Яндекс

Дополнительные главы JavaScript

Сергей Тощилин, формошлёп

План лекции

01 | Строгий режим

02 | eval

03 | BigInt

04 | Разделители разрядов

05 | flat(), flatMap()

06 | WeakMap, WeakSet

07 | Генераторы, Итераторы

08 | Модули

09 | Console

Строгий режим

Строгий режим

Позволяет использовать более строгий вариант JavaScript, чтобы создать современную и более здоровую экосистему

```
'use strict';

const array = [1, 2, 3, 4];

/* Какой-то код */
```

Описание строгого режима

Строгий режим

В строгом режиме будет выброшена SyntaxError, если используется:

- Восьмеричная запись числа
- Оператор `with`
- Оператор `delete` для удаления переменных
- Ключевые слова `eval` и `arguments` в параметрах функции
- В качестве идентификатора одно из следующих слов: `implements`, `interface`, `let`, `package`, `private`, `protected`, `public`, `static`, `yield`
- Повторяющийся идентификатор в полях объекта или параметрах функции

Строгий режим

Когда код интерпретируется в строгом режиме?

- В глобальной области видимости с того момента, когда была объявлена директива `'use strict';`
- Всегда в EcmaScript модулях
- Всегда в декларации классов
- Код внутри eval(), если внутри него есть директива `'use strict'` или если сам eval() был позван из строгого режима
- Если код функции использует строгий режим или код в области видимости, в которой была объявлена функция, использует строгий режим

ES10: Strict Mode Code

eval()

Определение

Встроенная в JavaScript функция, которая позволяет выполнить строку кода.

```
const code = 'arr = [1, 2, 3];';

eval(code); // arr is now [1, 2, 3]

console.log(arr); // [1, 2, 3]
```

Область видимости eval

Встроенная в JavaScript функция, которая позволяет выполнить строку с кодом.

```
const code = 'const arr = [1, 2, 3];';

eval(code);

console.log(arr); // ReferenceError: arr is not defined
```

> **В строгом режиме eval() создает лексическую область видимости**

P.S. const автоматически переключает интерпретатор в строгий режим

Что вернет?

Возвращает результат последней инструкции

```
const code = 'arr = [1, 2, 3]; 5;';

const res = eval(code);

console.log(res); // 5
```

Achtung!

Имеет доступ к локальным переменным

```
function doAnything = (arr) => {
    console.log(arr); // Массив полезных данных
    const code = 'arr = [1, 2, 3];';
    eval(code);
    console.log(arr); // [1, 2, 3]
}
```

> **Можно, но не нужно!**

BigInt

Проблема

JavaScript допускает потерю точности для числовых значений больше 2^{53}.

```
2 ** 1023 // 8.98846567431158e+307
2 ** 1024 // Infinity

Number.MAX_SAFE_INTEGER     // 9007199254740991
Number.MAX_SAFE_INTEGER + 1 // 9007199254740992
Number.MAX_SAFE_INTEGER + 2 // 9007199254740992
Number.MAX_SAFE_INTEGER + 3 // 9007199254740994
Number.MAX_SAFE_INTEGER + 4 // 9007199254740996
```

Определение

Встроенный в JavaScript объект, который позволяет представлять числовые значение больше 2^{53}.

```
BigInt(2 ** 1023) // 898846567431157953864652595394512...08n

BigInt(2 ** 1024) // RangeError
BigInt(Infinity)  // RangeError

BigInt(2) ** BigInt(1023)  // 898846567431157953864652...08n

BigInt(2) ** BigInt(1024) // 179769313486231590772930 5...16n
```

```
(2 ** 1023)n // SyntaxError

2 ** 1023n   // TypeError

2n ** 1023n  // 89884656743115795386465259539451...08n


BigInt(Number.MAX_SAFE_INTEGER + 2)  // 9007199254740992n

BigInt(Number.MAX_SAFE_INTEGER) + 2n // 9007199254740993n
```

Конструктор BigInt

```
new Number(1) // Number {1}
new BigInt(1) // TypeError: BigInt is not a constructor

BigInt(1)     // 1n
BigInt('1')   // 1n
BigInt(0.5)   // RangeError
BigInt('0.5') // SyntaxError

Object(123)                 // Number {123}
typeof Object(123)          // "object"
Object(BigInt(123))         // BigInt {123n}
typeof Object(BigInt(123)) // "object"
```

BigInt – новый примитив

Можно позвать `typeof`, можно преобразовывать

```
typeof BigInt(1) // "bigint"

Boolean(BigInt(1)) // true
Boolean(BigInt(0)) // false

BigInt(1347).toString() // '1347'
Number(BigInt(1347)) // 1347
```

BigInt – новый примитив

Можно сравнивать с другими типами. Нельзя совершать операции с другими типами.

```
1 <   BigInt(2) // true

1 <   BigInt(1) // false
1 >   BigInt(1) // false
1 >=  BigInt(1) // true
1 <=  BigInt(1) // true
1 === BigInt(1) // false
1 ==  BigInt(1) // true
```

BigInt.asIntN(width, value)

```
BigInt.asIntN(32, 1347n) // 1347n

BigInt.asIntN(32, 2n ** 30n)                          // 1073741824n

BigInt.asIntN(32, (2n ** 30n - 1n) * 2n + 1n) // 2147483647n
BigInt.asIntN(32, (2n ** 30n - 1n) * 2n + 2n) // -2147483648n
BigInt.asIntN(32, 2n ** 31n)                          // -2147483648n

BigInt.asIntN(32, 2n ** 32n)                          // 0n

BigInt.asIntN(64, 2n ** 32n)                          // 4294967296n
```

BigInt.asUintN(width, value)

```
BigInt.asUintN(32, 2n ** 30n)                          // 1073741824n

BigInt.asUintN(32, (2n ** 30n - 1n) * 2n + 1n) // 2147483647n
BigInt.asUintN(32, (2n ** 30n - 1n) * 2n + 2n) // 2147483648n
BigInt.asUintN(32, 2n ** 31n)                          // 2147483648n

BigInt.asUintN(32, (2n ** 31n - 1n) * 2n + 1n) // 4294967295n
BigInt.asUintN(32, (2n ** 31n - 1n) * 2n + 2n) // 0n
BigInt.asUintN(32, 2n ** 32n)                          // 0n

BigInt.asUintN(64, 2n ** 32n)                          // 4294967296n
```

Разделители разрядов

Numeric Separators

```
1000000000          // Это миллиард? 100 миллионов? 10 миллиардов?
101475938.38

1_000_000_000       // Это миллиард
101_475_938.38      // 100 миллионов

0.000_001           // Одна миллионная
0xFF_00_FF          // 16-ричные числа удобно смотреть по 2 разряда
0b1001_0011         // 2-ичные числа удобно смотреть по 4 разряда
```

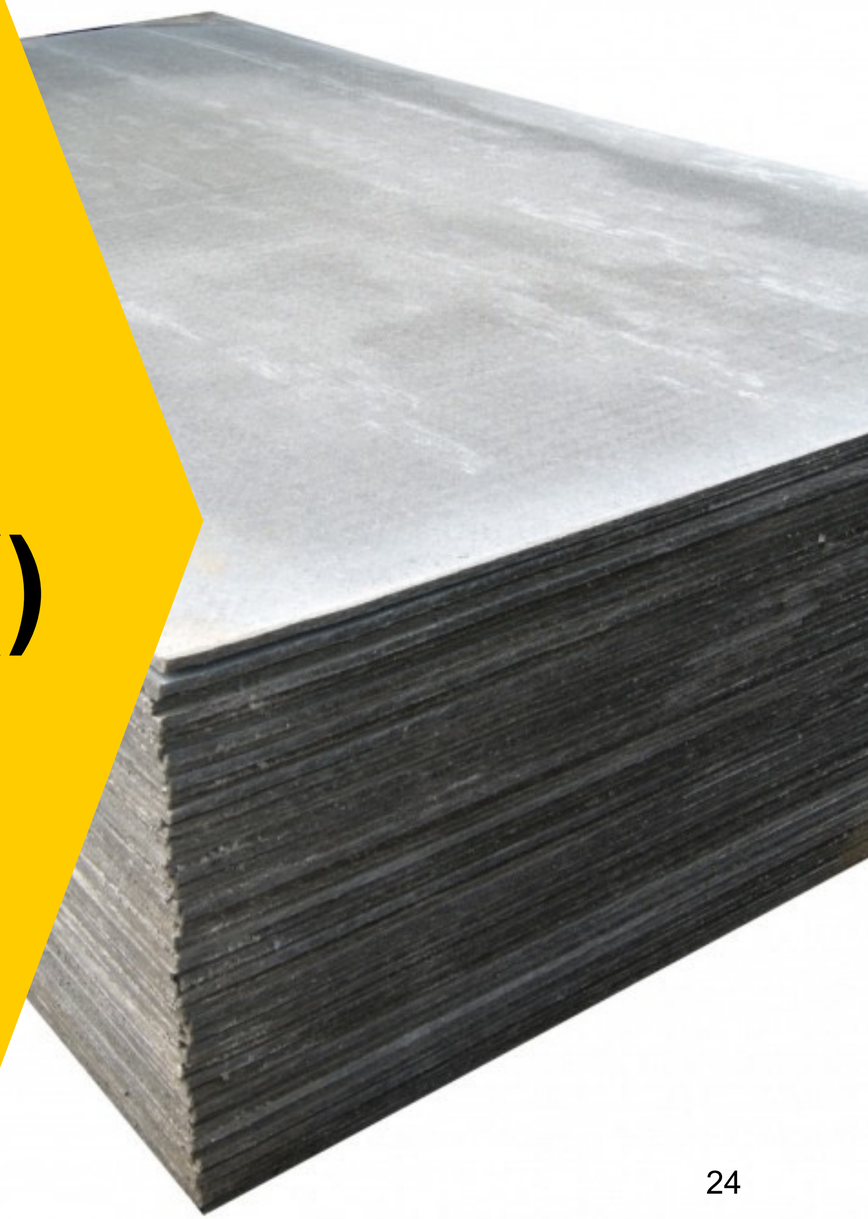
Поддержка

TC39: Stage 3

- V8: 7.5
- Chrome: 75
- Node.js: 12.5

[Предложение в репозитории TC39]

Array.prototype.flat(), Array.prototype.flatMap()



Array.prototype.flat()

```
const array = [1, [2, [3, [4, 5]]]];

array.flat();               // [1, 2, Array(2)]

array.flat(2);              // [1, 2, 3, Array(2)]

array.flat(Infinity); // [1, 2, 3, 4, 5]
```

Array.prototype.flatMap()

```
const array = [1, 2, 3, 4];

array.map(n => [n, n ** 2]);
// [Array(2), Array(2), Array(2), Array(2)]

array.map(n => [n, n ** 2]).flat();
// [1, 1, 2, 4, 3, 9, 4, 16]
```

Array.prototype.flatMap()

```
const array = [1, 2, 3, 4];

array.flatMap(n => [n, n ** 2]);
// [1, 1, 2, 4, 3, 9, 4, 16]
```

Array.prototype.flatMap()

```
const array2 = [1, [2, [3, [4, 5]]]];

array2.flatMap(n => [n, n ** 2]);
// [1, 2, Array(2)]
```

Array.prototype.flatMap(): зачем?

```
const arr = [
    'Строгий режим',
    'eval',
    'BigInt',
    'Разделители разрядов',
    'Array.flat(), Array.flatMap()',
    'WeakMap, WeakSet',
    'Генераторы, Итераторы',
    'Модули',
    'Console'
];

arr.flatMap(topic => topic.split(/\.|\s/));

/* [ 'Строгий', 'режим', 'eval', 'BigInt', 'Разделители', 'разрядов',
     'Array', 'flat(),', 'Array', 'flatMap()', 'WeakMap,', 'WeakSet',
     'Генераторы,', 'Итераторы', 'Модули', 'Console' ] */
```

Поддержка

TC39: Stage 4

- Chrome: 69
- Firefox: 62
- Node.js: 11

Предложение в репозитории TC39

WeakMap, WeakSet

Проблема

```
let obj = { anykey: 1 };
// { anykey: 1 } теперь хранится в памяти

obj = undefined;
// { anykey: 1 } недостижим,
// сборщик мусора удалит его из памяти
```

Проблема

```
let obj = { a: 2 };
// { a: 2 } теперь хранится в памяти

let arr = [1, obj, 3];

obj = undefined;
// { a: 2 } останется в памяти, ибо есть ссылка из arr

arr[1]; // { a: 2 }
arr;    // [1, {…}, 3]
```

Проблема

```
let obj = { a: 2 };
// { a: 2 } теперь хранится в памяти

let map = new Map([1, obj, 3].map((q, i) => [q, i]));

obj = undefined;
// { a: 2 } останется в памяти, ибо есть ссылка из map

map.keys(); // MapIterator {1, {…}, 3}
```

WeakMap, WeakSet

Стандартные встроенные объекты JavaScript

- WeakMap – коллекция пар ключ-значение. В отличие от Map, в качестве ключей могут использоваться только объекты.
- WeakSet – коллекция, элементами которой являются только объекты.

> **WeakMap и WeakSet не дают способов получить все их ключи или текущий размер. Свойства keys(), values(), entries(), size() не поддерживаются.**

WeakMap

```
// Общий синтаксис: new WeakMap([iterable]);

const weakMap =  new WeakMap();

const obj = { a: 5 };
const func = () => 5;

weakMap.set(obj, 'anything1');
weakMap.set(func, 'anything2');
weakMap.set({ a: 5 }, 'anything3');

weakMap.get(obj);       // 'anything1'
weakMap.get({ a: 5 }); // ???
```

WeakMap

```
// Общий синтаксис: new WeakMap([iterable]);

const weakMap =  new WeakMap();

const obj = { a: 5 };
const func = () => 5;

weakMap.set(obj, 'anything1');
weakMap.set(func, 'anything2');
weakMap.set({ a: 5 }, 'anything3');

weakMap.get(obj);         // 'anything1'
weakMap.get({ a: 5 }); // undefined
```

WeakMap решает проблему с висячей ссылкой

```
let obj = { a: 2 }

let map = new WeakMap(
   [new Number(1), obj, new Number(3)]
      .map((elem, i) => [elem, i])
);

console.log(map) // WeakMap {{…} => 1}

obj = undefined

console.log(map) // WeakMap {}
```

```
> const start = Date.now(); const time = () => console.log(Date.now() - start);
  let obj = { a: 2 }
< undefined
> time()
  let map = new WeakMap([new Number(1), obj, new Number(3)].map((elem, i) => [elem, i]));
  2867
< undefined
> time(); map;
  5239
< ▶ WeakMap {Number => 0, Number => 2, {…} => 1}
> time(); obj = undefined;
  9012
< undefined
> time(); map;
  25996
< ▶ WeakMap {}
```

```
> const start = Date.now(); const time = () => console.log(Date.now() - start);
  let obj = { a: 2 }
<· undefined
> time()
  let map = new WeakMap([new Number(1), obj, new Number(3)].map((elem, i) => [elem, i]));
  925
<· undefined
> time(); map;
  3217
<· ▶ WeakMap {Number => 0, {…} => 1, Number => 2}
> time(); map;
  12123
<· ▶ WeakMap {Number => 0, {…} => 1, Number => 2}
> time(); map;
  23131
<· ▶ WeakMap {{…} => 1}
```

WeakSet

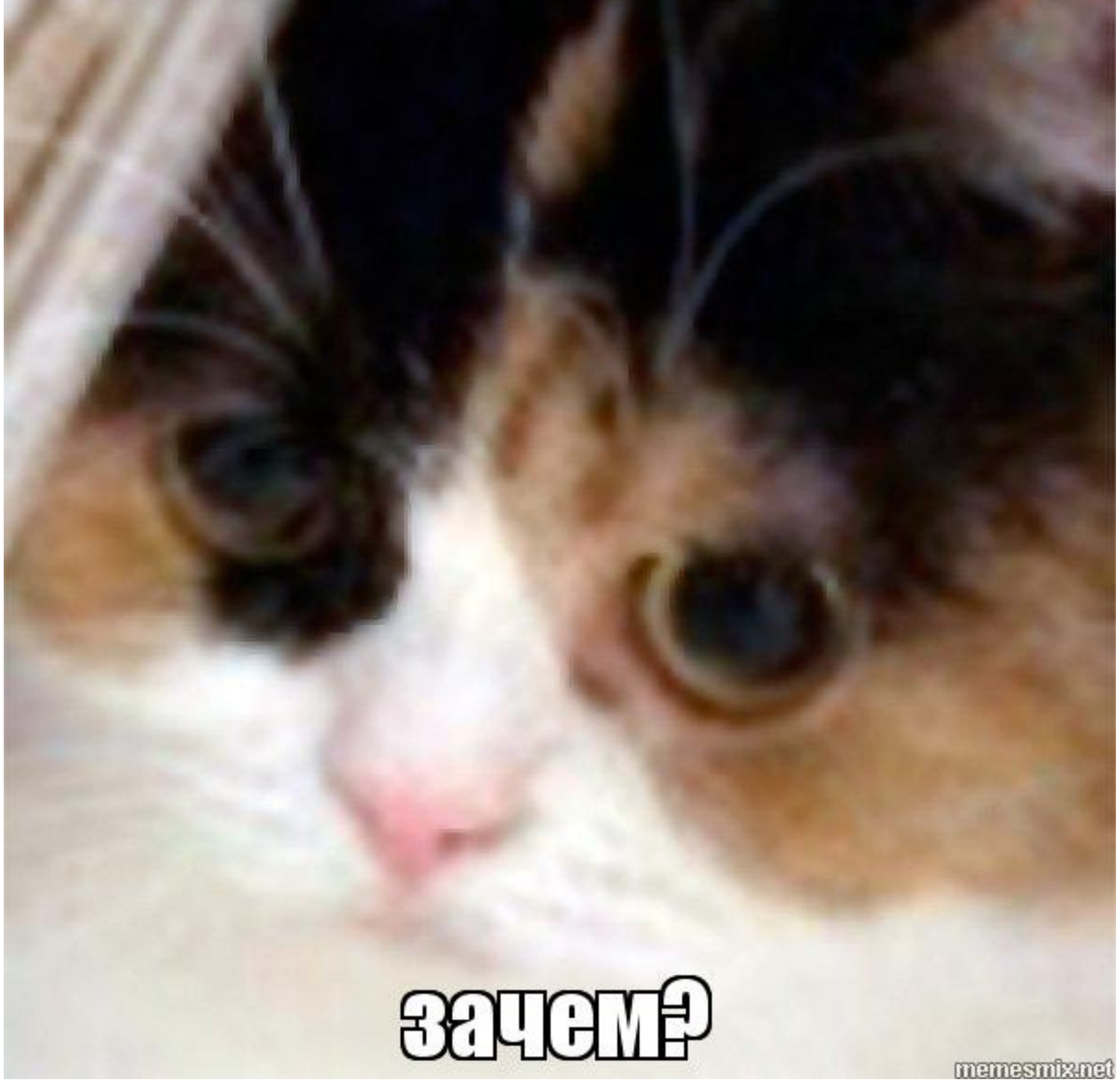
```
// Общий синтаксис: new WeakSet([iterable]);

const weakSet =  new WeakSet();

const obj = {};

weakSet.add(obj);
weakSet.add({});

weakSet.has(obj);         // true
weakSet.has({});          // false
```


ЗАЧЕМ?
memesmix.net


затем

WeakMap, WeakSet: применение

- Можно использовать в качестве привязанного хранилища (например, хранить данные в памяти сервера, пока соединение с пользователем не разорвется)
- Можно использовать для автоматической очистки кэша, взяв в качестве ключа значимую часть конфигурации запроса

Генераторы, итераторы

Лирика: системные символы

Символ (Symbol) – примитивный тип данных, использующийся для создания уникальных идентификаторов.

- Создаются с помощью функции `Symbol()`
- Существует множество системных символов, используемых внутри JavaScript, доступных как `Symbol.*`
- Системные символы предназначены для того, чтобы делать... всякие штуки
- Для понимания итераторов нам понадобится системный `Symbol.iterator`

Лирика: объекты

```
const obj = {
    a: 1,
    b: 2,
    ['c' + 'd']: 3,
    'kebab-str': 4
}

obj.a                   // 1
obj['a']                // 1
obj.cd                  // 3
obj['cd']               // 3
obj['c' + 'd']          // 3
obj.kebab-str           // ReferenceError
obj.'kebab-str'         // SyntaxError
obj.('kebab-str')       // SyntaxError
obj['kebab-str']        // 4
```

Лирика: объекты

```
const obj = {
    [newSymbol]: 1,
    [newSymbol2]: 2,
    [Symbol('key')]: 3,
    [Symbol('anykey')]: 4
}

obj.newSymbol // undefined
obj[newSymbol]// 1
obj[Symbol('key')] // undefined
obj[Symbol('anykey')] // undefined
```

Мы никогда не сможем получить доступ до значения,
если не сохранили ссылку на его символьный ключ

Итераторы

Javascript-объект с настраиваемым протоколом перебора.

> **Протокол перебора используется, например, в конструкции `for ... of ...` или в spread operator.**

- `Array`, `Map` и другие имеют встроенные протоколы перебора
- `Object` не имеет встроенного протокола
- Чтобы объект стал итерируемым, нужно в нем или в прототипе реализовать свойство `Symbol.iterator`
- `Symbol.iterator` должен иметь метод `next()`, который вернет объект со свойствами `done()` и `value`

Итераторы: пример использования

```
const str = 'any string';

const iterator = str[Symbol.iterator]();

> iterator.next()
< {value: "a", done: false}
> iterator.next()
< {value: "n", done: false}
...
> iterator.next()
< {value: "g", done: false}
> iterator.next()
< {value: undefined, done: true}
```

Итераторы можно создать или переопределить существующие

```
const str = 'any string';


str[Symbol.iterator] = function() {
    let index = 0;
    const max = this.length;
    return {
        next: function () {
            return index === max ?
                { done: true } :
                { value: index++, done: false }
        }
    }
};
```

Ничего не выйдет!

```
const str = 'any string';
// typeof str === 'string'

str[Symbol.iterator] = function() {
    let index = 0;
    const max = this.length;
    return {
        next: function () {
            return index === max ?
                { done: true } :
                { value: index++, done: false }
        }
    }
};
```

Итераторы можно создать или переопределить существующие

```
const str = new String('any string');
// typeof str === 'object'

str[Symbol.iterator] = function() {
    let index = 0;
    const max = this.length;
    return {
        next: function () {
            return index === max ?
                { done: true } :
                { value: index++, done: false }
        }
    }
};
```

Итераторы можно создать или переопределить существующие

```
const iterator = str[Symbol.iterator]();

> iterator.next()
< {value: 0, done: false}
> iterator.next()
< {value: 1, done: false}
...
> iterator.next()
< {value: 9, done: false}
> iterator.next()
< {done: true}
```

Генераторы

```
function* generator() {
    yield 1;
    yield 2;
    yield 3;
}

const gen = generator();

gen.next() // {value: 1, done: false}
gen.next() // {value: 2, done: false}
gen.next() // {value: 3, done: false}
gen.next() // {value: undefined, done: true}
```

Генераторы

Функции с возможностью выхода и повторного входа с сохранением контекста.

- При вызове возвращает итератор
- Такой итератор выполняется до первого встреченного `yield`
- `yield` возвращает значение или поставляет новый генератор с помощью `yield*`
- Вызов метода `next()` с аргументом прекращает выполнение функции-генератора, и заменяет инструкцию `yield` на которой было приостановлено выполнение на аргумент переданный в `next()`.

```
function* anotherGenerator(i) {
    yield i + 1;
    yield i + 2;
    yield i + 3;
}

function* generator(i) {
    yield i;
    yield* anotherGenerator(i);
    yield i + 10;
}

const gen = generator(10);

gen.next().value; // 10
gen.next().value; // 11
gen.next().value; // 12
gen.next().value; // 13
gen.next().value; // 20
```

Итераторы можно создать или переопределить существующие

```
const str = new String('any string');

str[Symbol.iterator] = function* () {
    yield 1;
    yield 2;
    yield 'вжух';
};
```

Итераторы можно создать или переопределить существующие

```
const iterator = str[Symbol.iterator]();

> iterator.next()
< {value: 1, done: false}
> iterator.next()
< {value: 2, done: false}
> iterator.next()
< {value: 'вжух', done: false}
> iterator.next()
< {value: undefined, done: true}
```

Модули

Модули

Чтобы упростить разработку и тестирование приложения, его код разделят на небольшие изолированные модули

- Модуль – отдельный файл с кодом на JavaScript
- Добавляет возможность экспортировать функциональность из одного модуля и импортировать её в другом

Модули

Чтобы упростить разработку и тестирование приложения, его код разделят на небольшие изолированные модули

- AMD – одна из самых старых модульных систем, изначально реализована библиотекой require.js.
- CommonJS – модульная система, созданная для сервера Node.js.
- UMD – ещё одна модульная система, предлагается как универсальная, совместима с AMD и CommonJS.

В ES2015 появилась стандартная система модулей, которая на данный момент поддерживается большинством браузеров и с недавнего времени Node.js

ES Модули: использование

Модули могут загружать друг друга и использовать директивы `export` и `import`:

```
// greeting.js
export function greet(user) {
    console.log(`Hello, ${user}!`);
}

// index.js
import { greet } from './greeting.js';

greet('Serezha') // Hello, Serezha!
```

ES Модули: использование

Можно сделать экспорт по умолчанию

```
// greeting.js
export default function greet(user) {
    console.log(`Hello, ${user}!`);
}

// index.js
import greet from './greeting.js';

greet('Serezha') // Hello, Serezha!
```

ES Модули: использование

Экспорту по умолчанию можно дать любое имя

```
// greeting.js
export default function greet(user) {
    console.log(`Hello, ${user}!`);
}

// index.js
import benzopila from './greeting.js';

benzopila('Serezha') // Hello, Serezha!
```

ES Модули: особенности

- Всегда `'use strict'`
- Отдельная область видимости
- Код внутри модуля выполняется единожды при импорте

Еще особенности, `import * as module`, алиасы для импортов

Модули: Node.js

В Node.js зачастую до сих пор используются CommonJS модули

- Node.js для каждого модуля делает доступным объект module, который описывает модуль
- Из модуля экпортируется значение помещённое в поле module.exports
- По умолчанию exports хранит пустой объект
- Node.js содержит встроенные модули ('fs', 'util' и другие)
- Node.js поддерживает импорт из файлов разных типов – json, js, mjs.

Модули: Node.js

```
// file: average.js

const average = (...nums) =>
   nums.reduce((acc, num) => acc + num, 0) / nums.length;

module.exports.average = average;


// index.js
const { average } = require('./average');

average(1, 2, 3, 4) // 2.5
```

ECMAScript модули начали работать в Node.js 13.2 без флага и будут совсем стабильными в Node.js 14 LTS

Console

Console

Console – глобальный объект

- Можно `console.log()`
- Можно `window.console.log() // Браузер`
- Можно `global.console.log() // Node.js`

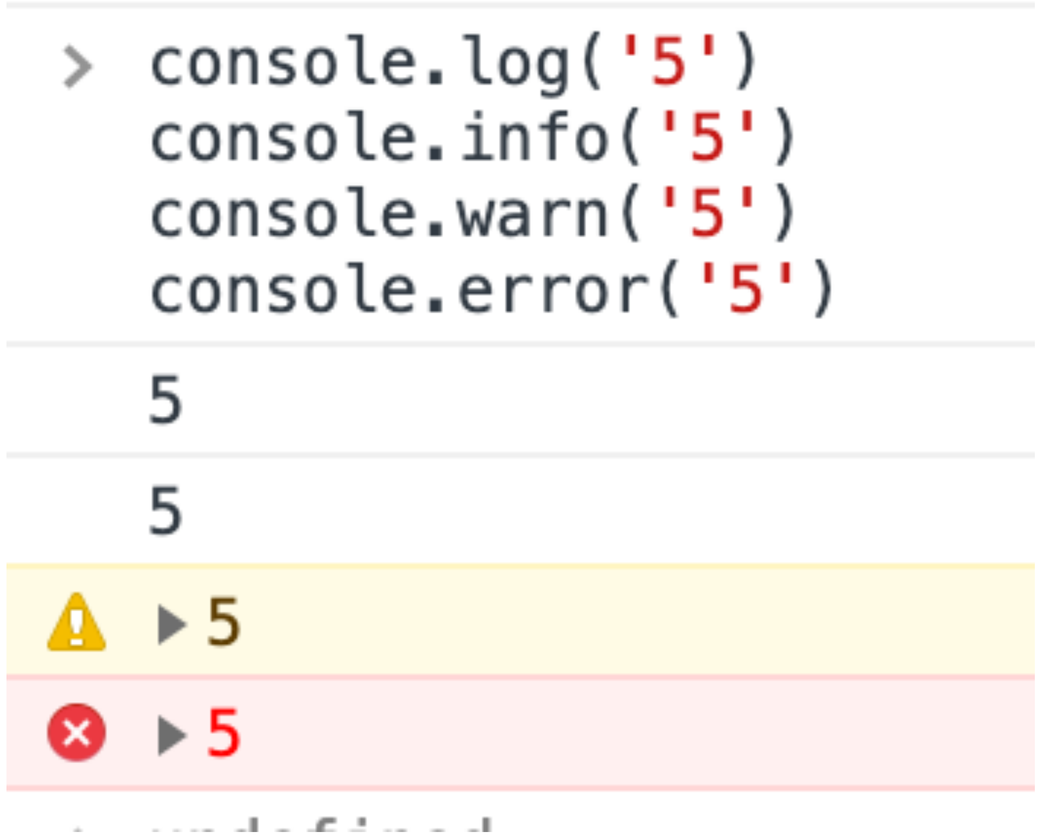
Зачем?

- Можно быстро проверить значение переменных в любом куске кода
- Можно поймать конкретный вызов функции
- Можно впихнуть во всех файлах и не сойти с ума

```
> console.log(console)
▼ console {debug: f, error: f, info: f, log: f, warn: f, …}
  ▶ assert: f assert()
  ▶ clear: f clear()
  ▶ context: f context()
  ▶ count: f count()
  ▶ countReset: f countReset()
  ▶ debug: f debug()
  ▶ dir: f dir()
  ▶ dirxml: f dirxml()
  ▶ error: f error()
  ▶ group: f group()
  ▶ groupCollapsed: f groupCollapsed()
  ▶ groupEnd: f groupEnd()
  ▶ info: f info()
  ▶ log: f log()
    memory: (...)
  ▶ profile: f profile()
  ▶ profileEnd: f profileEnd()
  ▶ table: f table()
  ▶ time: f time()
  ▶ timeEnd: f timeEnd()
  ▶ timeLog: f timeLog()
  ▶ timeStamp: f timeStamp()
  ▶ trace: f trace()
  ▶ warn: f warn()
```

Console: уровни логирования

Выводим обычное значение



Console: рецепты

Выводим сразу красиво

```
> const width = window.innerWidth;
  const height = window.innerHeight;
<· undefined
> console.log('qwrqrew', width, height);
  qwrqrew 1440 150
<· undefined
> console.log({width, height})
  ▶ {width: 1440, height: 150}
<· undefined
>
```

console.table()

Выводим красиво массив объектов

```
> const sections = [1, 2, 3, 4, 5].map((elem, i) => ({ a: elem, b:
  Math.random(), c: 'any string', i }));
<- undefined
> console.log(sections)
  ▶ (5) [{…}, {…}, {…}, {…}, {…}]                    VM3551:1
<- undefined
> console.table(sections)
                                                     VM3598:1
```

(index)	a	b	c	i
0	1	0.9484314810…	"any string"	0
1	2	0.0828825131…	"any string"	1
2	3	0.9179405034…	"any string"	2
3	4	0.1139401061…	"any string"	3
4	5	0.4224223624…	"any string"	4

▶ Array(5)

console.assert()

Условный вывод

```
> document.getElementsByTagName('div').length
< 762
> function forEachElements() {
      [...document.getElementsByTagName('div')].map((elem, i) => {
          if (i === 348) {
              console.log(elem);
          }
      })
  }
< undefined
> forEachElements()
                                                          VM5807:4
  ▶ <div class="doc-c-menu__item doc-c-menu__item_type_submenu">…</div>
< undefined
```

console.assert()

Условный вывод

```
> document.getElementsByTagName('div').length
< 762
> function forEachElements() {
      [...document.getElementsByTagName('div')].map((elem, i) => {
          console.assert(i !== 348, elem);
      })
  }
< undefined
> forEachElements()
✕ ▶ Assertion failed:                                    VM5382:3
  ▶ <div class="doc-c-menu__item doc-c-menu__item_type_submenu">…</div>
< undefined
>
```

console.time(), console.timeEnd()

```
(() => {
    const startTime = Date.now();

    for (let i = 0; i < 300; i++) {
        [...document.getElementsByTagName('div')].map(elem => {
            return JSON.parse(JSON.stringify(elem.attributes))
        })
    }

    const endTime = Date.now();

    console.log(endTime - startTime);
})()

> 872
```

console.time(), console.timeEnd()

```
(() => {
    console.time();

    for (let i = 0; i < 300; i++) {
        [...document.getElementsByTagName('div')].map(elem => {
            return JSON.parse(JSON.stringify(elem.attributes))
        })
    }

    console.timeEnd();
})()

> default: 877.7861328125ms
```

Вопросы?

Яндекс

Спасибо

Сергей Тощилин

Разработчик интерфейсов