### ****.call() .apply() .bind()****

### **Used for:**

* **Explicit binding** We can explicitly tell the JavaScript engine to set this to point to a certain value

### bind() allows us to easily set which specific object will be bound to this when a function or method is invoked

$("button").click(user.clickHandler.bind(user));

* apply() and call() are nearly identical and are frequently used in JavaScript for borrowing methods

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

What that means, is that we can call any function, and explicitly specify what this should reference within the calling function.

The main differences between bind() and call() is that the call() method:

1. Accepts additional parameters as well
2. Executes the function it was called upon right away.
3. The call() method does not make a copy of the function it is being called on.

call() and apply() serve the **exact same purpose.** The **only difference between how they work is that** call() expects all parameters to be passed in individually, whereas apply() expects an **array of all of our parameters**.

You can use call() and apply() to invoke the function immediately. bind() returns a bound function that, when executed later, will have the correct context (**"this"**) for calling the original function. So bind() can be used when the function needs to be called later in certain events when it's useful.

**call()** function.call(thisArg, arg1, arg2, ...)

### executes right away

### accepts list of arguments

### does NOT make copy of a function that is being called

**apply()** function.apply(thisArg, [argsArray])

### executes right away

### accepts single array of arguments

### does NOT make copy of a function that is being called

**bind()** function.bind(thisArg[, arg1[, arg2[, ...]]])

### stores value in variable for later execution

### accepts list of arguments

### makes a copy of a function that is being called later (creates a new function)

The **bind()** method creates a new function that, when called, has its this keyword set to the provided value. This is extremely powerful. It lets us explicitly define the value of this when calling a function.

**Difference between call** and **apply** is just that **apply** accepts parameters **in the** form of an **array** while **call** simply can accept a comma separated list of arguments. A **bind** function basically binds the context of something and then stores it into a variable for execution at a later stage.

Use .bind() when you want that function to later be called with a certain context, useful in events. Use .call() or .apply() when you want to invoke the function immediately, and modify the context (this)

.call()/.apply() call the function immediately, whereas .bind() returns a function that, when later executed, will have the correct context set for calling the original function. This way you can maintain context in async callbacks and events.

function.call(thisArg, arg1, arg2, ...)

function.apply(thisArg, [argsArray])

function.bind(thisArg[, arg1[, arg2[, ...]]])

## . call() or Function.prototype.call() example:

var obj = {name:"Luke"};

var greeting = function(city,country){

return "Hi "+this.name+", welcome to "+city+" in the " + country;

};

greeting.call(obj,"LONDON", "UK"); //Hi Luke, welcome to LONDON in the UK – RETURNS THE NEW SENTENCE

The first parameter in call() method sets the **"this"** value, which is the object, on which the function is invoked upon. In this case, it's the **"obj"** object above.

The rest of the parameters are the arguments to the actual function.

let food = {name: 'chips'}

let message = function(size, when){

return "I'll have "+this.name+" on a "+size+" plate "+when+"!";

}

message.call(food, "big", "today"); //I'll have chips on a big plate today!

## .apply() or Function.prototype.apply() example:

var obj = {name:"Frank"};

var args = ["LONDON", "UK"];

var greeting = function(city,country){

return "Hi "+this.name+", welcome to "+city+" in the " + country;

};

greeting.apply(obj, args); //Hi Frank, welcome to LONDON in the UK – RETURNS THE NEW SENTENCE

Similarly to call() method the first parameter in apply() method sets the **"this"** value which is the object upon which the function is invoked. In this case it's the **"obj"** object above. 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.

//example 2 // APPLY() is the same as .call(), but only accepts ARRAY of arguments

const pet = {pet: "dog", name: 'Rambo'};

const foods = ['tuna', 'chocolate', 'cake'];

const sentence = function(food1, food2, food3){

return "My "+this.pet+ " "+this.name+" likes "+food1+ " "+food2+" "+food3;

}

sentence.apply(pet, foods); //My dog Rambo likes tuna chocolate cake

## .bind() or Function.prototype.bind() example:

var obj = {name:"Frank"};

var greeting = function(city,country){

return "Hi "+this.name+", welcome to "+city+" in the " + country;

};

var bound = greeting.bind(obj);

bound("LONDON", "UK"); // Hi Frank, welcome to LONDON in the UK

In the above code sample for bind() we are returning a bound function with the context which will be invoked later.

The first parameter to the bind() method sets the value of **"this"** in the target function when the bound function is called. Please note that the value for first parameter is ignored if the bound function is constructed using the **"new"** operator.

The rest of the parameters following the first parameter in bind() method are passed as arguments which are prepended to the arguments provided to the bound function when invoking the target function.

const seafood = {salty: false, name: 'octopus'};

let result = function(flavour, typeOfFood){

if (this.salty){

return this.name+" is a salty "+typeOfFood+" with "+flavour+" flavour!!"

} else {

return this.name+" is a NON-salty "+typeOfFood+" with "+flavour+" flavour!!"

}

}

let bound = result.bind(seafood);

bound('tangy', 'sea dish'); //"octopus is a NON-salty sea dish with tangy flavour!!"