REACT JS

State

Sreekanth M. E.

Freelance Trainer & Consultant

http://www.SreekanthME.com

State is private and fully controlled by the component.

State is often called local or encapsulated. It is not accessible to any component other than the one that owns and sets it.

Props are defined when a component is created and they remain unchanged over time.

State is initialized when the component is created but can be updated throughout the component's lifecycle by user interactivity, data returned from server etc.

Components with state are referred to as Smart components or Stateful components.

State is a feature available only to JS classes. It is not available to JS functions.

Therefore, stateful components can be defined only with JS classes.

All state data are properties of the this.state object.

They can be accessed as follows:

this.state.<property_name>

It is okay to add additional variables (non-state) to the component class if there is a need to store something.

As a thumb rule, if something is not used in render() method, it shouldn't be in the state.

Class components should always call the base constructor (of React.Component class) with props argument.

State object is initialized in the constructor.

State properties must never be updated directly.

State must be updated using the inherited setState() method.

setState() method has two variants:

1. The first one takes an object as a parameter.
2. The second one takes a callback. The callback receives previous state and component props as arguments. It must return an object that can be used for state update.

State updates via setState() may happen asynchronously. So always prefer the second approach (callback).

When setState() is called, React merges the object provided as input into the current state.

The object provided as input need not contain all properties of the state object. It can contain only one or a subset of the state properties.

Whenever setState() is called something in the state object is updated, the component's render method gets automatically called.

This kicks of a cascade of render calls for any component whose output is also affected. The end result of all this is that what you see in your screen is the latest representation of your app's UI state.

Neither parent nor child components must care whether other components have a state or not.

A component may choose to pass its state down as props to its child components

This is commonly called a "top-down" or "unidirectional" data flow. Any state is always owned by some specific component, and any data or UI derived from that state can only affect components "below" them in the tree.

END OF CHAPTER

APPENDIX