> Эта страница была переведена с английского языка силами сообщества. Вы тоже можете внести свой вклад, присоединившись к русскоязычному сообществу MDN Web Docs.

var

Оператор `var` объявляет переменную, инициализируя её, при необходимости.

Интерактивный пример

Синтаксис

```
var varname1 [= value1 [, varname2 [, varname3 ... [, varnameN]]]];
```

`varnameN`

Имя переменной. Может использоваться любой допустимый идентификатор.

`valueN`

Значение переменной. Любое допустимое выражение. По умолчанию значение *`undefined`*.

Описание

Объявление переменной всегда обрабатывается до выполнения кода, где бы она ни находилась. Область видимости переменной, объявленной через `var`, это её текущий *контекст выполнения*. Который может ограничиваться функцией или быть глобальным, для переменных, объявленных за пределами функции.

Присвоение значения необъявленной переменной подразумевает, что она будет создана как глобальная переменная (переменная становится свойством глобального объекта) после выполнения присваивания значения. Различия между объявленной и

необъявленной переменными следующие:

1. Объявленные переменные ограничены контекстом выполнения, в котором они были объявлены. Необъявленные переменные всегда глобальны.

```
function x() {
  y = 1; // возбудит ReferenceError в "строгом режиме"
  var z = 2;
}

x();

console.log(y); // выведет "1"
console.log(z); // возбудит ReferenceError: z не определён вне x
```

2. Объявленные переменные инициализируются до выполнения любого кода. Необъявленные переменные не существуют до тех пор, пока к ним не выполнено присваивание.

JS

```
console.log(a); // Возбудит ReferenceError.
console.log("still going..."); // Не выполнится.
```

JS

```
var a;
console.log(a); // Выведет "undefined" или "", в зависимости от браузера.
console.log("still going..."); // Выведет "still going...".
```

3. Объявленные переменные, независимо от контекста выполнения, являются ненастраиваемыми свойствами. Необъявленные переменные это настраиваемые свойства (т.е. их можно удалять).

JS

```
var a = 1;
b = 2;

delete this.a; // Возбудит TypeError в "строгом режиме". В "нестрогом режиме" будет ошибка без уведомления.
delete this.b;

console.log(a, b); // Возбудит ReferenceError.
// Свойство 'b' было удалено и больше не существует.
```

Из-за перечисленных различий, использование необъявленных переменных может привести к непредсказуемым последствиям. Рекомендовано всегда объявлять переменные, вне зависимости, находятся они внутри функции или в глобальном контексте. Присваивание значения необъявленной переменной в строгом режиме (en-US) ECMAScript 5 возбуждает ошибку.

Поднятие переменных

Объявление переменных (как и любые другие объявления) обрабатываются до выполнения кода. Где бы не находилось объявление, это равнозначно тому, что переменную объявили в самом начале кода. Это значит, что переменная становится доступной до того, как она объявлена. Такое поведение называется "поднятием" (в некоторых источниках "всплытием").

JS

```
bla = 2;
var bla;
// ...

// читается как:
```

```
var bla;
bla = 2;
```

Поэтому объявление переменных рекомендовано выносить в начало их области видимости (в начало глобального кода или в начало функции). Это даёт понять какие переменные принадлежат функции (т.е. являются локальными), а какие обрабатываются в цепи областей видимости (т.е. являются глобальными).

Важно отметить, что подъем будет влиять на объявление переменной, но не на инициализацию её значения. Значение присваивается при выполнении оператора присваивания:

JS

```
function do_something() {
  console.log(bar); // выведет undefined
  var bar = 111;
  console.log(bar); // выведет 111
}

// ...неявно понимается как:

function do_something() {
  var bar;
  console.log(bar); // выведет undefined
  bar = 111;
  console.log(bar); // выведет 111
}
```

Примеры

Объявление и инициализация двух переменных

JS

```
var a = 0,
  b = 0;
```

Присвоение двум переменным одного строкового значения

JS

```
var a = "A";
var b = a;

// Равнозначно:

var a,
  b = (a = "A");
```

Следите за порядком присвоения значений переменным

JS

```
var x = y,
  y = "A";
console.log(x + y); // undefinedA
```

В примере, `x` и `y` объявлены до выполнение кода, присвоение выполняется позже. Когда происходит присваивание "`x = y`", `y` уже существует со значением '`undefined`', так что ошибка `ReferenceError` не генерируется. И переменной `x` присваивается неопределённое значение. Потом переменной `y` присваивается значение 'A'. Получается, что после выполнения первой строки кода `x === undefined && y === 'A'`, отсюда и результат.

Инициализация нескольких переменных

JS

```
var x = 0;

function f() {
  var x = (y = 1); // x - объявляется локально. y - глобально!
}
f();

console.log(x, y); // 0, 1
// значение x взято из глобальной переменной, как и ожидалось
// значение переменной y доступно глобально
```

Такой же пример, но в строгом режиме:

JS

```
"use strict";

var x = 0;
function f() {
  var x = (y = 1); // Throws a ReferenceError in strict mode.
}
f();

console.log(x, y);
```

Неявные глобальные переменные и внешняя область видимости

Переменные могут ссылаться на переменные внешней области видимости функции, и это может выглядеть неявно:

JS

```
var x = 0; // x объявлена глобально, затем присваивается значение 0

console.log(typeof z); // undefined, пока ещё z не существует

function a() {
  // когда функция a вызвана,
  var y = 2; // y объявляется локально в функции a, затем присваивается 2

  console.log(x, y); // 0 2

  function b() {
    // когда функция b вызвана
    x = 3; // присваивается 3 существующей глобальной x
    y = 4; // присваивается 4 существующей внешней y
    z = 5; // создаётся новая глобальная переменная z и присваивается значение 5.
  } // (Порождает ReferenceError в strict mode(строгом режиме).)

  b(); // вызов b создаёт z как глобальную переменную
  console.log(x, y, z); // 3 4 5
}

a(); // вызов a также вызывает b
console.log(x, z); // 3 5
console.log(typeof y); // undefined, так как y это локальная переменная для функции a
```

Спецификации

Specification
ECMAScript Language Specification # sec-variable-statement

Совместимость с браузерами

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	
var	Chrome 1	Edge 12	Firefox 1	Opera 3	Safari 1	Chrome 18 Android	Firefox 4 for Android	Opera 10.1 Android	Safari 1 on iOS	Sam Int

Tip: you can click/tap on a cell for more information.

Full support

Смотрите также

- let
- const
- How One Missing var Ruined our Launch

This page was last modified on 7 авг. 2023 г. by MDN contributors.