# 第十章 函数

## 1、箭头函数不适应场合

箭头函数不能使用 arguments、super 和 new.target，也不能用作构造函数。此外，箭头函数也没有 prototype 属性。

## 2、函数的名称

函数有一个只读的name属性，没名字的函数其name属性值为空字符串。

特别的，console.log((new Function()).name); // anonymous

## 3、arguments

arguments的值会自动同步到对应的命名参数。

function doAdd(num1, num2) {  
 arguments[1] = 10;  
 console.log(arguments[0] + num2);  
 }  
doAdd(10,20) //20  
  
  
//引用数据类型的参数，与对应的arguments值访问同一个地址  
function arrAdd(arr1,arr2){  
 arguments[1]=[1,2,3,4,5]  
 console.log(arr2 === arguments[1]) //true  
 console.log(arr1.concat(arr2))   
}  
  
arrAdd([0],[1]) //[0,1,2,3,4,5]

严格模式下，原始数据类型的参数，arguments的值不会同步；引用数据类型的参数，与对应的arguments值访问不同地址。在函数中尝试重写arguments 对象会导致语法错误。

### 4、“暂时性死区”其实来自let

给多个参数定义默认值实际上跟使用 let 关键字顺序声明变量一样。以下两段代码一样。

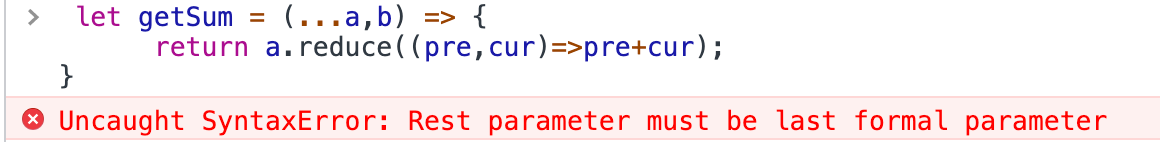
function makeKing(name = 'Henry', numerals = 'VIII') {  
 return `King ${name} ${numerals}`;  
 }  
 console.log(makeKing()); // King Henry VIII

function makeKing() {  
 let name = 'Henry';  
 let numerals = 'VIII';  
 return `King ${name} ${numerals}`;  
 }

function makeKing(name = numerals, numerals = name) {  
return `King ${name} ${numerals}`;  
}  
 console.log(makeKing()); // King Henry Henry  
  
  
//等价于  
function makeKing() {  
let name = numerals;  
let numerals = name;  
return `King ${name} ${numerals}`;  
}  
 console.log(makeKing()); // King Henry Henry

### 5、收集参数（形参带扩展运算符）

收集参数必须放在最后



### 6、caller

ECMAScript 5 也会给函数对象上添加一个属性:caller。这个属性引用的是调用当前函数的函数，如果是在全局作用域中调用的则为 null。

function outer() {  
 inner();  
 }  
   
function inner() {  
 console.log(inner.caller);  
 }   
outer(); //输出outer函数对象，因为inner是在outer中调用的，所以inner.caller 指向 outer()。

### 7、new.target

ES6新增new.target，如果函数被正常调用，则new.target的值是undefined；如果如果是使用new关键字调用的，则new.target将引用被调用的构造函数。

function King() {  
 if (!new.target) {  
 throw 'King must be instantiated using "new"'  
 }  
 console.log('King instantiated using "new"');  
 }  
 new King(); // King instantiated using "new"  
 King(); // Error: King must be instantiated using "new"

### 8、函数的length

函数的length的值为函数定义的命名参数的个数

### 9、prototype属性

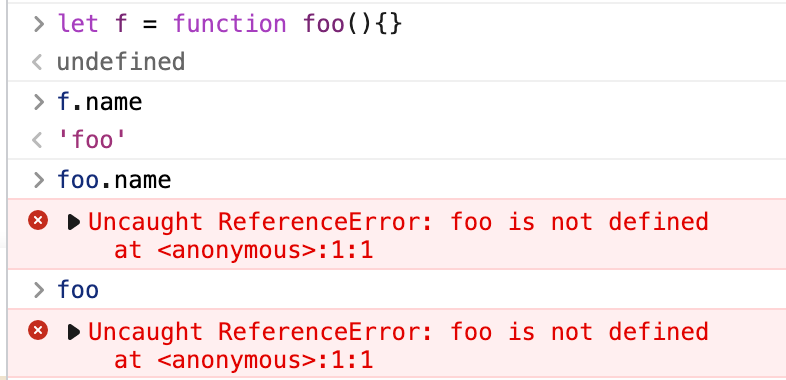
在 ECMAScript 5 中，prototype 属性是不可枚举的，因此使用 for-in 循环不会返回这个属性。

（使用 for in 循环遍历对象的属性时，原型链上的所有属性都将被访问，如果只遍历对象自身的属性，而不遍历继承于原型链上的属性，可以使用hasOwnProperty 方法过滤一下。）

### 10、函数的其他属性

继承的方法 toLocaleString()和 toString()始终返回函数的代码。valueOf()返回函数本身。

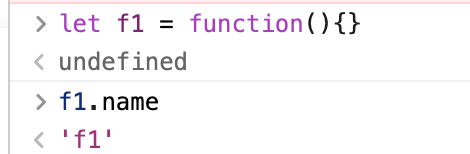
### 11、在函数表达式中给函数命名的情况（命名函数表达式）



foo只是一个标识，在外面作用域无效，但在函数里面有效

const factorial = (function f(num) {  
 if (num <= 1) {  
 return 1;  
 } else {  
 return num \* f(num - 1);  
 }   
 });

普通函数表达式（匿名函数）



### 12、“尾调用”，即外部函数的返回值是一个内部函数的返回值。

### 13、调用函数的过程

①为这个函数调用创建一个执行上下文（局部作用域），并创建一个作用域链，保存在内部的[[scope]]中。

②然后用 arguments 和其他命名参数来初始化这个函数的活动对象。

③将活动对象推入作用域链的前端

④函数内部的代码在访问变量时，就会使用给定的名称从作用域链中查找变量。

⑤函数执行完毕后，局部活动对象会被销毁，局部作用域销毁。（闭包除外）

因为闭包会保留它们包含函数的作用域，所以比其他函数更占用内存。过度使用闭 包可能导致内存过度占用，因此建议仅在十分必要时使用。

使用闭包和私有变量会导致作用域链变长，作用域链越长，则查找变量所需的时间 也越多。

注意：外部函数的this和arguments都是不能直接在内部函数中访问的。如果想访问包含作用域中 的 arguments 或this对象，则同样需要将其引用先保存到闭包能访问的另一个变量中。

### 13.5 作用域链

闭包的作用域链中包含自己的一个变量对象，然后是包含函数的变量对象，直到全局上下文的变量对象。

### 14、立即调用的函数表达式(IIFE，Immediately Invoked Function Expression)

// IIFE  
 (function () {  
 for (var i = 0; i < count; i++) {  
 console.log(i);  
 } })();  
  
console.log(i); // 抛出错误，因为i是定义在自执行函数作用域中，外部无法访问。

最大的作用是，创建了一个独立作用域，避免了命名冲突的问题。

### 15、引用数据类型作为参数

let object1 = {name:'laomao'}  
console.log(object1) //{name: 'laomao'}  
  
function changeObj(obj){  
 obj.name = 'mao'  
 return obj === object1  
}  
let result = changeObj(object1);  
console.log(result); // true  
console.log(object1); // {name: 'mao'}