Array.prototype.copyWithin()

Related Topics
Standard built-in objects
Array
Properties
- Array.length
- Array.prototype[@@unscopables]

Methods
- Array.from()
- Array.isArray()
- Array.observe()
- Array.of()
- Array.prototype.concat()
- *Array.prototype.copyWithin()*
- Array.prototype.entries()
- Array.prototype.every()
- Array.prototype.fill()
- Array.prototype.filter()
- Array.prototype.find()

The `copyWithin()` method shallow copies part of an array to another location in the same array and returns it without modifying its length.

JavaScript Demo: Array.copyWithin()

```
const array1 = ['a', 'b', 'c', 'd', 'e'];

// copy to index 0 the element at index 3
console.log(array1.copyWithin(0, 3, 4));
// expected output: Array ["d", "b", "c", "d", "e"]

// copy to index 1 all elements from index 3 to the end
console.log(array1.copyWithin(1, 3));
// expected output: Array ["d", "d", "e", "d", "e"]
```

Run ›
Reset

```
> Array ["d", "b", "c", "d", "e"]
> Array ["d", "d", "e", "d", "e"]
```

Syntax

```
arr.copyWithin(target[, start[, end]])
```

Parameters

target

Zero-based index at which to copy the sequence to. If negative, `target` will be counted from the end.

If `target` is at or greater than `arr.length`, nothing will be copied. If `target` is positioned after `start`, the copied sequence will be trimmed to fit `arr.length`.

`start` Optional

Zero-based index at which to start copying elements from. If negative, `start` will be counted from the end.

If `start` is omitted, `copyWithin` will copy from index 0.

`end` Optional

Zero-based index at which to end copying elements from. `copyWithin` copies up to but not including `end`. If negative, `end` will be counted from the end.

If `end` is omitted, `copyWithin` will copy until the last index (default to `arr.length`).

Return value

The modified array.

Description

The `copyWithin` works like C and C++'s `memmove`, and is a high-performance method to shift the data of an `Array`. This especially applies to the `TypedArray` method of the same name. The sequence is copied and pasted as one operation; pasted sequence will have the copied values even when the copy and paste region overlap.

The `copyWithin` function is intentionally *generic*, it does not require that its `this` value be an `Array` object.

The `copyWithin` method is a mutable method. It does not alter the length of `this`, but it will change its content and create new properties, if necessary.

Examples

Function.name

Function.prototype

Methods

Function.prototype.apply()

Function.prototype.bind()

Function.prototype.call()

Function.prototype.isGenerator()

Function.prototype.toSource()

Function.prototype.toString()

Object

Properties

Object.prototype.__count__

Object.prototype.__noSuchMethod__

Object.prototype.__parent__

Object.prototype.__proto__

Object.prototype.constructor

Methods

Object.prototype.__defineGetter__()

Object.prototype.__defineSetter__()

Object.prototype.__lookupGetter__()

Object.prototype.__lookupSetter__()

Object.prototype.hasOwnProperty()

Object.prototype.isPrototypeOf()

Object.prototype.propertyIsEnumerable()

Object.prototype.toLocaleString()

Object.prototype.toSource()

Object.prototype.toString()

Object.prototype.unwatch()

Object.prototype.valueOf()

Object.prototype.watch()

Object.setPrototypeOf()

```
[1, 2, 3, 4, 5].copyWithin(-2);
// [1, 2, 3, 1, 2]

[1, 2, 3, 4, 5].copyWithin(0, 3);
// [4, 5, 3, 4, 5]

[1, 2, 3, 4, 5].copyWithin(0, 3, 4);
// [4, 2, 3, 4, 5]

[1, 2, 3, 4, 5].copyWithin(-2, -3, -1);
// [1, 2, 3, 3, 4]

[].copyWithin.call({length: 5, 3: 1}, 0, 3);
// {0: 1, 3: 1, length: 5}

// ES2015 Typed Arrays are subclasses of Array
var i32a = new Int32Array([1, 2, 3, 4, 5]);

i32a.copyWithin(0, 2);
// Int32Array [3, 4, 5, 4, 5]

// On platforms that are not yet ES2015 compliant:
[].copyWithin.call(new Int32Array([1, 2, 3, 4, 5]), 0, 3, 4);
// Int32Array [4, 2, 3, 4, 5]
```

Polyfill

```
if (!Array.prototype.copyWithin) {
  Object.defineProperty(Array.prototype, 'copyWithin', {
    value: function(target, start/*, end*/) {
      // Steps 1-2.
      if (this == null) {
        throw new TypeError('this is null or not defined');
      }

      var O = Object(this);

      // Steps 3-5.
      var len = O.length >>> 0;

      // Steps 6-8.
      var relativeTarget = target >> 0;

```

```
    var to = relativeTarget < 0 ?
      Math.max(len + relativeTarget, 0) :
      Math.min(relativeTarget, len);

    // Steps 9-11.
    var relativeStart = start >> 0;

    var from = relativeStart < 0 ?
      Math.max(len + relativeStart, 0) :
      Math.min(relativeStart, len);

    // Steps 12-14.
    var end = arguments[2];
    var relativeEnd = end === undefined ? len : end >> 0;

    var final = relativeEnd < 0 ?
      Math.max(len + relativeEnd, 0) :
      Math.min(relativeEnd, len);

    // Step 15.
    var count = Math.min(final - from, len - to);

    // Steps 16-17.
    var direction = 1;

    if (from < to && to < (from + count)) {
      direction = -1;
      from += count - 1;
      to += count - 1;
    }

    // Step 18.
    while (count > 0) {
      if (from in O) {
        O[to] = O[from];
      } else {
        delete O[to];
      }

      from += direction;
      to += direction;
      count--;
    }

    // Step 19.
    return O;
  },
  configurable: true,
```

```
  writable: true
  });
}
```

Specifications

Specification	Status	Comment
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array.prototype.copyWithin' in that specification.	ST Standard	Initial definition.
ECMAScript 2016 (ECMA-262) The definition of 'Array.prototype.copyWithin' in that specification.	ST Standard	
ECMAScript Latest Draft (ECMA-262) The definition of 'Array.prototype.copyWithin' in that specification.	D Draft	

Browser compatibility

Update compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet	Node.js
copyWithin	45	12	32	No	32	9	Yes	45	32	Yes	Yes	Yes	4.0.0

What are we missing?

Full support

No support

See also

- `Array`

Last modified: May 4, 2019, by MDN contributors