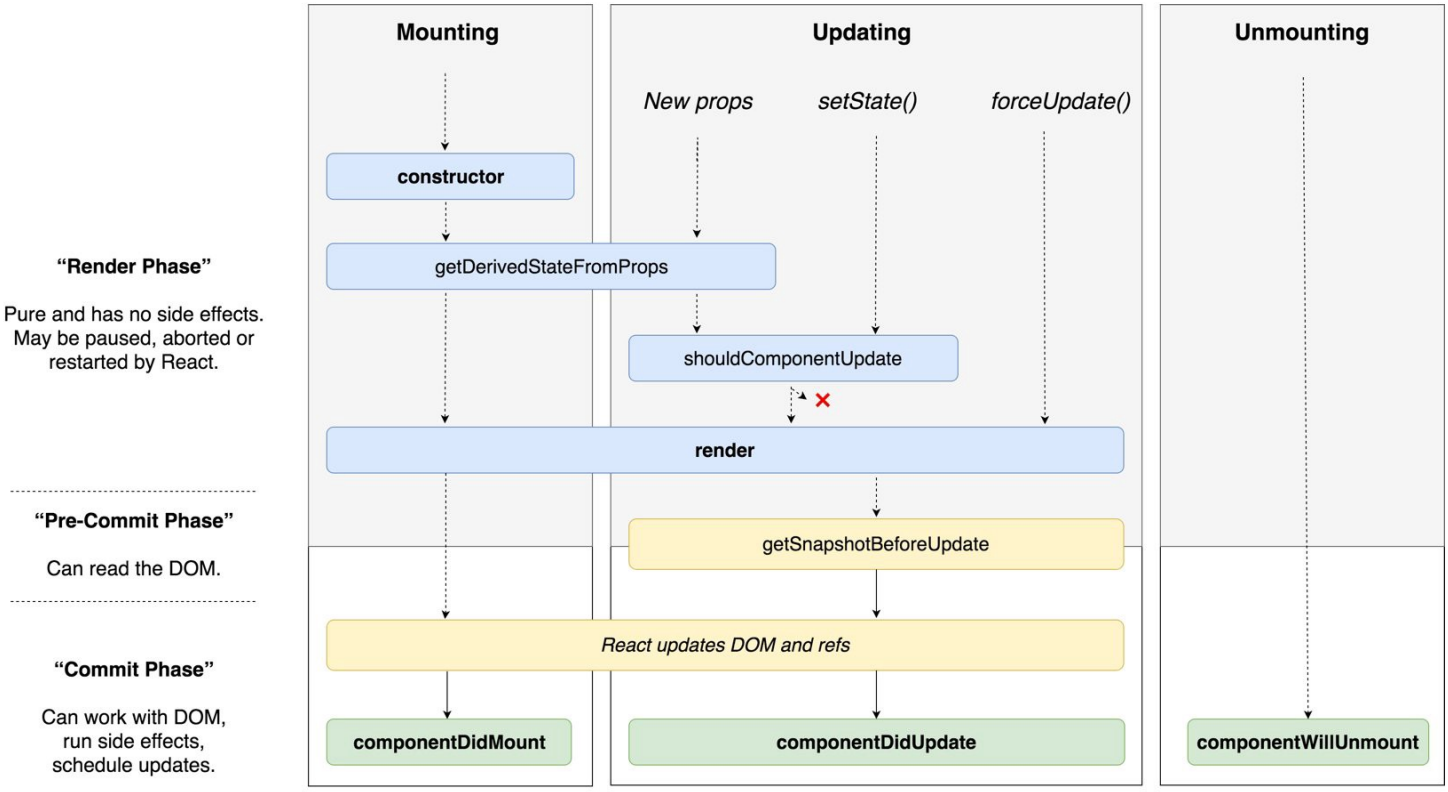
Understanding React component life-cycle

Ngoc Vuong Follow
May 26, 2018 · 5 min read

React 16.3

. . .



https://twitter.com/dan_abramov/status/981712092611989509

. . .

Each React component comes with several methods that allows developers to update the application state and reflect the change to UI. There are three main phases of a component which including **mounting**, **updating** and **unmounting**.

Mounting

These methods will be called when an instance of a React component is created and mounted into the DOM.

constructor()

This method get called before a React component is mounted. It is very essential to call `super(props)` before any statement in the constructor. This is because it will allow us to call the constructor of the parent class and initialize itself in case our class extends any other class which has constructor itself. Otherwise, `this.props` will be undefined whether the component has props or not.

The constructor is perfect for initializing state or bind the event handlers to the class instance. For example

```
constructor(props) {
    super(props);
    this.state = {
      count: 0,
      value: 'Hey There!',
    };
    this.handleClick = this.handleClick.bind(this);
};
```

DON'T do any side-effect in the constructor.

componentWillMount() / UNSAFE_componentWillMount()

`componentWillMount` will be called once before the component is mounted and it will run before the `render` function. This function has been used to fetch data from other endpoint in order to retrieve the data for the initial render by many developers (me included). However, this is not the case because there is no guarantee that the request will be finished before the `render` function is called.

According to official react page, it is advised not to do any side-effect or subscriptions in this functions. Instead, use `componentDidMount`

Also, `setState` will be useless as it will not trigger any re-render.

Note: This function has been **deprecated** and will be changed to `UNSAFE_componentWillMount` from version 17.

static getDerivedStateFromProps(nextProps, prevState)

This function is invoked when the component is mounted as well as receives new props whether they are changed or not. It is also called if the parent component is re-render, so if you want to only update change of value, it is vital to have the previous and new value comparison.

This function should return an object to update state or null because the static function will not have any acknowledge to `this` . If your component has state that is initialized from the props receiving from the parents. This function is right place to sync up your props and state. To do so, when you need the state to be updated, remember to return the object containing the property that need to be updated.

```
static getDerivedStateFromProps(nextProps, prevState) {
    if (nextProps.value !== prevState.value) {
      console.log('props changed. Return an object to change
state');
      return {
        value: nextProps.value,
      }
    }
```

componentDidMount()

After a component is mounted, this method would be invoked. This is the right place to load any data from endpoint or set up any subscription.

Calling here `setState` will trigger re-render, so use this method with caution.

Updating

componentWillReceiveProps(nextProps)/UNSAFE_componentWillReceiveProps(nextProps)

This function will be called:

- When the component received new props whether the props are changed or not. Therefore, if you want to set up new state based on props, remember to compare the value, pretty the same as `getDerivedStateFromProps`

```
componentWillReceiveProps(nextProps, prevState) {
    if (nextProps.value !== prevState.value) {
      console.log('props changed. Return an object to change
state');
      return {
        value: nextProps.value,
      }
    }
```

- When the parent component is re-rendered.

Calling `this.setState` won't trigger any render and it will not be called on the initial render. It is not recommended to use this method, instead use `getDerivedStateFromProps`

Note: This function has been **deprecated** and will be changed to `UNSAFE_componentWillReceiveProps` from version 17.

static getDerivedStateFromProps(nextProps, nextState)

In summary, this function will be called when:

- The component is initialized.
- Receiving new props whether they are changed or not.
- Parent component is re-rendered.

Note: when this function is implemented, it is likely that the `componentWill*` functions will not be invoked.

shouldComponentUpdate(nextProps, nextState)

This method will be invoked before rendering when receiving new props or state. By default, when there is a state or prop change, the

components will re-rendered. If you are in a situation when you do not wish the components to have many useless rendering, this method is the wonderful place to verify all the changes you needed before re-rendering. If you do not want to re-render the components, return `false` , otherwise it is expected to return `true` .

Note: When this functions return `false` , `UNSAFE_componentWillUpdate` , `render` and `componentDidUpdate` will not be invoked.

componentWillUpdate(nextProps, nextState) / UNSAFE_componentWillUpdate(nextProps, nextState)

This function will be invoked before every rendering when receiving new props or state. This can be used to perform some preparations before the update occurs.

This function will not be called on initial render. If you wish to update state based on props changes, consider `getdeviredStateFromProps` .

You cannot call `setState` in this function.

Note: This function has been **deprecated** and will be changed to `UNSAFE_componentWillUpdate` from version 17.

getSnapshotBeforeUpdate()

This method is called right before the virtual DOM is about to make change to the DOM, which allows our components to capture the current values. This is a new cycle that has been added since React 16.3 and it is recommended to return a value. Any value returned will be passed as a third parameter in `componentDidUpdate` .

componentDidUpdate(prevProps, prevState, snapshot)

This method will get called after every rendering occurs. As this method is only called once after the update, it is a suitable place for implement any side effect operations. However, do not forget to do the comparison between the previous and current props.

```
componentDidUpdate(prevProps, prevState, snapshot) {
    if (this.state.count !== prevState.count) {
      // state has change!! Do some side effect as you wish
```

```
    }
  }
```

Unmounting

componentWillUnmount()

When a component is unmounted or destroyed, this method will be called. This is a place to do some clean up like

- Invalidating timers
- Canceling any network request
- Remove event handlers
- Clean up any subscriptions

Calling `setState` here is useless because there will be no re-rendering on the component.

componentDidCatch(error, info)

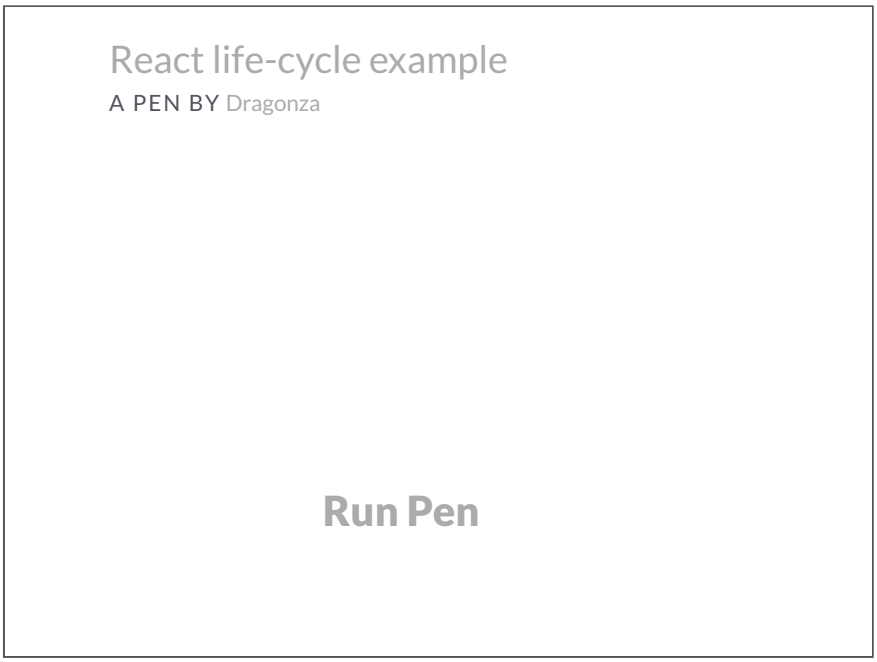
Since React 16, a new life-cycle has been introduced in order to capture uncaught error happening in the child components. This will enable us to handle the parent component to render the fallback UI when needed instead of crashing the component.

```
componentDidCatch(error, info) {
    this.setState({ hasError: true });
  }

  render() {
    if (this.state.hasError) {
     //custom fallback UI
      return <div>Something went wrong.</div>;
    }
    return this.props.children; //render children
  }
```

To see how the life cycle works, you can check out my example below. The explanation of each methods is all in the console, so it is recommended to open a new page for this example.

How it looks like in console

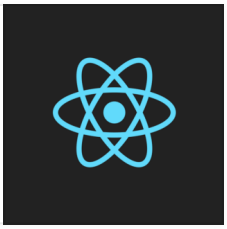


References

React.Component - React

A JavaScript library for building user interfaces

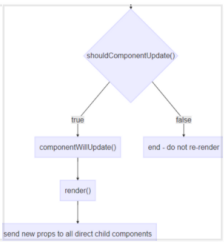
reactjs.org



Understanding React — Component life-cycle

React provides developers with many methods or "hooks" that are called during the life-cycle of an...

medium.com



Understanding React — React 16.3 + Component life-cycle

The release of 16.3 introduced some new life-cycle functions, which replace existing ones to provide...

medium.com