JavaScript Constructor/Prototype Pattern

Summary: in this tutorial, you'll learn how to use the JavaScript constructor/Prototype pattern to define a custom type in ES5.

Introduction to the JavaScript Constructor / Prototype pattern

The combination of the constructor and prototype patterns is the most common way to define custom types in ES5. In this pattern:

- The constructor pattern defines the object properties.
- The prototype pattern defines the object methods.

By using this pattern, all objects of the custom type share the methods defined in the prototype. Also, each object has its own properties.

This constructor/prototype pattern takes the best parts of both constructor and prototype patterns.

JavaScript Constructor / Prototype example

Suppose that you want to define a custom type called `Person` that has:

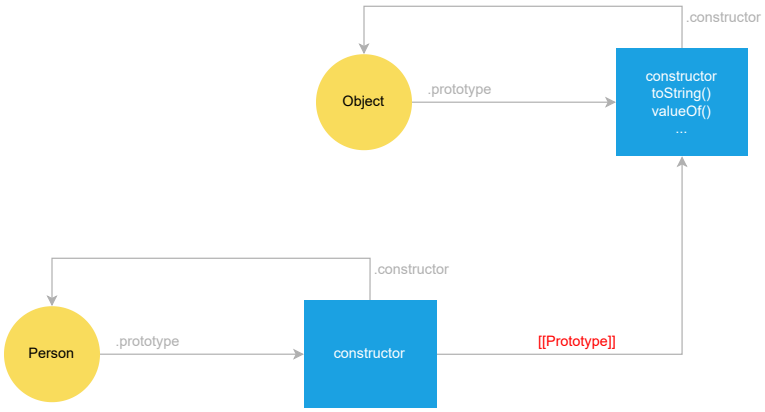
- Two properties `firstName` and `lastName` .
- One method `getFullName()` .

First, use the constructor function to initialize the properties:

```
function Person(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
}
```

Behind the scenes, the JavaScript engine defines a `Person` function denoted by the circle and an anonymous object denoted by the square.

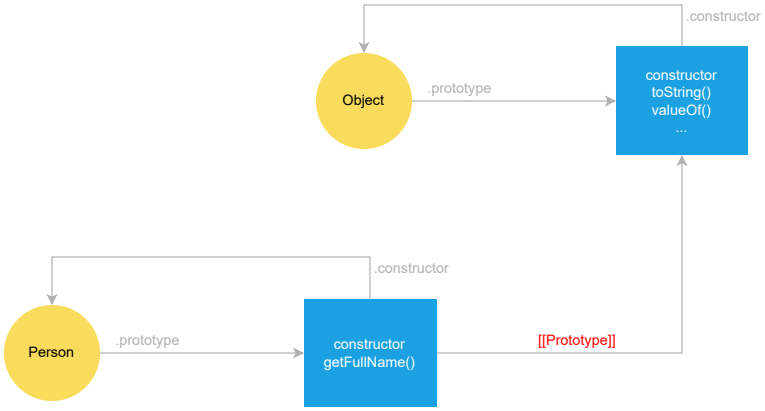
The `Person` function has the `prototype` property that references an anonymous object. The anonymous object has a `constructor` property that references the `Person` function:



Second, define the `getFullName()` method in the `prototype` object of the `Person` function:

```
Person.prototype.getFullName = function () {
  return this.firstName + ' ' + this.lastName;
};
```

JavaScript defines the `getFullName()` method on the `Person.prototype` object like this:



Third, create multiple instances of the `Person` type:

```
let p1 = new Person("John", "Doe");
let p2 = new Person("Jane", "Doe");

console.log(p1.getFullName());
console.log(p2.getFullName());
```

Output:

```
'John Doe'
'Jane Doe'
```

Javascript creates two objects `p1` and `p2`. These objects link to the `Person.prototype` object via the `[[Prototype]]` linkage:

Each object has its own properties `firstName` and `lastName`. However, they share the same `getFullName()` method.

When you call the `getFullName()` method on the `p1` or `p2` object, the JavaScript engine searches for the method on these objects. Because the JavaScript engine doesn't find the method there, it follows the prototype linkage and searches for the method in the `Person.prototype` object.

Because the `Person.prototype` object has the `getFullName()` method, JavaScript stops searching and executes the method.

Put it all together:

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

Person.prototype.getFullName = function () {
  return this.firstName + ' ' + this.lastName;
};

let p1 = new Person('John', 'Doe');
let p2 = new Person('Jane', 'Doe');

console.log(p1.getFullName());
console.log(p2.getFullName());
```

Classes in ES6

ES6 introduces the class keyword that makes the constructor/prototype pattern easier to use. For example, the following uses the `class` keyword to define the same `Person` type:

```
class Person {
        constructor(firstName, lastName) {
```

```
                this.firstName = firstName;
                this.lastName = lastName;
        }
        getFullName() {
                return this.firstName + " " + this.lastName;
        }
}

let p1 = new Person('John', 'Doe');
let p2 = new Person('Jane', 'Doe');

console.log(p1.getFullName());
console.log(p2.getFullName());
```

In this syntax, the `class` moves the property initialization to the `constructor` method. It also packs the `getFullName()` method in the same place as the `constructor` function.

The class syntax looks cleaner and less verbose. However, it's syntactic sugar over the constructor/prototype pattern with some enhancements.

For more information on the classes, check out the JavaScript class tutorial.

Summary

- Use JavaScript constructor/prototype to define a custom type in ES5.
- Initialize the object properties in the constructor function and define methods and properties that can be shared by all instances in the prototype object.