NPM: node package manager. Useful for handles package dependencies.

Package.json: manage dependencies and its version

Js has no classes.

Where in Java our myCar, asked to honk, says “go look at this class over there, which is my blueprint, to find the code you need”, JavaScript says “go look at that other object over there, which is my prototype, it has the code you are looking for”. Building objects via an object-object relationship is called Prototype-based programming, versus Class-based programming used in more traditional languages like Java.

In java : Car myCar = new Car();

In js:

var makeCar = function() {

var newCar = {}

newCar.honk = function() {

console.log('honk honk');

};

return newCar;

};

myCar1 = makeCar();

myCar2 = makeCar();

myCar3 = makeCar();

(1) 1000 mycar() will create 1000 objects allocating memory for 1000 functions.

1. they don’t share anything
2. Property:Whenever we call a function on an object, the JavaScript interpreter tries to find that function within the queried object. But if it doesn’t find the function within the object itself, it asks the object for the pointer to its prototype, then goes to the prototype, and asks for the function there. If it is found, it is then executed.

Var car = function() {};

Car.prototype.honk = function() {

Console.log(“honk honk”);

};

Var myCar1 = new Car();

Var myCar2 = new Car();

myCar1.honk(); // executes Car.prototype.honk() and outputs “honk honk”

myCar2.honk(); // executes Car.prototype.honk() and outputs “honk honk”

1. we can change Car.prototype in run time

var Car = function() {};

Car.prototype.honk = function() {

console.log('honk honk');

};

var myCar1 = new Car();

var myCar2 = new Car();

myCar1.honk();

myCar2.honk();

// executes Car.prototype.honk() and outputs "honk honk"

// executes Car.prototype.honk() and outputs "honk honk"

Car.prototype.honk = function() {

console.log('meep meep');

};

myCar1.honk();

myCar2.honk();

1. we can also treat only one of our cars differently at runtime:

var Car = function() {};

Car.prototype.honk = function() {

console.log('honk honk');

};

var myCar1 = new Car();

var myCar2 = new Car();

myCar1.honk();

myCar2.honk();

// executes Car.prototype.honk() and outputs "honk honk"

// executes Car.prototype.honk() and outputs "honk honk"

myCar2.honk = function() {

console.log('meep meep');

}; // add additional behavior for myCar2

myCar1.honk();

myCar2.honk();

// executes Car.prototype.honk() and outputs "honk honk"

// executes myCar2.honk() and outputs "meep meep"

1. different in java like language and js

That’s one of the major differences to class-based programming: while objects are relatively “rigid” e.g. in Java, where the structure of an object cannot be changed at runtime, in JavaScript, the prototype-based approach links objects of a certain class more loosely together, which allows to change the structure of objects at any time.

Comparsion:

var C = function() {

this.f = function(foo) {

console.log(foo);

};

};

var a = [];

for (var i = 0; i < 1000000; i++) {

a.push(new C());

}

Creates 1000000 objects that all have the function directly attahed to them.

var C = function() {};

C.prototype.f = function(foo) {

console.log(foo);

};

var a = [];

for (var i = 0; i < 1000000; i++) {

a.push(new C());

}

1. Inheritance(inheritance runs through a chain of prototypes)

var Vehicle = function() {};

Vehicle.prototype.drive = function() {

console.log('vrooom...');

};

var Car = function() {};

Car.prototype = new Vehicle();

Car.prototype.honk = function() {

console.log('honk honk');

};

var myCar = new Car();

myCar.honk();

myCar.drive();

// outputs "honk honk"

// outputs "vrooom..."

(8) There is only one single piece of JavaScript code that can be executed within

the event loop at any given time.

Read file system is expensive compare with others. So all IO operations in node is asynchronously.

1. Two principles for writing responsive Nodejs applications:

Handle IO-intensive operations through asynchronous operations

Keep your own code (that is, everything that happens synchronously within event loop iterations) as lean as possible