Understanding “this”

Dan Geabunea

Passionate Software Developer

@romaniancoder | www.dangeabunea.com

Overview

Thinking about “this” based on various scenarios

- Global context
- In constructor functions
- Methods
- Arrow Functions

Using “bind()”, “call()” & “apply()” to provide a value for “this”

this

Is the current execution context of a function

The object which “owns” the function

The object that is executing the function

Scenario-Based Approach

Global context

Strict/Non-strict mode

Methods

Constructor Functions

Arrow-functions

this

1. In non-arrow functions, “this” usually represents the object invoking the function

2. In arrow functions, “this” refers to the parent function’s scope

Using “this” in Global Context

“this” – Global Context

At a top level of a script, “this” refers to the global context

- Window object in HTML
- Global in Node

Top Level Functions

```
function printBrowser() {
  console.log(this.navigator.userAgent);
}

printBrowser();  // Mozilla, Chrome, Edge…
```

The value of “this” changes if you are using strict mode. In strict mode, “this” is undefined

Strict Mode

```
function printBrowser() {
  “use strict”;
  console.log(this.navigator.userAgent);
}

printBrowser();
// Cannot read properties of undefined
```

```
function Helicopter(speed) {
  this.speed = speed;
}


const h = new Helicopter(320);
console.log(h.speed);
```

Constructor Functions

When a function is used as a constructor, its “this” is bound to the new object being created

Using “this” in Top-Level Functions

Using “this” in Methods

“this” – in Methods

Refers to the object which is invoking the current method

This is determined at run-time, when you call the method

Objects

```
let airport = {
  name: 'Heathrow',
  nbDeparturesToday: 50,

  addDeparture() {
    this.nbDeparturesToday++;
    console.log(this.nbDeparturesToday);
  },
};

airport.addDeparture();
// airport.nbDeparturesToday = 51
```

Classes

```
class Airport {
  nbDeparturesToday = 0;

  constructor(name) {
    this.name = name;
  }

  addDeparture() {
    this.nbDeparturesToday++;
    console.log(this.nbDeparturesToday);
  }
}

const heathrow = new Airport('Heathrow');
heathrow.addDeparture();
// heathrow.nbDeparturesToday = 1
```

Using “this” in Methods

Using “this” in Arrow Functions

“this” – in Arrow Functions

Arrow functions do not have their own binding to “this” – they inherit the value of “this” from the parent scope

Arrow functions are not suitable to use in every scenario, especially as methods

Default Behavior

```
const printBrowser = () => {
  console.log(this.navigator.userAgent);
};

printBrowser(); // Will print the name of the browser
```

Default Behavior

```
const aircraft = {
  model: 'Airbus A330',
  capacity: 350,

  printModel: () => {
    console.log(this.model);
  },
};

aircraft.printModel();
// undefined
```

Behavior in Classes

```
class Aircraft {
  constructor(model, capacity) {
    this.model = model;
    this.capacity = capacity;
  }

  printModel = () => {
    console.log(this.model);
  };
}

const b737 = new Aircraft('Boeing 737', 190);
b737.printModel();

// Boeing 737
```

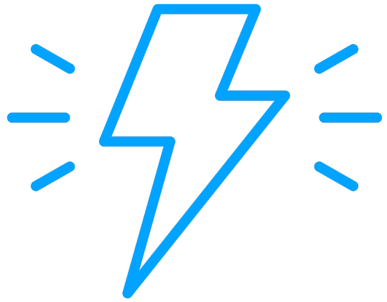
Behavior in Classes – behind the Scenes

```
function Aircraft(model, capacity) {
  this.model = model;
  this.capacity = capacity;

  this.printModel = () => {
    console.log(this.model);
  };
}


const b737 = new Aircraft('Boeing 737', 190);
b737.printModel(); // Boeing 737
```

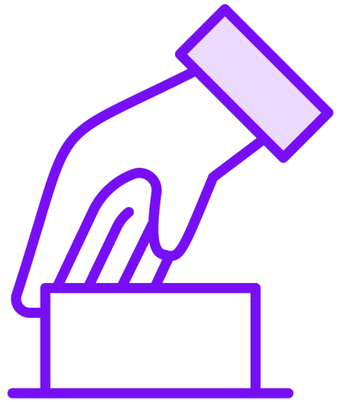
Using “this” in Arrow Functions

Controlling the Value of “this” Using bind(), call() and apply()

Understanding bind(), call(), apply()



Methods defined on the Function prototype used to invoke functions



You can provide the value for the “this” binding

```
const printModel = function () {
  console.log(this.model);
};

printModel(); // undefined

const workingPrintModel = printModel.bind({model: 'A380'});
workingPrintModel();
```

bind()

Creates a new function that has its “this” set to the provided value

Function.prototype.bind(thisObj)
Function.prototype.bind(thisObj, arg1, arg2...)

```
const printModel = function () {
  console.log(this.model);
};

printModel(); // undefined

const workingPrintModel = printModel.bind({model: 'A380'});
workingPrintModel(); // A380
```

bind()

Creates a new function that has its “this” set to the provided value

Function.prototype.bind(thisObj)
Function.prototype.bind(thisObj, arg1, arg2...)

```
const aircraft = { model: 'Airbus A330’, totalSeats: 350, seatsOccupied: 100 };


const addPassengers = function (nbPassengers) {
  const newCount = this.seatsOccupied + nbPassengers;
  if (newCount <= this.totalSeats) { this.seatsOccupied = newCount;}
};


addPassengers(3); // seatsOccupied: 100
addPassengers.call(aircraft, 3); // seatsOccupied: 103
```

call()

A function that can change the value of “this” when invoking a function

Function.prototype.call(thisObj)
Function.prototype.call(thisObj, args1, args2, ...)

```
const aircraft = { model: 'Airbus A330’, totalSeats: 350, seatsOccupied: 100 };


const addPassengers = function (nbPassengers) {
  const newCount = this.seatsOccupied + nbPassengers;
  if (newCount <= this.totalSeats) { this.seatsOccupied = newCount;}
};


addPassengers.apply(aircraft, [3]); // seatsOccupied: 103
```

apply()

Very similar to call()
Can pass an array as an argument list instead of using a rest parameter for arguments

Function.prototype.apply(thisObj)
Function.prototype.apply(thisObj, [arg1, arg2...])

Changing the Value of “this” Using bind(), call() & apply()

Summary

The “this” object takes many shapes in JavaScript

In a simplified way it usually refers to the object who is invoking the function

Arrow function inherit the “this” binding from their parent scope

You can control the value of “this” by using the bind(), call() and apply() methods

Practice makes perfect

Up Next:

Working With Asynchronous Functions