Home | JS Testing

Jae's Tech Note

Apply vs Call

The difference is that apply lets you invoke the function with arguments as an array; call requires the parameters be listed explicitly.
A useful mnemonic is AACC = "Apply for array and Call for comma."

See MDN's documentation on apply and call.

Pseudo syntax:

theFunction.apply(valueForThis, arrayOfArgs)
theFunction.call(valueForThis, arg1, arg2, ...)

.apply() and .call() are executed immediately, while .bind() returns a function, which can be executed at a later time. And, if you want to make .bind() execute immediately too, you can do so this way.

theFunction.bind(valueForThis, arg1, arg2, ...)();

JavaScript vs JAVA

JavaSript is a client based browser language and
Java is an object based programming language may used on many differnet plattforms.

.bind() preserves the context of this for future execution

Before:

```
var cat = {
    name: 'Neo',
    showName: function () {

        var self = this

        // this could be any context-switching function - async, callba
        setTimeout(function () {
            console.log(self.name)
        // `this.name` will be undefined
        })
```

```
        }
    }
```

cat.showName() -> Neo

After:

```
var cat = {
    name: 'Neo',
    showName: function () {

        var printer = function () {
            console.log(this.name)
        }.bind(this)
        // bind the current `this` (reference to `cat`) to the `printer`
        // while we have a valid reference

        setTimeout(printer)
    }
}
```

cat.showName() -> Neo

The bind() method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

```
var module = {
    x: 42,
    getX: function() {
        return this.x;
    }
}

var retrieveX = module.getX;
console.log(retrieveX()); // The function gets invoked at the global scope
// expected output: undefined

var boundGetX = retrieveX.bind(module);
console.log(boundGetX());
// expected output: 42
```

Deleting Properties : delete

The delete keyword deletes a property from an **object** not regular variable:

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
delete person.age; // or delete person["age"];

x is not an Object... so 'delete x' not affect.
The output would be 1. The delete operator is used to delete the property of an object. Here x is not an object, but rather it's the global variable of type number.

```
var x = 1;
var output = (function(){
    delete x;
    return x;
})();

console.log(output);
```

```
var Employee = {
    company: 'xyz'
}
var emp1 = Object.create(Employee);
delete emp1.company
console.log(emp1.company);
```

The output would be xyz. Here, emp1 object has company as its prototype property. **The delete operator doesn't delete prototype property**.

```
var trees = ["redwood","bay","cedar","oak","maple"];
delete trees[3];
console.log(trees); // ["redwood", "bay", "cedar", empty, "maple"]
console.log(trees[3] == null); // True
console.log(trees[3] == 'undefined'); // False
```

```
var bar = true;
console.log(bar + 0);
console.log(bar + "xyz");
console.log(bar + true);
console.log(bar + false);
```

===> 1, "truexyz", 2, 1

var z = 1, y = z = typeof y;
console.log(y);

===> undefined , cause operatr from ~~left to right~~ right to left.. typeof y is undefined

```
var foo = function bar(){
    // foo is visible here
    // bar is visible here
        console.log(typeof bar()); // Work here :)
};
// foo is visible here
// bar is undefined here
```

What is difference between the **function declarations below**?

```
var foo = function(){
// Some code
};
function bar(){
// Some code
};
```

The main difference is the **function foo is defined at run-time whereas function bar is defined at parse time.**
To understand this in better way, let's take a look at the code below:

Run-Time function declaration

```
foo(); // Calling foo function here will give an Error
var foo = function(){
    console.log("Hi I am inside Foo");
};
```

```
Parse-Time function declaration

bar(); // Calling bar function will not give an Error
function bar(){
    console.log("Hi I am inside bar");
};
------------------------------


Another advantage of this first-one way of declaration is
that you can declare functions based on certain conditions.
For example:

if(testCondition) {// If testCondition is true then
    var foo = function(){
    console.log("inside Foo with testCondition True value");
    };
}else{
    var foo = function(){
    console.log("inside Foo with testCondition false value");
    };
}

However, if you try to run similar code using the format below,
you'd get an error:

if(testCondition) {// If testCondition is true then
    function foo(){
    console.log("inside Foo with testCondition True value");
    };
}else{
    function foo(){
    console.log("inside Foo with testCondition false value");
    };
}
```

What is function hoisting in JavaScript?

Basically, the JavaScript interpreter looks ahead to find all variable declarations and then hoists them to the top of the function where they're declared.

```
var salary = "1000$";
(function () {
    console.log("Original salary was " + salary);
    var salary = "5000$";
    console.log("My New Salary " + salary);
})();
```

변수를 만나면 아래 scope 에 선언된지 먼저 찾아본다.
salary가 쓰이기 전에 뒷부분에 선언되 있으므로
undefined가 된다.
만약 변수가 쓰이고 난뒤에 선언이 안되있다면
윗부의 선언을 써서 "1000$" 가 된다.

==> Original salary was undefined

==> My New Salary 5000$
--> Because second var salary defind and used it before.

What is the instanceof operator in JavaScript? What would be the output of the code below?

```
var dog = new Animal();
dog instanceof Animal // Output : true

var name = new String("xyz");
name instanceof String // Output : true

var color1 = new String("green");
color1 instanceof String; // returns true
var color2 = "coral"; //no type specified
color2 instanceof String; // returns false (color2 is not a String
```

Object length

```
var counterArray = {
    A : 3,
    B : 4
};
counterArray["C"] = 1;
===> {A: 3, B: 4, C: 1}

console.log(Object.keys(counterArray).length); // 3
console.log(counterArray.length); // undefined

------------------------------------------------------------

function getSize(object){
    var count = 0;
    for(key in object){
      // hasOwnProperty method check own property of object
      if(object.hasOwnProperty(key)) count++;
    }
    return count;
}

------------------------------------------------------------

Object.length = function(){
    var count = 0;
    for(key in object){
        // hasOwnProperty method check own property of object
        if(object.hasOwnProperty(key)) count++;
    }
    return count;
}
//Get the size of any object using
```

```
console.log(Object.length(counterArray))
```

difference "currentTarget" and "target" property in javascript

target is the element that triggered the event (e.g., the user clicked on)
currentTarget is the element that the event listener is attached to.

IIFE (Immediately Invoked Function Expression)

target is the element that triggered the event (e.g., the user clicked on)
currentTarget is the element that the event listener is attached to.

```
function foo(){
}();
// ====> Error

// IIFE
(function foo(){
})();
// ===> OK
```

undefined, undeclaired, null

undeclaired : not defined
undefined : defined but no vlaue assigned (foo === undefined)
null : (foo === null) defined... null is a value.. value place holder... with "nothing" value, falsy

let foo; // undefined
const bar = null; // null
console.log(foo == bar); // true -- equality
console.log(foo === bar); // false -- equality and type

Simple HTTP Server

Live Server -- Visual Studio Code !
https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer

PHP Server -- Visual Studio Code !
https://marketplace.visualstudio.com/items?itemName=brapifra.phpserver

python -m SimpleHTTPServer 8888

npm install http-server -g
http-server ./ -p 8888

---

express : npm package to handle HTTP requests.. and server
REDUX : handle all components State

- easy to debug - Redux Devtools
- Store, actions, and reducers

Javascript General

getElementById
getElementByClassName
getElementByTagName
getElementByTagNameNS <---??? by NameSpace

type check?

```
var a = [1,2,3,4,5];
console.log(a.constructor);
console.log(a.constructor == Array); // true

var b = {
    name : "Jae",
    age: 52
}
console.log(b.constructor);
console.log(b.constructor == Object); // true
```

---

map() - loop through array ... that's it

(ex)

```
var a = [1,2,3,4,5,6];
var b = a.map(function(n){
console.log(n);
return n;
});
console.log(b);

    ----------------------------------------
    map() ****************

    var cars = [
        {name: "Toyota", color:"Gray"},
        {name: "Ferrari", color:"Red"},
        {name: "Lambo", color:"Yello"},
        {name: "Honda", color:"Black"}
    ];
    var carColor = cars.map(function(x){
        return x.name + ' is color ' + x.color;
    });
    console.log(carColor[2]);

    ----------------------------------------
    ES6 ****************

    var cars = [
        {name: "Toyota", color:"Gray"},
        {name: "Ferrari", color:"Red"},
        {name: "Lambo", color:"Yello"},
        {name: "Honda", color:"Black"}
    ];
    var carColor = cars.map(x => x.name + ' is color ' + x.color);

    console.log(carColor[2]);

    ****************************************
    // map Example 2 - convert Celsius to Fahrenheit
    var temps = [20, 18, 26, 40, 0, -10];
    var convert = function(tempArray){
        if(tempArray.constructor != Array){
            throw '"Conver" function requires an array !';
        } else {
            return tempArray.map(function(temp){
                return Math.round(temp * (9/5) + 32);
            });
        }
    }
    var newTemps = convert(temps);

    console.log(temps);
    console.log(newTemps);

    ****************************************
    // map Example 3

    var students = [
        { name: 'Steve Wonder',     grade1: 75,     grade2: 55 },
        { name: 'Jimmi Hendrix',    grade1: 35,     grade2: 75 },
        { name: 'Eric Clapton',     grade1: 66,     grade2: 65 },
        { name: 'Jae Moon',         grade1: 97,     grade2: 85 },
        { name: 'Myong Lee',        grade1: 85,     grade2: 75 },
        { name: 'Soo Won',          grade1: 98,     grade2: 100 },
        { name: 'Sae Won',          grade1: 100,    grade2: 99 }
    ];

    var passingGrade = 70;

    var results = students.map(function(student){
        var average = (student.grade1 + student.grade2) / 2;
```

```
        if(average > passingGrade){
            return {
                name: student.name,
                average: average,
                passed: true
            }
        } else {
            return {
                name: student.name,
                average: average,
                passed: false
            }
        }
    });

    console.log(results);

    ============================================================


    const companies = [
        {name: "Company 01", category: "Finance", start: 1981, end: 2003},
        {name: "Company 02", category: "Retail", start: 1933, end: 2013},
        {name: "Company 03", category: "Auto", start: 1954, end: 2004},
        {name: "Company 04", category: "Technology", start: 1966, end: 2002},
        {name: "Company 05", category: "Spa", start: 1971, end: 2011},
        {name: "Company 06", category: "Hotel", start: 1985, end: 2007}
    ];

    const ages = [33, 12, 15, 20, 21, 25, 28, 45, 60, 82, 32, 24, 18];

    // for **********************************

    for(let i=0; i= 21){
            canDrink.push(ages[i]);
        }
    }
    console.log("for loop",canDrink);

    // using filter ... good **********************************

    const canDrinkAry = ages.filter(function(age){
        if( age >= 21 ){
            return true;
        }
    });
    console.log("filter",canDrinkAry);

    // using filter & ES6 ... better **********************************

    const canDrinkAry1 = ages.filter(age => age > 21);
    console.log("filter ES6",canDrinkAry1);
```

Array.from(htmlCollection).forEach(function(){

JS ==> jQuery

```
document.querySelector("#select")  ===> $("#select").[0]
document.querySelectorAll(".name")  ===> $("#select")  ===> array  형식 forEach 가능
book.textContent = "test"  ===> $().text('test')
```

```
.innerHTML += " something"      <==== Append
.nodeType   ---> returns number of type
.nodeName ---> returns 'div'
.hasChildNodes  ---> true / false

.cloneNode(true)  -----> true : include all children / false : just target element
.parentNode == .parentElement
.childNodes   ===> return with empty line break elements also
.children        ====> return just elemets
.nextSibling    ===> return with empty line break elements also
.nextElementSiblig         ====> return just elemets
.removeChild('li')
.addEventListener('click', function(e){ .....  e.target

Event Bubbling...  Click event가 눌러진 Child element 만이 아니라 모든 부모가 클릭된걸로 인식..
<--->
Event Delegation

document.forms['someID']
.querySelector('input[type="text"]').value

document.createElement(...

li.appendChild(....

.style.marginTop = "60px"
.className
.classList.add(....
.classList.remove(....

getAttribute('class')...
getAttribute('href')...
setAttribute('class','someClassName')

.hasAttribute(..
.removeAttribute(...
.checked   <---- checkbox checked  true/false


-------------------------------------------------------------------


 <div data-targetId = "#some">asdc</div>
 e.target.dataset.targetId
 document.addEventListener('DOMContentLoaded', function(){
     ..............
 })
```

queus : shift, unshift... FIFO
stack : push, pop... LIFO

Prototype

```
function Employee(name){
    this.name = name;
    this.getName = function (){
        return this.name;
    }
}

var e1 = new Employee("Jae");
var e2 = new Employee("Moon");

document.write("E1 Name = "+ e1.getName() + "<br/>");
document.write("E2 Name = "+ e2.getName() + "<br/>");

//===> this works but e1, e2... having copies of Employee object... memory leak an

function Employee(name){
    this.name = name;
}

Employee.prototype.getName = function(){
    return this.name;
}

var e1 = new Employee("Jae");
var e2 = new Employee("Moon");

document.write("E1 Name = "+ e1.getName() + "<br/>");
document.write("E2 Name = "+ e2.getName() + "<br/>");

/* ===> Prototype
     1. No matter how many objects create, functions are load only once into memor
     2. Allows to override function if required.


prototype is a property of a Function object.
It is the prototype of objects constructed by that function.

__proto__ is internal property of an object, pointing to its prototype.
Current standards provide an equivalent Object.getPrototypeOf(O) method,
though de facto standard __proto__ is quicker.

You can find instanceof relationships by comparing a function's prototype to an
object's __proto__ chain, and you can break these relationships by changing protot
*/
```

Prototype

Almost every object(default JS obj doesn't have prototype) is linked to another object. That linked object is called the prototype
objects inherit properties and methods from it's aprototyp ancestry.
A prototype is automatically assigned to any object
You can define an objects prototype

JavaScript object creation patterns - factory , constructor pattern, prototype pattern

Factory Pattern (Style)

```
var peopleFactory = function (name, age, state){
    var temp = {}; // is same as var temp = new Object;
    var temp.age = age;
    var temp.name = name;
    var temp.state = state;
    var printPerson = function(){
        console.log(this.name + " , " + this.age + " , " + this.state);
    }
    return temp;
}
var p1 = peopleFactory("John", 23, "CA");
var p2 = peopleFactory("Joan", 21, "NC");

p1.printPerson();
p2.printPerson();
```

Construcor Pattern (Style)

```
var peopleConstructor = function (name, age, state){
    this.name = name;
    this.age = age;
    this.state = state;
    this.printPerson = function(){
        console.log(this.name + " , " + this.age + " , " + this.state);
    }
    return temp;
}
var p1 = new peopleConstructor("John", 23, "CA");
var p2 = new peopleConstructor("Joan", 21, "NC");

p1.printPerson();
p2.printPerson();
```

===> Problem.... keep copying objects when it create using new

Prototype Pattern (Style)

```
var peopleProto = function (){

}

peopleProto.prototype.age = 0;
peopleProto.prototype.name = '';
peopleProto.prototype.stage = '';
```

```
peopleProto.prototype.printPerson = function (){
    console.log(this.name + " , " + this.age + " , " + this.state);
}

var p1 = new peopleProto();
p1.name = "John";
p1.age = 23;
p1.state = "CA";

console.log('name' in p1); // true
console.log(p1.hasOwnProperty('name')); // true

var p2 = new peopleProto();
p2.name = "Joan";
p2.age = 21;
p2.state = "NC";

p1.printPerson();
p2.printPerson();
```

===> Problem.... keep copying objects when it create using new

Pizza.getCrust = function(){ return this.crust; };
Pizza.**prototype**.getCrust = function(){ return this.crust; };
==> put function to shared space ... which is prototype

Dynamic Prototype Pattern (Style)

```
var peopleDynamicProto = function (name, age, state){
    this.name = name;
    this.age = age;
    this.state = state;

    if(typeof this.printPerson !== 'function'){
        peopleDynamicProto.prototype.printPerson = function (){
            console.log(this.name + " , " + this.age + " , " + this.state);
        }

    }
}

var p1 = new peopleDynamicProto("John", 23, "CA");
var p2 = new peopleDynamicProto("Joan", 21, "NC");

p1.printPerson();
p2.printPerson();

console.log('name' in p1); // true
console.log(p1.hasOwnProperty('name')); // true

------------------------------------------------------------------------
Function callback

var add = function(a,b){return a+b};
var minus = function(a,b){return a-b};

var cal = function(a,b,func){
    return func(a,b);
}

cal(1,2,add);
cal(1,2,minus);
```

only once create prototype

Etc

CDN : content delivery network
A content delivery network (CDN) is a system of distributed servers (network) that deliver pages and other Web content to a user, based on the geographic locations of the user, the origin of the webpage and the content delivery server.