React 101
Codecademy
Stateless Components From Stateful Components

Stateless Components From Stateful Components

Stateless Components Inherit From Stateful Components

Let's learn a programming pattern.

Our programming pattern uses two React components: a stateful component, and a stateless component. "Stateful" describes any component that has a state property; "stateless" describes any component that does not.

In our pattern, a stateful component passes its state down to stateless components.

Build a Stateful Component Class

Let's make a stateful component pass its state to stateless component.

To make that happen we need two component classes: a stateful class, and a stateless class.

Example:

Stateful class:

Parent.js

```
import React from "react";
import ReactDOM from "react-dom";


class Parent extends React.Component {
```

```
  constructor(props) {
    super(props);
    this.state = {name: 'Frarthur'};
  }

  render() {
    return <div></div>;
  }
}
```

Build a Stateless Component Class

Great! We made a stateful component class named Parent.

Now, lets make our stateful component class.

Stateless Component:

Child.js

```
import React from "react";

export class Child extends React.Component {
  render() {
    return <h1>Hey, my name is {this.props.name}</h1>;
  }
}
```

Pass a Component's State

A <Parent /> is supposed to pass its state to a <Child />.

Setting up Parent to render Child and pass info to it.

```
import React from "react";
import ReactDOM from "react-dom";
import {Child} from "./Child";

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {name: 'Frarthur'};
  }

  render() {
    return <Child name={this.state.name} />;
  }
```

```
}

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

Don’t Update props

Great work! You just passed information from a stateful component to a stateful component to a stateless component. You will be doing a lot of that.

You learned earlier that a component can change its state by calling this.setState(). You may have been wondering: how does a component change its props?

The answer: it doesn’t!

A component should never update this.props. Look at Bad.js to see an example of what not to do.

```
import React from 'react';

class Bad extends React.Component {
  render() {
    this.props.message = 'yo'; // NOOOOOOOOOOOOO!!!
    return <h1>{this.props.message}</h1>;
  }
}
```

This is potentially confusing props and state store dynamic information. Dynamic information can change, by definition. If a component can’t change its props, then what are props for?

A React component should use props to store information that can be changed, but can only be changed by a different component.

A React component should use state to store information that the component itself can change.

Child Components Update Their Parent's State

Child Components Update their Parent's state

Here's how this works:

1. The parent component class defines a method that calls this.setState().
   For example look at .handleClick method below:

```
import React from 'react';
import ReactDOM from 'react-dom';
import { ChildClass } from './ChildClass';

class ParentClass extends React.Component {
  constructor(props) {
    super(props);

    this.state = { totalClicks: 0 };
  }

  handleClick() {
    const total = this.state.totalClicks;

    // calling handleClick will
    // result in a state change:
    this.setState(
      { totalClicks: total + 1 }
    );
  }
}
```

2. The parent component binds the newly-defined method to the current instance of the component in its constructor. This ensures that when we pass the method to the child component, it will still update the parent component.
   For, example look in constructor() method below:

```
import React from 'react';
```

```
import ReactDOM from 'react-dom';
import { ChildClass } from './ChildClass';

class ParentClass extends React.Component {
  constructor(props) {
    super(props);

    this.state = { totalClicks: 0 };

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    const total = this.state.totalClicks;

    // calling handleClick will
    // result in a state change:
    this.setState(
      { totalClicks: total + 1 }
    );
  }

  // The stateful component class passes down
  // handleClick to a stateless component class:
  render() {
    return (
      <ChildClass onClick={this.handleClick} />
    );
  }
}
```

3. Once the parent has defined a method that updates its state and bound to it, the parent then passes that method down to a child. Look in above example at the prop in render method.
4. The child receives the passed-down function, and uses it as an event handler.
   Look below for example: When a user clicks on the button a click event will fire. This will make the passed-down function get called, which will update the parent's state.

```
import React from 'react';
import ReactDOM from 'react-dom';

export class ChildClass extends React.Component {
  render() {
    return (
      // The stateless component class uses
      // the passed-down handleClick function,
      // accessed here as this.props.onClick,
      // as an event handler:
      <button onClick={this.props.onClick}>
        Click Me!
      </button>
    );
  }
}
```

Define an Event Handler

To make a child component update its parent's state, the first step is something that you've seen before: you must define a state changing method on the parent.

Child.js

```
import React from 'react';

export class Child extends React.Component {
  render() {
    return (
      <div>
        <h1>
          Hey my name is {this.props.name}!
        </h1>
        <select id="great-names">
          <option value="Frarthur">
            Frarthur
          </option>
```

```
          <option value="Gromulus">
            Gromulus
          </option>

          <option value="Thinkpiece">
            Thinkpiece
          </option>
        </select>
      </div>
    );
  }
}
```

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return <Child name={this.state.name} />
  }
}

ReactDOM.render(
  <Parent />,
```

```
  document.getElementById('app')
);
```

Notice how changeName function is defined to change the prop value to render a different name as per selected from the dropdown but it still doesn't work.

Pass the Event Handler

Update the Parent.js as such so that Child can use it in an event listener on the dropdown menu.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };
    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return <Child name={this.state.name} onChange={this.changeName} />
  }
}

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

Receive the Event Handler

```
import React from 'react';

export class Child extends React.Component {
 constructor(props) {
   super(props);
   this.handleChange = this.handleChange.bind(this);
 }
 handleChange(e) {
   const name = e.target.value;
   this.props.onChange(name);
 }
  render() {
    return (
      <div>
        <h1>
          Hey my name is {this.props.name}!
        </h1>
        <select id="great-names" onChange={this.handleChange}>
          <option value="Frarthur">
            Frarthur
          </option>

          <option value="Gromulus">
            Gromulus
          </option>

          <option value="Thinkpiece">
            Thinkpiece
          </option>
        </select>
      </div>
    );
  }
}
```

Update the child.js as such and voila we can update a parent prop from a child class.

Automatic Binding

Great work! Stateless components updating their parents' state is a React pattern that you'll see more and more. Learning to recognize it will help you understand how React apps are organized.

Child Components Update their Sibling's Props

Child Components Update Sibling Components

We saw how the pattern where a stateful, parent component passes down a prop to a stateless child component is actually part of a larger pattern: a stateful, parent component passes down an event handler to a stateless, child component. The child component then uses that event handler to update its parent's state.

Now we will explore one last pattern: a child component updates its parent's state, and the parent passes that state to a sibling component.

One Sibling to Display, Another to Change

One of the very first thing we learned about components is that they should only have one job.

Uptill now in our examples, child component was doing two jobs, rendering a dropdown and rendering the name.

Now we will break it into two components.

Now, we will have one stateless component display information, and a different stateless component offer the ability to change that information.

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
```

```
import { Sibling } from './Sibling';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };

    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return (
      <div>
        <Child
          name={this.state.name}
          onChange={this.changeName} />
        <Sibling />
      </div>
    );
  }
}

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

Child.js

```
import React from 'react';

export class Child extends React.Component {
```

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

    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(e) {
    const name = e.target.value;
    this.props.onChange(name);
  }

  render() {
    return (
      <div>
        <select
          id="great-names"
          onChange={this.handleChange}>

          <option value="Frarthur">Frarthur</option>
          <option value="Gromulus">Gromulus</option>
          <option value="Thinkpiece">Thinkpiece</option>
        </select>
      </div>
    );
  }
}
```

Sibling.js

```
import React from 'react';

export class Sibling extends React.Component {
  render() {

    return (
      <div>
        <h1>Hey, my name is Frarthur!</h1>
        <h2>Don't you think Frarthur is the prettiest name ever?</h2>
        <h2>Sure am glad that my parents picked Frarthur!</h2>
```

```
      </div>
    );
  }
}
```

As of now the dropdown does not works and name is not changed.

Pass the Right props to the Right Siblings

Parent.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Child } from './Child';
import { Sibling } from './Sibling';

class Parent extends React.Component {
  constructor(props) {
    super(props);

    this.state = { name: 'Frarthur' };

    this.changeName = this.changeName.bind(this);
  }

  changeName(newName) {
    this.setState({
      name: newName
    });
  }

  render() {
    return (
      <div>
        <Child  onChange={this.changeName} />
        <Sibling name={this.state.name} />
      </div>
    );
```

```
  }
}

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

Each siblings receive the right props.

Display information in a Sibling Component

```
import React from 'react';

export class Sibling extends React.Component {
  render() {
    const name = this.props.name;
    return (
      <div>
        <h1>Hey, my name is {name}!</h1>
        <h2>Don't you think {name} is the prettiest name ever?</h2>
        <h2>Sure am glad that my parents picked {name}!</h2>
      </div>
    );
  }
}
```

On updating the sibling.js to update the name as per the received prop our app works as required.