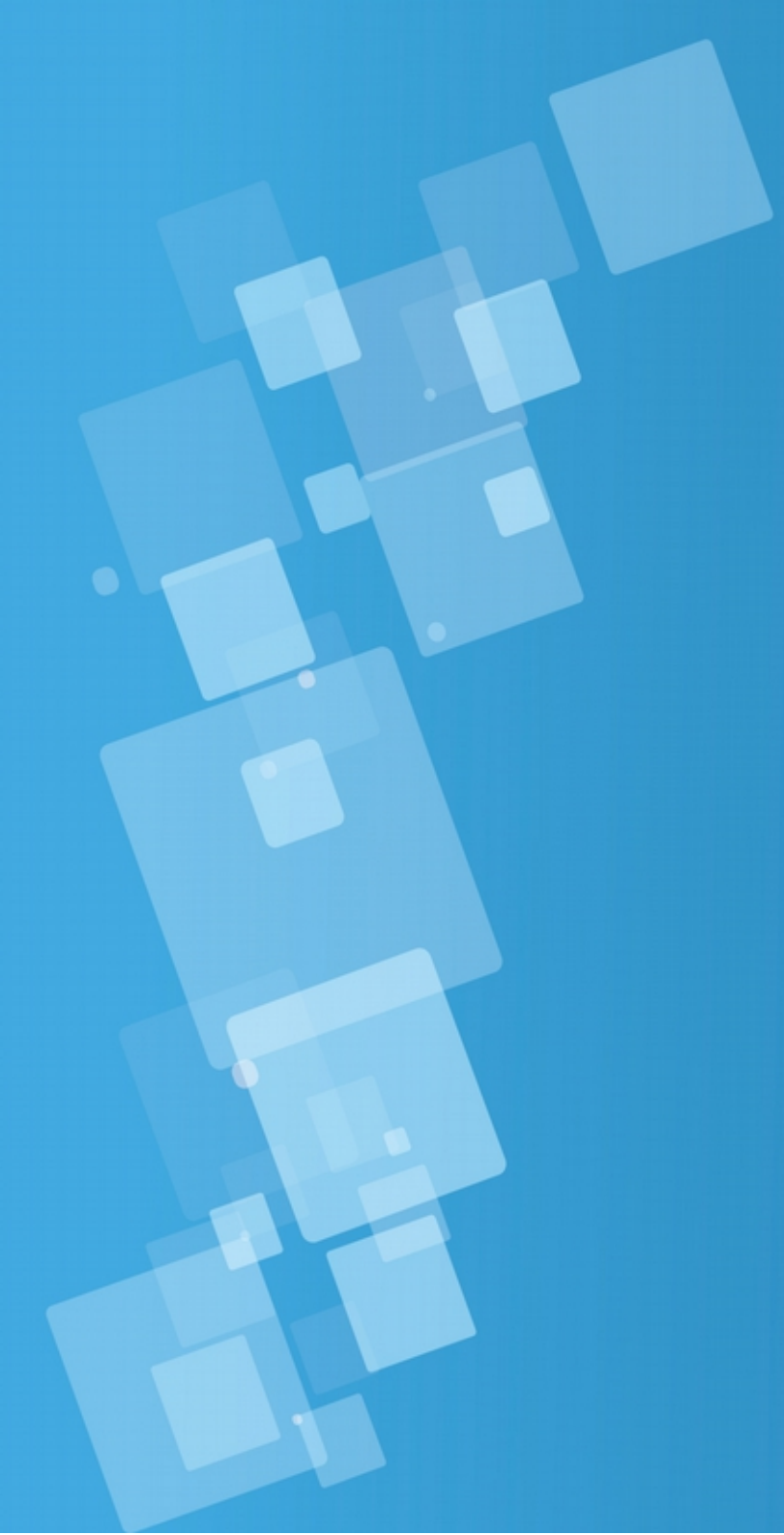


React.js

Binding functions

Leandro.frigerio@devugees.org

Intro

React makes it **painless** to create interactive UIs.
Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

On a basic level it is taking a component's state, a Javascript object, and using Javascript functions to manipulate that state. The fact that it relies so heavily on plain Javascript can be good and bad depending on how strong your Javascript skills are.

The ideas of classes, event handling, importing, exporting, callback functions, etc. are all used in React and if you aren't comfortable with them, then your time with React may be rough until you shore up those skills.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

THIS

One Javascript concept in particular is used a lot in React applications and can challenge you the first (...) time you encounter it: **this**

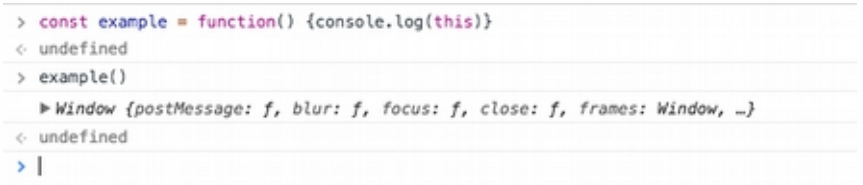
React doesn't treat 'this' any differently than Javascript. It's just that you may encounter it more in React since it uses classes and constructors to create components.

Those **components** have methods attached to them.
Components contain their own **state** and pass down properties to **child** components.

All of these things are referred to using **this** so if you write a React application, you will be using **this** a lot.

Let's try this

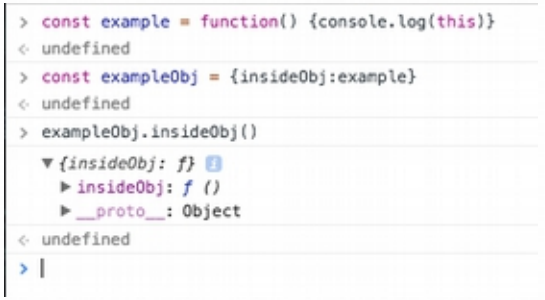
Since **this** is used in React the same way it is used in Javascript, it may be helpful to explain why **this** is used at all. It comes down to **context**.
When you invoke a function, 'this' determines which object is the focus of the function.



This is just creating a variable whose value is a function and logs out the value of **this**. When you invoke the **example()** function, it returns the value of **this** within the context, which in this case is the **Window** object. The global object that all Javascript functions run in unless otherwise specified.

One step beyond

Here the original but this time instead of invoking it in the global scope, we made it the value inside of an object. In Javascript, objects define their own scope, so what happens when I invoke this function?



What is logged to the console is now an “**Object**”, not the “**Window**” like it was before. The context for this has changed because it was invoked from somewhere else

This **context switching** can be hard to keep track of but is very important and will break your React application if you don’t remember to set it correctly.

In React

Here we have an App component that renders a button that when clicked calls **a handleClick**() method that will change the state of App to track the number of times the button has been clicked.

Two console.log are placed, one in the render and one inside the **handleClick()** method

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {

  constructor(props) {
      super(props);
      this.state = {
        counter: 0
  };
}

  handleClick() {
      console.log('From handleClick()', this);
    }

  render() {
    console.log('From render()', this);
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <div className="App-intro">
          <button onClick={this.handleClick}>Click Me!</button>
          <p>Number of Times Clicked = {this.state.counter}</p>
        </div>
      </div>
    );
  }
}
```

The problem

```
From render() ▶App {props: {…}, context: {…}, refs: {…}, updater: {…}, state: {…}, …}   App.js:19
From handleClick() undefined                                                              App.js:15
>
```

From inside render() **this** is set to the App (an instance of this App component), but at this point, **this** in the function is **null**.

If we try to use the function and setState,

this.setState(prevState => { return {counter: prevState.counter + 1}});

it return an error

The solution

How do you change what 'this' refers to?
Luckily ES6 has a few built-in methods that do exactly that:
the .bind() method

Every Javascript object has three built-in methods that can change the **this** context for a function.
The first two, .call() and .apply() are similar in that the first argument you pass to them is the new context and then you can pass them individual arguments or an array respectively. What they also have in common is that the function you call them on is then run immediately. That won't do in this case, since we want handleClick() to only be run when the button is clicked. The third built-in Javascript method can do this.

The .bind() method is similar to the other two in that you pass it the context you want to bind the function to, but it does not immediately run the function.

Instead a copy of the function with the switched context is returned.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind

The constructor

Since the **this** context of App is the one that **handleClick**() should be using, we have to use the .**bind**() method on it in the constructor for that component.
If this line is added to the constructor, all will work as expected:

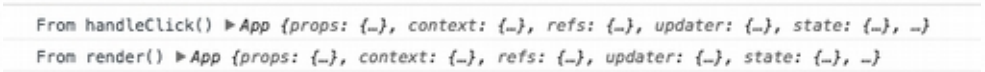
```
class App extends Component {

  constructor(props) {
      super(props);
      this.state = {
        counter: 0
  };
  this.handleClick = this.handleClick.bind(this);
}
```

The first this.handleClick refers to the handleClick() method. Since this is done in the constructor, ‘this’ refers to App. The second this.handleClick is also referring to the same handleClick() method but we are now calling .bind() on it. The final ‘this’ is the context we are passing to .bind() and it is referring to the App context.

Result

We are setting the value of **handleClick**() to equal the result of calling .**bind**() on **handleClick**(), and passing .**bind**() the context of **this** which refers to the App.

This will have the result of making the context for **handleClick**() be **App** whenever it is called which is what we want since that is also where the **state** is located and that is what we are trying to change.



Wrap-up: approch overview 1

1. Use React.createClass

If you use React.createClass, React autobinds all functions to this.
So the this keyword is bound to your component's instance automatically:

```
// This magically works with React.createClass
// because `this` is bound for you.
onChange={this.handleChange}
```

However, with the advent of ES6 classes, this non-standard approach to creating classes isn't the future of React. In fact, createClass is likely to be extracted from React core in a future release.

2. Bind in Render

If you use an ES6 class, React no longer autobinds. You need to .bind(this)

Why you want to use arrow function in Class Property

Use Arrow Function in Class Property has advantages

Arrow functions adopt the **this** binding of the enclosing scope (in other words, they don't change the meaning of this), so things just work automatically.

It avoids the performance issues of approaches #2 and #3.

It avoids the repetition in approach #4.

It's straightforward to refactor from the ES5 createClass style into this style by converting relevant functions into arrow functions.

Wrap-up: approch overview 2

3. Use Arrow Function in Render
This approach is similar to #2. You can avoid changing the this context by using an arrow function in render: onChange={e => this.handleChange(e)}

4. Bind in Constructor
One way to avoid binding in render is to bind in the constructor

5. Use Arrow Function in Class Property
handleChange = () => { … }

Conclusion

Keeping track of **this** can be tough as applications get more complex.

It is often a stumbling block for those new and not so new to Javascript and React. Just remember that there are functions like **bind()** to help you reassign the value of **this** when needed.

Resources

- Binding functions in React
  https://codeburst.io/binding-functions-in-react-b168d2d006cb
- MDN .bind
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind
- MDN This Operator
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this
- React binding patterns
  https://medium.freecodecamp.org/react-binding-patterns-5-approaches-for-handling-this-92c651b5af56
- Understanding Bind
  https://www.smashingmagazine.com/2014/01/understanding-javascript-function-prototype-bind/