

🏠 → [Язык JavaScript](#) → [Объекты: основы](#)

📅 22 февраля 2023 г.

# Конструктор, оператор "new"

Обычный синтаксис `{...}` позволяет создать только один объект. Но зачастую нам нужно создать множество похожих, однотипных объектов, таких как пользователи, элементы меню и так далее.

Это можно сделать при помощи функции-конструктора и оператора `"new"`.

## Функция-конструктор

Функции-конструкторы технически являются обычными функциями. Но есть два соглашения:

1. Имя функции-конструктора должно начинаться с большой буквы.
2. Функция-конструктор должна выполняться только с помощью оператора `"new"`.

Например:

```
1 function User(name) {
2   this.name = name;
3   this.isAdmin = false;
4 }
5
6 let user = new User("Jack");
7
8 alert(user.name); // Jack
9 alert(user.isAdmin); // false
```



Когда функция вызывается как `new User(...)`, происходит следующее:

1. Создаётся новый пустой объект, и он присваивается `this`.
2. Выполняется тело функции. Обычно оно модифицирует `this`, добавляя туда новые свойства.
3. Возвращается значение `this`.

Другими словами, `new User(...)` делает что-то вроде:

```
1 function User(name) {
2   // this = {}; (неявно)
3
4   // добавляет свойства к this
5   this.name = name;
6   this.isAdmin = false;
7
8   // return this; (неявно)
9 }
```

Таким образом, `let user = new User("Jack")` возвращает тот же результат, что и:

```
1 let user = {
2   name: "Jack",
3   isAdmin: false
4 };
```

Теперь, если нам будет необходимо создать других пользователей, мы можем просто вызвать `new User("Ann")`, `new User("Alice")` и так далее. Данная конструкция гораздо удобнее и читабельнее, чем многократное создание литерала объекта.

Это и является основной целью конструкторов – реализовать код для многократного создания однотипных объектов.

Давайте ещё раз отметим – технически любая функция (кроме стрелочных функций, поскольку у них нет `this`) может использоваться в качестве конструктора. Его можно запустить с помощью `new`, и он выполнит выше указанный алгоритм. Подобные функции должны начинаться с заглавной буквы – это общепринятое соглашение, чтобы было ясно, что функция должна вызываться с помощью «new».

#### **new function() { ... }**

Если в нашем коде присутствует большое количество строк, создающих один сложный объект, то мы можем обернуть их в функцию-конструктор, которая будет немедленно вызвана, вот так:

```
1 // создаём функцию и сразу же вызываем её с помощью new
2 let user = new function() {
3   this.name = "John";
4   this.isAdmin = false;
5
6   // ...другой код для создания пользователя
7   // возможна любая сложная логика и инструкции
8   // локальные переменные и так далее
9 };
```

Такой конструктор не может быть вызван снова, так как он нигде не сохраняется, просто создаётся и тут же вызывается. Таким образом, этот трюк направлен на инкапсуляцию кода, который создаёт отдельный объект, без возможности повторного использования в будущем.

## Проверка на вызов в режиме конструктора: `new.target`

#### **Продвинутая возможность**

Синтаксис из этого раздела используется крайне редко. Вы можете пропустить его, если не хотите углубляться в детали языка.

Используя специальное свойство `new.target` внутри функции, мы можем проверить, вызвана ли функция при помощи оператора `new` или без него.

В случае обычного вызова функции `new.target` будет `undefined`. Если же она была вызвана при помощи `new`, `new.target` будет равен самой функции.



```
1 function User() {
2   alert(new.target);
3 }
4
5 // без "new":
6 User(); // undefined
7
8 // с "new":
9 new User(); // function User { ... }
```

Это можно использовать внутри функции, чтобы узнать, была ли она вызвана при помощи `new`, «в режиме конструктора», или без него, «в обычном режиме».

Также мы можем сделать, чтобы вызовы с `new` и без него делали одно и то же:



```
1 function User(name) {
2   if (!new.target) { // в случае, если вы вызвали меня без оператора new
3     return new User(name); // ...я добавлю new за вас
4   }
5
6   this.name = name;
7 }
8
9 let john = User("John"); // переадресовывает вызов на new User
10 alert(john.name); // John
```

Такой подход иногда используется в библиотеках, чтобы сделать синтаксис более гибким. Чтобы люди могли вызывать функцию с `new` и без него, и она все ещё могла работать.

Впрочем, вероятно, это не очень хорошая практика использовать этот трюк везде, так как отсутствие `new` может ввести разработчика в заблуждение. С `new` мы точно знаем, что создаётся новый объект.

## Возврат значения из конструктора, return

Обычно конструкторы не имеют оператора `return`. Их задача – записать все необходимое в `this`, и это автоматически становится результатом.

Но если `return` всё же есть, то применяется простое правило:

- При вызове `return` с объектом, вместо `this` вернётся объект.
- При вызове `return` с примитивным значением, оно проигнорируется.

Другими словами, `return` с объектом возвращает этот объект, во всех остальных случаях возвращается `this`.

К примеру, здесь `return` замещает `this`, возвращая объект:



```
1 function BigUser() {
2
3   this.name = "John";
4
5   return { name: "Godzilla" }; // <-- возвращает этот объект
6 }
```

```
7
8 alert( new BigUser().name ); // Godzilla, получили этот объект
```

А вот пример с пустым `return` (или мы могли бы поставить примитив после `return`, неважно):

```
1 function SmallUser() {
2     this.name = "John";
3     return; // <-- возвращает this
4 }
5
6 alert( new SmallUser().name ); // John
```

Обычно у конструкторов отсутствует `return`. Здесь мы упомянули особое поведение с возвращаемыми объектами в основном для полноты картины.

### Пропуск скобок

Кстати, мы можем не ставить круглые скобки после `new`:

```
1 let user = new User; // <-- без скобок
2 // то же, что и
3 let user = new User();
```

Пропуск скобок считается плохой практикой, но просто чтобы вы знали, такой синтаксис разрешён спецификацией.

## Создание методов в конструкторе

Использование конструкторов для создания объектов даёт большую гибкость. Функции-конструкторы могут иметь параметры, определяющие, как создавать объект и что в него записывать.

Конечно, мы можем добавить к `this` не только свойства, но и методы.

Например, `new User(name)` ниже создаёт объект с заданным `name` и методом `sayHi`:

```
1 function User(name) {
2     this.name = name;
3
4     this.sayHi = function() {
5         alert( "Меня зовут: " + this.name );
6     };
7 }
8
9 let john = new User("John");
10
11 john.sayHi(); // Меня зовут: John
12
13 /*
```

```
14 john = {  
15     name: "John",  
16     sayHi: function() { ... }  
17 }  
18 */
```

Для создания сложных объектов есть и более продвинутый синтаксис – **классы**, который мы рассмотрим позже.

## Итого

- Функции-конструкторы или просто конструкторы, являются обычными функциями, но существует общепринятое соглашение именовать их с заглавной буквы.
- Функции-конструкторы следует вызывать только с помощью **new**. Такой вызов подразумевает создание пустого **this** в начале и возврат заполненного в конце.

Мы можем использовать конструкторы для создания множества похожих объектов.

JavaScript предоставляет функции-конструкторы для множества встроенных объектов языка: таких как **Date**, **Set**, и других, которые нам ещё предстоит изучить.

### Мы ещё вернёмся к объектам!

В этой главе мы рассмотрели только основы объектов и конструкторов. Данная информация необходима нам для дальнейшего изучения типов данных и функций в последующих главах.

Как только мы с ними разберёмся, мы вернёмся к объектам для более детального изучения в главах **Прототипы, наследование и Классы**.

## Задачи

### Две функции - один объект

важность: 2

Возможно ли создать функции **A** и **B**, чтобы `new A() == new B()`?

```
1 function A() { ... }  
2 function B() { ... }  
3  
4 let a = new A();  
5 let b = new B();  
6  
7 alert( a == b ); // true
```

Если да – приведите пример вашего кода.

решение

Да, возможно.



Если функция возвращает объект, то `new` вернёт его вместо `this`.

Таким образом, они могут, к примеру, возвращать один и тот же внешне определённый объект `obj`:

```
1 let obj = {};  
2  
3 function A() { return obj; }  
4 function B() { return obj; }  
5  
6 alert( new A() == new B() ); // true
```

## Создайте калькулятор при помощи конструктора, new Calculator

важность: 5

Создайте функцию-конструктор `Calculator`, которая создаёт объекты с тремя методами:

- `read()` запрашивает два значения при помощи `prompt` и сохраняет их значение в свойствах объекта.
- `sum()` возвращает сумму этих свойств.
- `mul()` возвращает произведение этих свойств.

Например:

```
1 let calculator = new Calculator();  
2 calculator.read();  
3  
4 alert( "Sum=" + calculator.sum() );  
5 alert( "Mul=" + calculator.mul() );
```

[Запустить демо](#)

[Открыть песочницу с тестами для задачи.](#)

решение

```
1 function Calculator() {  
2  
3   this.read = function() {  
4     this.a = +prompt('a?', 0);  
5     this.b = +prompt('b?', 0);  
6   };  
7  
8   this.sum = function() {  
9     return this.a + this.b;  
10  };  
11  
12  this.mul = function() {  
13    return this.a * this.b;  
14  };  
}
```

```
15 }
16
17 let calculator = new Calculator();
18 calculator.read();
19
20 alert( "Sum=" + calculator.sum() );
21 alert( "Mul=" + calculator.mul() );
```

[Открыть решение с тестами в песочнице.](#)

## Создайте new Accumulator

важность: 5

Создайте функцию-конструктор `Accumulator(startingValue)`.

Объект, который она создаёт, должен уметь следующее:

- Хранить «текущее значение» в свойстве `value`. Начальное значение устанавливается в аргументе конструктора `startingValue`.
- Метод `read()` должен использовать `prompt` для считывания нового числа и прибавления его к `value`.

Другими словами, свойство `value` представляет собой сумму всех введённых пользователем значений, с учётом начального значения `startingValue`.

Ниже вы можете посмотреть работу кода:

```
1 let accumulator = new Accumulator(1); // начальное значение 1
2
3 accumulator.read(); // прибавляет введённое пользователем значение к текущему
4 accumulator.read(); // прибавляет введённое пользователем значение к текущему
5
6 alert(accumulator.value); // выведет сумму этих значений
```

[Запустить демо](#)

[Открыть песочницу с тестами для задачи.](#)

решение

```
1 function Accumulator(startingValue) {
2   this.value = startingValue;
3
4   this.read = function() {
5     this.value += +prompt('Сколько нужно добавить?', 0);
6   };
7
8 }
9
10 let accumulator = new Accumulator(1);
```

```
11 accumulator.read();
12 accumulator.read();
13 alert(accumulator.value);
```

[Открыть решение с тестами в песочнице.](#)



[Предыдущий урок](#)

[Следующий урок](#)



Поделиться



[Карта учебника](#)

Проводим [курсы по JavaScript и фреймворкам.](#)



## Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите [на GitHub](#).
- Для одной строки кода используйте тег `<code>` , для нескольких строк кода — тег `<pre>` , если больше 10 строк — ссылку на песочницу ([plnkr](#), [JSBin](#), [codepen...](#))
- Если что-то непонятно в статье — пишите, что именно и с какого места.



Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS ?

Имя

 33

Поделиться

Лучшие Новые Старые



**Джун на фронте**

день назад

Приглашаю в **блог JS-прогресса** !

В тг **Джун на фронте** я врываюсь в веб! Учусь делать динамичные сайты, без банальных конструкторов



Мы вместе пройдем собесы, отправим первый отклик, выполним тестовы, разберем алгоритмы и окунемся в море кода!

0 0 Ответить • Поделиться ›

**K**

**KingNut Academy**

9 дней назад

Разбор задач, тестов и всего что поможет при прохождении собеседования на должность frontend developer (html, css, js, ts, react). Заходи к нам в ТГ @interview\_masters

0 1 Ответить • Поделиться ›

**J**

**JuranTouran**

10 дней назад

Кто мне пояснит, ранее говорилось ,что:

```
let a = {};
```

```
let b = {}; // два независимых объекта
```

```
alert( a == b ); // false
```

А в первой задаче решение вот такое:

```
let obj = {};
```

```
function A() { return obj; }
```

```
function B() { return obj; }
```

```
let a = new A();
```

```
let b = new B();
```

```
console.log(a == b); //true
```

Разве, объект а и б не являются двумя независимыми объектами???

В первом случае у нас есть ссылка "a" и ссылка "b", они ссылаются на два разных объекта, хоть содержимое этих объектов идентичное. Во втором случае у нас есть ссылка "obj", и обе функции возвращают ссылку "obj", то есть это одна и та же ссылка, которое указывает на один и тот же объект, поэтому они равны.

00

Ответить • Поделиться ›

J

JuranTouran

→ Klishe

9 дней назад

Спасибо!

00

Ответить • Поделиться ›

A

adawda adwadw

месяц назад

Не совсем понятно, в чем разница между new, и обычной функцией возвращающей объект?

```
function f() {
  return {
    key: 'value'
  }
}
```

```
const obj = f();
```


Или

```
function MakeObj() {
  this.key = 'value';
}
```

```
const obj = new MakeObj();
```

10

Ответить • Поделиться ›



G455

→ adawda adwadw

месяц назад

разницу ты более менее поймешь когда дойдешь до глав - F.prototype и Встроенные прототипы. Функция-конструктор задаёт всем объектам созданным через неё прототип, либо по умолчанию, либо тот который укажешь в свойстве Function.prototype (у функций есть свойства, про это узнаешь в главе - Объект функции, NFE). В общем когда продвинешься дальше станет понятнее.

10

Ответить • Поделиться ›

P

PVG

месяц назад

```
function User() {
  this.Read = function () {
    this.firstNumber = +prompt('Enter first number', 0);
    this.secondNumber = +prompt('Enter second number', 0);
  }
  this.Calc = function () {
    alert(this.firstNumber + this.secondNumber);
  }

  this.Mult = function () {
    alert(this.firstNumber * this.secondNumber)
  }
}

let f = new User();
f.Read();
f.Calc();
f.Mult();
```

```
function Accumulator(startingValue) {
  this.value = startingValue;
  this.Read = function () {
    this.value += +prompt("Enter number to sum", 0);
    alert(this.value)
  }
}

let t = new Accumulator(2);
t.Read();
```

0 0 Ответить • Поделиться ›



**Player\_50B**

2 месяца назад



```
const calculator = new Calculator();

function Calculator(){
  this.read = function(){
    this.a = +prompt("a?", "0");
    this.b = +prompt("b?", "0");
  }
  this.sum = function(){
    return this.a + this.b;
  }
  this.mul = function(){
    return this.a * this.b;
  }
}

calculator.read();
alert( "Sum=" + calculator.sum() );
alert( "Mul=" + calculator.mul() );
```

0 0 Ответить • Поделиться ›



**Player\_50B**

2 месяца назад edited



```
const accumulator = new Accumulator(1);

function Accumulator(startingValue){
  this.value = startingValue;
  this.read = function (){
    user_values = +prompt("Сколько нужно добавить?", 0);
    return this.value += user_values;
  }
}

accumulator.read();
accumulator.read();
alert(accumulator.value);
```

0 0 Ответить • Поделиться ›



**Максим Григорьев**

➔ Player\_50B



— — — месяц назад

Так не работает

0 0 Ответить • Поделиться ›



**Player\_50B**

➔ Максим Григорьев

месяц назад

Почему? У меня работает как в демо задачи.

0 0 Ответить • Поделиться ›

**G**

**Gleb Milovanov**

➔ Player\_50B

месяц назад

если использовать use strict, то не будет работать, потому что user\_values изначально не задано

3 0 Ответить • Поделиться ›

**V**

**Vladimir Degtyarev**

2 месяца назад

```
function Accumulator(value) {
  this.value = value;
  this.read = function() {
    this.number = +prompt('Введите число: ', 0);
    this.value += this.number;
  };
}
```

1 0 Ответить • Поделиться ›



**Alexander Mandrov**

3 месяца назад

Обстоятельства и желания сподвигли меня на создание собственного канала. Там я рассказываю о жизни и работе, о том, как стал программистом в 20 лет, коротко и по делу

ТГ @unsleeping706

0 6 Ответить • Поделиться ›

**M**

**Michael Any**

3 месяца назад

```
function A() {
  return A
}
function B() {
  return A
}

let a = new A();
let b = new B();

alert( a == b );
```

0 0 Ответить • Поделиться ›

0 0 Ответить • Поделиться ›



**Megaton**

3 месяца назад



Вообще не понимаю this и эту тему, соответственно.

4 0 Ответить • Поделиться ›



**Paul Pumphut**

→ Megaton

3 месяца назад



Плохо...

0 3 Ответить • Поделиться ›



**Stranger**

3 месяца назад



Разъясните, пожалуйста, такой момент:

если мы, например, в функции-конструкторе задаём скрытое свойство с помощью переменной `let _age = age`, то где оно хранится? В созданный объект ведь переменная не передается? И когда мы например вызовем геттер, откуда он будет получать значение? Из локального окружения функции-конструкторе (замыкание)?

0 0 Ответить • Поделиться ›



**Dima Kim**

→ Stranger

2 месяца назад



`let _age`; нельзя

если ты хочешь создать переменную то только с `this` следовательно `this.let _age`

итого:

`this.let _age = age`, где `this.let _age` будет принадлежать объекту, а `age` - передаваемый параметр из вне

1 0 Ответить • Поделиться ›

[Загрузить ещё комментарии](#)

[Подписаться](#)

[Privacy](#)

[Не продавайте мои данные](#)

