

Конструкторы

Гоголев Сергей

Много объектов одного типа

```
var billy = {  
  name: 'Billy',  
  sleep: function () {}  
};
```

```
var willy = {  
  name: 'Willy',  
  sleep: function () {}  
};
```

Проблема: дублирование кода
при создании объекта

Решение: использовать
конструктор объектов

Самодельный конструктор

```
function createStudent(name) {  
    return {  
        name: name,  
        sleep: function () {  
            console.info('zzzZZ ...');  
        }  
    };  
}
```

```
var billy = createStudent('Billy');
```

```
var willy = createStudent('Willy');
```

Самоделный конструктор

Проблема: каждый раз создаём метод `sleep()`

Решение: вынести этот метод в прототип

Самодельный конструктор

```
var studentProto = {  
  sleep: function () {  
    console.info('zzzzZZ ...');  
  }  
};
```

```
function createStudent(name) {  
  var student = {  
    name: name  
  };  
  
  Object.setPrototypeOf(student, studentProto);  
  
  return student;  
}
```

Самоделный конструктор

```
var billy = createStudent('Billy');  
var willy = createStudent('Willy');
```

```
billy.sleep();  
// zzzZZ ...
```

```
willy.sleep();  
// zzzZZ ...
```

Конструктор «из коробки»

Любая функция, вызванная оператором `new`

```
var billy = new createStudent('Billy');
```

```
function createStudent(name) {  
    this.name = name;  
}
```

```
function createStudent(name) {  
    // var this = {};  
    this.name = name;  
    // return this;  
}
```

`this` указывает на создаваемый объект

Конструктор «из коробки»

```
function createStudent(name) {  
    this.name = name;  
}  
  
var billy = new createStudent('Billy');
```

Чуть больше семантики

```
function student(name) {  
    this.name = name;  
}  
  
var billy = new student('Billy');
```


Правило именования конструкторов

Чтобы отличить функцию-конструктор от обычной, их именуют с заглавной буквы.

```
function Student(name) {  
    this.name = name;  
  
}  
  
var billy = new Student('Billy');
```

Зачем отличать конструкторы от обычных?

```
function Student(name) {  
    this.name = name;  
}
```

```
var billy = Student('Billy');
```

Поле появится в глобальном объекте!

```
window.name === 'Billy'; // true
```

use strict;

TypeError: Cannot set property 'name' of
undefined

Возвращаем значение из конструктора

```
function Student(name) {  
    this.name = name;  
  
    return {  
        name: 'Muahahahahaha!'  
    };  
}
```

```
var billy = new Student('Billy');  
  
console.info(billy.name);  
// Muahahahahaha
```

Возвращаем значение из конструктора

```
function Student(name) {  
    this.name = name;  
  
    return null; // Evil mode on!  
}
```

```
var billy = new Student('Billy');  
  
console.info(billy.name);
```

```
// Billy
```

Автоматическая привязка прототипа

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype = {  
    sleep: function () {}  
};
```

```
function Student(name) {  
    // var this = {};  
    this.name = name;  
    // Object.setPrototypeOf(this, Student.prototype);  
    // return this;  
}
```

Автоматическая привязка прототипа

```
function Student(name) {  
    this.name = name;  
}  
Student.prototype = {  
    sleep: function () {}  
};
```

```
var billy = new Student('Billy');
```

```
var billy = {  
    name: 'Billy',  
    [[Prototype]]: <Student.prototype>  
};
```

Прямо как `Object.prototype`!

Object.prototype

```
Object.prototype = {  
  toString: function () {},  
  hasOwnProperty: function () {}  
};
```

```
var billy = { name: 'Billy' };
```

```
var billy = new Object({ name: 'Billy' });
```

Особое поле .prototype

1. Есть у каждой функции

```
function kawabanga(name) {  
    console.info('kawabanga!');  
}
```

2. Хранит объект

3. Имеет смысл только при вызове функции как конструктора

4. Имеет вложенное поле .constructor

Особое поле `.constructor`

- неперечисляемое
- хранит ссылку на саму функцию

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype.constructor === Student; // true
```

```
var billy = new Student('Billy');
```

```
console.info(billy.constructor.name); // Student
```

Конструктор «из коробки»

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype = {  
    sleep: function () {}  
};
```

Проблема: уничтожаем поле `.constructor`

Решение: не перезаписывать `.prototype`

Конструктор «из коробки»

```
function Student(name) {  
    this.name = name;  
}  
  
Student.prototype.sleep = function () {  
  
    console.info('zzzzZZ ...');  
}
```

```
var billy = new Student('Billy');  
  
billy.sleep(); // zzzzzZ ...
```

```
billy.constructor === Student; // true
```

Строим цепочку прототипов

```
function Student(name) {  
    this.name = name;  
}  
  
Student.prototype.sleep = function () {};
```

```
function Person() {  
    this.type = 'human';  
}  
  
Person.prototype.getName = function () {  
    return this.name;  
}
```

Строим цепочку прототипов

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype = Person.prototype;
```

```
Student.prototype.sleep = function () {};
```

```
var billy = new Student('Billy');
```

```
billy.getName();  
// Billy
```

Строим цепочку прототипов

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype = Person.prototype;  
Student.prototype.sleep = function () {};
```

```
function Lecturer(name) {  
    this.name = name;  
}
```

```
Lecturer.prototype = Person.prototype;
```

```
var sergey = new Lecturer('Sergey');
```

```
sergey.sleep(); // zzzZZ ...
```

billy



Student.prototype === Person.prototype



Object.prototype



null

billy



Student.prototype



Person.prototype



Object.prototype



null

Object.create()

```
function Student(name) {  
    this.name = name;  
}  
Student.prototype = Object.create(Person.prototype);  
Student.prototype.sleep = function () {};
```

```
function Lecturer(name) {  
    this.name = name;  
}  
Lecturer.prototype = Object.create(Person.prototype);
```

```
var sergey = new Lecturer('Sergey');  
sergey.sleep();
```

TypeError: sergey.sleep is not a function

billy



Student.prototype



Person.prototype



Object.prototype



null

sergey



Lecturer.prototype



Person.prototype



Object.prototype



null

Object.create()

Создаёт пустой объект, прототипом которого становится объект, переданный первым аргументом

```
var fruitProto = {  
    isUsefull: true  
}  
  
var apple = Object.create(fruitProto);  
  
apple.isUsefull; // true
```

Object.create()

```
var apple = Object.create(fruitProto);
```

```
Object.create = function(prototype) {  
    // Простейший конструктор пустых объектов  
    function EmptyFunction() {};  
  
    EmptyFunction.prototype = prototype;  
  
    return new EmptyFunction();  
};
```

Object.create()

```
var foreverAlone = Object.create(null);
```

```
foreverAlone.hasOwnProperty; // undefined
```

foreverAlone



null

Object.create()

```
Student.prototype = Object.create(Person.prototype);
```

```
Object.create = function(prototype) {  
    function EmptyFunction() {};  
  
    EmptyFunction.prototype = prototype;  
  
    return new EmptyFunction();  
};
```

```
Student.prototype = {  
    [[Prototype]]: <Person.prototype>  
}
```

Object.create()

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype = Object.create(Person.prototype);  
Student.prototype.sleep = function () {};
```

Object.create()

```
function Student(name) {  
    this.name = name;  
}
```

```
Student.prototype = Object.create(Person.prototype);  
Student.prototype.sleep = function () {};
```

```
Student.prototype.constructor = Student;
```


Итак, общее решение

```
function Person() {  
    this.type = 'human';  
}  
Person.prototype.getName = function () {  
    return this.name;  
};
```

```
function Student(name) {  
    this.name = name;  
}  
Student.prototype = Object.create(Person.prototype);  
Student.prototype.sleep = function () {};  
Student.prototype.constructor = Student;
```

```
var billy = new Student('Billy');
```

Object.getPrototypeOf()

```
var student = {  
  name: 'Billy',  
  [[Prototype]]: <person>  
}
```

```
var person = {  
  type: 'human',  
  getName: function () {}  
}
```

```
Object.getPrototypeOf(student) === person; // true
```

Object.prototype.isPrototypeOf()

```
function Student(name) {  
    this.name = name;  
}  
Student.prototype = Object.create(Person.prototype);  
Student.prototype.sleep = function () {};  
Student.prototype.constructor = Student;
```

```
var billy = new Student('Billy');
```

```
Student.prototype.isPrototypeOf(billy); // true
```

```
Person.prototype.isPrototypeOf(billy); // true
```

```
Object.prototype.isPrototypeOf(billy); // true
```

instanceof

```
function Student(name) {  
    this.name = name;  
}  
Student.prototype = Object.create(Person.prototype);  
Student.prototype.sleep = function () {};  
Student.prototype.constructor = Student;
```

```
var billy = new Student('Billy');
```

```
billy instanceof Student; // true
```

```
billy instanceof Person; // true
```

```
billy instanceof Object; // true
```

instanceof

```
billy instanceof Person;
```

```
billy.__proto__ === Person.prototype;  
// false -> Может, там null?
```

```
billy.__proto__ === null;  
// false -> Идём дальше по цепочке
```

```
billy.__proto__.__proto__ === Person.prototype;  
// true -> Возвращаем true
```

instanceof

```
var foreverAlone = Object.create(null);
```

foreverAlone  null

```
foreverAlone instanceof Object; // false
```

```
Object.create(null).__proto__ === Object.prototype;  
// false -> Может, там null?
```

```
Object.create(null).__proto__ === null;  
// true -> Так и есть, возвращаем false!
```

Дублирование кода в конструкторах

```
function Student(name) {  
    this.name = name;  
}
```

```
function Lecturer(name) {  
    this.name = name;  
}
```

```
function Person() {  
    this.type = 'human';  
}
```

Проблема: дублирование кода в

Дублирование кода в конструкторах

```
function Student(name) {  
    this.name = name;  
}
```

```
function Person() {  
  
    this.type = 'human';  
}
```

```
function Student(name) {}  
  
function Person(name) {  
    this.type = 'human';  
    this.name = name;  
}
```


Вызов одного конструктора внутри другого

```
function Person(name) {  
    this.type = 'human';  
    this.name = name;  
}
```

```
function Student(name) {}
```

```
function Student(name) {  
    // this ссылается на новый объект студента  
    Person.call(this, name);  
}
```

```
var billy = new Student('Billy');  
  
console.info(billy.name); // undefined
```

Вызов затеняемого метода в затеняющем

```
function Person(name) {  
    this.name = name;  
}  
Person.prototype.getName = function () {  
    return this.name;  
}
```

```
Student.prototype = Object.create(Person.prototype);  
Student.prototype.getName = function () {  
    return 'Student ' + this.getName();  
};
```

```
var billy = new Student('Billy');  
billy.getName();
```

RangeError: Maximum call stack size exceeded

Вызов затеняемого метода в затеняющем

```
function Person(name) {  
    this.name = name;  
}  
Person.prototype.getName = function () {  
    return this.name;  
}
```

```
Student.prototype = Object.create(Person.prototype);  
Student.prototype.getStudentName = function () {  
    return 'Student ' + this.getName();  
};
```

```
var billy = new Student('Billy');  
  
billy.getStudentName();
```

Вызов затеняемого метода в затеняющем

```
function Person(name) {  
    this.type = 'human';  
    this.name = name;  
}  
Person.prototype.getName = function () {  
  
    return this.name;  
}
```

```
Student.prototype = Object.create(Person.prototype);  
Student.prototype.getName = function () {  
    return 'Student ' +  
        Person.prototype.getName.call(this);  
};
```

`new` , `prototype` , `Object.create`

Можно проще!

Object.create()

```
var personProto = {  
  getName: function () {  
    return this.name;  
  }  
};
```

```
var studentProto = Object.create(personProto);
```

studentProto



personProto

```
studentProto.sleep = function () {};
```

Object.create()

```
var billy = Object.create(studentProto);
```

billy



studentProto



personProto

```
billy.name = 'Billy';
```

Object.create()

```
var personProto = {};  
personProto.getName = function () { return this.name; }
```

```
var studentProto = Object.create(personProto);  
studentProto.sleep = function () {};
```

```
var billy = Object.create(studentProto);  
billy.name = 'Billy';
```


Object.create()

```
var apple = Object.create(fruit, {  
  shape: { value: 'round', writable: false },  
  color: { value: 'Green' },  
  amount: { writable: true }  
});
```

```
apple.amount = 'half';
```

Object.create()

```
var personProto = {};  
personProto.getName = function () { return this.name; }
```

```
var studentProto = Object.create(personProto);  
studentProto.sleep = function () {};
```

```
studentProto.create = function (name) {  
    return Object.create(this, {  
        name: { value: name }  
    });  
}
```

```
var billy = studentProto.create('Billy');
```

new или Object.create?



«Классы»

```
function Student(name) {  
    this.name = name;  
}  
Student.prototype.getName = function () {  
    return this.name;  
};
```

```
class Student {  
    constructor(name) {  
        this.name = name;  
    }  
  
    getName() {  
        return this.name;  
    }  
}
```

«Классы»

```
class Student {  
    // ...  
}
```

```
var billy = new Student('Billy');
```

```
billy.getName(); // Billy
```

```
Student.prototype.isPrototypeOf(billy); // true
```

```
typeof Student; // function
```

class или Object.create?

Common Misconceptions About Inheritance in
JavaScript, Eric Elliott