



ПРОТОТИП И КОНСТРУКТОР ОБЪЕКТА



МИХАИЛ КУЗНЕЦОВ



МИХАИЛ КУЗНЕЦОВ

Разработчик в ING Bank



[@mkuznetcov](https://www.instagram.com/mkuznetcov)

ПЛАН ЗАНЯТИЯ

1. Контекст выполнения
2. Конструктор объекта
3. Прототип объекта
4. Конструктор `Object`
5. Конструктор `Date` и его методы
6. Методы объекта `Math`



КОНТЕКСТ ВЫПОЛНЕНИЯ

ГЛОБАЛЬНЫЙ ОБЪЕКТ `window`

Любая var-переменная или функция, определённая в *глобальной области видимости*, хранятся в рамках глобального объекта `window`.

```
1  var seven = 7;
2  function takeFive() {
3      return 5;
4  }
5
6  console.log( seven ); // 5
7  console.log( window.seven ); // тоже 5
8
9  console.log( takeFive() ); // 5
10 console.log( window.takeFive() ); // тоже 5
```

КОНТЕКСТ. `this`

У любой функции есть ключевое слово `this`. Оно указывает на тот объект, к которому эта функция прикреплена.

В глобальной области видимости `this` указывает на `window`.

```
1 function getThis() {  
2   console.log( this );  
3 }  
4  
5 let person = {  
6   getThis  
7 }  
8  
9 person.getThis(); // объект person  
10 window.getThis(); // объект window  
11 getThis(); // объект window
```

КОНТЕКСТ. СТРОГИЙ РЕЖИМ

В **строгом режиме** `this` для функций глобальной области видимости, вызванных без `window` имеют значение `undefined`.

```
1  'use strict';
2  function getThis() {
3      console.log( this );
4  }
5
6  let person = {
7      getThis
8  }
9
10 person.getThis(); // объект person
11 window.getThis(); // объект window
12 getThis(); // undefined
```

ЗАЧЕМ НУЖЕН `this` ?

`this` позволяет избавиться от дублирования кода.

До `this`:

```
1  let ivan = {  
2    firstName: 'Иван',  
3    showName() {  
4      console.log( `Имя: ${ivan.firstName}` );  
5    }  
6  }  
7  
8  let oleg = {  
9    firstName: 'Олег',  
10   showName() {  
11     console.log( `Имя: ${oleg.firstName}` );  
12   }  
13 }  
14  
15 ivan.showName(); //
```


После `this`:

```
1 function showName() {  
2   console.log( `Имя: ${this.firstName}` );  
3 }  
4  
5 let ivan = {  
6   firstName: 'Иван',  
7   showName  
8 }  
9  
10 let oleg = {  
11   firstName: 'Олег',  
12   showName  
13 }
```

Для самостоятельного ознакомления: [Ключевое слово this в JavaScript для начинающих.](#)



КОНСТРУКТОР ОБЪЕКТА

ПРИМЕР

Предположим, реализуем CRM, для этого создаем объект клиента.

```
1  const person = {};  
2  person.name = 'Vasya';  
3  person.gender = 'M';
```

Переиспользовать такой код не получится, нарушается принцип DRY (будет повторяться логика создания и наполнения объекта).

Решение: использовать конструктор.

КОНСТРУКТОР ОБЪЕКТА

Конструктор - специальный блок инструкций, вызываемый при создании объекта.

```
1 function Person(name, gender) {  
2   this.name = name;  
3   this.gender = gender;  
4 }
```

ОПЕРАТОР `new`

Позволяет создавать объекты через вызов функций.

Особенности работы функций, вызванных через оператор `new`:

- Создаётся новый пустой объект;
- Ключевое слово `this` получает ссылку на этот объект;
- Функция выполняется;
- Возвращается `this` без явного указания.

СОЗДАНИЕ НОВЫХ ОБЪЕКТОВ ПРИ ПОМОЩИ ОПЕРАТОРА `new` (ДЕМО)

Таким образом создание новых объектов может быть реализовано вызыванием обычной функции с оператором `new`.

```
1 function Car(engine) {  
2   this.engine = engine;  
3 }  
4  
5 const car = new Car('v8');
```



ПРОТОТИП ОБЪЕКТА

ПРИМЕР

Вам необходимо реализовать один и тот же метод `getNumber` в двух классах `Car` и `Bus`.

```
1  const getNumber = () => {  
2    return this.number;  
3  }  
4  
5  Car.getNubmer = getNumber;  
6  // ...  
7  Bus.getNumber = getNumber;
```

Недостаток подхода заключается в том, что необходимо прописывать каждый раз метод `getNumber` всем транспортным средствам.

Решение: использовать прототипное наследование.

ПРОТОТИП ОБЪЕКТА

Объекты можно организовать в цепочки так, чтобы свойство, не найденное в одном объекте, автоматически искалось бы в другом.

Явное обозначение прототипа происходит через свойство `prototype` конструктора.

Таким образом реализуется прототипное наследование.

ПРОТОТИП ОБЪЕКТА

```
1  function Vehicle(n) {  
2      this.numbers = n;  
3  }  
4  
5  Vehicle.getNumber = function() {  
6      return this.number;  
7  }  
8  
9  function Car() {  
10     this.number = 4;  
11 }  
12  
13 function Bus() {  
14     this.number = 6;  
15 }  
16  
17 Car.prototype = Vehicle;  
18 Bus.prototype = Vehicle;
```



ВНИМАНИЕ

Использование `__proto__` не рекомендовано и опасно!

<https://developer.mozilla.org>

ПРИМЕР ПРОТОТИПА

```
1  const Predator = {
2    food: 'meat',
3  };
4
5  function Tiger(name) {
6    this.name = name;
7  }
8
9  const tiger = new Tiger('Vasya');
10
11 console.log(tiger.name); // 'Vasya'
12 console.log(tiger.food); // undefined
```

ПРИМЕР ПРОТОТИПА (ДЕМО)

```
1  const predator = {  
2    food: 'meat',  
3  };  
4  
5  function Tiger(name) {  
6    this.name = name;  
7  }  
8  Tiger.prototype = predator; // определяем прототип  
9  
10 const tiger = new Tiger('Vasya');  
11  
12 console.log(tiger.name); // 'Vasya'  
13 console.log(tiger.food); // 'meat'
```



КОНСТРУКТОР **Object**

ЧТО ЗА `Object` ?

Если посмотреть на цепочки прототипов различных типов, то в конце любой цепочки будет `Object`.

```
1 new Boolean()  
2 //Boolean {false}  
3 //    __proto__: Boolean  
4 //    constructor: f Boolean()  
5 //    __proto__:  
6 //    constructor: f Object()  
7  
8 new String()  
9 //String { "", length: 0}  
10 //    __proto__: String  
11 //    constructor: f String()  
12 //    __proto__:  
13 //    constructor: f Object()
```

Закономерный вопрос, а что за класс `Object`, и почему он есть в большинстве цепочек наследования?

ВАЖНОСТЬ КОНСТРУКТОРА `Object`

Все объекты в JavaScript являются потомками `Object`.

Соответственно, все объекты наследуют методы и свойства из прототипа объекта `Object.prototype`.

```
String => Object
```

```
Array => Object
```


КОНСТРУКТОР `Object`

Создает объект такого типа, который соответствует переданному значению.

```
new Object(true); // Boolean { true }  
new Object('js'); // String { "js" }
```

Если переданным значением является `null` или `undefined`, создаёт и возвращает пустой объект.

```
new Object(undefined) // {}  
new Object(null) // {}
```

Создается пустой объект обертку.

ВЫЗОВ В НЕ-КОНСТРУКТОРНОМ КОНТЕКСТЕ

В таком случае, `Object` ведёт себя идентично коду `new Object()`.

```
Object.toString // (new Object()).toString
```

`Object.prototype.hasOwnProperty()`

Метод определяет, содержит ли объект указанное свойство в качестве собственного свойства объекта.

В отличие от оператора `in` не проверяет наличие указанного свойства по цепочке прототипов.

Object.prototype.hasOwnProperty()

```
1  const Predator = {
2    food: 'meat',
3  };
4
5  function Tiger(name) {
6    this.name = name;
7  }
8  Tiger.prototype = Predator; // определяем прототип
9
10 const tiger = new Tiger('Vasya');
11
12 tiger.hasOwnProperty('name'); // true
13 tiger.hasOwnProperty('food'); // false
14
15 'name' in tiger; // true
16 'food' in tiger; // true
```

Object.create(prototype)

Создаёт новый объект с указанным объектом прототипа (prototype).

```
const tiger = Object.create(predator);  
tiger.food; // meat
```

Эквивалентно предыдущему примеру.

Object.assign(target, ...sources)

Создаёт новый объект путём копирования значений всех собственных перечисляемых свойств из одного или более исходных объектов в целевой объект. В новом стандарте есть аналог — `...`.

```
1  const person = {
2    name: 'Frederic',
3  };
4
5  const account = {
6    balance: 14000,
7  };
8
9  const info1 = Object.assign({}, person, account);
10 info1.name; // 'Frederic'
11 info1.balance; // '14000'
12
13 const info2 = {
14   ...person,
15   ...account,
16 };
17 info2.name; // 'Frederic'
18 info2.balance; // '14000'
```

`Object.keys(obj)` | `Object.values(obj)`

`keys` — возвращает массив, содержащий имена всех собственных перечисляемых свойств переданного объекта.

`values` — возвращает массив значений перечисляемых свойств объекта в том же порядке что и цикл `for...in`.

```
Object.keys(info); // ["name", "balance"]
```


```
Object.values(info); // ["Frederic", 14000]
```

Object.entries(obj)

Метод возвращает массив собственных перечисляемых свойств указанного объекта в формате `[key, value]`, в том же порядке, что и в цикле `for...in` (разница в том, что `for-in` также перечисляет свойства из цепочки прототипов).

```
Object.entries(info); // [ ["name", "Frederic"], ["balance", 14000] ]
```

Он не идет вглубь по прототипам!



КОНСТРУКТОР `Date` И ЕГО МЕТОДЫ



РАБОТА С ДАТАМИ

Для работы с датами в JS предусмотрен класс `Date`.

Он сложный, противоречивый и местами опасный.

КОНСТРУКТОР `Date`

Дата и время представлены в JavaScript одним объектом: `Date`.

```
1 new Date();
2 /*
3  "Fri Nov 23 2018 01:03:26 GMT+0300
4  (Москва, стандартное время)"
5  */
6
7 new Date(2018, 0, 1, 12, 00);
8 /*
9  "Mon Jan 01 2018 12:00:00 GMT+0300
10 (Москва, стандартное время)"
11 */
```

Отсчёт месяцев начинается с нуля.

Отсчёт дней недели (для `getDay()`) тоже начинается с нуля (и это воскресенье).

СТАНДАРТНЫЕ ФОРМАТЫ ДАТ И ВРЕМЕНИ

ISO 86001 - в виде строки

```
date.toString();  
/*  
"Thu Nov 22 2018 21:48:54 GMT+0300  
(Москва, стандартное время)"  
*/
```

Удобно для сортировок и «человеко-читаемо».

Unix Timestamp - в виде числа

```
+date; // 1542912588582
```

Удобно находить разницу между временными метками.

УСТАНОВКА КОМПОНЕНТА ДАТ

```
1 date.setFullYear(year [, month, date])
2 date.setMonth(month [, date])
3 date.setDate(date)
4 date.setHours(hour [, min, sec, ms])
5 date.setMinutes(min [, sec, ms])
6 date.setSeconds(sec [, ms])
7 date.setMilliseconds(ms)
8 ...
```

<https://developer.mozilla.org>

ПОЛУЧЕНИЕ КОМПОНЕНТА ДАТ

```
const date = new Date();  
date.getFullYear() // 2018  
date.getMonth() // 10  
date.getDate() // 22  
date.getHours() // 21  
date.getMinutes() // 32  
date.getSeconds() // 41  
date.getMilliseconds() // 122  
...
```

<https://developer.mozilla.org>

МЕТОД `Date.now()`

Метод `Date.now()` возвращает дату сразу в виде миллисекунд.

```
Date.now() // 1542912986688
```

Такого результата можно добиться.

```
Number(new Date()); // 1542912986688
```

АВТОИСПРАВЛЕНИЕ ДАТЫ

Неправильные компоненты даты автоматически распределяются по остальным.

```
new Date(2018, 12, 40);  
// Sat Feb 09 2019 00:00:00 GMT+0300
```

Внимание! Будьте осторожны с этой особенностью `Date`.

Intl

Объект, позволяющий с легкостью интернационализировать даты.

```
1  const date = new Date();  
2  
3  const formatter = new Intl.DateTimeFormat("ru");  
4  formatter.format(date) // "23.11.2018"
```

Не забудьте подключать полифилл при использовании для поддержки в старых версиях браузеров.



МЕТОДЫ ОБЪЕКТА Math

ПРИМЕР

Предположим вы реализуете интерфейс и вам понадобилось сгенерировать случайное число, или его округлить.

Для подобных случаев в JS предусмотрен объект `Math`, в котором есть все часто используемые функции касающиеся математических выражений.

ОБЪЕКТ **Math**

Math - глобальный объект, помогающий в математических вычислениях.

Имеет свойства, константы:

```
Math.PI; // 3.141592653589793  
Math.E; // 2.718281828459045
```

И методы:

```
Math.random(); // Случайное число от 0 до 1  
Math.max(12, 13, 432, -1); // 432  
Math.min(12, 13, 432, -1); // -1
```

ОКРУГЛЕНИЕ ЧИСЕЛ

```
Math.floor(2.6); // в меньшую сторону  
Math.round(2.6); // до ближайшего целого  
Math.ceil(2.1); // в большую сторону
```

МОДУЛЬ ЧИСЛА

Для нахождения модуля числа используют `Math.abs(value)`.

```
Math.abs(-2); // 2
```

```
Math.abs(2); // 2
```

ЧТО МЫ УЗНАЛИ?

- Познакомились с контекстом `this`.
- Разобрали понятие конструктора и оператор `new`.
- Изучили цепочки прототипов, и иерархию базовых классов в соответствии с ней.
- Познакомились с объектом `Math`, помогающим в математических вычислениях, и классом `Date`, облегчающим манипуляции с датами.



ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.



Спасибо за внимание! Время задавать вопросы

МИХАИЛ КУЗНЕЦОВ

