line breaks removal : https://www.textfixer.com/tools/remove-line-breaks.php

#region What are lifecycleMethods?

You can think of React lifecycle methods as the series of events that happen from the birth of a React component to its death.

Mounting – Birth of your component

Update – Growth of your component

Unmount – Death of your component

ALL THE LIFECYCLE METHODS CAN BE CALLED W.R.T ASYNC/AWAIT PARADIGM

https://www.digitalocean.com/community/tutorials/react-replacing-component-lifecycles-with-useeffect

#endregion

Lifecycle method diagram

https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

mount: constructor,static getDerivedStateFromProps,render,componentDidMount

update:static getDerivedStateFromProps,shouldComponentUpdate,render,componentDidUpdate

unmount:compnentWillUmount

#region MOUNT PHASE

Mounting Phase Life cycle hooks

constructor()

static getDerivedStateFromProps()

render()

componentWillMount()//Deprecated react 16.4 version unsafe to use

componentDidMount()

#region CTOR

constructor should always be a pure function, no side-effects i.e no ajax calls with in ctors, should not modify state , they have to be made in componentDidMount(), pure functions are idempotent call once or 10 times has to return the same output.

what are pure functions? Explain and then move onto ctor

constructor : Is used for two main operations

a) To intialise the values, state values also

b) To bind this to class methods like handlers

SCENARIOS :

Scenario 1: When ctor is not supplied to class

class CtorComponent extends Component{

render(){ return <h1>This is CtorComponent {this.props.name}</h1> } }

In this scenario, react will supply ctor(props){super(props)} to your class out of the box [Try in Babel TryOut]

you compiled class will be

class CtorComponent extends Component{

ctor(args){super(args)} //here args are props

render(){ return <h1>This is CtorComponent {this.props.name}</h1> } }

https://medium.com/@swsinswsin/react-js-state-without-constructor-fixing-954dbb11fcf1

//Scenario 2: ideal scenario for ctor

class CtorComponent extends Component{

ctor(props){super(props); console.log(this.props)}

render() { console.log(this.props); return <h1>Ajay</h1>}

}

//Scenario 3: ctor no props passed to super

Props in render are supplied by react framework, if props are not passed they will not be available only in ctor, other lifecycle methods will have access to this.props

class CtorComponent extends Component{

constructor(){ //NO PROPS ARE GIVEN

super(); //NO PROPS ARE PASSED

console.log(this.props) //Undefined

}

render() {

console.log(this.props); //defined {name:'Ajay'}

return <h1>Ajay</h1>

}

}

scenario 4: ctor with props, super with no props

If you want to make use of this.props inside your constructor , then only you need to pass props to super(), else not needed. Any way "props" is still available

class CtorComponent extends Component{

constructor(props){ //NO PROPS ARE GIVEN

super(); //NO PROPS ARE PASSED

console.log(this.props) //Undefined

console.log(props) //defined {name:'Ajay'}

}

render() {

console.log(this.props); //defined {name:'Ajay'}

return <h1>Ajay</h1>

}

}

Key points to note :

------------------

ES6 Rule for class, YOU CANNOT USE THIS IN A CLASS UNTIL SUPER IS CALLED.

1) If you want to make use of this.props inside your constructor , then only you need to pass props to super(), else not needed. Any way "props" is still available(scenario 4)

passing or not passing props to super has no effect on later uses of this.props outside constructor. That is render, shouldComponentUpdate, or event handlers always have access to it.(scenario 3 and scenario 4)

2) If in React class you are not passing the props in super(), "this.props" will remain undefined only in ctor, for other life cycle methods like componentDidMount(),render() etc this.props is present and defined, its supplied by react

What is return type of super() ?

constructor(){

let res = super() //returns the class itself, if you pass props, this will have props defined else undefined

console.log("res",res) //same as this

console.log(res === this) //true , this also will have defined props if props passed to super()

}

Is it possible to write constructor less classes in react/Js ?

Yes, From ES.Next onwards based on ES.next class properties proposal https://hackernoon.com/the-constructor-is-dead-long-live-the-constructor-c10871bea599

https://daveceddia.com/where-initialize-state-react/#:~:text=So%2C%20is%20it%20ever%20ok,the%20component's%20internally%2Dcontrolled%20state.

By default CRA babel comes with class property transform, we can start using without ctor.

Why is ctor/render is called twice in development mode??

ctor/render and other functions(check the link) methods are pure functions, when in <React.StrictMode> these functions are called twice, to double check for side effects. https://reactjs.org/docs/strict-mode.html#detecting-unexpected-side-effects

When these methods are called twice you need to see the same result, if the results are different, it means there are side effects, this is been done out of the box by <React.StrictMode>

NOTE: <React.StrictMode> only applies to development mode. Lifecycles will not be double-invoked in production mode.

Note:

Conceptually, React does work in two phases:

The render phase determines what changes need to be made to e.g. the DOM. During this phase, React calls render and then compares the result to the previous render.

The pre-commit stage : Can read the DOM at this stage

The commit phase is when React applies any changes. (In the case of React DOM, this is when React inserts, updates, and removes DOM nodes.) React also calls lifecycles like componentDidMount and componentDidUpdate during this phase.

commit phase is fast, where as render phase is slow hence react breaks up render into small parts and execute thus avoid blocking the browser.

Is it good Practise to initialise state with props?

So, is it ever ok to initialize state based on props? Yes. The original version of the React docs mentioned this:

However, it’s not an anti-pattern if you make it clear that the prop is only seed data for the component’s internally-controlled state.

Think of it this way: it’s fine if the state needs a starting value which the component will then control.

Ask yourself: does this component “own” the data?

Does it just need a jump-start from a prop?

Those are good reasons to initialize state from a prop.

you can assign a prop to state only when the prop is a seed value/default value

Why state should not be given props when they change ?

https://blog.alexdevero.com/react-best-practices-pt1/#5-don8217t-use-props-in-initial-state

assume state is intialised with props , this happens in ctor, only when the component is mounted. After the component mount , if props are changed then , the values of prop are not updated in the intialised state, this may cause buggy code. Hence in that case we need to use componentDidUpdate lifecycle hook.

#endregion CTOR

#region static getDerivedStateFromProps(nextProps,prevState) is replacement for componentWillRecieveProps(nextProps)

https://www.digitalocean.com/community/tutorials/react-get-derived-state

**\*\*\*getDerivedStateFromProps is invoked whenever a component receives new props.**

1) **getDerivedStateFromProps** is invoked **right before calling the render method**, both on the initial mount and on subsequent UPDATES, i.e on subsequent calls if there are no updates to props getDerivedStateFromProps will not be called.

2) It should return an object to update the state, or null to update nothing. UPDATES THE STATE AND REPLACE THE STATE

3) this method is fired on every render, regardless of the cause.

4) getDerivedStateFromProps is introduced from 16.3, to replace componentWillRecieveProps , which is unsafe and not asynchronous.

5) getDerivedStateFromProps + componentDidUpdate together can cover all the usecases related to componentWillRecieveProps(nextProps)

6) We cannot access 'this' in getDerivedStateFromProps hence we get updated props as nextProps and this.state as 'prevState'

Consider the following scenario:

//Parent

class A { state={label:'XYZ'} componentDidMoun(){this.setState({label:'PQRST'})} render() {<B label={this.state.label}/>}}

class B { state={label:this.props.label} //\*\*

static getDerivedStateFromProps(nextProps,prevState){ if(nextProps.label !== prevState.label){return

{label:nextProps.label}//RE-ASSIGN,NOT MERGE}

//0r {...prevState,label:nextProps.label}

}

render() {.....}

}

In the above example you have initialized state with props in component B , this happens in ctor() highlighted as \*\*, (babel transforms to ctor)

ctor code executes only once during component lifecycle during mount, the other times whenever there is an update in props in class A, B will get updated only using getDerivedStateFromProps().

Why need to update to new LifeCycle

Firstly to get advantage of new asynchronous nature of React and further more old lifecycle methods have often been misunderstood and subtly misused.

When you call action from componentWillReceiveProps and that action do some change in store then corresponding call component’s componentWillReceiveProps, so it almost lead to infinite loop and many useless render call, until it being used carefully.

Similarly calling setState in componentWillUpdate will trigger shouldComponentUpdate and then again componentWillUpdate, which is also lead to infinite methods call, until it being used carefully.

https://medium.com/hackernoon/replacing-componentwillreceiveprops-with-getderivedstatefromprops-c3956f7ce607#:~:text=getDerivedStateFromProps%20is%20one%20of%20those,when%20it%20receives%20new%20props.

Find more info on this...

https://itnext.io/how-to-updating-state-on-prop-changes-2296a03ef08c

#endregion

#region render

The render() method is the only required method in a class component.render should be a PURE function like CTOR

render() will not be invoked if shouldComponentUpdate() returns false.

Render it should examine this.props and this.state and return following :

1) React elements

2) Arrays and Fragments

3) Portals\*\*(different topic)

4) Boolean and nulls

5) String and numbers

render should be pure, it should not change the state of the component, it should return same output, when supplied with same input anytime.

render doesn't interact with browser, if you want interaction with browser, that should be accomplished using componentDidMount and other lifecycle methods.

render() returns React elements

-------------------------------

render(){<CustomComponent label='Ajay'/>}

render() returns Fragments and Arrays

---------------------------

render(){<React.Fragment> <ComponentA/> <ComponentB/> <ComponentC/> </React.Fragment>}

Short hand syntax : render(){<> <ComponentA/> <ComponentB/> <ComponentC/> </>}

Note shorthand syntax for React.Fragment is <>...</>

Fragments let you group a list of children without adding extra nodes to the DOM.

Usecase for Fragments , consider following

//Parent render

render(){<table><tr> <Columns/> </tr></table>}

//Child(Columns) render

render(){<div><td>FirstName</td> <td>LastName</td> </div>}

But above example will lead to invalid table rendering because div is not a valid element inside <tr>

In this case we can use <React.Fragment> or(</>) instead of <div>, which will not add any extra dom element and makes the table render valid.

ReactFragment with an array:

<dl>

{props.items.map(item => (

// Without the `key`, React will fire a key warning

<React.Fragment key={item.id}> //WITHOUT KEY REACT WILL THROW A WARNING

<dt>{item.term}</dt>

<dd>{item.description}</dd>

</React.Fragment>

))}

</dl>

NOTE <dl> is a description List element : https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dl#:~:text=The%20HTML%20element%20represents,of%20key%2Dvalue%20pairs).

render() with portals:

https://css-tricks.com/using-react-portals-to-render-children-outside-the-dom-hierarchy/

\*\*\*\*https://codeburst.io/reacts-portals-in-3-minutes-9b2efb74e9a9 same example in functional way

If there need be to render outside the react dom, then we can use portal, its mostly used for Modals

Tooltips , where over-flow:hidden etc

render() with primitives Number boolean strings null work as expected

Can I update state in render()?

No , render should be pure. React throws a warning.

render() {

// BAD: Do not do this!

this.setState({ foo: 'bar' });

return (

<div className={ classNames('person', this.state.mode) }>

{ this.props.name } (age: { this.props.age })

</div>

);

}

#endregion

#region componentWillMount DEPRECATED

#endregion

#region componentDidMount(), doesn't have any arguments

references: https://linguinecode.com/post/understanding-react-componentdidmount

componentDidMount() is invoked immediately after a component is mounted (inserted into the tree). Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request.

ComponentDidMount is an impure function, it can cause side effects, It can contain external calls , once DOM is rendered in render().

componentDidMount will be executed after the render method , It is called ONCE in the component life cycle.It will not be called in subsequent re-renders.

To test above statement

class ComponentDidMountDemo extends Component{

constructor(props){

super(props)

this.state={ label:'DEFAULT' }

}

componentDidMount(){

console.log('componentDidMount',this.props)

//this will cause a render to trigger again

this.setState((prevState) =>({...prevState,label:'componentDidMount, WILL NOT BE CALLED after setState render'}))

}

render(){

console.log('render',this.state)

return <h1>This is ComponentDidMountDemo</h1>

}

}

In the above example the componentDidMount will not be called for every render.

Is there a scenario where componentDidMount be called multiple times ??

Yes they can be called multiple time when the child component in parent has a key property, which changes. Each time a new key value is assigned, a new child component is rendered , this will trigger ComponentDidMount again and again.

Eg:

//Parent Component

class ComponentDidMountMultipleParent extends Component{

constructor(props){

super(props)

this.state={hits:0}

}

handleClickHandler(){ this.setState((prevState)=>({hits:prevState.hits+1}))}

render(){

return (<div><button onClick={()=>this.handleClickHandler()}>Increment</button> <ComponentDidMountMultipleChild hits={this.state.hits} key={this.state.hits}/>

</div>

)

}

}

//Child Component

class ComponentDidMountMultipleChild extends React.Component{

componentDidMount() {console.log('componentDidMount called')}

render(){return <h1>{this.props.hits} times clicked</h1>}

}

export default ComponentDidMountMultipleChild

NOTE: In the above example when key prop is removed from Child , componentDidMount will be executed only once

componentDidMount is used for side-effects code like API, external integrations etc

Eg: The below timeout will be called only once , and not on every second elapsed

class ComponentDidMountAPI extends React.Component{

state={ hits:0 }

componentDidMount(){

//Psudo code for API call

setInterval(()=>{

return this.setState((prevState)=>({hits:prevState.hits+1}))

},1000)

console.log('componentDidMount')

}

render(){ return (<h1>{this.state.hits}</h1>) }

}

ComponentDidMount will work reverse of render order.

https://developmentarc.gitbooks.io/react-indepth/content/life\_cycle/birth/post\_mount\_with\_component\_did\_mount.html

Child\_componentDidMount -->parent\_ComponentDidMount()

parent\_render() --> child\_render()

Once ComponentDidMount of parent is executed(which is last), then the changed state is passed onto all the child elements, this is where we get to use

static getDerivedStateFromProps(nextProps,prevState), where the state for each of the child component can be updated.

https://engineering.musefind.com/react-lifecycle-methods-how-and-when-to-use-them-2111a1b692b1 , explains the above point

#endregion

#endregion MOUNT PHASE

#region UPDATE PHASE

Update Phase LifeCycle Hooks:

1)static getDerivedStateFromProps(also called in MOUNT Phase)

2) shouldComponentUpdate

3) render

4) componentDidUpdate

#region static getDerivedStateFromProps(nextProps,prevState) is replacement for componentWillRecieveProps(nextProps)

https://www.digitalocean.com/community/tutorials/react-get-derived-state

NOTE: getDerivedStateFromProps SHOULD NOT have any side-effects.

1) getDerivedStateFromProps is invoked right before calling the render method, both on the initial mount and on subsequent UPDATES, i.e on subsequent calls if there are no updates to props getDerivedStateFromProps will not be called.

2) It should return an object to update the state, or null to update nothing. UPDATES THE STATE , REPLACES THE STATE. IT DOESN't MERGE THE STATE.

3) this method is fired on every render, regardless of the cause.

4) getDerivedStateFromProps is introduced from 16.3, to replace componentWillRecieveProps , which is unsafe and not asynchronous.

5) getDerivedStateFromProps + componentDidUpdate together can cover all the usecases related to componentWillRecieveProps(nextProps)

6) We cannot access 'this' in getDerivedStateFromProps hence we get updated props as nextProps and this.state as 'prevState'

Consider the folowing scenario:

//Parent

class A { state={label:'XYZ'} componentDidMoun(){this.setState({label:'PQRST'})} render() {<B label={this.state.label}/>}}

class B { state={label:this.props.label} //\*\*

static getDerivedStateFromProps(nextProps,prevState){ if(nextProps.label !== prevState.label){return {label:nextProps.label}//RE-ASSIGN,NOT MERGE}}

render() {.....}

}

In the above example you have initialized state with props in component B , this happens in ctor() highlighted as \*\*, (babel transforms to ctor)

ctor code executes only once during component lifecycle during mount, the other times whenever there is an update in props in class A, B will get updated only using getDerivedStateFromProps

Why need to update to new LifeCycle

Firstly to get advantage of new asynchronous nature of React and further more old lifecycle methods have often been misunderstood and subtly misused.

When you call action from componentWillReceiveProps and that action do some change in store then corresponding call component’s componentWillReceiveProps, so it almost lead to infinite loop and many useless render call, until it being used carefully.

Similarly calling setState in componentWillUpdate will trigger shouldComponentUpdate and then again componentWillUpdate, which is also lead to infinite methods call, until it being used carefully.

https://medium.com/hackernoon/replacing-componentwillreceiveprops-with-getderivedstatefromprops-c3956f7ce607#:~:text=getDerivedStateFromProps%20is%20one%20of%20those,when%20it%20receives%20new%20props.

Find more info on this...

https://itnext.io/how-to-updating-state-on-prop-changes-2296a03ef08c

NOTE : getDerivedStateFromProps

#endregion getDerivedStateFromProps

#region shouldComponentUpdate(nextProps, nextState)

shouldComponentUpdate is means through which React will know ,component’s output is not affected by the current change in state or props.

shouldComponentUpdate is called whenever the props or state changes, so the parameters are nextProps,nextState.

The default behaviour is, whenever the state changes(i.e setState) occurs re-render happens , shouldComponentUpdate runs on every RE-RENDER. For initial Render and forceUpdate() is used shouldComponentUpdate is not called.

if shouldComponentUpdate is not implemented it is set to always return true by default, which means on every prop & state change your component will re-render.

NOTE: The sole purpose of shouldComponentUpdate is to optimize the performance and not reduce the number of re-renders.

To skip the re-renders we need to use PURE COMPONENT instead of shouldComponentUpdate.

PureComponent performs a shallow comparison of props and state, and reduces the chance that you’ll skip a necessary update.

If you are confident we can write ShouldComponentUpdate and return null after comparing this.props Vs nextProps and this.state Vs nextState

We do not recommend doing deep equality checks or using JSON.stringify() in shouldComponentUpdate(). It is very inefficient and will harm performance

Note that returning false does not prevent child components from re-rendering when their state changes.

if shouldComponentUpdate() returns false, then UNSAFE\_componentWillUpdate(), render(), and componentDidUpdate() will not be invoked

first getDerivedStateFromProps is called , then shouldComponentUpdate is called

eg:

//Parent component

import React from "react"; import SimplifyShouldComponentUpdateChild from './SimplifyShouldComponentUpdateChild' class SimplifyShouldComponentUpdate extends React.Component { state = { name: "Ajay", id: "72", title: "Sr.Software Engineer", salary: "1000", };

componentDidMount() { this.setState({ title: "Lead Engineer", salary: "1500" }); }

render() { return ( <SimplifyShouldComponentUpdateChild {...this.state}/> ); } } export default SimplifyShouldComponentUpdate;

//Child Component

import React from 'react' class SimplifyShouldComponentUpdateChild extends React.Component {

shouldComponentUpdate(nextProps, nextState) {

console.log('shouldComponentUpdate')

const isTitleChange =! this.props.title !== nextProps.title;

const isSalaryChange =! this.props.salary !== nextProps.title;

return isTitleChange || isSalaryChange;

//return false

}

render() { console.log(this.props) return ( <div> <div>{this.props.name}</div> <div>{this.props.id}</div> <div>{this.props.title}</div> <div>{this.props.salary}</div> </div> ); } } export default SimplifyShouldComponentUpdateChild

What is PureComponent?

Gotchas are also covered here.

https://lucybain.com/blog/2018/react-js-pure-component/

shouldComponentUpdate :

https://lucybain.com/blog/2017/react-js-when-to-rerender/

#endregion shouldComponentUpdate(nextProps, nextState)

#region render

#endregion render

#region componentDidUpdate(prevProps, prevState, snapshot)

https://www.newline.co/@dmitryrogozhny/using-componentdidupdate-in-react--f037b5aa

componentDidUpdate is invoked immediately after updating occurs. componentDidUpdate method is not called for the initial render.

The componentDidUpdate gets called after a render and componentDidMount, hence

componentDidUpdate is used for operations on DOM, side-effects causing api calls if the props are changed.

Eg:

componentDidUpdate(prevProps,prevState) {

// Typical usage (don't forget to compare props):

if (this.props.userID !== prevProps.userID) {

this.fetchData(this.props.userID);

}

}

You may call setState() immediately in componentDidUpdate() but note that it must be wrapped in a condition like in the example above, or you’ll cause an infinite loop.

componentDidUpdate()is called after componentDidMount() and can be useful to perform some action when the state changes.

When do we use componentDidUpdate()?

componentDidUpdate() is when we need to call an external API on condition that the previous state and the current state have changed. The call to the API would be conditional to the state being changed. If there is no state change, no API is called.

For DOM manipulation.

---------------------

class MyComponent extends React.Component {

state = { size: 100 };

boxRef = React.createRef();

handleIncrementClick = () => {

// set new state to force update

this.setState((state, props) => ({ size: state.size + 10 }));

};

componentDidUpdate(prevState, prevProps) {

// react to update and change the HTML element's size

this.boxRef.current.style.width = `${this.state.size}px`;

this.boxRef.current.style.height = `${this.state.size}px`;

}

render() {

return (

<div>

<button onClick={this.handleIncrementClick}>Resize</button>

<div style={{backGroundColor:red}} ref={this.boxRef}></div>

</div>

);

}

}

API Way:

See the example codes

//Parent Component

//ComponentDidUpdateDemoParent import React from "react"; import ComponentDidUpdateDemo2Child from './ComponentDidUpdateDemo2Child'

const options = [ { key: null, name: null }, { key: "1", name: "Emp1" }, { key: "2", name: "Emp2" }, ];

class ComponentDidUpdateDemoParent extends React.Component {

state = { empValue: null };

onChangeHandler = (event) => { this.setState({ empValue: event.target.value }); };

findName = (optKey) => { return options.find(({ key }) => key === optKey)?.name }

render() { return ( <div> <div><label>Select an Employee</label> </div> <select onChange={this.onChangeHandler}> {options.map(({ key, name }) => { return ( <option key={key} value={key}> {name} </option> ); })} </select> { this.state.empValue &&(<div> Selected Value : {this.findName(this.state.empValue)} <div><ComponentDidUpdateDemo2Child id={this.state.empValue}/></div> </div>)} </div> ); } }

export default ComponentDidUpdateDemoParent;

//Child Component

import React from "react";

class ComponentDidUpdateDemo2Child extends React.Component {

state = { projects: [] };

fetchProjects(id){ console.log({id}) if(!id) return null;

//PSUEDO API CALL

setTimeout(()=>{ this.setState({ projects: id === "1" ? ["Alpha", "Beta"] : ["Charlie", "Delta"], }); },300) }

componentDidMount(){ console.log("componentDidMount is called "); this.fetchProjects(this.props.id) }

componentDidUpdate(prevProps,prevState) { if (prevProps.id !== this.props.id) { console.log("componentDidUpdate is called "); this.fetchProjects(this.props.id) } }

render() { return this.props.id && ( <div> ComponentDidUpdateDemo2Child {this.state.projects.map((project) => { return <div>{project}</div> })} </div> ); } }

export default ComponentDidUpdateDemo2Child;

#endregion componentDidUpdate(prevProps, prevState, snapshot)

#endregion UPDATE PHASE

#region UNMOUNT PHASE

#region componentWillUnmount()

1) componentWillUnmount() is invoked immediately before a component is unmounted and destroyed

2) Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount

3) You should not call setState() in componentWillUnmount() because the component will never be re-rendered. Once a component instance is unmounted, it will never be mounted again.

#endregion componentWillUnmount()

#endregion UNMOUNT PHASE

#region LifeCycle Methods to HOOKS

Refer Session2\_Hooks\_Functions\_Part2.txt

#endregion