@Code.Clash

Closures

JavaScript

JavaScript Closure

Before you learn about closures, you need to understand two concepts:

- Nested Function
- Returning a function

```
function OuterFunction() {

    var outerVariable = 1;

    function InnerFunction() {
        alert(outerVariable);
    }

    return InnerFunction();
}
```

- In above example Inner function can access variables and parameters of an outer function

JavaScript Closure

Closure is one of important concept in JavaScript.

It is widely discussed and still confused concept. Let's understand what the closure is.

“Closure means that an inner function always has access to the variables and parameters of its outer function, even after the outer function has returned.”

Example

Now, as per the definition above, InnerFunction() can access outerVariable even if it will be executed separately. Consider the following example.

```
function OuterFunction() {
    var outerVariable = 100;

    function InnerFunction() {
        alert(outerVariable);
    }
    return InnerFunction;
}
// innerFunc reffer to the InnerFuntion return by OuterFunction
var innerFunc = OuterFunction();

innerFunc(); // 100
```

- So now, when you call innerFunc(), it can still access outerVariable which is declared in OuterFunction().
- This is called Closure.

Example

Inner function does not keep the separate copy of outer variables but it reference outer variables, that means value of the outer variables will be changed if you change it using inner function.

```
function Counter() {
    var counter = 0;

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

    return IncreaseCounter;
}

var counter = Counter();
alert(counter()); // 1
alert(counter()); // 2
alert(counter()); // 3
alert(counter()); // 4
```

- Outer function Counter returns the reference of inner function `IncreaseCounter()`.
- IncreaseCounter increases the outer variable `counter` to one.

When to use Closure?

Closure is useful in hiding implementation detail in JavaScript. In other words, it can be useful to create private variables or functions

```
var counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  };
})();

alert(counter.value()); // 0
counter.increment();
counter.increment();
alert(counter.value()); // 2
counter.decrement();
alert(counter.value()); // 1
```

- In the above example, increment(), decrement() and value() becomes public function
- because they are included in the return object,
- whereas changeBy() function becomes private function because it is not returned and only used internally by increment() and decrement().

I Have Already Posted On Important CSS And Javascript Topics, Roadmap, Useful Resources, And Cheat Sheets.

If You Have Any Queries Then Let Me Know In The Comment Box.

@Code.Clash

Check Out My Profile For Such Excellent Posts On Web Development.

Imtiyaz Nandasaniya