JS Arrow Functions & JavaScript Arrays Methods

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#instance_methods

JavaScript: The Definitive Guide, Seventh Edition, by David Flanagan

Arrow Function Expressions

A compact alternative to a traditional function expression but is limited and can't be used in all situations.

- Arrow functions don't have their own bindings to `this`, `argument`, or `super` and should not be used as *(class or constructor)* `methods`.
- Arrow functions cannot be used as `constructors`. Calling them with `new` throws a TypeError.

```
// Traditional Function (no arguments)
let a = 4
let b = 2
function (){
   return a + b + 100
}

// Arrow Function
let a = 4
let b = 2
() => a + b + 100
```

```
//No param
() => expression

// One param
param => expression

// Multiple param
(param1, paramN) => expression

// Multiline statements
param1 => {
       statement1;
       …
       statementN;
}

(param1, paramN) => {
       statement1;
       …
       statementN;
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Comparing traditional functions to arrow functions

```
// Traditional Function (one argument)
function  (a){
    return a + 100
}
// Arrow Function Break Down
// 1. Remove the word "function" and place arrow
between the argument and opening body bracket
(a)=> {
    return a + 100
}

// 2. Remove the body braces and word "return" --
the return is implied.
(a)=> a + 100

// 3. Remove the argument parentheses
a => a + 100
```

```
// Traditional Function (multiple arguments)
function (a, b){
        return a + b + 100
}

// Arrow Function
(a, b) => a + b + 100
```

```
// Traditional Function (multiline statements)
function (a, b){
        let chuck=42
        return a + b + chuck
}

// Arrow Function
(a, b) => {
        let chuck=42
        return a + b + chuck
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Function Types

- An **anonymous function** is a function without a function name. Only function expressions can be anonymous, function declarations must have a name

```
// Anonymous function created as a function expression
function () {}

// Anonymous function created as an arrow function
() => {}
```

- A **named function** is a function with a function name

```
// Function declaration
function foo() {}

// Named function expression
Const barFn=function bar() {}

// Arrow function
const barAF = () => {}
```

https://developer.mozilla.org/en-US/docs/Glossary/Function

The way to create and pass a callback function to other functions

A function that is passed to another function as a parameter is a callback function.

1. Arrow function

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

2. Named function

```
function isMorethanFive(value) {   return value > 5 }
const filterNums = [12, 5, 8, 130, 44].filter(isMorethanFive);
// filterNums is [12, 8, 130, 44]
```

3. Anonymous function

```
const nums = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
const primeNums =  nums.filter(function (num) {
    for (let i = 2; num > i; i++) {
      if (num % i === 0) {
        return false
      }}
    return num > 1
  })
// primeNums is [ 2, 3, 5, 7, 11, 13 ]
```

Array Iterator Methods

Array Iterator Methods: Iterator methods loop over the elements of an array

- **forEach()** iterates through an array, invoking a function you specify for each element
- **map()** passes each element of the array on which it is invoked to the function you specify and returns an array containing the values returned by your function.
- **filter()** returns an array containing a subset of the elements of the array on which it is invoked.
- **find()** returns the matching element, If no matching element is found, find() returns undefined
- **findIndex()** returns the index of the matching element. If no matching element is found, If no matching element is found, find() returns -1
- **every() and some()** they apply a predicate function you specify to the elements of the array, then return true or false.
- **reduce()** combine the elements of an array, using the function you specify, to produce a single value.

Stack and Queue Methods

Stack and queue methods add and remove array elements to and from the beginning and the end of an array.

- **push()** appends one or more new elements to the end of an array and returns the new length of the array.
  pop() deletes the last element of an array, decrements the array length, and returns the value that it removed.
- **unshift()** adds an element or elements to the beginning of the array, shifts the existing array elements up to higher indexes to make room, and returns the new length of the array.
- **shift()** removes and returns the first element of the array, shifting all subsequent elements down one place to occupy the newly vacant space at the start of the array.

Subarray Methods

Subarray methods are for extracting, deleting, inserting, filling, and copying contiguous regions of a larger array.

- **slice()** returns a slice, or subarray, of the specified array. Its two arguments specify the start and end of the slice to be returned.
- **splice()** a general-purpose method for inserting or removing elements from an array.
- **fill()** sets the elements of an array, or a slice of an array, to a specified value. It mutates the array it is called on, and also returns the modified array:

Searching and Sorting Methods

Searching and sorting methods are for locating elements within an array and for sorting the elements of an array.

- **indexOf()** search an array for an element with a specified value and return the index of the first such element found, or -1 if none is found.
- **includes()** takes a single argument and returns true if the array contains that value or false otherwise. It does not tell you the index of the value, only whether it exists.
- **sort()** sorts the elements of an array in place and returns the sorted array.
- **reverse()** reverses the order of the elements of an array and returns the reversed array.

Array to String Conversions

Array to String Conversions

- **join()** converts all the elements of an array to strings and concatenates them, returning the resulting string.

Array Iterator Methods

Array Methods

- **`forEach()`** method iterates through an array, invoking a function you specify for each element.

```
let letters = [...'hello world']
let uppercase = ''
letters.forEach((letter) => {
    uppercase += letter.toUpperCase()
})
console.log(`uppercase: ${uppercase}`) // "HELLO WORLD"
```

```
const products = [
  { id: 123, name: 'bag' },
  { id: 2, name: 'pen' },
  { id: 33, name: 'BAG' }
]

products.forEach((product) => console.log(product.id))//123 2 33
```

Array Methods

- **`filter()`** creates a new array with all elements that pass the test implemented by the provided function.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present']
const result = words.filter(word => word.length > 6)
console.log(result)
//["exuberant", "destruction", "present"]
```

- **`map()`** creates a new array populated with the results of calling a provided function on every element in the calling array.

```
const array2 = [1, 4, 9, 16]
const map1 = array2.map(x => x * 2)
console.log(map1)
// [2, 8, 18, 32]
```

```
const products = [
  { id: 123, name: 'bag' },
  { id: 2, name: 'pen' },
  { id: 33, name: 'BAG' }
]
const map2 = products.map((product) => product.name)
//[('bag', 'pen', 'BAG')]
```

Array Methods

- **`find ()`** returns the **value of the first element** in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.

```
const array6 = [5, 12, 8, 130, 44]
const found = array6.find(element => element > 10)
console.log(found) //12
```

- **`findIndex()`** returns the **index of the first element** in the array that satisfies the provided testing function. Otherwise, it returns -1, indicating that no element passed the test.

```
const array7 = [5, 12, 8, 130, 44]
console.log(array7.findIndex(element => element > 13))
//3
```

Array Methods

- **`every()`** method tests whether **all elements** in the array pass the test implemented by the provided function. It returns a Boolean value.

```
const isBelowThreshold = (currentValue) => currentValue < 40
const array3 = [1, 30, 39, 29, 10, 13]
console.log(`array3.every(isBelowThreshold): ${array3.every(isBelowThreshold)}`) //true
```

```
console.log(`array3.some(isBelowThreshold):${array3.some(isBelowThreshold)}`) //true
```

- **`some()`** method tests whether **at least one element** in the array passes the test implemented by the provided function.

```
const array = [1, 2, 3, 4, 5] // checks whether an element is even
const even = (element) => element % 2 === 0
console.log(`array.some(even):${array.some(even)}`) // true
```

```
console.log(`array.every(even):${array.every(even)}`) // false
```

Array Methods

- **`reduce()`** combine the elements of an array, using the function you specify, to produce a single value.

```
const array1 = [1, 2, 3, 4]
const reducer = (total, currentValue) => total + currentValue
// 1 + 2 + 3 + 4
console.log(`array1.reduce(reducer): ${array1.reduce(reducer)}`)// 10
```

```
const nums = [5, 4, 100, -1, 0, 2, 19]
const maxNum = nums.reduce((max, num) => Math.max(max, num)) //100
```

```
const product = [1, 'RED', 102]
const productDetail = product.reduce((detail, prop) => {
  return prop === 'RED' ? detail?.concat('R') : detail?.concat(prop)
}, ‘’) //1R02
```

Stack and Queue methods

Array Methods

- **`push()`** method adds one or more elements to the end of an array and returns the new length of the array.

```
const animals = ['pigs', 'goats', 'sheep']
const count = animals.push('cows')
console.log(count)// 4
console.log(animals)// ["pigs", "goats", "sheep", "cows"]

animals.push('chickens', 'cats', 'dogs')
console.log(animals)
// ["pigs", "goats", "sheep", "cows", "chickens", "cats", "dogs"]
```

- **`pop()`** method removes the last element from an array and returns that element. This method changes the length of the array.

```
console.log(animals.pop()) //dogs
console.log(animals)
// ["pigs", "goats", "sheep", "cows", "chickens", "cats"]
console.log(`length: ${animals.length}`) //6
```

Array Methods

- **`shift()`** removes the first element from an array and returns that removed element**.** This method changes the length of the array.

```
const array4 = [1, 2, 3]
const firstElement = array4.shift()
console.log(array4)// [2, 3]
console.log(firstElement)// 1
```

- **`unshift()`** method adds one or more elements to the beginning of an array and returns the new length of the array.

```
const array5 = [1, 2, 3]
console.log(array5.unshift(4, 5))// 5
console.log(array5)// [4, 5, 1, 2, 3]
```

Subarray Methods

Array Methods

```
let arrDeletedItems = arr.splice(start[, deleteCount[, item1[, item2[, ...]]]])
```

- **`splice()`** changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

```
const months = ['Jan', 'March', 'April', 'June']
months.splice(1, 0, 'Feb')
// inserts at index 1
console.log(months)
// ["Jan", "Feb", "March", "April", "June"]
months.splice(4, 1, 'May')
// replaces 1 element at index 4
console.log(months)
// ["Jan", "Feb", "March", "April", "May"]

let myFish = ['angel', 'clown', 'drum', 'mandarin', 'sturgeon']
let removed = myFish.splice(3, 1)
// myFish is ["angel", "clown", "drum", "sturgeon"]
// removed is ["mandarin"]
```

`start` The index at which to start changing the array.

`deleteCount` is 0 or negative, no elements are removed.

Array Methods

- **`slice()`** returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array.
- A negative index can be used, indicating an offset from the end of the sequence.

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant']
console.log(animals.slice(2))
// ["camel", "duck", "elephant"]
console.log(animals.slice(2, 4))
// ["camel", "duck"]
console.log(animals.slice(1, 5))
// ["bison", "camel", "duck", "elephant"]
console.log(animals.slice(-2))
// ["duck", "elephant"]
console.log(animals.slice(2, -1))
// ["camel", "duck"]
console.log(animals.slice())
// ["ant", "bison", "camel", "duck", "elephant"]
```

Array Methods

- **`fill()`** method changes all elements in an array to a static value, from a start index (default 0) to an end index (default array.length). It returns the modified array.

```
const array8 = [1, 2, 3, 4]

// fill with 0 from position 2 until position 3
console.log(`array8.fill(0, 2, 4): ${array8.fill(0, 2, 4)}`)
// [1, 2, 0, 0]

// fill with 5 from position 1
console.log(`array8.fill(5, 1): ${array8.fill(5, 1)}`)
// [1, 5, 5, 5]

console.log(`array8.fill(6): ${array8.fill(6)}`)
// [6, 6, 6, 6]
```

Searching and Sorting Methods

Array Methods

- **`includes()`** method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

```
const array9 = [1, 2, 3]
console.log(array9.includes(2))      // true
const pets = ['cat', 'dog', 'bat']
console.log(pets.includes('cat'))  // true
console.log(pets.includes('at'))  //  false
```

When comparing strings and characters, includes() is *case-sensitive*.

Array Methods

- **`reverse ()`** method reverses an array in place. The first array element becomes the last, and the last array element becomes the first.

```
const array10 = ['one', 'two', 'three']
console.log('array10:', array10)
//  ["one", "two", "three"]

const reversed = array10.reverse()
console.log('reversed:', reversed)
//  ["three", "two", "one"]
```

Array Methods

- **`sort()`** method sorts the elements of an array in place and returns the sorted array. The default sort order is ascending, **built upon converting the elements into strings**, then comparing their sequences of UTF-16 code units values.

```
const months2 = ['March', 'jan', 'feb', 'Dec']
months2.sort()
console.log(`months2: ${months2}`)
// expected output: Array [Dec,March,feb,jan]

const array14 = [1, 30, 4, 21, 100000]
array14.sort()
console.log(`array14: ${array14}`)
// expected output: Array [1, 100000, 21, 30, 4]
```

```
const products = [
   { id: 123, name: 'bag' },  { id: 2, name: 'pen' },   { id: 33, name: 'mouse' }
]
const sortByProductId = products.sort((a, b) =>a.id - b.id)
```

Array Methods

- **`concat ()`** method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

```
const array11 = ['a', 'b', 'c']
const array12 = ['d', 'e', 'f']
const array13 = array11.concat(array12)

console.log(array13)
["a", "b", "c", "d", "e", "f"]
```

- **`indexOf()`** method returns the first index at which a given element can be found in the array, or -1 if it is not present.

```
const beasts = ['ant', 'bison', 'camel', 'duck', 'bison']
console.log(beasts.indexOf('bison'))// 1
// start from index 2
console.log(beasts.indexOf('bison', 2))// 4
console.log(beasts.indexOf('giraffe'))// -1
```

Array to String Conversions

Array Methods

- **`join()`** method creates and returns a new string by concatenating all of the elements in an array (or an array-like object), separated by commas or a specified separator string. If omitted, the array elements are separated with a comma (",").

```
const elements = ['Fire', 'Air', 'Water']
console.log(elements.join())
// "Fire,Air,Water‶
console.log(elements.join(' '))
// Fire Air Water
```