6:59:00 PM.

Header paragraph from parent props

Header paragraph1 from parent props

How to change the state of a child component from its parent in React?

We will take two components, Parent and Child. And our Parent component will set the value depends on the Child Component. Child component holds the Input field and we are going to send the input field value to the Parent component.

How to get the state/value of a child component from its parent in React?

Parent Component [App.js]
Child input element is: Empty!
Child component [shared/child.js]
Enter Name
Submit

Toggle Button Close!

- 1
- 2
- 3
- 4
- 5

- Hello World
- Installation

Hello World

Welcome to learning React!

Installation

You can install React from npm.

Hello Vijaya Kumar - HOC!

Hello Click me, see the count of no.of clicks! - HOC!

Count is: 0

Statefull and Stateless components

- Robin
- Hello
- James
- Watson
- Vijay

Pure Component:

Pure Components in React are the components which do not re-renders when the value of state and props has been updated with the same values. If the value of the previous state or props and the new state or props is the same, the component is not re-rendered.

Excercise
Cooking
Reacting
Raju

Features of React Pure Components:

Prevents re-rendering of Component if props or state is the same

Takes care of shouldComponentUpdate() implicitly

State() and Props are Shallow Compared

Pure Components are more performant in certain cases

When to use PureComponent:

We want to avoid re-rendering cycles of component when its props and state are not changed, and

The state and props of component are immutable, and

We do not plan to implement own shouldComponentUpdate() lifecycle method

Controlled Component:

In a controlled component, the form data is handled by the state within the component. The state within the component serves as “the single source of truth” for the input elements that are rendered by the component.

About Controlled components

Your message here..

the message is:

Uncontrolled Component:

Uncontrolled components act more like traditional HTML form elements. The data for each input element is stored in the DOM, not in the component. Instead of writing an event handler for all of your state updates, It uses ref to retrieve values from the DOM. Refs provide a way to access DOM nodes or React elements created in the render method.

About Unctrolled components

Your message here..

React.cloneElement:

The React.cloneElement() function returns a copy of a specified element. Additional props and children can be passed on in the function. This function is used when a parent component wants to add or modify the prop(s) of its children.

Hello World!

Hello World!

React Hook - useState ():

The useState() is a Hook that allows to have state variables in functional components.

Count: 0 Increase Decrease

The useState() function takes as argument a value for the initial state. In this case, the count starts out with 0. In addition, the hook returns an array of two values: count and setCount. It's up to you to name the two values, because they are destructured from the returned array where renaming is allowed.

React Hook - useReducer ():

It accepts a reducer function with the application initial state, returns the current application state, then dispatches a function.

Here, we first define an initialState and a reducer. When a user clicks a button, it will dispatch an action which updates the count and the updated count will be displayed. We could define as many actions as possible in the reducer, but the limitation of this pattern is that actions are finite.

0 +1 -1 reset

React Hook - useContext ():

The React Context API allows to easily access data at different levels of the component tree, without having to pass data down through props.

The answer is 100.

React Hook - useEffect ():

In react when we use class based components we get access to lifecycle methods(like componentDidMount, componentDidUpdat, etc). But when we want use a functional component and also we want to use lifecycle methods, then using useEffect() we can implement those lifecycle methods.

Limitation:

When useEffect() is used to get data from server.

The first argument is a callback that will be fired after browser layout and paint.

Therefore it does not block the painting process of the browser.

The second argument is an array of values (usually props).

If any of the value in the array changes, the callback will be fired after every render.

When it is not present, the callback will always be fired after every render.

When it is an empty list, the callback will only be fired once, similar to componentDidMount.

React.useEffect

React Hook - Remo()

React.memo is a higher order component. It's similar to React.PureComponent but for function components instead of classes

If your function component renders the same result given the same props, you can wrap it in a call to React.memo for a performance boost in some cases by memoizing the result. This means that React will skip rendering the component, and reuse the last rendered result.

React.memo only checks for prop changes. If your function component wrapped in React.memo has a useState or useContext Hook in its implementation, it will still rerender when state or context change.

Remo

Refs :

Refs provide a way to access DOM nodes or React elements created in the render method. React Refs are a useful feature that act as a means to reference a DOM element or a class component from within a parent component.

Refs also provide some flexibility for referencing elements within a child component from a parent component, in the form of ref forwarding.

When to Use Refs:

Managing focus, text selection, or media playback.

Triggering imperative animations.

Integrating with third-party DOM libraries.

When not to use refs:

Should not be used with functional components because they dont have instances.

Not to be used on things that can be done declaritvely.

React Ref - createRef

Value:

Submit

React Lifecycle methods:

React provides several methods that notify us when certain stage of this process occurs. These methods are called the component lifecycle methods and they are invoked in a predictable order. The lifecycle of the component is divided into four phases.

Mouting methods:

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

constructor(), getDerivedStateFromProps(), render(), componentDidMount()

The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.

The getDerivedStateFromProps() method is called right before rendering the element(s) in the DOM. It takes state as an argument, and returns an object with changes to the state.

My Favorite Color: yellow

The render() method is required, and is the method that actual outputs HTML to the DOM.

The componentDidMount() method is called after the component is rendered.

Updating methods:

The next phase in the lifecycle is when a component is updated. A component is updated whenever there is a change in the component's state or props. React has five built-in methods that gets called, in this order, when a component is updated:

getDerivedStateFromProps()

shouldComponentUpdate()

render()

getSnapshotBeforeUpdate()

componentDidUpdate()

The 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.

My Favorite Color is : yellow

Change color

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not. The default value is true .

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

My Favorite Color is yellow

Before the update, the favorite was red
The updated favorite is yellow

The componentDidUpdate() method is called after the component is updated in the DOM.

Unmouting methods:

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.

Example: Click the button to delete the Hello World Unmount child compoent:

Hello World! from Unmount child compoent

Delete Hello World Unmount Child

React Routers:

React router implements a component-based approach to routing. It provides different routing components according to the needs of the application and platform. React Router keeps your UI in sync with the URL. It has a simple API with powerful features like lazy loading, dynamic route matching, and location transition handling built right in.

In which lifecycle event do you make AJAX requests in React?

According to official React docs, the recommended place to do Ajax requests is in**componentDidMount()**which is a lifecycle method that runs after the React component has been mounted to the DOM. This is so you can use **setState()** to update your component when the data is retrieved.

Error: Failed to fetch

Redux

Redux is a predictable state container for JavaScript applications. It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. With Redux, the state of your application is kept in a store, and each component can access any state that it needs from this store.

How Redux works

There is a central store that holds the entire state of the application. Each component can access the stored state without having to send down props from one component to another. There are three building parts: actions, store, and reducers.

Benefits and limitations of Redux

1. State transfer - State is stored together in a single place called the 'store.' While you do not need to store all the state variables in the 'store,' it is especially important to when state is being shared by multiple components or in a more complex architecture. It also allows you to call state data from any component easily.

Predictability - Redux is "a predictable state container for Javascript apps." Because reducers are pure functions, the same result will always be produced when a state and action are passed in.

Maintainability - Redux provides a strict structure for how the code and state should be managed, which makes the architecture easy to replicate and scale for somebody who has previous experience with Redux

Ease of testing and debugging - Redux makes it easy to test and debug your code since it offers powerful tools such as Redux DevTools in which you can time travel to debug, track your changes, and much more to streamline your development process.

What are redux core concepts?

Actions in Redux - Action is static information about the event that initiates a state change. When you update your state with Redux, you always start with an action. Actions are in the form of Javascript objects, containing a type and an optional payload . Actions are sent using the**store.dispatch()** method. Actions are created via an action creator.

Reducers in Redux - Reducers are pure functions that take the current state of an application, perform an action, and return a new state. These states are stored as objects, and they

combine multiple reducers: The combineReducers() helper function turns an object whose values are different reducing functions into a single reducing function you can pass to createStore.

Store in Redux - A Store is an object that holds the whole state tree of your application. The Redux store is the application state stored as objects. Whenever the store is updated, it will update the React components subscribed to it. The store has the responsibility of storing, reading, and updating state.

Dispatch - Dispatch is a method that triggers an action with type and payload to Reducer.

Subscribe - Subscribe is a method that is used to subscribe data/state from the Store.

Provider - The Provider is a component that has a reference to the Store and provides the data from the Store to the component it wraps.

Connect - Connect is a function that communicates with the Provider.

Middleware - Middleware is the suggested way to extend Redux with custom functionality. Middlewares are used to dispatch async functions. We configure Middleware's while creating a store.

Redux Core Principal:

Single Source of Truth: The state of your whole application is stored in an object tree within a single store

State Is Read-Only: The only way to change the state is to dispatch an action, an object describing what happened.

Changes Are Made With Pure Functions: To specify how the state tree is transformed by actions, you write pure reducers.

https://www.valentinog.com/blog/redux/

To view redux store. type window.store.getStore() in console tab.

Add articles to Redux Store. type in console tab. `store.dispatch( addArticle( curly brace title: 'React Redux Tutorial for Beginners', id: 1 curly brace close) );`

Footer paragraph from parent props