1. **Примитивные типы:**

**ES5:**

В ECMAScript 5 поддерживаются пять [*примитивных*](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) [типов данных](http://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85):

* [числовой](http://ru.wikipedia.org/w/index.php?title=%D0%A7%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D0%BE%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Number*),
* [строковый](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *String*),
* [логический](http://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Boolean*),
* [нулевой](http://ru.wikipedia.org/w/index.php?title=%D0%9D%D1%83%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Null*),
* [неопределённый](http://ru.wikipedia.org/w/index.php?title=%D0%9D%D0%B5%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Undefined*)

**ES6:**

В ECMAScript 6 будут поддерживаются шесть [*примитивных*](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) [типов данных](http://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85):

* [числовой](http://ru.wikipedia.org/w/index.php?title=%D0%A7%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D0%BE%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Number*),
* [строковый](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *String*),
* [логический](http://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Boolean*),
* [нулевой](http://ru.wikipedia.org/w/index.php?title=%D0%9D%D1%83%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Null*),
* [неопределённый](http://ru.wikipedia.org/w/index.php?title=%D0%9D%D0%B5%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Undefined*)
* символьный (англ. *Symbol*)

***Комментарий:***

**typeof null**

//раньше

console.log(typeof null);//=>"object"

//теперь

console.log(typeof null);//=>"null"

Тип Symbol в ES6 является множеством всех значений non – String (не строковых), которые могут быть использованы в качестве ключа свойства объекта.

Каждое из возможных значения Symbol является уникальным и неизменным.

Каждое значение Symbol неизменно (immutable) держит соответствующее значение [[Description]], которое либо **не определено**, либо имеет значение типа String.

Well-known symbols are built-in Symbol values that are explicitly referenced by algorithms of this specification. They are typically used as the keys of properties whose values serve as extension points of a specification algorithm. Unless otherwise specified, well-known symbols values are shared by all Code Realms.

Within this specification a well-known symbol is referred to by using a notation of the form @@name, where “name” is one of the values listed in [Table 1](https://people.mozilla.org/~jorendorff/es6-draft.html#table-1).

Table 1— Well-known Symbols

|  |  |  |
| --- | --- | --- |
| **Specification Name** | **[[Description]]** | **Value and Purpose** |
| @@create | **"Symbol.create"** | A method used to allocate an object. Called from the [[Construct]] internal method. |
| @@hasInstance | **"Symbol.hasInstance"** | A method that determines if a constructor object recognizes an object as one of the constructor’s instances. Called by the semantics of the **instanceof** operator. |
| @@isConcatSpreadable | **"Symbol.isConcatSpreadable"** | A Boolean value that if true indicates that an object should be flatten to its array elements by [Array.prototype.concat](https://people.mozilla.org/~jorendorff/es6-draft.html" \l "sec-array.prototype.concat). |
| @@isRegExp | **"Symbol.isRegExp"** | A Boolean value that if true indicates that an object may be used as a regular expression. |
| @@iterator | **"Symbol.iterator"** | A method that returns the default iterator for an object. Called by the semantics of the for-of statement. |
| @@toPrimitive | **"Symbol.toPrimitive"** | A method that converts an object to a corresponding primitive value. Called by the [ToPrimitive](https://people.mozilla.org/~jorendorff/es6-draft.html" \l "sec-toprimitive) abstract operation. |
| @@toStringTag | **"Symbol.toStringTag"** | A string value that is used in the creation of the default string description of an object. Called by the built-in method[Object.prototype.toString](https://people.mozilla.org/~jorendorff/es6-draft.html#sec-object.prototype.tostring). |
| @@unscopables | **"Symbol.unscopables"** | An Array of string values that are property names that are excluded from the with environment bindings of the associated objects. |

1. **Примитивные объекты:**

**ES5:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Объект** | **Описание** |  |  |
| **Global** | Cоздается исполняющей системой JavaScript перед началом исполнения сценария |  |  |
| **Object** | Прототип остальных объектов |  |  |
| **Number** | Числовые объекты |  |  |
| **Boolean** | Логические объекты |  |  |
| **Function** | Функции |  |  |

**ES6:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Объект** | **Описание** |  |  |
| **Global** | Cоздается исполняющей системой JavaScript перед началом исполнения сценария |  |  |
| **Object** | Прототип остальных объектов |  |  |
| **Number** | Числовые объекты |  |  |
| **Boolean** | Логические объекты |  |  |
| **Symbol** | Символьные объекты |  |  |
| **Function** | Функции |  |  |

***Комментарий:***

**Symbol object -** member of the Object type that is an instance of the standard built-in **Symbol** constructor.

1. **Функции:**

**ES5:**

* Особый объект, который можно вызывать.
* Способы создания:

- function declaration: function myFunc(arg) { ... }

- function expression: myFunc = function(arg) { ... }

- анонимная функция: function(arg){ ... }

* Функции можно вкладывать друг в друга.
* На function declaration воздействует hoisting (подъем) – обработка вне очереди.

**ES6:**

* В новой версии появится ключевое слово let, которое позволит объявлять переменные с блочной областью видимости (block scope) в функциях:

function f(a) {

if (a < 0) {

let i = 3;

}

console.log(i); // ReferenceError: i is not defined }

f(-1)

* В функциях добавилась возможность объявлять у параметров значения по умолчанию:

function setLevel(newLevel = 0)

{

...

}

setLevel(); // newLevel = 0

setLevel(5); // newLevel = 5

setLevel(undefined); // newLevel = 0

* В функциях также появилась возможность указывать именованные параметры:

function foo({ from, to = 10 })

{

...

}

foo({ from: 1, to: 5 });

foo({ to: 5, from: 1 });

foo({ from: 1 });

* Именованные параметры можно комбинировать с обычным (позиционными параметрами):

function foo(positional, { named1, named2 })

{

...

}

foo(123, { named1: 'abc', named2: 'def' })

foo(123, { named2: 'def', named1: 'abc' })

# Стрелочные функции (Arrow functions)

В общем случае, синтаксис стрелочных функций выглядит так:

var fun = (x) => x;

Объявление стрелочной функции, которая принимает один аргумент и просто возвращает его, выглядит очень просто:

var reflect = value => value;

// эквивалент

var reflect = function(value) { return value; }

Но если вы хотите объявить более одного параметра, то должны обрамить список параметров в круглые скобки:

var sum = (num1, num2) => num1 + num2;

// эквивалент

var sum = function(num1, num2) { return num1 + num2; };

Аналогично, функция безо всяких аргументов, должна иметь пустой список параметров, заключённый в круглые скобки:

var sum = () => 1 + 2; // эквивалент var sum = function() { return 1 + 2; };

***Комментарий:***

Стрелочные функции это одно из интереснейших нововведений в ECMAScript 6, которое, имея краткий синтаксис определения, упростит передачу функций в качестве значения параметра другой функции.

1. **Массивы:**

**ES5:**

* «Упорядоченный» набор элементов, поименованных числами (0-based).

Это не отдельный тип, а наследник Object.

* Способы создания:

- литерал: [123, 'hello'] – лучше

- конструктор: new Array(123, 'hello'), можно без new

* Чтение и запись элементов:

- arr[0]

- arr[10] = true

* Чтение несуществующего индекса вернет undeﬁned.

Удалять элемент нужно не оператором delete, а методом splice().

**ES6:**

#### **Set, Map и WeakMap Set** хранит уникальные значения, **Map** и **WeakMap** — ассоциативные массивы, где ключами могут выступать любые объекты и функции.

var st=new Set;

st.add(5);

st.add(5);

console.log(st.has(5));//=>true

st.delete(5);

console.log(st.has(5));//=>false

st.add(console);

console.log(st.has(console));//=>true

var map=new WeakMap;

var obj={foo:"bar"};

map.set(obj,"val");

console.log(map.get(obj));//=>"val"

map.delete(obj);

console.log(map.get(obj));//=>undefined

***Комментарий:***

#### По идее, WeakMap отличается от Map тем, что при удалении всех ссылок на объект, представляющий ключ, удаляется и пара ключ-значение из массива. Однако, в данной реализации V8, Map и WeakMap — одно и то же, т.к. в ней пока не реализован итератор.

1. **Циклы:**

**ES5:**

Цикл for-in в ES5 итерирует по всем полям объекта (включая наследованных). Т.е. итерироваться по значениям массива можно, но опасно:

let arr = [ "blue", "green" ];

arr.notAnIndex = 123;

Array.prototype.protoProp = 456;

for(var x in arr)

{

console.log(x); // Напечатает blue, green, notAnIndex, protoProp

}

**ES6:**

В ECMAScript 6 появится цикл for-of, который решит данную проблему:

for(var x of arr)

{

console.log(x); // Напечатает только blue, green

}

***Комментарий:***

Также, возможно, в язык добавится оператор yield, с помощью которого можно легко и красиво писать кастомные итераторы.

1. **ООП:**

**ES5:**

* Что можно было добиться с помощью прототипов:

// Supertype

function Person(name) {

this.name = name;

}

Person.prototype.describe = function () {

return "Person called " + this.name;

};

// Subtype

function Employee(name, title) {

Person.call(this, name);

this.title = title;

}

Employee.prototype = Object.create(Person.prototype);

Employee.prototype.constructor = Employee;

Employee.prototype.describe = function () {

return Person.prototype.describe.call(this) + " (" + this.title + ")";

};

**ES6:**

* В ECMAScript 6 появятся классы (то, чего можно было добиться с помощью прототипов в ES5, теперь можно будет добиться с помощью классов):

// Supertype

class Person{

constructor(name){

this.name = name;

}

describe() {

return "Person called " + this.name;

}

}

// Subtype

class Employee extends Person {

constructor(name, title) {

super.constructor(name);

this.title = title;

}

describe() {

return super.describe() + " (" + this.title + ")";

}

}

* Теперь можно использовать эти классы:

let jane = new Employee("Jane", "CTO");

jane instanceof Person; // true

jane instanceof Employee; // true

jane.describe(); // 'Person called Jane (CTO)'

* В ES6 появятся модули:

module Math {

export function sum(x, y) {

return x + y;

}

export var pi = 3.141593;

// Не видна снаружи

function internal() {

...

}

}

***Комментарий***:

Как видно, классы в ECMAScript 6 — это просто синтаксический сахар над конструкторами и функциями.

Появились модули, их можно вкладывать друг в друга, можно подгружать из веба или через файловую систему, все глобальные переменные в модуле, являются глобальными переменными только в этом модуле, также возможны циклические зависимости между модулями.

***Список использованных источников:***

<http://kangax.github.io/compat-table/es6/>

<http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts>

<http://www.ecma-international.org/ecma-262/5.1/#sec-4.2.1>

<https://people.mozilla.org/~jorendorff/es6-draft.html#sec-terms-and-definitions-object>

<http://www.2ality.com/>