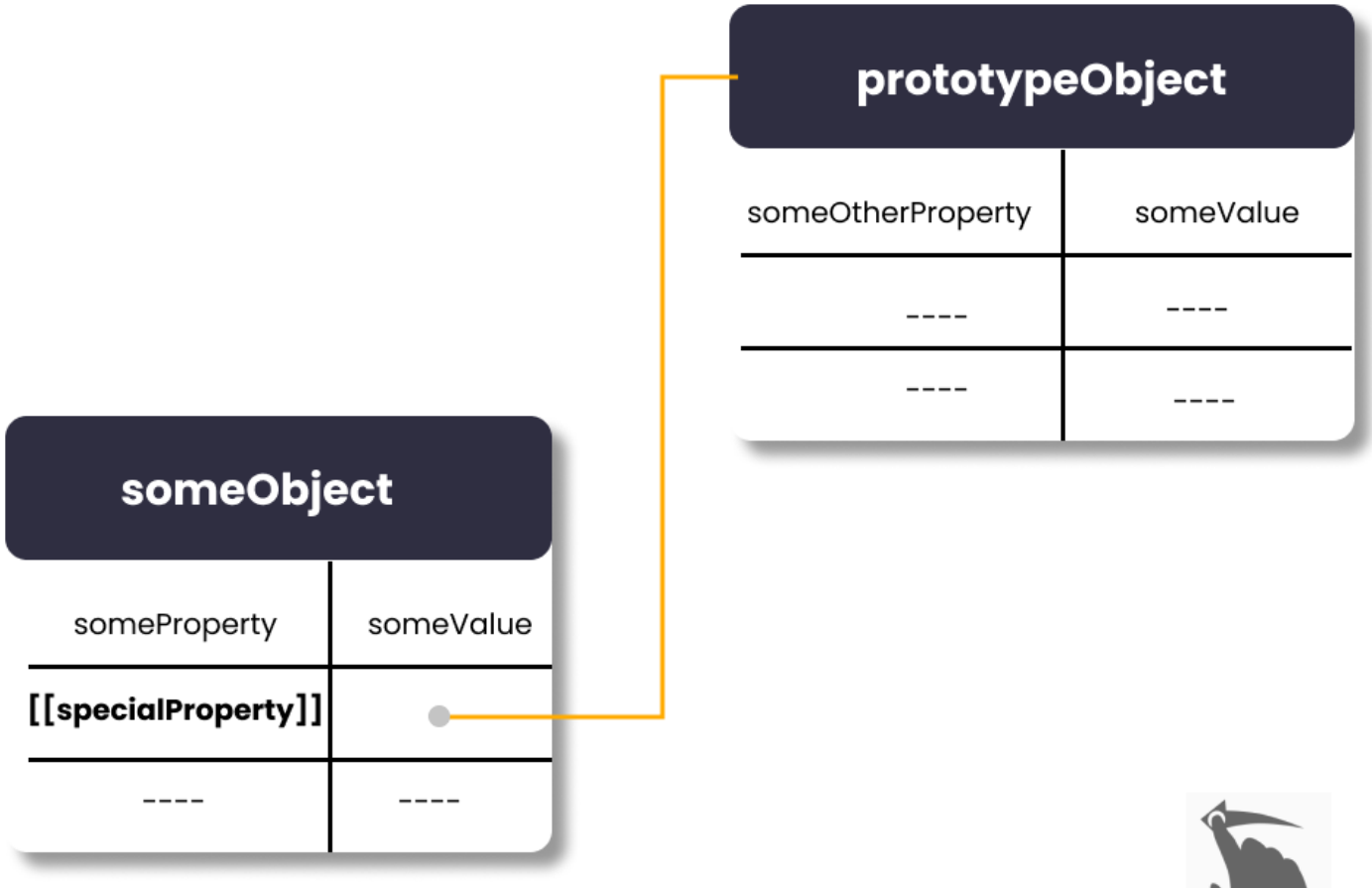
Prototypes in JS

antra verma

In this post, **we will dive into some of the concepts around prototypes in javascript** like: prototype, __proto__, prototype chain, etc.

"When you create some object in javaScript, **the javaScript engine attaches a special property** to it, **which points to another object** which has its own methods and properties, called as the **prototype** of that some object".



Let me show it through some code:

```
const someObject = {
  someProperty: 'some value',
  someMethod: function(){
    return 'some thing'
  }};
```

```
> console.log(someObject);
  ▼ {someProperty: 'some value', someMethod: ƒ}
    ▶ someMethod: ƒ ()
      someProperty: "some value"
    ▶ [[Prototype]]: Object
```

As you can see, we only added **someProperty** and **someMethod** in **'someObject'**, but we get to see an extra property **[[Prototype]]** there which is implicitly added by the javascript.

This special property is referring to the base Object which has its own methods and properties.

So, how can you access the prototype of someObject ?

```
//no ways
someObject.[[Prototype]]

//the old way
someObject.__proto__

//the new way
Object.getPrototypeOf(someObject)
```

Let's take a look at another example:

- let me create a simple plain object using object literal:

```
const user = {
  getRole: function (){
    return this.role;
  }
};
```

- Let's create another object using Object.create() :

```
const admin = Object.create(user);
admin.role = 'admin';

console.log(admin);
// {role: 'admin'}
```

So what's happenning here is that, we created a new object called admin with one property 'role' and set the internal property [[Prototype]] of admin to user object.

- now **admin** has access to all the properties and methods that **user** has.

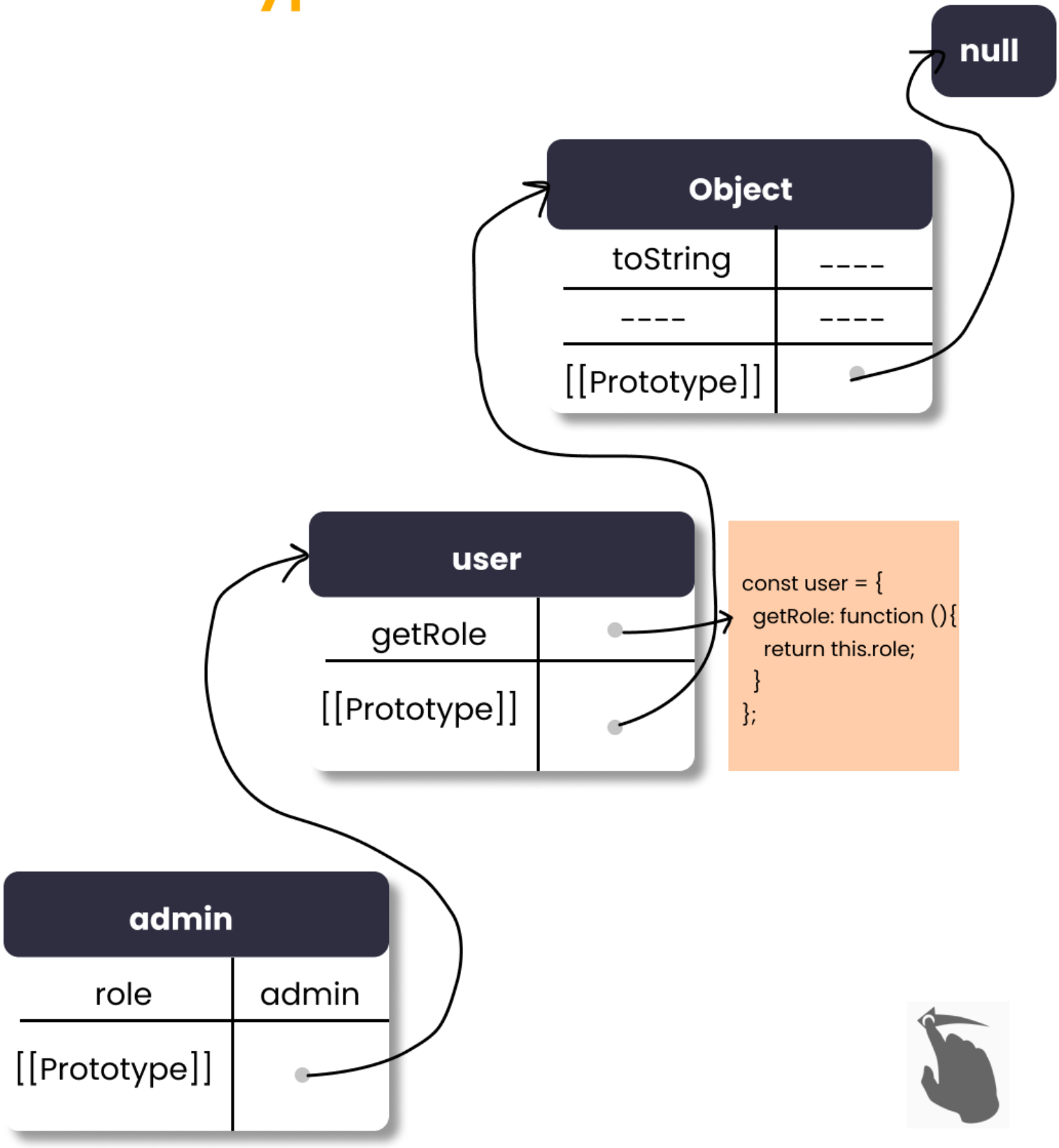
```
console.log(admin.getRole());
// admin
```

- In other words, when you try to access a property (or a method) in admin, javaScript will first look for that property in admin itself.

- If it does not find it there, it will move further and look for that in user object, if it is still not found there, it will be searched in the **prototype of user object** which **is the base Object**, then move to the **prototype of Object which is null**.

- By definition null has no prototype of its own, so the look up will end here. This is what **prototype chain** is!

Prototype Chain



[[Prototype]] vs __proto__ vs prototype

[[Prototype]]:

As we saw, in the example earlier, this is the internal hidden property that stores the reference to the prototype of an object.

__proto__:

__proto__ is used to access the prototype of an object.

prototype:

Yet another property created for an object, when you use the **new** keyword to create an instance of the object.

Was it helpful?