Содержание

ГЛАВА 6

Массивы

Массив — это упорядоченный набор значений. Каждое значение называется *элементом*, и каждый элемент имеет обозначенную числом позицию в наборе, которая называется *индексом*. Иными словами, индекс — это номер элемента в массиве. Массивы JavaScript *нетипизированные*. Элемент массива может иметь любой тип, а разные элементы одного массива могут иметь разные типы. Элемент массива может быть даже объектом или другим массивом. Это позволяет создавать сложные структуры данных, такие как массивы объектов и многомерные массивы. В JavaScript нумерация элементов массивов начинается с нуля, и применяются 32-битовые целочисленные индексы. Следовательно, первый элемент имеет номер `0`, а максимальный номер элемента равен $2^{32}-2 = 4294967294$. Максимальное количество элементов составляет 4294967295. Массивы JavaScript *динамические*: при необходимости они могут уменьшаться или увеличиваться. Поэтому нет необходимости объявлять фиксированный размер массива при его создании, а при изменении размера нет необходимости повторно выделять память для массива. Каждый массив имеет свойство `length` (длина), возвращающее текущее количество элементов массива.

Массивы JavaScript — это объекты специального вида. Индексы напоминают целочисленные имена свойств, но на самом деле это нечто большее. Различные реализации интерпретаторов по-разному оптимизируют массивы, по-

этому обычно обращение к индексированным элементам массива выполняется намного быстрее, чем к свойствам объекта.

Объекты массивов наследуют свойства от объекта `Array.prototype`, который определяет мощный набор методов манипулирования массивами. Большинство этих методов *обобщенные* (generic). Это означает, что они правильно работают не только с истинными массивами, но и с "массивоподобными" объектами, такими как строки символов.

Создание массива

Легче всего создать массив с помощью литерала массива, который представляет собой заключенный в квадратные скобки список элементов, разделенных запятыми.

```
var empty = [];              // Пустой массив
var primes = [2, 3, 5, 7];   // Массив из четырех чисел
var misc = [{}, true, "a"]; // Элементы разных типов
```

Значения в литерале массива необязательно должны быть константами, но могут быть произвольными выражениями.

```
var base = 1024;
var table = [base, base+1, base+2, base+3];
```

Литерал массива может содержать объектные литералы и литералы других массивов.

```
var b = [[1,{x:1, y:2}], [2, {x:3, y:4}]];
```

Если в литерале массива две запятые находятся рядом без значения между ними, то элемент считается пропущенным, а массив называется *разреженным* (sparse). Пропущенный элемент имеет значение `undefined`.

```
var count = [1,,3];  // Элемент 1 не определен
count[1]             // => undefined
var undefs = [,,];   // Элементов нет, но длина = 2
```

Синтаксис литералов массивов разрешает добавить завершающую запятую. Это означает, что, например, массив `[1,2,]` состоит из двух, а не из трех элементов.

Еще один способ создания массива состоит в применении конструктора `Array()`, который можно вызвать одним из трех способов.

- Вызов без аргументов.

  ```
  var a = new Array();
  ```

- Создается пустой массив без элементов, эквивалентный литералу `[]`.

- Вызов с одним целочисленным аргументом, задающим длину (количество элементов) массива.

  ```
  var a = new Array(10);
  ```

  Будет создан массив заданной длины. Эту форму конструктора можно использовать для создания массива, когда заранее известно максимальное количество элементов. Обратите внимание на то, что в создаваемый массив при этом не записываются никакие значения, а индексные свойства массива `"0"`, `"1"` и последующие не определены.

- Явное задание нескольких элементов массива.

  ```
  var a = new Array(5, 4, 3, 2, 1, "testing");
  ```

  В этой форме аргументы конструктора становятся элементами нового массива.

Важно отметить, что применить литерал массива почти всегда проще, чем конструктор `Array()`.

Элементы и длина массива

Обращение к элементу массива выполняется с помощью оператора []. Ссылка на массив должна находиться слева от квадратных скобок. В квадратных скобках необходимо поместить произвольное выражение, возвращающее (или преобразуемое в) целочисленное положительное значение. Этот синтаксис используется как для чтения, так и для записи значения элемента. Ниже приведены примеры правильных выражений.

```
var a = ["world"];     // Создание массива
var value = a[0];      // Чтение элемента 0
a[1] = 3.14;           // Запись элемента 1
i = 2;
a[i] = 3;              // Запись элемента 2
a[i + 1] = "Привет!"; // Запись элемента 3
a[a[i]] = a[0];        // Чтение и запись элементов
```

Не забывайте, что массивы — это объекты специального вида. Квадратные скобки, используемые для обращения к элементу массива, работают так же, как квадратные скобки, применяемые для обращения к свойству объекта. Интерпретатор преобразует заданные вами числовые индексы массива в строки. Например, индекс 1 становится строкой "1". После этого строка используется в качестве имени свойства.

У каждого массива есть свойство length (длина), причем наличие именно этого свойства отличает массивы от обычных объектов. Свойство length содержит количество элементов массива (предполагается, что опущенных элементов нет). Значение свойства length всегда на единицу больше, чем самый высокий индекс массива.

```
[].length            // => 0: массив не имеет элементов
['a','b','c'].length // => 3: наибольший индекс равен 2
```

Каждый массив является объектом, поэтому можно создать в нем свойство с произвольным именем. Фактически от обычных объектов массивы отличаются только тем, что при использовании имени свойства, являющегося целочисленным значением, меньшим $2^{32}-1$ (или преобразуемого в него), массив автоматически создает и заполняет свойство `length`.

Свойство `length` доступно для записи. Если присвоить ему положительное целое значение `n`, меньшее текущего значения, то все элементы массива с индексом, бо́льшим или равным `n`, будут удалены.

```
a=[1,2,3,4,5]; // Создание массива из 5 элементов
a.length = 3;  // Теперь массив такой: [1,2,3]
a.length = 0;  // Удаление всех элементов
a.length = 5;  // Опять 5 элементов, но пустых
```

Можно также присвоить свойству `length` значение, большее, чем текущая длина массива. В результате будут созданы пустые элементы, и массив станет разреженным.

Перечисление элементов массива

Пройти по элементам массива проще всего с помощью цикла `for` (см. главу 4).

```
var keys = Object.keys(o); // Массив имен свойств
var values = []            // Массив для значений свойств
for(var i = 0; i < keys.length; i++) {
var key = keys[i];         // Получение имен свойств
values[i] = o[key];        // Сохранение значений
}
```

Во вложенных циклах и в других ситуациях, в которых важна производительность, рекомендуется извлечь длину массива один раз перед началом цикла, чтобы не извлекать ее на каждой итерации.

```
for(var i = 0, len = keys.length; i < len; i++) {
  // Тело цикла
}
```

В ECMAScript 5 определен ряд новых методов, предназначенных для прохода по элементам массива путем передачи методу каждого элемента в порядке возрастания индексов функции, определяемой в коде. Наиболее общий из этих методов — `forEach()`.

```
var data = [1,2,3,4,5];  // Создание массива
var sumOfSquares = 0;    // Переменная-накопитель
data.forEach(function(x) {
   sumOfSquares += x*x;  // Накопление квадратов
          });
sumOfSquares             // =>55: 1+4+9+16+25
```

Многомерные массивы

Спецификация JavaScript не поддерживает истинные многомерные массивы, но их можно успешно имитировать, создавая массивы массивов. Для обращения к значению в массиве массива нужно всего лишь написать оператор `[]` два раза подряд. Предположим, что `matrix` — это массив массивов, содержащий числа. Для обращения к отдельному числу в этом массиве нужно написать `matrix[x][y]`. В приведенном ниже примере двухмерный массив используется для создания таблицы умножения, хорошо знакомой вам еще со школы.

```
// Создание многомерного массива
var table = new Array(10); // 10 строк таблицы
for(var i = 0; i < table.length; i++)
  table[i] = new Array(10); // 10 столбцов

// Инициализация массива
for(var row = 0; row < table.length; row++) {
```

```
    for(col = 0; col < table[row].length; col++) {
      table[row][col] = row*col;
    }
  }

  // Извлечение значения 5*7 из таблицы умножения
  var product = table[5][7]; // => 35
```

Методы массивов

С объектами массивов ассоциированы многие полезные методы, рассматриваемые в данном разделе.

join()

Метод `Array.join()` преобразует все элементы массива в строки, выполняет их конкатенацию и возвращает полученную таким образом строку. Можно задать строку-разделитель, отделяющую элементы массива один от другого в результирующей строке. Если разделитель не задан, в этом качестве используется запятая.

```
var a = [1, 2, 3];
a.join(); // => "1,2,3"
a.join(" "); // => "1 2 3"
a.join(""); // => "123"
var b = new Array(5);
b.join('-') // => '----'
```

Метод `String.split()` выполняет обратную операцию: он создает массив, разбив полученную строку на части.

reverse()

Метод `Array.reverse()` изменяет последовательность элементов массива на обратную и возвращает "реверсированный" массив. Эта операция выполняется "на месте", т.е. метод не создает еще один массив с переупорядочен-

ными элементами, а переупорядочивает элементы в существующем массиве.

```
var a = [1,2,3];
a.reverse().join() // => "3,2,1"
a[0] // => 3: теперь [3,2,1]
```

sort()

Метод Array.sort() сортирует элементы массива "на месте" и возвращает отсортированный массив. Когда sort() вызван без аргументов, он сортирует элементы массива в алфавитном порядке.

```
var a = new Array("вишня", "яблоко", "апельсин");
a.sort();
var s = a.join(", ");
            // s == "апельсин, вишня, яблоко"
```

Если в массиве есть неопределенные элементы, то в процессе сортировки они перемещаются в конец массива.

Чтобы отсортировать массив в порядке, отличном от алфавитного, необходимо передать методу sort() функцию сравнения, которая определяет, какой элемент массива из двух полученных должен быть первым в отсортированной последовательности. Если первый аргумент должен находиться в отсортированной последовательности раньше второго, функция сравнения возвращает отрицательное число. Если же первый аргумент должен находиться после второго, она возвращает положительное число. Если два аргумента эквивалентны (т.е. их последовательность не играет роли), функция сравнения должна возвращать нуль. Например, чтобы отсортировать элементы массива не в алфавитном, а в числовом порядке, можно написать следующий код.

```
var a = [33, 4, 1111, 222];
```

```
a.sort();       // В алфавитном порядке: 1111,222,33,4
a.sort(function(a,b) {
                // В числовом порядке: 4,33,222,1111
   return a-b;  // Возвращает <0, 0, or >0
  });
a.sort(function(a,b) {return b-a});
                    // Обратный числовой порядок
```

Выполнить сортировку в алфавитном порядке, не чувствительном к регистру, можно следующим образом.

```
a = ['жук', 'Лиса', 'кот']
a.sort();
 // Алфавитный порядок, чувствительный к регистру:
             ['Лиса','жук',кот']
a.sort(function(s,t) {
  var a = s.toLowerCase();
  var b = t.toLowerCase();
  if (a < b) return -1;
  if (a > b) return 1;
  return 0;
}); // => ['жук','кот','Лиса']
```

concat()

Метод `Array.concat()` создает и возвращает новый массив, содержащий элементы исходного массива и значения аргументов, заданные при вызове. Если каждый из аргументов является массивом, то элементами результирующего массива становятся элементы массивов аргументов, а не массивы аргументов. Метод `concat()` не изменяет массив, через который он вызван. Ниже приведен ряд примеров.

```
var a = [1,2,3];
a.concat(4, 5)         // Вернул [1,2,3,4,5]
a.concat([4,5]);       // Вернул [1,2,3,4,5]
a.concat([4,5],[6,7])  // Вернул [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]]) // Вернул [1,2,3,4,5,[6,7]]
```

slice()

Метод Array.slice() возвращает фрагмент указанного массива. Два аргумента задают начало и конец возвращаемого фрагмента. Результирующий массив содержит все элементы, начиная с заданного первым аргументом, вплоть до элемента, заданного вторым аргументом, но не включая его. Если передается только один аргумент, метод возвращает все элементы, начиная с указанной позиции до конца массива. Если один из аргументов отрицательный, он задает отсчет элементов массива, начиная с конца. Не забывайте, что метод slice() не изменяет массив, через который он вызван, а возвращает в качестве результата новый массив.

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

splice()

Метод Array.splice() вставляет новый или удаляет существующий элемент массива. В отличие от slice() и concat(), этот метод изменяет массив, через который он вызван.

Первый аргумент splice() задает позицию в массиве, в которой нужно начать вставку или удаление элементов. Второй аргумент задает количество элементов, удаляемых из массива. Если второй аргумент опущен, удаляются все элементы, начиная с указанного первым аргументом, вплоть до конца массива. Метод возвращает массив удаленных элементов или пустой массив (если ни один элемент не удален).

```
var a = [1,2,3,4,5,6,7,8];
a.splice(4);    // Возвращает [5,6,7,8];
```

```
                // массив a теперь равен [1,2,3,4]
a.splice(1,2); // Возвращает [2,3];
                // массив a теперь равен [1,4]
a.splice(1,1); // Возвращает [4];
                // массив a теперь равен [1]
```

Первые два аргумента задают, какие элементы массива должны быть удалены. После них могут находиться дополнительные аргументы, которые задают элементы, вставляемые в массив, начиная с позиции, заданной первым аргументом.

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

Обратите внимание на то, что в отличие от `concat()` метод `splice()` вставляет массивы, а не их элементы.

push() и pop()

Методы `push()` и `pop()` позволяют работать с массивами таким образом, как будто это стеки. Метод `push()` присоединяет один или несколько элементов к концу стека (т.е. массива) и возвращает новую длину массива. Метод `pop()` выполняет обратную операцию: удаляет последний элемент стека, уменьшает длину массива и возвращает удаленное значение. Оба метода изменяют массив "на месте", не создавая измененную копию массива.

```
var stack = [];     // stack: []
stack.push(1,2);    // stack: [1,2], вернул 2
stack.pop();        // stack: [1], вернул 2
stack.push(3);      // stack: [1,3], вернул 2
stack.pop();        // stack: [1], вернул 3
stack.push([4,5]); // stack: [1,[4,5]], вернул 2
stack.pop()         // stack: [1], вернул [4,5]
stack.pop();        // stack: [], вернул 1
```

unshift() и shift()

Методы unshift() и shift() работают аналогично методам push() и pop(), но вставляют и удаляют элементы в начале, а не в конце массива. Метод unshift() вставляет элемент или элементы в начало массива и сдвигает существующие элементы в сторону более высоких индексов, чтобы освободить место для вставляемых элементов. Этот метод возвращает длину нового массива. Метод shift() удаляет первый элемент массива, сдвигает оставшиеся элементы влево и возвращает удаленный элемент. Ниже приведен ряд примеров.

```
var a = [];          // a:[]
a.unshift(1);        // a:[1], вернул 1
a.unshift(22);       // a:[22,1], вернул 2
a.shift();           // a:[1], вернул 22
a.unshift(3,[4,5]); // a:[3,[4,5],1], вернул 3
a.shift();           // a:[[4,5],1], вернул 3
a.shift();           // a:[1], вернул [4,5]
a.shift();           // a:[], вернул 1
```

toString()

Массив, как и любой объект JavaScript, имеет метод toString(). В случае массива этот метод преобразует каждый элемент в строку (вызвав метод toString() этого элемента) и возвращает список полученных строк, разделенных запятыми. Обратите внимание на то, что результирующая строка не содержит квадратных скобок или каких-либо иных разделителей значений элементов, кроме запятых.

```
[1,2,3].toString()          // => '1,2,3'
["a", "b", "c"].toString() // => 'a,b,c'
[1, [2,'c']].toString()     // => '1,2,c'
```

Методы массивов ECMAScript 5

В ECMAScript 5 определены девять новых методов массивов для прохода по элементам, преобразования, фильтрации, проверки, свертки и поиска. Большинство из этих методов принимают функцию в качестве первого аргумента и вызывают ее по одному разу для каждого (или по крайней мере для некоторых) элемента массива. В большинстве случаев предоставляемая вами функция вызывается с тремя аргументами: значением элемента массива, индексом элемента массива и самим массивом. Иногда необходим только первый аргумент, а второй и третий игнорируются. Большинство методов массивов, принимающих функцию через первый аргумент, принимают также необязательный второй аргумент. Если он задан, функция вызывается таким образом, будто она является методом второго аргумента. Таким образом, второй передаваемый вами аргумент становится в теле функции значением `this`. Возвращаемое значение функции может по-разному использоваться разными методами. Ни один из методов массивов, определенных в ECMAScript 5, не изменяет массив, через который он вызван, но функция, передаваемая методу массива может изменять массив.

forEach()

Метод `forEach()` проходит по массиву, вызывая заданную функцию для каждого элемента.

```
var data = [1,2,3,4,5]; // Сумма элементов
var sum = 0;            // Начинаем с нуля
data.forEach(function(value) { sum += value; });
sum // => 15

// Увеличение каждого элемента
data.forEach(function(v, i, a) { a[i] = v + 1; });
data // => [2,3,4,5,6]
```

map()

Метод map() передает заданной функции каждый элемент массива, через который он вызван, и возвращает новый массив, содержащий значения, возвращенные функцией. Следовательно, этот метод преобразует каждый элемент массива согласно заданному алгоритму.

```
a = [1, 2, 3];
b = a.map(function(x) { return x*x; });
// Теперь массив b равен [1, 4, 9]
```

filter()

Метод filter() возвращает массив, содержащий подмножество элементов исходного массива, через который он вызван. Передаваемая методу функция должна возвращать значение true или false. Если функция, получив элемент, возвращает true или значение, преобразуемое в true, то данный элемент включается в результирующее подмножество и добавляется в массив, который станет возвращаемым значением метода.

```
a = [5, 4, 3, 2, 1];
a.filter(function(x) { return x < 3 });
                                // => [2,1]
a.filter(function(x,i) { return i%2==0 });
                                // => [5,3,1]
```

every() и some()

Методы every() и some() являются предикатами массива: они применяют заданную функцию к элементам массива и возвращают значение true или false.

Метод every() напоминает квантор всеобщности ∀: он возвращает true, если заданная функция вернула true для каждого элемента массива.

```
a = [1,2,3,4,5];
// Все ли значения меньше 10?
a.every(function(x) { return x < 10; }) //=>true
// Все ли значения четные?
a.every(function(x) {return x%2 === 0;}) //=>false
```

Метод `some()` напоминает квантор существования ∃: он возвращает `true`, если существует хотя бы один элемент массива, для которого заданная функция возвращает `true`. Значение `false` будет возвращено, если заданная функция вернет `false` для всех элементов массива.

```
a = [1,2,3,4,5];
// Есть ли четное число?
a.some(function(x) { return x%2===0; }) //=> true
// Есть ли элемент, не являющийся числом?
a.some(isNaN) //=> false
```

Обратите внимание на то, что методы `every()` и `some()` прекращают проход по элементам массива, как только становится известным возвращаемое значение. При вызове через пустой массив метод `every()` возвращает значение `true`, а метод `some()` — значение `false`.

reduce() и reduceRight()

Методы `reduce()` и `reduceRight()` объединяют элементы массива с помощью заданной функции, возвращая единственное значение. В функциональном программировании это часто используемая операция, которую обычно называют *сверткой* (reduction).

```
var a = [1,2,3,4,5]
// Вычисление суммы элементов
a.reduce(function(x,y) { return x+y }, 0); // =>15
// Вычисление произведения элементов
a.reduce(function(x,y) { return x*y }, 1); // =>120
// Поиск наибольшего элемента
a.reduce(function(x,y) { return (x>y)?x:y; });
                                          // =>5
```

Метод `reduce()` принимает два аргумента. Первый — функция, выполняющая операцию свертки. Задача свертки состоит в том, чтобы каким-либо образом объединить, или свернуть, два значения в одно. Эта функция должна возвращать свернутое значение. В приведенных выше примерах функция выполняет свертку двух значений путем их суммирования, умножения или выбора большего значения. Второй (необязательный) аргумент — начальное значение, передаваемое функции.

Функции свертки, используемые в методе `reduce()`, существенно отличаются от функций, используемых в `forEach()` и `map()`. Значение элемента, индекс и массив сдвигаются и передаются через второй, третий и четвертый аргументы. В первом аргументе хранится аккумулированный результат процесса свертки. При первом вызове функции первый аргумент содержит начальное значение, передаваемое через второй аргумент методу `reduce()`. При следующих вызовах первый аргумент получает значения, получаемые при предыдущих вызовах. Например, при суммировании функция свертки в первый раз вызывается с аргументами `0` и `1`. Функция суммирует их и возвращает `1`. При втором вызове функция получает `1` и `2` и возвращает `3` и т.д. И наконец, окончательная сумма, равная `15`, возвращается методом `reduce()`.

Обратите внимание на то, что в третьем примере, посвященном поиску наибольшего элемента, передается только один аргумент, а начальное значение не задается. Если вызвать метод `reduce()` без начального значения, он применит в этом качестве первый элемент массива. Следовательно, при первом вызове первым и вторым аргументами будут считаться первый и второй элементы массива. Заметьте, что в примерах с суммированием и перемножением также можно было опустить начальные значения.

Метод reduceRight() работает так же, как reduce(), за исключением того, что обработка массива в нем выполняется наоборот: начинается с верхних индексов и продвигается к нижним.

indexOf() и lastIndexOf()

Методы indexOf() и lastIndexOf() ищут в массиве элемент с заданным значением и возвращают индекс его первого вхождения или -1, если он не найден. Метод indexOf() проходит массив от начала до конца, а метод lastIndexOf() — от конца к началу.

```
a = [0,1,2,1,0];
a.indexOf(1) // => 1: a[1] равен 1
a.lastIndexOf(1) // => 3: a[3] равен 1
a.indexOf(3) // => -1: такого элемента нет
```

В отличие от других методов, описанных в данном разделе, методы indexOf() и lastIndexOf() не принимают функцию в качестве аргумента. В них первый аргумент — искомое значение, а второй не обязателен: он задает индекс, начиная с которого следует выполнить поиск. Если он опущен, метод indexOf() начинает поиск с начала, а lastIndexOf() — с конца. Второй аргумент может быть отрицательным; в этом случае поиск начинается с другого конца, т.е. в первом случае — с конца, а во втором — с начала.

Тип Array

Выше уже упоминалось о том, что массивы являются объектами специального вида. Когда в программе встречается неизвестный объект (будем надеяться, не летающий), иногда бывает полезно выяснить, массив ли это. В реализациях ECMAScript 5 это можно сделать с помощью функции Array.isArray().

```
Array.isArray([]) // => true
Array.isArray({}) // => false
```

В любой другой версии JavaScript это можно сделать с помощью следующей функции.

```
var isArray = Array.isArray || function(o) {
  var ts = Object.prototype.toString;
  return typeof o === "object" &&
         ts.call(o) === "[object Array]";
};
```

"Массивоподобные" объекты

Как указывалось выше, массивы — это объекты, имеющие специальное свойство `length` (длина). "Массивоподобный" объект — это обычный объект JavaScript со свойством `length` и числовыми свойствами, играющими роль индексов. На практике массивоподобные объекты встречаются редко (за исключением строк, которые тоже считаются массивоподобными объектами). Вызывать через них методы массивов нельзя. Кроме того, со свойством `length` не ассоциировано специальное поведение. В данном случае это обычное свойство. Тем не менее по элементам массивоподобного объекта можно проходить в цикле так же, как по элементам настоящего массива.

```
// Создание массивоподобного объекта
var a = {"0":"a", "1":"b", "2":"c", length:3};
// Проход по этому объекту как по массиву
var total = 0;
for(var i = 0; i < a.length; i++)
      total += a[i];
```

Многие алгоритмы массивов работают с массивоподобными объектами так же, как и с настоящими массивами, а многие методы массивов JavaScript специально

определены как обобщенные (generic) и могут правильно обрабатывать массивоподобные объекты. Эти объекты не наследуют прототип `Array.prototype`, поэтому вызывать методы массивов непосредственно через них нельзя. Однако их можно вызвать с помощью метода `call()` следующим образом (подробнее об этом — в главе 7).

```
// Создание массивоподобного объекта
var a = {"0":"a", "1":"b", "2":"c", length:3};
Array.prototype.join.call(a, "+") // => "a+b+c"
Array.prototype.map.call(a, function(x) {
       return x.toUpperCase();
})                          // => ["A","B","C"]
// Создание настоящего массива на основе
// массивоподобного объекта
Array.prototype.slice.call(a, 0) // => ["a","b","c"]
```

Некоторые браузеры определяют обобщенные функции массивов непосредственно через конструктор `Array`. В браузерах, поддерживающих эти функции, можно использовать код следующего вида.

```
var a = {"0":"a", "1":"b", "2":"c", length:3};
Array.join(a, "+")
Array.slice(a, 0)
Array.map(a,
     function(x) { return x.toUpperCase(); })
```

Строки в качестве массивов

В реализациях ECMAScript 5 (и многих браузерах предыдущих версий, включая IE8, выпущенных до появления ECMAScript 5) строки ведут себя так же, как массивы в режиме "только чтение". Это означает, что для обращения к отдельным символам строки можно использовать не только метод `charAt()`, но и квадратные скобки.

```
var s = test;
s.charAt(0) // => "t"
s[1]        // => "e"
```

Тем не менее оператор `typeof` возвращает значение `string`, а метод `Array.isArray()`, получивший имя строки, возвращает значение `false`. Поэтому отличить строку от массива несложно.

Главное преимущество индексации строк состоит всего лишь в том, что можно заменить вызовы метода `charAt()` выражениями с квадратными скобками, которые делают программу более компактной и легкой для визуального восприятия. Кроме того, тот факт, что строки ведут себя, как массивы, означает, что к ним можно применять обобщенные методы массивов, как в следующем примере. Приведенный ниже код удаляет из строки гласные буквы.

```
s = "Java"
Array.prototype.join.call(s, " ") // => "J a v a"
Array.prototype.filter.call(s, function(x) {
      return x.match(/[^aeiou]/); // Поиск согласных букв
   }).join("")          // => "Jv"
```