目录

[第一节 面向对象编程介绍 1](#_Toc12118)

[1.两大编程思想 1](#_Toc981)

[1.1 面向过程编程POP(Process-oriented programming) 1](#_Toc19778)

[1.2 面向对象编程OOP(Object Oriented Programming) 1](#_Toc18797)

[1.3 面向过程和面向对象的对比 2](#_Toc1026)

[第二节 ES6中的类和对象 2](#_Toc23986)

[1.面向对象 2](#_Toc12359)

[2.对象 2](#_Toc26057)

[3.类 class 3](#_Toc6387)

[1.1创建类 3](#_Toc12822)

[1.2类constructor构造函数 3](#_Toc1465)

[1.3 类添加方法 4](#_Toc18649)

[第三节 类的继承 6](#_Toc13262)

[1.继承 6](#_Toc19104)

[2.super关键字 6](#_Toc24841)

[3.三个注意点 7](#_Toc10077)

[第四节 构造函数和原型 7](#_Toc10535)

[1.概述 7](#_Toc4387)

[2.构造函数 8](#_Toc3600)

[1.1 构造函数的问题 9](#_Toc16855)

[1.2 构造函数原型prototype 9](#_Toc2888)

[3.对象原型\_proto\_ 10](#_Toc4951)

[4.constructor构造函数 10](#_Toc22007)

[5.构造函数、实例、原型对象三者之间的关系 11](#_Toc9698)

[6.原型链 11](#_Toc13870)

[7.JavaScript的成员查找机制（规则） 11](#_Toc28366)

[8.原型对象this指向 12](#_Toc2900)

[9.扩展内置对象 12](#_Toc6597)

[第五节 继承 13](#_Toc23765)

[1.call( ) 13](#_Toc14633)

[2.借用构造函数继承父类型属性 13](#_Toc8597)

[3.借用原型对象继承父类型方法 14](#_Toc26184)

[4.类的本质 14](#_Toc31670)

[5.ES5中的新增方法 15](#_Toc14475)

[1.1数组方法 15](#_Toc21410)

[1.2 forEach和some的区别 18](#_Toc29383)

[1.3字符串方法 18](#_Toc21645)

[1.4 对象方法 18](#_Toc78)

[第六节 函数进阶 20](#_Toc7614)

[1.函数的定义和调用 20](#_Toc26500)

[1.1函数的定义方式 20](#_Toc22367)

[1.2 函数的调用方式 21](#_Toc17909)

[2.this 22](#_Toc3758)

[1.1 函数内this的指向 22](#_Toc19009)

[1.2 改变函数内部this的指向 22](#_Toc7356)

[3.严格模式 25](#_Toc610)

[1.1 什么是严格模式 25](#_Toc13342)

[1.2 开启严格模式 25](#_Toc18703)

[1.3 严格模式中的变化 27](#_Toc21826)

[4.高阶函数 28](#_Toc13130)

[5.闭包 28](#_Toc9792)

[1.1变量作用域 28](#_Toc10081)

[1.2什么是闭包 28](#_Toc21090)

[6.递归 29](#_Toc20709)

[1.1浅拷贝和深拷贝 29](#_Toc30390)

[第七节 正则表达式 31](#_Toc356)

[1.什么是正则表达式 31](#_Toc3175)

[2.正则表达式的特点 31](#_Toc19434)

[3.正则表达式在JavaScript中的使用 31](#_Toc1714)

[1.1创建正则表达式 32](#_Toc22931)

[1.2 测试正则表达式test 32](#_Toc18867)

[4.正则表达式中的特殊字符 32](#_Toc10170)

[1.1正则表达式的组成 32](#_Toc2140)

[1.2 边界符 33](#_Toc9350)

[1.3 字符类 33](#_Toc19276)

[1.4 量词符 34](#_Toc1243)

[1.5 括号总结 35](#_Toc12148)

[1.6 预定义类 36](#_Toc18186)

[5.正则表达式中的替换 36](#_Toc8856)

[1.1 replace替换 36](#_Toc12201)

[1.2 正则表达式参数 36](#_Toc21816)

[第八节 ES6简介 37](#_Toc14967)

[1.let关键字 37](#_Toc5221)

[2.经典面试题（一） 38](#_Toc1864)

[3.const关键字 39](#_Toc5000)

[4.let、var、const的区别 40](#_Toc19262)

[5.解构赋值 40](#_Toc11564)

[6.箭头函数 41](#_Toc20701)

[7.剩余参数 42](#_Toc6984)

[8.Array的扩展方法 43](#_Toc9666)

[1.1 扩展运算符（展开语法） 43](#_Toc27287)

[1.2 构造函数方法：Array.from( ) 44](#_Toc19198)

[1.3 实例方法：find( ) 45](#_Toc18950)

[1.4 实例方法：findIndex( ) 46](#_Toc9577)

[1.5 实例方法：includes( ) 46](#_Toc7744)

[9.String（字符串）的扩展方法 47](#_Toc5479)

[1.1 模板字符串 47](#_Toc17116)

[1.2 实例方法: startsWith( )和endsWith( ) 48](#_Toc6642)

[1.3 实例方法: repeat( ) 48](#_Toc28092)

[10.Set数据结构 48](#_Toc16622)

[1.1 利用Set数据结构做数组去重 49](#_Toc3736)

[1.2 实例方法 49](#_Toc30473)

[1.3 遍历 50](#_Toc5080)

**第一节 面向对象编程介绍**

**1.两大编程思想**

·面向过程

·面向对象

**1.1 面向过程编程POP(Process-oriented programming)**

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候再一个一个的依次调用就可以了。



**面向过程，就是按照我们分析好了的步骤，按照步骤解决问题。**

**1.2 面向对象编程OOP(Object Oriented Programming)**

面向对象是把事务分解成为一个个对象，然后由对象之间分工与合作。

举个栗子∶将大象装进冰箱，面向对象做法。

先找出对象，并写出这些对象的功能:

1.大象对象

·进去

2.冰箱对象

·打开

·关闭

1. 使用大象和冰箱的功能

**面向对象是以对象功能来划分问题，而不是步骤。**

在面向对象程序开发思想中，每一个对象都是功能中心，具有明确分工。

面向对象编程具有灵活、代码可复用、容易维护和开发的优点，更适合多人合作的大型软件项目。

面向对象的特性:

·封装性

·继承性

·多态性

**1.3 面向过程和面向对象的对比**

****

**面向过程，就是按照我们分析好了的步骤，按照步骤解决问题。**

**面向对象是以对象功能来划分问题，而不是步骤。**

**第二节 ES6中的类和对象**

**1.面向对象**

·面向对象更贴近我们的实际生活,可以使用面向对象描述现实世界事物.但是事物分为具体的事物和抽象的事物。

·面向对象的思维特点:

1．抽取（抽象）对象共用的属性和行为组织(封装)成一个类(模板)

2．对类进行实例化,获取类的对象

·面向对象编程我们考虑的是有哪些对象，按照面向对象的思维特点不断的创建对象,使用对象,指挥对象做事情。

**2.对象**

·现实生活中︰万物皆对象，对象是一个具体的事物，看得见摸得着的实物。例如，一本书、一辆汽车、一个人可以是“对象”，一个数据库、一张网页、一个与远程服务器的连接也可以是“对象”。

·在JavaScript 中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等。

·对象是由属性和方法组成的:

·属性∶事物的特征，在对象中用属性来表示(常用名词)

·方法:事物的行为，在对象中用方法来表示(常用动词)

**3.类 class**

·在ES6中新增加了类的概念，可以使用class关键字声明一个类，之后以这个类来实例化对象。

·类抽象了对象的公共部分，它泛指某一大类( class )

·对象特指某一个，通过类实例化一个具体的对象

·面向对象的思维特点:

1．抽取（抽象）对象共用的属性和行为组织(封装)成一个类(模板)

2．对类进行实例化,获取类的对象

**1.1创建类**

语法：



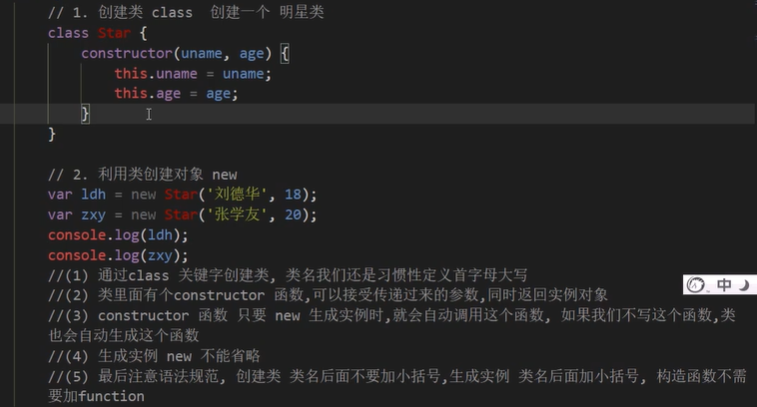
创建实例：

T9}7KC1HPIN3OX(A6F4[PYR

注意：类必须使用new实例化对象。

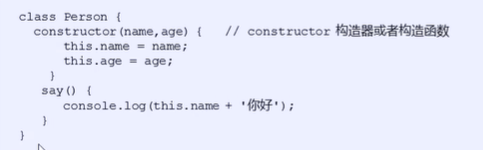
**1.2类constructor构造函数**

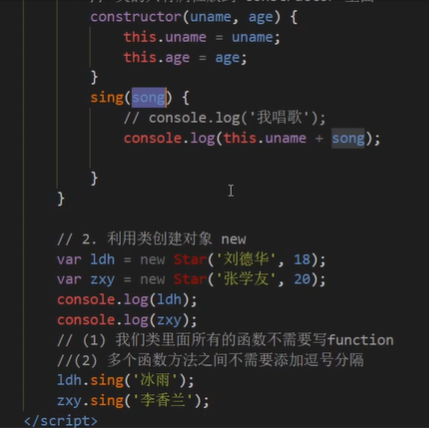
constructor( )方法是类的构造函数(默认方法)，用于传递参数,返回实例对象，通过new命令生成对象实例时，自动调用该方法。如果没有显示定义,类内部会自动给我们创建一个constructor( )



**1.3 类添加方法**

语法：





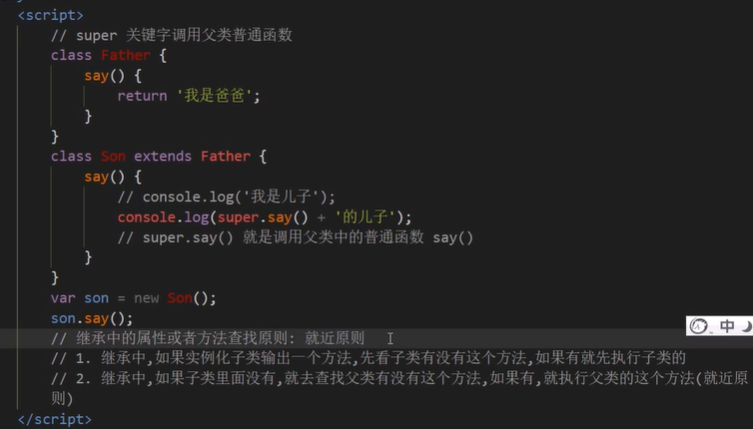
**第三节 类的继承**

**1.继承**

程序中的继承︰子类可以继承父类的一些属性和方法。

语法：

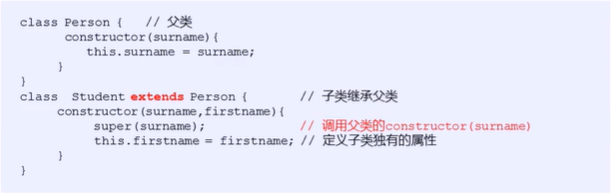




**2.super关键字**

super关键字用于访问和调用父类对象上的函数。可以调用父类的构造函数，也可以调用父类的普通函数。

·语法：



·注意:子类在构造函数中使用super,必须放到this前面(必须先调用父类的构造方法在使用子类构造方法)

**3.三个注意点**

·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。

**第四节 构造函数和原型**

**1.概述**

·在典型的OOP的语言中(如Java )，都存在类的概念，类就是对象的模板，对象就是类的实例，但在ES6之前，JS中并没用引入类的概念。

·ES6，全称ECMAScript 6.0，2015.06发版。但是目前浏览器的JavaScript是ES5版本，大多数高版本的浏览器也支持ES6，不过只实现了ES6的部分特性和功能。

·在ES6之前，对象不是基于类创建的，而是用一种称为构建函数的特殊函数来定义对象和它们的特征。

·创建对象可以通过以下三种方式:

1.对象字面量

2.new Object

3.自定义构造函数

**2.构造函数**

·构造函数是一种特殊的函数，主要用来初始化对象，即为对象成员变量赋初始值，它总与new一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。

·New在执行时会做四件事情:

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。

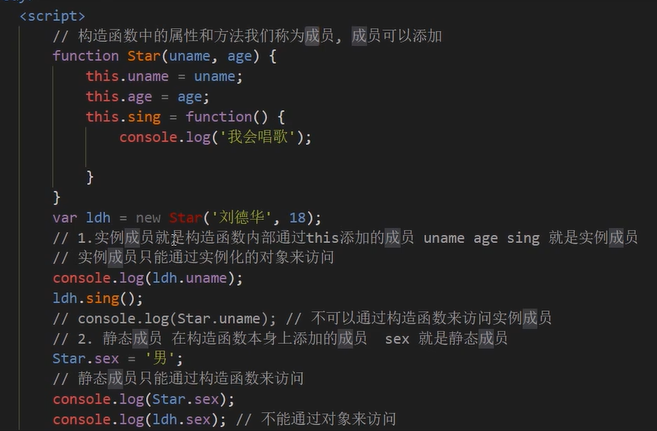
③执行构造函数里面的代码，给这个新对象添加属性和方法。

④返回这个新对象（所以构造函数里面不需要return )。

·JavaScript的构造函数中可以添加一些成员，可以在构造函数本身上添加，也可以在构造函数内部的this上添加。通过这两种方式添加的成员，就分别称为静态成员和实例成员。

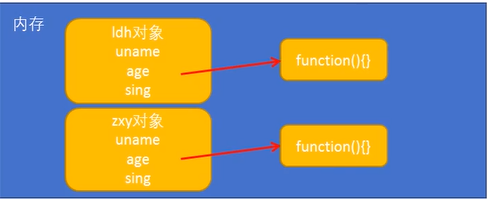
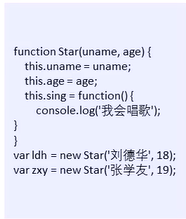
·静态成员:在构造函数本上添加的成员称为静态成员，只能由构造函数本身来访问

·实例成员:在构造函数内部创建的对象成员称为实例成员，只能由实例化的对象来访问



**1.1 构造函数的问题**

构造函数方法很好用,但是存在浪费内存的问题。



**1.2 构造函数原型prototype**

·构造函数通过原型分配的函数是所有对象所共享的。

·JavaScript规定，每一个构造函数都有一个prototype属性，指向另一个对象。注意这个prototype就是一个对象，这个对象的所有属性和方法都会被构造函数所拥有。

·我们可以把那些不变的方法，直接定义在prototype对象上，这样所有对象的实例就可以共享这些方法。

·一般情况下,我们的公共属性定义到构造函数里面，公共的方法我们放到原型对象身上。

1.原型是什么?

一个对象，我们也称为prototype为原型对象。

2.原型的作用是什么?

共享方法。

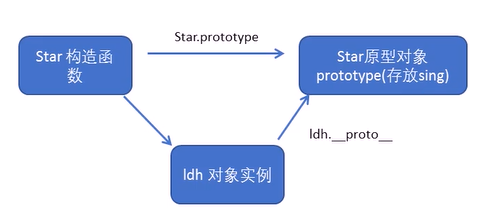


**3.对象原型\_proto\_**

对象都会有一个属性\_proto\_指向构造函数的prototype原型对象，之所以我们对象可以使用构造函数prototype原型对象的属性和方法，就是因为对象有\_proto\_原型的存在。

·\_proto\_对象原型和原型对象prototype是等价的。

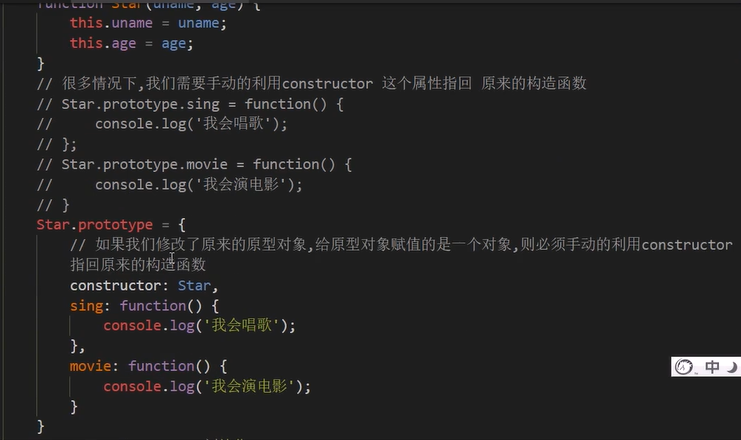
·\_proto\_对象原型的意义就在于为对象的查找机制提供一个方向，或者说一条路线，但是它是一个非标准属性，因此实际开发中，不可以使用这个属性，它只是内部指向原型对象prototype



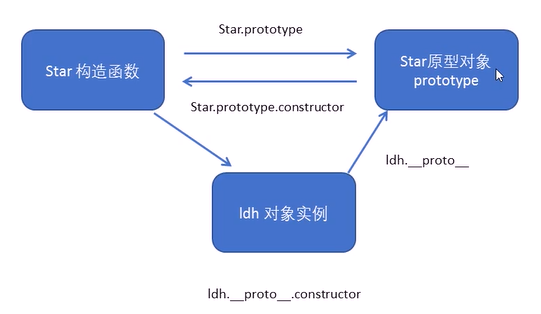
**4.constructor构造函数**

·对象原型(\_proto\_)和构造函数( prototype )原型对象里面都有一个属性constructor属性，constructor我们称为构造函数，因为它指回构造函数本身。

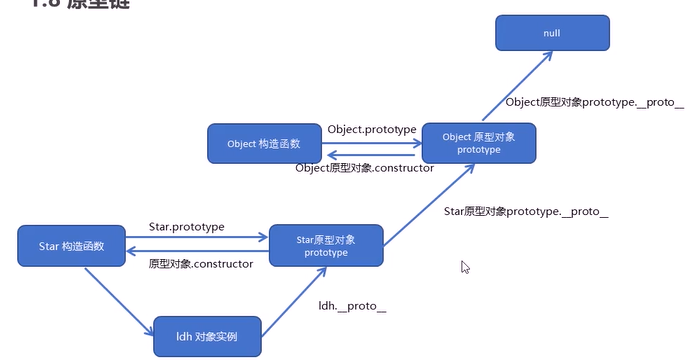
·constructor主要用于记录该对象引用于哪个构造函数，它可以让原型对象重新指向原来的构造函数。



**5.构造函数、实例、原型对象三者之间的关系**



**6.原型链**



**7.JavaScript的成员查找机制（规则）**

①当访问一个对象的属性(包括方法)时，首先查找这个对象自身有没有该属性。

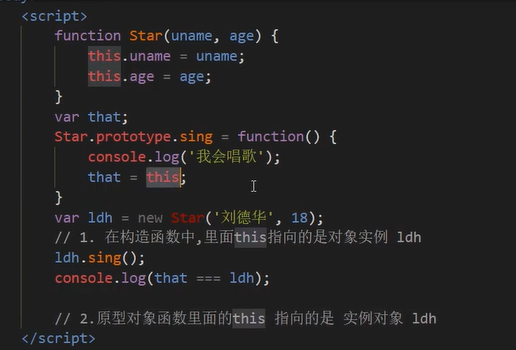
②如果没有就查找它的原型(也就是\_proto\_指向的prototype原型对象)。

③如果还没有就查找原型对象的原型(Object的原型对象)。

④依此类推一直找到Object为止 ( null )。

⑤\_proto\_对象原型的意义就在于为对象成员查找机制提供一个方向，或者说一条路线。

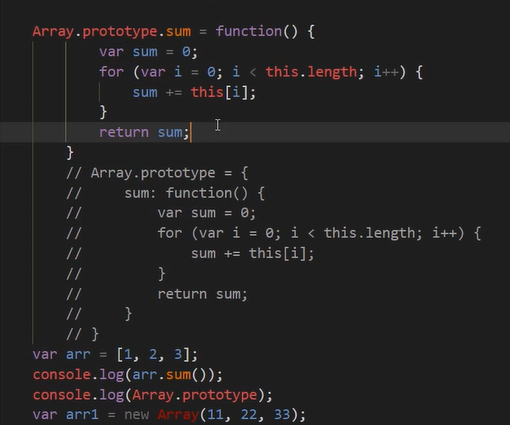
**8.原型对象this指向**



**9.扩展内置对象**

·可以通过原型对象，对原来的内置对象进行扩展自定义的方法。比如给数组增加自定义求偶数和的功能。

·注意∶数组和字符串内置对象不能给原型对象覆盖操作Array.prototype ={}，只能是Aray.prototype.xx = function（）{}的方式。



**第五节 继承**

ES6之前并没有给我们提供extends继承。我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

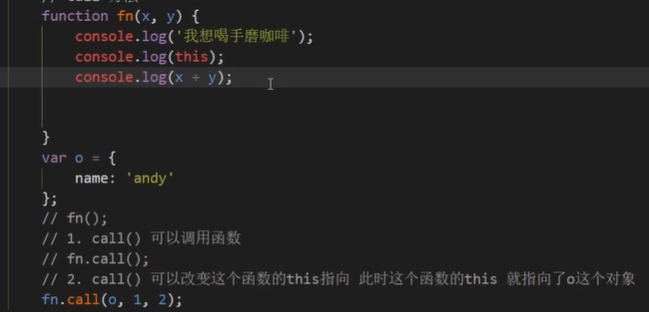
**1.call( )**

调用这个函数并且修改函数运行时的this指向

MXZ5XS~{Y@8{GFXCS}IP%)S

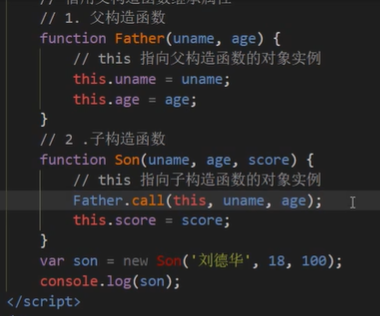
·thisArg :当前调用函数this的指向对象

·arg1 , arg2∶传递的其他参数

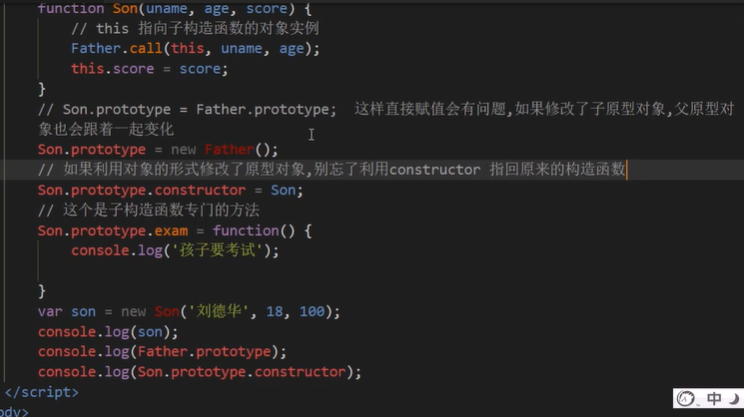


**2.借用构造函数继承父类型属性**

核心原理:通过call( )把父类型的this 指向子类型的this，这样就可以实现子类型继承父类型的属性。



**3.借用原型对象继承父类型方法**



**4.类的本质**

1.class本质还是function.

2.类的所有方法都定义在类的prototype属性上

3.类创建的实例,里面也有\_proto\_指向类的prototype原型对象

4.所以ES6的类它的绝大部分功能，ES5都可以做到，新的class写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已。

5.所以ES6的类其实就是语法糖。

6.语法糖:语法糖就是一种便捷写法，简单理解,有两种方法可以实现同样的功能,但是一种写法更加清晰、方便，那么这个方法就是语法糖。



**5.ES5中的新增方法**

ES5中给我们新增了一些方法，可以很方便的操作数组或者字符串，这些方法主要包括:

·数组方法

·字符串方法

·对象方法

**1.1数组方法**

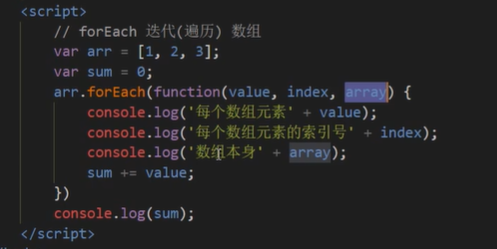
迭代(遍历)方法: forEach( )、map( )、filter( )、some( )、every( )、reduce( ) ;

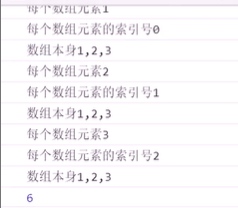
73R1RKAFK2)8QAGK[E~30@M

·currentValue:数组当前项的值

·index:数组当前项的索引

·arr:数组对象本身





58DA]5485P2LTAD%%(KJ]QM

·filter( )方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素,主要用于筛选数组

·注意它直接返回一个新数组

·currentValue:数组当前项的值

·index:数组当前项的索引

·arr:数组对象本身



@6O36SRF]GZ9BB`LO$FPN5X

·Some()方法用于检测数组中的元素是否满足指定条件.通俗点查找数组中是否有满足条件的元素

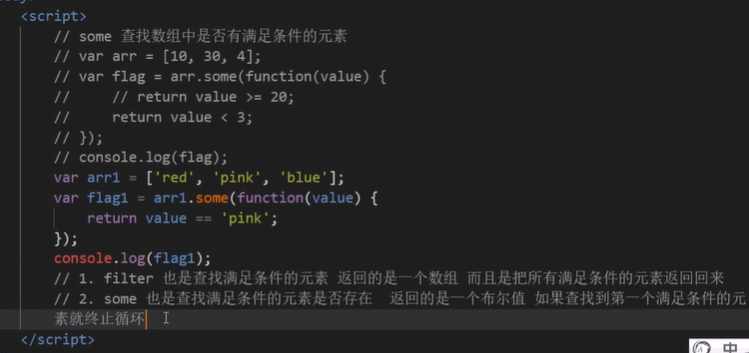
·注意它返回值是布尔值如果查找到这个元素,就返回true，如果查找不到就返回false

·如果找到第一个满足条件的元素,则终止循环不在继续查找

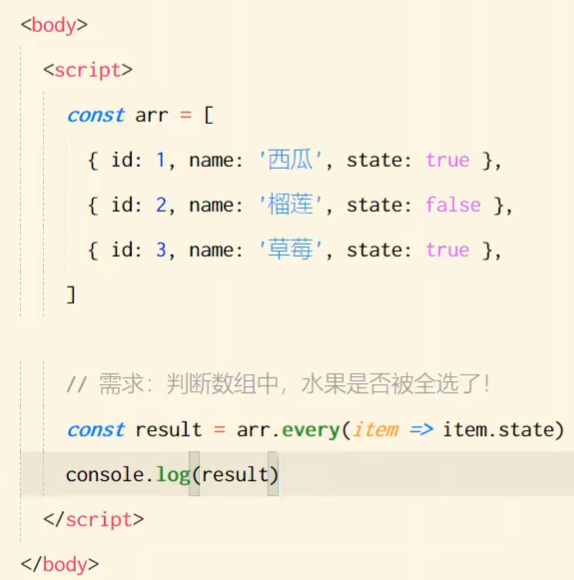
·currentValue:数组当前项的值

·index:数组当前项的索引

·arr :数组对象本身



every( )可用来判断数组中的元素是否被选中，返回的值是布尔型



reduce( )可用来把每一次循环的结果累加起来



**1.2 forEach和some的区别**

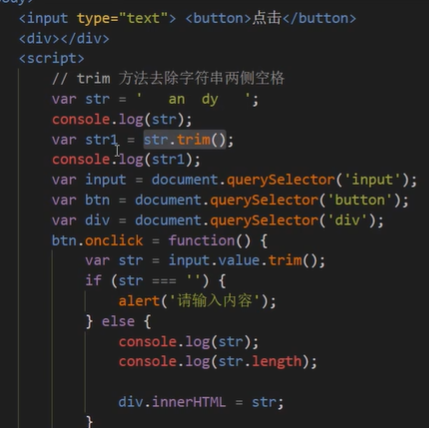
如果查询数组中唯一的元素，用some方法更合适，因为它找到这个元素，就不再进行循环，效率更高。在forEach和filter里面return不会终止迭代，在some里面遇到return true就是终止遍历，迭代效率更高。

**1.3字符串方法**

trim( )方法会从一个字符串的两端删除空白字符。

4}Y]JC6]_OVAMECVS7O53QQ

trim ( )方法并不影响原字符串本身，它返回的是一个新的字符串。



**1.4 对象方法**

①Object.defineProperty( )定义对象中新属性或修改原有的属性。

}Q5N(7CWWK64G}{0G$9EX5C

·obj:必需。目标对象

·prop:必需。需定义或修改的属性的名字

·descriptor :必需。目标属性所拥有的特性

Object.defineProperty( )第三个参数descriptor说明:以对象形式{}书写

·value:设置属性的值默认为undefined

·writable:值是否可以重写。true | false 默认为false

·enumerable:目标属性是否可以被枚举（是否可以看见）。true | false默认为false

·configurable:目标属性是否可以被删除或是否可以再次修改特性true | false 默认为false。如果为false则不允许删除或者修改这个属性。

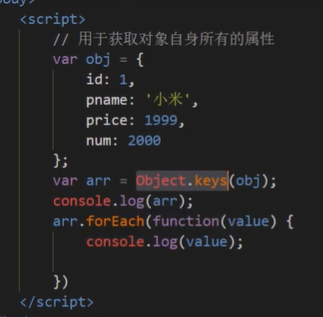


②Object.keys( )用于获取对象自身所有的属性

@49(CK`}HBW7YUPSYV@O$(D

·效果类似for...in

·返回一个由属性名组成的数组



**第六节 函数进阶**

**1.函数的定义和调用**

**1.1函数的定义方式**

1. 函数声明方式function关键字(命名函数)

2.函数表达式(匿名函数)

3.new Function( )

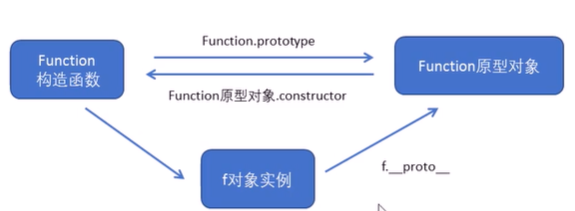
9}B6R0EH2SJ7`$DPU}YV6HX

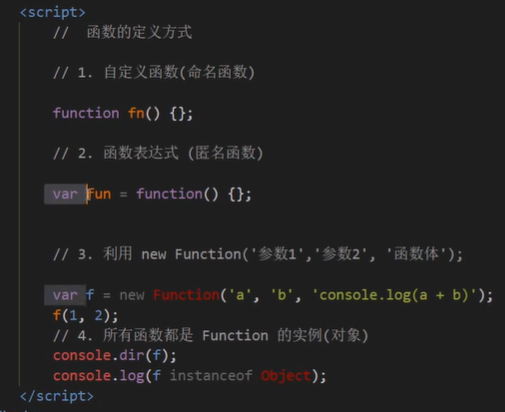
·Function里面参数都必须是字符串格式

·第三种方式执行效率低，也不方便书写，因此较少使用

·所有函数都是Function的实例(对象)

·函数也属于对象





**1.2 函数的调用方式**

1.普通函数

2.对象的方法

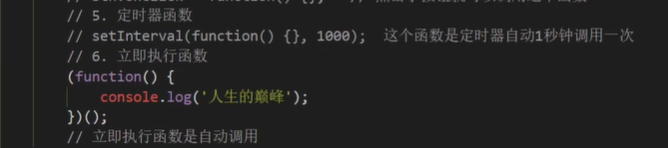
3.构造函数

4.绑定事件函数

5.定时器函数

6.立即执行函数





**2.this**

**1.1 函数内this的指向**

这些this的指向，是当我们调用函数的时候确定的。调用方式的不同决定了this的指向不同，一般指向我们的调用者。



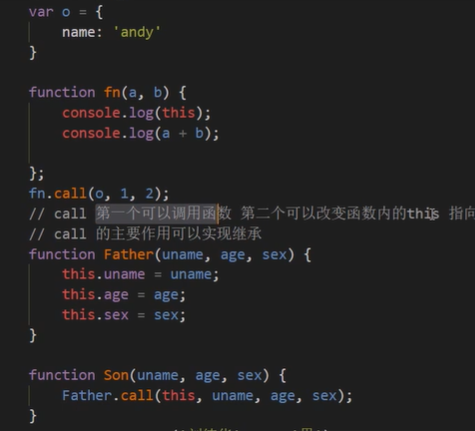
**1.2 改变函数内部this的指向**

JavaScript为我们专门提供了一些函数方法来帮我们更优雅的处理函数内部this 的指向问题，常用的有 bind( )、call( )、 apply( )三种方法。

**① call方法**

call( )方法调用一个对象。简单理解为调用函数的方式，但是它可以改变函数的this指向。

I3JJ0Y@XE@`_UY]OZJ84IOY



**② apply方法**

apply( )方法调用一个函数。简单理解为调用函数的方式，但是它可以改变函数的this指向。

LT69M)YD3MNP~]Z6E[OFAS4

·thisArg :在fun函数运行时指定的this值

·argsArray :传递的值，必须包含在数组里面

·返回值就是函数的返回值，因为它就是调用函数



**③ bind方法**

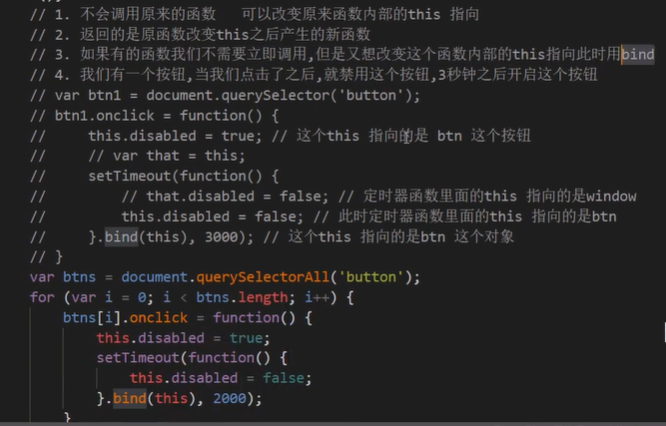
bind( )方法不会调用函数。但是能改变函数内部this指向

E8K{DZNBK)L$Z0CV59P%PRC

·thisArg :在fun函数运行时指定的this值

·arg1 , arg2:传递的其他参数

·返回由指定的this值和初始化参数改造的原函数拷贝



**② call、apply、bind方法总结**

**·相同点:**

都可以改变函数内部的this指向。

**·区别点:**

1. call和 apply 会调用函数,并且改变函数内部this指向。

2. call和apply传递的参数不一样, call传递参数aru1, aru2..形式 apply必须数组形式[arg]。

3. bind 不会调用函数,可以改变函数内部this指向。

**·主要应用场景:**

1. call经常做继承。

2. apply经常跟数组有关系.比如借助于数学对象实现数组最大值最小值。

3. bind不调用函数,但是还想改变this指向.比如改变定时器内部的this指向。

**3.严格模式**

**1.1 什么是严格模式**

·JavaScript除了提供正常模式外，还提供了严格模式( strict mode )。ES5的严格模式是采用具有限制性JavaScript变体的一种方式，即在严格的条件下运行JS代码。

·严格模式在IE10以上版本的浏览器中才会被支持，旧版本浏览器中会被忽略。

·严格模式对正常的JavaScript语义做了一些更改:

1．消除了Javascript语法的一些不合理、不严谨之处，来少了一些怪异行为。2．消除代码运行的一些不安全之处，保证代码运行的安全。

3．提高编译器效率，增加运行速度。

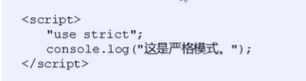
4．禁用了在ECMAScript的未来版本中可能会定义的一些语法，为未来新版本的Javascript做好铺垫。比如一些保留字如: class, enum, export, extends, import, super不能做变量名

**1.2 开启严格模式**

严格模式可以应用到整个脚本或个别函数中。因此在使用时，我们可以将严格模式分为为脚本开启严格模式和为函数开启严格模式两种情况。

**①为脚本开启严格模式**

为整个脚本文件开启严格模式，需要在所有语句之前放一个特定语句“use strict”;(或‘use strict' ;)。



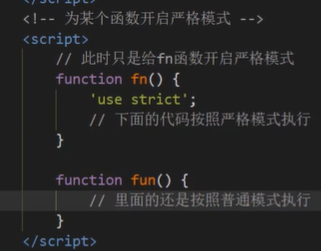
因为"use strict"加了引号，所以老版本的浏览器会把它当作一行普通字符串而忽略。

有的 script脚本是严格模式，有的script脚本是正常模式，这样不利于文件合并，所以可以将整个脚本文件放在一个立即执行的匿名函数之中。这样独立创建一个作用域而不影响其他script脚本文件。



**①为函数开启严格模式**

要给某个函数开启严格模式，需要把“use strict”:(或 'use strict';)声明放在函数体所有语句之前。



**1.3 严格模式中的变化**

严格模式对Javascript的语法和行为，都做了一些改变。

**①变量规定**

·在正常模式中，如果一个变量没有声明就赋值，默认是全局变量。严格模式禁止这种用法，变量都必须先用var命令声明，然后再使用。

·严禁删除已经声明变量。例如，delete x;语法是错误的。

**②严格模式下this指向问题**

·以前在全局作用域函数中的 this指向window对象。

**·严格模式下全局作用域中函数中的this是 undefined。**

·以前构造函数时不加new也可以调用,当普通函数，this指向全局对象。

·严格模式下,如果构造函数不加new调用, this会报错。

·new实例化的构造函数指向创建的对象实例。

·定时器this还是指向window。

·事件、对象还是指向调用者。

**③函数变化**

·函数不能有重名的参数

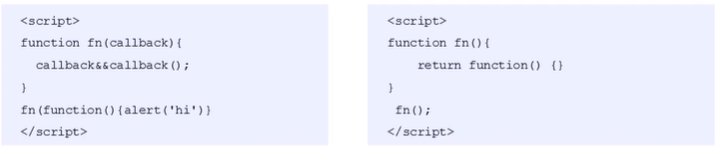
·函数必须声明在顶层.新版本的JavaScript 会引入“块级作用域”(ES6中已引入)。为了与新版本接轨，不允许在非函数的代码块内声明函数。

**更多严格模式要求参考：**

**https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict\_mode**

**4.高阶函数**

高阶函数是对其他函数进行操作的函数，它接收函数作为参数或将函数作为返回值输出。



此时fn就是一个高阶函数。

函数也是一种数据类型，同样可以作为参数，传递给另外一个参数使用。最典型的就是作为回调函数。

**5.闭包**

**1.1变量作用域**

变量根据作用域的不同分为两种:全局变量和局部变量。

1. 函数内部可以使用全局变量。

⒉函数外部不可以使用局部变量。

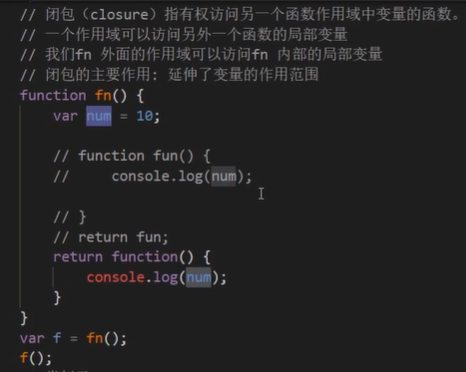
3.当函数执行完毕，本作用域内的局部变量会销毁。

**1.2什么是闭包**

闭包( closure )指有权访问另一个函数作用域中变量的函数。----- JavaScript高级程序设计

简单理解就是，一个作用域可以访问另外一个函数内部的局部变量。

闭包的作用：延伸了变量的作用范围。

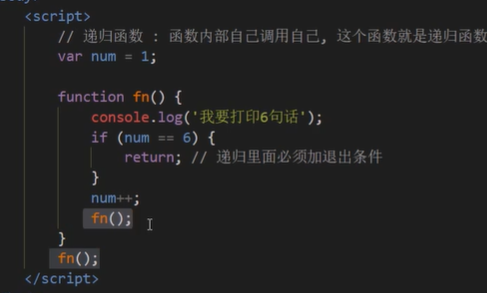


**6.递归**

如果一个函数在内部可以调用其本身，那么这个函数就是递归函数。简单理解:函数内部自己调用自己这个函数就是递归函数。

递归函数的作用和循环效果一样。

由于递归很容易发生“栈溢出”错误( stack overflow )，所以必须要加退出条件return。

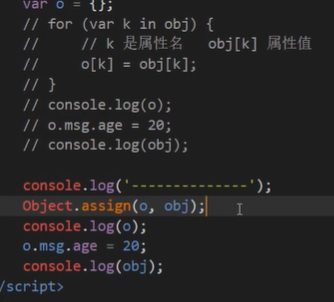


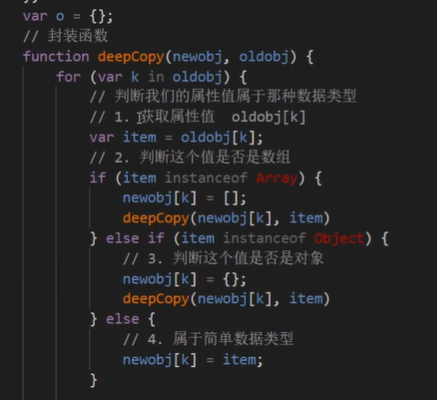
**1.1浅拷贝和深拷贝**

1. ．浅拷贝只是拷贝一层,更深层次对象级别的只拷贝引用。（拷贝地址给新的对象）

2．深拷贝拷贝多层,每一级别的数据都会拷贝。（重新开辟地址给对象）

**3.Object.assign( target， ...sources) es6新增方法可以浅拷贝**





**第七节 正则表达式**

**1.什么是正则表达式**

正则表达式(Regular Expression )是用于匹配字符串中字符组合的模式。在JavaScript中，正则表达式也是对象。

正则表通常被用来检索、替换那些符合某个模式（规则)的文本，例如验证表单;用户名表单只能输入英文字母、数字或者下划线，昵称输入框中可以输入中文(匹配)。此外，正则表达式还常用于过滤掉页面内容中的一些敏感词(替换)，或从字符串中获取我们想要的特定部分(提取)等。

其他语言也会使用正则表达式，本阶段我们主要是利用JavaScript 正则表达式完成表单验证。

**2.正则表达式的特点**

1.灵活性、逻辑性和功能性非常的强。

2.可以迅速地用极简单的方式达到字符串的复杂控制。

3.对于刚接触的人来说，比较晦涩难懂。比如:^\w+([-+.J]lw+)\*@\w+([-J]\w+)\*\L.Iw+([-Jw+)\*$

4.实际开发,一般都是直接复制写好的正则表达式.但是要求会使用正则表达式并且根据实际情况修改正则表达式. 比如用户名:/^[a-z0-9\_-][3,16}$/

**3.正则表达式在JavaScript中的使用**

**1.1创建正则表达式**

在JavaScript中，可以通过两种方式创建一个正则表达式。

**①通过调用RegExp对象的构造函数创建**

0]8HTSFTCNCVL}K`D_}K98Q

**②通过字面量创建**

DC~WZ8POFLXHAG2WB}W7VWY

//注释中间放表达式就是正则字面量

**1.2 测试正则表达式test**

test( )正则对象方法，用于检测字符串是否符合该规则，该对象会返回true或false，其参数是测试字符串。

NV0{NY%I`TWNSVCS5K{3%8O

1. regexobj是写的正则表达式。

2. str我们要测试的文本。

3．就是检测str文本是否符合我们写的正则表达式规范。



**4.正则表达式中的特殊字符**

**1.1正则表达式的组成**

一个正则表达式可以由简单的字符构成，比如/abc/，也可以是简单和特殊字符的组合，比如/ab\*c/。其中特殊字符也被称为元字符，在正则表达式中是具有特殊意义的专用符号，如~、$、+等。

特殊字符非常多，可以参考∶

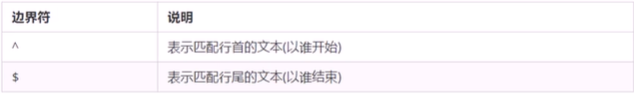
·MDN:https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide/Regular\_Expressions

·jQuery手册:正则表达式部分

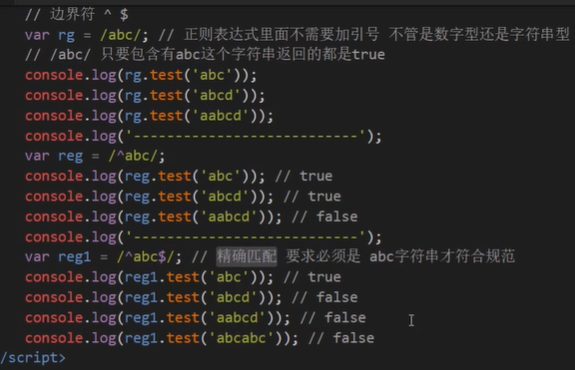
·正则测试工具: http://tool.oschina.net/regex

**1.2 边界符**

正则表达式中的边界符(位置符）用来提示字符所处的位置，主要有两个字符。



如果^和$在一起，表示必须是精确匹配。



**1.3 字符类**

字符类表示有一系列字符可供选择，只要匹配其中一个就可以了。所有可供选择的字符都放在方括号内。

[-]方括号内部范围符-



字符组合

W7GN0%}JK5%77K_$[C6HW~7

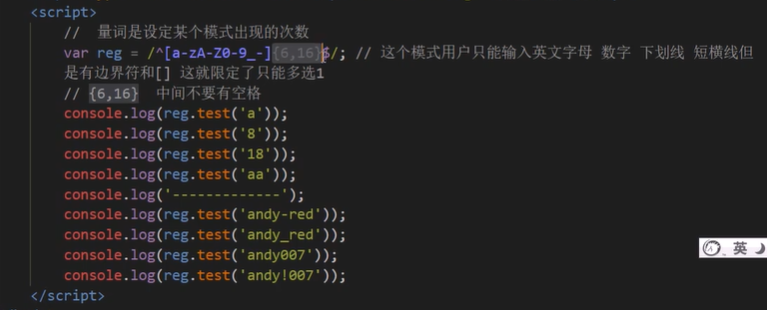
[^]方括号内部取反符^

N7QFAY2N6(U(GBWL9%1Q0GG

**1.4 量词符**

量词符用来设定某个模式出现的次数。





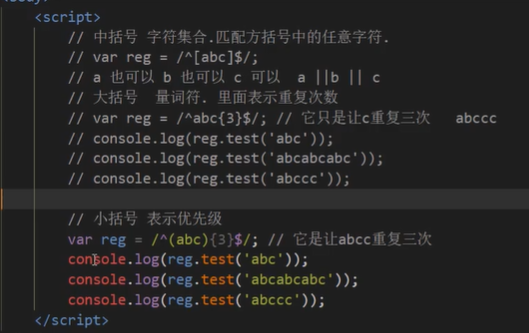
**1.5 括号总结**

1.大括号量词符。里面表示重复次数

2.中括号字符集合。匹配方括号中的任意字符

3.小括号表示优先级

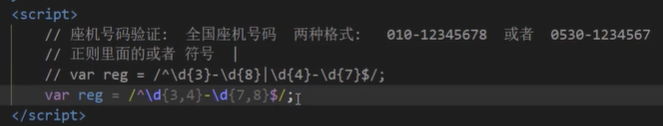
可以在线测试: https://c.runoob.com/



**1.6 预定义类**

预定义类指的是某些常见模式的简写方式。





**5.正则表达式中的替换**

**1.1 replace替换**

replace( )方法可以实现替换字符串操作，用来替换的参数可以是一个字符串或是一个正则表达式。

U(527LT]CDPIN]C]51X7V]Q

1. 第一个参数:被替换的字符串或者正则表达式

2.第二个参数:替换为的字符串

3.返回值是一个替换完毕的新字符串

**1.2 正则表达式参数**

D{E3}L2CXD6~4[1)M%~R((J

switch(也称为修饰符)按照什么样的模式来匹配.有三种值:

·g∶全局匹配

·i:忽略大小写

·gi :全局匹配＋忽略大小写



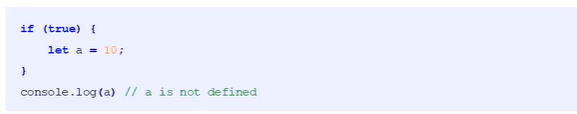
**第八节 ES6简介**

ES的全称是ECMAScript ,它是由ECMA国际标准化组织,制定的一项脚本语言的标准化规范。ES6实际上是一个泛指，泛指ES2015及后续的版本。

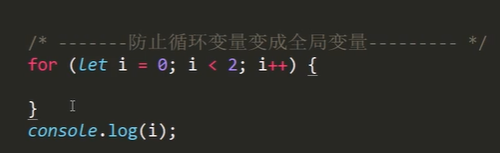
**1.let关键字**

ES6中新增的用于声明变量的关键字。

·let声明的变量只在所外于的块级有效（作用于一个大括号内的作用域就是块级作用域）。



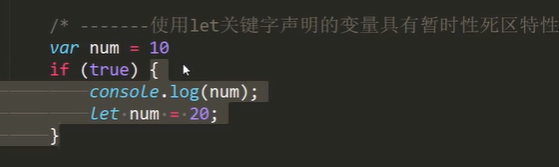
注意:使用let关键字声明的变量才具有块级作用域，使用var声明的变量不具备块级作用域特性。防止循环变量变成全局变量



·不存在变量提升(先声明再使用)

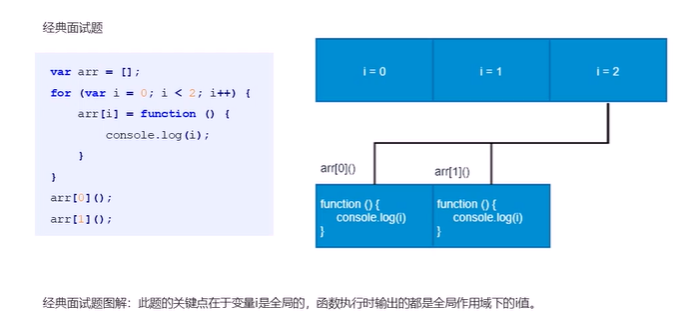
8MN57HLKEW2%9K5AC6WLTVQ

·暂时性死区（在块级作用域声明的let变量会与整个块级作用域进行绑定）

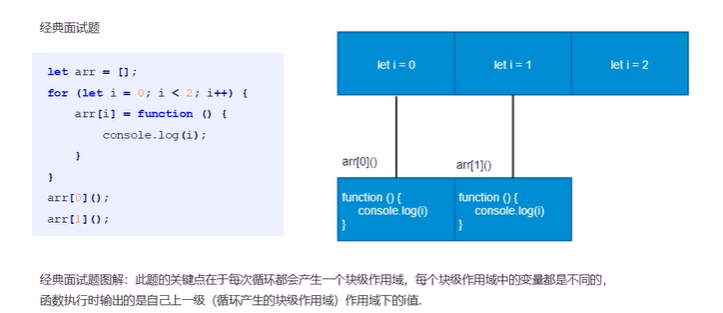


在大括号内使用num这个变量，和外部没有任何关系。

**2.经典面试题（一）**



经典面试题图解:此题的关键点在于变量i是全局的，函数执行时输出的都是全局作用域下的值,

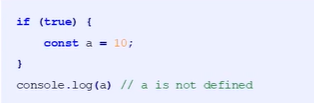


经典面试题图解:此题的关键点在于每次循环都会产生一个块级作用域，每个块级作用域中的变量都是不同的,函数执行时输出的是自己上一级(循环产生的块级作用域)作用域下的i值.

**3.const关键字**

作用:声明常量，常量就是值(内存地址)不能变化的量。

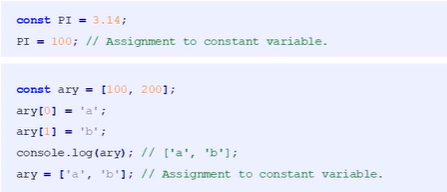
·具有块级作用域



·声明常量时必须赋值

6$Z3DRO[S__2%N1{C}3Z$]N

·常量赋值后,值不能修改（不能改变值的存储地址）。



**4.let、var、const的区别**

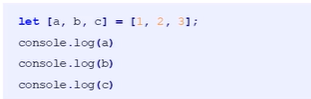
1.使用var声明的变量，其作用域为该语句所在的函数内，且存在变量提升现象。2.使用let声明的变量，其作用域为该语句所在的代码块内，不存在变量提升。3.使用const声明的是常量，在后面出现的代码中不能再修改该常量的值。



**5.解构赋值**

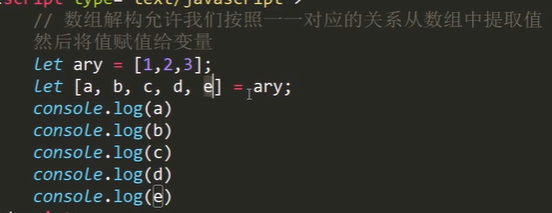
ES6中允许从数组中提取值，按照对应位置，对变量赋值。对象也可以实现解构。

**数组解构**

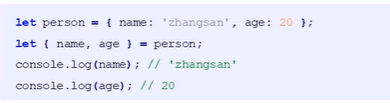


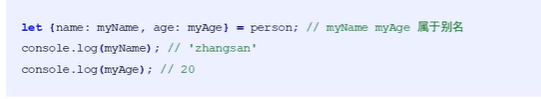
如果解构不成功。变量的值为undefined。

J~4RWY`DWJV2VZR(F]JT2%K



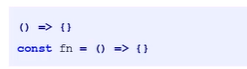
**对象解构**



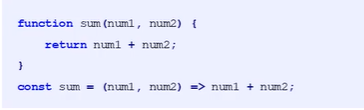


**6.箭头函数**

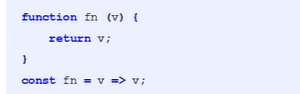
ES6中新增的定义函数的方式，用来简化函数定义语法

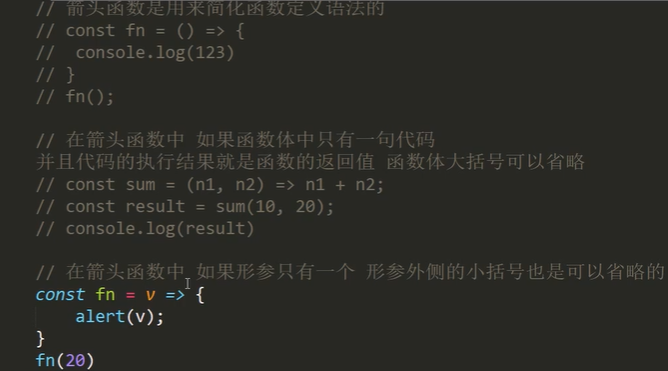


·函数体中只有一句代码，且代码的执行结果就是返回值，可以省略大括号

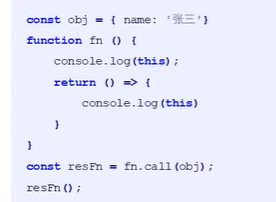


·如果形参只有一个，可以省略小括号





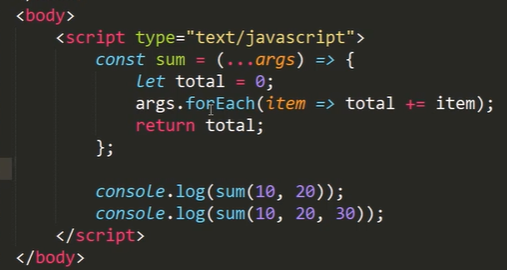
·箭头函数不绑定this关键字，箭头函数中的this，指向的是函数定义位置指向的this。



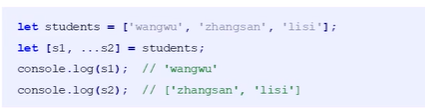
**7.剩余参数**

剩余参数语法允许我们将一个不定数量的参数表示为一个数组。





剩余参数和解构配合使用



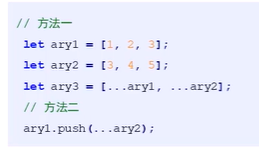
**8.Array的扩展方法**

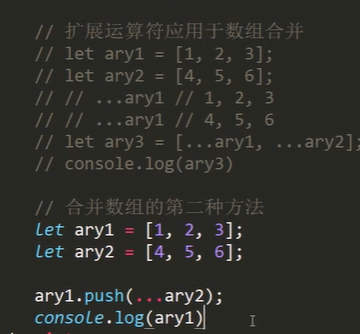
**1.1 扩展运算符（展开语法）**

·扩展运算符可以将数组或者对象转为用逗号分隔的参数序列。



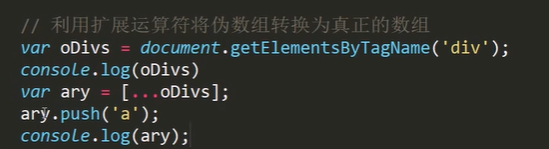
·扩展运算符可以应用于合并数组





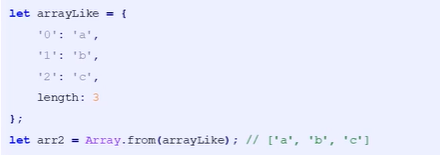
·将伪数组或可遍历对象转换为真正的数组

~]VB6C7D3FUM[)QFH1PT4NP



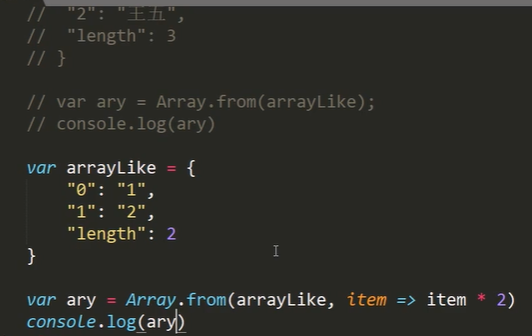
**1.2 构造函数方法：Array.from( )**

将伪数组或可遍历对象转换为真正的数组



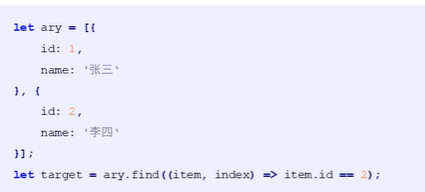
方法还可以接受第二个参数，作用类似于数组的map方法，用来对每个元素进行处理，将处理后的值放入返回的数组。





**1.3 实例方法：find( )**

用于找出第一个符合条件的数组成员，如果没有找到返回undefined

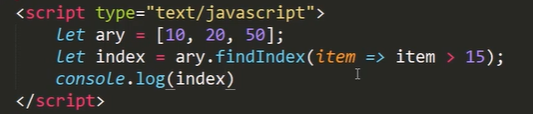




**1.4 实例方法：findIndex( )**

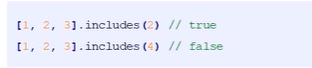
用于找出第一个符合条件的数组成员的位置。如果没有找到返回-1

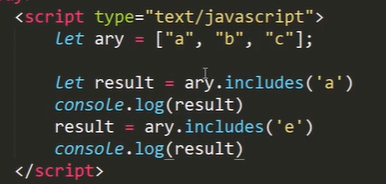




**1.5 实例方法：includes( )**

表示某个数组是否包含某个值,返回布尔值。





**9.String（字符串）的扩展方法**

**1.1 模板字符串**

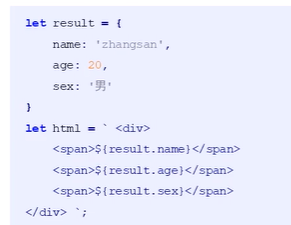
ES6新增的创建字符串的方式。使用反引号(**` `**)定义。

B1]N9])}FRD8(E4ME%H277Y

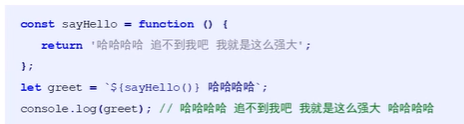
·模板字符串中可以解析变量。

D$`YSLO4YRD~W~NNSF9PM1K

·模板字符串可以换行



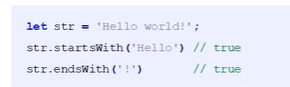
·在模板字符串中可以调用函数



**1.2 实例方法: startsWith( )和endsWith( )**

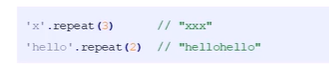
·startsWith( ):表示参数字符串是否在原字符串的头部，返回布尔值

·endsWith( ):表示参数字符串是否在原字符串的尾部，返回布尔值



**1.3 实例方法: repeat( )**

repeat方法表示将原字符串重复n次，返回一个新字符串。



**10.Set数据结构**

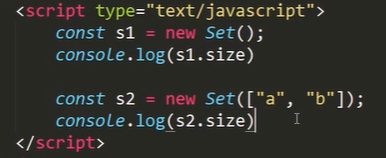
ES6提供了新的数据结构Set。它类似于数组，但是成员的值都是唯一的，没有重复的值。

Set本身是一个构造函数，用来生成Set数据结构

]3PTH038GAT%GEYZQ%]2COH

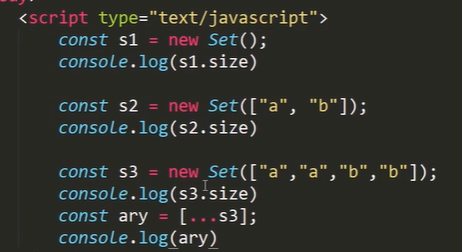
Set函数可以接受一个数组作为参数，用来初始化

)][_D~%5(Z37_OZ@]%[FFPI



**1.1 利用Set数据结构做数组去重**

向Set数据结构中传递初始值时，会把重复的值删掉



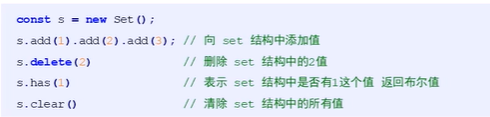
**1.2 实例方法**

·add(value):添加某个值,返回Set结构本身

·delete(value):删除某个值，返回一个布尔值，表示删除是否成功

·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



**1.3 遍历**

Set结构的实例与数组一样，也拥有forEach方法，用于对每个成员执行某种操作。没有返回值。

[HJK9H95S_(2FLQ`(F[57{3

