Классы

*Классы* – это новая возможность для создания однотипных объектов удобно. Он имеет такой синтаксис:

class Class {

constructor () {

console.log (arguments)

}

method1 () {

}

method2 () {

}

}

let obj = new Class(12, 120);

Первым объявляют метод constructor(), который неявно вызывается при создании. Классы – это синтаксический сахар, но с некоторыми отличиями:

* Неявно поставлен enumerable: false поэтому нельзя посмотреть через for … in
* Не может без new
* Классы всегда используют use strict

Также есть Class Expression по аналогии с Function Expression и Class Named Expression соотвественно. Их используют редко, к примеру, создать класс на лету:

function makeClass (search) {

return class {

requestMethod () {

console.log(search);

}

}

}

Классы также унаследовали get/set:

class Person {

constructor(fullName) {

// Вызывается set

this.fullName = fullName;

}

get fullName() {

return this.\_fullName;

}

set fullName(value) {

this.\_fullName = value;

}

}

Свойства можно создать прямо в классе, если только их инициализировать. Все это создается вместе с new и не попадает в прототип.

class Person {

isAdmin = false;

constructor(name = 'Alex') {

this.name = name;

}

toString() {

return `${this.name}: ${this.isAdmin}`;

}

}

Доступ к свойствам можно получить только через this.

Статические свойства

Статические свойства ввели относительно недавно. Объявляются с помощью static.

Для того чтоб обратиться к статическому свойству внутри другого статического используется this. А обратиться внутри не статического свойства можно только через сам класс или this.constructor:

class Programmer {

static \_counter = 0;

constructor(name, type, stack) {

this.name = name;

this.type = type;

this.stack = stack;

this.constructor.\_counter++;

}

static createJSDEv(name) {

return new Programmer(name, 'js', {

platform: 'node.js',

frameworks: [

'react native',

'electrone.js'

]

})

}

static getCount() {

return this.\_counter;

}

}

let dev = Programmer.createJSDEv('Alex');

let dev1 = Programmer.createJSDEv('Michael');

let dev2 = Programmer.createJSDEv('Ivan');

console.log (Programmer.getCount ());

Здесь в переменной \_count введется счетчик, того сколько раз мы вызвали constructor, т.е. создали объектов. Для безопасности мы ее скрываем и возвращаем только в методе getCount.

Метод createJSDEv представляет собой утилиту, в данном случае создать объект по какому-либо шаблону.

Технически статические свойства – это свойство самой функции и сравни тем, что присваивание самой функции.

Модификатор доступа

Private свойства, которые можно объявить с помощью #. Вызов вне класса приведет к ошибке, но использовать private рискованно, т.к. нормально реализован, только в последних версиях Chrome.

Есть соглашение о том, что для реализации protected используется \_.

Наследование

Самое неудобное в прототипах – это наследование. В классах наследование упрощено и более гибкое, но по-прежнему в основу легли прототипы. Рассмотрим основные моменты наследования на примере:

class Animal {

constructor(mass, speed, periodSleep, beetwenEat) {

this.mass = mass;

this.speed = speed;

this.periodSleep = periodSleep;

this.beetwenEat = beetwenEat;

}

eat() {

console.log ('eat');

setTimeout(

() => console.log ('shit'),

this.beetwenEat

);

}

sleep() {

console.log ('sleep')

setTimeout(() => console.log('wake up'), this.periodSleep);

}

}

class Cat extends Animal{

constructor(mass, name, color) {

super(mass, 55, 8000, 1000);

this.name = name;

this.color = color;

}

mur () {

console.log ('Mur');

}

sleep() {

console.log('В клубочек');

super.sleep();

}

}

let pet = new Cat(4, 'Button', ['#f2f2f2', '#333']);

pet.sleep();

Первое наследование осуществляется при помощи ключевого слова extends. Класс Cat должен из конструктора *инициализировать* конструктор Animal через метод super. Если бы у Cat не было бы конструктора, то он унаследовал бы его из Animal.

Можно *переопределить* метод класса. Внутри переопределенного метода часто вызывают такой же метод из родительского класса или другой любой. В таком случаи можно воспользоваться *super* – this к классу родителю, т.е. прототипу.

В наследовании помогает оператор instanceof, который позволяет узнать принадлежит объект классу или принадлежит к классу потомка.

let pet = new Cat(4, 'Button', ['#f2f2f2', '#333']);

let a = new Animal(50, 20, 30000, 10000);

console.log (pet instanceof Cat);

console.log (pet instanceof Animal);

console.log (a instanceof Cat);