Современные возможности JavaScript

Пожалуйста, зачекиньтесь на лекции

Что есть JavaScript?

- JavaScript
- ECMA-262
- ISO/IEC 16262:2011(E)

Что есть JavaScript?

- JavaScript – язык программирования
- ECMA-262 – спецификация ECMAScript (последняя версия: 7th издание в июле 2016 года)
- ISO/IEC 16262:2011(E) – international standard

Как это было:

- ***Brendan Eich*** разработал прототип языка в мае 1995 за 10 дней
- Тогда он назывался ***Mocha***
- В сентябре 1995 в бета-версии браузера Netscape Navigator 2.0 он был выпущен под названием ***LiveScript***
- В декабре его переименовали в ***JavaScript***
- Июнь 1997 – организация ***Ecma International*** выпустила первую версию стандарта ***ECMA-262***, в котором описывала спецификацию ***ECMAScript***
- Июнь 1998 – спецификация ECMAScript 2 и международный стандарт ***ISO/IEC 16262***

Как это было:

- 2003 – ECMAScript 4 (ActionScript)
- Декабрь 2009 – ECMAScript 5
- Июнь 2011 – ECMAScript 5.1, ISO/IEC 16262:2011
- Июнь 2015 – ECMAScript 2015 (ES6 Harmony) === ES6
- …
- Июнь 2016 – ECMAScript 2016
- …

Современное состояние JavaScript

- Деструктуризация
- Классы
- Промисы
- Итераторы
- Генераторы
- ES6-модули

Погнали!

Объявление переменных


```
1   // 'use strict';
2
3   if (true) {
4       var value1 = 42;
5   }
6
7   console.log(value1);    // 42
8   console.log(value2);    // undefined
9
10  var value2 = 42;
11
```

```
1   'use strict';
2
3   if (true) {
4       let value1 = 42;
5   }
6
7   console.log(value1);    // ReferenceError
8   console.log(value2);    // ReferenceError
9
10  let value2 = 42;
11
```

Поднятие (JavaScript Hoisting)


```
1  foo();              // 42
2  bar = true;         // OK
3
4  function foo() {
5      console.log(42);
6  }
7  var bar;
8
9
```

```
1  'use strict';
2
3  new Foo();          // ReferenceError
4  bar = true;         // ReferenceError
5
6  class Foo {
7      constructor() {
8      }
9  }
10 let bar;
11
```

Ключевое слово *const*


```
1   'use strict';
2
3   const v1 = 42;
4   const v2 = [1, 2];
5   const v3 = {foo: 'bar'};
6
7   // SyntaxError: Missing initializer in const declaration
8   const v4;
9
10  v1 = 1337;          // TypeError: Assignment to constant variable
11  v2.push(3);         // OK
12  v3.foo = 'baz';     // OK
13
```

“Настоящие” константы


```
1   'use strict';
2
3   const obj = { foo: 'bar' };
4
5   Object.seal(obj);         // запечатывает объект
6   obj.foo = 'baz';          // OK - менять свойства можно
7
8   // TypeError: Can't add property foobar, object is not extensible
9   obj.foobar = null;
10
11  Object.freeze(obj);       // замораживает объект
12
13  // TypeError: Cannot assign to read only property 'foo' of object '#<Object>'
14  obj.foo = 'quux';
15
```

Новые возможности работы со строками


```
1  // Template Strings
2  let user = `Jhon Snow`;
3  let greet = `Hello, ${user}`;        // greet === 'Hello, Jhon Snow'
4
5  // Unicode support
6  let s1 = `\u2033`;        // ″, символ двойного штриха
7  let s2 = `\u{20331}`;     // 傱, китайский иероглиф с этим кодом
8
```

Ряд полезных методов:

- `str.includes(s)` – проверяет, включает ли одна строка в себя другую, возвращает `true/false`
- `str.endsWith(s)` – возвращает `true`, если строка `str` заканчивается подстрокой `s`
- `str.startsWith(s)` – возвращает `true`, если строка `str` начинается со строки `s`
- `str.repeat(times)` – повторяет строку `str` times раз

Деструктуризация

Деструктуризация


```
1  let [,name, fam] = `Lord Jhon Snow`.split(` `);
2  console.log(name);  // Jhon
3
4  [name = `Иван`, fam = `Иванов`] = [`Пётр`];
5  console.log(name, fam); // Пётр Иванов
```

Деструктуризация


```
8  let {username, age} = {username: `Jhon Snow`, age: 20, title: `lord`};
9  console.log(`${username}, ${age} лет`);      // Jhon Show, 20 лет
10
11 let opts = {
12     height: 120,
13     width: 560,
14     attrs: {classname: `button__text`}
15 };
16
17 let {width: w = 600, attrs: {classname}} = opts;
18 console.log(w, classname);                   // 560 button__text
19
```

Функции


```
1  // Параметры по умолчанию
2  function hello(name = 'Anon') {
3      console.log(`Hello, ${name}!`);
4  }
5
6  hello();              // Hello, Anon!
7  hello('Jhon');        // Hello, Jhon!
8
```

Функции


```
9
10  // Деструктуризация аргументов функции
11  function square({width, height}) {
12      return width * height;
13  }
14
15  let rectangle = {
16      width: 20,
17      height: 30
18  };
19
20  console.log(square(rectangle));     // 600
```

Оператор расширения

Оператор расширения


```
1   let numbers = [13, 24, 522, 3.14, -7];
2   // Math.max принимает аргументы через запятую
3   Math.max(numbers);      // не сработает!
4
5
6   // Раньше делали так:
7   Math.max.apply(null, numbers);  // 522
8
9
10  // Новый оператор spread: ...
11  Math.max(...numbers);           // 522
12
```

Оператор расширения


```
1   let a = [2, 3, 4];
2   let b = [1, ...a, 5];   // [ 1, 2, 3, 4, 5 ]
3   console.log(b);
4
```

rest-параметры


```
1   // При деструктуризации
2   let names = [`Jhon`, `Charlie`, `Emma`];
3   let [first, ...others] = names;
4
5   console.log(others);    // [ 'Charlie', 'Emma' ]
6
7   // При вызове функций
8   function f(count, ...nums) {
9       // не используем arguments
10      console.log(`${count} чисел: ${nums}`);
11  }
12
13  f(5, 11, 42, 37, 59, 66);   // 5 чисел: 11,42,37,59,66
14
```

Стрелочные функции


```
1   // короткий синтаксис определения
2   let square = num => num * num;
3   console.log([1, 2, 3, 4].map(square));      // 1 4 9 16
4   // console.log([1,2,3,4].map(num => num * num));
5
6
7   let hello = () => console.log('Hello!');
8   let sum = (n1, n2) => n1 + n2;
9   let modul = number => {
10      if (number < 0) {
11          return -number;
12      }
13      return number;
14  };
15
```

Стрелочные функции

- Не имеют своего `this`
- Реальное значение `this` определяется в момент создания функции
- Всегда анонимны
- Не имеют своего `arguments`

```
1   // было
2   fetch('/api/user/123')
3       .then(function (response) {
4           this.status = response.status;
5       }.bind(this));
6
7   // стало
8   fetch('/api/user/123')
9       .then(response=> {
10          this.status = response.status;
11      });
12
```

Новые возможности работы с объектами


```
let login = 'user', password = 'qwerty123';

let user = {
    login,
    password,
    [`secret-${password}`]: 42,
    hello() {
        console.log(`Hello, ${this.login}`);
        console.log(this[`secret-${this.password}`]);
    }
};

user.hello();       // Hello, user \n 42

```

Новые возможности работы с объектами

Дополнительно:

- Метод `Object.assign(target, src1, src2...)` – копирует свойства из всех аргументов в первый объект
- Метод `Object.is(value1, value2)` проверяет два значения на равенство

```
1  let obj1 = {name: `Jhon`, age: 18};
2  let obj2 = {fam: `Snow`, age: 22};
3  let result = Object.assign({}, obj1, obj2);
4
5  console.log(result);
6  // { name: 'Jhon', age: 22, fam: 'Snow' }
7
```

Set, Map, WeakSet и WeakMap


```
1   // Map – ассоциативный массив (хеш-таблица)
2   let map = new Map();
3   map.set(`1`, `str1`);
4   map.set(1, `num1`);
5   map.set(true, `bool1`);
6
7   map.has(key);   // проверяет наличие такого ключа
8   map.get(key);   // получает элемент по ключу
9   map.delete(key);        // удаляет элемент с таким ключом
10
11  map.keys();     // перебор ключей, значений, пар в Map
12  map.values();
13  map.entries();
14
```

Set, Map, WeakSet и WeakMap


```
1   // Set – множество уникальных элементов
2   let set = new Set();
3   let [a, b] = [{prop: `value1`}, {prop: `value2`}];
4
5   set.add(a); // повторяющиеся элементы хранятся один раз
6   set.add(b);
7   set.add(a);
8   set.add(a);
9   set.add(b);
10
11  console.log(set.size);  // 2
12  set.forEach(elem => console.log(elem.prop));    // value1 value2
13
```

Set, Map, WeakSet и WeakMap


```
1   // Перебор элементов коллекции
2   let map = new Map([[`key1`, 1], [`key2`, 2], [`key3`, 3]]); // три элемента
3
4   map.forEach(function (value, key, allMap) {
5       console.log(`${key}=${value} `);        // key1=1 key2=2 key3=3
6   };
7
8   map.entries().forEach(...); // ошибка: не массив!
9
10
11  // key1=1 key2=2 key3=3
12  [...map.entries()].forEach(el => console.log(`${el[0]}=${el[1]} `));
13
```

ES6 классы


```
1  class User {
2      constructor(name) {
3          this.name = name;
4      }
5      hello() {
6          console.log(`Hello, ${this.name}`);
7      }
8  }
9
```

ES6 классы


```
10  class Admin extends User {
11      constructor(name, age) {
12          super(name);
13          this.Age = age;
14      }
15
16      set Age(years) {
17          console.log(`${years} лет`);
18      }
19  }
20
21  let admin = new Admin(`Jnoh`, 19);  // 19 лет
22  admin.hello();                      // Hello, Jhon
23  admin.Age = 25;                     // 25 лет
24
```

Символы Symbol

Уникальные значения


```
1   // без new
2   let [sym1, sym2] = [Symbol(), Symbol(`label`)];
3
4   console.log(typeof sym1);                  // symbol
5   console.log(sym2 == Symbol(`label`));      // false
6
7
8   // реестр глобальных символов
9   let me = Symbol.for(`Jhon`);
10  console.log(Symbol.for(`Jhon`) === me); // true
11  console.log(Symbol.keyFor(me));            // Jhon
12
```

Уникальные значения


```
let Me = {
    [Symbol.for(`sayHello`)]() {
        console.log(`Hello, ${this.name}`);
    },
    name: `Anon`
};

Me[Symbol.for(`sayHello`)]();       // Hello, Anon

```

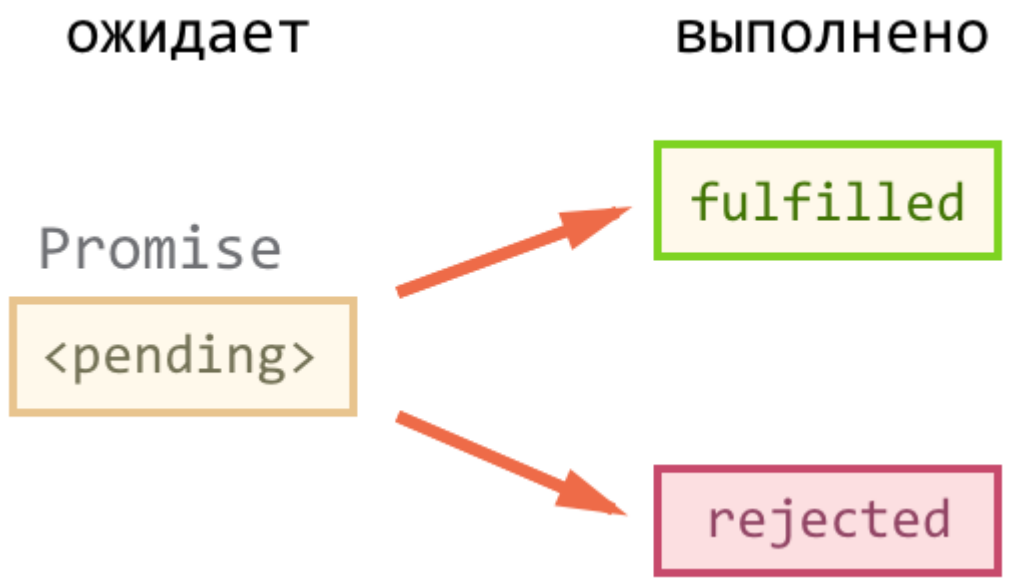
Promise

Promise – “обещание”

Промисы предоставляют удобный способ организации асинхронного кода.

Promise – это специальный объект, который содержит своё состояние. Вначале pending («ожидание»), затем – одно из:

- fulfilled («выполнено успешно»)
- rejected («выполнено с ошибкой»)



В браузере промисы могут быть полезны при работе с сетью (HTTP-запросы – асинхронные операции). На сервере у промисов множество применений

Promise – “обещание”

Метод
`promise.then(onFulfilled, onRejected)`
Позволяет добавить коллбек к промису. При этом,

- метод `onFulfilled` вызовется при успешном выполнении промиса
- метод `onRejected` – при ошибке.

Вместо
`promise.then(null, onRejected)`
можно писать просто
`promise.catch(onRejected)`

`onFulfilled` и `onRejected` – это функции-колбеки

```
1  // метод fetch возвращает промис
2  let promise = fetch(`/logo.png`);
3  promise
4      .then(function (response) {
5          console.log(`Промис без ошибок`);
6          console.log(`${response.status}`);
7      })
8      .catch(function (err) {
9          console.log(`Ошибка в промисе`);
10         console.error(err);
11     });
12
```

Promise – “обещание”


```
1   // метод fetch возвращает промис
2   let promise = fetch(`/logo.png`);
3   promise
4       .then(function (response) {
5           console.log(`Промис без ошибок`);
6           console.log(`${response.status}`);
7       })
8       .catch(function (err) {
9           console.log(`Ошибка в промисе`);
10          console.error(err);
11      });
12
```

Промисификация


```
1   // оборачивание асинхронного функционала в обёртку, возвращающую промис
2   function wait() {
3       return new Promise(function (resolve, reject) {
4           setTimeout(function () {
5               resolve('OK');
6           }, 2000);
7       });
8   }
9
10
11  wait()
12      .then(() => console.log('Прошло 2 сек'));
13
```

Promise.prototype.*


```
1   // Выполнит все промисы и вернёт массив результатов
2   Promise.all([iterable])
3
4   // Вернёт первый выполнившийся промис
5   Promise.race([iterable])
6
7   // Создаст успешно выполнившийся промис
8   Promise.resolve(value)
9
10  // Создаст промис, выполнившийся с ошибкой
11  Promise.reject(error)
12
```

Итераторы

Концепция итерируемости

Итераторы – расширяющая понятие «массив» концепция, которая пронизывает современный стандарт JavaScript сверху донизу. Итерируемые или, иными словами, «перебираемые» объекты – это те, содержимое которых можно перебрать в цикле:

- обычные массивы
- псевдомассив `arguments`
- строки
- списки DOM-нод в браузере
- генераторы
- `Map`, `Set`
- пользовательские итерируемые объекты

В общем смысле, итератор – это объект, предоставляющий метод `next()`, который возвращает следующий элемент определённой последовательности.

Концепция итерируемости


```
1  // для перебора итерируемых объектов используется новый вид циклов
2  for (variable of iterable) {
3      void 0;
4  }
5
6  const iterable1 = [10, 20, 30];
7  const iterable2 = 'hello';
8
9
10 for (let num of iterable1) {
11     console.log(num);        // 10 20 30
12 }
13
14 for (let char of iterable2) {
15     console.log(char);       // h e l l o
16 }
17
```

Концепция итерируемости


```
1   // Оператор расширения как раз выполняет итерирование по объекту
2   // и возвращает массив значений:
3
4   console.log(...[1, 2, 4, 8]);        // 1 2 4 8
5   console.log(...`hello`);             // h e l l o
6
```

Symbol.iterator


```
1  let itetable = ['a', 'bb', 'ccc'];
2  let iter = itetable[Symbol.iterator]();
3
4  console.log(iter.next());   // { value: 'a', done: false }
5  console.log(iter.next());   // { value: 'bb', done: false }
6  console.log(iter.next());   // { value: 'ccc', done: false }
7  console.log(iter.next());   // { value: undefined, done: true }
8
```

Symbol.iterator


```
let range = {from: 1, to: 5};

// сделаем объект range итерируемым
range[Symbol.iterator] = function () {
    let current = this.from;
    let last = this.to;
    // метод должен вернуть объект с методом next()
    return {
        next: () => {
            return ((current <= last) ? {done: false, value: current++} : {done: true});
        }
    }
};

for (let num of range) {
    console.log(num);       // 1 2 3 4 5
}

console.log(...range);

```

Генераторы

Создание генераторов


```
1  function * gen() {
2      yield 1;
3      yield 2;
4      return 3;
5  }
6
7  let generator = gen();
8  console.log(generator.next());  // { value: 1, done: false }
9  console.log(generator.next());  // { value: 2, done: false }
10 console.log(generator.next());  // { value: 3, done: true }
11 console.log(generator.next());  // { value: undefined, done: true }
12
```

Создание генераторов


```
let range = function *(begin, end) {
    for (let v = begin; v <= end; v++) {
        yield v
    }
};

console.log(...range(5, 12));   // 5 6 7 8 9 10 11 12
```

Композиция генераторов


```
let twice = function *(value) {
    yield value; yield value;
};

let twicer = function *(array) {
    for (let item of array) {
        yield * twice(item);
    }
};

console.log(...twicer(['a', 42, true, {foo: 'bar'}]));
// Выведем: a a 42 42 true true { foo: 'bar' } { foo: 'bar' }
```

ES6-модули

ES6-модули


```
(function() {
    const admin = {
        username: 'Jhon Snow',
        password: 'passw0rd'
    };

    const hello = function (user) {
        console.log(`Hello, ${user.username}`);
    };

    window.admin = admin;
    window.hello = hello;
})();

```

```
(function() {
    const admin = window.admin;
    const hello = window.hello;

    // Hello, Jhon Snow
    hello(admin);
})();

```

ES6-модули


```
'use strict';

const admin = {
    username: 'Jhon Snow',
    password: 'passw0rd'
};

const hello = function (user) {
    console.log(`Hello, ${user.username}`);
};

export {admin, hello};

```

```
import {admin, hello} from './lib.js';

hello(admin);   // Hello, Jhon Snow

// ИЛИ ТАК:
import * as lib from './lib.js';

lib.hello(lib.admin);   // Hello, Jhon Snow


// ИЛИ ТАК:
import {admin as me, hello as hi} from `lib.js`;

hi(me);     // Hello, Jhon Snow

```

[MDN - import statement]

Поддержка браузерами

Поддержка браузерами

Таблица поддержки разных версий языка браузерами

- ✓ http://kangax.github.io/compat-table/es6/

Возможности браузеров

- ✓ http://caniuse.com/

Полифиллы

- ✓ «Полифилл» – это библиотека, которая добавляет в старые браузеры поддержку возможностей, которые в современных браузерах являются встроенными

Транспайлинг

- ✓ Конвертация кода программы, написанного на одном ЯП в другой ЯП

Babel

Многофункциональный транспайлер.

Позволяет использовать самые последние возможности JavaScript.

Поддерживает транспайлинг:

- Из версий языка выше ES5 (т.е. ES6, ES2016, ES.Next, etc…)
- Кода, написанного для react-приложений (расширение файлов `.jsx`)

✓ Официальный сайт: http://babeljs.io/

✓ Попробовать онлайн: http://babeljs.io/repl/

Babel – использование


```
# установка пакетов
$ npm install --save-dev babel-cli babel-preset-latest

# конфигурация
$ nano .babelrc
{
    "presets": ["latest"]
}

# запуск
$ babel modern.js --watch --out-file compiled.js

```

Текущие версии JavaScript’a

- ✓ Декабрь 1999 – ECMAScript 3
- ✓ ECMAScript 4 – всё сложно
- ✓ Декабрь 2009 – ECMAScript 5
  - ✓ Июнь 2011 – ECMAScript 5.1 (ISO/IEC 16262:2011)
- ✓ Июнь 2015 – ECMAScript 6 (ECMAScript 2015)
- ✓ Июнь 2016 – ECMAScript 7 (ECMAScript 2016)
- ✓ . . .
- ✓ Июнь 2017 – ECMAScript 2017 (и так далее)

ES.Next – так временно называют совокупность новых возможностей языка, которые могут войти в следующую версию спецификации. Фичи из ES.Next правильнее называть *“предложения”* (*proposals*), потому что они всё ещё находятся на стадии обсуждения.

Процесс *TC39*

TC39 (технический комитет 39) — занимается развитием *JavaScript*. Его членами являются компании (помимо прочих, все основные производители браузеров). TC39 регулярно собирается, на встречах присутствуют участники, представляющие интересы компаний, и приглашенные эксперты

Процесс TC39:

- ✓ 0 этап: идея
- ✓ 1 этап: предложение
- ✓ 2 этап: черновик
- ✓ 3 этап: кандидат
- ✓ 4 этап: финал

ECMAScript 2016


```
1  // exponentiation operator
2
3  console.log(2 ** 8);    // 256
4  console.log(2 ** -1);   // 0.5
5  console.log(16 ** 0.5); // 4
6
7
```

```
1  // Array.prototype.includes
2
3  // Вернёт true
4  ['a', 'b', 'c'].includes('a');
5
6  // Вернёт false
7  ['a', 'b', 'c'].includes('d');
8
9
10 [NaN, 0].includes(NaN);     // true
11 [NaN, 0].indexOf(NaN);     // -1
12
13
```

ECMAScript *Proposals*

Будущее JavaScript

Текущие предложения (*proposals)*:

- ✓ https://github.com/tc39/proposals/

Stage-4:

- ✓ https://github.com/tc39/proposals/blob/master/finished-proposals.md

- `Object.values` / `Object.entries`
- Async Functions
- String padding

Stage-3, stage-2:

- Object Rest/Spread Properties
- Class and Property Decorators
- `Promise.prototype.finally`

Благодарю за внимание!

Пожалуйста, зачекиньтесь на лекции и оставьте обратную связь

Лекция, код примеров, дополнительные материалы находятся по ссылке:
https://github.com/frontend-park-mail-ru/modern-es