Веб-разработка
Лекция №4. JavaScript.

Шумилов Вадим Валерьевич, к.т.н.

Тензор, 2017

Особенности

Особенности

Переменные.

- JavaScript – язык с гибкой типизацией
- Для объявления используется директива var
  - Рекомендуется ВСЕГДА использовать при объявлении
- Тип никак явно не указывается

Особенности

Типы данных

```
var a = 1;     // число
var a = ‘a’;   // строка
var a = true; // булевский
var a = null; // специальное значение null
var a = undefined; // еще одно «спецзначение»
```

Особенности

Особенности операторов

Оператор сложения +

- Используется и для конкатенации строк
- Если хотя бы один операнд – строка, второй также будет приведен к строке

```
1 + 2;   // 3, число
‘1’ + 2; // ‘12’, строка
```

Особенности

Особенности операторов

Сравнение

- === проверяет типа операндов
- Остальные сравнения проводят числовую конвертацию типа
  - `false == 0, true > 0`
- Кроме строк – строки сравниваются лексикографически, по unicode-кодам символа
  - `‘a’ > ‘Я’; true`

Особенности

Особенности операторов

Сравнение. Интересное исключение – null и undefined.

- null и undefined равны друг другу, но не равны ничему другому (в т.ч. и 0)
- В других сравнениях:
  - null -> 0
  - undefined -> NaN

Особенности

Особенности операторов

Деление на 0 не вызывает ошибки

```
1 / 0; // Infinity
-1 / 0; // -Infinity
```

Особенности

Особенности операторов

При ошибке вычислений возвращается специальное значение NaN (Not a Number)

```
3 / “2”; // NaN
```

Подробнее о типах данных

Типы

У любого типа (кроме значений null и undefined) есть методы и свойства.
Даже у числа и строки

```
42.3.toFixed(0); // 42
“Hello”.length; // 5
```

Типы

У любого типа (кроме значений null и undefined) есть методы и свойства.
Даже у числа и строки

```
42.toFixed(0); // Error!
(42).toFixed(0); // 42
```

Типы. Числа

Любое число можно преобразовать в любую систему счисления

```
(42).toString(16); // 2a
(424242).toString(32); // c9ui
```

Типы. Строки

Можно писать в любых кавычках.

```
“This is a string”;
‘This is a string’;

“This \”is\” a string”;
```

Типы. Строки

Доступ к элементам строки

- `s[idx]`
- `s.charAt(idx)`

```
""[0]; // undefined
"".charAt(0); // ""
```

Типы. Строки

Доступ к элементам строки

```
“a”.charCodeAt(0); // 97

String.fromCharCode(97); // “a”
```

Типы. Строки

Доступ к элементам строки

- ~~s[idx]~~
- s.charAt(idx)

~~“”[0]; // undefined~~

“”.charAt(0); // “”

Типы. Строки

Все строки хранятся в UTF-8

Типы. Строки

Строки не мутабельные. Нельзя изменить строку.

```
var s = “123”;
s[0] = “a”;
s; // “123”
```

Типы. Объекты

Объекты. По-сути это хэш-таблицы.

```
var o = {}; // можно new Object(); но не нужно
o.key = 10;
```

Типы. Объекты

Могут хранить любые типы

```
var o = {};
o.key = 10;
o.another = “string”;
o.more = {};
```

Типы. Объекты

Ключ – любая строка. Ключи регистронезависимые.

```
var o = {};
o.key = 10;
o.another = “string”;
o.more = {};
o.strange_key = 0;
o.MORE = 100500;
```

Типы. Объекты

К свойствам можно обращаться через переменную

```
var o = {};
var key = “another”;
o[key] = “value”;
```

Типы. Объекты

Можно сразу объявлять с полями

```
var o = {
   key: “value”,
   another: 123,
   obj: {
      x: 10
   }
};
```

Типы. Объекты

Обход объекта

```
var o = { ... };
for (var key in o) {
   console.log(key);
}

// для свежих браузеров...
Object.keys(o); // массив строк-ключей
```

Типы. Объекты

У объекта можно удалить свойство

```
var o = {
   a: 1;
};
o.a; // 1
delete o.a;
o.a; // undefined
```

Типы. Массивы

```
var a = [];
var b = [ 1, 2, 3 ];

a.length; // 0;
b.length; // 3;
```

Типы. Массивы

```
var a = [];
a[0] = 1;
a[1] = 2;
a; // 1,2
```

Типы. Массивы

```
var a = [];
a[0] = 1;
a[1] = 2;
a; // 1,2
a.length; // 2
```

Типы. Массивы

```
var a = [];
a[100] = 1;
a.length; // ???
```

Типы. Массивы

```
var a = [];
a[100] = 1;
a.length; // 101
```

Типы. Массивы

Обход массивов – по индексу от 0 до length;

```
var a = [1, 2, 3];
for (var i = 0, l = a.length; i < l; i++) {
   console.log(a[i]);
}
```

Типы. Массивы

Обход массивов – по индексу от 0 до length;

```
var a = [1, 2, 3];
a[100] = 100;
for (var i = 0, l = a.length; i < l; i++) {
   console.log(a[i]);
}

// 1,2,3,undefined,....,100
```

Типы. Массивы

Массив – тот же объект.

```
var a = [1, 2, 3];
a.x = “y”;
a[“foo”] = “bar”;
```

Типы. Массивы

Длина массива – изменяемое свойство!

```
var a = [1, 2, 3];
a; // 1,2,3
a.length; // 3
a.length = 2;
a; // 1,2
a.length; // 2
```

Типы. Массивы

Создание через new Array();

Выбор автора – НЕ НАДО! Только по особым случаям!

```
new Array(5); // ,,,,
new Array(1,2,3); // 1,2,3
new Array(-1); // ???
```

Типы. Массивы

Создание через new Array();

Выбор автора – НЕ НАДО! Только по особым случаям!

```
new Array(5); // ,,,,
new Array(1,2,3); // 1,2,3
new Array(-1); // ОШИБКА!
```

Типы. Функции

```
function f() { ... }

typeof f; // “function”
f();
f(1, 2, 3);
```

Типы. Функции

```
function f(a) {
   console.log(a);
}

f(); // undefined
f(1); // 1
f(1, 2, 3); // 1
```

Типы. Функции

```
function f() {
   console.log(arguments);
}

f(); // []
f(1); // [1]
f(1, 2, 3); // [1,2,3]
```

Типы. Функции

Можно использовать вместе

```
function f(a) {
   // a === arguments[0];
   if (a) {
      console.log(arguments[1]);
   }
}

f(1, 2); // 2;
```

Типы. Функции

```
for (var i = 0; i < arguments.length; i++) {
    console.log(arguments[i]);
}
```

Типы. Функции

Но это не настоящий массив!!!

```
arguments.push(10); // ERROR!
```

Типы. Функции

Можно узнать, со сколькими аргументами был объявлена функция.

```
function tellMe(f) {
   console.log(f.length);
}

tellMe(function(a,b,c){}); // 3;
tellMe(function(){}); // 0
```

Типы. Функции

Кстати, на функцию тоже можно повесить свойства!

```
function f() { ... }

f.prop = 1;
f.prop; // 1
```

Области видимости и замыкания

Область видимости

Где доступна переменная?

```
function test() {
    for(var i = 0; i < 10; i++) {
        var j = i + 1;
        doWork(i, j);
    }
    console.log(i, j); // ?, ?
}
```

Область видимости

Hoisting. «Всплытие» переменных.

```
function test() {
    var i, j;
    for(i = 0; i < 10; i++) {
        j = i + 1;
        doWork(i, j);
    }
    console.log(i, j); // ?, ?
}
```

Область видимости

Hoisting. «Всплытие» переменных.

```
function test() {
    var i, j;
    for(i = 0; i < 10; i++) {
        j = i + 1;
        doWork(i, j);
    }
    console.log(i, j); // 10, 10
}
```

Область видимости

```
function test() {
    var i;
    for(i = 0; i < 10; i++) {
        doWork(i);
    }
    return function () {
        console.log(i);
    }
}
```

Область видимости

```
var f = test();
typeof f; // “function”
f(); // ???
```

Область видимости

```
function test() {
    var i;
    for(i = 0; i < 10; i++) {
        doWork(i);
    }
    return function () {
        console.log(i);
    }
}
```

Область видимости

```
function test() {
    var i;
    for(i = 0; i < 10; i++) {
        doWork(i);
    }
    return function () {
        console.log(i);
    }
}
```

Область видимости


```
var f = test();
typeof f; // “function”
f(); // 10
```

Область видимости

- Область видимости переменной ограничивается функцией
- Определение переменной «всплывает» к «верхней точке» области видимости
- Переменную видно в ее «родной» области видимости и всех «нижележащих»
- При обращении к переменной она ищется в текущей области видимости, потом выше и т.д.
- Всегда текущее значение переменной

Область видимости

- На «самом верху» – глобальная область видимости, «глобальный объект».
  - В браузере это объект window

Область видимости

```
var greeting = “Hello”;

function greet(name) {
    console.log(greeting + “, “ + name + “!”);
}

greet(“John”); // Hello, John!
greeting = “Привет”;
greet(“Вася”); // Привет, Вася!
```

Контекст вызова

Контекст исполнения

- **Любая функция** при вызове получает контекст исполнения
- Контекст исполнения доступен внутри функции через ключевое слово **this**
- Контекст определяется **в момент вызова** функции

Контекст исполнения

- При вызове простой функции, контекстом будет глобальный объект
- При вызове «от объекта» – этот объект

Контекст исполнения


```
function x() {
      console.log(this);
}

x(); // глобальный объект
```

Контекст исполнения

```
var o = {
    foo: 1,
    bar: function() {
        console.log(this.foo);
    }
};

o.bar(); // 1
```

Контекст исполнения

```
var o = {
    foo: 1,
    bar: {
        baz: function() {
            console.log(this.foo);
        }
    }
};

o.bar.baz(); // ??
```

Контекст исполнения

```
var o = {
    foo: 1,
    bar: {
        baz: function() {
            console.log(this.foo);
        }
    }
};

o.bar.baz(); // undefined
```

Контекст исполнения

```
var o = {
    foo: 1,
    bar: function() {
        console.log(this.foo);
    }
};

var f = o.bar;
f(); // ??
```

Контекст исполнения

```
var o = {
    foo: 1,
    bar: function() {
        console.log(this.foo);
    }
};

var f = o.bar;
f(); // undefined
```

Контекст исполнения

- Контекст можно задать явно
  - call
  - apply
- Позволяют вызвать функцию в заданном контексте с заданными аргументами

Контекст исполнения

```
function f(a, b) {
    console.log(this, a, b);
}

f(1, 2); // глобальный объект, 1, 2
f.call(10, -1, -2); // 10, -1, -2
f.apply(“str”, [ 1, 2 ]); // “str”, 1, 2
```

Контекст исполнения

```
// для вызова в контексте глобального объекта
f.apply(null, [ 10, 20 ]);
```

Внутренности. Event Loop.

Event Loop

Event Loop – он же «цикл событий»...

Event Loop

JavaScript – однопоточный. В нем не бывает несколько конкурирующих задач*.

Event Loop

JavaScript – однопоточный. В нем не бывает несколько конкурирующих задач*.

* Есть WebWorkers, но все равно **конкурирующих** не бывает.

Event Loop

JavaScript – однопоточный. В нем не бывает несколько конкурирующих задач*.

Нельзя приостановить выполнение функции.

Event Loop

Можем отложить выполнение кода

```
// выполнить один раз
setTimeout(function() { ... }, 100);

// выполнять постоянно
setInterval(function() { ... }, 10);
```

Event Loop

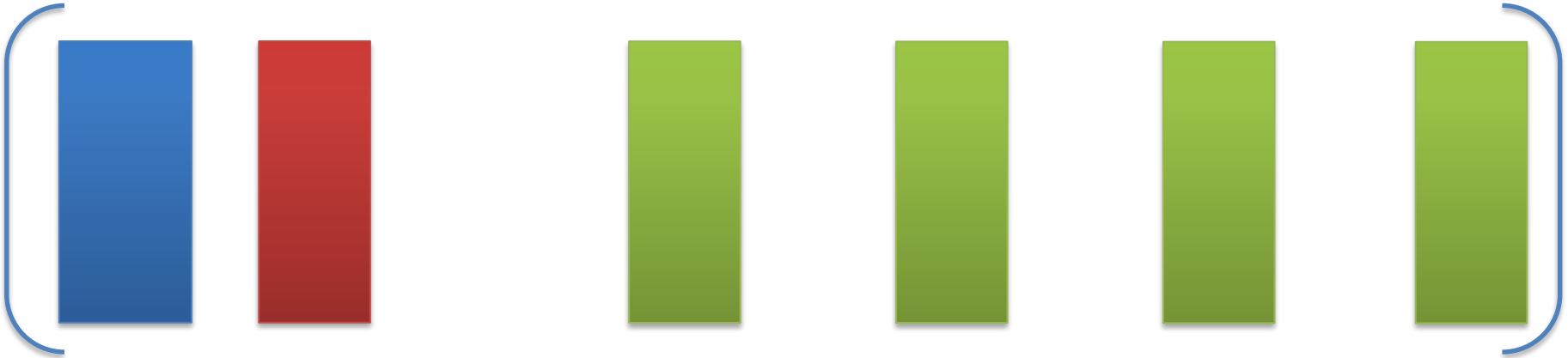
```
setInterval(function() {
   console.log(“.”);
}, 1000);

setTimeout(function() {
  while (true) {
      // бесполезный бесконечный цикл
  }
  }, 10);
  // сколько точек увидим через минуту?
```

Event Loop

Правильный ответ – ни одной

Event Loop



Основной код

setTimeout

setInterval

Event Loop

Основной код

setTimeout

setInterval

Внутренности. GC

Garbage Collector

Управление памятью или сборка мусора

Garbage Collector

- Управление памятью – автоматическое
- Не нужно явно заботиться о выделении или освобождении памяти
- Память «чистит» сборщик мусора (он же garbage collector, он же GC)

Garbage Collector

- Основывается на понятии «достижимости» объекта в памяти
- Существуют объекты, достижимые по умолчанию
  - Все глобальные переменные
  - То, что доступно «со стэка», то есть те функции и переменные в них, которые сейчас выполняются или ожидают окончания выполнения других
- Все остальные могут быть доступны по «ссылкам»
- А могут не быть

Garbage Collector

Алгоритм работы (очень упрощенно)

На входе:

1. Все имеющиеся объекты
2. Набор «корней»

Garbage Collector

Алгоритм работы (очень упрощенно)

1. От каждого корня пройти по ссылкам вглубь до конца
2. Каждый объект который встретим на пути – пометим
3. Пройдем по всем объектам и удалим те, что остались без пометки

Garbage Collector

Последствия:

1. Когда GC работает – все останавливается
2. Чем больше памяти выделено, тем дольше идет сборка мусора*

* В современных движках все несколько сложнее и п.2 не всегда верен

Garbage Collector

```
var o = {
    prop: function() { ... }
}
```

Global -> o -> o.prop -> function

Garbage Collector

```
o.prop = 10;
```

Global -> o -> o.prop function

\

` -> 10

Garbage Collector

```
o.prop = 10;
```

Global -> o -> o.prop function

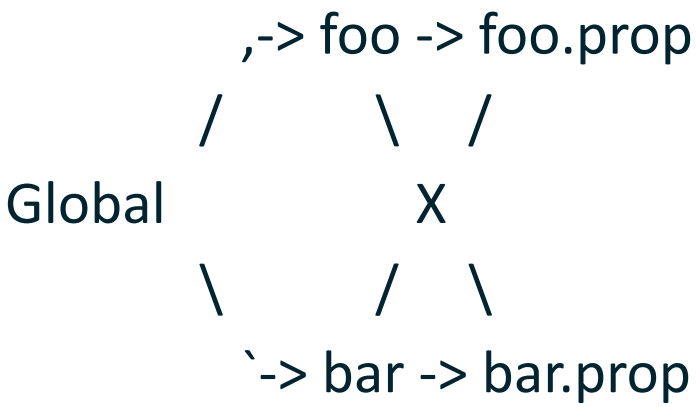
\

` -> 10

Garbage Collector

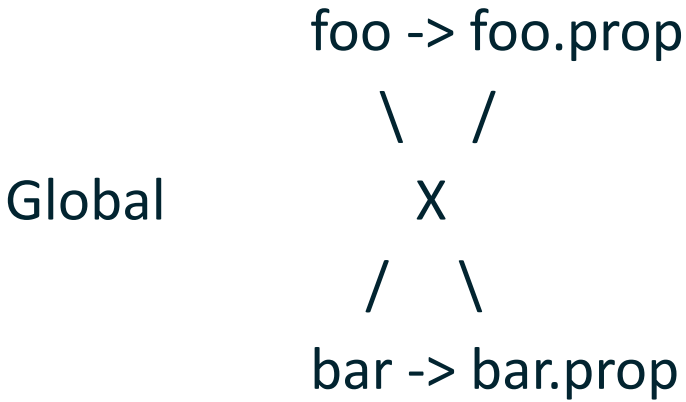
```
var foo = {}, bar = {};

foo.prop = bar;
bar.prop = foo;
```



Garbage Collector

```
foo = null;
bar = null;
```



Garbage Collector

ВАЖНО!

Функции имеют ссылки на (почти*) все, что они видят через замыкание.

*Современные движки оптимизируют и не держат ссылки на то, что функция видит но не использует.

Garbage Collector

```
function f() {
    var i = 0, j = 1;

    // Видит: i, j. Использует: i
    function g() {
        console.log(i);
    }
}
```

Полезные ссылки

Учебник

- http://javascript.ru/
- http://learn.javascript.ru/

Вопросы есть?

Спасибо за внимание!