React Component API

ReactJS component is a top-level API. It makes the code completely individual and reusable in the application. It includes various methods for:

- Creating elements
- Transforming elements
- Fragments

Here, we are going to explain the three most important methods available in the React component API.

1. setState()
2. forceUpdate()
3. findDOMNode()

setState()

This method is used to update the state of the component. This method does not always replace the state immediately. Instead, it only adds changes to the original state. It is a primary method that is used to update the user interface(UI) in response to event handlers and server responses.

> Note: In the ES6 classes, this.method.bind(this) is used to manually bind the setState() method.

Syntax

```
this.stateState(object newState[, function callback]);
```

In the above syntax, there is an optional **callback** function which is executed once setState() is completed and the component is re-rendered.

Example

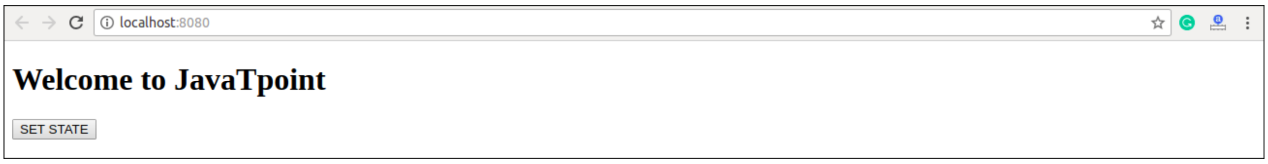
```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
class App extends React.Component {
  constructor() {
      super();
      this.state = {
         msg: "Welcome to JavaTpoint"
      };
      this.updateSetState = this.updateSetState.bind(this);
   }
   updateSetState() {
       this.setState({
         msg:"Its a best ReactJS tutorial"
       });
   }
   render() {
      return (
         <div>
            <h1>{this.state.msg}</h1>
            <button onClick = {this.updateSetState}>SET STATE</button>
         </div>
      );
   }
}
export default App;
```

Main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';

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

Output:

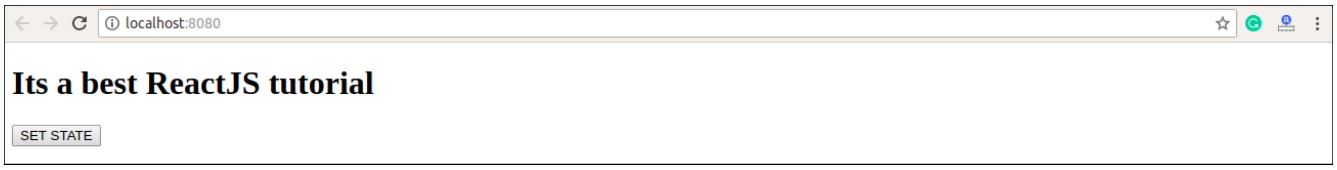


localhost:8080

Welcome to JavaTpoint

SET STATE

When you click on the **SET STATE** button, you will see the following screen with the updated message.


localhost:8080

Its a best ReactJS tutorial

forceUpdate()

This method allows us to update the component manually.

Syntax

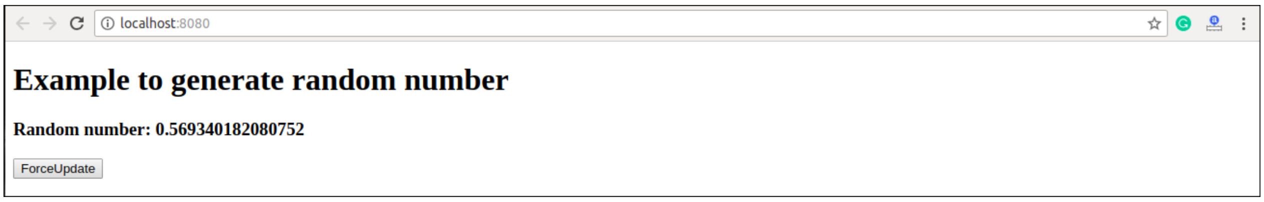
```
Component.forceUpdate(callback);
```

Example

App.js

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor() {
    super();
    this.forceUpdateState = this.forceUpdateState.bind(this);
  }
  forceUpdateState() {
    this.forceUpdate();
  };
  render() {
    return (
      <div>
        <h1>Example to generate random number</h1>
        <h3>Random number: {Math.random()}</h3>
        <button onClick = {this.forceUpdateState}>ForceUpdate</button>
      </div>
    );
  }
}
export default App;
```

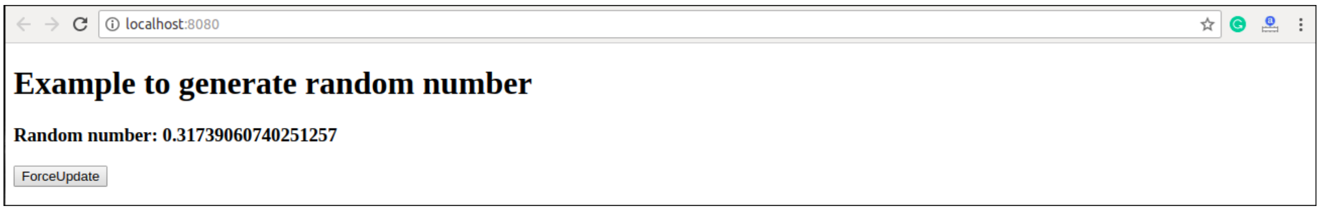
Output:


localhost:8080

Example to generate random number

Random number: 0.569340182080752

Each time when you click on **ForceUpdate** button, it will generate the **random** number. It can be shown in the below image.


localhost:8080

Example to generate random number

Random number: 0.31739060740251257

findDOMNode()

For DOM manipulation, you need to use **ReactDOM.findDOMNode()** method. This method allows us to find or access the underlying DOM node.

Syntax

```
ReactDOM.findDOMNode(component);
```

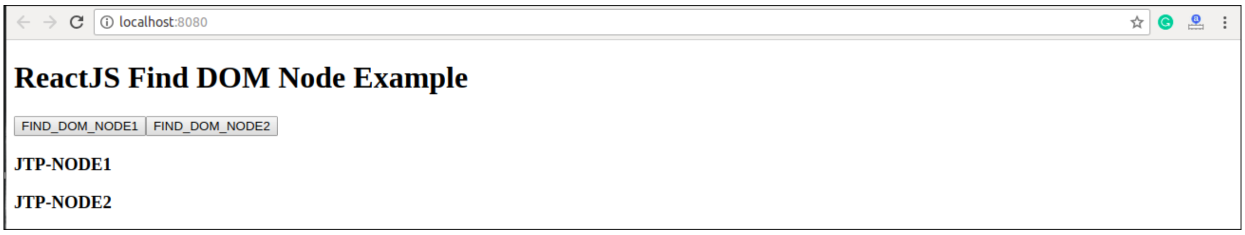
Example

For DOM manipulation, first, you need to import this line: **import ReactDOM** from '**react-dom**' in your **App.js** file.

App.js

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
class App extends React.Component {
  constructor() {
    super();
    this.findDomNodeHandler1 = this.findDomNodeHandler1.bind(this);
    this.findDomNodeHandler2 = this.findDomNodeHandler2.bind(this);
  };
  findDomNodeHandler1() {
    var myDiv = document.getElementById('myDivOne');
    ReactDOM.findDOMNode(myDivOne).style.color = 'red';
  }
  findDomNodeHandler2() {
    var myDiv = document.getElementById('myDivTwo');
    ReactDOM.findDOMNode(myDivTwo).style.color = 'blue';
  }
  render() {
    return (
      <div>
        <h1>ReactJS Find DOM Node Example</h1>
        <button onClick = {this.findDomNodeHandler1}>FIND_DOM_NODE1</button>
        <button onClick = {this.findDomNodeHandler2}>FIND_DOM_NODE2</button>
        <h3 id = "myDivOne">JTP-NODE1</h3>
        <h3 id = "myDivTwo">JTP-NODE2</h3>
      </div>
    );
  }
}
export default App;
```

Output:

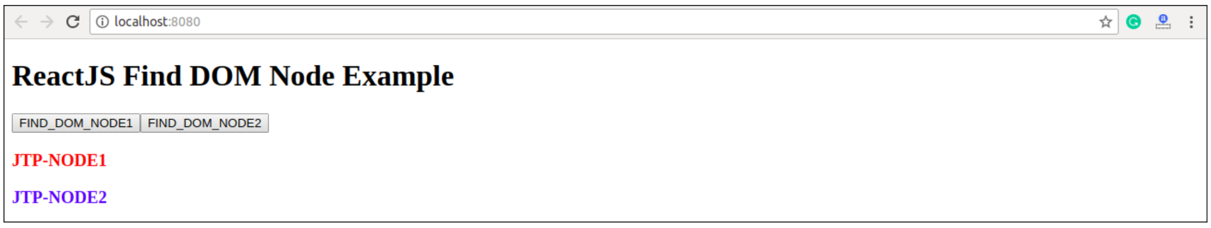


ReactJS Find DOM Node Example

JTP-NODE1

JTP-NODE2

Once you click on the **button**, the color of the node gets changed. It can be shown in the below screen.



ReactJS Find DOM Node Example

JTP-NODE1

JTP-NODE2

Next →← Prev

React Component Life-Cycle

In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle. These lifecycle methods are not very complicated and called at various points during a component's life. The lifecycle of the component is divided into **four phases**. They are:

1. Initial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase

Each phase contains some lifecycle methods that are specific to the particular phase. Let us discuss each of these phases one by one.

1. Initial Phase

It is the **birth** phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component. The initial phase only occurs once and consists of the following methods.

- **getDefaultProps()**
  It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.
- **getInitialState()**
  It is used to specify the default value of this.state. It is invoked before the creation of the component.

2. Mounting Phase

In this phase, the instance of a component is created and inserted into the DOM. It consists of the following methods.

- **componentWillMount()**
  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.
- **componentDidMount()**
  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
- **render()**
  This method is defined in each and every component. It is responsible for returning a single root **HTML node** element. If you don't want to render anything, you can return a **null** or **false** value.

3. Updating Phase

It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State**. This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself. Unlike the Birth or Death phase, this phase repeats again and again. This phase consists of the following methods.

- **componentWillRecieveProps()**
  It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using **this.setState()** method.
- **shouldComponentUpdate()**
  It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.
- **componentWillUpdate()**
  It is invoked just before the component updating occurs. Here, you can't change the component state by invoking **this.setState()** method. It will not be called, if **shouldComponentUpdate()** returns false.
- **render()**
  It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String

and Number. If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.

- **componentDidUpdate()**
  It is invoked immediately after the component updating occurs. In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

4. Unmounting Phase

It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM. This phase contains only one method and is given below.

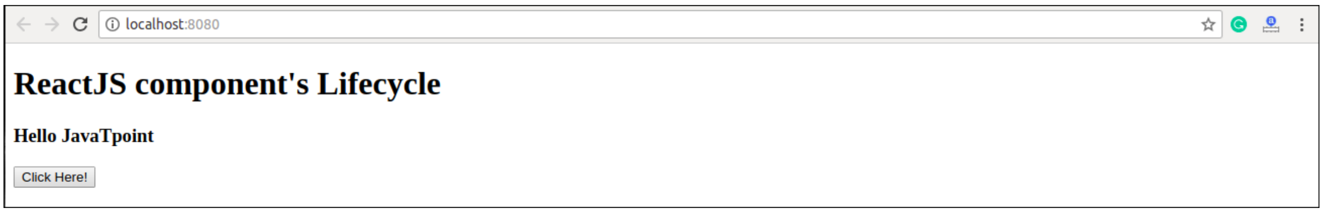
- **componentWillUnmount()**
  This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary **cleanup** related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

Example

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

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {hello: "JavaTpoint"};
    this.changeState = this.changeState.bind(this)
  }
  render() {
    return (
      <div>
        <h1>ReactJS component's Lifecycle</h1>
        <h3>Hello {this.state.hello}</h3>
        <button onClick = {this.changeState}>Click Here!</button>
      </div>
    );
  }
  componentWillMount() {
    console.log('Component Will MOUNT!')
  }
  componentDidMount() {
    console.log('Component Did MOUNT!')
  }
  changeState(){
    this.setState({hello:"All!!- Its a great reactjs tutorial."});
  }
  componentWillReceiveProps(newProps) {
    console.log('Component Will Recieve Props!')
  }
  shouldComponentUpdate(newProps, newState) {
    return true;
  }
  componentWillUpdate(nextProps, nextState) {
    console.log('Component Will UPDATE!');
  }
  componentDidUpdate(prevProps, prevState) {
    console.log('Component Did UPDATE!')
  }
  componentWillUnmount() {
    console.log('Component Will UNMOUNT!')
  }
}
export default App;
```

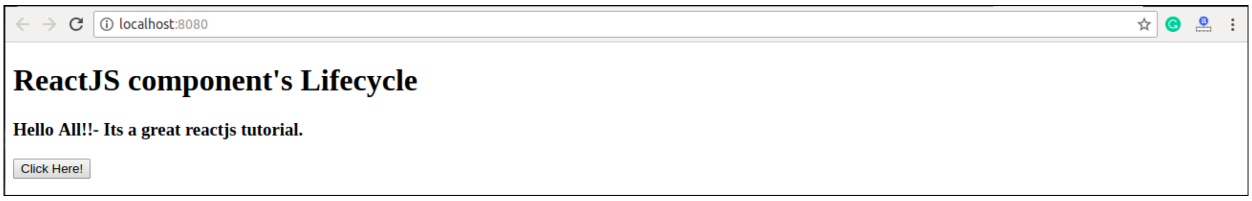
Output:



ReactJS component's Lifecycle

Hello JavaTpoint

When you click on the **Click Here** Button, you get the updated result which is shown in the below screen.


localhost:8080

ReactJS component's Lifecycle

Hello All!!- Its a great reactjs tutorial.