I’ve been using [React](http://facebook.github.io/react/) a bunch lately, and as I build bigger applications with it, I’ve needed components that need the same functionality. So, I spent an hour or so messing around with [mixins](http://facebook.github.io/react/docs/reusable-components.html" \l "mixins) and thought I would share what I found.

Why Mixins?

React shies away from subclassing components, but as we all know, repeating yourself is just as bad as writing the same code over and over. So to give the same functionality to multiple components, you encapsulate it into a mixin and include it in your classes. One could go so far as to say React favors composition over inheritance. Oh yeah. I went there.

Making a Simple Mixin

Let’s say we are creating an app and we know that for some reason we need to set a default name prop on several components. Don’t ask why; it’s just easier that way.

Rather than write the same getDefaultProps method on each component, we can create a mixin like so:

var DefaultNameMixin = {

getDefaultProps: function () {

return {name: "Skippy"};

}

};

There’s nothing magical about this: it’s just a simple object.

Adding it to a React Component

To use our mixin, we simply ensure our component has a mixins property and wrap our mixin in an array as its value:

var ComponentOne = React.createClass({

mixins: [DefaultNameMixin],

render: function() {

return <h2>Hello {this.props.name}</h2>;

}

});

React.renderComponent(<ComponentOne />, document.body);

[JSFiddle example](http://jsfiddle.net/veddermatic/qzh0ug5y/1/) of a simple mixin in action.

Look Ma, Reuse!

As you can imagine, we can include our mixin in any number of components:

var ComponentTwo = React.createClass({

mixins: [DefaultNameMixin],

render: function () {

return (

<div>

<h4>{this.props.name}</h4>

<p>Favorite food: {this.props.food}</p>

</div>

);

}

});

[JSFiddle example](http://jsfiddle.net/veddermatic/qzh0ug5y/2/) of our goofy mixin being used by multiple components.

Lifecycle Methods: Duplicate Away!

If your mixin includes one of the [React lifecycle methods](http://facebook.github.io/react/docs/component-specs.html#lifecycle-methods), don’t fret: you didn’t “use it up.” You can still use these methods inside your components, and both will be called:

var DefaultNameMixin = {

getDefaultProps: function () {

return {name: "Skippy"};

}

};

var ComponentTwo = React.createClass({

mixins: [DefaultNameMixin],

getDefaultProps: function () {

return {food: "Pancakes"};

},

render: function () {

return (

<div>

<h4>{this.props.name}</h4>

<p>Favorite food: {this.props.food}</p>

</div>

);

}

});

[JSFiddle example](http://jsfiddle.net/veddermatic/qzh0ug5y/3/) showing both default props being set.

Both instances of getDefaultProps will get called, and so we will wind up with a default name of Skippy and favorite food of Pancakes. Any of the [React lifecyle methods](http://facebook.github.io/react/docs/component-specs.html#lifecycle-methods)or [Component Spec methods or properties](http://facebook.github.io/react/docs/component-specs.html#component-specifications) can be duplicated in a mixin without issues, with the following exceptions:

* render : Having more than one render method doesn’t make sense. React will yell at you if you try having two of them:
* Uncaught Error: Invariant Violation: ReactCompositeComponentInterface:
* You are attempting to define `render` on your component more than once.
* This conflict may be due to a mixin.
* displayName : This can be set multiple times without error via mixin, but it is overwritten, so the last value “wins”.

It’s worth pointing out that mixins is something that you *can* define multiple times, meaning that mixins can include mixins:

var UselessMixin = {

componentDidMount: function () {

console.log("asdas");

}

};

var LolMixin = {

mixins: [UselessMixin]

};

var PantsOpinion = React.createClass({

mixins: [LolMixin],

render: function () {

return (<p>I dislike pants</p>);

}

});

React.renderComponent(<PantsOpinion />, document.body);

This will very happily log “asdas” to the console when the component is mounted.[JSFiddle example](http://jsfiddle.net/qzh0ug5y/17/)

Multiple Mixins

The fact that our mixins are inside an array is a subtle hint that we can include more than one at a time:

var DefaultNameMixin = {

getDefaultProps: function () {

return {name: "Lizie"};

}

};

var DefaultFoodMixin = {

getDefaultProps: function () {

return {food: "Pancakes"};

}

};

var ComponentTwo = React.createClass({

mixins: [DefaultNameMixin, DefaultFoodMixin],

render: function () {

return (

<div>

<h4>{this.props.name}</h4>

<p>Favorite food: {this.props.food}</p>

</div>

);

}

});

[JSFiddle example](http://jsfiddle.net/veddermatic/92nwqss2/1/) of multiple mixins in action.

Gotchas

There are a few things to note when using mixins that can cause headaches. Fortunately, the list seems pretty small. Here are the few I’ve found during my experimentation.

Gotcha: Setting The Same Prop or State

The ability to have multiple calls to getDefaultProps (and getIntialState as well) *can*get you into trouble if you attempt to define the same prop in different places:

// set the name prop here...

var DefaultNameMixin = {

getDefaultProps: function () {

return {name: "Skippy"};

}

};

var ComponentTwo = React.createClass({

mixins: [DefaultNameMixin],

// ... and set the name prop here

getDefaultProps: function () {

return {

food: "Pancakes",

name: "Lizzie"

};

},

render: function () {

return (

<div>

<h4>{this.props.name}</h4>

<p>Favorite food: {this.props.food}</p>

</div>

);

}

});

You’ll get this in your console:

Uncaught Error: Invariant Violation: mergeObjectsWithNoDuplicateKeys():

Tried to merge two objects with the same key: name

So don’t do that. OK? This same error will happen if we attempt to set the same statekey in multiple calls to getIntialState. So don’t do that either.

[JSFiddle example](http://jsfiddle.net/veddermatic/qzh0ug5y/4/) (look at your js console to see the error)

Gotcha: Defining The Same Method

Creating a method of the same name in multiple mixins or a mixin and a component will throw the same error as if you attempted to duplicate the render method:

var LogOnMountMixin = {

componentDidMount: function () {

console.log("mixin mount method");

this.logBlah()

},

// add a logBlah method here...

logBlah: function () {

console.log("blah");

}

};

var MoreLogOnMountMixin = {

componentDidMount: function () {

console.log("another mixin mount method");

},

// ... and again here.

logBlah: function () {

console.log("something other than blah");

}

};

This will generate the spot on error message:

Uncaught Error: Invariant Violation: ReactCompositeComponentInterface:

You are attempting to define `logBlah` on your component more than once.

This conflict may be due to a mixin.

Multiple Lifecyle Method Call Order of Execution

What happens when we have a lifecyle method in both the component and a mixin?

var LogOnMountMixin = {

componentDidMount: function () {

console.log("mixin mount method");

}

};

var ComponentOne = React.createClass({

mixins: [LogOnMountMixin],

componentDidMount: function () {

console.log("component one mount method");

},

render: function() {

return <h2>Hello {this.props.name}</h2>;

}

});

Our mixin methods always run first, as we see this in the console:

mixin mount method

component one mount method

When more than one mixin includes the same lifecyle method, they are called in left to right (low index to high) order as one would expect from an array:

var LogOnMountMixin = {

componentDidMount: function () {

console.log("mixin mount method");

}

};

var MoreLogOnMountMixin = {

componentDidMount: function () {

console.log("another mixin mount method");

}

};

var ComponentOne = React.createClass({

mixins: [MoreLogOnMountMixin, LogOnMountMixin],

componentDidMount: function () {

console.log("component one mount method");

},

...

var ComponentTwo = React.createClass({

mixins: [LogOnMountMixin, MoreLogOnMountMixin],

componentDidMount: function () {

console.log("component two mount method");

},

...

This would give us the following console output:

another mixin mount method

mixin mount method

component one mount method

mixin mount method

another mixin mount method

component two mount method

* [email](mailto:info@simblestudios.com)