Методы

В JS объекты могут иметь *метод* – это функция, которая принадлежит только объекту.

let petro = {

name: 'Petro',

old: 29,

countProg: 0,

work() {

this.countProg++;

console.log ('программа: ' + this.countProg);

}

}

petro.work ();

petro.work ();

petro.work ();

Здесь создается метод work, который выполняет работу. Внутри метода используется ключевое слово this, которое используется когда нужно взаимодействовать с другими переменными объектами. Так что this – это просто ссылка на текущий объект.

Вместо this можно было просто использовать прямую ссылку на объект, как показано ниже:

let petro = {

name: 'Petro',

old: 29,

countProg: 0,

work() {

petro.countProg++;

console.log ('программа: ' + petro.countProg);

}

}

Но этот способ неправильный, ведь этот метод можно переназначить другому объекту и таким способом все сломается. Также this неявляется *фиксированным*, т.е. может меняться, к примеру, в Java this в классах может меняться.

let petro = {

name: 'Petro',

old: 29,

countProg: 0,

}

let alex = {

name: 'alex',

old: 40,

countProg: 0,

}

let work = function() {

this.countProg++;

console.log (this.name + ' программа: ' + this.countProg);

}

petro.work = work;

alex.work = work;

petro.work ();

alex.work ();

petro.work ();

alex.work ();

alex.work ();

В функциях также есть this, который равен window, но в строгом режиме он будет равен undefined, т.е. this нет у функций. Также в стрелочных функциях нет this.

Флаги

До этого объект – эта пара ключ и значение (value). Но можно дополнительно поставить *флаг* – модификатор для этого свойство:

* writable – если false, то значение нельзя изменять и удалить
* enumerable – если false, то свойство будет пропускаться в циклах и его нельзя получить через []
* configurable – если false, то изменение флагов после будет невозможно.

По-умолчанию их значение всегда true. Их нельзя изменить прямо, ведь они записаны *дескриптор*, но можно изменить его методом defineProperty():

defineProperty (*объект*, *имя свойства*, *объект со свойствами*)

Пример:

let john = {

old: 12,

name: "John"

};

Object.defineProperty(john, "name", {

writable: false,

enumerable: false

});

for (let key in john) {

console.log(key + ': ' + john[key]);

}

john.name = 'delete';

console.log (john.name);

Примером configurable: false является Math.PI, у которого заранее есть writable: false.

Геттеры / Сеттеры

Геттеры и сеттеры встроены в js:

let user = {

\_name: 'alex',

\_family: 'klimentov',

get fullName () {

return this.\_name + ' ' + this.\_family;

},

set fullName (value) {

[this.\_name, this.\_family] = value.split (' ');

}

}

user.fullName = 'Alex Kotov';

console.log (user.fullName);

Здесь используется соглашение, насчет \_, которое указывает на то, что переменную нельзя использовать вне этого объекта.

let user = {

\_old: 1,

set old(value) {

if (value > 0 && value < 130) {

alert('Невалидное значение');

return;

}

this.\_old = value;

},

get old () {

return this.\_old;

}

}