**Javascript Revisit**

1 Lamda first class functions

eg: test.js

*var Lane = {*

*name: "Lane the Lambda",*

*description: function () {*

*return "A person named " + this.name;*

*}*

*};*

*console.log(Lane.description());*

xiaofengmaclap:myApp xiaofeng$ node test.js

A person named Lane the Lambda

eg: test1.js

*var Lane = {*

*name: "Lane the Lambda",*

*description: function () {*

*return "A person named " + this.name;*

*}*

*};*

*var Fred = {*

*name: "Fred the Functor",*

*descr: Lane.description*

*};*

*console.log(Fred.descr());*

xiaofengmaclap:myApp xiaofeng$ node test1.js

A person named Fred the Functor

eg: test2.js

*var Lane = {*

*name: "Lane the Lambda",*

*description: function () {*

*return "A person named " + this.name;*

*}*

*};*

*var Fred = {*

*name: "Fred the Functor",*

*descr: Lane.description*

*};*

*var descr = Lane.description;*

*console.log(descr());*

Output is:

xiaofengmaclap:myApp xiaofeng$ node test3.js

A person named undefined

2 Function

2.1 Function.prototype.call()

syntax: *fun*.call(*thisArg*[, *arg1*[, *arg2*[, ...]]])

The call() method calls a function with a given this value and arguments provided individually.

**Note:** While the syntax of this function is almost identical to that of apply(), the fundamental difference is that call() accepts an **argument list**, while apply() accepts a **single array of arguments**.

### Return value

The result of calling the function with the specified this value and arguments.

A different this object can be assigned when calling an existing function. this refers to the current object, the calling object. With call, you can write a method once and then inherit it in another object, without having to rewrite the method for the new object.

For right usage, check the following examples,

Example 1 Using call to chain constructors for an object

/\*

\* This example is to demo a chain of constructors

\* Run this as Node Application in eclipse

\*

\* \*/

***function*** *Product(name, price) {*

***this****.name = name;*

***this****.price = price;*

***if*** *(price < 0) {*

***throw*** *RangeError('Cannot create product ' +*

***this****.name + ' with a negative price');*

*}*

*}*

***function*** *Food(name, price) {*

*Product.call(****this****, name, price);*

***this****.category = 'food';*

*}*

***function*** *Toy(name, price) {*

*Product.call(****this****,name, price);*

***this****.category = 'toy';*

*}*

***var*** *cheese =* ***new*** *Food('feta', 5);*

***var*** *fun =* ***new*** *Toy('robot', 40);*

*console.log(cheese);*

*console.log(fun);*

*Explanation: the ‘this’ parameter got passed into the Product.call() method, is to pass Food itself in, and then in Product method, the this.name and this.price becomes Food.name and Food.price. Please try to change to the following and see the difference,*

*function Food(name, price) {*

*//Product.call(this, name, price);*

*Product( name, price);*

*this.category = 'food';*

*}*

Example 2 Using call to invoke an anonymous function

*/\**

*\* In this purely constructed example, we create an anonymous function*

*\* and use call to invoke it on every object in an array. The main purpose*

*\* of the anonymous function here is to add a print function to every object,*

*\* which is able to print the right index of the object in the array. Passing*

*\* the object as this value was not strictly necessary, but is done for*

*\* explanatory purpose.*

*\**

*\* animals[i] is the this. i is the param passed to function(i) which is the*

*\* real argument.*

*\*/*

***var*** *animals = [*

*{ species: 'Lion', name: 'King' },*

*{ species: 'Whale', name: 'Fail' }*

*];*

***for*** *(****var*** *i = 0; i < animals.length; i++) {*

*(****function****(i) {*

***this****.print =* ***function****() {*

*console.log('#' + i + ' ' +* ***this****.species*

*+ ': ' +* ***this****.name);*

*}*

***this****.print();*

*}).call(animals[i], i);*

*}*

Example 3: Bind this to an object

*/\**

*In below example, when we will call greet the value of this will*

*be bind to object i.*

*\*/*

*function greet() {*

*var reply = [this.person, 'Is An Awesome', this.role].join(' ');*

*console.log(reply);*

*}*

*var i = {*

*person: 'Douglas Crockford', role: 'Javascript Developer'*

*};*

*greet.call(i); // Douglas Crockford Is An Awesome Javascript Developer*

2.2 Function.prototype.apply()

## **Syntax:** *fun*.apply(*thisArg,* [*argsArray*])

Example is covered in later.

2.3 Function() constructor

Functions can also be defined with a built-in JavaScript function constructor called Function(). This is a feature you don't really need. It is the same thing using function keyword.

For example,

var myFunction = new Function("a", "b", "return a \* b");  
  
var x = myFunction(4, 3);

2.4 this keyword

In JavaScript, the thing called this, is the object that "owns" the current code.

The value of this, when used in a function, is the object that "owns" the function.

Who owns the method?

The following method belongs to the HTML page which is the default global object. In a browser the page object is the browser window. The function above automatically becomes a window function.

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2);           // myFunction(10, 2) will return 20

Thus, myFunction() and window.myFunction() is the same function.

In the above case, when call myFunction(), 'this' is window object.

For example,

function myFunction() {  
    return this;  
}  
myFunction();                // Will return the window object

2.5 invoking function

You can get a function object literal, and directly call function properties.

You can use constructor to create function object and then call.

// This is a function constructor:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
  
// This creates a new object  
var x = new myFunction("John","Doe");  
x.firstName;

You can use .call() , .apply() to call a function.

function myFunction(a, b) {  
    return a \* b;  
}  
myObject = myFunction.call(myObject, 10, 2);     // Will return 20

or

function myFunction(a, b) {  
    return a \* b;  
}  
myArray = [10, 2];  
myObject = myFunction.apply(myObject, myArray);  // Will also return 20

2.6 closure

This deals with variable scope, local and global. Private variables can be made in closure.

Private/local variable: A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

Global variable: In a web page, global variables belong to the window object. Variables created without the keyword var, are always global, even if they are created inside a function.

For example, the following will all update the global variable.

var counter = 0;  
function add() {  
    counter += 1;  
}  
add();  
add();  
add();  
// the counter is now equal to 3

The above works but any other functions can change counter, so we want to protect it not being changed by unintended methods. However, the following does not work because it is resetting the value of counter to 0 in each add() call.

function add() {  
    var counter = 0;  
    counter += 1;  
}  
  
add();  
add();  
add();  
  
// the counter should now be 3, but it does not work !

Since every function can access the local variables as well as the variables above them, so we can come up with the following scratch of code,

*function add() {  
    var counter = 0;  
    function plus() {counter += 1;}  
    plus();      
    return counter;   
}*

If you can access plus from outside and call it three times that will be great. Then, we need closure.

A closure is a function having access to the parent scope, even after the parent function has closed.

The following example will work,

<!DOCTYPE html>

<html>

<body>

<p>Counting with a local variable.</p>

<button type="button" onclick="myFunction()">Count!</button>

<p id="demo">0</p>

<script>

var add = (function () {

var counter = 0;

return function () {return counter += 1;}

})();

function myFunction(){

document.getElementById("demo").innerHTML = add();

}

</script>

</body>

</html>

What is closure?

A closure is an inner function that has access to the outer (enclosing) function’s variables—scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function’s variables, and it has access to the global variables.

The inner function has access not only to the outer function’s variables, but also to the outer function’s parameters. Note that the inner function cannot call the outer function’s arguments object, however, even though it can call the outer function’s parameters directly.

Simple closure, closure1.js

|  |  |
| --- | --- |
| // define a function that increments a counter in a loop |  |
|  | function closureExample() { |
|  |  |
|  | var i = 0; |
|  |  |
|  | for (i = 0; i< 3 ;i++) { |
|  | setTimeout(function() { |
|  | console.log('counter value is ' + i); |
|  | }, 1000); |
|  | } |
|  |  |
|  | } |
|  | // call the example function |
|  | closureExample(); |

A few facts,

1/ The anonymous function inside of the loop is not executed right away. It is executed after the timeout for 1 sec each time.

2/ closureExample() exists when all three timeout happens.

3/ when the anonymous function executes the log then closureExample has long ended. Much like in different threads.

So how var I is passed to the anonymous function?

Here is how javascript engine works, it creates a link of the outer scope variable and keep that link in a special function scoped execution context. Such a function with a 'memory' about the environment where it was created is simply known as a closure.

How to see closure?

Use chrome developer tools, it can be found in 'scope variables'.

Closure is a reference not a copy.

The following output shows that, for main thread I goes from 1 to 3 in less than 1 sec, and when anonymous function gets executed, it is already 3.

xiaofengmaclap:Desktop xiaofeng$ node closure1

counter value is 3

counter value is 3

counter value is 3

If you would like to avoid this, you can see the following example,

closure2.js

function asyncOperation(counter) {

setTimeout(function() {

console.log('counter value is ' + counter);

}, 1000);

}

function otherClosureExample() {

var i = 0;

for (i = 0; i < 3 ;i++) {

asyncOperation(i);

}

}

otherClosureExample();

the output is,

xiaofengmaclap:Desktop xiaofeng$ node closure2

counter value is 0

counter value is 1

counter value is 2

The reason is that function asyncOperation(counter) gets a copy. closures keep references and not copies (even of primitive values), so make sure that that is really the intended logic.

Another closure example,

function add(value1) {

return function doAdd(value2) {

return value1 + value2;

};

}

var increment = add(1);

var foo = increment(2);

// foo equals 3

Ref:

[https://medium.freecodecamp.com/whats-a-javascript-closure-in-plain-english-please-6a1fc1d2ff1c#.cej5gtfi0](https://medium.freecodecamp.com/whats-a-javascript-closure-in-plain-english-please-6a1fc1d2ff1c" \l ".cej5gtfi0)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

3 JavaScript Object Prototypes

Every JavaScript object has a prototype. The prototype is also an object. All JavaScript objects inherit the properties and methods from their prototype. The Object.prototype is on the top of the prototype chain.Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype. Objects created with new Date() inherit the Date.prototype.

With a constructor function, you can use the new keyword to create new objects from the same prototype: PrototypeExample.js

*function Person(first, last, age, eyecolor) {*

*this.firstName = first;*

*this.lastName = last;*

*this.age = age;*

*this.eyeColor = eyecolor;*

*}*

*var myFather = new Person("John", "Doe", 50, "blue");*

*var myMother = new Person("Sally", "Rally", 48, "green");*

*console.log(myFather);*

*console.log(myMother);*

Adding new properties for some existing object, you can do this,

myFather.nationality = "English";

or

myFather.name = function () {  
    return this.firstName + " " + this.lastName;  
};

but you can't do the following,

Person.nationality = "English";

you have to add it to constructor, like the following,

function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
    this.nationality = "English"  
}

and in the constructor, you can also add method,

function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
    this.name = function() {return this.firstName + " " +this.lastName;};  
}

but with prototype, you can simply all these, for example,

function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}  
Person.prototype.nationality = "English";

or,

function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}

Person.prototype.name = function() {  
    return this.firstName + " " + this.lastName;  
};

4 Objects and class

In javascript, there is no class concept, but if a symbol is made as a class, then you can have a bunch of methods, and the clean way to do it is like the following,

*methodName : function() { code lines }*

*you can access it with,*

*objectName.methodName()*

*We have seen the example in the beginning of this chapter.*

*For example,*

*<p id="demo"></p>*

*<script>*

*var person = {*

*firstName: "John",*

*lastName : "Doe",*

*id : 5566,*

*fullName : function() {*

*return this.firstName + " " + this.lastName;*

*}*

*};*

*document.getElementById("demo").innerHTML = person.fullName();*

*</script>*

*a method name without () is the definiton of that method,*

*For example, the following will print out the function definition.*

*<p id="demo"></p>*

*<script>*

*var person = {*

*firstName: "John",*

*lastName : "Doe",*

*id : 5566,*

*fullName : function() {*

*return this.firstName + " " + this.lastName;*

*}*

*};*

*document.getElementById("demo").innerHTML = person.fullName;*

4.1 Object

*Everything is object. Except primitive values like string and numerics, all are objects.*

* Numbers can be objects (or primitive data treated as objects)
* Booleans can be objects (or primitive data treated as objects)
* Strings can be objects (or primitive data treated as objects)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are objects

*Create Object*

*There are different ways to create new objects:*

* Define and create a single object, using an object literal.
* Define and create a single object, with the keyword new.
* Define an object constructor, and then create objects of the constructed type.

*var person = {*

*firstName: "John",*

*lastName : "Doe",*

*id : 5566,*

*fullName : function() {*

*return this.firstName + " " + this.lastName;*

*}*

*};*

*this way person is already an object, you don't need to new();*

*The other way is to create object with new operator,*

varperson = newObject();person.firstName = "John";person.lastName = "Doe";person.age = 50;person.eyeColor = "blue";

4.2 Property

Properties are the values associated with a JavaScript object.

Properties can usually be changed, added, and deleted, but some are read only.

The syntax for accessing the property of an object is:

objectName.property // person.age

or

objectName["property"] // person["age"]

or

objectName[expression] // x = "age"; person[x]

5 Javascript array and JSON

5.1 define array,

varcars = ["Saab", "Volvo", "BMW"];

*or*

varcars = newArray("Saab", "Volvo", "BMW");

Object is like a hash map, but array can only be index by numeric offset. Array is mutable, you can change its element's value.

varperson = {firstName:"John", lastName:"Doe", age:46};

where you can use key:value, key to reference the value.

Array is very flex that it can be objects,

myArray[0] = Date.now;myArray[1] = myFunction;myArray[2] = myCars;

*Array methods:*

*array.sort()*

*array.*sort(function(a, b){returna - b});

*array.reverse()*

*array.length*

*array.push()*

*array.pop()*

*array.shift()*

unshift("Lemon") : add new element to the left.

*array.toString()*

*array.join(“\*”)*

*Array.isArray(myArray)*

*Note: to declare array, do not use new Array() but use [].*

*To check if a data structure is an array, you can use isArray(arr) or use the following,*

functionisArray(x) {returnx.constructor.toString().indexOf("Array") > -1;}

*1/ typeof only gives object.*

*2/ comparing function*

function(a, b){return a – b} , ascending

function(a, b){return b – a}, descending

function(a, b){return a.year – b.year}, comparing property value

comparing string properties,

cars.sort(function(a, b){  
    var x = a.type.toLowerCase();  
    var y = b.type.toLowerCase();  
    if (x < y) {return -1;}  
    if (x > y) {return 1;}  
    return 0;  
});

JSON (**J**ava**S**cript **O**bject **N**otation)

It is a list of([]) of objects {key:value, key:value...}

{  
"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]  
}

convert JSON to javascript

using built-in method, JSON.parse() ,

var text = '{ "employees" : [' +  
'{ "firstName":"John" , "lastName":"Doe" },' +  
'{ "firstName":"Anna" , "lastName":"Smith" },' +  
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

var obj = JSON.parse(text);

10.6 data types

5 data types: string, number, boolean, object, function

3 types of objects: Array, Object, Date.

2 types can not contain values: null, undefined

To find data type, use typeof operator,

For example,

<!DOCTYPE html>

<html>

<body>

<p>The constructor property returns the constructor function for a variable or an

object.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

"john".constructor + "<br>" +

(3.14).constructor + "<br>" +

false.constructor + "<br>" +

[1,2,3,4].constructor + "<br>" +

{name:'john', age:34}.constructor + "<br>" +

new Date().constructor + "<br>" +

function () {}.constructor;

</script>

</body>

</html>

7 Strict mode, debug,

 "use strict"; you can not use undeclared variables. It can be declared in a function as well.

8 JS and DOM navigation

Use the relative tree structure,

* parentNode
* childNodes[nodenumber]
* firstChild
* lastChild
* nextSibling
* previousSibling

here are two special properties that allow access to the full document:

* document.body - The body of the document
* document.documentElement - The full document

<script>  
alert(document.body.innerHTML);  
</script>

and

<script>  
alert(document.documentElement.innerHTML);  
</script>

9 Event and EventListener

9.1 How to attach handler function to the event of dom element?

JS addEventListener(event, handlerFunc) gives a lot of power to the programming.You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events. You can add event listeners to any DOM object not only HTML elements. i.e the window object.

Syntax,

element.addEventListener(event, function, useCapture);

For example, you can add listener to the window object,

*<!DOCTYPE html>*

*<html>*

*<body>*

*<p>This example uses the addEventListener() method on the window object.</p>*

*<p>Try resizing this browser window to trigger the "resize" event handler.</p>*

*<p id="demo"></p>*

*<script>*

*window.addEventListener("resize", function(){*

*document.getElementById("demo").innerHTML = Math.random();*

*});*

*</script>*

*</body>*

*</html>*

note that, the event is “click” it is not dom's 'onclick'.

9.2 Remove event listener

Syntax,

element.removeEventListener("mousemove", myFunction);

You can also detect if you have the method for that element, the following is the example of cross-browser solution,

*<!DOCTYPE html>*

*<html>*

*<head>*

*<script>*

*document.getElementById("myBtn").addEventListener("click", displayDate);*

***function*** *displayDate() {*

*document.getElementById("demo").innerHTML = Date();*

*}*

*</script>*

*</head>*

*<body>*

*<p>This example uses the addEventListener() method to attach a click event to a button.</p>*

*<button id="myBtn">Try it</button>*

*<p id="demo"></p>*

*</body>*

*</html>*

Note that the order of javascript <script></script> matters. It is usually in the <head></head> because it gets loaded first. And if you put the <script></script> in the <body>, it must be put after the <button></button>, otherwise it will not work. The reason is that it follows the order when parsing html file, when it tries to getElementById, that button isn't available yet.

There are many way to manipulate dom element's event.

1/ <button onclick="displayDate()">Try it</button>, using dom's onclick event. This is the html way.

2/ In javascript, assign the javascript handler function to the dom element's onclick event.

<script>  
document.getElementById("myBtn").onclick = displayDate;  
</script>

3/ use addEventListener()

10 DOM CSS manipulation

element.style.property=value

For a complete list of css property and their usage, please refer to,

http://www.w3schools.com/jsref/dom\_obj\_style.asp

11 Documents

**ref:**

http://www.w3schools.com/js/js\_object\_prototypes.asp

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Object/prototype

http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/

https://javascriptweblog.wordpress.com/2010/06/07/understanding-javascript-prototypes/

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Function/call (done)