**React.Component**

[Components](https://facebook.github.io/react/docs/components-and-props.html) let you split the UI into independent, reusable pieces, and think about each piece in isolation. React.Component is provided by [React](https://facebook.github.io/react/docs/react-api.html).

**Overview**

React.Component is an abstract base class, so it rarely makes sense to refer toReact.Component directly. Instead, you will typically subclass it, and define at least a[render()](https://facebook.github.io/react/docs/react-component.html#render) method.

Normally you would define a React component as a plain [JavaScript class](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes):

class Greeting extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

If you don't use ES6 yet, you may use the [React.createClass](https://facebook.github.io/react/docs/react-api.html#createclass) helper instead. Take a look at[Using React without ES6](https://facebook.github.io/react/docs/react-without-es6.html) to learn more.

**The Component Lifecycle**

Each component has several "lifecycle methods" that you can override to run code at particular times in the process. Methods prefixed with **will** are called right before something happens, and methods prefixed with **did** are called right after something happens.

**Mounting**

These methods are called when an instance of a component is being created and inserted into the DOM:

* [constructor()](https://facebook.github.io/react/docs/react-component.html#constructor)
* [componentWillMount()](https://facebook.github.io/react/docs/react-component.html#componentwillmount)
* [render()](https://facebook.github.io/react/docs/react-component.html#render)
* [componentDidMount()](https://facebook.github.io/react/docs/react-component.html#componentdidmount)

**Updating**

An update can be caused by changes to props or state. These methods are called when a component is being re-rendered:

* [componentWillReceiveProps()](https://facebook.github.io/react/docs/react-component.html#componentwillreceiveprops)
* [shouldComponentUpdate()](https://facebook.github.io/react/docs/react-component.html#shouldcomponentupdate)
* [componentWillUpdate()](https://facebook.github.io/react/docs/react-component.html#componentwillupdate)
* [render()](https://facebook.github.io/react/docs/react-component.html#render)
* [componentDidUpdate()](https://facebook.github.io/react/docs/react-component.html#componentdidupdate)

**Unmounting**

This method is called when a component is being removed from the DOM:

* [componentWillUnmount()](https://facebook.github.io/react/docs/react-component.html#componentwillunmount)

**Other APIs**

Each component also provides some other APIs:

* [setState()](https://facebook.github.io/react/docs/react-component.html#setstate)
* [forceUpdate()](https://facebook.github.io/react/docs/react-component.html#forceupdate)

**Class Properties**

* [defaultProps](https://facebook.github.io/react/docs/react-component.html#defaultprops)
* [displayName](https://facebook.github.io/react/docs/react-component.html#displayname)
* [propTypes](https://facebook.github.io/react/docs/react-component.html#proptypes)

**Instance Properties**

* [props](https://facebook.github.io/react/docs/react-component.html#props)
* [state](https://facebook.github.io/react/docs/react-component.html#state)

**Reference**

**render()**

render()

The render() method is required.

When called, it should examine this.props and this.state and return a single React element. This element can be either a representation of a native DOM component, such as<div />, or another composite component that you've defined yourself.

You can also return null or false to indicate that you don't want anything rendered. When returning null or false, ReactDOM.findDOMNode(this) will return null.

The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser. If you need to interact with the browser, perform your work incomponentDidMount() or the other lifecycle methods instead. Keeping render() pure makes components easier to think about.

**Note**

render() will not be invoked if [shouldComponentUpdate()](https://facebook.github.io/react/docs/react-component.html#shouldcomponentupdate) returns false.

**constructor()**

constructor(props)

The constructor for a React component is called before it is mounted. When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefined in the constructor, which can lead to bugs.

The constructor is the right place to initialize state. If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.

It's okay to initialize state based on props if you know what you're doing. Here's an example of a valid React.Component subclass constructor:

constructor(props) {

super(props);

this.state = {

color: props.initialColor

};

}

Beware of this pattern, as it effectively "forks" the props and can lead to bugs. Instead of syncing props to state, you often want to [lift the state up](https://facebook.github.io/react/docs/lifting-state-up.html).

If you "fork" props by using them for state, you might also want to implement[componentWillReceiveProps(nextProps)](https://facebook.github.io/react/docs/react-component.html#componentwillreceiveprops) to keep the state up-to-date with them. But lifting state up is often easier and less bug-prone.

**componentWillMount()**

componentWillMount()

componentWillMount() is invoked immediately before mounting occurs. It is called beforerender(), therefore setting state in this method will not trigger a re-rendering. Avoid introducing any side-effects or subscriptions in this method.

This is the only lifecycle hook called on server rendering. Generally, we recommend using the constructor() instead.

**componentDidMount()**

componentDidMount()

componentDidMount() is invoked immediately after a component is mounted. Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request. Setting state in this method will trigger a re-rendering.

**componentWillReceiveProps()**

componentWillReceiveProps(nextProps)

componentWillReceiveProps() is invoked before a mounted component receives new props. If you need to update the state in response to prop changes (for example, to reset it), you may compare this.props and nextProps and perform state transitions usingthis.setState() in this method.

Note that React may call this method even if the props have not changed, so make sure to compare the current and next values if you only want to handle changes. This may occur when the parent component causes your component to re-render.

React doesn't call componentWillReceiveProps with initial props during [mounting](https://facebook.github.io/react/docs/react-component.html#mounting). It only calls this method if some of component's props may update. Calling this.setStategenerally doesn't trigger componentWillReceiveProps.

**shouldComponentUpdate()**

shouldComponentUpdate(nextProps, nextState)

Use shouldComponentUpdate() to let React know if a component's output is not affected by the current change in state or props. The default behavior is to re-render on every state change, and in the vast majority of cases you should rely on the default behavior.

shouldComponentUpdate() is invoked before rendering when new props or state are being received. Defaults to true. This method is not called for the initial render or whenforceUpdate() is used.

Returning false does not prevent child components from re-rendering when *their* state changes.

Currently, if shouldComponentUpdate() returns false, then [componentWillUpdate()](https://facebook.github.io/react/docs/react-component.html#componentwillupdate),[render()](https://facebook.github.io/react/docs/react-component.html#render), and [componentDidUpdate()](https://facebook.github.io/react/docs/react-component.html#componentdidupdate) will not be invoked. Note that in the future React may treat shouldComponentUpdate() as a hint rather than a strict directive, and returningfalse may still result in a re-rendering of the component.

If you determine a specific component is slow after profiling, you may change it to inherit from [React.PureComponent](https://facebook.github.io/react/docs/react-api.html#react.purecomponent) which implements shouldComponentUpdate() with a shallow prop and state comparison. If you are confident you want to write it by hand, you may compare this.props with nextProps and this.state with nextState and return false to tell React the update can be skipped.

**componentWillUpdate()**

componentWillUpdate(nextProps, nextState)

componentWillUpdate() is invoked immediately before rendering when new props or state are being received. Use this as an opportunity to perform preparation before an update occurs. This method is not called for the initial render.

Note that you cannot call this.setState() here. If you need to update state in response to a prop change, use componentWillReceiveProps() instead.

**Note**

componentWillUpdate() will not be invoked if [shouldComponentUpdate()](https://facebook.github.io/react/docs/react-component.html#shouldcomponentupdate) returns false.

**componentDidUpdate()**

componentDidUpdate(prevProps, prevState)

componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.

Use this as an opportunity to operate on the DOM when the component has been updated. This is also a good place to do network requests as long as you compare the current props to previous props (e.g. a network request may not be necessary if the props have not changed).

**Note**

componentDidUpdate() will not be invoked if [shouldComponentUpdate()](https://facebook.github.io/react/docs/react-component.html#shouldcomponentupdate) returns false.

**componentWillUnmount()**

componentWillUnmount()

componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any DOM elements that were created incomponentDidMount

**setState()**

setState(nextState, callback)

Performs a shallow merge of nextState into current state. This is the primary method you use to trigger UI updates from event handlers and server request callbacks.

The first argument can be an object (containing zero or more keys to update) or a function (of state and props) that returns an object containing keys to update.

Here is the simple object usage:

this.setState({mykey: 'my new value'});

It's also possible to pass a function with the signature function(state, props) => newState. This enqueues an atomic update that consults the previous value of state and props before setting any values. For instance, suppose we wanted to increment a value in state by props.step:

this.setState((prevState, props) => {

return {myInteger: prevState.myInteger + props.step};

});

The second parameter is an optional callback function that will be executed once setStateis completed and the component is re-rendered. Generally we recommend usingcomponentDidUpdate() for such logic instead.

setState() does not immediately mutate this.state but creates a pending state transition. Accessing this.state after calling this method can potentially return the existing value.

There is no guarantee of synchronous operation of calls to setState and calls may be batched for performance gains.

setState() will always lead to a re-render unless shouldComponentUpdate() returnsfalse. If mutable objects are being used and conditional rendering logic cannot be implemented in shouldComponentUpdate(), calling setState() only when the new state differs from the previous state will avoid unnecessary re-renders.

**forceUpdate()**

component.forceUpdate(callback)

By default, when your component's state or props change, your component will re-render. If your render() method depends on some other data, you can tell React that the component needs re-rendering by calling forceUpdate().

Calling forceUpdate() will cause render() to be called on the component, skippingshouldComponentUpdate(). This will trigger the normal lifecycle methods for child components, including the shouldComponentUpdate() method of each child. React will still only update the DOM if the markup changes.

Normally you should try to avoid all uses of forceUpdate() and only read from this.propsand this.state in render().

**Class Properties**

**defaultProps**

defaultProps can be defined as a property on the component class itself, to set the default props for the class. This is used for undefined props, but not for null props. For example:

class CustomButton extends React.Component {

// ...

}

CustomButton.defaultProps = {

color: 'blue'

};

If props.color is not provided, it will be set by default to 'blue':

render() {

return <CustomButton /> ; // props.color will be set to blue

}

If props.color is set to null, it will remain null:

render() {

return <CustomButton color={null} /> ; // props.color will remain null

}

**displayName**

The displayName string is used in debugging messages. JSX sets this value automatically; see [JSX in Depth](https://facebook.github.io/react/docs/jsx-in-depth.html).

**propTypes**

propTypes can be defined as a property on the component class itself, to define what types the props should be. It should be a map from prop names to types as defined in[React.PropTypes](https://facebook.github.io/react/docs/react-api.html#react.proptypes). In development mode, when an invalid value is provided for a prop, a warning will be shown in the JavaScript console. In production mode, propTypes checks are skipped for efficiency.

For example, this code ensures that the color prop is a string:

class CustomButton extends React.Component {

// ...

}

CustomButton.propTypes = {

color: React.PropTypes.string

};

We recommend using [Flow](https://flowtype.org/) when possible, to get compile-time typechecking instead of runtime typechecking. [Flow has built-in support for React](https://flowtype.org/docs/react.html) so it's easy to run static analysis on a React app.

**Instance Properties**

**props**

this.props contains the props that were defined by the caller of this component. See[Components and Props](https://facebook.github.io/react/docs/components-and-props.html) for an introduction to props.

In particular, this.props.children is a special prop, typically defined by the child tags in the JSX expression rather than in the tag itself.

**state**

The state contains data specific to this component that may change over time. The state is user-defined, and it should be a plain JavaScript object.

If you don't use it in render(), it shouldn't be on the state. For example, you can put timer IDs directly on the instance.

See [State and Lifecycle](https://facebook.github.io/react/docs/state-and-lifecycle.html) for more information about the state.

Never mutate this.state directly, as calling setState() afterwards may replace the mutation you made. Treat this.state as if it were immutable.