JavaScript Immediately Invoked Function Expression

Summary: in this tutorial, you will learn about JavaScript immediately invoked function expressions (IIFE).

TL;DR

A JavaScript immediately invoked function expression is a function defined as an expression and executed immediately after creation. The following shows the syntax of defining an immediately invoked function expression:

```
(function(){
    //...
})();
```

Why IIFEs

When you define a function, the JavaScript engine adds the function to the global object. See the following example:

```
function add(a,b) {
    return a + b;
}
```

In web browsers, the JavaScript engine adds the `add()` function to the `window` global object:

```
console.log(window.add);
```

Likewise, if you declare a variable outside of a function using the `var` keyword, the JavaScript engine also adds the variable to the global object:

```
var counter = 10;
console.log(window.counter); // 10
```

If you have many global variables and functions, the JavaScript engine will only release the memory allocated for them until the global object loses its scopes.

As a result, the script may use the memory inefficiently. On top of that, having global variables and functions will likely cause name collisions.

One way to prevent the functions and variables from polluting the global object is to use immediately invoked function expressions.

In JavaScript, you can have the following expressions:

```
'This is a string';
(10+20);
```

This syntax is correct even though the expressions have no effect. A function can be also declared as an expression which is called a function expression:

```
let sum = function(a, b) {
    return a + b;
}
```

In this syntax, the part on the right side of the assignment operator(`=`) is a function expression. Because a function is an expression, you can wrap it inside parentheses:

```
let sum = (function(a, b) {
    return a + b;
});
```

In this example, the `sum` variable is referenced as the anonymous function that adds two arguments.

In addition, you can execute the function immediately after creating it:

```
let sum = (function(a,b){
    return a + b;
})(10, 20);

console.log(sum);
```

In this example, the `sum` variable holds the result of the function call.

The following expression is called an immediately invoked function expression (IIFE) because the function is created as an expression and executed immediately:

```
(function(a,b){
        return a + b;
})(10,20);
```

This is the general syntax for defining an IIFE:

```
(function(){
    //...
})();
```

Note that you can use an arrow function to define an IIFE:

```
(() => {
    //...
})();
```

By placing functions and variables inside an immediately invoked function expression, you can avoid polluting them to the global object:

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

    function add(a, b) {
        return a + b;
    }

    console.log(add(10,20)); // 30
}());
```

Named IIFE

An IIFE can have a name. However, it cannot be invoked again after execution:

```
(function namedIIFE() {
    //...
})();
```

IIFE starting with a semicolon (;)

Sometimes, you may see an IIFE that starts with a semicolon(;):

```
;(function() {
/* */
})();
```

In this syntax, the semicolon is used to terminate the statement in case two or more JavaScript files are blindly concatenated into a single file.

For example, you may have two file `lib1.js` and `lib2.js` which use IIFEs:

```
(function(){
    // ...
})()
```

```
(function(){
    // ...
})()
```

If you use a code bundler tool to concatenate code from both files into a single file, without the semicolon (`;`) the concatenated JavaScript code will cause a syntax error.

IIFE in actions

Suppose that you have a library called calculator.js with the following functions:

```
function add(a, b) {
    return a + b;
}

function mutiply(a, b) {
    return a * b;
}
```

And you load the `calculator.js` in an HTML document.

Later, you also want to load another JavaScript library called `app.js` to the same document:

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>JavaScript IIFE</title>
</head>
<body>
  <script src="calculator.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

The `app.js` also has the `add()` function:

```
function add() {
    return 'add';
}
```

When you use the `add()` function in the HTML document, it returns the `'add'` string instead of the sum of two numbers:

```
let result = add(10, 20);
console.log(result); // 'add'
```

This is because the `add()` function in the `app.js` overrides the `add()` function in the `calculator.js` library.

To fix this, you can apply IIFE in the calculator.js as follows:

```
const calculator = (function () {
    function add(a, b) {
        return a + b;
    }

    function multiply(a, b) {
        return a * b;
    }
    return {
        add: add,
        multiply: multiply
    }
})();
```

The IIFE returns an object that contains the `add` and `multiply` methods that reference the `add()` and `multiply()` functions. In the HTML document, you can use the `calculator.js` library as follows:

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
```

```
  <title>JavaScript IIFE</title>
</head>
<body>
  <script src="js/calculator.js"></script>
  <script src="js/app.js"></script>
  <script>
    let result = calculator.add(10, 20); // add in app.js
    console.log(result); // 30
    console.log(add()); // add in the app.js
  </script>
</body>
</html>
```

The `calculator.add()` called the `add()` function exported by the `calculator.js` while the second call to the `add()` function references the `add()` function in the `app.js`.

jQuery & IIFE

The following HTML document uses the jQuery library:

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>JavaScript IIFE - jQuery</title>
</head>
<body>
  <h1>jQuery Demo</h1>
  <script src="https://code.jquery.com/jquery-3.4.1.slim.js"
    integrity="sha256-BTlTdQO9/fascB1drekrDVkaKd9PkwBymMlHOiG+qLI=" crossorigin="anonymous"><
  <script>
    let counter = 1;
    $('h1').click(function () {
      $(this).text('jQuery Demo' + ' Clicked ' + counter++);
    });
  </script>
</body>
</html>
```

When you import the jQuery library, you can access many useful jQuery functions via the `$` or `jQuery` object. Under the hood, jQuery uses the IIFE to expose its functionality.

By doing this, jQuery just needs to use one global variable (`$`) to expose a ton of functions without polluting the global object.

The following example illustrates how to change the jQuery $ object to _ inside the IIFE:

```
(function (_) {
     let counter = 1;
     _('h1').click(function () {
       _(this).text('jQuery Demo' + ' Clicked ' + counter++);
     });
 })(jQuery);
```

In this example, we passed the jQuery object into the IIFE and used the `_` argument instead.

In this tutorial, you will have learned about the JavaScript immediately invoked function expressions (IIFE) and their purposes.