JavaScript Array sort: Sorting Array Elements

Summary: in this tutorial, you will learn how to use the JavaScript Array `sort()` method to sort arrays of numbers, string, and objects.

Introduction to JavaScript Array sort() method

The `sort()` method allows you to sort elements of an array (http://www.javascripttutorial.net/javascript-array/) in place. Besides returning the sorted array, the `sort()` method changes the positions of the elements in the original array.

By default, the `sort()` method sorts the array elements in ascending order with the smallest value first and largest value last.

The `sort()` method casts elements to strings and compares the strings to determine the orders.

Consider the following example:

```
let numbers = [0, 1 , 2, 3, 10, 20, 30 ];
numbers.sort();
console.log(numbers);
```

The output is:

```
[ 0, 1, 10, 2, 20, 3, 30 ]
```

In this example, the `sort()` method places 10 before 2 because the string "10" comes before "2" when doing a string comparison.

To fix this, you need to pass a compare function to the `sort()` method. The `sort()` method will use the compare function to determine the orders of elements.

The following illustrates the syntax of the `sort()` method:

```
array.sort(comparefunction)
```

The `sort()` method accepts an optional argument which is a function (http://www.javascripttutorial.net/javascript-function/) that compares two elements of the array.

If you omit the compare function, the `sort()` method sorts the elements with the sort order based on the Unicode code point values of elements as mentioned earlier.

The compare function of the `sort()` method accepts two arguments and returns a value that determines the sort order. The following illustrates the syntax of the compare function:

```
function compare(a,b) {
  // ...
}
```

The `compare()` function accepts two arguments `a` and `b`. The `sort()` method will sort elements based on the return value of the `compare()` function with the following rules:

1. If `compare(a,b)` is less than zero, the `sort()` method sorts `a` to a lower index than `b`. In other words, `a` will come first.
2. If `compare(a,b)` is greater than zero, the `sort()` method sort `b` to a lower index than `a`, i.e., b will come first.
3. If `compare(a,b)` returns zero, the `sort()` method considers a equals b and leaves their positions unchanged.

To fix the issue of sorting the number, you can use the following syntax:

```
let numbers = [0, 1 , 2, 3, 10, 20, 30 ];
numbers.sort( function( a , b){
    if(a > b) return 1;
    if(a < b) return -1;
    return 0;
});

console.log(numbers);
```

Output:

```
[ 0,  1,  2, 3, 10, 20, 30 ]
```

Or you can define the comparison function using the [arrow function (https://www.javascripttutorial.net/es6/javascript-arrow-function/)](https://www.javascripttutorial.net/es6/javascript-arrow-function/) syntax:

```
let numbers = [0, 1 , 2, 3, 10, 20, 30 ];
numbers.sort((a,b) => {
    if(a > b) return 1;
    if(a < b) return -1;
    return 0;
});

console.log(numbers);
```

And the following is the simplest since the elements of the array are numbers:

```
let numbers = [0, 1, 2, 3, 10, 20, 30];
numbers.sort((a, b) => a - b);
```

```
console.log(numbers);
```

Sorting an array of strings

Suppose you have an array of string named `animals` as follows:

```
let animals = [
    'cat', 'dog', 'elephant', 'bee', 'ant'
];
```

To sort the elements of the `animals` array in ascending order alphabetically, you use the `sort()` method without passing the compare function as shown in the following example:

```
let animals = [
    'cat', 'dog', 'elephant', 'bee', 'ant'
];
animals.sort();

console.log(animals);
```

Output:

```
[ 'ant', 'bee', 'cat', 'dog', 'elephant' ]
```

To sort the `animals` array in descending order, you need to change the logic of the compare function and pass it to the `sort()` method as the following example.

```
let animals = [
    'cat', 'dog', 'elephant', 'bee', 'ant'
];
```

```
animals.sort((a, b) => {
    if (a > b)
        return -1;
    if (a < b)
        return 1;
    return 0;
});

console.log(animals);
```

Output:

```
[ 'elephant', 'dog', 'cat', 'bee', 'ant' ]
```

Suppose you have an array that contains elements in both uppercase and lowercase as follows:

```
// sorting array with mixed cases
let mixedCaseAnimals = [
    'Cat', 'dog', 'Elephant', 'bee', 'ant'
];
```

To sort this array alphabetically, you need to use a custom compare function to convert all elements to the same case e.g., uppercase for comparison and pass that function to the `sort()` method.

```
let mixedCaseAnimals = [
    'Cat', 'dog', 'Elephant', 'bee', 'ant'
];

mixedCaseAnimals.sort(function (a, b) {
    let x = a.toUpperCase(),
```

```
        y = b.toUpperCase();
    return x == y ? 0 : x > y ? 1 : -1;

});
```

Output:

```
[ 'ant', 'bee', 'Cat', 'dog', 'Elephant' ]
```

Sorting an array of strings with non-ASCII characters

The `sort()` method is working fine with the strings with ASCII characters. However, for the strings with non-ASCII characters e.g., é, è, etc., the `sort()` method will not work correctly. For example:

```
let animaux = ['zèbre', 'abeille', 'écureuil', 'chat'];
animaux.sort();

console.log(animaux);
```

As you see, the `écureuil` string should come before the `zèbre` string.

To resolve this, you use the `localeCompare()` method of the `String` object to compare strings in a specific locale, like this:

```
animaux.sort(function (a, b) {
    return a.localeCompare(b);
});
console.log(animaux);
```

Output:

```
[ 'abeille', 'chat', 'écureuill', 'zèbree' ]
```

The elements of the `animaux` array now are in the correct order.

Sorting an array of numbers

Suppose you have an array of numbers named `scores` as in the following example.

```
let scores = [
    9, 80, 10, 20, 5, 70
];
```

To sort an array of numbers numerically, you need to pass into a custom comparison function that compares two numbers.

The following example sorts the `scores` array numerically in ascending order.

```
let scores = [
    9, 80, 10, 20, 5, 70
];
// sort numbers in ascending order
scores.sort((a, b) => a - b);

console.log(scores);
```

Output:

```
[ 5, 9, 10, 20, 70, 80 ]
```

To sort an array of numbers numerically in descending order, you just need to reverse the logic in the compare function as shown in the following example:

```
let scores = [
    9, 80, 10, 20, 5, 70
];
// descending order
scores.sort((a, b) => b - a);
console.log(scores);
```

Output:

```
[80, 70, 20, 10, 9, 5]
```

Sorting an array of objects by a specified property

The following is an array of `employee` objects, where each object contains three properties: `name`, `salary` and `hireDate`.

```
let employees = [
    {name: 'John', salary: 90000, hireDate: "July 1, 2010"},
    {name: 'David', salary: 75000, hireDate: "August 15, 2009"},
    {name: 'Ana', salary: 80000, hireDate: "December 12, 2011"}
];
```

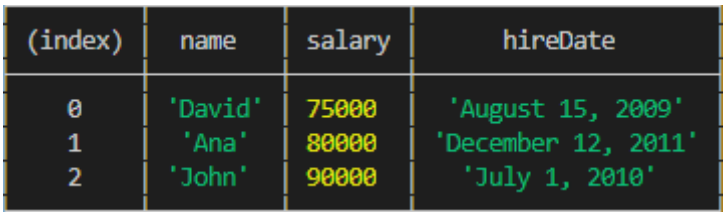
Sorting objects by a numeric property

The following example shows how to sort the employees by `salary` in ascending order.

```
// sort by salary
employees.sort(function (x, y) {
    return x.salary - y.salary;
});

console.table(employees);
```

Output:



(index)	name	salary	hireDate
0	'David'	75000	'August 15, 2009'
1	'Ana'	80000	'December 12, 2011'
2	'John'	90000	'July 1, 2010'

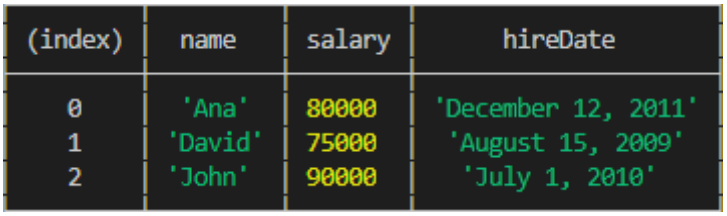
This example is similar to the example of sorting an array of numbers in ascending order. The difference is that it compares the `salary` property of two objects instead.

Sorting objects by a string property

To sort the `employees` array by `name` property case-insensitively, you pass the compare function that compares two strings case-insensitively as follows:

```
employees.sort(function (x, y) {
    let a = x.name.toUpperCase(),
        b = y.name.toUpperCase();
    return a == b ? 0 : a > b ? 1 : -1;
});

console.table(employees);
```



(index)	name	salary	hireDate
0	'Ana'	80000	'December 12, 2011'
1	'David'	75000	'August 15, 2009'
2	'John'	90000	'July 1, 2010'

Sorting objects by the date property

Suppose, you wish to sort employees based on each employee's hire date.

The hire date data is stored in the `hireDate` property of the employee object. However, it is just a string that represents a valid date, not the `Date` object.

Therefore, to sort employees by hire date, you first have to create a valid `Date` object from the date string, and then compare two dates, which is the same as comparing two numbers.

Here is the solution:

```
employees.sort(function (x, y) {
    let a = new Date(x.hireDate),
        b = new Date(y.hireDate);
    return a - b;
});

console.table(employees);
```

(index)	name	salary	hireDate
0	'David'	75000	'August 15, 2009'
1	'John'	90000	'July 1, 2010'
2	'Ana'	80000	'December 12, 2011'

Optimizing JavaScript Array sort() method

In fact, the `sort()` method calls the compare function multiple times for each element in the array.

See the following example:

```
let rivers = ['Nile', 'Amazon', 'Congo', 'Mississippi', 'Rio-Grande'];

rivers.sort(function (a, b) {
    console.log(a, b);
    return a.length - b.length;
});
```

Output:

```
Amazon Nile
Congo Amazon
Congo Amazon
Congo Nile
Mississippi Congo
Mississippi Amazon
Rio-Grande Amazon
Rio-Grande Mississippi
```

How it works:

1. First, declare an array `rivers` that consists of the famous river names.
2. Second, sort the `rivers` array by the length of its element using the `sort()` method. We output the elements of the `rivers` array to the web console whenever the `sort()` method invokes the comparison function .

As shown in the output above, each element has been evaluated multiple times e.g., Amazon 4 times, Congo 2 times, etc.

If the number of array elements is increasing, it will potentially decrease the performance.

You cannot reduce the number of times that comparison function is executed. However, you can reduce the work that the comparison has to do. This technique is called Schwartzian Transform (http://en.wikipedia.org/wiki/Schwartzian_transform) .

To implement this, you follow these steps:

1. First, extract the actual values into a temporary array using the map() (http://www.javascripttutorial.net/javascript-array-map/) method.
2. Second, sort the temporary array with the elements that are already evaluated (or transformed).
3. Third, walk the temporary array to get an array with the right order.

Here is the solution:

```
// temporary array holds objects with position
// and length of element
var lengths = rivers.map(function (e, i) {
    return {index: i, value: e.length };
});

// sorting the lengths array containing the lengths of
// river names
lengths.sort(function (a, b) {
    return +(a.value > b.value) || +(a.value === b.value) - 1;
});

// copy element back to the array
var sortedRivers = lengths.map(function (e) {
    return rivers[e.index];
});

console.log(sortedRivers);
```

Output:

```
[ 'Nile', 'Congo', 'Amazon', 'Rio-Grande', 'Mississippi' ]
```

In this tutorial, you have learned how to use the JavaScript Array `sort()` method to sort arrays of strings, numbers, dates, and objects.