CodeX · Follow publication

CODEX

The prototype chain: how JavaScript really works

5 min read · Feb 19, 2021

Adam Reeves Follow

Listen Share More

If, like me, your previous programming experience was in Object Oriented languages like Java or C#, JavaScript's prototype based approach to inheritance might take a minute to get your head around.

You can get by just fine without delving too deeply, or even thinking about prototypes. Where it starts getting more interesting though, is when we throw constructor functions and classes into the mix.

So, let's dive in 🤿

What is a prototype?

Every object in JavaScript can be linked to a prototype object which is the mechanism through which inheritance is provided.

You can see the prototype an object is pointing to using the `__proto__` property.

As a side note: there is some controversy around using `__proto__` and it's actually deprecated in favour of `Object.getPrototypeOf()` . You can read more about it here.

When a property or method is called on an object, if the object doesn't contain it, the JavaScript engine will then look to the object's prototype to resolve the property or method.

You may have seen the term **prototype chain** before. This is where the prototype of an object itself can also contain a prototype and so on.

If an object doesn't contain a property or method, the JavaScript engine will continue up the prototype chain to try and resolve it.

Constructor functions

You may have seen some functions in JavaScript that are called with the `new` key word, for example `new Date()` — these are **constructor functions.** Keeping in theme with the rest of the article, here's what a constructor function might look like and how it can be used:

```
function Swan(colour, mood) {
    this.colour = colour;
    this.mood = mood;
}
```

```
let swan = new Swan('white', 'angry');
```

The above code block creates an object, `swan` using the constructor function `Swan` with the properties `colour` and `mood` set to `'white'` and `'angry'` respectively. Notice the convention of capitalising constructor functions.

You may have noticed the use of the `this` keyword and if you've spent much time with JavaScript or other Object Oriented languages you've probably seen it used before. Be careful though, it might not behave how you expect in JavaScript compared to OO languages like Java.

this

We won't go into too much detail on `this` as it's one of the more confusing areas of JavaScript associated with binding and could easily have a post dedicated to it (let me know if you'd find this helpful!).

To cut a long story short, the value of `this` can vary depending on where a function is called.

When used in the context of a constructor function like the above example, the JavaScript engine creates an internal `this` context which is bound to the new object being created using the constructor function — `swan` in our case.

Inheritance using the prototype chain

> In JavaScript, all functions have a Prototype property and all objects have a `__proto__` property that points to the prototype of their constructor function.

The `__proto__` property can be thought of as pointing to the prototype of the function it was created from.

Remember, we can also obtain the prototype of an object using `Object.getPrototypeOf(yourObject)` and it's better practice to do so.

Let's explore our previous example a bit more:

```
function Swan(colour, mood) {
    this.colour = colour;
    this.mood = mood;
}

let swan = new Swan('white', 'angry')

Swan.prototype.isSassy = () => true;
```

The Swan has a prototype property that we add the `isSassy()` function too.

When we create the `swan` object using the `Swan` constructor function, this means the `swan` object's `__proto__` property will point to the prototype of `Swan` .

Give it a go. You can copy and paste the above code snippet and view the prototype of `swan` using `swan.__proto__` . This will return:

```
> swan.__proto__
{isSassy: ƒ, constructor: ƒ}
    isSassy: () => true
    constructor: ƒ Swan(colour, mood)
    __proto__: Object
```

The `constructor` of `swan.__proto__` tells us that it was constructed using the `Swan` function and we can also see that the `isSassy()` function is on the prototype `swan.__proto__` is linking to as expected.

One useful features of an object created with a constructor is it allows us to check it's type:

```
swan instanceof Swan //returns true
```

MDN defines the `instanceof` operator as follows:

> *The* ***`instanceof`*** ***operator*** *tests to see if the* `prototype` *property of a constructor appears anywhere in the prototype chain of an object. The return value is a boolean value.*

Applying that to our example:

The `Swan` constructor has the following prototype:

```
Swan(colour, mood) {
    this.colour = colour;
    this.mood = mood;
}
```

and `swan.__proto__` has the following constructor property as we previously demonstrated:

```
Swan(colour, mood) {
    this.colour = colour;
    this.mood = mood;
}
```

Therefore, as the `Swan` prototype appears in the prototype chain of `swan` the boolean `true` is returned.

One important thing to note is that the entire prototype chain is checked. Bear this in mind if you have more complex prototype chains and also, remember that by default all new objects `__proto__` property point to `Object.prototype` in their prototype chain so checking if `<arbitraryObject> instanceof Object` will always return true.

Classes in JavaScript aren't what you think they are

So now we've covered the fundamentals of prototypes and constructors there's one more thing I wanted to talk about in this post.

When I first started working with JavaScript professionally, fresh out of university where a lot of time is spent on widely used Object Oriented languages (mainly Java in my case), it's incredibly easy to see classes in JavaScript and start connecting the dots.

In Object Oriented languages such as Java, a class generally serves as a blueprint and — excluding any static behaviour — whenever a new object is instantiated, it is instantiated as its own copy and occupies its own space in memory.

In JavaScript however, a class is just syntactic sugar and is still in fact, a function. When a class is 'instantiated', the new object actually has a prototype that is pointing to the class methods (which are in fact, properties on a prototype!) and that contains a constructor referencing it— sound familiar?

To demonstrate my point, look at the following:

```
class RubberDuckClass {
    constructor(colour) {
        this.colour = colour
    }
    doesFloat() {
        return true;
    }
}

function RubberDuckConstructor(colour) {
    this.colour = colour;
}

RubberDuckConstructor.prototype.doesFloat = () => true;

let rubberDuckClassObject = new RubberDuckClass('yellow');
let RubberDuckConstructorObject = new RubberDuckConstructor('blue');
```

In this example, we can actually prove that a class is in fact nothing more than a function wearing fancy clothes

rubberDuckClassObject:

- has a `colour` property set to argument ('yellow')
- prototype:
  - constructor references `RubberDuckClass`
  - has `doesFloat()` function

rubberDuckConstructorObject:

- has a `colour` property set to argument ('blue')
- prototype:
  - constructor references `RubberDuckConstructor`
  - has `doesFloat()` function

Although on the surface, JavaScript classes look like they're behaving as we'd expect in the conventional class based design pattern; under the hood, classes are still nothing more than functions with inheritance capabilities provided by the prototype mechanism.

And, in my opinion, this is why classes in JavaScript can be misleading. Eventually a time will come when you as a developer will need to dig a bit deeper into the language, and it's at that point you'll be in for a shock when you discover classes might not be what you thought they were.

JavaScript Inheritance Programming Coding

Follow

Published in CodeX

27K followers · Last published 21 hours ago

Everything connected with Tech & Code. Follow to join our 1M+ monthly readers

Follow

Written by Adam Reeves

16 followers · 7 following

Professional Software Engineer, man of many hobbies

No responses yet

sahil siddiqui

What are your thoughts?

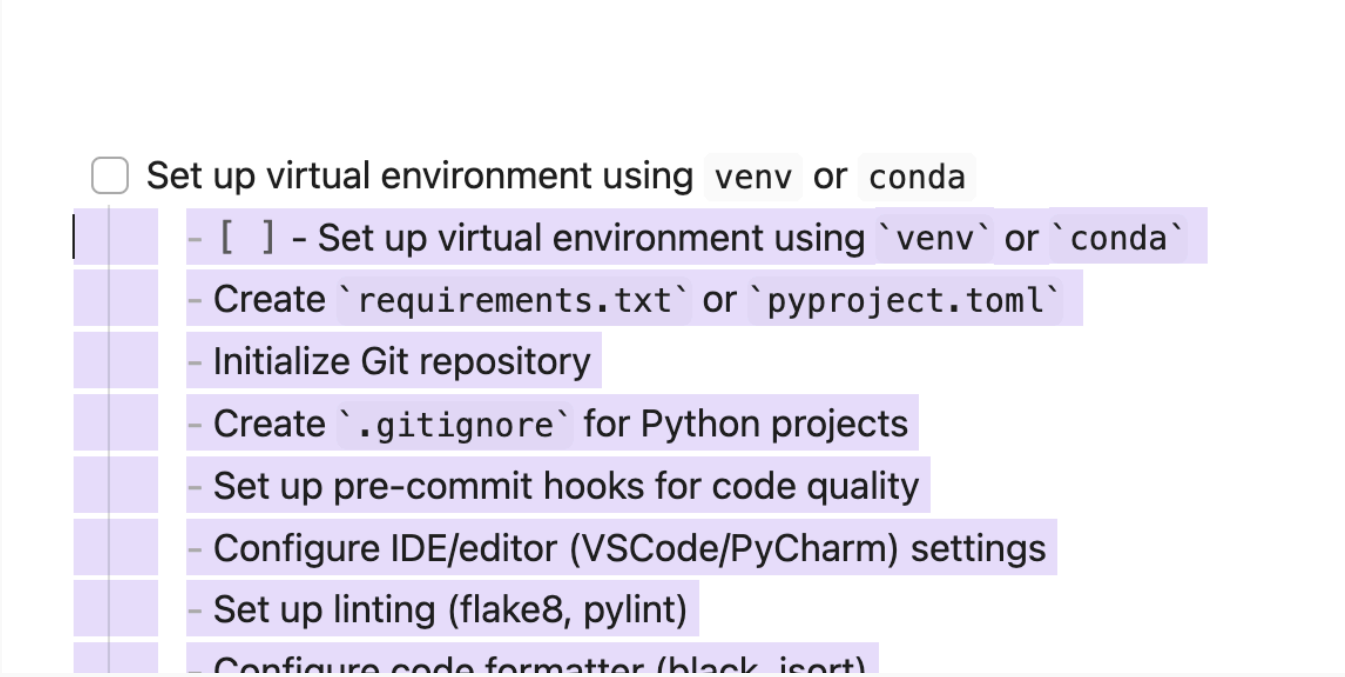
More from Adam Reeves and CodeX

In CodeX by Adam Reeves

Binding in JS explained

Digging into the elusive `this` in JS

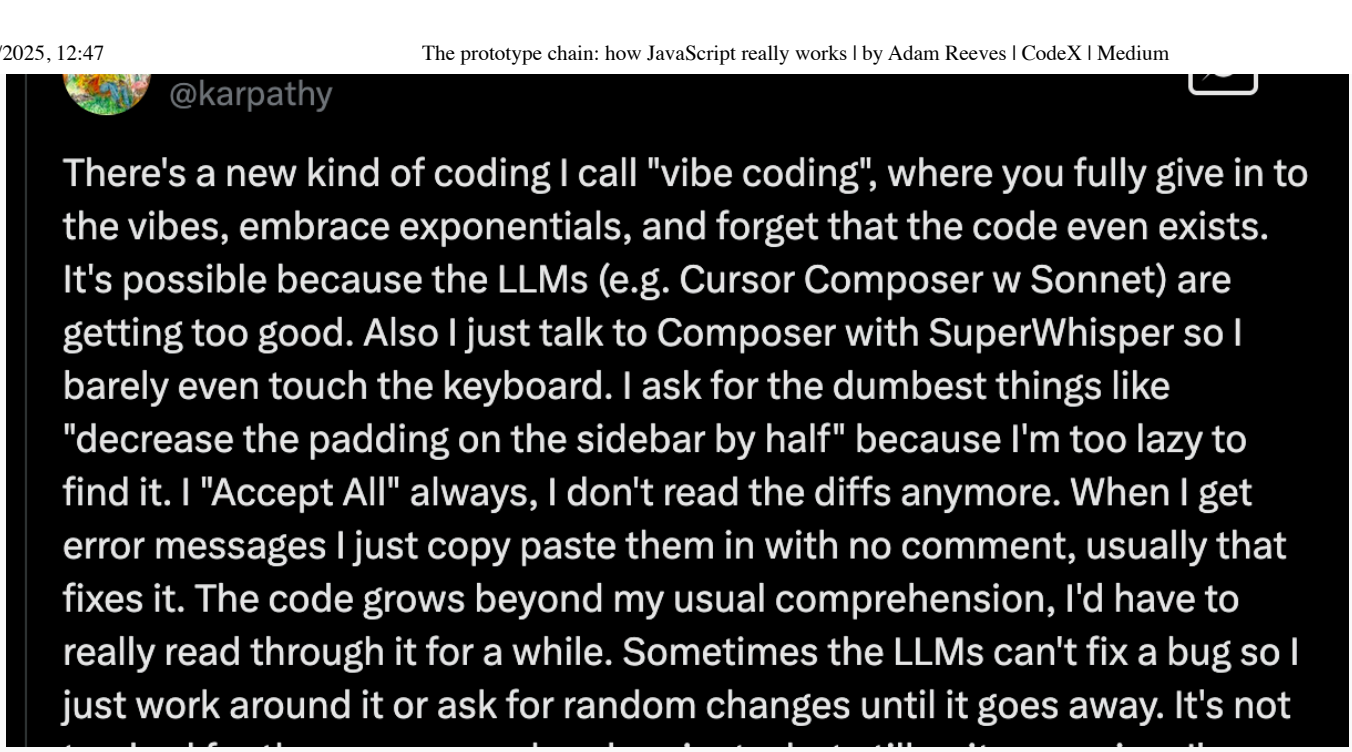
Feb 28, 2021 106 1



In CodeX by AI Rabbit

Goodbye Obsidian

Feb 6 1.5K 73



In CodeX by Austin Starks

I “vibe-coded” over 160,000 lines of code. It IS real.

When I was getting my MS from CMU and coding up the algorithmic trading platform NextTrade, I wrote every single goddamn line of code...

Mar 2 990 57

In JavaScript in Plain English by Adam Reeves

Property Descriptors in JavaScript

How you can use property descriptors to give your code more control.

Mar 6, 2021 45

See all from Adam Reeves

See all from CodeX

Recommended from Medium

Farid Vatani

How JavaScript Web Workers Keep Your UI Fast and Responsive

Have you ever had the entire page freeze after clicking a button on a website? JavaScript typically runs in a single thread. Your user...

Apr 26

In Front-end World by Priyanshu Rajput

The Future of JavaScript: A Look at ECMAScript 2025 and Beyond

JavaScript is evolving at an unprecedented pace, continuously shaping the modern web. With each new ECMAScript (ES) release, developers...

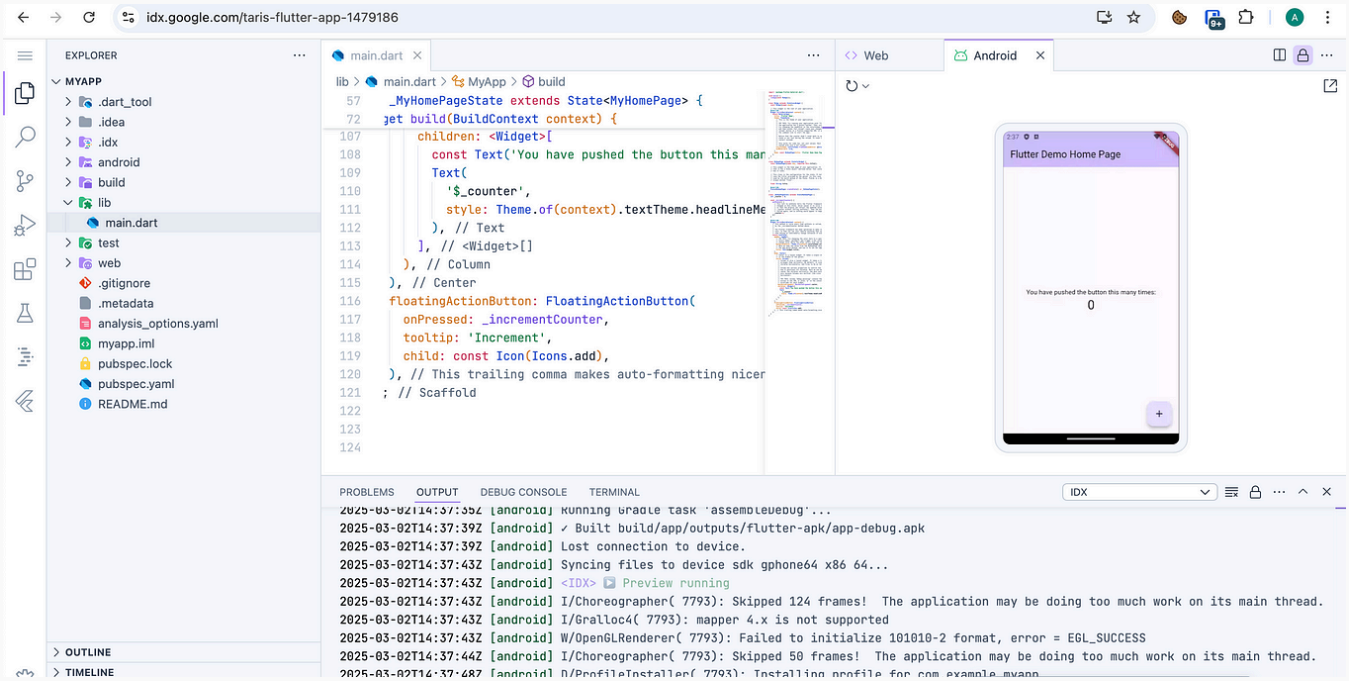
Feb 22 42

Suneel Yadav

🚀 Mastering call(), apply(), and bind() in JavaScript—For Frontend Developers!

As a JavaScript and React developer, I've struggled with interview questions around call(), apply(), and bind(). These simple-sounding...

Apr 27 10



In Coding Beauty by Tari Ibaba

This new IDE from Google is an absolute game changer

This new IDE from Google is seriously revolutionary.

Mar 12 5.1K 300

In JavaScript in Plain English by Rahul Kaklotar

You Really Don't Know These JavaScript Tricks

JavaScript is the backbone of modern web development, but even experienced developers may overlook some of its powerful features and...

Apr 30 50 3

In Full Stack Forge by Daniel Scott

7 React Patterns That Made Me a Better Front-End Developer

And Probably Saved Me From Smashing My Laptop Into a Wall

Apr 11 873 27

See more recommendations