arguments.callee是指向参数arguments对象的函数，在这里就是obj咯。

函数也是一个对象

callee 在函数的内部，表示当前函数的引用

callee 是 arguments 对象的一个属性。它可以用于引用该函数的函数体内当前正在执行的函数。这在函数的名称是未知时很有用，例如在没有名称的函数表达式 (也称为“匿名函数”)内。

caller表示函数的调用者。

The **function declaration** defines a function with the specified parameters. You can also define functions using the **Function constructor** and a **function expression**. A function created with a function declaration is a Function object and has all the properties, methods and behavior of Function objects.

|  |  |
| --- | --- |
| **function declaration** | **function expression** |
| function name ( [param, [, param, [..., param]]]) { [statements] } | var myFunction = function [name] ([param1 [,param2[, ..., paramN]]]) { statements}; |
| Function declarations in JS are hoisting the function definition. You can use the function before you declared it. Note that function expression are not hoisted. 函数表达式不被提升 | The function name can be **omit**ted, in which case the function is anonymous. The name is only local to the function body.  for conditional creation use function expressions. |

如果不声明函数名称，它肯定是表达式，声明的话要通过上下文来区分。如果是作为赋值表达式的一部分的话（或者当函数被grouping operator包装的时候），它就是函数表达式，如果被包含在一个函数体内，或者位于程序中，它就是函数声明。

构造函数 var fn = new Function(“前面的都是形参” “最后一个是函数体”);

全局变量：在最外层声明的变量，在哪里都可以使用。在函数内部，但是前面没有加var声明的变量也是全局变量。循环内部定义的变量也是全局变量。

局部变量：在函数内部声明的变量就是局部变量，只能在当前函数内部使用。

一个函数的形参在函数预解析的过程中，会自动声明提升成一个局部变量，提升完毕后，会直接给这个局部变量赋实参的值，如果没有传实参，那么形参的值是undefined。在变量声明提升的过程中，如果当前作用域已经存在这个变量，那么什么都不做，相当于重复声明。但是如果是同名函数的话，在提升过程中会把上面的函数替换掉。

预解析会提升var和function关键字，同时提升function的函数体。

递归：函数自身调用自身的编程技巧。一定要注意结束条件，不然造成死循环。一般用来做标记的变量很多都是全局变量。

回调函数就是一个通过[函数指针](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E6%8C%87%E9%92%88/2674905" \t "https://baike.baidu.com/item/_blank)调用的函数。如果你把函数的[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88/2878304" \t "https://baike.baidu.com/item/_blank)（地址）作为[参数传递](https://baike.baidu.com/item/%E5%8F%82%E6%95%B0%E4%BC%A0%E9%80%92/9019335" \t "https://baike.baidu.com/item/_blank)给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

键值对：key value pair

该数据结构的特点是以空间换时间，通过键查询值能够提高效率。

for (var i = 0; i<= 5; i++) {

obj[“n”+i] = i;

}

中括号语法是通过传入字符串来查找属性名，而这个字符串可以通过编程来拼接，这样就很灵活了。

遍历对象：

for (var key in object) {

console.log(object[key]);

}

fn.arguments 实参列表，只能在函数内部使用，通过这个属性可以获取每个实参的值；fn.length 形参个数；fn.name 函数名。

### undefined表示“缺少值”，就是此处应该有一个值，但是还没有定义。典型用法是：

1. 声明变量但是未赋值；
2. 声明了length，但是未给数组的项赋值；
3. 函数如果没有返回值，默认返回undefined；所以console.log(fn());打印结果是undefined，console.log(fn); 打印的是函数体。
4. 调用函数时，应该提供的参数没有提供，也就是形式参数没有接收到实际参数的值，那么它的值是undefined；如果传入的实参个数多于形参，那么多余的实参不起任何作用。
5. 对象属性如果没有赋值，系统自动赋给它undefined；

重载指的是函数名相同，但是函数参数和返回值不同。当调用这个函数时，会自动根据传入的参数个数去找到相匹配的函数去执行。

在传参数的时候，是把实参的值拷贝了一份赋值给形参，形式参数的改变，不会影响实际参数。

## RETURN

The return statement ends function execution and specifies a value to be returned to the function caller. if omitted, undefined is returned instead.

**不管return的有没有值，它后面的代码都不会执行，遇到return函数就结束了。**

JS没有重载，只要函数名相同，就认为是同一个函数，如果写了多个，后面的就会把前面的覆盖掉。

JS的特点：弱类型、单线程、面向对象。

F8：逐断点执行；F10：逐步执行；F11：进入函数内部。shift + F11跳出函数。

解析器有函数声明提升的功能。变量也可以提升，但只提升声明，不提升赋值。函数的提升也是只提升声明，只不过函数的声明就是那个函数体。

变量后面赋值的函数是没有名字的，这种函数叫匿名函数。

匿名函数的使用：

1. 声明对象方法
2. 立即执行函数：有的函数定义完成后，将来不需要再调用，只是现在执行一下即可

(function ( ) {

console.log(“a”); } ) ( );

### 函数的四种调用模式

1. 函数模式：单独独立调用的就是函数。this表示全局对象

function fn(){

console.log(this);}

fn(); //打印结果：window

1. 方法模式：方法本身就是函数，但是方法不是单独独立的调用，而是通过一个对象引导调用。this表示引导方法的宿主对象

obj.fn = function(){

console.log(this)};

obj.fn(); //打印结果：Object {}

特殊情况：

var arr = [function fn() {console.log(1);}];

arr[0](); //这种也属于方法调用，因为是通过arr找到的fn

1. 构造器模式：用new引导。new是一个运算符，专门用来申请创建对象，创建出来的对象传递给构造函数的this(this指的就是new出来的实例对象)，利用构造函数对其进行初始化。

* 返回值

1. 如果不写 return 语句, 那么构造函数默认返回 this (即省略return this)
2. 如果在构造函数中写上 return 语句, 并紧跟基本类型( return num, return 1223 ). 则忽略返回数据
3. 如果在构造函数中写上 return 语句, 并且紧跟引用类型, 那么构造函数返回该引用类型数据

* 如果调用构造函数的时候, 构造函数没有参数, 圆括号是可以省略的

var p = new Person;

* 如果希望创建对象并直接调用其方法：( new Person () ).sayHello()

1. 可以省略调整结构性的圆括号：new Person().sayHello()
2. 如果想要省略构造函数的圆括号, 就必须添加结构性的圆括号：(new Person).sayHello()
3. 上下文模式（这两种形式功能完全一样, 唯一不同的是参数的形式）

* call 形式：函数名.call( ... )
* apply 形式：函数名.apply( ... )，foo.apply( o )的好处在于不会污染o对象。

call 和 apply如果第一个参数是null / underfined，this指向window；

call 和 apply如果第一个参数是基本数据类型，会自动将基本数据类型转换成对应的包装类型。

包装类型有三种：

function fn(){}

var str = “abc”;

var str = new String(“abc”);

str.f = fn;

str.f();

var num = new Number(123);

var boolean = new Boolean(true);

为什么要有call和apply？

var stu = {

name: "Tom",

introduce: function() {

console.log(this.name);

}

}

var btn = document.querySelector("button");

btn.addEventListener("click", stu.introduce);

打印结果是空，为什么不是Tom呢？

因为当点击按钮的时候，function是btn调用的，所以function中的this自然指向btn，而btn是没有name属性的，所以打印空。

addEventListener行等价于：

btn.onclick = stu.introduce;

怎样才能打印出Tom呢？

有两种方法：

1. 把addEventListener注掉，加上以下代码：

btn.onclick = function() {

stu.introduce();

}

function是由btn调用的，而introduce是stu调用的，所以this自然指的是stu。

1. 直接改变stu.introduce的指向：

btn.onclick = stu.introduce.bind(stu);

详解：

var foo = fn.bind(person);

bind的作用是复制fn，并把fn中的this永远指向person，然后返回该复制品。foo中的this和person永远绑定在一起。而fn中的this仍然是谁调它就指向谁，并没有和谁绑定。

### 面试题解析

1. 方法模式

var length = 10; //定义一个全局变量length，这个变量亦即window的一个属性

function fn() {

console.log( this.length );

}

var obj = {

length: 5,

method: function ( fn ) {

fn();

arguments[ 0 ]();

}

};

obj.method( fn, 1 );

解题思路：

1. 预解析：先提升变量和函数体;
2. 执行代码：分别给变量赋值；
3. 执行最后一行：通过obj用点语法调用method方法，method先调用fn，此时是函数调用，所以this指的是window，而window正好有length属性，属性值是10；
4. arguments是参数对象，里面放的是传入的实参，arguments[ 0 ]取到的是第一个实参fn，然后调用fn，此fn不是单独独立调用的，而是通过arguments对象找到的，所以属于方法调用，this指的是调用它的宿主对象arguments，而arguments.length是2（因为只有两个参数）。
5. 构造器模式

function Foo () { //声明一个函数

getName = function(){ alert(1); };

return this;

}

function getName () { //声明一个函数

alert(5);

}

Foo.getName = function() { alert(2); }; //foo也是个对象，为它添加getName方法

Foo.prototype.getName = function(){ alert(3); }; //为foo的原型属性添加方法

getName = function(){ alert(4); }; //覆盖alert(5)；

Foo.getName(); // = 2 方法调用

getName(); // = 4 函数调用

Foo().getName(); // = 1 调用foo，做了两件事：重新给getName赋值alert(1)；返回window，然后window调用getName

getName(); // = 1

new Foo.getName(); // = 2 Foo.getName()会alert(2)，如果没有参数，构造函数后的圆括号可以省略，但是要写成(new Foo).getName();

new Foo().getName(); // = 3 new Foo()会创建出一个实例，构造函数中并没有为this绑定任何属性或方法，所以new出来的对象什么也接收不到，只好到原型属性中去找。

new new Foo().getName(); // = 3

### 借用构造方法实现继承

function Person ( name, age, gender ) {

this.name = name;

this.age = age;

this.gender = gender;

}

function Student ( name, age, gender, course ) {

Person.call( this, name, age, gender );

this.course = course;

}

var p = new Student ( 'jim', 19, 'male', '前端' );