

Язык JavaScript (ООП в
прототипном стиле)

JS

Функция как шаблон для новых объектов

```
function Storage() {  
    this.store = [];  
    this.set = function (key, value) {  
        this.store[key] = value;  
    };  
    this.get = function (key) {  
        return this.store[key];  
    };  
}
```

С помощью ключевого слова `new` и функции шаблона, мы можем создать объект с одними и теми же свойствами столько раз, сколько нам необходимо:

```
let store1 = new Storage();  
store1.set('name', 'Peter');  
let store2 = new Storage();
```

Прототипы

Приведенный подход работает замечательно, но он расточителен. При создании тысячи объектов Storage будет создана тысяча копий каждого метода и свойства, что требует много памяти. На основе **prototype** можно создать объекты, использующие общие свойства и методы.

```
function Storage() {  
    this.store = [];  
}  
Storage.prototype.set = function (key, value) {  
    this.store[key] = value;  
}  
Storage.prototype.get = function (key) {  
    return this.store[key];  
}
```

Прототипы

При создании объекта через `new`, в его прототип `__proto__` записывается ссылка из `prototype` функции-конструктора.

Объект, на который указывает ссылка `__proto__`, называется «прототипом».

Свойство `prototype` имеет смысл только у конструктора.

Добавление новых методов к уже созданным объектам

```
function Storage() {  
    this.store = [];  
}
```

```
var hub = new Storage();
```

```
Storage.prototype.set = function (key, value) {  
    this.store[key] = value;  
}
```

// Будет доступен метод set для всех ранее созданных объектов

```
hub.set('name', 'Peter');
```

Свойство constructor

Для каждого объекта созданного в JS в свойствах объекта определен конструктор. Он указывает на функцию шаблон из которого создавался объект.

```
let store1 = new Storage();  
store1.constructor;
```

Встроенные "классы" в JavaScript

```
let obj = {};  
alert( obj ); // "[object Object]" ?
```

```
let arr = [1, 2, 3];  
alert( arr ); // 1,2,3 ???
```

```
let user = "Вася";  
alert( user.toUpperCase() ); //Откуда функция??
```

```
let user = "Вася";  
user.age = 30;  
alert( user.age ); //???
```

Сравнение функционального и прототипного стиля

```
function Storage() {  
    this._store = [];  
};
```

```
Storage.prototype.set = function(key, value) {  
    this.store[key] = value;  
};
```

```
Storage.prototype.get = function(key){  
    if(!key) return;  
    return this.store[key];  
};
```


Сравнение функционального и прототипного стиля

Достоинства:

Функциональный стиль записывает в каждый объект и свойства и методы, а прототипный – только свойства. Поэтому прототипный стиль – быстрее и экономнее по памяти.

Недостатки:

При создании методов через прототип, мы теряем возможность использовать локальные переменные как приватные свойства, у них больше нет общей области видимости с конструктором.

Представленный недостаток весьма условен, так как при наследовании в функциональном стиле всё равно пришлось бы использовать `this`, чтобы потомок получил видимость внутренней переменной.

Наследование в прототипном стиле

```
function Shape(centerX, centerY){
    this.centerX = centerX;
    this.centerY = centerY;
};

Shape.prototype.toString = function(){
    return "Координаты центра " + this.centerX + ":" + this.centerY;
};

function Circle(centerX, centerY, radius){
    this.centerX = centerX;
    this.centerY = centerY;
    this.radius = radius;
};

Circle.prototype.toString = function(){
    return "Координаты центра " + this.centerX + ":" + this.centerY + "
Радиус " + this.radius;
};

Circle.prototype.__proto__ = Shape.prototype;
```

Смешанное наследование

```
function Shape(centerX, centerY){
    this.centerX = centerX;
    this.centerY = centerY;
};

Shape.prototype.toString = function(){
    return "Координаты центра " + this.centerX + ":" + this.centerY;
};

function Circle(centerX, centerY, radius){
    Shape.call(this, centerX, centerY);
    this.radius = radius;
};

Circle.prototype.toString = function(){
    return "Координаты центра " + this.centerX + ":" + this.centerY + "
Радиус " + this.radius;
};

Circle.prototype.__proto__ = Shape.prototype;
```