEffect(() => {
    prepareBehaviourOne();
    return releaseBehaviourOne;
  });

  useEffect(() => {
    prepareBehaviourTwo();
    return releaseBehaviourTwo;
  });
  ```

  Notice that code relating to `BehaviourOne` is now distinctly separated from that relating to `BehaviourTwo`, whereas before it was intermingled within each lifecycle method.

- less boilerplate, owing to an elimination of any need to repeat the same code across multiple lifecycle methods (such as is common between `componentDidMount` and `componentDidUpdate`)—for example:

  ```
  componentDidMount() {
    doStuff();
  }

  componentDidUpdate() {
    doStuff();
  }
  ```

```
useEffect(doStuff); // you'll probably use an arrow function in reality
```

Share Follow

answered Oct 29, 2018 at 19:48
eggyal
118k 18 200 231

13

Here is an example from *ReactConf2018 Dan Abramov's* talk explaining the difference:

Here are the few findings from the below example:

1. You'll writing less boilerplate code using hooks
2. Accessing lifecycles updates and states updates with `useEffect()`
3. Regarding performace one aspect is:

> Unlike componentDidMount and componentDidUpdate, the function passed to useEffect fires after layout and paint, during a deferred event

4. Code sharing will too much easy and useEffect() can be implemented multiple times for different purposes within the same component.
5. you can control component re render more efficiently by passing an array as second argument to `useEffect()` hook that is very effective when you just pass empty array [] to render component on only mounting and unmounting.
6. Use Multiple `useEffect()` hooks to Separate Concerns and react will:

> Hooks lets us split the code based on what it is doing rather than a lifecycle method name. React will apply every effect used by the component, in the order they were specified

Using Classes:

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
```

```
    componentDidUpdate() {
      document.title = `You clicked ${this.state.count} times`;
    }

    render() {
      return (
        <div>
          <p>You clicked {this.state.count} times</p>
          <button onClick={() => this.setState({ count: this.state.count + 1 })}>
            Click me
          </button>
        </div>
      );
    }
  }
```

Using Hooks:

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

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Share Follow edited Oct 29, 2018 at 19:24 answered Oct 29, 2018 at 18:32

user Sakhi Mansoor
6,992 5 19 34

2

useEffect runs when a state changed.

```
import { useState, useEffect } from 'react';
function Example() {
  const [Age, setAge] = useState(33);
  const [Somestate,setSomestate]=useState(initilaestate);
  useEffect(() => {
    console.log('the age is changed to ',Age);
```

Join Stack Overflow to learn, share knowledge, and build your career. Sign up

```
    console.log('the age is changed to ',Age);
  },[someState]);//here you tell useEffect what state to watch if you want to watch the
changing of a  particular state and here we care about someState
  return (
    <div>
      <p>age increased to  {Age}</p>
      <button onClick={() => setAge(count + 1)}>
       Increase age by One
      </button>
    </div>
  );
}
```

Share Follow

answered Oct 19, 2020 at 10:07

Salah
358 2 12