**React.createClass versus extends React.Component**

Two ways to do the same thing. Almost. React traditionally provided theReact.createClass method to create component classes, and released a small syntax sugar update to allow for better use with ES6 modules by extends React.Component, which extends the Component class instead of calling createClass.

These differences are subtle in places, but have quite a few interesting differences worth exploring, which will allow you to make the best decision for which is best for you.

**Syntax differences**

First, let’s explore the syntax differences by looking at two code examples and annotating them.

**React.createClass**

Here we have a const with a React class assigned, with the important render function following on to complete a typical base component definition.

import React from 'react';

const Contacts = React.createClass({

render() {

return (

<div></div>

);

}

});

export default Contacts;

**React.Component**

Let’s take the above React.createClass definition and convert it to use an ES6 class.

import React from 'react';

class Contacts extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div></div>

);

}

}

export default Contacts;

From a JavaScript perspective we’re now using ES6 classes, typically this would be used with something like Babel to compile the ES6 to ES5 to work in other browsers. With this change, we introduce the constructor, where we need to call super() to pass the props to React.Component.

For the React changes, we now create a class called “Contacts” and extend fromReact.Component instead of accessing React.createClass directly, which uses less React boilerplate and more JavaScript. This is an important change to note further changes this syntax swap brings.

**propTypes and getDefaultProps**

There are important changes in how we use and declare default props, their types and setting initial states, let’s take a look.

**React.createClass**

In the React.createClass version, the propTypes property is an Object in which we can declare the type for each prop. The getDefaultProps property is a function that returns an Object to create initial props.

import React from 'react';

const Contacts = React.createClass({

propTypes: {

},

getDefaultProps() {

return {

};

},

render() {

return (

<div></div>

);

}

});

export default Contacts;

**React.Component**

This uses propTypes as a property on the actual Contacts class instead of a property as part of the createClass definition Object. I think it’s nicer syntax to create class properties so it’s much clearer what are React APIs versus your own on the definition Object.

The getDefaultProps has now changed to just an Object property on the class calleddefaultProps, as it’s no longer a “get” function, it’s just an Object. I like this syntax as it avoids more React boilerplate, just plain JavaScript.

import React from 'react';

class Contacts extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div></div>

);

}

}

Contacts.propTypes = {

};

Contacts.defaultProps = {

};

export default Contacts;

**State differences**

State is an interesting change, now we’re using constructors the implementation of initial states changes.

**React.createClass**

We have a getInitialState function, which simply returns an Object of initial states.

import React from 'react';

const Contacts = React.createClass({

getInitialState () {

return {

};

},

render() {

return (

<div></div>

);

}

});

export default Contacts;

**React.Component**

The getInitialState function is deceased, and now we declare all state as a simple initialisation property in the constructor, which I think is much more JavaScript-like and less “API” driven.

import React from 'react';

class Contacts extends React.Component {

constructor(props) {

super(props);

this.state = {

};

}

render() {

return (

<div></div>

);

}

}

export default Contacts;

**“this” differences**

Using React.createClass will automatically bind this values correctly for us, but changes when using ES6 classes affect this.

**React.createClass**

Note the onClick declaration with this.handleClick bound. When this method gets called React will apply the right execution context to handleClick.

import React from 'react';

const Contacts = React.createClass({

handleClick() {

console.log(this); // React Component instance

},

render() {

return (

<div onClick={this.handleClick}></div>

);

}

});

export default Contacts;

**React.Component**

With ES6 classes this is slightly different, properties of the class do not automatically bind to the React class instance.

import React from 'react';

class Contacts extends React.Component {

constructor(props) {

super(props);

}

handleClick() {

console.log(this); // null

}

render() {

return (

<div onClick={this.handleClick}></div>

);

}

}

export default Contacts;

There are a few ways we could bind the right context, here’s how we could bind inline:

import React from 'react';

class Contacts extends React.Component {

constructor(props) {

super(props);

}

handleClick() {

console.log(this); // React Component instance

}

render() {

return (

<div onClick={this.handleClick.bind(this)}></div>

);

}

}

export default Contacts;

Alternatively we could change the context of this.handleClick inside the constructorto avoid inline repetition, which may be a better approach if moving to this syntax to avoid touching JSX at all:

import React from 'react';

class Contacts extends React.Component {

constructor(props) {

super(props);

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

}

handleClick() {

console.log(this); // React Component instance

}

render() {

return (

<div onClick={this.handleClick}></div>

);

}

}

export default Contacts;

**Mixins**

React mixins are no longer supported when using React components written in ES6.

**React.createClass**

With React.createClass we can add mixins to components using a mixins property which takes an Array of available mixins. These then extend the component class.

import React from 'react';

var SomeMixin = {

doSomething() {

}

};

const Contacts = React.createClass({

mixins: [SomeMixin],

handleClick() {

this.doSomething(); // use mixin

},

render() {

return (

<div onClick={this.handleClick}></div>

);

}

});

export default Contacts;

**React.Component**

Mixins aren’t supported in ES6 classes.

**Recommendations**

Facebook does suggest the future removal of React.createClass completely in favour of ES6 classes - ([source](https://facebook.github.io/react/blog/2015/03/10/react-v0.13.html)). For now, use what makes sense, they’re both just syntax with different semantics that do the same thing - they’re both classes!

AddThis Sharing Buttons