Introduction to Closures in JavaScript

17th Day

A closure is a combination of a function bundled together with references to its surrounding state i.e. the lexical environment. In other words, a closure provides you access from an inner function to an outer function's scope.Most of the Developers use closures in JavaScript consciously or unconsciously. It provides better control over the code when using them.

Example:

```
function foo()
{
var x = 10;
function inner(){
return x;
}
return inner;
}
var get_func_inner = foo();
console.log(get_func_inner());
console.log(get_func_inner());
console.log(get_func_inner());
```

The output will be:

```
10

10

10
```

Here, you can access the variable x which is defined in function foo() through function inner() as the later preserves the scope chain of enclosing function at the time of execution of enclosing function. Thus, the inner function knows the value of x through it's scope chain. This is how you can use closures in JavaScript.

Closures allow you to associate the lexical environment with a function that operates on that data. This has obvious parallels toobject-oriented programming, where objects allow us to associate the object's properties with one or more methods.

Consequently, you can use a closure anywhere that you might normally use an object with only a single method.

Examle:

```
function makeSizer(size) {
return function() {
document.body.style.fontSize = size + 'px';
};
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);
```

The above example is generally attached as a callback: a single function which is executed in response to the event.

Scope Chain.

Closures in JavaScript have three scopes such as:

1). Local Scope

2). Outer Functions Scope

3). Global Scope

A common mistake is not realizing that, in the case where the outer function is itself a nested function, access to the outer function's scope includes the enclosing scope of the outer function, effectively creating a chain of function scopes.

Exaample:

```
// global scope
var x = 10;
function sum(a){
return function(b){
return function(c){
// outer functions scope
return function(d){
// local scope
return a + b + c + d + x;
}}}}

console.log(sum(1)(2)(3)(4)); // log 20
```

It can also be written without anonymous functions:

```
// global scope
var x = 10;
function sum(a){
return function sum2(b){
return function sum3(c){
// outer functions scope
return function sum4(d){
// local scope
return a + b + c + d + x;
}
}
}
}

var s = sum(1);
var s1 = s(2);
var s2 = s1(3);
var s3 = s2(4);
console.log(s3) //log 20
```

In the above example, there is a series of nested functions all of which have access to the outer scope of a function. Thus, you can say that closures have access to all outer function scopes within which they were declared.

Closure Within a Loop

You can use closures in JavaScript to store an anonymous function at every index of an array. Let’s take an example and see how closures are used within a loop.

Example:

```
function outer()
{
var arr = [];
var i;
for (i = 0; i < 3; i++)
{
// storing anonymus function
arr[i] = function () { return i; }
}

// returning the array.
return arr;
}
var get_arr = outer();
console.log(get_arr[0]());
console.log(get_arr[1]());
console.log(get_arr[2]());
```

Output will be:

```
3
3
3
3
```

I hope you understood how closures in JavaScript works and how they are used to get a better control of the code.

Developed by: Sayantini (Trainer at Edureka)

https://www.linkedin.com/in/sayantini-deb-b08230159/

Regards Lux Tech Academy

Best wishes.