0、this：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/this>

1、实现继承的较好的方式：圣杯模式。即加入一个中间层。

例：function F(){}

F.prototypr = Father.prototype

Son.prototype = new F();

这样做的好处是，改动Son.prototype里的东西，不会影响到Father.prototype。

最终封装起来即可：

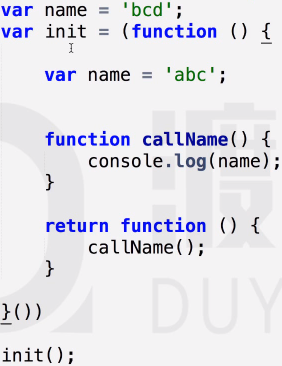
function inherit(Target,Origin) {  
 function F() {}  
 F.prototype = Origin.prototype;  
 Target.prototype = new F();

Target.prototype.constructor = Target;  
 Target.prototype.uber = Origin.prototype;  
}

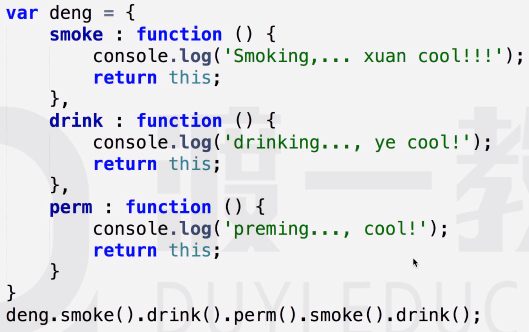
作为对比，建议使用下面的方法：

var ***inherit*** = (function () {  
 var F = function () {};  
 return function (Target, Origin) {  
 F.prototype = Origin.prototype;  
 Target.prototype = new F();  
 Target.prototype.constructor = Target;  
 Target.prototype.uber = Origin.prototype;  
 }  
}());

2、善用闭包的方式实现私有化，避免相同变量名冲突：



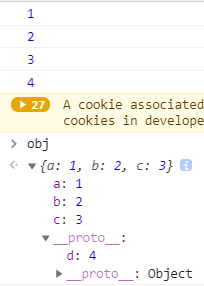
3、return this;实现一个对象里的方法连续调用：

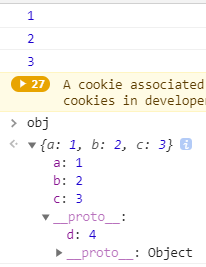


4、关于for-in循环的补充：

会返回所有的属性名一次（顺序不定）（String类型），并且\_\_proto\_\_也会被返回（返回\_\_proto\_\_的前提是该属性值修改过，如果上探到原型链的最顶端，就不会返回）

可以用 [Obejct].hasOwnProperty([String]) 来判断该属性是不是对象本身的属性。

var ***obj*** = {  
 a: 1,  
 b: 2,  
 c: 3,  
 \_\_proto\_\_ : {  
 d:4  
 }  
};  
for (var ***key*** in ***obj***) {  
 ***console***.log(***obj***[***key***]);  
}

var ***obj*** = {  
 a: 1,  
 b: 2,  
 c: 3,  
 \_\_proto\_\_ : {  
 d:4  
 }  
};  
for (var ***key*** in ***obj***) {  
 if(***obj***.hasOwnProperty(***key***)){  
 ***console***.log(***obj***[***key***]);  
 }  
}

补充：一般就不用in操作符了，父亲有的属性in也会返回true，即in判断的只是该对象里有没有这个属性（包括继承自父亲的），有就返回true

5、关于instanceof的用法：

A instanceof B

看A对象的原型链上有没有B的原型

6、typeof 共返回6种数据格式：

1、object

2、undefined

3、string

4、number

5、boolean

6、function

特别注意Array和null返回的都是object

6、关于arguments.callee：

返回函数自身的引用，当函数没有函数名的时候可以用这个方法来获得引用，例如在立即执行函数和函数赋值给一个变量时。

arguments上只有callee和length属性

7、深度克隆步骤：遍历（for(var ***prop*** in obj)）

1、判断是不是原始值

2、判断是数组还是对象

3、建立相应的数组或对象

示例：

function deepClone(origin,target) {  
 var target = target || {},  
 toStr = ***Object***.prototype.toString,  
 arrStr = "[object Array]";  
 for (var prop in origin){  
 if (origin.hasOwnProperty(prop)){  
 if (origin[prop]!== null &&  
 typeof origin[prop] == 'object'){  
 if (toStr.call(origin[prop]) == arrStr){  
 target[prop] = [];  
 }else{  
 target[prop] = {};  
 }  
 deepClone(origin[prop],target[prop]);  
 }else{  
 target[prop] = origin[prop];  
 }  
 }  
 }  
 return target;  
}

8、三目运算符：

对数字比大小的补充：在进行比较时，“10” > 9，这里的“10” 会转化成10再比较，如果是“20” > “10” 则会ASCII码比较

9、数组函数：

sort([function (a,b){ 自定义函数规则，return a-b为升序，return b-a为降序 }])

<https://www.w3school.com.cn/js/jsref_sort.asp>

**说明**

如果调用该方法时没有使用参数，将按字母顺序对数组中的元素进行排序，说得更精确点，是按照字符编码的顺序进行排序。要实现这一点，首先应把数组的元素都转换成字符串（如有必要），以便进行比较。

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/sort>

数组的所有实例：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array#%E6%95%B0%E7%BB%84%E5%AE%9E%E4%BE%8B>

10、类数组：

示例：

var ***obj*** = {  
 "0" : 'a',  
 "1" : 'b',  
 "2" : 'c',  
 "length" : 3,  
 "push" : ***Array***.prototype.push,

"splice" : ***Array***.prototype.splice  
}

要求：属性要为索引（数字）属性，必须要有length属性，最好加上push。

类数组的push方法内部原理：（牢记）

***Array***.prototype.push = function (target) {  
 this[this.length] = target;  
 this.length++;  
}

11、数组去重：

***Array***.prototype.unique = function () {  
 var temp = {},  
 arr = [],  
 len = this.length;  
 for(var i = 0;i<len;i++){  
 if(!temp[this[i]]){  
 temp[this[i]] = "exist";  
 arr.push(this[i]);  
 }  
 }  
 return arr;  
}

12、经过var的属性/值是不可以delete的，比如var num =123;即使在window.num可以访问到，但是不能delete，如果是直接window.num = 123;，是可以delete的。

13、this的作用：

1、预编译this --> window

2、谁调用this指向谁

3、call apply 可以改变this指向

（call：<https://www.w3school.com.cn/js/js_function_call.asp>

apply：<https://www.w3school.com.cn/js/js_function_apply.asp>）

4、全局this--> window

14、继承的6种方法：

<https://www.cnblogs.com/ranyonsue/p/11201730.html>

15、Error.name的六种值对应的信息：

1. EvalError：eval()的使用与定义不一致

2. RangeError：数值越界

3. ReferenceError：非法或不能识别的引用数值

4. SyntaxError：发生语法解析错误

5. TypeError：操作数类型错误

6. URIError：URI处理函数使用不当

16、关于严格模式：

两种用法：

全局严格模式

局部函数内严格模式（推荐）

就是一行字符串，不会对不兼容严格模式的浏览器产生影响。

不支持with,arguments.callee,func.caller,变量赋值前必须声明，局部this必须被赋值(Person.call(null/undefined) 赋值什么就是什么),拒绝重复属性和参数

17、封装获取浏览器窗口大小：

function getViewportOffset() {  
 if (***window***.innerWidth){  
 return{  
 w: ***window***.innerWidth,  
 h:***window***.innerHeight  
 }  
 }else {  
 if (***document***.compatMode === "BackCompat"){  
 return {  
 w:***document***.body.clientWidth,  
 h:***document***.body.clientHeight  
 }  
 }else {  
 return {  
 w:***document***.documentElement.clientWidth,  
 h:***document***.documentElement.clientHeight  
 }  
 }  
 }  
}

18、封装滚动条方法：

function getScrollOffset() {  
 if(***window***.pageXOffset){  
 return{  
 x:***window***.pageXOffset,  
 y:***window***.pageYOffset  
 }  
 }else{  
 return{  
 x:***document***.body.scrollLeft  
 +***document***.documentElement.scrollLeft,  
 y:***document***.body.scrollTop  
 +***document***.documentElement.scrollTop  
 }  
 }  
}

19、关于获取样式的方法：

1、***window***.getComputedStyle(ele,null).prop;

2、ele.style.prop;

方法一获取的是显示样式的值，怎么显示就是什么，第二个参数一般为null，当要选出伪元素的样式时，就代入一个伪元素（如：”after”，注意是字符串的形式）；

方法二获取的只是行间样式，html怎么写的就获取的什么样的。

20、封装获取style方法：

function getStyle(elem,prop){  
 if(***window***.getComputedStyle){  
 return ***window***.getComputedStyle(elem,null.[prop]);  
 }else {  
 return elem.currentStyle[prop];  
 }  
}

21、封装事件添加函数：

function addEvent(elem,type,handle){  
 if(elem.addEventListener){  
 elem.addEventListener(type,handle,false);  
 }else if (elem.attachEvent){  
 elem.attachEvent('on'+type,function () {  
 handle.call(elem);  
 })  
 }else {  
 elem['on'+type] = handle;  
 }  
}

22、事件执行顺序：

执行>捕获>冒泡

在执行的时，又有以下顺序规则：

代码中，谁先绑定谁先执行。

例如：

wrapper.addEventListener('onclick',function () {  
 ***console***.log('wrapperBubble');  
},false);  
content.addEventListener('onclick',function () {  
 ***console***.log('contentBubble');  
},false);  
box.addEventListener('onclick',function () {  
 ***console***.log('boxBubble');  
},false);  
wrapper.addEventListener('onclick',function () {  
 ***console***.log('wrapper');  
},true);  
content.addEventListener('onclick',function () {  
 ***console***.log('content');  
},true);  
box.addEventListener('onclick',function () {  
 ***console***.log('box');  
},true);

点击最里层的结果：

wrapper

content

boxBubble

box

contentBubble

wrapperBubble

23、封装取消冒泡：

function stopBubble(event) {  
 if (event.stopPropagation()){  
 event.stopPropagation();  
 }else{  
 event.cancelBubble = true;  
 }  
}

24、封装取消右键默认事件：

function cancelHandler(event) {  
 if (event.preventDefault){  
 event.preventDefault();  
 }else {  
 event.returnValue = false;  
 }  
}

25、reg.lastIndex只有在正则表达式带有g时，游标(lastIndex)才会移动。

正则表达式默认为“贪婪匹配”原则，若要打破，则在表达式最后加上?

26、ES6

27、类注意点：

1）在ES6中类没有变量提升，所以必须先定义类，才能通过类实例化对象；

2）类里面的共有的属性和方法一定要加this使用；

3）类里面的this指向问题；

4）constructor里面的this指向实例对象，方法里面的this指向这个方法的调用者。

28、new的时候发生的事情：

1）在内存中创建一个新的空对象；

2）让this指向这个新的对象；

3）执行构造函数里的代码，给新对象添加属性和方法；

4）返回这个新对象，不需要手动return。

29、静态成员、实例成员：

1）静态成员只能通过构造函数来访问

2）实例成员就是构造函数内部通过this添加的成员，实例成员只能通过实例化的对象来访问。

例：

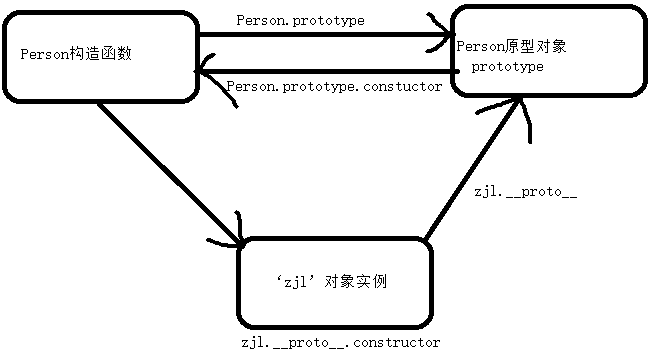
function Person(name,age) {  
 this.name = name;  
 this.age = age;  
}  
Person.sex = 'male'; //static member  
var ***me*** = new Person('zjl',18);  
***console***.log(***me***); //Person{name:...,age:...}  
***console***.log(***me***.sex); //undefined  
***console***.log(Person.sex) //male

***console***.log(Person.prototype===***me***.\_\_proto\_\_) //true

（注：每个对象都有一个\_\_proto\_\_属性，指向构造函数的prototype原型对象，可以在prototype上给一些方法和属性，这样每个后代都有这些方法和属性，但不可以使用这些方法和属性，只是使用的时候会通过原型链一层层找到）

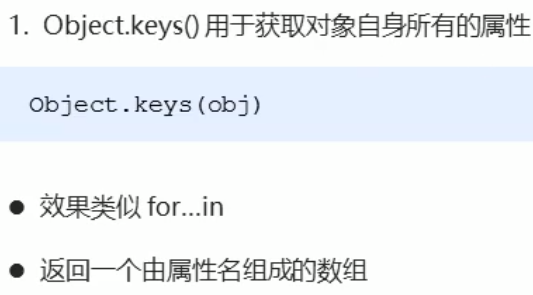
30、关于constructor的用法：

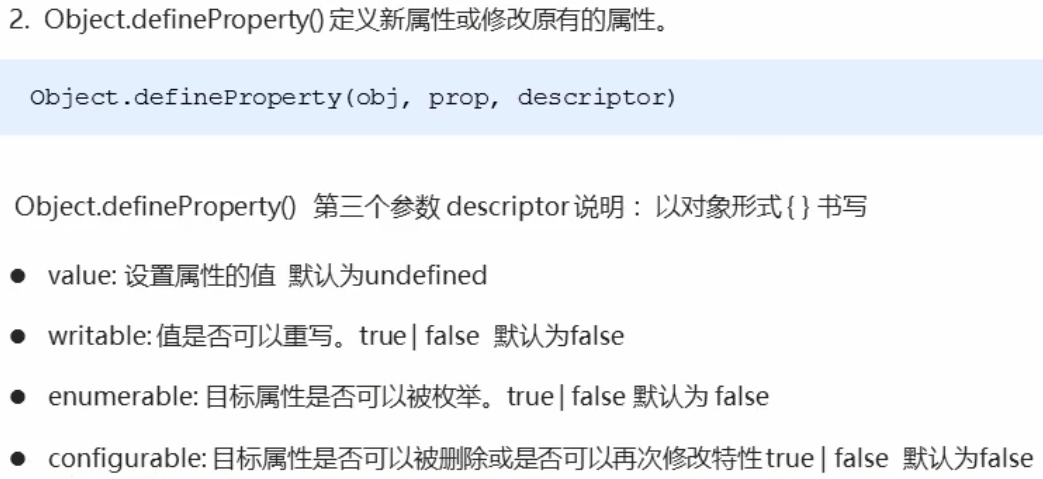
Person.prototype = {  
 //如果通过对象赋值修改了原来的原型对象，  
 // 则必须手动添加一个constructor指回原来的构造函数  
 constructor: Person,  
 walk: function () {  
 ***console***.log('walk');  
 },  
 run: function () {  
 ***console***.log('run');  
 }  
}



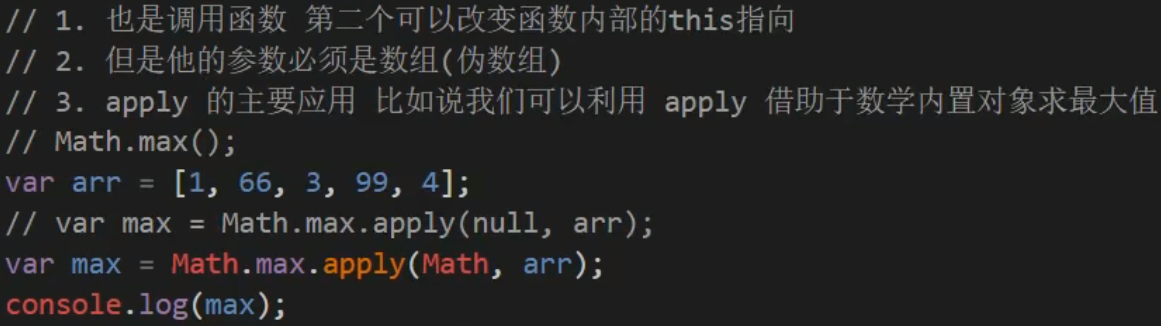
31、Array函数中，foreach/filter函数无终止，some可以。即在调用some时，可以设置一个终止条件（if），然后在if体里写return true即可终止循环。

32、





33、apply的应用：



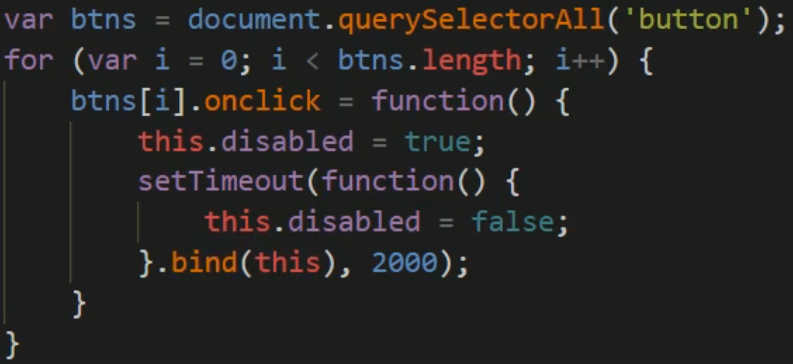
34、bind()：

1）不会调用原来的函数，可以改变原来函数内部的this指向；

2）返回的是原函数改变this之后产生的新函数；

3）如果有的函数我们不需要立即调用，但又想改变这个函数内部的this指向此时用bind

4）应用：给一（组）按钮添加开启可用倒计时：



35、严格模式：’use strict’;

1）变量名必须先声明再使用；

2）不能随意删除已经声明好的变量；

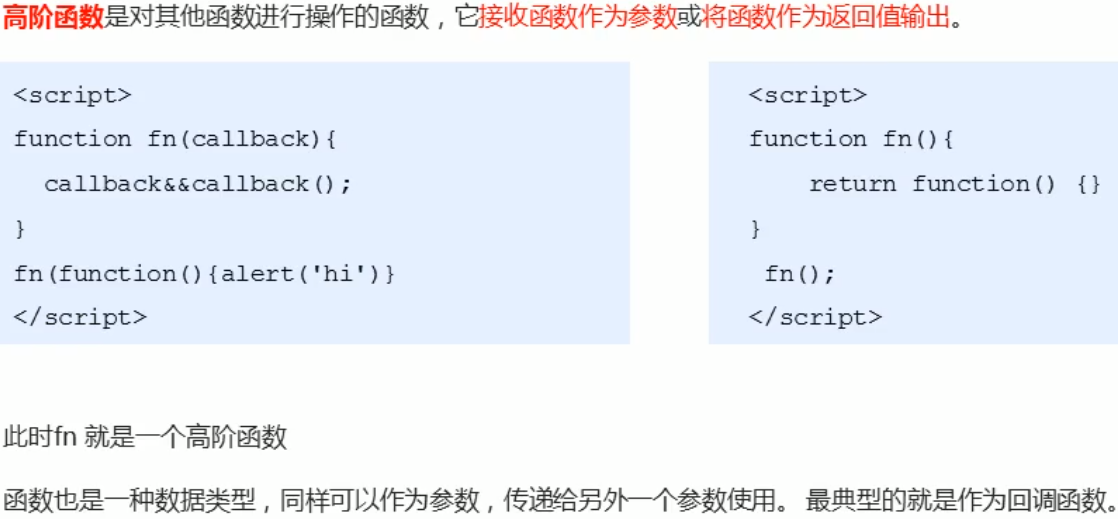
3）全局作用域中函数中的this是undefined，原本是会指向window，因此如果直接调用构造函数和报错，因为严格模式下this是undefined；

4）定时器setTimeout里的this还是指向window，事件、对象还是指向调用者；

5）函数形参不可重复

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode>

36、高阶函数：



37、闭包：

1）指一个函数有权访问另一个函数作用域中变量的函数；

2）一个作用域可以访问另一个函数的局部变量；

3）主要作用是延伸了变量的作用范围

38、递归在深浅拷贝中的应用：

浅拷贝：用Object.assign(target,origin);

深拷贝：

重复调用，找准出口。

39、let关键字：

1）无变量提升；

2）暂时性死区，块级作用域内（即一对花括号内）的变量，即使与全局内的变量同名，只要是没先声明就去使用，就报错（块级内的同名变量不会出去找，只在块级作用域里找）。

40、const：

MDN描述：

此声明创建一个常量，其作用域可以是全局或本地声明的块。 与[var](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/var)变量不同，全局常量不会变为窗口对象的属性。需要一个常数的初始化器；也就是说，您必须在声明的同一语句中指定它的值（这是有道理的，因为以后不能更改）。

**const声明**创建一个值的只读引用。但这并不意味着它所持有的值是不可变的，只是变量标识符不能重新分配。例如，在引用内容是对象的情况下，这意味着可以改变对象的内容（例如，其参数）。

关于“[暂存死区](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/let#Temporal_dead_zone_and_errors_with_let)”的所有讨论都适用于[let](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/let)和const。

一个常量不能和它所在作用域内的其他变量或函数拥有相同的名称。

1）块级作用域

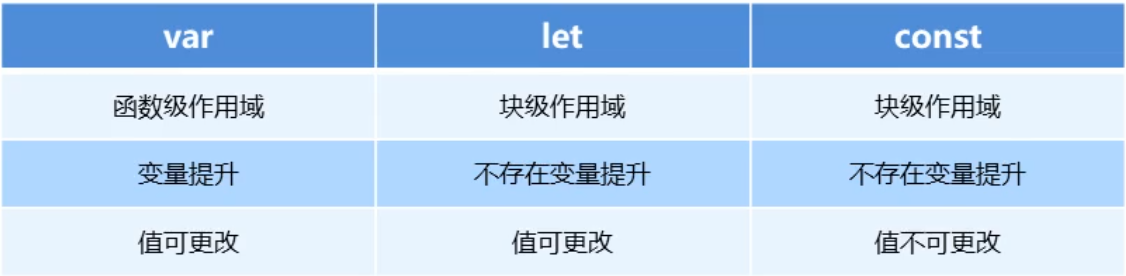
2）

41、let、const、var的区别：

1）使用var声明的变量，其作用域为该语句所在的函数内，且存在变量提升现象。

2）使用let声明的变量，其作用域为该语句所在的代码块内，不存在变量提升。

3）使用const声明的是常量,在后面出现的代码中不能再修改该常量的值。



42、

1）数组解构：按照一一对应的关系从数组中提取值然后赋值给变量。

let ary = [1,2,3];

let [a,b,c,d] = ary / [1,2,3];

console.log(a); // 1

console.log(b); // 2

console.log(c); // 3

console.log(d); // undefined

2）对象解构：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment#%E8%A7%A3%E6%9E%84%E5%AF%B9%E8%B1%A1>

（1）基本赋值：

var o = {p: 42, q: true};

var {p, q} = o;

console.log(p); // 42

console.log(q); // true

（2）给新的变量名赋值：

可以从一个对象中提取变量并赋值给和对象属性名不同的新的变量名。

var o = {p: 42, q: true};

var {p: foo, q: bar} = o;

console.log(foo); // 42

console.log(bar); // true

（3）默认值

变量可以先赋予默认值。当要提取的对象没有对应的属性，变量就被赋予默认值。

var {a = 10, b = 5} = {a: 3};

console.log(a); // 3

console.log(b); // 5

（4）给新的变量命名并提供默认值

一个属性可以同时 1）从一个对象解构，并分配给一个不同名称的变量 2）分配一个默认值，以防未解构的值是 undefined。

var {a:aa = 10, b:bb = 5} = {a: 3};

console.log(aa); // 3

console.log(bb); // 5

43、箭头函数：不能用于给原型（prototype）赋值

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

如果函数体中只有一句代码，并且代码执行结果就是函数的返回值，函数体大括号可以省略：

const sum=(n1,n2)=>n1+n2;

const result = sum(10,20) ;

console. log(result); // 30

如果形参只有一个形参外侧的小括号也是可以省略：

const fn = v => {

alert(v);

};

注意：

1）箭头函数不绑定this关键字，箭头函数中的this，指向函数定义位置的上下文this，而不是谁调用指向谁

例如：

const ***obj*** = {  
 age: "apple",  
 say: () => {  
 alert(this.age);  
 }  
};  
***obj***.say(); // undefined

2）



44、剩余参数：

1）函数内无arguments，要用剩余参数的方法取得多余的参数：

(...args) => {}

function (...args) {}

2）配合解构：

let students = ['wangwu' ，‘zhangsan'，'lisi'] ;

let [s1，…s2] = students ;

console.log(s1) ; // ' wangwu'

console.1og(s2) ; // ['zhangsan', 'lisi']

45、扩展运算符：

1）

let ***a*** = ['a','b','c'];  
let [***f***,...***c***] = ***a***;  
***console***.log(...***c***); // b c  
***console***.log('b','c'); // b c

如果在log里逗号分隔多个参数，输出是以空格形式分隔，扩展运算符也相当于把数组的每一项用逗号分开，用log输出，也是以空格形式分隔。

2）用于合并数组：

let ***a*** = ['a','b','c'];  
let ***b*** = ['d','e','f'];  
let ***sum*** = [...***a***,...***b***];   
***console***.log(***sum***); //["a", "b", "c", "d", "e", "f"];

***a***.push(...***b***); //即为('d','e','f')  
***console***.log(***sum***) //["a", "b", "c", "d", "e", "f"];

3）把伪数组或可遍历对象转换为真正的数组：

实例：

let ***divs*** = ***document***.getElementsByTagName('div');  
***console***.log(***divs***); //HTMLCollection(5) [div, div, div, div, div]  
let ***myd*** = [...***divs***];  
***console***.log(***myd*** instanceof ***Array***); // true  
***console***.log(***myd***); //[div, div, div, div, div]

46、Array的扩展方法：

1）构造函数方法: Array.from()

（1）将类数组或可遍历对象转换为真正的数组

let arrayLike = {

'0' :‘a’,

'1': 'b',

'2': 'c'

length: 3

};

let arr2 = Array. from(arrayLike); // ['a', 'b', 'c']

（2）可接受第二个参数，作用类似于数组的map方法，用来对每个元素进行处理，将处理后的值放入返回的数组。

let arrayLike = {

"0": 1,

"1": 2,

"length": 2 }

let newAry = Array. from(aryLike, item = => item \*2) // [ 2 , 4 ]

2）find(),findIndex() ：

前者找出第一个符合传入函数要求的值，后者返回索引。

3）includes()：

查看数组中是否有传入参数的值，有则返回true，否则返回false。

47、String扩展方法：

（1）模板字符串：

1）模板字符串：定义模板字符串``

2）使用` ${ } `

let ***str*** = `rich`;  
let f = () =>　"rich";  
***console***.log(`I'm not ${***str***}`); //I'm not rich  
***console***.log(`I'm not ${f()}`);//I'm not rich

（2）方法：

1）startsWith():表示参数字符串是否在原字符串的头部,返回布尔值

2）endsWith():表示参数字符串是否在原字符串的尾部，返回布尔值

3）repeat()：表示将原字符串重复n次，返回一个新字符串。

'x' . repeat (3)// "xxx"

'hello' . repeat (2)// "hellohello"

48、Set数据结构：没有重复值

1）主要结构：

const ***s1*** = new ***Set***();  
***console***.log(***s1***); // Set(0) {}  
const ***s2*** = new ***Set***(['a','b']);  
***console***.log(***s2***); //Set(2) {"a", "b"}

//利用无重复值特性为数组去冗余  
const ***s3*** = new ***Set***(['a','b','b' ]);  
***console***.log(***s3***); //Set(2) {"a", "b"}  
const ***s4*** = [...***s3***];  
***console***.log(***s4***); //["a", "b"]

2）常用方法：

add(value):添加某个值，返回Set结构本身；

delete(value):删除某个值，返迥一个布尔值，示删除是否成功；

has(value):返回一个布尔值，表示该值是否为Set的成员；

clear():清除所有成员，没有返回值。

遍历：

forEach方法,用于对每个成员执行某种操作,没有返回值。

s.forEach(value => console. log (value))

49、Array.prototype.map()：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/map>

50、**filter()** 方法创建一个新数组, 其包含通过所提供函数实现的测试的所有元素。

filter函数等的使用技巧：封装

假定有两个数组(A, B),根据A中id值,过滤掉B数组不符合的数据：

function example(A,B){

    return B.filter(function(A){

        return B.value === A.value;

    })

}

51、reduce：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce>

52、函数参数默认值：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Default_parameters>

53、数组、对象解构：

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment>