Module 11

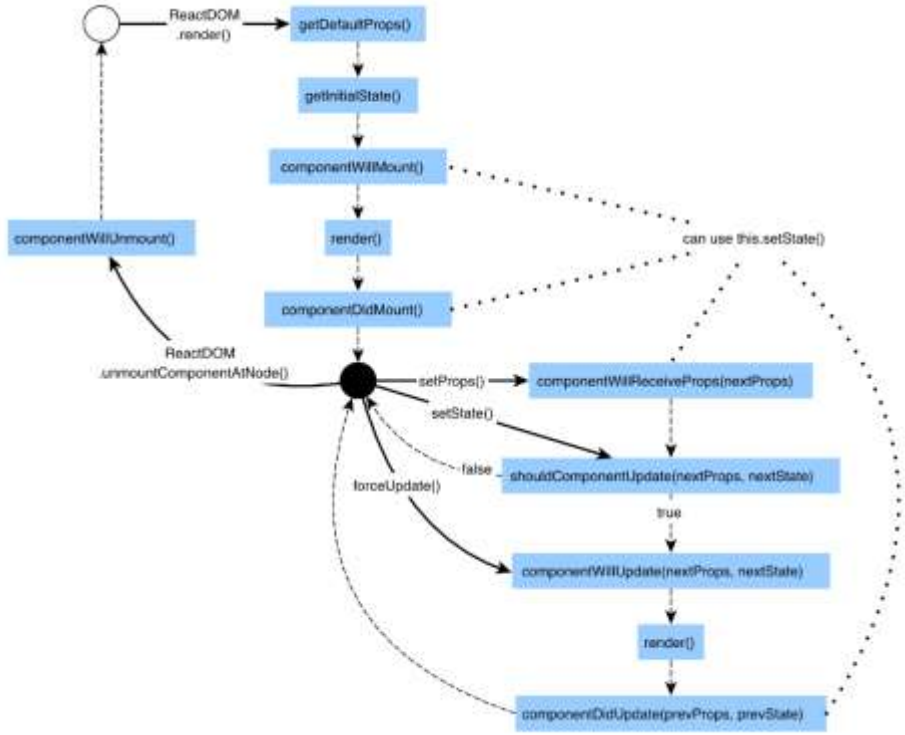
The Component Lifecycle

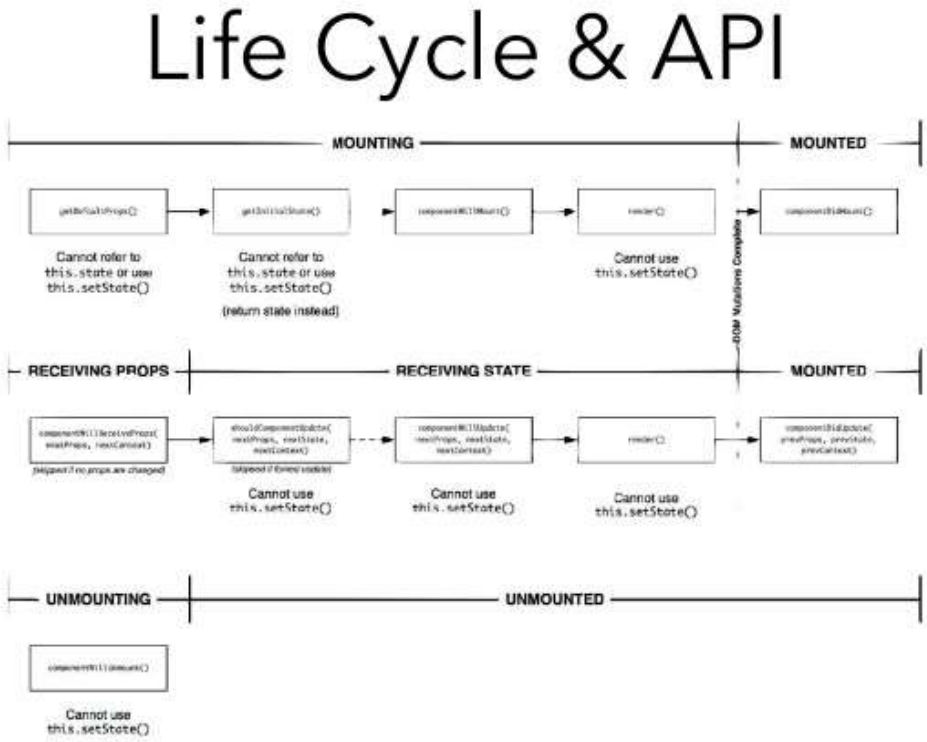
11. The Component Lifecycle

In the beginning, we started off with a very simple view of components and what they do. As we learned more about React and did cooler and more involved things, it turns out our components aren't all that simple. They help deal with properties, state, events, and often are responsible for the well-being of other components as well. Keeping track of everything components do sometimes can be tough.

To help with this, React provides us with something known as lifecycle methods. Lifecycle methods are (unsurprisingly) special methods that automatically get called as our component goes about its business. They notify us of important milestones in our component's life, and we can use these notifications to simply pay attention or change what our component is about to do.

In this chapter, we look at these lifecycle methods and learn all about what we can do with them.





https://www.facebook.com/photo.php?fbid=10154774642145430&set=gm.1491610957781164&type=1&theater

Meet the Lifecycle Methods

Lifecycle methods are not very complicated. We can think of them as glorified event handlers that get called at various points in a component's life, and just like event handlers, you can write some code to do things at those various points. Before we go further, it is time for you to quickly meet our lifecycle methods. They are:

- componentWillMount
- componentDidMount
- componentWillUnmount
- componentWillUpdate
- componentDidUpdate
- shouldComponentUpdate
- componentWillReceiveProps

We aren't quite done yet. There are three more methods that we are going to throw into the mix even though they aren't strictly lifecycle methods, and they are:

- getInitialState
- getDefaultProps
- render

Some of these names probably sound familiar to you, and some you are probably seeing for the first time. Don't worry. By the end of all this, you'll be on a first name basis with all of them! What we are going to do is look at these lifecycle methods from various angles—starting with some code!

See the Lifecycle Methods in Action

Learning about these lifecycle methods is about as exciting as memorizing names for foreign places (or distant star systems!) you have no plans to visit. To help make all of this more bearable, I am going to first have you play with them through a simple example before we get all academic and read about them.

To play with this example, go to the following URL: https://www.kirupa.com/react/lifecycle_example.htm Once this page loads, you'll see a variation of the counter example we saw earlier (see Figure 11-1).

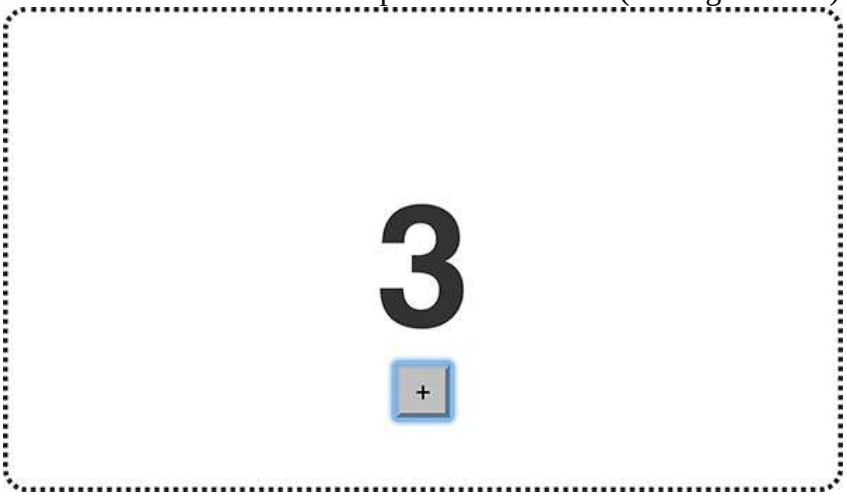


Figure 11-1 A variation on the counter example.

Don't click on the button or anything just yet. If you have already clicked on the button, just refresh the page to start the example from the beginning. There is a reason why I am saying that. We want to see this page as it is before we interact with it!

Now, bring up your browser's developer tools and take a look at the Console tab. In Chrome, you'll see something that looks like Figure 11-2.

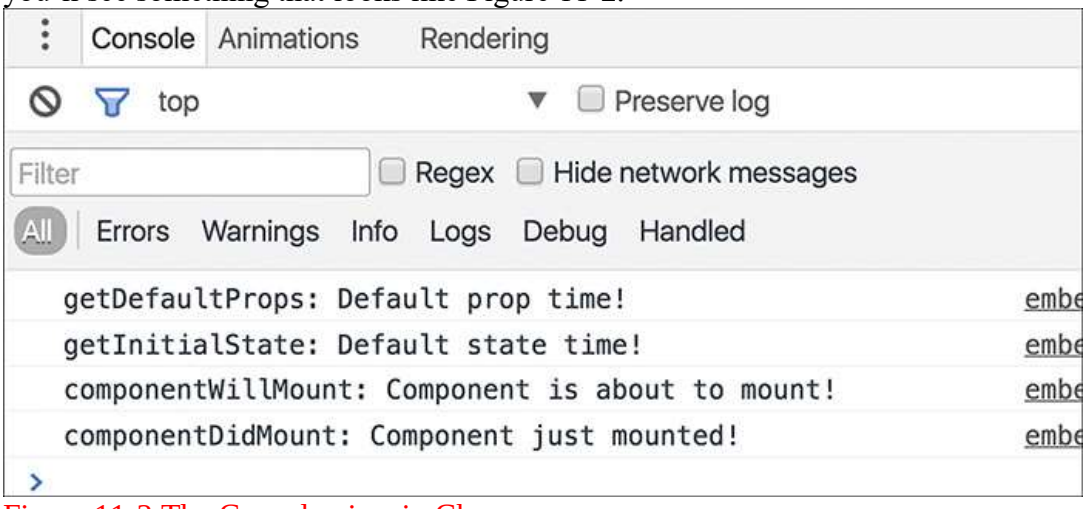


Figure 11-2 The Console view in Chrome.

Notice what you see printed. You will see some messages, and these messages start out with the name of what looks like a lifecycle method. If you click on the plus button once, notice that your Console will show more lifecycle methods getting called (see Figure 11-3).

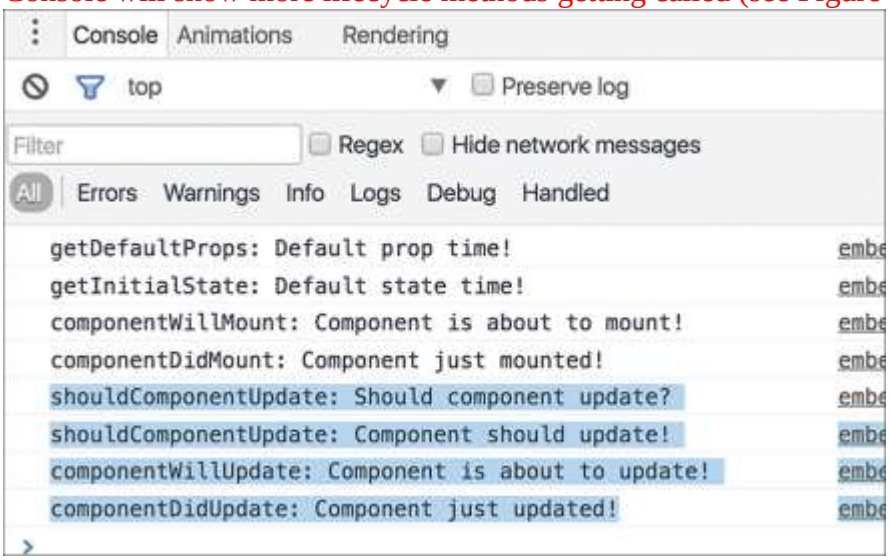


Figure 11-3 More lifecycle methods getting called.

Play with this example for a bit. What this example does is allow you to place all of these lifecycle methods in the context of a component that we've already seen earlier. As you keep hitting the plus button, more lifecycle method entries will show up.

Eventually, once your counter approaches a **value of 5**, your example will just disappear with the following entry showing up in your console: componentWillUnmount: Component is about to be removed from the DOM! At this point, you have reached the end of this example. Of course, to start over, you can just refresh the page!

Now that you've seen the example, let's take a quick look at the component that is responsible for all of this:

```
var CounterParent = React.createClass({
  getDefaultProps: function(){
    console.log("getDefaultProps: Default prop time!");
    return {};
  },
  getInitialState: function() {
    console.log("getInitialState: Default state time!");
    return {
      count: 0
    };
  },
  increase: function() {
    this.setState({
      count: this.state.count + 1
    });
  },
  componentWillUpdate: function(newProps, newState) {
      console.log("componentWillUpdate: Component is about to update!");
  },
  componentDidUpdate: function(currentProps, currentState) {
      console.log("componentDidUpdate: Component just updated!");
  },
  componentWillMount: function() {
      console.log("componentWillMount: Component is about to mount!");
  },
  componentDidMount: function() {
      console.log("componentDidMount: Component just mounted!");
  },
  componentWillUnmount: function() {
      console.log("componentWillUnmount: Component is about to be removed from the
DOM!");
  },
  shouldComponentUpdate: function(newProps, newState) {
    console.log("shouldComponentUpdate: Should component update?");

    if (newState.count < 5) {
      console.log("shouldComponentUpdate: Component should update!");
      return true;
    } else {
      ReactDOM.unmountComponentAtNode(destination);
      console.log("shouldComponentUpdate: Component should not update!");
      return false;
    }
```

```
  },
  componentWillReceiveProps: function(newProps){
    console.log("componentWillReceiveProps: Component will get new props!");
  },
  render: function() {
      var backgroundStyle = {
        padding: 50,
        border: "#333 2px dotted",
        width: 250,
        height: 100,
        borderRadius: 10,
        textAlign: "center"
      };

      return (
        <div style={backgroundStyle}>
          <Counter display={this.state.count}/>
          <button onClick={this.increase}>
            +
          </button>
        </div>
      );
    }
});
```

Take a few moments to look what all of this code does. It seems lengthy, but a bulk of it is just each lifecycle method listed with a console.log statement defined. Once you've gone through this code, play with the example one more time. Trust me. The more time you spend in the example and figure out what is going on, the more fun you are going to have. The following sections where we look at each lifecycle method across the rendering, updating, and unmounting phases is going to be dreadfully boring. Don't say I didn't warn you.

The Initial Rendering Phase

When your component is about to start its life and make its way to the DOM, the following lifecycle methods get called (see Figure 11-4).

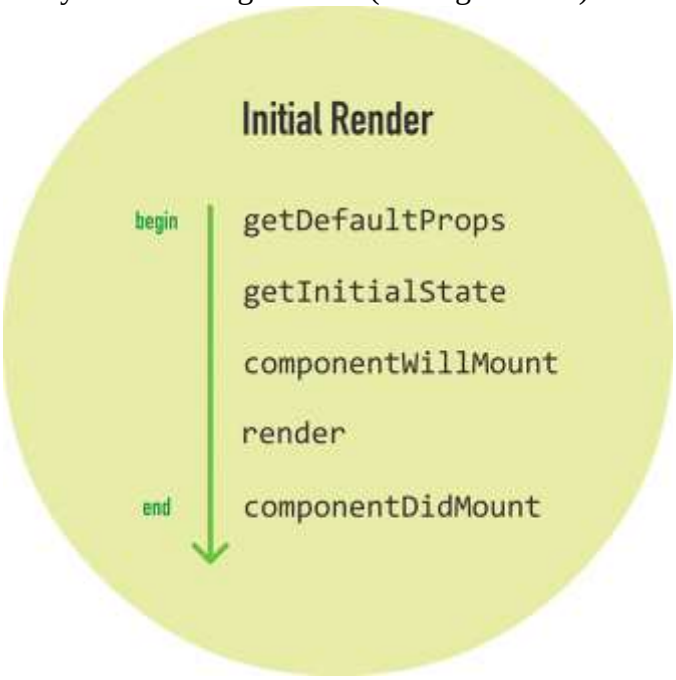


Figure 11-4 The lifecycle methods called initially.

What you saw in your console when the example was loaded was a less colorful version of what you saw here. Now, we are going to go a bit further and learn more about what each of these lifecycle methods do.

getDefaultProps
This method allows you to specify the default value of this.props. It gets called before your component is even created or any props from parents are passed in.

getInitialState
This method allows you to specify the default value of this.state before your component is created. Just like getDefaultProps, it too gets called before your component is created.

componentWillMount
This is the last method that gets called before your component gets rendered to the DOM. There is an important thing to note here. If you were to call setState inside this method, your component will not re-render (aka have the render method get called and update what gets displayed on screen).

render
This one should be very familiar to you by now. Every component must have this method defined, and it is responsible for returning a single root node (which may have many child nodes inside it). If you don't wish to render anything (for some fancy optimization you might be going for), simply return null or false.

componentDidMount
This method gets called immediately after your component renders and gets placed on the DOM. At this point, you can safely perform any DOM querying operations without worrying about whether your component has made it or not. If you have any code that depends on your component being ready, you can specify all of that code here as well.

With the exception of the render method, all of these lifecycle methods can fire only once. That's quite different from the methods we are about to see next.

The Updating Phase

After your components get added to the DOM, they can potentially update and re-render when a prop or state change occurs. During this time, a different collection of lifecycle methods will get called.

Dealing with State Changes

First, let's look at a state change! When a state change occurs, we mentioned earlier that your component will call its render method again. Any components that rely on the output of this component will also get their render methods called as well. This is done to ensure that our component is always displaying the latest version of itself. All of that is true, but that is only a partial representation of what happens.

When a state change happens, all the lifecycle methods in Figure 11-5 get called.

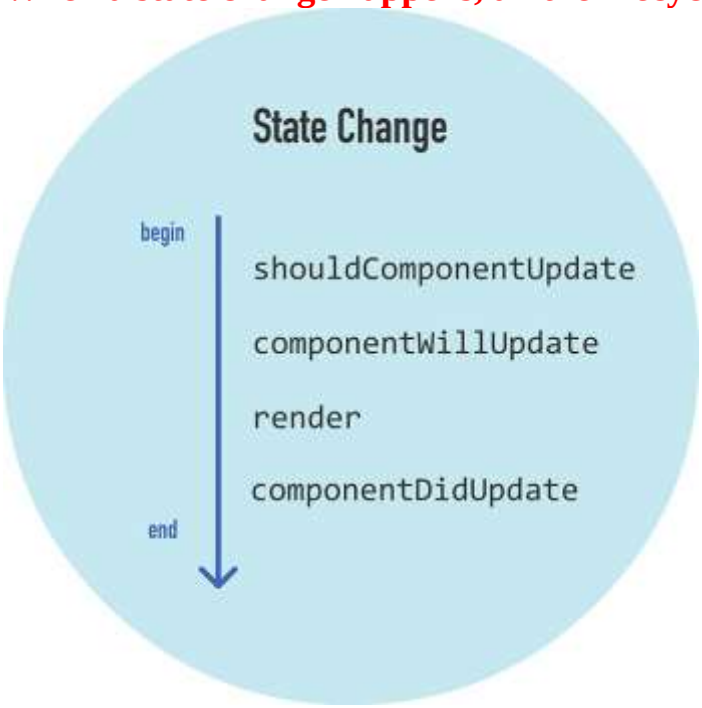


Figure 11-5 Lifecycle methods called when a state change happens.

What these lifecycle methods do is outlined in the following sections.

shouldComponentUpdate

Sometimes, you don't want your component to update when a state change occurs. This method allows you to control this updating behavior. If you use this method and return a true value, the component will update. If this method returns a false value, this component will skip updating.

That probably sounds a little bit confusing, so here is a simple snippet:

```
shouldComponentUpdate: function(newProps, newState) {

  if (newState.id <= 2) {
    console.log("Component should update!");

    return true;
  } else {
    console.log("Component should not update!");

    return false;
  }
}
```

This method gets called with two arguments which we name newProps and newState. What we are doing in this snippet of code is checking whether the new value of our id state property is less than or equal to 2. If the value is less than or equal to 2, we return true to indicate that this component should update. If the value is not less than or equal to 2, we return false to indicate that this component should not update.

componentWillUpdate
This method gets called just before your component is about to update. Nothing too exciting here. One thing to note is that you can't change your state by calling this.setState from this method.

render
If you didn't override the update via shouldComponentUpdate (by returning false), the code inside render will get called again to ensure your component displays itself properly.

componentDidUpdate
This method gets called after your component updates and the render method has been called. If you need to execute any code after the update takes place, this is the place to stash it.

Dealing with Prop Changes

The other time your component updates is when its prop value changes after it has been rendered into the DOM. In this scenario, the lifecycle methods in Figure 11-6 get called.

Figure 11-6 Lifecycle methods when the component's prop value changes.

The only method that is new here is **componentWillReceiveProps**. This method returns one argument, and this argument is an object that contains the new prop values that are about to be assigned to it.

We saw the rest of the lifecycle methods earlier when looking at state changes, so let's not revisit them again. Their behavior is identical when dealing with a prop change.

The Unmounting Phase

The last phase we are going to look at is when your component is about to be destroyed and removed from the DOM (see Figure 11-7).

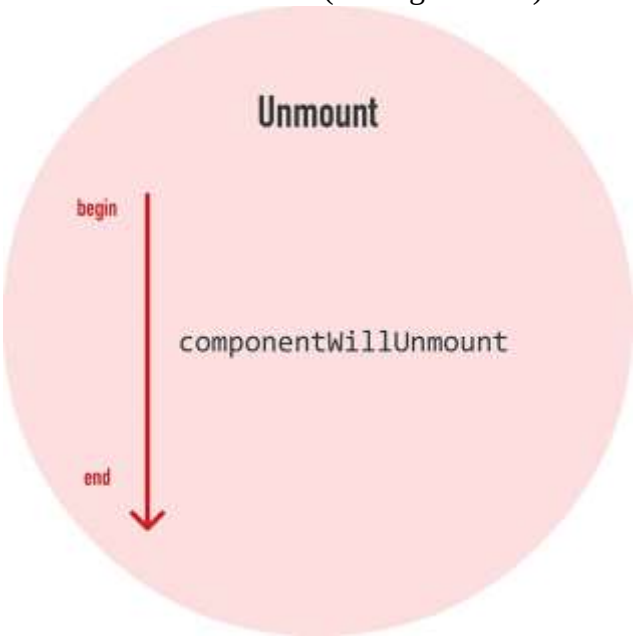


Figure 11-7 Only one lifecycle method is active when your component is about to be destroyed and removed from the DOM.

There is only one lifecycle method that is active here, and that is **componentWillUnmount**. You'll perform any cleanup-related tasks here such as removing event listeners, stopping timers, etc. After this method gets called, your component is removed from the DOM and you can say Bye! to it.

Conclusion

Our components are fascinating little things. On the surface, they seem like they don't have much going on. Like a good documentary about the oceans, when we look a little deeper and closer, it's almost like seeing a whole other world. As it turns out, React is constantly watching and notifying your component every time something interesting happens. All of this is done via the (extremely boring) lifecycle methods that we spent this entire tutorial looking at. Now, I want to reassure you that knowing what each lifecycle method does and when it gets called will come in handy one day. Everything you've learned isn't just trivial knowledge, though your friends will be impressed if you can describe all of the lifecycle methods from memory. Go ahead and try it the next time you see them.