Map vs filter vs reduce

<https://codeburst.io/array-methods-explained-filter-vs-map-vs-reduce-vs-foreach-ea3127c6d319>

Design pattern in javascript

Let vs var

Scope based example

function varTest() {

var x = 1;

if (true) {

var x = 2; // same variable!

console.log(x); // 2

}

console.log(x); // 2

}

function letTest() {

let x = 1;

if (true) {

let x = 2; // different variable

console.log(x); // 2

}

console.log(x); // 1

}

var a = 1;

var b = 2;

if (a === 1) {

var a = 11; // the scope is global

let b = 22; // the scope is inside the if-block

console.log(a); // 11

console.log(b); // 22

}

console.log(a); // 11

console.log(b); // 2

Property level example

At the top level of programs and functions, **let**, unlike **var**, does not create a property on the global object. For example:

var x = 'global';

let y = 'global';

console.log(this.x); // "global"

console.log(this.y); // undefined

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

Arrow function

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

es5 vs es6

<https://www.wintellect.com/5-great-features-in-es6-harmony/>

classes

constructor,

extends,

arrow function

promise without other dependency(like $q)

Here is the ES5 version

var myModule = require('./myModule');

Here is the ES6 version

import myModule from './myModule';

var obj1 = { a: 1, b: 2, c: 3, d: 4 }  
var a = obj1.a  
var b = obj1.b  
var c = obj1.c  
var d = obj1.d

Time consuming 😕. Oh wait! we have ES6 here to rescue us.

const obj1 = { a: 1, b: 2, c: 3, d: 4 }  
const {  
 a,  
 b,  
 c,  
 d  
} = obj1

We define a object like this in ES5

var a = 1  
var b = 2  
var c = 3  
var d = 4

var obj1 = { a: a, b: b, c: c, d: d }

In ES6 you will do something like this:

var a = 1  
var b = 2  
var c = 3  
var d = 4

var obj1 = { a, b, c, d }

export const x = 1;  
export const y = 2;  
export const z = 'String';

And import them something like this

import {x, y, z} from './myModule';

custom error handling

<https://rclayton.silvrback.com/custom-errors-in-node-js>

<https://gist.github.com/justmoon/15511f92e5216fa2624b>

<https://medium.com/learn-with-talkrise/custom-errors-with-node-express-27b91fe2d947>

sort array based on multiple key of object

var obj = [

{

"one": 1,

"two": 2,

}, {

"one": 1,

"two": 9

}, {

"one": 3,

"two": 5

}

];

obj.sort(function(a, b) {

return a["one"] - b["one"] || a["two"] - b["two"];});

why javascript is single-threades?

Taking about actual javascript runtime v8 engine. It don’t have event loop and single thread and dom. Javascript runtime itself a call stack. That’s why it is single threaded because it has only one call stack.

Asyn is actually provided by browser ,nodejs. setTimeout is not part of javascript runtime. Actually this setTimeout create a timer outside of the javascript runtime and making handshaking between timer and runtime when set timeout done.

callback queue and event queue

Web Api handsover the task to queue and it executes task.

## Architectural Constraints

REST defines **6 architectural constraints** which make any web service – a true RESTful API.

1. [Uniform interface](https://restfulapi.net/rest-architectural-constraints/#uniform-interface)
2. [Client–server](https://restfulapi.net/rest-architectural-constraints/#client-server)
3. [Stateless](https://restfulapi.net/rest-architectural-constraints/#stateless)
4. [Cacheable](https://restfulapi.net/rest-architectural-constraints/#cacheable)
5. [Layered system](https://restfulapi.net/rest-architectural-constraints/#layered-system)
6. [Code on demand (optional)](https://restfulapi.net/rest-architectural-constraints/#code-on-demand)

Javascript runtime visualization

<http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMjAwMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwoKc2V0VGltZW91dChmdW5jdGlvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coIkNsaWNrIHRoZSBidXR0b24hIik7Cn0sIDUwMDApOwoKY29uc29sZS5sb2coIldlbGNvbWUgdG8gbG91cGUuIik7!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D>

bind vs call vs apply in javascript

<https://www.codementor.io/niladrisekhardutta/how-to-call-apply-and-bind-in-javascript-8i1jca6jp>

these 3 methods are used to control the invocation of the function. call() and apply() were introduced in ECMAScript 3 while bind() was added as part of ECMAScript 5.

//Demo with javascript .call()

var obj = {name:"Niladri"};

var greeting = function(a,b,c){

return "welcome "+this.name+" to "+a+" "+b+" in "+c;

};

console.log(greeting.call(obj,"Newtown","KOLKATA","WB"));

// returns output as welcome Niladri to Newtown KOLKATA in WB

//Demo with javascript .apply()

var obj = {name:"Niladri"};

var greeting = function(a,b,c){

return "welcome "+this.name+" to "+a+" "+b+" in "+c;

};

// array of arguments to the actual function

var args = ["Newtown","KOLKATA","WB"];

console.log("Output using .apply() below ")

console.log(greeting.apply(obj,args));

/\* The output will be

Output using .apply() below

welcome Niladri to Newtown KOLKATA in WB \*/

The only difference of apply() with the call() method is that the second parameter of the apply() method accepts the arguments to the actual function as an array.

//Use .bind() javascript

var obj = {name:"Niladri"};

var greeting = function(a,b,c){

return "welcome "+this.name+" to "+a+" "+b+" in "+c;

};

//creates a bound function that has same body and parameters

var bound = greeting.bind(obj);

console.dir(bound); ///returns a function

console.log("Output using .bind() below ");

console.log(bound("Newtown","KOLKATA","WB")); //call the bound function

/\* the output will be

Output using .bind() below

welcome Niladri to Newtown KOLKATA in WB \*/

In the above code sample for bind() we are returning a bound function with the context which will be invoked later. We can see the bound function in the console as below .

Mongoos vs mongodb

<https://medium.com/@bugwheels94/performance-difference-in-mongoose-vs-mongodb-60be831c69ad>

<http://voidcanvas.com/mongoose-vs-mongodb-native/>

webpack

<https://webpack.js.org/api/node/>

<https://blog.appdynamics.com/engineering/understanding-node-js-memory-leaks/>