Разработка пользовательского веб-интерфейса

JS: массивы, словари, JSON

Сергей Геннадьевич Синица

КубГУ, 2020

sin@kubsu.ru

Массивы

Массив (Array) в JavaScript является глобальным объектом, который используется для создания массивов.

Массивы представляют собой высокоуровневые спископодобные объекты.

Могут быть с числовыми индексами либо со строчными ключами (словари).

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array

Создание массива

```
[element0, element1, ..., elementN]

new Array(element0, element1[, ...[,
elementN]])

new Array(arrayLength)
```

Создание массива

```
var fruits = new Array('Яблоко', 'Банан');

// то же самое что

var fruits = ['Яблоко', 'Банан'];

console.log(fruits.length);
// 2
```

Доступ к элементам по индексу

```
var first = fruits[0];
// Яблоко

var last = fruits[fruits.length - 1];
// Банан
```

Итерирование по массиву

```
fruits.forEach(function(item, index, array){
  console.log(item, index);
});
// Яблоко 0
// Банан 1
```

Добавление элемента в конец

```
var newLength = fruits.push('Апельсин');
// ["Яблоко", "Банан", "Апельсин"]
```

Удаление последнего элемента

```
var last = fruits.pop();
// ["Яблоко", "Банан"];
```

Удаление первого элемента

```
var first = fruits.shift();
// ["Банан"];
```

Поиск номера элемента

```
fruits.push('Манго');
// ["Клубника", "Банан", "Манго"]

var pos = fruits.indexOf('Банан');
// 1
```

Удаление элемента

```
var removedItem = fruits.splice(pos, 1);

// ["Клубника", "Манго"]
```

Удаление нескольких элементов

```
var vegetables = ['Капуста', 'Репа', 'Редиска', 'Морковка'];
console.log(vegetables);
// ["Капуста", "Репа", "Редиска", "Морковка"]


var pos = 1, n = 2;


var removedItems = vegetables.splice(pos, n);
// так можно удалить элементы, n определяет количество элементов
для удаления,
// начиная с позиции(pos) и далее в направлении конца массива.


console.log(vegetables);
// ["Капуста", "Морковка"] (исходный массив изменён)


console.log(removedItems);
// ["Репа", "Редиска"]
```

Создание копии массива

```
var shallowCopy = fruits.slice();
// ["Клубника", "Манго"]

var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// Array ["camel", "duck"]

console.log(animals.slice(-2));
// Array ["duck", "elephant"]

console.log(animals.slice(2, -2));
// Array ["camel"]
```

Slice

Метод slice() не изменяет исходный массив, а возвращает новую «одноуровневую» копию, содержащую копии элементов, вырезанных из исходного массива. Элементы исходного массива копируются в новый массив по следующим правилам:

Ссылки на объекты (но не фактические объекты): метод slice() копирует ссылки на объекты в новый массив. И оригинал, и новый массив ссылаются на один и тот же объект. То есть, если объект по ссылке будет изменён, изменения будут видны и в новом, и в исходном массивах.

Строки и числа (но не объекты String и Number): метод slice() копирует значения строк и чисел в новый массив. Изменения строки или числа в одном массиве никак не затрагивает другой.

Если к любому массиву будет добавлен новый элемент, это никак не повлияет на другой массив.

Доступ к элементам

```
var arr = ['первый элемент', 'второй элемент', 'последний элемент'];
console.log(arr[0]);              // напечатает 'первый элемент'
console.log(arr[1]);              // напечатает 'второй элемент'
console.log(arr[arr.length - 1]); // напечатает 'последний элемент'
```

Длина массива

```
// Некоторые встроенные методы массива (например, join, slice, indexOf и т.д.) учитывают значение свойства length при своём вызове. Другие методы (например, push, splice и т.д.) в результате своей работы также обновляют свойство length массива.


var fruits = [];
fruits.push('банан', 'яблоко', 'персик');

console.log(fruits.length); // 3
```

Длина массива

```
// При установке элемента в массиве, если он имеет
действительный индекс и этот индекс выходит за пределы
текущих границ массива, движок соответствующим образом
обновит свойство length:

fruits[5] = 'манго';
console.log(fruits[5]);              // 'манго'
console.log(Object.keys(fruits));    // ['0', '1', '2', '5']
console.log(fruits.length);          // 6

// Увеличиваем свойство length
fruits.length = 10;
console.log(Object.keys(fruits));    // ['0', '1', '2', '5']
console.log(fruits.length);          // 10
```

Длина массива

```
// При установке элемента в массиве, если он имеет
действительный индекс и этот индекс выходит за пределы
текущих границ массива, движок соответствующим образом
обновит свойство length:

fruits[5] = 'манго';
console.log(fruits[5]);              // 'манго'
console.log(Object.keys(fruits));    // ['0', '1', '2', '5']
console.log(fruits.length);          // 6

// Увеличиваем свойство length
fruits.length = 10;
console.log(Object.keys(fruits));    // ['0', '1', '2', '5']
console.log(fruits.length);          // 10
```

Длина массива

```
// Уменьшение свойства length приведёт к удалению элементов

fruits.length = 2;
console.log(Object.keys(fruits)); // ['0', '1']
console.log(fruits.length); // 2
```

Сортировка

```
var fruit = ['арбузы', 'бананы', 'Вишня'];
fruit.sort(); // ['Вишня', 'арбузы', 'бананы']

var scores = [1, 2, 10, 21];
scores.sort(); // [1, 10, 2, 21]

var things = ['слово', 'Слово', '1 Слово', '2 Слова'];
things.sort(); // ['1 Слово', '2 Слова', 'Слово', 'слово']
// В Unicode, числа находятся перед буквами в верхнем регистре,
// а те, в свою очередь, перед буквами в нижнем регистре.
```

Сортировка

```
var numbers = [4, 2, 5, 1, 3];
numbers.sort(function(a, b) {
  return a - b;
});
console.log(numbers); // [1, 2, 3, 4, 5]

/*    Если compareFunction(a, b) меньше 0, сортировка поставит a
по меньшему индексу, чем b, то есть, a идёт первым.
    Если compareFunction(a, b) вернёт 0, сортировка оставит a и b
неизменными по отношению друг к другу, но отсортирует их по
отношению ко всем другим элементам. Обратите внимание: стандарт
ECMAscript не гарантирует данное поведение, и ему следуют не все
браузеры (например, версии Mozilla по крайней мере, до 2003 года).
    Если compareFunction(a, b) больше 0, сортировка поставит b по
меньшему индексу, чем a.
    Функция compareFunction(a, b) должна всегда возвращать
одинаковое значение для определённой пары элементов a и b. Если
будут возвращаться непоследовательные результаты, порядок
сортировки будет не определён.
*/
```

Сопоставления с regex

```
/* Результатом сопоставления регулярного выражения строке является
JavaScript-массив. Этот массив имеет свойства и элементы,
предоставляющие информацию о сопоставлении. Подобные массивы
возвращаются методами RegExp.exec, String.match и String.replace.
*/

// Сопоставляется с одним символом d, за которым следует один
// или более символов b, за которыми следует один символ d
// Запоминаются сопоставившиеся символы b и следующий за ними
символ d
// Регистр игнорируется

var myRe = /d(b+)(d)/i;
var myArray = myRe.exec('cdbBdbsbz');

// myArray['input'] содержит cdbBdbsbz
// myArray['index'] содержит 1
// myArray[0] содержит dbBd
// myArray[1] содержит bB
// myArray[2] содержит d
```

Массив это… словарь? Объект?

```
var myCar = new Object();
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;

myCar.color; // undefined

// Аналогично
myCar["make"] = "Ford";
myCar["model"] = "Mustang";
myCar["year"] = 1969;

/*Разница в Array.prototype, в котором представлены все методы,
присущие массивам. Каждый новый массив наследует эти методы из
Array.prototype.

Важно отметить, что значением свойства prototype в Array.prototype
является Object.prototype. Это означает, что массивы — это просто
объекты, но с дополнительными методами. Нет ничего такого, что делает
объект, но не смог бы сделать массив.*/
```

Объект и массив хранят пары свойств ключ-значение

```
var myObj = new Object(),
    str = "myString",
    rand = Math.random(),
    obj = new Object();

myObj.type              = "Dot syntax";
myObj["date created"]   = "String with space";
myObj[str]              = "String value";
myObj[rand]             = "Random Number";
// JS вызывает метод obj.toString() для получения ключа
myObj[obj]              = "Object";
myObj[""]               = "Even an empty string";
myObj[0]                = "Numeric key";

myObj.myString – работает
MyObj.0 – ошибка!
```

Создание объекта синтаксически

```
var myHonda = {
  color: "red",
  wheels: 4,
  engine: {
    cylinders: 4,
    size: 2.2
  }
};
```

Перечисление всех свойств объекта

```
// Цикл for...in – перебирает все перечисляемые свойства объекта и его цепочку прототипов

var string1 = "";
var object1 = {a: 1, b: 2, c: 3};

for (var property1 in object1) {
  string1 += object1[property1];
}

console.log(string1);
// "123"

// Object.keys(o) – метод возвращает массив со всеми собственными (те, что в цепочке прототипов, не войдут в массив) именами перечисляемых свойств объекта o.

// Object.getOwnPropertyNames(o) – метод возвращает массив содержащий все имена своих свойств (перечисляемых и неперечисляемых) объекта o.
```

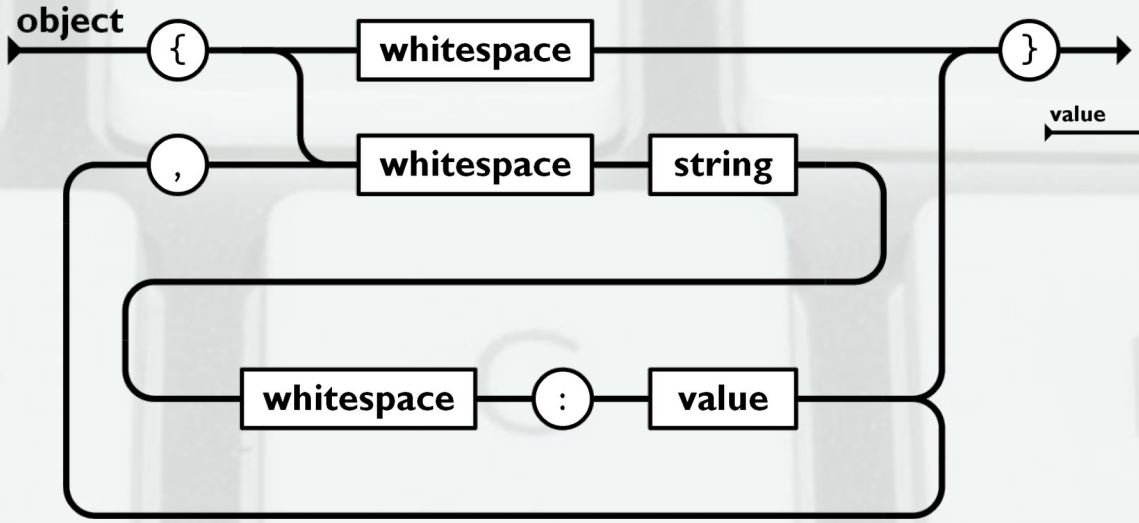
Объекты с зарезервированными словами

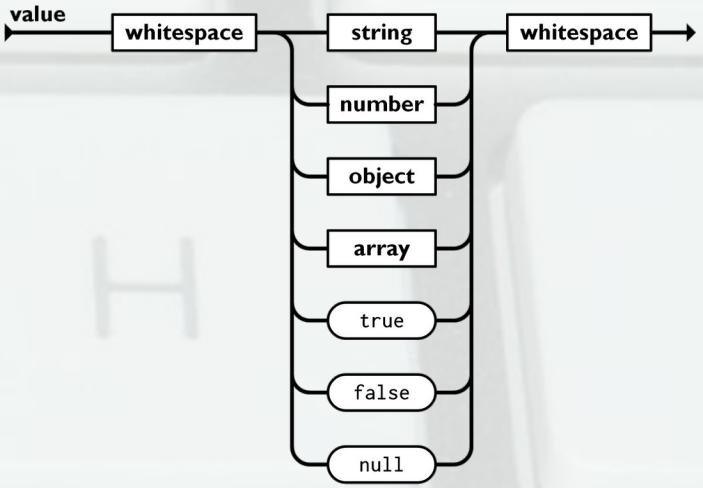
```
var promise = {
  'var'  : 'text',
  'array': [1, 2, 3, 4]
};

// только скобочная нотация
console.log(promise['array']);
```

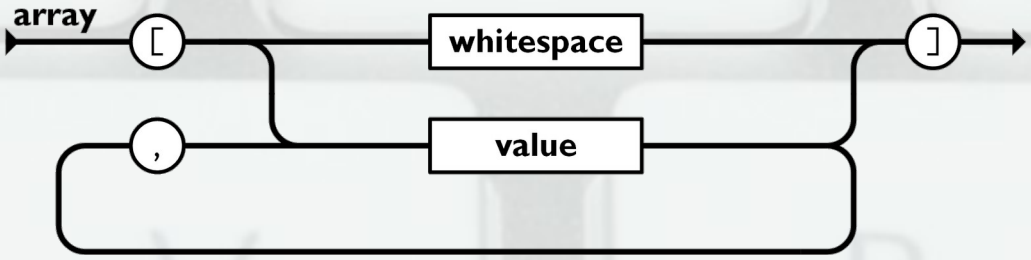
Сортировка объектов по свойству

```
var items = [
  { name: 'Edward', value: 21 },
  { name: 'Sharpe', value: 37 },
  { name: 'And', value: 45 },
  { name: 'The', value: -12 },
  { name: 'Magnetic' },
  { name: 'Zeros', value: 37 }
];
items.sort(function (a, b) {
  if (a.name > b.name) {
    return 1;
  }
  if (a.name < b.name) {
    return -1;
  }
  // a должно быть равным b
  return 0;
});
```

JavaScript Object Notation (JSON)







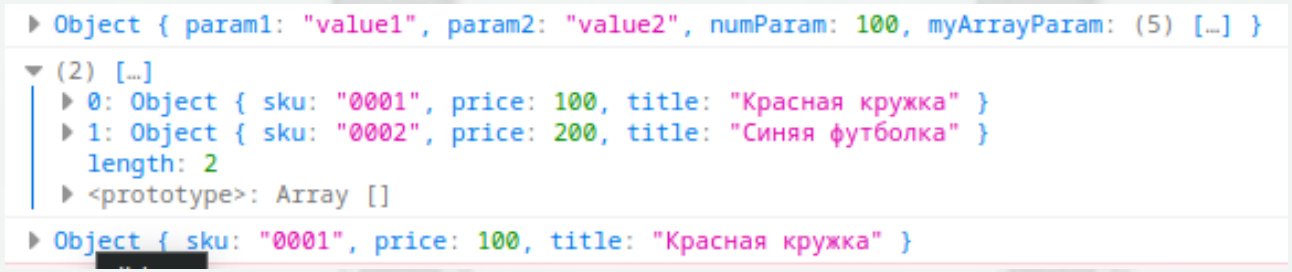
https://www.json.org/
subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999

JSON пример

```
{
    "glossary": {
        "title": "example glossary",
                "GlossDiv": {
                        "title": "S",
                        "GlossList": {
                                "GlossEntry": {
                                        "ID": "SGML",
                                        "SortAs": "SGML",
                                        "GlossTerm": "Standard Generalized Markup Language",
                                        "Acronym": "SGML",
                                        "Abbrev": "ISO 8879:1986",
                                        "GlossDef": {
                                                "para": "A meta-markup language, used to create markup
languages such as DocBook.",
                                                "GlossSeeAlso": ["GML", "XML"]
                                        },
                                        "GlossSee": "markup"
                                }
                        }
                }
        }
}
```

JSON.parse() пример

```
// JSON задает объект JavaScript перечислением полей и значений.
// Может содержать вложенные массивы и объекты.
let params = {
  "param1": "value1",
  "param2": "value2",
  "numParam": 100,
  "myArrayParam": [
    1, 2, 3,
    "string",
    {"obj": []}
  ]
};
console.log(params);
```




```
// Из баз данных или по сети получаем JSON обычно в виде строки и десериализуем с помощью метода JSON.parse()
let stringParams = '[{"sku": "0001", "price": 100, "title": "Красная кружка"}, {"sku": "0002", "price": 200, "title": "Синяя футболка"}]';
params = JSON.parse(stringParams);
console.log(params);
// Распечатываем данные первого товара.
console.log(params[0]);
```

MAP: отображение массива

```
// ECMA-262 в 5-м издании

// Метод map() создаёт новый массив с результатом вызова
указанной функции для каждого элемента массива.

var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
// теперь roots равен [1, 2, 3], а numbers всё ещё равен
[1, 4, 9]

var doubles = numbers.map(function(num) {
  return num * 2;
});
// теперь doubles равен [2, 8, 18], а numbers всё ещё равен
[1, 4, 9]
```

MAP: отображение массива

```
var elems = document.querySelectorAll(
  'select option:checked');

var values = [].map.call(
  elems,
  function(obj) {
    return obj.value;
  });
```

MAP: массив байт в кодировке ASCII из строки

```
var map = Array.prototype.map;

var a = map.call('Hello World',
  function(x) { return x.charCodeAt(0); });

// теперь a равен [72, 101, 108, 108, 111,
32, 87, 111, 114, 108, 100]
```

MAP: переворачивание строки

```
var str = '12345';
[].map.call(str, function(x) {
  return x;
}).reverse().join('');

// Вывод: '54321'
```

MAP: parseInt

```
['1', '2', '3'].map(parseInt);
// Хотя ожидаемый результат вызова равен [1, 2, 3],
// в действительности получаем [1, NaN, NaN]


// Функция parseInt часто используется с одним аргументом, но она принимает два.
// Первый аргумент является выражением, а второй - основанием системы счисления.
// В функцию callback Array.prototype.map передаёт 3 аргумента:
// элемент, его индекс и сам массив.
// Третий аргумент игнорируется parseInt, но не второй, следовательно,
// возможна путаница.


function returnInt(element) {
  return parseInt(element, 10);
}


['1', '2', '3'].map(returnInt);
// Результатом является массив чисел (как и ожидалось)


// Можно так:
['1', '2', '3'].map(Number); // [1, 2, 3]
```

Объект Number

```
new Number(value);
var a = new Number('123'); // a === 123 is false

var b = Number('123'); // b === 123 is true
a instanceof Number; // is true
b instanceof Number; // is false
```

Пример с формой и JS

При смене значения селекта или радиокнопки печатаем value.
При выборе Option 3 необходимо скрывать группу радиокнопок.

```
<form>
  <select name="myselect">
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
  </select>

  <div class="myradios" id="myradios">
    <label><input type="radio" name="myradio" value="r1">R1</label>
    <label><input type="radio" name="myradio" value="r2">R2</label>
    <label><input type="radio" name="myradio" value="r3">R3</label>
  </div>

</form>
```

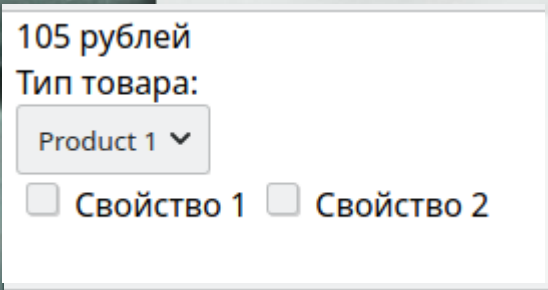
```
<script>
window.addEventListener('DOMContentLoaded', function (event) {
  let s = document.getElementsByName("myselect");
  s[0].addEventListener("change", function(event) {
    let select = event.target;
    let radios = document.getElementById("myradios");
    console.log(select.value);
    // Можно использовать getElementsByClassName()
    if (select.value == "3") {
      radios.style.display = "none";
    }
    else {
      radios.style.display = "block";
    }
  });

  let r = document.querySelectorAll(".myradios input[type=radio]");
  r.forEach(function(radio) {
    radio.addEventListener("change", function(event) {
      let r = event.target;
      console.log(r.value);
    });
  });

});
</script>
```

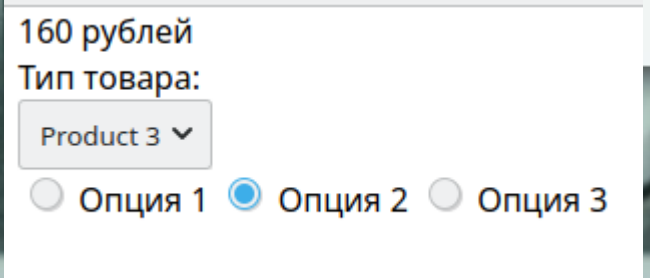
Пример с формой и JS 2

При смене значений селектов или радиокнопок выводим стоимость товара.
Для Product 1 и 2 показывать только чекбоксы, для Product 3 только радиокнопки.
Считать стоимость по параметрам в объекте:

```
{
  prodTypes:
    [100, 200, 150],
  prodOptions: {
    option2: 10,
    option3: 5,
  },
  prodProperties: {
    prop1: 1,
    prop2: 2,
  }
};
```

```
<form>
    <select name="prodType">
      <option value="1">Product 1</option>
      <option value="2">Product 2</option>
      <option value="3">Product 3</option>
    </select>
    <div id="radios">
      <label>
          <input type="radio" name="prodOptions" value="option1">
          Опция 1</label>
      <label>
          <input type="radio" name="prodOptions" value="option2">
          Опция 2</label>
      <label>
          <input type="radio" name="prodOptions" value="option3">
          Опция 3</label>
    </div>
    <div id="checkboxes">
      <label><input type="checkbox" name="prop1"> Свойство 1</label>
      <label><input type="checkbox" name="prop2"> Свойство 2</label>
    </div>
</form>
```





Пример с формой и JS 2

См. код в лабе 6
в каталоге web6.