**Arbind Chaudhary**

# **Lifecycle of Components**

Lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component’s existence.

· **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.

· **Mounting:** Mounting is the stage of rendering the JSX returned by the render method itself.

1. constructor()

2. getDerivedStateFromProps()

3. render()

4. componentDidMount()

· **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.

1. getDerivedStateFromProps()

2. shouldComponentUpdate()

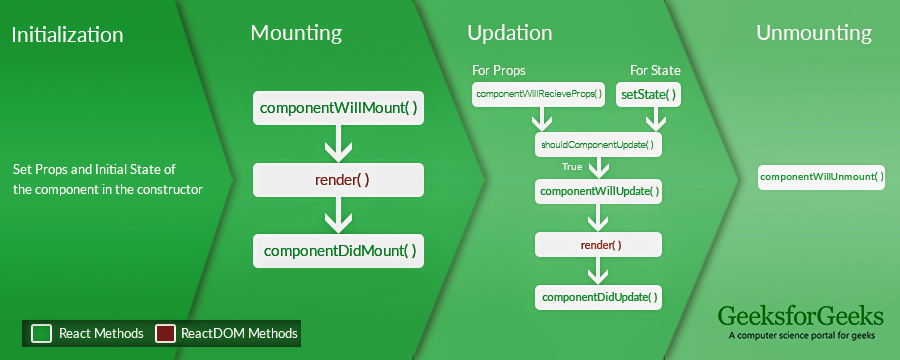
3. render()

4. getSnapshotBeforeUpdate()

5. componentDidUpdate()

· **Unmounting:** As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

1. componentWillUnmount()



**Mountin**

· **Constructor**

*constructor(props)*

The constructor is a method that’s automatically called during the creation of an object from a class. It can handle your initial setup stuff like defaulting some properties of the object, or sanity checking the arguments that were passed in. Simply put, the constructor aids in constructing things.

*import React, { Component } from 'react';*

*class App extends Component {*

*constructor(props){*

*super(props);*

*this.state = {*

*data: 'ConsultAdd'*

*}*

*this.handleEvent = this.handleEvent.bind(this);*

*}*

*handleEvent(){*

*console.log(this.props);*

*}*

*render() {*

*return (*

*<div className="App">*

*<h2>React Constructor</h2>*

*<input type ="text" value={this.state.data} />*

*<button onClick={this.handleEvent}>Please Click</button>*

*</div>*

*);*

*}*

*}*

*export default App;*

*import React from 'react';*

*import ReactDOM from 'react-dom';*

*import App from './App.js';*

*ReactDOM.render(<App />, document.getElementById('app'));*

· **getDerivedStateFromProps()**

method is called right before rendering the element(s) in the DOM. This is the natural place to set the state object based on the initial props. It takes state as an argument, and returns an object with changes to the state.

*class Header extends React.Component {*

*constructor(props) {*

*super(props);*

*this.state = {food: "Chapati"};*

*}*

*static getDerivedStateFromProps(props, state) {*

*return {food: props.food };*

*}*

*render() {*

*return (*

*<h1>My Favorite food is {this.state.food}</h1>*

*);*

*}*

*}*

*ReactDOM.render(<Header food="Noodles"/>, document.getElementById('root'))*

· **Render**

The render method outputs HTML to DOM.

*const element = <h1>ConsultAdd</h1>;*

*ReactDOM.render(element, document.getElementById('root'));*

· **componentDidMount**

After all the elements of the page is rendered correctly, this method is called. After the markup is set on the page, this technique called by React itself to either fetch the data from An External API or perform some unique operations which need the JSX elements.

*import React, { Component } from 'react';*

*class App extends Component {*

*constructor(props){*

*super(props);*

*this.state = {*

*data: 'Jordan Belfort'*

*}*

*}*

*getData(){*

*setTimeout(() => {*

*console.log('Our data is fetched');*

*this.setState({*

*data: 'ConsultAdd'*

*})*

*}, 1000)*

*}*

*componentDidMount(){*

*this.getData();*

*}*

*render() {*

*return(*

*<div>*

*{this.state.data}*

*</div>*

*)*

*}*

*}*

*export default App;*

**Updating**

· **getDerivedStateFromProps**

getDerivedStateFromProps is invoked every time a component is rendered. It takes in two arguments: the next props object (which may be the same as the previous object) and the previous state object of the component in question. When implementing this method, we need to return the changes to our component state or null (or {}) if no changes need to be made.

*class Header extends React.Component {*

*constructor(props) {*

*super(props);*

*this.state = {favoritecolor: "red"};*

*}*

*static getDerivedStateFromProps(props, state) {*

*return {favoritecolor: props.favcol };*

*}*

*changeColor = () => {*

*this.setState({favoritecolor: "blue"});*

*}*

*render() {*

*return (*

*<div>*

*<h1>My Favorite Color is {this.state.favoritecolor}</h1>*

*<button type="button" onClick={this.changeColor}>Change color</button>*

*</div>*

*);*

*}*

*}*

*ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));*·

**shouldComponentUpdate**

A shouldComponentUpdate method, called with nextProps as the first argument, and nextState is the second. shouldComponentUpdate should always return a boolean — an answer to the question, “should I re-render?” Yes, little component, you should. The default is that it always returns true.

*class App extends React.Component {*

*constructor() {*

*super();*

*this.state = {*

*value: true,*

*countOfClicks: 0*

*};*

*this.pickRandom = this.pickRandom.bind(this);*

*}*

*pickRandom() {*

*this.setState({*

*value: Math.random() > 0.5, // randomly picks true or false*

*countOfClicks: this.state.countOfClicks + 1*

*});*

*}*

*// comment out the below to re-render on every click*

*shouldComponentUpdate(nextProps, nextState) {*

*return this.state.value != nextState.value;*

*}*

*render() {*

*return (*

*<div>*

*shouldComponentUpdate demo*

*<p><b>{this.state.value.toString()}</b></p>*

*<p>Count of clicks: <b>{this.state.countOfClicks}</b></p>*

*<button onClick={this.pickRandom}>*

*Click to randomly select: true or false*

*</button>*

*</div>*

*);*

*}*

*}*

*ReactDOM.render(*

*<App />,*

*document.getElementById('app')*

*);*

· **getSnapshotBeforeUpdate**

getSnapshotBeforeUpdate gets called before the most recent render() output. It allows the engineer to capture some information about the DOM before it’s changed. This method should return a value or null.

*import React from 'react';*

*import ReactDOM from 'react-dom';*

*class ChatLists extends React.Component{*

*constructor(props){*

*super(props);*

*this.chatListRef = React.createRef();*

*}*

*getSnapshotBeforeUpdate(prevProps, prevState) {*

*if (prevProps.chatList.length < this.props.chatList.length) {*

*const chatList = this.chatListRef.current;*

*return chatList.scrollHeight - chatList.scrollTop;*

*}*

*return null;*

*}*

*componentDidUpdate(prevProps, prevState, snapshot) {*

*if (snapshot) {*

*const chatList = this.chatListRef.current;*

*chatList.scrollTop = chatList.scrollHeight - snapshot;*

*}*

*}*

*render() {*

*return (*

*{*

*}*

*);*

*}*

*}*

· **componentDidUpdate**

componentDidUpdate()is called after componentDidMount() and can be useful to perform some action when the state changes. componentDidUpdate() takes as its first two arguments the previous props and the previous state. Inside the method we can check if a condition is met and perform an action based on it.

*export default class Task extends React.Component {*

*constructor(props, context) {*

*super(props, context);*

*this.state = {*

*name: "",*

*age: "",*

*country: ""*

*};*

*}*

*componentDidUpdate() {*

*this.\_commitAutoSave();*

*}*

*\_changeName = (e) => {*

*this.setState({name: e.target.value});*

*}*

*\_changeAge = (e) => {*

*this.setState({age: e.target.value});*

*}*

*\_changeCountry = (e) => {*

*this.setState({country: e.target.value});*

*}*

*\_commitAutoSave = () => {*

*Ajax.postJSON('/someAPI/json/autosave', {*

*name: this.state.name,*

*age: this.state.age,*

*country: this.state.country*

*});*

*}*

*render() {*

*let {name, age, country} = this.state;*

*return (*

*<form>*

*<input type="text" value={name} onChange={this.\_changeName} />*

*<input type="text" value={age} onChange={this.\_changeAge} />*

*<input type="text" value={country} onChange={this.\_changeCountry} />*

*</form>*

*);*

*}*

*}*

**Unmounting**

· **componentWillUnmount**

componentWillUnmount is the last function to be called immediately before the component is removed from the DOM. It is generally used to perform clean-up for any DOM-elements or timers created in componentWillMount.

*import React from "react"*

*export default class App extends React.Component {*

*isMounted = false*

*componentDidMount(){*

*this.isMounted = true*

*if (this.isMounted){*

*this.setState({'anyState': anyState}) //call setState if component isMounted*

*}*

*}*

*componentWillUnmount(){*

*this.isMounted = false*

*}*

*render(){*

*return(*

*<div />*

*)*

*}*

*}*