JavaScript 函數基礎觀念

向皓田

2018/11/2

基本語法

```
function functionName(arg0, arg1, ..., argN) }
    statements;
}
```

函數範例

```
function hello(name, message) {
    console.log("Hello " + name + "," + message);
}
```

函數呼叫方式

```
hello("Leo Shiang", "How are you?");
```

函數返回值

- 函數可以不用有返回值

```
function sum(a, b) {
}

console.log(sum(1, 2));
```

會顯示什麼?

引數（Argument）vs. 參數（Parameter）

parameter

```
function hello(name, message) {
    console.log("Hello " + name + "," + message);
}


hello("Leo Shiang", "How are you?");
```

argument

Argument 的特性

- argument 與函數 parameter 的數量可以不一樣
- argument 與函數 parameter 的資料型態可以不一樣

函數的內部屬性 arguments

- 在函數內部可以透過 arguments 取得所有傳入的引數
- arguments 是一個行為像陣列的物件
- arguments 可以用陣列的方式存取
- arguments 內的元素與 parameter 是相連結的
- 沒有傳值的 parameter 其預設值為 undefined

透過 arguments 取得引數的數量

```
function howManyArgs() {
    console.log(arguments.length);
}

howManyArgs("beauty", "of", "javascript");
howManyArgs();
howManyArgs(new Date());
```

arguments 與 parameter 相連結

```
function add(a ,b) {
    arguments[1] = 10;
    return arguments[0] + b;
}

console.log(add(1, 2));
```

基本觀念

- 函數是一個物件
- 每一個函數都是 Function 類別的實體
- 函數也有屬性和方法
- 函數名稱是一個指向函數物件的指標

函數宣告（Function Declaration）

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

函數表示式（Function Expression）

- Expression 結束要加分號
- Expression 不需要函數名稱

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

函數宣告與表示式作用完全相同

```
1  function funcDeclaration(a, b) {
2      return a + b;
3  }
4
5  var funcExpression = function (a, b) {
6      return a + b;
7  }
8
9  console.log(funcDeclaration);
10 console.log(funcExpression);
```

第 10 行，第 29 欄

過濾輸出資料

```
funcDeclaration()
    arguments: null
    caller: null
    length: 2
    name: "funcDeclaration"
  ▶ prototype: Object { … }
  ▶ <prototype>: function ()
funcExpression()
    arguments: null
    caller: null
    length: 2
    name: "funcExpression"
  ▶ prototype: Object { … }
  ▶ <prototype>: function ()
```

函數名稱是指向函數物件的指標

一個函數可以有不同的名字

```
function sum(a, b) {
    return a + b;
}
console.log(sum(1,2));

var total = sum;
console.log(total(1,2));

sum = null;
console.log(total(1,2));
```

這一行會出錯嗎?

函數宣告與表示式的差異

- 解析器會先讀取函數宣告，讓它在執行其他程式碼之前可用
- 函數表示式要等到解析器要執行呼叫的程式碼時，才會真的被解析執行

函數宣告提升

```
console.log(sum(1, 2));

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

函數表示式

```
1  console.log(sum(1, 2));
2
3  var sum = function(a, b) {
4      return a + b;
5  };
```

函數表示式

- #3 sum 在 expression 中，因此在執行 #3 之前， sum 的內容是 undefined
- 由於 #1 執行時就會發生錯誤，因此 #3 也不會執行到

函數可以當作值來傳遞

```
function callFunction(someFunction, someArgument) {
    return someFunction(someArgument);
}

function add10(num) {
    return num + 10;
}

var result = callFunction(add10, 10);
console.log(result);
```

```
function greeting(name) {
    return "Hello, " + name;
}

var message = callFunction(greeting, "Leo Shiang");
console.log(message);
```

函數內部屬性

- arguments
- argument.callee
- argument.callee.caller

arguments.callee

```
function factorial(num) {
    if (num <= 1) return 1;
    return num * factorial(num - 1);
}

console.log(factorial(3));
```

消除耦合

```
function factorial(num) {
    if (num <= 1) return 1;
    return num * arguments.callee(num - 1);
}


console.log(factorial(3));
```

消除耦合

```
function factorial(num) {
    if (num <= 1) return 1;
    return num * arguments.callee(num - 1);
}

var trueFactorial = factorial;

factorial = function() {
    return 0;
}

console.log(trueFactorial(3));
console.log(factorial(3));
```