12 extremely useful tricks for JavaScript programmers

Let's TipsMake.com learn 12 extremely useful tips for JavaScript programmers in this article!

1. 5 free online HTML editing tools that test the best code
2. 13 skills needed to become Frontend Developer
3. Beginners of computer programming need to focus on what?
4. 13 important SQL statements Programmer needs to know

In this article, TipsMake.com will share **12 extremely useful tricks for JavaScript programmers** . These tips will help you reduce the amount of code as well as help the code run better. Invite you to welcome reading!

1. Convert to boolean using operator !!

Sometimes we need to check if a variable exists or has a valid value, to see them as *true values* . To determine, you can use **`!!`** (Double negation operator) a simple **`!!variable`** , will automatically convert any boolean data type and this variable will return **`false`** when it has values ??like: **`0 ; null ; "" ; undefined 0 ; null ; "" ; undefined 0 ; null ; "" ; undefined`** or **`NaN`** , otherwise it will return **`true`** . To better understand how it works, take a look at the following simple example:

```
function Account(cash) {
 this.cash = cash;
 this.hasMoney = !!cash;
}
var account = new Account(100.50);
console.log(account.cash); // 100.50
console.log(account.hasMoney); // true

var emptyAccount = new Account(0);
console.log(emptyAccount.cash); // 0
console.log(emptyAccount.hasMoney); // false
```

In the above example, if the **`account.cash`** value is greater than 0, **`account.hasMoney`** will have a value of true.

2. Convert to number using the + operator

This procedure is excellent and easy to perform. However, it only works with numeric strings, otherwise it returns **`NaN`** (Not a Number - Not a number). Take a look at the following example:

```
function toNumber(strNumber) {
 return +strNumber;
}
console.log(toNumber("1234")); // 1234
console.log(toNumber("ACB")); // NaN
```

This trick also works for both **`Date`** and in this case it returns the timestamp number:

```
console.log(+new Date()) // 1461288164385
```

3. Shorten the conditions

If you see a code like the one below:

```
if (conected) {
 login();
}
```

You can shorten it by combining a variable (to be validated) and a function using **`&&`** (AND operator) in the middle. For example, the above code may become more concise in a line:

```
conected && login();
```

You can do the same to check if the property or function exists in the object. Similar to the code below:

```
user && user.login();
```

4. Set the default value using the || operator

Currently, ES6 has the default parameter feature. To simulate this feature in older browsers, you can use || (OR operator) by inserting the default value as the second parameter to use. If the first parameter returns **`false`** , the

second parameter will be used as the default value. See the following example:

```
function User(name, age) {
 this.name = name || "Oliver Queen";
 this.age = age || 27;
}
var user1 = new User();
console.log(user1.name); // Oliver Queen
console.log(user1.age); // 27  console.log(user1.age); // 27
var user2 = new User("Barry Allen", 25);
console.log(user2.name); // Barry Allen
console.log(user2.age); // 25
```

5. Store array.length in the loop

This trick is very simple and has a great impact on performance when processing large arrays in the loop. Basically, most people use a **`for`** loop to browse the array as follows:

```
for (var i = 0; i  array.length; i++) {
 console.log(array[i]);
}
```

It's okay to work with small arrays, but if you handle large arrays, this code will recalculate the size of the array after each iteration and that will cause a bit of delay. To avoid this, you can store the array.length in a variable to use it instead of calling **`array.length`** in each iteration:

```
var length = array.length;
for (var i = 0; i  length; i++) {
 console.log(array[i]);
}
```

To make it look more compact, simply rewrite the following:

```
for (var i = 0, length = array.length; i  length; i++) {
 console.log(array[i]);
}
```

6. Identify attributes in an object

This trick is extremely useful when you need to check if the property exists and avoid running unspecified functions or attributes. If you intend to write code that runs on multiple browsers, you can also use this technique.

For example, imagine you need to write code that is compatible with the old Web browser Internet Explorer 6 - IE6 and you want to use **`document.querySelector()`** to get some elements by their ID. However, in IE6 this function does not exist. To check if the function exists, use the **`in`** operator as the example below:

```
if ('querySelector' in document) {
 document.querySelector("#id");
} else {
 document.getElementById("id");
```

```
}
```

In this case, if there is no **querySelector** function in the document, we can use **document.getElementById()** instead.

7. Get the last element in the array

Array.prototype.slice(begin, end) can cut the array when you set the **begin** and **end** parameters. But if you do not enter the **end** parameter, this function will automatically set the maximum value for the array. Surely few people know that this function can accept negative values ??and if you set the **begin** parameter to be a negative number, you will get the last element from the array:

```
var array = [1, 2, 3, 4, 5, 6];
console.log(array.slice(-1)); // [6]
console.log(array.slice(-2)); // [5,6]
console.log(array.slice(-3)); // [4,5,6]
```

8. Short cut array

This technique can lock the array size, which is useful when you want to delete some elements of the array based on the number of elements you set. For example, if you have an array of 10 elements but you only want to get the first 5 elements, you can truncate the array, making it smaller by setting **array.length = 5** . See the following example:

```
var array = [1, 2, 3, 4, 5, 6];
console.log(array.length); // 6
array.length = 3;
console.log(array.length); // 3
console.log(array); // [1,2,3]
```

9. Replace the whole

The **String.replace()** function allows to use **String** and **Regex** to replace strings, but this function only replaces the first substring that appears. You can simulate a function **replaceAll()** using **/g** at the end of the Regex:

```
var string = "john john";
console.log(string.replace(/hn/, "ana")); // "joana john"
console.log(string.replace(/hn/g, "ana")); // "joana joana"
```

10. Include the array

If you need to put two arrays together, you can use the **Array.concat()** function:

```
var array1 = [1, 2, 3];
var array2 = [4, 5, 6];
console.log(array1.concat(array2)); // [1,2,3,4,5,6];
```

However, this function is not the best way to merge large arrays as it will consume a lot of memory by creating a new array. In this case, you can use **`Array.push.apply(arr1, arr2)`** instead of creating a new array, it will merge the second array into the first array thereby reducing memory usage:

```
var array1 = [1, 2, 3];
var array2 = [4, 5, 6];
console.log(array1.push.apply(array1, array2)); // [1,2,3,4,5,6];
```

11. Convert NodeList to array

If you run **`document.querySelectorAll("p")`** , it will return an array containing DOM elements, NodeList objects. But this object doesn't have all the functions of the array like: **`sort()`** ; **`reduce()`** ; **`map()`** ; **`filter()`** **`sort()`** ; **`reduce()`** ; **`map()`** ; **`filter()`** **`sort()`** ; **`reduce()`** ; **`map()`** ; **`filter()`** . To be able to use these and many other available functions of the array, you need to convert the NodeList into an array. You only need to use the function: **`[].slice.call(elements)`** :

```
var elements = document.querySelectorAll("p"); // NodeList

var arrayElements = [].slice.call(elements); // Now the NodeList is an array


var arrayElements = Array.from(elements); // This is another way of converting No
```

12. Mix elements in the array

To tamper with the elements in the array without using separate libraries like *Lodash* , you only need to use the following trick:

```
var list = [1, 2, 3];
console.log(list.sort(function() {
 return Math.random() - 0.5
})); // [2,1,3]
```

Conclude

Now you have learned some useful JavaScript tricks that are mostly used to minimize code and some tricks used in popular JavaScript frameworks like Lodash, Underscore.js, Strings.js and many frameworks. other.

Hope you enjoy this article and if you still know any other useful JavaScript tips, let us know in the comment section below!

Refer to some more articles:

1. Do you know the 15 hottest programming languages ??on this GitHub?
2. 7 Framework JavaScript for mobile application development
3. If you want a successful career, find out about the five 2018 technology trends!

Having fun!

similar articles on tips and guides. Thank you for reading and for following us regularly.