Get started Open in app

BetterProgramming

Follow 200K Followers

This is your **last** free member-only story this month. Sign up for Medium and get an extra one

JavaScript Best Practices — Classes

Cleaning up our JavaScript code is easy with default parameters and property shorthands

John Au-Yeung Apr 17, 2020 · 2 min read ★

Photo by Kimberly Farmer on Unsplash.

Cleaning up our JavaScript code is easy with default parameters and property shorthands.

In this article, we'll look at the best practices for creating classes.

ES6 Class Basics

To make the prototypical inheritance model of JavaScript less confusing, the class syntax has been introduced in ES6.

It has the `constructor` , instance variables, and methods just like classes in languages like Java do, but they act differently.

For instance, we can create a class by writing:

We have the `name` instance variable and the `speak` method. `this` is the instance of the class itself.

Getters and Setters

Getters are methods that only return values, whereas setters are methods that let us set a value to an instance variable.

They let us do either in a controlled manner rather than accessing instance variables directly.

We can also add getters and setters to the class as follows:

With these getters and setters, we can write the following to get `name` :

```
const person = new Person('joe');
console.log(person.name);
```

And set `name`:

```
const person = new Person('joe');
person.name = 'jane';
console.log(person.name);
```

Child or Subclasses

We can also create child classes with the `extends` keyword. For instance, we can write:

We created a `Dog` class that overrides the `speak` method of the `Animal` class.

Also, we call the parent constructor with `super` in the `Dog`'s `constructor` and call `super.speak` to call the `speak` method of `Animal`.

This is much cleaner than the old constructor function syntax for inheritance.

Static Methods and Properties

We can add static properties to a class by using the `static` keyword.

Doing that will help us add a property to it that's shared by all instances.

For instance, we can write:

```
class Dog {
  static type() {
    return 'dog';
  }
}
```

Then we can call it as a static method as follows:

```
console.log(Dog.type())
```

To create static properties, we write the following:

```
class Dog {}
Dog.type = 'dog';

console.log(Dog.type);
```

We set the `type` property to `'dog'` and it's shared across all instances.

This syntax is valid since classes are just functions and functions are treated the same as any other object in JavaScript.

Private Fields

Future versions of JavaScript can also let us define private class fields. There's no equivalent of this in constructor functions.

They are denoted by the `#` symbol in front of the field name.

Conclusion

We can define constructors with the class syntax to make them look easier to understand than constructor functions.

The classes look similar to Java classes but act very differently underneath. We have to understand the underlying behavior before we use it. This means that we should understand prototypical inheritance before using JavaScript classes.

Sign up for programming bytes

By Better Programming

A monthly newsletter covering the best programming articles published across Medium. Code tutorials, advice, career opportunities, and more! Take a look.

Get this newsletter

Programming JavaScript Nodej Angular React

About Write Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play