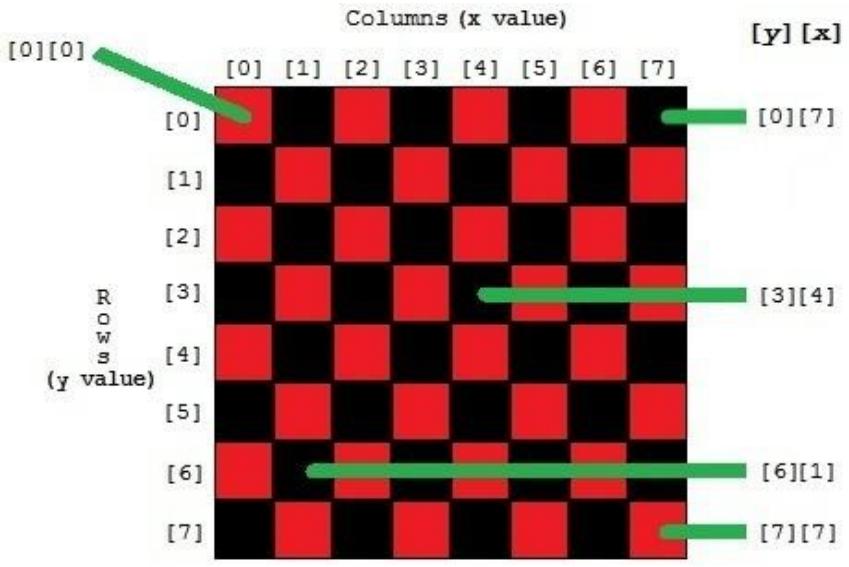
Introduction to 2D Array - 2-dimensional array in JavaScript

In the following article, we will introduce and show you some basic operations to create and access 2-dimensional arrays - 2D array in JavaScript. Essentially, a 2-dimensional array is a concept of a matrix of matrices - a matrix, used to store information. Each 1 element contains 2 separate indexes: row (y) - column and column (x) - column.

TipsMake.com - In the following article, we will introduce and guide you to some basic operations to create and access 2-dimensional arrays - 2D array in JavaScript. Essentially, a 2-dimensional array is a concept of a matrix of matrices - a matrix, used to store information. Each 1 element contains 2 separate indexes: row (y) - column and column (x) - column. The matrix will process each time you enter the data stream and column. For example, if you want to display data on each line with each year and each column with each month, or when you perform the process of creating a financial report for a company, organization, business . then the use of Using matrix is ??an extremely reasonable solution, corresponding to the business situation and performance of each department, department . or compared with the overall target of the whole company.



Think of a chessboard, with 8 lines (marked from **0> 7**), 8 columns (also marked from **0> 7**). If you want to check the exact position of the upper cell, then go to **chessboard [0] [0]** , the next cell will be position **[0] [1]** , and continue to replace the value of x into **[0] [7]** to go all the edges from left to right of the board. Similarly, change the y value if you want to move from top to bottom, for example **[6] [1]** is the second cell of the **7th** column.

The symbol we are using here is quite consistent with the variables in **JavaScript: [y] [x]** , all arrays start at 0, so it can be roughly understood in terms of the question: 'Distance How many left-sided adjectives' or 'position 0' will be exactly the first coordinates from the left.

One of the most commonly used conventions when applied to matrices is to use x and y variables, for example, x is always a column index (distance, adjective left), and y is only Line book (distance from top down). Therefore, the coordinates **y and x** corresponding to **[0] [0]** are the first component in the upper left corner, **[0] [1]** is the next second component from top to bottom, **[1] [n]** is the second line .

JavaScript and 2D Array:

But in fact, **JavaScript** does not support **2D Array** . And the most common way to handle data in two-dimensional arrays is to create Array objects, including many **Array** objects inside.

Using 1-dimensional arrays is similar to 2-dimensional arrays:

In many practical cases, we can easily 'squeeze' a 1-dimensional array into 2-dimensional arrays using a logical programming operator. For example, we have a string of characters as follows:

```
var sData = "abcdefghijABCDEFGHIJäß © ÐëØ> ½îÿ";
```

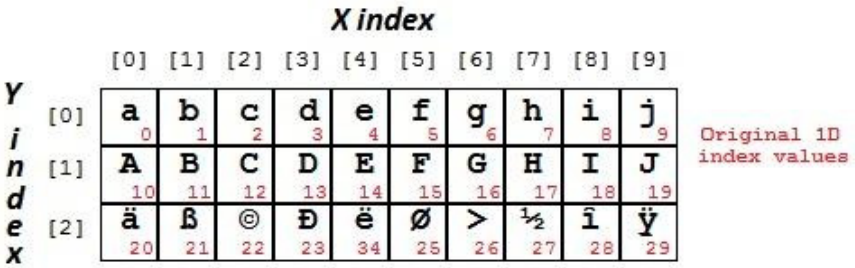
Here, we can easily see a simple model, with 3 consecutive sequences of 10 characters, corresponding to that **10 x 3** matrix:

```
// column: 0123456789
var sData = "abcdefghij" // row 0 (starts at offset 0)
+ "ABCDEFGHIJ" // row 1 (starts at offset 10)
+ "äß © ÐëØ> ½îÿ" // row 2 (starts at offset 20)
```

Each line here will contain 10 columns, we can easily calculate and determine the first position of any line by multiplying the number of lines by 10. The general formula here will take the form:

```
(y * row_length) + x
```

where **y** is the number of rows, **x** is the number of columns, **row_length** is the number of columns in each line.



With data like the above example, we can construct the 2-dimensional array structure as follows:

```
// assumes data is a string, sData and hàng có 10 columns
function GetCellValue (y, x) {
nRowStart = y * 10;
```

```
nOffset = nRowStart + x;
sElementValue = sData.substr (nOffset, 1);  // access one element
return (sElementValue);
}
.
// y, x (row, column)
alert (GetCellValue (0.0));  // displays a
alert (GetCellValue (0.1));  // displays b
alert (GetCellValue (0.2));  // displays c
alert (GetCellValue (1,2));  // displays C
alert (GetCellValue (2.2));  // displays ©
alert (GetCellValue (0.9));  // displays j
alert (GetCellValue (1.9));  // displays J
alert (GetCellValue (2.9));  // displays ÿ
```

After that, we were able to access the data string similar to the 2-dimensional array with each individual character. And so on, we can completely force the 2-dimensional array structure into a one-dimensional array by assigning special data access functions. But this is not a popular way of doing **JavaScript** .

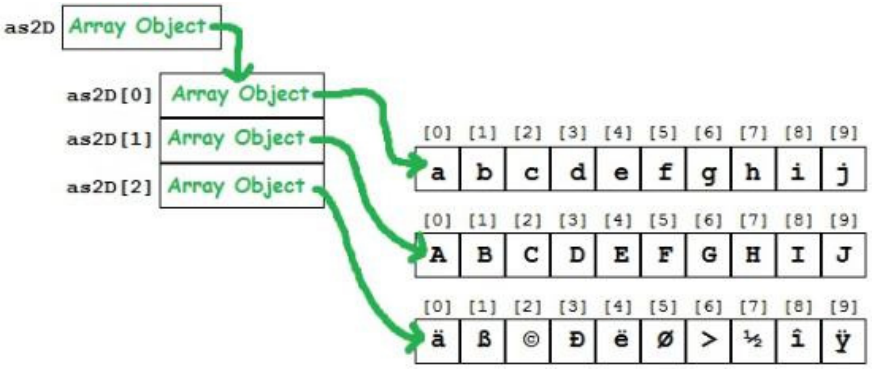
Use array array:

In fact, the regular use of 2-dimensional arrays in **JavaScript** is to create a 1-dimensional array, then assign each of them inside with another 1-dimensional array. If going into specific analysis, the function below is one of the simple ways to create and fix 2-dimensional arrays:

```
as2D = new Array ();  // an array of "whatever"
as2D [0] = new Array ("a", "b", "c", "d", "e", "f", "g", "h", "i", "j");
as2D [1] = new Array ("A", "B", "C", "D", "E", "F", "G", "H", "I", "J");
as2D [2] = new Array ("ä", "ß", "©", "Ð", "ë", "Ø", ">", "½", "î", "ÿ");
```

At that time, we were able to build and identify the array of data with 3 objects, each with 10 different character strings. And now, continue to use **JavaScript** syntax to access as usual:

```
alert (as2D [0] [0]);  // displays a
alert (as2D [2] [2]);  // displays ©
alert (as2D [2] [9]);  // displays ÿ
```



Use […]:

Syntax:

```
var a = [item0, item1, item2, .];
```

is an acronym for:

```
var a = new Array (item0, item1, item2, .);
```

Thereby, we can understand that:

```
[] similar to new arrays and no data

["a"] is similar to the new array with 1 data string

["a", "b"] is similar to the new array with 2 data series
```

and so on. Because of the measurement, you can define and construct the array as above with the syntax:

```
var as2D = [
["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"],
["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"],
["ä", "ß", "©", "Ð", "ë", "Ø", ">", "½", "î", "ÿ"]
];
```

With such syntax, **JavaScript** can easily be built into arrays, similar to the syntax:

```
as2D [n] = new Array (a, b, c, .)
```

has been used before. And the way to access the data is no different.

Use the Array.push () function:

Push () function when applied to objects will perform the function of assigning a new object (or string) to the last position. This is often used to define an array from the beginning, we can use the syntax:

```
var as2D = new Array ();
as2D [0] = new Array ();
as2D [0] .push ("a");
as2D [0] .push ("b");
```

```
as2D [0] .push ("c", "d", "e", "f", "g", "h", "i");
as2D [0] .push ("j");

as2D.push (New Array ("A", "B", "C", "D", "E", "F", "G", "H", "I", "J"));

as2D.push (["ä", "ß", "©", "Ð", "ë", "Ø", ">", "½", "î", "ÿ"]);
```

The above syntax is used to create an array-like object in the array, and how it works similar to the above example. However, it should be noted that the **push ()** function allows users to stack single data sections (like lines **3,4 and 6**) or double data (line **5**), while lines **7 and 8** will stack all the data into the top position of the array. We can see the difference from the above example when there is no illustration:

```
var as2D = [];  // or: new Array ();
as2D.push (["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]);
as2D.push (["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]);
as2D.push (["ä", "ß", "©", "Ð", "ë", "Ø", ">", "½", "î", "ÿ"]);
```

Use String.split () function:

The **split ()** function of the **String** object in **JavaScript** returns an **Array** object, and is very frequently used in fixed Array with previously initialized variables:

```
var sData1 = "a, b, c, d, e, f, g, h, i, j";
var sData2 = "A, B, C, D, E, F, G, H, I, J";
var sData3 = "ä, ß, ©, Ð, ë, Ø,>, ½, î, ÿ";
var as2D = [];  // or: new Array ();
as2D [0] = sData1.split (",");
as2D [1] = sData2.split (",");
as2D [2] = sData3.split (",");
```

The second parameter in the **split ()** function has the function of confirming all delimiters, in which case we use commas. There is one rule as follows: if the separator character has an empty form **('')** , the returned result will be a separate character fish data array.

```
var sData1 = "abcdefghij";
var sData2 = "ABCDEFGHIJ";
var sData3 = "äß © ÐëØ> ½îÿ";
var as2D = [];  // or: new Array ();
as2D [0] = sData1.split ("");
as2D [1] = sData2.split ("");
as2D [2] = sData3.split ("");
```

or:

```
var as2D = [];
as2D [0] = "abcdefghij" .split ("");
as2D [1] = "ABCDEFGHIJ" .split ("");
as2D [2] = "äß © ÐëØ> ½îÿ" .split ("");
```

or:

```
var as2D = "abcdefghij, ABCDEFGHIJ, äß © ÐëØ> ½îÿ" .split (",")
as2D [0] = as2D [0] .split ("");
as2D [1] = as2D [1] .split ("");
as2D [2] = as2D [2] .split ("");
```

or even:

```
var as2D = [
"abcdefghij" .split (""),
"ABCDEFGHIJ" .split (""),
"äß © ÐëØ> ½îÿ" .split ("")
];
```

If you compare the final code with C ++, JavaScript is a bit different: declaring var is only part of the procedure done during execution.

Create 2-dimensional array:

Use the repeat command for:

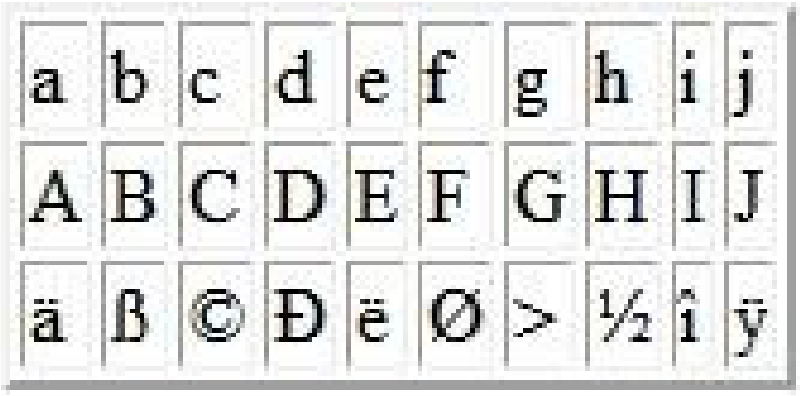
The main reason for creating and using 2-dimensional arrays is that at a given time or location throughout the program, we must use repeating command structures. For example:

```
cho (var y = 0; y 3; y ++) {
for (var x = 0; x 10; x ++) {
// do something with myArray [y] [x]
}
}
```

And this process to serve the program when going through each row and column to access data at the corresponding location. For example:

```
sOut var = "";
cho (var y = 0; y   sOut + = "";
cho (var x = 0; x   sOut + = "" + as2D [y] [x] + "";
}
sOut + = "";
}
sOut + = "";
```

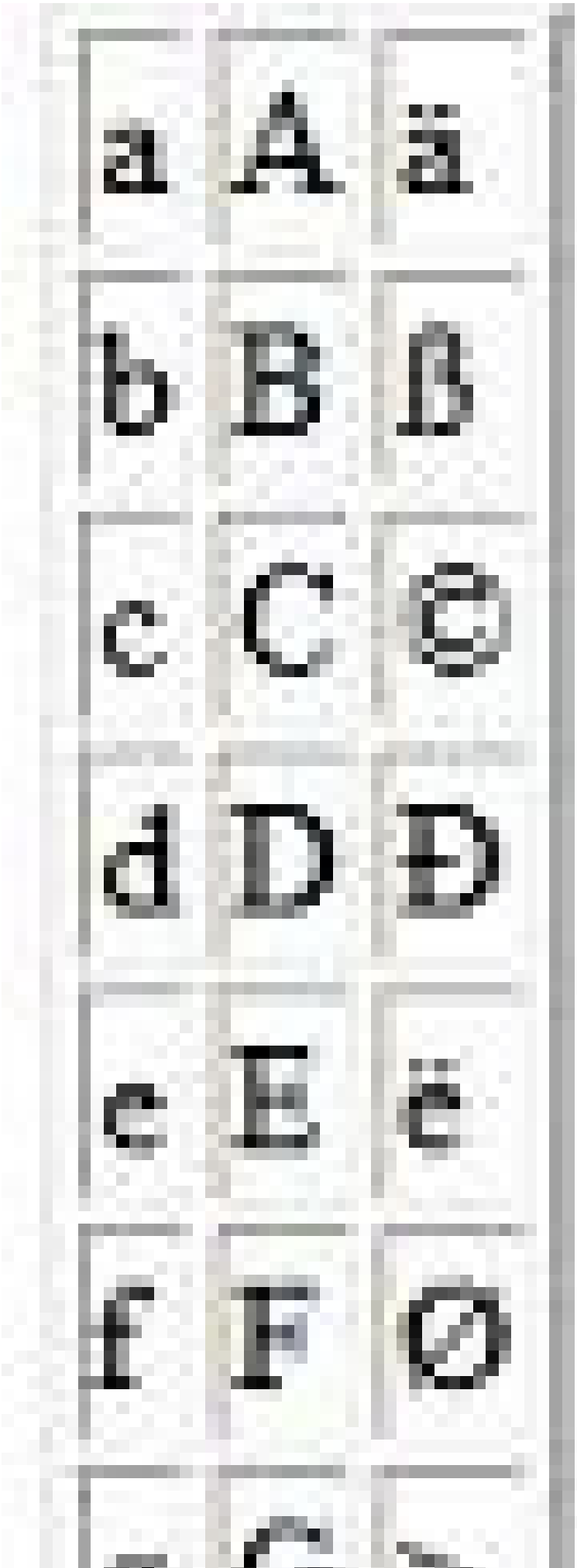
will create an **HTML** page that looks like this:



And if changing the position of the row and column for each other:

```
var nClmsPerRow = as2D [0] .length;  // Like the same length
cho (var x = 0; x   sOut + = "";
cho (var y = 0; y   sOut + = "" + as2D [y] [x] + "";
}
sOut + = "";
}
```

Our table will have 10 rows and 3 columns:



Use the repeat command for . in:

Besides, **JavaScript** also provides the user with a fairly special looping command through the data array, which is the **for . in** function. Using this function is quite simple once you know about the end condition of the loop (the last part in the data array). And it is used together with **Collection and Array** . For Array, the general syntax will take the form:

```
for (nIdxVar in aArray)
```

Each 1 loop will set **nIdxVar to** a repeating index (0, 1, 2, .), and this process will end when reaching the last position in the array. Here are some code snippets that access all components in the 2-dimensional array example above:

```
for (y in as2D) {
for (x in as2D [y]) {
// do something with as2D [y] [x];
}
}
```

The actual value part of **for . in** will appear when we have the sparse array; In particular, some components have not been clearly defined. Examples are as follows:

```
var aSparse = new Array;
aSparse [0] = ["zero", "one", "two"];
aSparse [4] = [, "forty-one",];
aSparse [5] = ["fifty", "fifty-one", "fifty-two"];
for (y in aSparse) {
for (x in aSparse [y]) {
alert ("y, x = (" + y + "," + x + ") value:" + aSparse [y] [x]);
}
}
```

will ignore lines 1> 3, columns 0 and 2 of line 4, the entire value here will not be defined. And the results returned here will take the form:

```
y, x = (0,0) = 0
y, x = (0,1) = 1
y, x = (0.2) = 2
y, x = (4.1) = 40 - 1
y, x = (5.0) = 50
y, x = (5.1) = 50 - 1
y, x = (5.2) = 50 - 2
```

Comparing 2-dimensional arrays and object combinations:

In fact, 2-dimensional arrays are suitable for 'representing' a matrix with predefined objects. For example: 1 chess board has 8 x 8 cells, 1 screen will have **1680 x 1050** pixel matrix . And so, the matrix **x - y** formula is frequently used in advanced programs. written in **JavaScript** .

The more common cases of 1-dimensional arrays are lists of groups of different objects and components. And this structure is often applied in database aggregators, list of objects on a web page, data related to shopping cart processes .

Good luck!