My Wild & Crazy Adventure with JavaScript Prototypes

Joel Tollefson
HTML 200A – Final Paper
March 16, 2018

Foreword

When it came to learning about JS prototypes, I though why not just jump into the deep end. What's the worst that could happen?

It happened… I flailed about and tried to keep my head above water while I treaded over constructors, objects, inheritance, important prototype syntax that I typed incorrectly so it didn't work (like you need not one but two "_" before and after the "prototype" statement), and then different way to call prototypes depending on your browser…

So I managed to get out, and decided to start all over. This time, learning the basic dog stroke before going in deeper. This is partly why I start all the way back to the basics of – what is an object. So here's my journey.

BASIC of JS Prototypes

Because prototypes work with objects, let's start at the very beginning… [2]

What is an object?

A formal definition is: An object in JavaScript is any unordered collection of key-value pairs. So anything that is NOT a primitive – number, string, boolean, etc. – is an object

An example is creating an object that defines the properties of a car – it's simple:

```
var car = {
        make: "Subaru",
        model: "Outback",
        year: "2016",
        color: "white"
};
```

What is a JS prototype?

A prototype is an object from which other objects inherit properties. I think of a prototype as a physical prototype model with definitions that other objects can inherit. It's really all about inheriting properties.

Which objects have prototypes?

Any object can be a prototype. Since prototypes are objects, every prototype can have a prototype – kind of like a daisy chain of prototypes.

Then there's the new concept of a constructor – what is a constructor?

Okay – you need to learn so much just to understand prototypes. A "constructor" – also called an "object constructor" is used to create a large number of similar objects – so you don't have to repeat the creation of the object. You define the constructor once – then you use it every type you want to create a new object. This is similar to a function. So here's an example – following the example of "car" above:

```
function Auto(make, model, year, color) {
        this.make= make;
        this.model= model;
        this.year= year;
        this.color= color;
}
```

So we named this Auto – and the parameters of the function (make, model,…) match the properties we want to supply of EACH car. Note that this constructor doesn't return anything – like a function would. We've assinged each parameter (make, model,…) to an object , such as: this.make= make

Now to use this, all I have to do is call it; for example:

```
var autoJoel = new Auto("Subaru", "Outback", 2016, "white");
var autoMary= new Auto("Honda", "Civic", 2009, "silver");
var autoJohn= new Auto("Tesla", "S9", 2018, "red");
var allCars= [autoJoel, autoMary];
```

Then I can use this in code, a snippet of which might be something like:

```
If (allCars[0].year < 2015 ) {
        console.log("The car is " +  allCars[0].year + "years old.  This is an old car")
} else {
        console.log("The car is " +  allCars[0].year + "years old.  This is a newer car")
}
```

How are prototypes defined?

This is where things get confusing… There are really 3 different methods of defining an object prototype – depending on the browser:

Let's assume you have defined an object named "a":

```
var a= {} ;
```

Method 1: use getPrototypeOf (supported on: Firefox, Safari, Chrome and IE9)

```
Object.getPrototypeOf(a);
```

Method 2: use __proto__ (supported on all browsers except IE)
(NOTE: *this uses 2 "_" in front of "proto", and 2 "_" after proto; I didn't know this and tried using it and it never worked until I discovered this syntax*)

```
a.__proto__;
```

Method 3: use "constructor" (supported on all browsers)
(NOTE: *this uses 2 "_" in front of "proto", and 2 "_" after proto*)

```
a.constructor.prototype;
```

So – what's the power of prototypes?

The best thing about prototypes, is that many different objects can inherit the properties from a single prototype. So the prototype just defines the properties one time – but it can get inherited many times by all objects that refer to it. So – define ONCE – use MANY times.

Obviously, this makes it easier to create the code and maintain it. Apparently, it also then makes for better efficient running code – because less memory is needed. Instead of defining hundreds of single objects requiring a lot more memory, you can just reference the prototype.

The *Head First JS Programming* guide [5] spend a lot time talking about how prototypes are called. I guess this is important – because every object in JS has a prototype. So, when JS gets an object, it first attempts to find a property in that object, and then, if not found, it sends it to the object's prototype to get the inheritance. And this inheritance can be "chained" as long as you want.

Can you show me a simple example of a prototype?

Here's an example of how prototyping can be used to have an object inherit the properties of another object. [1]

```
// first let's create an object named "alien"
var alien = {
    kind: 'Martian'
}
// 2nd create an object named person
var person = {
    kind: 'human'
}
// 3rd add an object named joel
var joel={}

// use _proto_ to assign alien as a prototype of joel => joel
//  now inherits the properties of alien
joel.__proto__ = alien
```

```
// test it – display value
console.log("Joel is a: " joel.kind)  //  output: Martian

// assign person as prototype of joel
joel.__proto__= person

// test it – display the value
console.log("Joel is a: " + joel.kind ) //  output: human
```

In the above example, joel can inherit the property of "alien" , or, the property of "person". If alien is inherited, joel=martian; if person is inherited, joel=human. So this just shows a simple example of how inheritance works, and how prototypes can be used.

I could do all of this with just using functions and objects – so why use prototypes?

Because prototypes inherit and re-use existing properties including methods. For example, in the previous example about cars, here's the code again.

```
// start with the Auto constructor

function Auto(make, model, year, color) {
      this.make= make;
      this.model= model;
      this.year= year;
      this.color= color;
}

      var autoJoel = new Auto("Subaru", "Outback", 2016, "white");
      var autoMary= new Auto("Honda", "Civic", 2009, "silver");
               var autoJohn= new Auto("Tesla", "S9", 2018, "red");
      var allCars= [autoJoel, autoMary];


      If (allCars[0].year < 2015 ) {
            console.log("The car is " +  allCars[0].year + "years old.  This
      is an old car")
      } else {
            console.log("The car is " +  allCars[0].year + "years old.  This
      is a newer car")
      }
```

In the above code, three auto objects are created. Now imagine we had a database of 50,000 car owners with information about their car; each would be an object that defines the properties of the car **So, there would be 50,000 Auto constructors**. This takes a lot of memory.

Example of JS Prototype Code

So – finally, here's a complete example of using constructors and prototypes. I nailed it!

The code is also included in the GitHub site.

```
// JS file: prototype_auto.js

// start with the Auto constructor

function Auto(make, model, year, color, price) {

    this.make= make;

    this.model= model;

    this.year= year;

    this.color= color;

    this.price= price;

}


// let's start by adding a property named category to the Auto prototype

    Auto.prototype.category = "car";


// define a property named age and method to the Auto prototpye

    Auto.prototype.age = function() {

      if (this.year < 2015) {

          console.log("The " + this.make + " " + this.model + " year is " +
this.year + ".  This is an older car");

      } else {

          console.log("The " + this.make + " " +  this.model + " year is " +
this.year + ".  This is a newer car");

      }

};


// define a property named expensive and method to the Auto prototpye

    Auto.prototype.expensive = function() {

      if (this.price < 50000) {
```

```
          console.log("The price of the " + this.make + " " +  this.model + "
is $" +  this.price + ". This is inexpensive");
      } else {
          console.log("The price of the " + this.make + " " +  this.model + "
is $" +  this.price + ". This is EXPENSIVE!");
      }
};


    Auto.prototype.exterior= function(){
      console.log("The color is: " + this.color);
    };


// create the data base of cars
      var autoJoel = new Auto("Subaru", "Outback", 2016, "white", 32000);
      var autoMary = new Auto("Honda", "Civic", 2009, "silver", 20000);
      var autoJohn = new Auto("Tesla", "S9", 2018, "red", 60000);



// now call the methods for each car;  each car inherits the methods from the
Auto prototype
    autoJoel.age();
    autoJoel.expensive();
    autoJoel.exterior();


    autoMary.age();
    autoMary.expensive();
    autoMary.exterior();


    autoJohn.age();
    autoJohn.expensive();
    autoJohn.exterior();
```

Here's the output code copied from the console when running the code:

prototype_auto.js:18 The Subaru Outback year is 2016. This is a newer car

prototype_auto.js:25 The price of the Subaru Outback is $32000. This is inexpensive

prototype_auto.js:32 The color is: white

prototype_auto.js:16 The Honda Civic year is 2009. This is an older car

prototype_auto.js:25 The price of the Honda Civic is $20000. This is inexpensive

prototype_auto.js:32 The color is: silver

prototype_auto.js:18 The Tesla S9 year is 2018. This is a newer car

prototype_auto.js:27 The price of the Tesla S9 is $60000. This is EXPENSIVE!

prototype_auto.js:32 The color is: red

In Conclusion

This was a fantastic and fun project. I definitely learned about prototypes and constructors. I even managed to write some code and make it work.

Note that the index.html file and 2 JS script files (prototype.js and prototype_auto.js) are included in the GitHub site.

References

(1) “Plain English Guide to JavaScript Prototypes “; http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/

(2) “Understanding JavaScript Prototyypes”; https://javascriptweblog.wordpress.com/2010/06/07/understanding-javascript-prototypes/

(3) “JavaScript Prototype in Plain Language”; http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/

(4) w3schools.com, “JavaScirpt Object Prototypes”; https://www.w3schools.com/js/js_object_prototypes.asp

(5) Head First JavaScript Programming, by Eric Freemand & Elisabeth Robson .