Основы JS

Объекты и объектные типы

Объекты: основы

- Неупорядоченная коллекция *свойств*
- Свойство имеет имя и значение
- Имя свойства - строка, значение может быть любым
- Три способа создания объектов: литерал, `new`, `Object.create()`

```
let empty = {}; // Пустой объект
let point = {x: 5, y: 6};
let x = 10, y = 15;
let point2 = {x, y};  // то же самое что и {x: x, y: y}

let suffix = "name";
let book = {
    "book title": "A Clockwork Orange", // Имя свойства с пробелом
    "class": "fiction",   // Зарезервированное слово
    "dynamicId": getId(), // Значение свойства - любое выражение
    "tags": ["fiction", "Burgess"], // И любая структура данных
    ["Author's " + suffix]: "Anthony Burgess", // "Author's name"

    getName: function() { return null; },  // Метод
    getName2() { return null; }            // Метод: сокращенный синтаксис
};
```

Прототип объекта

- Каждый объект имеет скрытый атрибут `[[Prototype]]`
- `[[Prototype]]` может указывать на объект-*прототип* или быть `null`
- Все объекты-литералы имеют прототипом `Object.prototype`
- **Очень важно** не путать атрибут `[[Prototype]]` и свойство `prototype`
- Последовательность прототипов называется *цепочкой прототипов*
- `Object.create()` создает новый объект с заданным прототипом

```
let obj = Object.create({x: 1, y: 2}); // obj наследует x и y
let obj2 = Object.create(null);        // obj2 не имеет прототипа
let obj3 = Object.create(Object.prototype); // obj3 - обычный объект
```

Проверка прототипов объекта

- `Object.getPrototypeOf(obj)`- получить ссылку на прототип объекта
- `Object.prototype.isPrototypeOf(obj)`- является ли объект прототипом другого
- `Object.prototype.__proto__` - (*legacy*) - ссылка на прототип объекта

```
let a = {x: 1},
    b = Object.create(a),
    c = Object.create(b);

Object.getPrototypeOf(c) === b; // => true
Object.getPrototypeOf(b) === a; // => true
Object.getPrototypeOf(c) === a; // => false
c.__proto__ === b;              // true
b.__proto__ === a;              // true
c.__proto__ === a;              // false

a.isPrototypeOf(b);             // true
a.isPrototypeOf(c);             // true
```

Установка прототипа

- Есть 3.5 способа задать прототип объекта
- Создание объекта с помощью `Object.create(proto, props)`
- Создание объекта с помощью функции конструктора и оператора `new`
- `Object.setPrototypeOf(obj, proto)`- установить прототип
- `Object.prototype.__proto__` - нестандартное свойство-указатель на прототип (*legacy*)
- Как правило, прототип нужно задавать сразу при создании объекта
- Изменение прототипа в "runtime" редко бывает полезным, и чаще вредит

```
let a = {x: 1, y: 2},
    b = {x: 3, y: 4},
    c = Object.create(a);

console.log(c.__proto__ === a);               // true
Object.setPrototypeOf(c, b);
console.log(Object.getPrototypeOf(c) === b); // true
```

Чтение и запись свойств

- Операторы `[]` и `.` - обращение к свойству объекта
- Обращение к несуществующему свойству => `undefined`
- Обращение к свойству несуществующего объекта => `ReferenceError`
- Свойство ищется вначале в объекте, затем выше по цепочке прототипов
- Присвоение значения *всегда* выполняется в оригинальном объекте

```
let obj = {x: 1, "first name": "Shmebulock"},
    suffix = "name";

obj.x;                  // 1
obj["first " + suffix]; // "Shmebulock"
obj["interface"];       // undefined
```

```
let obj0 = {a: 'a0', b: 'b0', c: 'c0'},
    obj1 = Object.create(obj0),
    obj2 = Object.create(obj1);

obj1.a = 'a1';
obj1.b = 'b1';
obj2.a = 'a2';

console.log(obj2.a, obj2.b, obj2.c); // => 'a2 b1 c0'
console.log(obj1.a, obj1.b, obj1.c); // => 'a1 b1 c0'
console.log(obj0.a, obj0.b, obj0.c); // => 'a0 b0 c0'
```

Атрибуты свойств

- Каждое свойство объекта имеет четыре атрибута
- `[[value]]` - значение свойства
- `[[Writable]]` - доступность для перезаписи
- `[[Enumerable]]` - доступность для перечисления
- `[[Configurable]]` - доступность для настройки
- "Обычное" свойство - атрибуты равны `true` (`value => undefined`)
- *Дескриптор свойства* - объект, описывающий атрибуты свойства

```
let a = {y: 1};
Object.defineProperty(a, 'x', {
    writable:     true, // false по умолчанию
    enumerable:   true, // false по умолчанию
    configurable: true, // false по умолчанию
    value: 15           // undefined по умолчанию
});
console.log(a.x);       // => 15
```

Управление атрибутами свойств (методы `Object`)

- `.create(proto[, descrs])`- описать свойства при создании объекта
- `.defineProperties(obj, descrs)`- изменить список свойств
- `.defineProperty(obj, prop, descr)`- изменить одно свойство
- `.getOwnPropertyDescriptor(obj, prop)`- получить дескриптор свойства
- `.getOwnPropertyDescriptors(obj)`- получить дескрипторы свойств
- Атрибуты свойства (define) по умолчанию => `false`, `value => undefined`

```
let obj = {x: 1};
Object.defineProperties(obj, {
    y: {value: 2},
    z: {writable: true}
});

console.log(Object.getOwnPropertyDescriptors(obj));
/* {
    x: {configurable: true, enumerable: true, value: 1, writable: true},
    y: {configurable: false, enumerable: false, value: 2, writable: false},
    z: {configurable: false, enumerable: false, value: undefined, writable: true},
}
*/
```

Атрибуты свойств: поведение

- `[[Writable]]: false` запрещает присвоение значения свойству
- `[[Enumerable]]: false` скрывает свойство от `for..in` и `Object.keys()`
- `[[Configurable]]: false` запрещает удаление и изменение свойства

```
'use strict';
let obj = Object.create(null, { // Объект без прототипа, но с полем x
    x: {configurable: true, value: 15},
    y: {enumerable: true, value: 10}
});
console.log(obj.x); // => 15
obj.x = 24;         // => TypeError, присвоение non-writable полю (strict mode)
delete obj.y;       // => TypeError, удаление non-configurable поля (strict mode)

for (let i in obj) console.log(i); // выведет только 'y'

Object.defineProperty(obj, 'x', {value: 24});
console.log(obj.x); // => 24 (ah_ty_hitraya_zh.jpg)

Object.defineProperty(obj, 'x', { configurable: false });
Object.defineProperty(obj, 'x', {value: 30}); // TypeError, non-configurable свойство
```

Прочие действия и проверки со свойствами

- `delete` - удалить собственное свойство из объекта
- `in` - проверить наличие свойства в объекте или его прототипах
- `.keys(obj)` - список собственных *перечислимых* свойств
- `.getOwnPropertyNames(obj)`- список *собственных* свойств
- `.prototype.hasOwnProperty(prop)`- наличие *собственного* свойства

```
let obj = Object.create({x: 'x0', y: 'y0', z: 'z0'}, {
    y: {enumerable: false, writable: true, configurable: true, value: 'y1'},
});
obj.x = 'x1';
obj.z = 'z1';

'z' in obj;     // true
delete obj.z;
'z' in obj;     // true, значение все еще есть в прототипе
obj.hasOwnProperty('z'); // false

Object.keys(obj); // ['x'] - только собственные перечислимые свойства
Object.getOwnPropertyNames(obj); // ['y', 'x'] - все собственные свойства
```

Объекты: разное

- `Object.is()` - проверка эквивалентности, которую мы заслуживаем
- `Object.assign()` - слияние всех перечислимых собственных свойств одного объекта в другой

```
Object.is(0, -0);      // false
Object.is(-0, -0);     // true
Object.is(NaN, 0/0);   // true

let o1 = { a: 1, b: 1, c: 1 },
    o2 = { b: 2, c: 2 };
    o3 = { c: 3 };

let obj = Object.assign({}, o1, o2, o3);
console.log(obj); // { a: 1, b: 2, c: 3 }
```

Массивы

- *Массив* - упорядоченная коллекция значений
- Массив в JS - это объект с целочисленными именами свойств
- Обращение к несуществующему индексу => `undefined`
- Свойство `length` - длина массива
- Создание массива: литерал, `new Array()`, `Array.from()`
- `Array.isArray()` - является ли объект массивом

```
let a = [1, 2, 3, 4],         // Создание через литерал
    b = [1, 'abc', {x: 5}, [1, 2, 3]],  // Любые типы данных
    c = new Array(10),        // Массив с 10 элементами undefined
    d = new Array(1, 2, 3);   // Массив с 3 элементами: 1, 2, 3
    e = Array.of(10);         // Массив из 1 элемента: 10

a.length;     // 4
c.length;     // 10
e.length;     // 1
Array.isArray(a);  // true
Array.isArray({}); // false
```

Работа с содержимым массива

- `[]` - обращение к элементу массива
- `push()`, `pop()` - добавление/удаление элемента в конце
- `unshift()`, `shift()` - добавление/удаление элемента в начале
- `slice()`, `splice()` - получить или заменить несколько элементов

```
let a = [1, 2, 3, 4, 5];
3 in a;    // true
5 in a;    // false
a[3];      // 4
a[5];      // undefined

a.pop();   // => 5
a.length;  // 4
a.push(6);
a.shift(); // => 1
a;         // [2, 3, 4, 6]
```

```
let a = [1, 2, 3, 4, 5];
a.slice(0, 3)); // [1, 2, 3]
a.slice(3);     // [4, 5]
a.slice(1, -1); // [2, 3, 4]

let b = [1, 2, 3, 4, 5];
b.splice(3);    // => [4, 5]; b => [1, 2, 3]
b.splice(1, 1); // => [2]; b => [1, 3]
b.splice(0, 1, 'a', 'b', 'c');
// => [1], b => ['a', 'b', 'c', 3]
```

Работа с массивами

- `join()` - склеивание всех элементов в строку
- `sort()` - сортировка элементов
- `concat()` - конкатенация массивов
- `reverse()` - изменить порядок элементов на обратный

```
let a = [2, 5, 3, 4, 1];
a.join(':');    // "2:5:3:4:1"
a.reverse();    // [1, 4, 3, 5, 2], массив изменился
a.sort();       // [1, 2, 3, 4, 5]
a.sort(function(a, b) {
    return b - a;
});                 // [5, 4, 3, 2, 1]
a.concat([0, -1]); // [5, 4, 3, 2, 1, 0, -1]
a;                 // [5, 4, 3, 2, 1]
```

Поиск элемента в массиве

- `indexOf(val[, fromIndex])`, `lastIndexOf(val[, fromIndex])`- получить индекс элемента, или -1
- `find(val)`, `findIndex(val)` - поиск элементов по условию
- `includes(val)` - проверка, находится ли элемент в массиве

```
let a = [1, 2, 3, 4, 5, 3];
a.indexOf(10);       // -1
a.indexOf(3);        // 2
a.indexOf(3, 3);     // 5
a.lastIndexOf(3);    // 5
a.includes(10);      // false
a.includes(4);       // true

a.find(function(val) { return val > 3; });      // 4
a.findIndex(function(val) { return val > 3; }); // 3
```

Массивы: let's get more functional

- `forEach()` - перебор элементов массива
- `map()` - создание нового массива через модификацию оригинального
- `filter()` - фильтрация элементов массива
- `every()`, `some()` - проверка элементов на соответствие предикату
- `reduce()` - вычислить значение, пройдя по значениям массива

```
let a = [1, 2, 3, 4, 5],
    sum = 0;

a.forEach(function(value) { sum += value; });
sum;     // 15
a.forEach(function(value, index) { a[index] = value + 1; }); // a => [2, 3, 4, 5, 6]

a.map(function(value) {return value + 1;}); // => [3, 4, 5, 6, 7]
a;       // [2, 3, 4, 5, 6], массив не изменился

a.filter(function(val) {return val % 2});  // => [3, 5]
a.every(function(val)  {return val > 0});  // => true, все больше нуля
a.every(function(val)  {return val % 2});  // => false, не все нечетные
a.some(function(val)   {return val % 2});  // => true, некоторые нечетные
```

Array-like objects

- Вполне, но не совсем, абсолютно не похожи на массивы
- Обычные объекты, с целочисленными индексами и свойством `length`
- Не имеют методов, которые наследуют от `Array` обычные массивы
- Строки, объект `arguments`, `document.getElementsByTagName()`
- Из array-like объекта можно создать массив: `Array.from()`

```
Array.from('test');   // ['t', 'e', 's', 't']
[...'test'];          // то же самое

let a = {0: 'a', 1: 'b', 2: 'c', length: 3};
a.map;                // undefined; a - обычный объект
[...a];               // TypeError, у a нет итератора

Array.from(a);        // ['a', 'b', 'c']
Array.from(a).map;    // function ...
```

Объекты как множества (sets) и контейнеры "ключ - значение" (maps)

- Ключами объектов могут быть только строки
- Объекты наследуют свойства из `Object.prototype`
- Размер контейнера - `keys().length`

```
let map1 = {};
    map2 = Object.create(null);
'toString' in map1;      // true
'toString' in map2;      // false

// Присваивание одному и тому же полю:
map2[5] = 1;
map2["5"] = 2;
map2[[5]] = 3;
map2[{toString: function() {return "5";}}] = 4;

map2[NaN] = 1;
map2["NaN"] = 2;

map2[{}] = 1;
map2[{x: 1}] = 2;   // Свойство с ключом "[object Object]" O_o
```

Тип Set

- Упорядоченная коллекция значений без повторов
- Создается с помощью конструктора `new Set([iterable])`
- Значения сравниваются по правилам `Object.is()` (кроме +-0)
- `size` - число значений в множестве
- `add(), delete()` - добавить/удалить значение в/из множества
- `has()` - проверить наличие элемента в множестве
- `clear()` - очистить все множество
- `forEach()` - перебор значений в множестве

```
let s = new Set();
s.add(5); s.add("5"); s.add(5);
s.size;     // 2, дубликат проигнорирован
s.has("5"); // true
s.delete("55"); // нет такого значения, игнорируется
s.clear();
s.size;         // 0

let s2 = new Set(['a', 'b', 'c']);
s2.forEach(function(value) { console.log(value)}); // => a, b, c
```

Тип Map

- Упорядоченная коллекция пар "ключ-значение"
- Создается с помощью конструктора `new Map([iterable])`
- Ключи сравниваются по правилам `Object.is()` (кроме +-0)
- `size` - число пар "ключ-значение" в контейнере,
- `set(), get()` - установить/изменить/получить значение по ключу

```
let map = new Map(),
    a = {}, b = {};

map.set(5, "five");
map.set("5", "string five");
map.set(a, "first object");
map.set(b, "second object");

map.get(5);    // "five"
map.get("5");  // "string five"
map.get(a);    // "first object"
map.get(b);    // "second object"

map.size;      // 4
```

Map: методы

- `has()` - проверить наличие ключа
- `delete()` - удалить один ключ/значение
- `clear()` - очистить все
- `forEach()` - перебор значений

```
let map = new Map([["name", "Nicholas"], ["age", 25]]);

map.has("name");    // true
map.delete("name");
map.has("name"); // false
map.get("name"); // undefined
map.size;        // 1

map.clear();
map.has("age");  // false
map.size;        // 0

map.set("key1", "a");
map.set("key2", "b");
map.forEach(function(value, key) {console.log(`value: ${value}, key: ${key}`)});
// => value: a, key: key1
// => value: b, key: key2
```

Итераторы для `Array`, `Set`, `Map` и преобразования между типами

- `values()` - значения; используется умолчанию для `Array` и `Set`
- `keys()` - ключи; для множеств то же самое что и `values()`
- `entries()` - пары "ключ-значение"; используется по умолчанию для `Map`;
- `Array` => `Map/Set` через конструктор `Map/Set`
- `Map/Set` => `Array` с помощью `Array.from()` или spread синтаксиса: `[...x]`

```
let a = ['a', 'b', 'c', 'a'],
    s  = new Set(a);          // множество с элементами 'a', 'b', 'c'
    s2 = new Set(a.keys());   // множество с элементами 0, 1, 2, 3
    m1 = new Map(a.entries()); // хэш 0 => 'a', 1 => 'b', 2 => 'c', 3 => 'a'

    s3 = new Set(m1);         // множество [0, 'a'], [1, 'b'], [2, 'c'], [3, 'a']

    a2 = [...s3.entries()];
    // массив [[[0, 'a'], [0, 'a']], [[1, 'b'], [1, 'b']], [[2, 'c'], [2, 'c']], [[3, 'a'],
```