

Deep Dive on the React Lifecycle

A photograph of a family of four sitting on a wooden dock. From left to right: a woman in a red and white striped tank top, a young child in a patterned onesie, a man in a grey and black striped polo shirt, and a young girl with blonde hair tied back. They are all smiling at the camera. The background shows a calm lake and some trees.

whoami

Jonathan Creamer

whoami

- Currently Senior Front End Engineer at [Lonely Planet](#)
- Past JavaScript Engineer appendTo
- Nashville, TN
- Love JavaScript, tweet at [@jcreamer898](#), blog at [jonathancreamer.com](#)
- [Microsoft MVP](#)



Agenda

1. What is a React component?
2. Lifecycle methods
3. Use cases
4. Testing

Pure Functions

A function is pure if...

Given the same input, will always return the same output.

Produces no side effects.

Relies on no external mutable state.

- One way to build a React component is as a pure function

Can you even Component bro?

```
function HelloWorld({  
  text  
) {  
  return (  
    <h1>{text}</h1>  
  );  
}
```

```
ReactDOM.render(<HelloWorld text="hello world" />, document.body);
```

- Might be all you need
- Simple
- State forced to higher level container components

JSX to JS

```
function HelloWorld(_ref) {  
  var text = _ref.text;  
  
  return React.createElement(  
    "h1",  
    null,  
    text  
  );  
}
```

- JSX is an abstraction over creating element trees
- Different renderers like ReactDOM
- lowercase names refer to built-in components
- Capitalized names refer to custom components

React Components as Classes

```
class HelloWorld extends React.Component {  
  render() {  
    const { text } = this.props;  
    return (  
      <h1>{text}</h1>  
    );  
  }  
}
```

- If all you have is render, stay functional
- An abstract base class meant to be extended
- Don't screw with props, they're Read Only
- Sometimes we need some state

Lifecycle Methods

Mounting, Updating, Unmounting

Mounting

- `constructor()`
- `componentWillMount()`
- `render()`
- `componentDidMount()`

Updating

- `componentWillReceiveProps()`
- `shouldComponentUpdate()`
- `componentWillUpdate()`
- `render()`
- `componentDidUpdate()`

Unmounting

- `componentWillUnmount()`

constructor

constructor aka init

```
constructor(props) {  
  super(props);  
  this.state = {};  
}
```

- Perform any initial setup
- Called once per mounted component
- Initialize state
- Must call super(props)

State

- What's going on around here right now?
- Has a user clicked or typed anything?
- Any data needing to be fetched?
- Any stored information?

setState

```
class WYSIWYG extends React.Component {  
  update() { /*...*/ }  
  render() {  
    const { text } = this.state;  
    return (  
      <textarea  
        onChange={this.update}  
        value={text}  
      />  
    );  
  }  
}
```

- Pull from state
- A lot of things happen when you do this...

setState

```
constructor(props) {
  super(props);

  this.state = {
    text: "",
  };

  this.update = this.update.bind(this);
}
update(e) {
  const text = e.target.value;
  this.setState({ text });
}
```

- Must bind for this to work
- After setState, render will fire
- Do NOT update state w/ this.state.text = "foo";

async setState

```
// WRONG
this.setState({ count: this.state.count + props.count });
doSomething(this.state.count);
```

```
// RIGHT
this.setState((prevState, props) => ({
  counter: prevState.count + props.count
}));
```

- Don't rely on current state when computing

componentWillMount

aka beforeDomReady

componentWillMount

- props and state both ready
- Can use for calling setState
- Most times just use constructor

componentDidMount

aka onDomReady

componentDidMount

- jQuery plugin time :trollface:
- DOM is ready here
- Stand up plugins
- ref is now a function
- Dispatch actions

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor(props) {
    super(props);
    this.state = { /* init state */ };
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = {};
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = {};
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentDidMount

```
import ace from "aceeditor";

export default class Editor extends React.Component {
  constructor() {
    super();
    this.state = {};
  }
  componentDidMount() {
    this.editor = ace.edit(this.$text);
  }
  render() {
    return (
      <div
        ref={(node) => this.$text = node}
      />
    )
  }
}
```

componentWillUnmount

aka destroy

Be a good citizen

```
componentWillUnmount() {  
  this.editor.destroy();  
}
```

- Remove any event handlers or plugins
- No leaks

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

```
class Chat extends Component {  
  constructor(props) {  
    super(props);  
    this.state = { messages: [] };  
  }  
  componentDidMount() {  
    this.subscription = postal.subscribe({  
      topic: "message.added",  
      callback: (message) => {  
        this.setState({  
          messages: [...this.state.messages, message]  
        })  
      }  
    });  
  }  
  componentWillUnmount() {  
    this.subscription.unsubscribe();  
  }  
  render() {  
    return (  
      <Messages messages={this.state.message} />  
    )  
  }  
}
```

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

componentWillUnmount

```
class Chat extends Component {
  constructor(props) {
    super(props);
    this.state = { messages: [] };
  }
  componentDidMount() {
    this.subscription = postal.subscribe({
      topic: "message.added",
      callback: (message) => {
        this.setState({
          messages: [...this.state.messages, message]
        })
      }
    });
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() {
    return (
      <Messages messages={this.state.message} />
    )
  }
}
```

IRL

IRL

The screenshot shows a web browser window with the Lonely Planet logo at the top left. To its right are navigation links: Search, Video, Destinations (which is highlighted in blue), Bookings, Shop, and Sign In. Below these links is a breadcrumb navigation path: North America > USA > Nashville. The main content area has a large heading "Restaurants" with a dropdown arrow icon to its right. To the right of the heading is a "TYPE" filter section containing eight categories with checkboxes: American, Southern US, Modern American, Mexican, Bakery, Cafe, Sandwiches, Market, Breakfast, and Diner. A "MORE CATEGORIES" link is located below the checkbox grid. On the left side of the main content area, there is a card for a restaurant named "Bastion". The card includes the text "TOP CHOICE MODERN AMERICAN IN SOUTH NASHVILLE" and a brief description: "Bastion's nondescript entrance – within the eminently cool bar of the same name inside...".

- ## San Francisco Restaurants

componentDidMount IRL

```
export default class PoiDetail extends React.PureComponent {  
  componentDidMount() {  
    if (!this.props.poi) {  
      this.fetchPoi(this.props.params.id);  
    }  
  
    if (!this.props.related) {  
      this.fetchRelated(this.props.params.id);  
    }  
  }  
}
```

- List is mounted on load, Poi gets mounted on route change
- Use react-router params to get ID

componentWillReceiveProps

aka onChange

I before E...

The screenshot shows a GitHub repository page for `componentWillReceiveProps`. The top navigation bar includes links for Pull requests, Issues, and Gist. Below the navigation, there are links for Repositories, Code (4K), Commits (55), Issues (108), Wikis (4), and Users, along with an Advanced search link. A sidebar on the right displays a list of languages and their counts: JavaScript (3,646), CoffeeScript (65), TypeScript (65), JSX (62), Unity3D Asset (59), Markdown (58), HTML (54), Text (7), TeX (2), and C# (1). The main content area shows 4,094 code results for the query. A code snippet from a file named `speed.jsx` is displayed, showing the `componentWillReceiveProps` method highlighted. The code is as follows:

```
5     constructor(props){  
6         super(props);  
7         this.state = {  
8             speed: this.props.speed  
9         };  
10    }  
11  
12    componentWillReceiveProps(nextProps) {  
13    }  
14  
15    render() {  
16        // return();  
17    }  
18  
19    }  
20  
21 }
```

componentWillReceiveProps

- props have changed
- NOT called on initial render
- Update state based on props
- Dispatch actions
- Be careful, can cause loops i.e. don't compare objects
- Update state if props don't match

componentWillReceiveProps

```
class AvatarUploader extends Component {
  constructor(props) {
    super(props);
    this.state = { src: props.src };
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.src !== this.props.src) {
      this.setState({
        src: nextProps.src,
      });
    }
  }
  uploadFiles() {
    this.props.upload()
  }
  render() { /* ... */ }
}
```

- Useful when state is set with a prop.
- **Anti-pattern alert!**

componentWillReceiveProps

```
class Editor extends PureComponent {
  constructor(props) { /*... */ }
  upload(files) {
    this.props.dispatch(uploadAction(files));
  }
  render() {
    const { image } = this.props;
    return (
      <AvatarUploader
        src={image}
        upload={this.upload}
      />
    )
  }
}

const mapStateToProps = (state) => ({ image: state.image })
export default connect(mapStateToProps)(Editor);
```

- image gets passed in to AvatarUploader
- Causes componentWillReceiveProps to fire

React Router

```
class SightComponent extends Component {  
  render() {  
    return (  
      <div>  
        <Link to="/a/poi-sig/381139/362228">Country Music Hall of Fame</Link>  
      </div>  
    )  
  }  
}
```

- **Country Music Hall of Fame**
- If a component is mounted already

React Router

```
componentWillReceiveProps(nextProps) {  
  const { id: currentId } = this.props.params;  
  const { id: nextId } = nextProps.params;  
  
  if (currentId !== nextId) {  
    this.props.fetchPoi(nextId);  
  }  
}
```

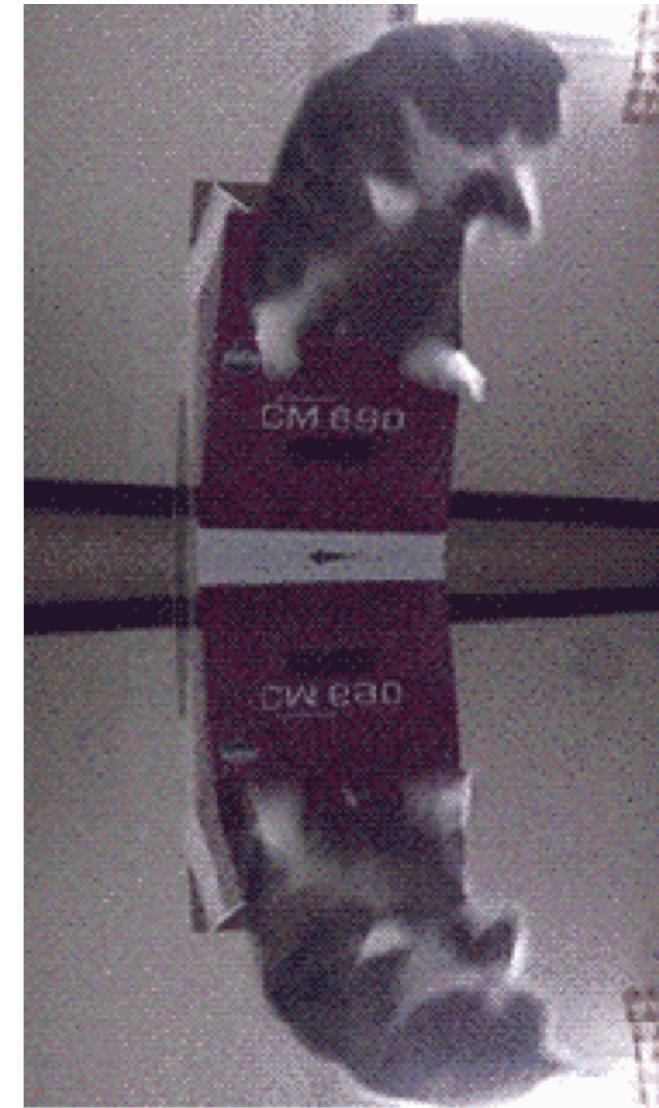
- Compare last id to current id
- Name variables for readability

100ps

```
componentWillReceiveProps(nextProps) {  
  const { id: currentId } = this.props.params;  
  const { id: nextId } = nextProps.poi;  
  
  if (currentId !== nextId) {  
    this.props.fetchPoi(nextId);  
  }  
  if (this.props.poi !== nextProps.poi) {  
    // W/ redux, this is probably always true!  
  }  
}
```

- Next poi.id won't match
- Be careful comparing objects too! They change a lot.

loops



⚠ pictureisunrelated.com

shouldComponentUpdate

aka shouldRender

shouldComponentUpdate

```
shouldComponentUpdate(this, nextProps, nextState) {  
  return shallowCompare(this, nextProps, nextState);  
}
```

```
export class Counter extends React.PureComponent {  
  // ...  
}
```

- Tell the component whether or not to render
- Can **increase performance**
- Used to call shallowCompare
- MOST of the time, use React.PureComponent

shouldComponentUpdate

```
shouldComponentUpdate(nextProps) {  
  return nextProps.poi.id !== this.props.poi.id;  
}
```

- If you need fine grain control
- Parent could change a prop
- Return false will cancel...
- componentWillUpdate, render, and componentDidUpdate

componentWillUpdate

aka beforeRender

componentWillUpdate

- Called BEFORE render like when setState called
- Do NOT call setState here
- Useful for triggering CSS animations or transitions

componentDidUpdate

aka afterRender

componentDidUpdate

```
componentDidUpdate(prevProps) {  
  if (prevProps.poi.id !== this.poi.id) {  
    window.scrollTop = 0;  
    this.props.pageView();  
  }  
}
```

- Gives you previous props prevProps to compare before actions
- Called AFTER render
- Update the DOM

Full Example

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

React Router

```
render() {  
  return (  
    <Link to={poiLink(1234, 362228)} />  
  )  
}
```

- Click a link
- Will pass props down to PoiDetail

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```



```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

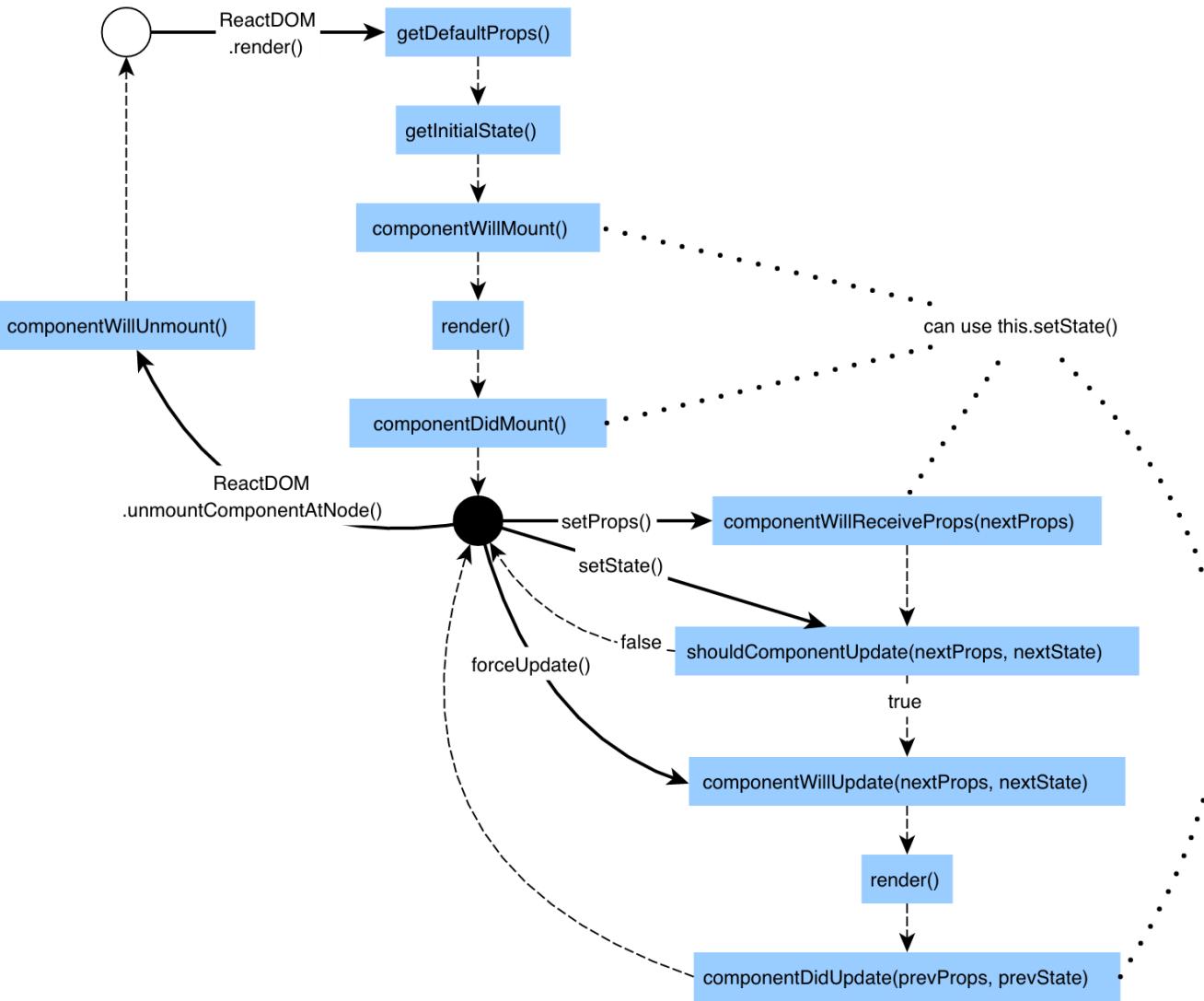
```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

tl;dr;

Name of thing	Sorta like...	Mounted?	Can you even setState?	What would you say... ya do here?
constructor	initialize()	nope	nope	init stuff NO side effects
componentWillMount	beforeDomReady()	nope	yeah but don't	Only needed in createClass now use constructor for most things
render	render	nope	please no	render stuff and don't set any state please
componentDidMount	domReady()	yup	yup	DOM is a go init jQuery plugins dispatch stuff
componentWillReceiveProps	onChange()	yup	yup	Props changed feel free to update state if needed
componentWillUpdate	beforeRender()	yup	nope	The props or state changed need to do anything else before rendering?
shouldComponentUpdate	shouldRender()	yup	nope	So yeah something changed but do we REALLY need to update?
componentDidUpdate	afterRender()	yup	yup	Great success we've rendered a thing... anything else?
componentWillUnmount	destroy()	too late	too late	Only you can prevent memory leaks aka unbind things

```
--  
71  /**  
72   * ----- The Life-Cycle of a Composite Component -----  
73   *  
74   * - constructor: Initialization of state. The instance is now retained.  
75   *   - componentWillMount  
76   *   - render  
77   *   - [children's constructors]  
78   *     - [children's componentWillMount and render]  
79   *     - [children's componentDidMount]  
80   *     - componentDidMount  
81   *  
82   *     Update Phases:  
83   *     - componentWillMountReceiveProps (only called if parent updated)  
84   *     - shouldComponentUpdate  
85   *       - componentWillUpdate  
86   *       - render  
87   *       - [children's constructors or receive props phases]  
88   *       - componentDidUpdate  
89   *  
90   *       - componentWillUnmount  
91   *       - [children's componentWillUnmount]  
92   *     - [children destroyed]  
93   * - (destroyed): The instance is now blank, released by React and ready for GC.  
94   *  
95   * -----  
96  */
```



- <https://tylermcginnis.com/an-introduction-to-life-cycle-events-in-react-js/>

Testing

Testing

```
npm i install -S jest enzyme
```

- Jest for suite
- Enzyme for sweet
- Super important to test
- Always test logic
- Use **mount** to run JSDom

Testing willMount

```
describe("Detail Page", () => {
  it("should fetch a page if there isn't one loaded", () => {
    const fetch = jest.fn();
    const wrapper = mount(
      <Details
        poi={null}
        fetchPoi={fetch}
      />
    );
    expect(fetch).not.toHaveBeenCalled();
  });
});
```

Testing willReceiveProps

```
describe("Detail Page", () => {
  it("should fetch a new page", () => {
    const fetch = jest.fn();
    const wrapper = mount(
      <Details
        poi={{ id: 1}}
        params={{
          id: 1
        }}
        fetchPoi={fetch}
      />
    );
    wrapper.setProps({
      params: { id: 2 }
    });

    expect(fetch).toHaveBeenCalled();
  });
});
```

- Use setProps

Additional Resources

- <https://gist.github.com/jcreamer898/aeaf4b7a08b9871c3a48ad4bb7ccb35c>
- <https://engineering.musefind.com/react-lifecycle-methods-how-and-when-to-use-them-2111a1b692b1>
- https://developmentarc.gitbooks.io/react-indepth/content/lifecycle/lifecyclemethods_overview.html
- <http://busypeoples.github.io/post/react-component-lifecycle/>

Thanks!



@jcreamer898

jonathancreamer.com