Prototypes

Contents

Overview

The built-in prototype object holds all properties (including methods) that will be inherited by the instance.
In essence, this defines the structure of the object and is comparable to a class.
All new instances of the object will inherit the members of the prototype.
References to data structures - such as objects/arrays - that are added to the prototype are shared between instances.
Accessing these shared properties is slightly slower as the instance is searched before the prototype, but it does mean that we save memory allocation.
The general advice is to declare members in the constructor (private fields) and to declare methods (functions) within the prototype. This approach gives the separation between data and behaviour that we expect.

Note: This covers traditional JavaScript; when programming in JavaScript we encourage you to use the more modern way of doing things, which provides easier syntax to achieve the same things!

Tutorial

Explaining Prototypes

Prototypes are the mechanism by which Javascript objects inherit features from one another. Objects can have a `prototype` **object**, which acts as a template object that it inherits methods and properties from.
An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is referred to as a **prototype chain**, and explains why different objects have properties and methods defined on other objects available to them.
In JavaScript, a link is made between the object instance and its prototype (its `__proto__` property) which is derived from the `prototype` property on the constructor) and the properties and methods are found by walking up the chair of prototypes.

Let's create a simple function and start playing around with it:

```
function Person(first, middle, last, age) {
  this.name = {
    first: first,
    middle: middle,
    last: last,
  };
  this.age = age;
}
```

Let's try add a property to the existing constructor.

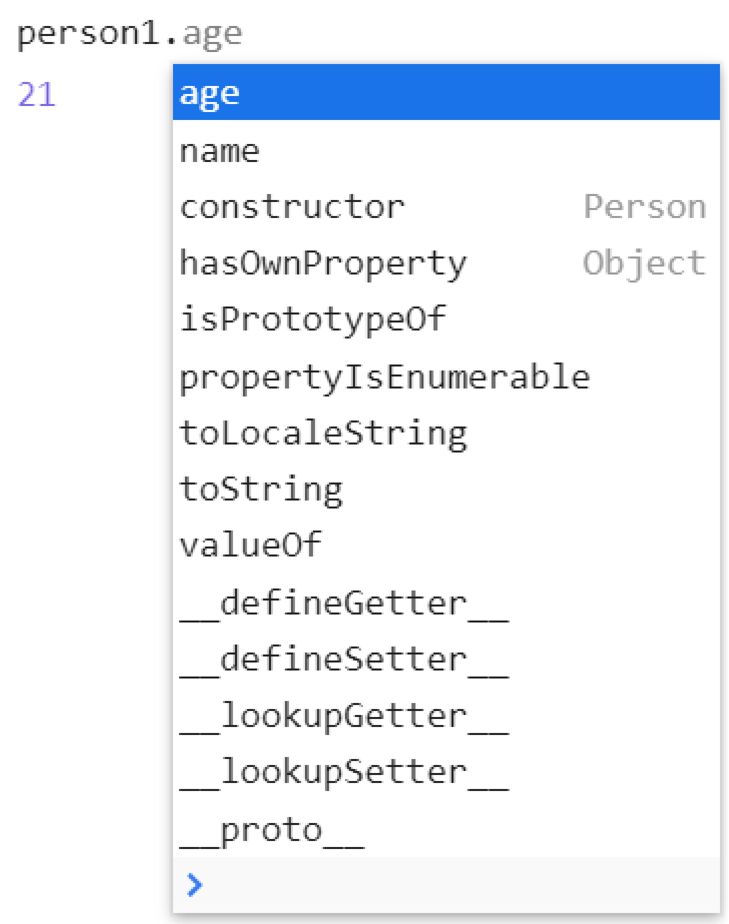
```
Person.nationality = "English";
let person1 = new Person("Chris", "P.", "Bacon", 21);
```

Now lets try to access the Nationality property from the instance we created.

```
console.log(person1.nationality); // undefined
```

This will not work, it returns undefined because we cannot add a new property to an existing object constructor.

Let's see what properties we have available to us:
In a browser console write the above code, then in a new line type `person1.` then assess the output.
You should see something similar to the below picture:



```
person1.age
21
age
name
constructor          Person
hasOwnProperty       Object
isPrototypeOf
propertyIsEnumerable
toLocaleString
toString
valueOf
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
__proto__
```

In this list, you should see all of the members defined in `person1`'s constructor - `Person()`,`name` and `age`; however, you will also see some other members - `toString`, `valueOf`, `.__proto__`, and such. Odd right? This is because these are defined on `Person()`'s constructor's prototype object, which is `Object` - now stay with me, this is because - **everything in javascript is an object** therefore we 'inherit' those methods from the Object class. We can then access the `.__proto__` (that is double underscore before and after the word proto) object and add our own properties to it

Going back to the original question, how do I add properties to an existing object constructor? Well, in order to do that we must go about it in a different method...

The JavaScript `prototype` property allows you to add new properties to object constructors, let's use the example we were working on before:

```
function Person(first, middle, last, age) {
  this.name = {
    first: first,
    middle: middle,
    last: last,
  };
  this.age = age;
}
Person.prototype.nationality = "English";
```

Now if I declared a new instance of the object:

```
let human = new Person("Chris", "P.", "Bacon", 22);
console.log(human.nationality); // English
```

Note:
If you are going to modify the prototype, only modify your own. Never modify the prototypes of standard JavaScript Objects

I can also use the prototype property to add new methods to object constructors:

```
function Student(firstName, lastName, DOB) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.DOB = DOB;
}
Student.prototype.details = function () {
  return (
    "The student's name is " +
    this.firstName +
    " " +
    this.lastName +
    " their DOB is " +
    this.DOB
  );
};

let pupil1 = new Student("Donald", "Duck", "19/05/1976");
console.log(pupil1.details()); // The student's name is Donald Duck their DOB is
19/05/1976
```

Summary:

- All JavaScript object inherit properties and methods from a prototype.
- We cannot add a new property to an existing object constructor so we use the prototype object.
- You can inspect the object by using the `.__proto__` notation in a browser console.
- This is the traditional way of manipulating an object, you must be familiar with the modern way of writing the above code.
- Never modify a JavaScript standard prototype object, always modify your own.

Exercises

1. Create a function called Dog which has a few data members.
2. Create a 'behaviour' for the Dog using prototype
3. Create an instance of the object and call the prototype function.
4. Study how the `.__proto__` object works in a browser console.

▶ Solution