__proto__ vs prototype in JavaScript

3 min read · May 16, 2019

Venkat Iyengar Follow

The topic of __proto__ vs prototype is admittedly one of the most baffling in JavaScript and certainly not easy to wrap your head around. There's a lot to it, but as I've noticed over time, the most critical thing when learning to code is to get your fundamentals sound. So in this article, all we're going to do is to go over very rudimentary examples, just so you understand the very basics of what the dunder proto (__proto__) and prototype properties to.

Tip: the best way to really get to grips with it is to open up your console and try it out yourself, to actually see what's going on.

Before even getting started, here's the simplest explanation I could think of, of the difference between the two:

- The __proto__ is an object within **every** object that points out (references) the prototype that has been set for that object. `__proto__` is the actual object that is used in the lookup chain to resolve methods, etc.
- The prototype property is only present for functions and is a property that's set only if you're using the 'new' keyword when creating objects with this (constructor) function.

The snippet below should explain this slightly more clearly:

```
var house = {color: "brown",
            size: "huge",
            expensive: true}

house.prototype // undefined
house.__proto__ // {constructor: ƒ, __defineGetter__: ƒ, __defineSet... (__pro

// Note: 'house' is an object that has a __proto__ object, but does not have a prototyp

function add (a,b) {
return a + b
}

add.prototype // {constructor: ƒ}
add.__proto__ // ƒ () { [native code] }

// Note: 'add' is a function that has a __proto__ and a prototype property. Try this co
```

__proto__ vs prototype: 1 hosted with ❤️ by GitHub view raw

As mentioned, prototypes are only present as properties on functions. In order to reference an object's properties into another, we can use the 'setPrototypeOf' method, which will allow one object to reference and have access to the properties of the other. However, that object will still not have a prototype property, but instead, a __proto__ object that will reference that property. Let's hope the snippet below helps make some sense of that:

```
var cat : {breed: "Russian Blue"} // cat.__proto__ lists all its methods and cat.protot
var dog: {age: 7} // same for the dog variable
// Now, let's see how we can get the cat object to access the properties of the dog obj

Object.setPrototypeOf(cat, dog) // allows the cat object to access the properties of th

// Even though we've used 'setPrototypeOf', the cat object still doesn't have a prototy

cat.hasOwnProperty('prototype') // returns false

cat.age // returns 7



```

__proto__ vs prototype: 2 hosted with ❤️ by GitHub view raw

So you'll see here that we're used setPrototypeOf, but when we check the cat object, we can only see the __proto__ which references the prototype within the constructor object of the dog object. It's definitely quite confusing at first, but again, just remember that even though we're using 'setPrototypeOf' in order to let one object access the properties of another, this does not create a prototype for the object. If you put the example above into your console and then check if the cat object has a prototype, you'll get 'false'.

Now let's quickly see an example using a constructor function, so we can really get the difference between how properties are accessed. To do that, we'll first define a constructor function and create an object using the 'new' keyword that will set the prototype of the constructor function. Then, we'll see the difference between accessing another object's property via the object itself and also, via the constructor property.

```
// Method 1: We'll define a constructor functoin and create an object using the new key
function Person (name, age) {
        this.name = name;
        this.age = age;
}

var person1 = new Person ("Mark", 25) // defined an object called person1 using the Per
var person2 = {nationality: "Australian"} // defines a simple object literal with a nat
// Now to let person1 access person2's nationality, we can use the same setPrototypeOf
Object.setPrototypeOf(person1, person2) // same as the cat and dog example above
person1.nationality // returns 'Australian' as expected

// Method 2: If we do the same thing but instead, we set the prototype of the construct
var person1 = new Person ("Mark", 25)
var person2 = {nationality: "Australian"}
Object.setPrototypeOf(Person, person2) // we've replaced person1 with Person

person1.nationality // returns 'undefined', because person1 can access person2's proper
person1.constructor.nationality // returns "Australian"
```

__proto__ vs prototype: 3 hosted with ❤ by GitHub view raw

Now I know this all may seem really confusing at first but try to reread it and most importantly, try it out yourself to see what's actually going on under the hood. In the above example, if you'll see it's not person one that has access to the prototype property of Person, but rather, it's constructor (function). What I mean is, when checking in the console, here's what you'll find:

- person1.hasOwnProperty('prototype') // returns false
- person1.constructor.hasOwnProperty('prototype') // returns true

Hopefully the difference between __proto__ and prototype are a bit clearer now and I've proven what I set out to, which is:

- The __proto__ is an object within **every** object that points out (references) the prototype that has been set for that object. `__proto__` is the actual object that is used in the lookup chain to resolve methods, etc.
- The prototype property is only present for functions and is a property that's set only if you're using the 'new' keyword when creating objects with this (constructor) function. As just mentioned, this means that it will only be available via the constructor of the object that's created using the 'new' keyword.

If it's too much for now, just remember that the only objects that have the prototype property at creation are functions. The __proto__ only references the prototype that has been set for an object and is used to build the prototype for when you create an object with the 'new' keyword.

I'm just starting out and will be posting an article every day so follow along if you enjoyed the read! :)

JavaScript Proto Prototype Constructor

Follow

Written by Venkat Iyengar

57 followers · 36 following