前端基本功-JS基础

目录

[前端基本功-JS基础 1](#_Toc479172225)

[day01 4](#_Toc479172226)

[01阶段概述和JS简介.avi 4](#_Toc479172227)

[02浏览器原理简介.avi 6](#_Toc479172228)

[03基本输出语句.avi 7](#_Toc479172229)

[04代码注释.avi 9](#_Toc479172230)

[05JS的书写位置.avi 10](#_Toc479172231)

[06script标签的属性.avi 14](#_Toc479172232)

[07变量的命名.avi 15](#_Toc479172233)

[08变量的声明赋值和使用.avi 16](#_Toc479172234)

[09变量在内存中的存储.avi 17](#_Toc479172235)

[10交换变量练习.avi 18](#_Toc479172236)

[11数据类型.avi 19](#_Toc479172237)

[12number类型.avi 20](#_Toc479172238)

[13数值范围.avi 21](#_Toc479172239)

[14数值检测.avi 21](#_Toc479172240)

[15string类型.avi 22](#_Toc479172241)

[16字符串的不可变性.avi 23](#_Toc479172242)

[17字符串的拼接.avi 24](#_Toc479172243)

[18boolean类型.avi 24](#_Toc479172244)

[19undefined类型.avi 26](#_Toc479172245)

[20其他类型值转string.avi 26](#_Toc479172246)

[21其他类型值转number.avi 28](#_Toc479172247)

[22其他类型值转boolean.avi 29](#_Toc479172248)

[23算术运算符.avi 29](#_Toc479172249)

[24一元运算符.avi 30](#_Toc479172250)

[25逻辑运算符1.avi 31](#_Toc479172251)

[26逻辑运算符2.avi 32](#_Toc479172252)

[27其他运算符.avi 32](#_Toc479172253)

[28练习.avi 34](#_Toc479172254)

[29选择语句if.avi 34](#_Toc479172255)

[30选择语句if练习.avi 35](#_Toc479172256)

[31三元运算符.avi 37](#_Toc479172257)

[32三元练习.avi 37](#_Toc479172258)

[33选择语句switch.avi 38](#_Toc479172259)

[day02 39](#_Toc479172260)

[01选择语句switch练习.avi 40](#_Toc479172261)

[02循环语句for.avi 41](#_Toc479172262)

[03for循环的简单练习.avi 42](#_Toc479172263)

[04for循环的中等练习.avi 44](#_Toc479172264)

[05for循环的困难练习.avi 45](#_Toc479172265)

[06break和continue.avi 46](#_Toc479172266)

[07循环语句while.avi 47](#_Toc479172267)

[08循环语句while练习.avi 49](#_Toc479172268)

[09数组的声明.avi 50](#_Toc479172269)

[10数组的基本属性.avi 51](#_Toc479172270)

[11数组的遍历.avi 51](#_Toc479172271)

[12length属性的应用.avi 52](#_Toc479172272)

[13length属性详解.avi 53](#_Toc479172273)

[14数组练习1 .avi 54](#_Toc479172274)

[15数组练习2 .avi 55](#_Toc479172275)

[16数组练习3 .avi 56](#_Toc479172276)

[17冒泡排序1排序原理 .avi 57](#_Toc479172277)

[18冒泡排序2优化内层 .avi 58](#_Toc479172278)

[19冒泡排序3外层内层 .avi 60](#_Toc479172279)

[day03 61](#_Toc479172280)

[01函数介绍.avi 61](#_Toc479172281)

[02函数的参数.avi 63](#_Toc479172282)

[03函数的返回值.avi 65](#_Toc479172283)

[04函数练习1.avi 66](#_Toc479172284)

[05函数练习2.avi 67](#_Toc479172285)

[06函数练习3.avi 68](#_Toc479172286)

[07函数练习4.avi 69](#_Toc479172287)

[08函数的调试.avi 70](#_Toc479172288)

[09参数详解.avi 71](#_Toc479172289)

[10函数的返回值.avi 71](#_Toc479172290)

[11函数的两种定义方式.avi 72](#_Toc479172291)

[12全局变量和局部变量.avi 73](#_Toc479172292)

[13递归.avi 74](#_Toc479172293)

[14回调.avi 74](#_Toc479172294)

[15键值对.avi 75](#_Toc479172295)

[16英雄对象.avi 75](#_Toc479172296)

[17学生对象.avi 76](#_Toc479172297)

[18对象字面量.avi 77](#_Toc479172298)

[19访问属性的两种方式.avi 78](#_Toc479172299)

[20for循环的两种方式.avi 79](#_Toc479172300)

[day04 80](#_Toc479172301)

[01批量创建对象.avi 80](#_Toc479172302)

[02this关键字.avi 82](#_Toc479172303)

[03构造函数模式.avi 83](#_Toc479172304)

[04值类型和引用类型.avi 84](#_Toc479172305)

[05值类型和引用类型作参数.avi 86](#_Toc479172306)

[06面试题.avi 87](#_Toc479172307)

[07数组.avi 89](#_Toc479172308)

[08数组的检测.avi 90](#_Toc479172309)

[09数组的增删方法.avi 91](#_Toc479172310)

[10数组的join方法.avi 92](#_Toc479172311)

[11数组的reverse方法.avi 93](#_Toc479172312)

[12数组的迭代方法.avi 94](#_Toc479172313)

[13数组的位置方法.avi 95](#_Toc479172314)

[14寻找每个a出现的位置.avi 96](#_Toc479172315)

[15统计元素的出现次数.avi 97](#_Toc479172316)

[16数组的其他方法1.avi 98](#_Toc479172317)

[17数组的其他方法2.avi 100](#_Toc479172318)

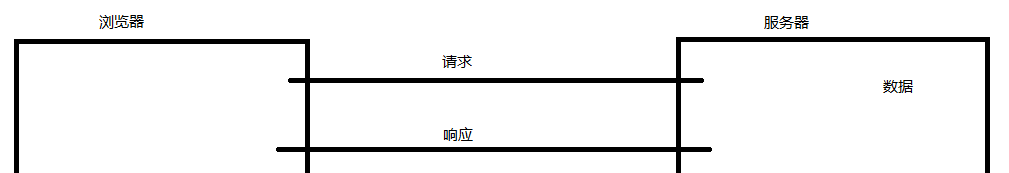
day01

01阶段概述和JS简介.avi

学习目标:

1. 掌握JavaScript(以下简称JS)的基本语法
2. 培养一定的编程思想

为什么要有JS呢?



**JS最初的目的是用来对客户端的输入信息预检测**

由于以前的网速慢,如果一旦用户输入的信息格式错误,那来回太消耗时间和资源了

所以可以在客户端将一些格式给规范好

比如:

长度应为6~18个字符

用户名必须以英文字母开头

该用户名已被注册

如果不满足以上条件,直接不发送给服务器

**JS的发展**

技术在发展,时代在进步

JS慢慢的在进化

第一阶段:

在JS的发展过程中,有一段时间一度被用于小广告,当然这个阶段很快就过去了



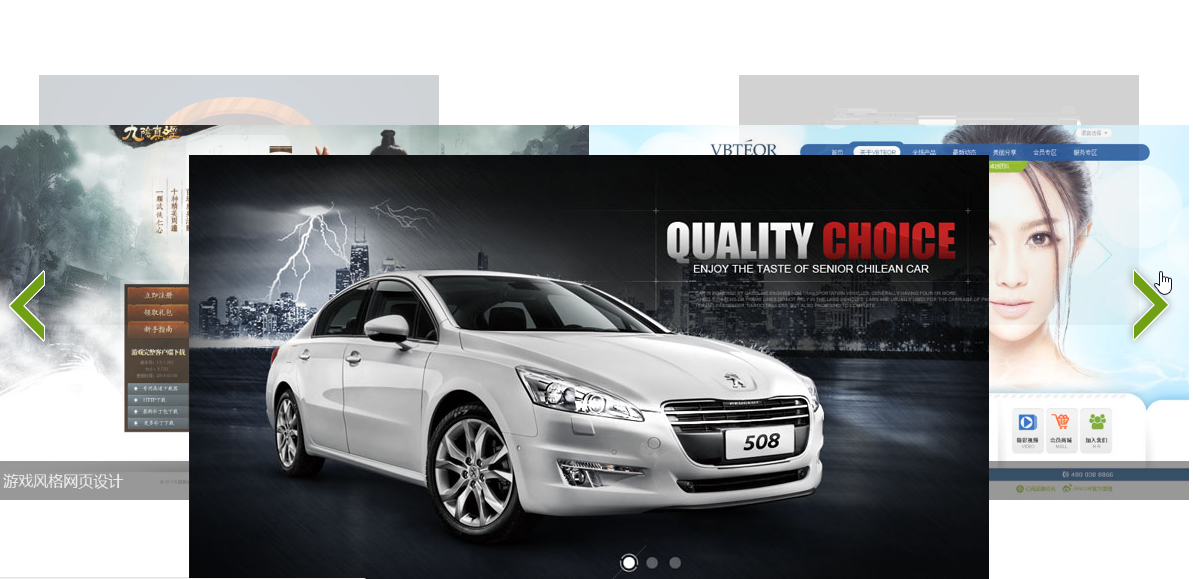
第二阶段:

Google将JS用来做异步框架,实现搜索框实时提示



第三阶段:

人们都是视觉动物,都会追求更美好的界面,所以在这个阶段开始,JS被大量用于做一些界面特效



第四阶段:

人类的本性是堕落的,是慵懒的,特别是世界上最懒的一个人种:**程序猿**

他们认为JS仅仅只能做网页前端一点都不酷,有一颗一统天下的雄心

所以在这个阶段,JS可以做一些移动端的应用了,也就是**WebApp**

第五阶段:

人类的野心也是很强大的,咱们发现JS全都是在前端技术领域非常牛逼,但是后台依旧要依赖服务器应用,所以这时候又出现了一个很厉害的技术:**NodeJS**

最后阶段:

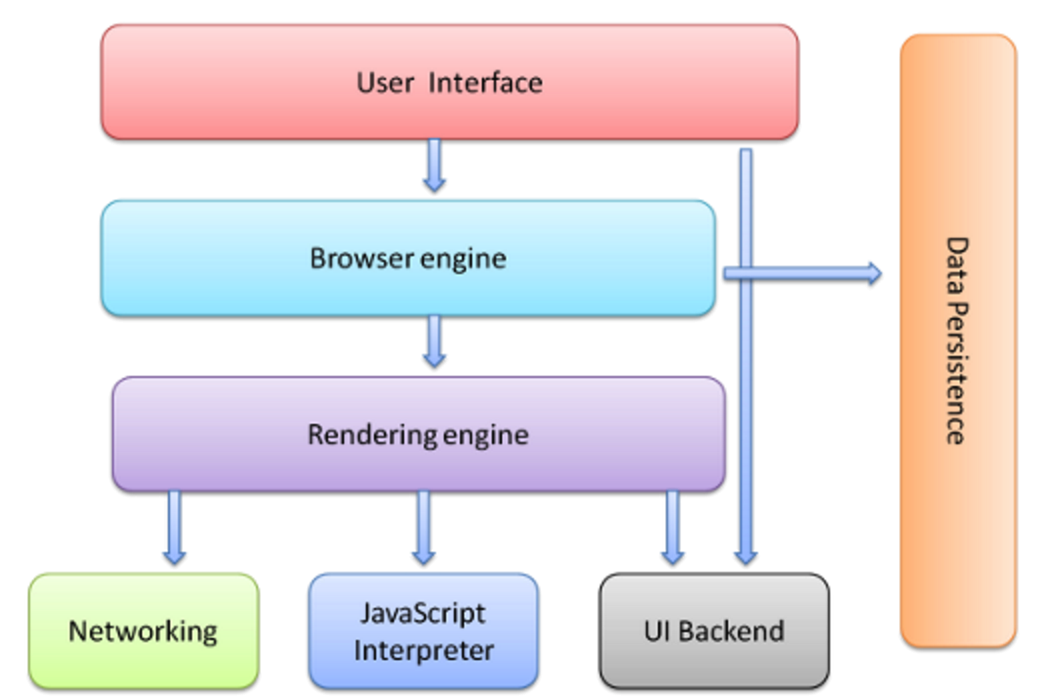
JS(PHP)已经是世界上最好的语言了...

这里推荐两个网站给大家:

<http://codecombat.cn/> 编程游戏

<http://www.webhek.com/> HTML+CSS+JS特效

02浏览器原理简介.avi



用户界面: 位于最上层,也就是我们用户看到的页面

浏览器引擎: 用户界面显示的东西都是由浏览器引擎提供的,浏览器引擎可以对下层的渲染引擎进行操作,提取数据显示给用户界面.还会用到数据持久化,说人话也就是存储一些数据到本地

渲染引擎: 用于解析代码,提供数据给上层浏览器引擎,渲染引擎需要依赖网络模块,JS的解析器,UI后端(组合框,弹框等)

通过以上三层,我们用户才能看到最终的页面,三层都是各司其职的

03基本输出语句.avi

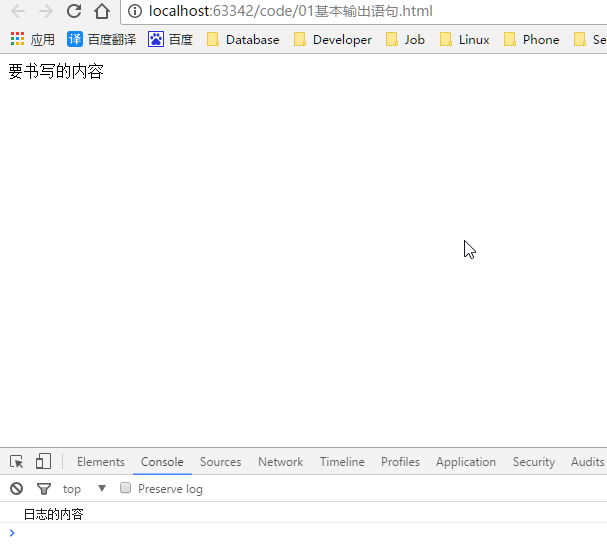
基本输出方式有几种:

第一种为控制台输出,需要通过开发者调试工具来查看

第二种为直接在页面上写文字

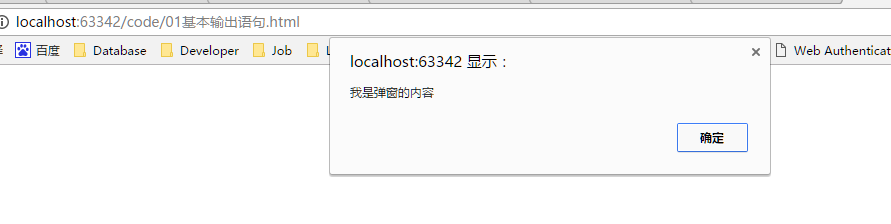
<**script**>  
 *//在控制台打印输出日志* **console**.log(**"日志的内容"**);  
 *//在文档中写内容* **document**.write(**"要书写的内容"**);  
</**script**>

效果如下:



第三种:弹窗

<**script**>**window**.alert(**"我是弹窗的内容"**); //window.可以省略  
</**script**>



04代码注释.avi

为什么需要注释呢?

代码写多了之后会越来越复杂,所以这时候很多代码都需要一些说明解释,就可以通过注释来进行标注

还有一些代码不需要执行,但是又想留着的内容

**注释的分类**

单行注释: ctrl+/

<**script**>  
 *//单行注释*</**script**>

多行注释: ctrl+shift+/

**多行注释不能嵌套使用!**

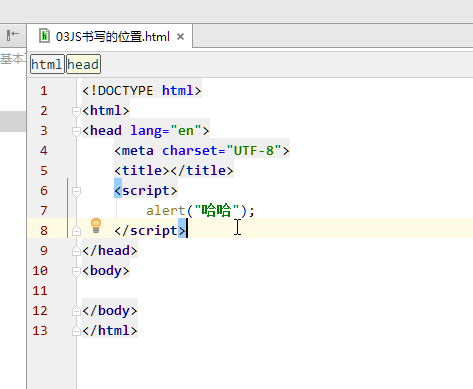
<**script**>  
 *//单行注释  
 /\*  
 多行注释  
 多行注释  
 多行注释  
 多行注释  
 多行注释  
 \*/*</**script**>

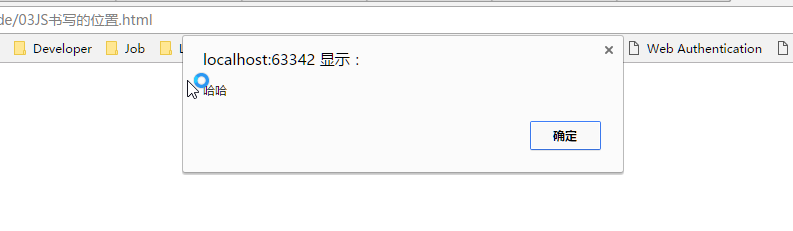
文档注释:

<**script**>  
 */\*\*  
 \*   
 \*/* **function** *test* () {  
  
 }  
</**script**>

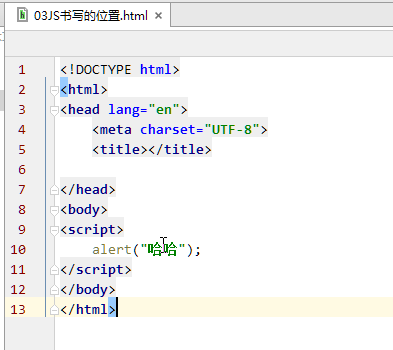
05JS的书写位置.avi

1. JS可以写在head标签中

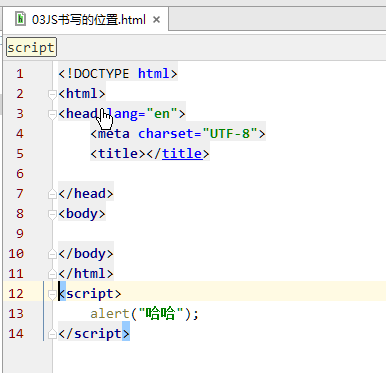


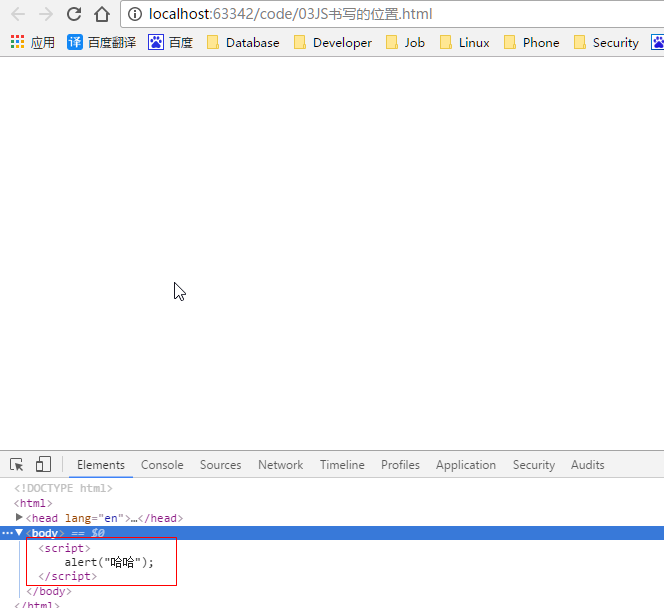


1. 也可以写在body标签中



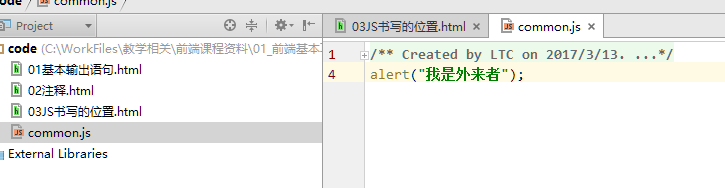
1. 也可以写在html标签外部(这样书写是不规范的,浏览器会帮我们挪到body标签中)





1. 引用外部的JS文件

新建一个js文件



有两种方式引入外部的js

第一种: 和css引用一样

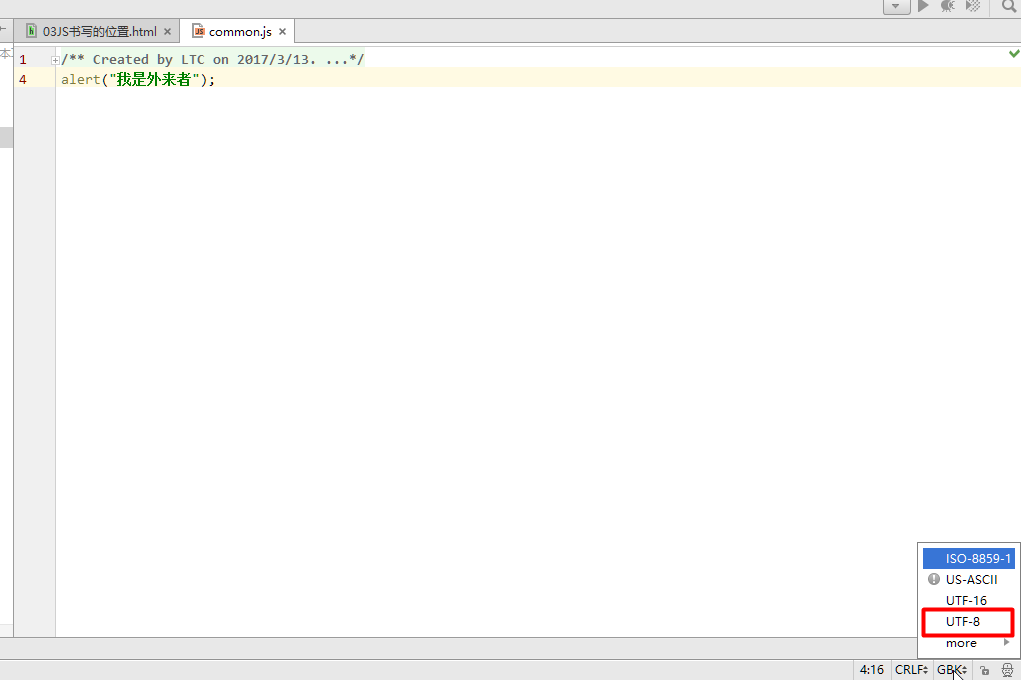
第二种: 在script标签中加入src属性



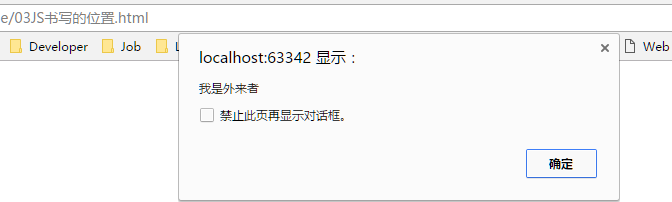
同样弹出了,但是是乱码,这是为何呢?



需要将外部js文件右下角文件编码格式改为utf-8



然后重写一下中文部分即可



06script标签的属性.avi

**记住src属性适用于引用文件路径的即可**

其他属性都是了解内容,不需要掌握



07变量的命名.avi

学习变量的命名之前要搞明白一个问题

**为什么要用变量?**

为了方便我们表示一个数据,后期控制数据跟方便,也等于是给数据起了个名字

那既然要起名字,一定会有一些**规范**和**规则**

规则: 必须遵守的

由字母、数字、下划线、$ 组成且**开头不能是数字**

不能是关键字和保留字

区分大小写

关键字和保留字有很多,不需要强行去记忆,因为开发工具webstorm会提示我们报错的,写多了自然就记住了

规范: 强烈建议遵守的

变量的名称要有实际意义 **见名知意**

变量的命名遵守驼峰命名法，首字母小写,第二个单词的首字母大写

例如：userName



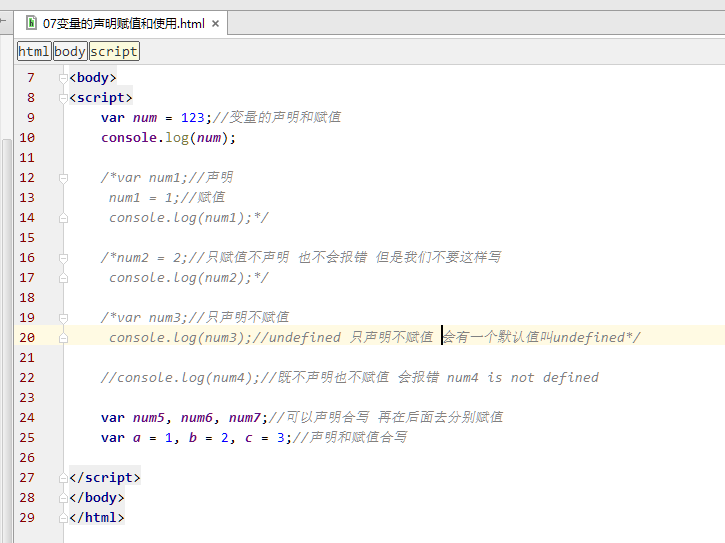
08变量的声明赋值和使用.avi

变量的声明和赋值有多种方式

我们平时用的最多的是声明+赋值

**var**是关键字,表示num是一个变量,123是这个变量的值

**var *num*** = 123;



09变量在内存中的存储.avi

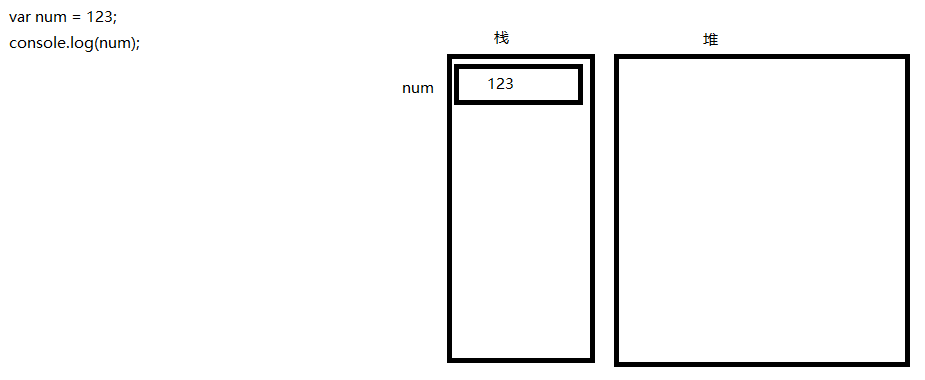
计算机中有两片重要的内存区域,栈和堆

当执行var num = 123;这句代码时

第一步会在栈内存区域开辟空间,将123放入该空间,该空间会具有一个地址值

第二步将这个地址值赋值给一个叫num的变量(这一步也叫num指向那一片内存区域)

完成声明+赋值操作



10交换变量练习.avi

交换两个变量的值其实很简单

定义一个第三方变量临时存储一个值

然后修改a为b

再将b改为第三方变量值

**不使用第三方变量来交换变量不需要掌握**

<**script**>  
 *//交换两个变量的值  
 /\*var a = 10, b = 20;  
 console.log(a);  
 console.log(b);  
 var temp = a;  
 a = b;  
 b = temp;  
 console.log(a);  
 console.log(b);\*/  
 //不使用第三个变量交换两个数值类型变量的值* **var *a*** = 10, ***b*** = 20;  
 ***a*** = ***a*** + ***b***;  
 ***b*** = ***a*** - ***b***;*//b中保存成了a原来的值* ***a*** = ***a*** - ***b***;*//a中保存了 b原来的值* **console**.log(***a***);  
 **console**.log(***b***);  
</**script**>

11数据类型.avi

JS是弱类型的语言

弱类型的意思就是,在声明变量时,不需要指定具体的数据类型

**但是要注意!弱类型的语言并不代表JS没有数据类型!**

**var *a***;*//undefined****a*** = 123;*//number****a*** = **"123"**;*//string***console**.log(***a***);*//这样虽然不会报错 但是不推荐大家这样使用*

数据类型分为两大类:

基本(值)数据类型

number

string

boolean

undefined

null

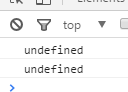
引用数据类型

暂不介绍,后期重点解释

想查看一个变量 的数据类型,可以通过typeof方法来查看

<**script**>  
 **var *a*** = 123;*//number* **var *b*** = **"123"**;*//string* **var *c*** = **true**;*//bool* **var *d*** = **false**;*//bool* **var *e***;*//undefined* **console**.log(**typeof**(***e***));  
 **console**.log(**typeof *e***);  
  
</**script**>

效果如下:



12number类型.avi

<**script**>  
 *//十进制 十个符号可以表示所有数值 0123456789* **var *a*** = 123;*//字面量 直接量  
  
 //八进制 八个符号 01234567* **var *b*** = 010;*//八进制的字面量就是在数值前面加0  
 //十六进制 十六个符号 0123456789ABCDEF* **var *c*** = 0x10;*//十六进制的字面量就是在数值前面加0x* **console**.log(***b***);*//8 控制台打印输出的时候会将其他进制的值转换成十进制* **console**.log(***c***);*//16 控制台打印输出的时候会将其他进制的值转换成十进制  
  
 //浮点型字面量* **var *n*** = 5e+4;*//xe+y x乘以10的y次方 50000* **var *m*** = 5e-2;*//xe+y x乘以10的y次方 0.05* **console**.log(***n***);  
 **console**.log(***m***);  
  
 **var *num1*** = 0.1;  
 **var *num2*** = 0.2;  
 *//浮点型的数字在计算的时候 会丢失精度* **console**.log(***num1*** + ***num2***);  
  
</**script**>

number类型大致分为整型和浮点型

整型有十进制,八进制,十六进制之分

**注意:在JS中没有二进制**

浮点型其实一般用于表示小数,科学计数法很少使用,了解即可

**注意: 在计算机中浮点型的运算都会出现精度的丢失,一定不要拿浮点型的数进行等值判断,可以做大于或小于的判断**

13数值范围.avi

因为数字是无限大的,所以计算机中也肯定无法将所有数字都表示完

所以会有一个范围:

最小的**数值**：Number.MIN\_VALUE，这个值为： 5e-324 (无限接近于0的数字)

最大的**数值**：Number.MAX\_VALUE，这个值为： 1.7976931348623157e+308

正无穷：Infinity 大于正数最大值就是该值

负无穷：-Infinity 小于负数最大值的就是该值

**数值范围为了解内容,知道在JS中可以通过Number.MAX\_VALUE获取最大值即可**

14数值检测.avi

**var *num*** = **"a"** / 2;*//NaN not a number 不是一个（有实际含义的）数值  
//console.log(num);***console**.log(**typeof *num***);*//数据类型是number***console**.log(***num*** == ***num***);*//false 自己和自己都不想等*

使用一个字母除以一个数值,得到的结果为NaN,表示不是一个数值

可以通过typeof进行判断,发现数据类型为number

最后使用==来判断,结果为false,自己和自己不相等 (**这里先暂时记住即可**)

isNaN();  
*//不是一个有意义的数值就是true  
//是有意义的数值就是false***console**.log(isNaN(**NaN**));*//true***console**.log(isNaN(***num***));*//true***console**.log(isNaN(1));*//false***console**.log(isNaN(**Infinity**));*//false 表示无穷大是有意义的*

**isNaN()用于判断是否为一个有意义的数值,返回true表示没有意义的数值,返回false表示有意义的数值**

15string类型.avi

字符串类型: 用于表示字符的类型

**var *str*** = **"aaaa"**;  
**var *str1*** = **'aaaa'**;  
*//var str2 = 'aaaa";//引号必须是对应的***var *str3*** = **'<meta charset="UTF-8">'**;  
  
*//length属性***console**.log(***str3***.**length**);  
  
*//转义字符  
//可以把普通字符转换出特殊含义***console**.log(**"aa\naa"**);  
*//可以把特殊字符转换成原有含义***console**.log(**"aa\"aa"**);

**重点:length属性(表示字符串的字符个数)**

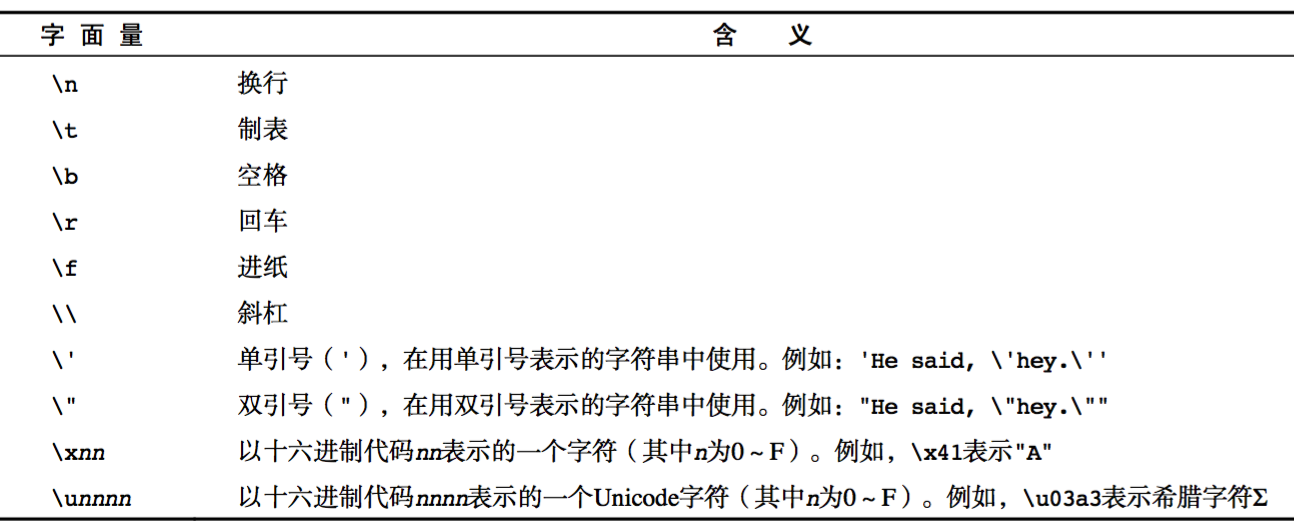
转义字符为次重点,了解几个常用的即可

\n: 换行

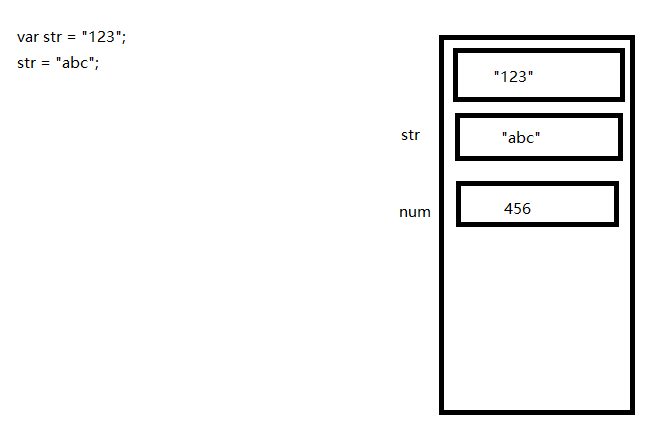
\t: 制表

\b: 空格

\r: 回车



16字符串的不可变性.avi



在内存中将一个变量赋值,然后再次修改该变量的值,那以前的值就没用了,平白无故多损失了一点点内存,如果这样拼接字符串多了就会较为消耗性能

要改变某个变量保存的字符串，首先要销毁变量与原字符串的关系，然后再用一个包含新值的字符串填充该变量

**var *str*** = **"123"**;  
*//str = "abc";  
//console.log(str);//所谓的不可变性 指的是内存中的不可变性  
  
/\*for (var i = 0; i < 1000000; i++) {  
 str += "1";//拼接字符串非常消耗性能,速度很慢  
 }\*/***var *num*** = 123;  
**for** (**var *i*** = 0; ***i*** < 1000000; ***i***++) {  
 ***num*** += 1;//没有拼接字符串 速度飞快  
}  
**console**.log(***num***);

**该内容暂时了解即可**

17字符串的拼接.avi

很多时候需要用到将两个字符串拼接起来使用,可以通过 “+” 相连

变量拼接中,只要有字符串类型,就会自动转为字符串类型

**注意: 字符串拼接只有”+”拼接字符串类型的数据才是,其他符号均为算术运算符**

**var *a*** = **"100"**;  
**var *b*** = 100;  
**var *c*** = ***a*** + ***b***;  
**console**.log(***c***);*//200 100100 结果是100100  
//数值相加 字符相连 只要有字符串就表示字符串的相连***var *d*** = ***a*** - ***b***;  
**console**.log(***d***);

如果要拼接变量可以通过以下方式来拼接

*//字符串拼接的小技巧***var *minute*** = 53;  
**var *str*** = **"现在时间是53分"**;  
**var *str*** = **"现在时间是"** + **"53分"**;  
**var *str*** = **"现在时间是"** + **"53"** + **"分"**;  
**var *str*** = **"现在时间是"** + ***minute*** + **"分"**;  
**var *str*** = **"现在时间是"** + ***minute*** + **"分"**;*//逗逗加加*

18boolean类型.avi

boolean类型取值只有true和false严格区分大小写 真和假

**var *bool1*** = **true**;  
**var *bool2*** = **false**;  
*//var bool3 = True;//严格区分大小写  
//console.log(bool1);*

但是在js中,所有数据类型都有对应的boolean值

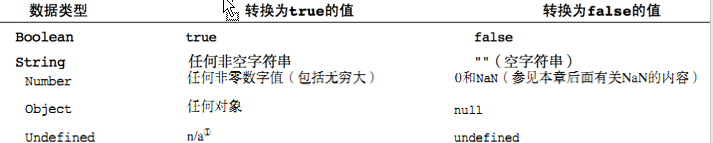
啥意思呢?

比如在number类型中,0和NaN表示false,其他值表示true

在string类型中,””(空字符串)表示false,其他表示true

Object类型中,null表示false,其他都为true

undefined类型中,undefined为false,n/a②为true



怎样将其他类型转为boolean类型呢?

可以使用Boolean()方法:

*//字符串中的"" 数值型中的 0和NaN 布尔型false null undefined --- 转换成false***console**.log(Boolean(**""**));  
**console**.log(Boolean(0));  
**console**.log(Boolean(**NaN**));  
**console**.log(Boolean(**false**));  
**console**.log(Boolean(**null**));  
**console**.log(Boolean(**undefined**));  
**console**.log(Boolean(**"0"**));*//true***console**.log(Boolean(**"false"**));*//true*

还有一种情况会出现类型转换,就是流程控制语句的条件表达式中会临时转换

如下:

*//在流程控制语句中 其它类型的数据会被临时转换成布尔值***var *message*** = **"hello"**;  
**if** (***message***) {*//true* alert(***message*** + **" world"**);  
}

19undefined类型.avi

如果变量没赋值,类型就是undefined

undefined对应的boolean值为false

以下会弹出无值的对话框

**var *message***;  
**if** (***message***) {  
 alert(**"有值"**);  
} **else** {  
 alert(**"无值"**);  
}

20其他类型值转string.avi

数据类型转换必须要掌握的三点:

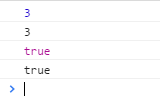
1. 转换成string类型
2. 转换成number类型
3. 转换成boolean类型

先学习将其他类型转为string

将number和boolean类型转为string可以通过**toString()方法**:

**var *num*** = 3;  
**var** bool = **true**;  
**console**.log(***num***);  
**console**.log(***num***.toString());  
**console**.log(bool);  
**console**.log(bool.toString());

效果如下: Chrome浏览器的控制台很智能,会区分string/number/boolean类型的颜色,可以明显看到他们的颜色不同



以上两种数据类型以及一般情况都可以通过toString()方法来转为string类型,**但是有些特殊情况,比如说null和undefined没有toString()方法:**

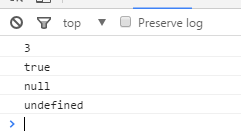
*//null和undefined没有toString()***console**.log(**null**.toString());  
**console**.log(**undefined**.toString());



接下来学习另一种方法,**可以将任意类型转为string,最通用的一种方式String()方法**:

*//String() 什么类型都可以转成字符串***console**.log(String(***num***));  
**console**.log(String(***bool***));  
**console**.log(String(**null**));  
**console**.log(String(**undefined**));

效果如下: 没有其他颜色表示为string类型



以上方式都是**显示的调用一些方法来实现数据类型转换**的

还有一些情况为**隐式转换数据类型**

何为隐式呢?

**就是没有明确调用转换数据类型的方法,系统自动转换的方式**

其实就是字符串拼接,系统会自动将其他类型转为string

**注意: 只要用”+”相连时有字符串就会进行字符串拼接**

*//3.隐式转换***var *a*** = 123;  
**var *a*** = **true**;  
**var *str*** = ***a*** + **""**;  
**console**.log(***str***);

21其他类型值转number.avi

可以通过Number()方法进行转换

**需要注意,一旦有不符合数值规范的字符串,那就会返回NaN**

*//Number()* **console**.log(Number(**true**));*//1* **console**.log(Number(**false**));*//0* **console**.log(Number(**null**));*//0* **console**.log(Number(**undefined**));*//NaN* **console**.log(Number(123));*//123* **console**.log(Number(**"123"**));*//123* **console**.log(Number(**"123px"**));*//NaN 在转换字符串的时候 只要有不认识的 就返回NaN*

如果想仅仅捕获一个字符串中的数值而舍弃多余的字符

可以使用parseInt()和parseFloat()方法来完成

**parseInt是解析为整数,parseFloat()是解析为小数**

*//parseInt() parseFloat()***console**.log(parseInt(**"123px"**));*//123 把认识的取出来 其他的忽略***console**.log(parseInt(**"123.12px"**));*//123 不认识小数 所以后面的也会忽略***console**.log(parseInt(**"123.92px"**));*//123 不认识小数 所以后面的也会忽略***console**.log(parseFloat(**"123.92px"**));*//123.92 认识小数*

以上都是使用方法进行显示转换

同样,**转为number类型也有隐式转换的情况:**

*//隐式转换***var** str = **"123"**;  
**var *num*** = str - 0;  
**var *num*** = str + 0;*//+是字符串的拼接 所以不行***var *num*** = str \* 1;  
**var *num*** = str / 1;  
**var *num*** = +str;  
**console**.log(***num***);

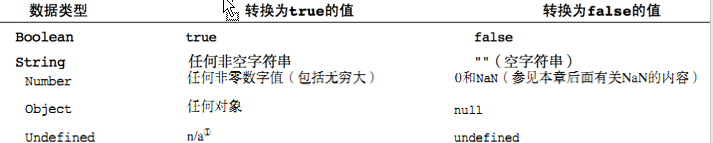
以上,使用任何算术运算符都可以进行隐式转换,但除了+号以外,因为一旦”+”运算中出现了字符串,那就会进行字符串拼接,而变为string类型

22其他类型值转boolean.avi

同上面两种类型转换,都有一个通用的方法,这里就是**Boolean()方法**

也有一种隐式转换的情况: 在流程控制语句中作为条件表达式时会转为boolean类型

在上面讲解boolean类型时已经介绍过了,复习一下即可



隐式转换小技巧:

**var *str*** = **"123"**;  
**var *bool*** = !!***str***;  
**console**.log(***bool***);

**加入一个!会进行逻辑运算,也就会触发隐式转换**

使用两个!的原因是因为:

str本来不是空字符串,所以结果应该为true

但是一个!表示取反,也就是会得到false的结果

再加一个!的目的是为了保证原结果不变的情况下进行数据转换

23算术运算符.avi

在数学中除数是不能为0的,但是在JS中,除数为0得出的结果为无限大

这是因为除数越接近0,得到的结果就越大

比如:

1/1=1

1/0.1=10

1/0.01=100

1/0.001=1000

所以当除数为0的时候结果为无限大

同样0/0没有任何意义,所以得出的结果是NaN非数字

%就是取余数

任何数 % 0都是没有意义的,所以是NaN非数字

**console**.log(1 / 0);*//Infinity***console**.log(0 / 0);*//NaN***console**.log(0 / 5);*//0除以任何数都是0***console**.log(5 % 2);*//1***console**.log(5 % 0);*//NaN***console**.log(5 % 6);*//和比自己大的数字取余 结果是自己*

24一元运算符.avi

自增一和自减一运算

**var *a*** = 1;  
**console**.log(***a***++);*//1 先参与运算 再自增***console**.log(***a***);*//2***var *b*** = 1;  
**console**.log(++***b***);*//2 先自增 在参与运算***console**.log(***b***);*//2*

**小练习: 理解了这个练习就完全掌握该知识点了,一般情况不会这么变态**

*// 3 2 + 3***var *a*** = 1; **var *b*** = ++***a*** + ++***a***; **console**.log(***b***);*//5  
// 3 1 + 3***var *a*** = 1; **var *b*** = ***a***++ + ++***a***; **console**.log(***b***);*//4  
// 3 1 + 2***var *a*** = 1; **var *b*** = ***a***++ + ***a***++; **console**.log(***b***);*//3  
// 3 2 + 2***var *a*** = 1; **var *b*** = ++***a*** + ***a***++; **console**.log(***b***);*//4*

25逻辑运算符1.avi

逻辑运算符最常用的有两个:

与 &

有0则0

或 |

有1则1

简单的与运算:

*//&& 必须两个都是true才是true***console**.log(**true** && **true**);*//true***console**.log(**false** && **true**);*//false***console**.log(**true** && **false**);*//false***console**.log(**false** && **false**);*//false*

简单的或运算:

**console**.log(**true** || **true**);*//true***console**.log(**false** || **true**);*//true***console**.log(**true** || **false**);*//true***console**.log(**false** || **false**);*//false*

**短路运算:**

**返回对表达式结果起到决定性作用的值**

比如短路与运算:

与运算 true && true 只看前面的一个true并不能决定整个表达式最终的结果,所以还会看后面的true

与运算false && true 只看前面的一个false就知道这个与运算一定是false了,所以不会执行后面的true,因为根据与运算的特性有0则0

console.log(**"前面"** && **"后面"**);*//"后面"*console.log(**""** && **"后面"**);*//""*console.log(**"前面"** && **""**);*//""*console.log(**null** && **""**);*//null*

26逻辑运算符2.avi

短路或运算:

原理同短路与运算,只是与和或的特性不同

比如:

或运算true || true 只看前面的一个true就知道这个或运算的结果了,所以不会执行后面的true

*// || 只要有true就是true***console**.log(**"前面"** || **"后面"**);*//"前面"***console**.log(**""** || **"后面"**);*//后面***console**.log(**"前面"** || **""**);*//前面***console**.log(**""** || undefined);*//undefined*

**注意: 现在不太理解短路运算没关系,到后面实际应用需要使用到时自然就掌握了**

27其他运算符.avi

剩下的一些运算符比较简单,相对于与和或运算也不太常用

**非运算: 取反**

如果操作数是对象，返回false

如果操作数是空字符串，返回true

如果操作数是非空字符串，返回false

如果操作数是0，返回true

如果操作数是任意非0值，返回false

如果操作数是null,undefined,NaN，返回true

上面的规则可以简化成

本身为false的值：false、""、0和NaN、null、undefined

前面加个!非运算,就会取反

**比较运算**

< <= >= >

== != 相等和不相等

=== !== 全等和不全等

相等只比较值，全等比较的是值和类型

例如：

var result = "55" == 55; //true

var result = "55" === 55; //false 值相等，类型不相等

var result = 55 === 55; //true

**赋值运算**

= += -= \*= /= %=

例如：

var num = 0;

num += 5;//相当于 num = num + 5;

**运算符的优先级**

优先级从高到底

() 优先级最高

一元运算符 ++ -- !

算数运算符 先\* / % 后 + -

关系运算符 > >= < <=

相等运算符 == != === !==

逻辑运算符 先&& 后||

**记住一点: 不知道优先级时就加括号!**

28练习.avi

拆解分析即可: 遇到括号先算括号 一级一级的计算即可

*//练习1  
// true  
// true && true  
// (false || true) && !(false&&true )***var *bool*** = ((4 >= 6) || (**"人"** != **"狗"**)) && !(((12 \* 2) == 144) && **true**);  
*//console.log(bool);  
//练习2***var *num*** = 10;  
*// true  
// true && true***if** (5 == ***num*** / 2 && (2 + 2 \* ***num***).toString() === **"22"**) {  
 **console**.log(**true**);  
}

29选择语句if.avi

流程控制语句: 可以用于控制程序的执行流程

当满足条件时执行某一段代码

*//示例：是否年满18岁***var *age*** = 12;  
**if** (***age*** >= 18) {  
 **console**.log(**"可以访问"**);  
 }

当age大于等于18时就打印可以访问

现在不够完善,可以继续加一个else来判断

**if** (***age*** >= 18) {  
 **console**.log(**"可以访问"**);  
 } **else** {  
 **console**.log(**"不可以"**);  
 }

else表示不满足条件时执行的代码

如果其中有多种情况可以加入elseif来进行更细化的判断

**if** (***age*** >= 18) {  
 **console**.log(**"可以访问"**);  
} **else if** (***age*** >= 16) {  
 **console**.log(**"差不多了"**);  
} **else** {  
 **console**.log(**"太小了"**);  
}

elseif表示当if不满足时继续判断

练习:

**素质教育（把分数变成ABCDE）**

例如: 当分数高于90为A,高于80为B,高于70为C,高于60为D,低于60为E

**今天星期几**

var week= new Date().getDay();

**从两个数中找最大值**

30选择语句if练习.avi

当分数高于90时打印A

否则如果当分数高于80时打印B

否则如果当分数高于70时打印C

否则如果当分数高于60时打印D

其他情况就打印E

*//素质教育（把分数变成ABCDE）***var *score*** = 52;  
 **if** (***score*** >= 90) {  
 **console**.log(**"A"**);  
 } **else if** (***score*** >= 80) {  
 **console**.log(**"B"**);  
 } **else if** (***score*** >= 70) {  
 **console**.log(**"C"**);  
 } **else if** (***score*** >= 60) {  
 **console**.log(**"D"**);  
 } **else** {  
 **console**.log(**"E"**);  
 }  
原理同上  
*//今天星期几***var *week*** = **new** Date().getDay();*//获取今天的星期 0是周日 1-6周一到周六***if** (***week*** === 0) {  
 **console**.log(**"周日"**);  
 } **else if** (***week*** === 1) {  
 **console**.log(**"周一"**);  
 } **else if** (***week*** === 2) {  
 **console**.log(**"周二"**);  
 } **else if** (***week*** === 3) {  
 **console**.log(**"周三"**);  
 } **else if** (***week*** === 4) {  
 **console**.log(**"周四"**);  
 } **else if** (***week*** === 5) {  
 **console**.log(**"周五"**);  
 } **else if** (***week*** === 6) {  
 **console**.log(**"周六"**);  
 }  
使用第三方变量存储最大值,判断a和b的大小,谁大存储谁,最后打印即可  
*//从两个数中找最大值***var *a*** = 10;  
 **var *b*** = 20;  
 **var *max***;  
 **if** (***a*** > ***b***) {  
 ***max*** = ***a***;  
 }**else**{  
 ***max*** = ***b***;  
 }  
 **console**.log(***max***);

31三元运算符.avi

就是对简单的if语法的简化

格式如下:

条件表达式 ? 结果为true执行的代码 : 结果为false执行的代码;

**var *sex*** = 1;

**if** (***sex*** === 1) {  
 **console**.log(**"男"**);  
 } **else** {  
 **console**.log(**"女"**);  
 }

***sex*** === 1 ? **console**.log(**"男"**) : **console**.log(**"女"**);  
**console**.log(***sex*** === 1 ? **"男"** : **"女"**);

以上代码可以发现使用if会很繁琐

使用三元短短的一行即可解决

练习:

是否年满18岁

从两个数中找最大值

从三个数中找最大值

32三元练习.avi

*//是否年满18岁***var *age*** = 20;  
**if** (***age*** >= 18) {  
 **console**.log(**"可以访问"**);  
} **else** {  
 **console**.log(**"走开"**);  
}  
*//age >= 18 ? console.log("可以") : console.log("走开");***console**.log(***age*** >= 18 ? **"可以"** : **"走开"**);  
*//从两个数中找最大值***var *a*** = 10;  
**var *b*** = 20;  
**var *max*** = ***a*** > ***b*** ? ***a*** : ***b***;  
**console**.log(***max***);  
*//从三个数中找最大值***var *a*** = 10;  
**var *b*** = 20;  
**var *c*** = 30;  
**var *max*** = (***a*** > ***b*** ? ***a*** : ***b***) > ***c*** ? (***a*** > ***b*** ? ***a*** : ***b***) : ***c***;  
**console**.log(***max***);

33选择语句switch.avi

作用和if相似,结构比if清晰,适合用于做等值判断

**注意:**

1. **switch语句做的是全等比较,也就是说条件中的值必须和case中的值类型相同,否则无法比较**
2. **case之间要加break;否则就算条件满足了也会继续执行下一个case中的代码,这种现象被称为switch穿透**

语法：

switch (条件) {

case value:

statement

break; // break 关键字会导致代码执行流跳出 switch 语句

case value:

statement

break;

default:

statement

}

代码如下:

*//var a = "10";***var *a*** = 10;  
 **switch** (***a***) {  
 **case** 10:  
 **console**.log(**"10"**);  
 **case** 20:  
 **console**.log(**"20"**);  
 **case** 30:  
 **console**.log(**"30"**);  
 **default** :  
 **console**.log(**"haha"**);  
 }  
*//10  
//10 20 30 haha  
//haha 注意 switch做的是全等比较***var *a*** = **"a"**;  
**switch** (***a***) {  
 **case "a"**:  
 **console**.log(**"a"**);  
 **case "b"**:  
 **console**.log(**"b"**);  
 **default** :  
 **console**.log(**"default"**);  
}  
*// a b default 注意如果没有break switch中的语句会一直向下执行*

练习:

今天星期几

var week= new Date().getDay();

素质教育（把分数变成ABCDE）

千万不要写100个case哟

day02

01选择语句switch练习.avi

获取今天的日期,进行switch判断并打印即可完成

*//今天星期几***var *week*** = **new** Date().getDay();  
**switch** (***week***) {  
 **case** 0:  
 **console**.log(**"周日"**);  
 **break**;  
 **case** 1:  
 **console**.log(**"周1"**);  
 **break**;  
 **case** 2:  
 **console**.log(**"周2"**);  
 **break**;  
 **case** 3:  
 **console**.log(**"周3"**);  
 **break**;  
 **case** 4:  
 **console**.log(**"周4"**);  
 **break**;  
 **case** 5:  
 **console**.log(**"周5"**);  
 **break**;  
 **case** 6:  
 **console**.log(**"周6"**);  
 **break**;  
}

由于switch只能做等值判断

所以要判断这种多段数值范围区域,可以通过除以10之后取整来进行判断

*//素质教育（把分数变成ABCDE）  
//千万不要写100个case哟***var *score*** = 100;  
*//ABCDE 90 80 70 60****score*** = parseInt(***score*** / 10);*//9.3 9***switch** (***score***) {  
 **case** 10:  
 **console**.log(**"A"**);  
 **break**;  
 **case** 9:  
 **console**.log(**"A"**);  
 **break**;  
 **case** 8:  
 **console**.log(**"B"**);  
 **break**;  
 **case** 7:  
 **console**.log(**"C"**);  
 **break**;  
 **case** 6:  
 **console**.log(**"D"**);  
 **break**;  
 **default** :  
 **console**.log(**"E"**);  
}

02循环语句for.avi

当需要重复执行一些任务的时候,拷贝粘贴显然就太麻烦了

可以使用for循环来完成,语法如下:

for(初始化变量 ; 条件表达式 ; 循环后表达式) {

循环执行的代码...

}

*//重复打印500个0***console**.log(0);  
 **console**.log(0);  
 **console**.log(0);  
 **console**.log(0);  
 **console**.log(0);  
*//循环  
// 初始化语句 ; 控制表达式 ; 循环后表达式***for** (**var *i*** = 1; ***i*** <= 5; ***i***++) {  
 **console**.log(0);  
}  
  
*//作用等同于上面的代码***var *i*** = 1  
 **for** (; ***i*** <= 5;) {  
 **console**.log(0);  
 ***i***++  
 }  
  
*//打印1到5***for** (**var *i*** = 1; ***i*** <= 5; ***i***++) {  
 **console**.log(***i***);  
}

03for循环的简单练习.avi

1. 打印输出5次100
2. 打印输出1-100这些数字（包括100）
3. 打印输出1-100的和、平均值
4. 打印输出1到100之间的数的乘积（包括100）
5. 计算1到100之间所有不能被3整除的整数的和
6. 计算1到1000之间所有不能被7整除的整数的和
7. 求1到100之间所有偶数(even)的和
8. 求1到100之间所有奇数(odd)的和
9. 同时将1到100之间所有奇数和和偶数和分别求出

*//打印输出5次100***for** (**var *i*** = 1; ***i*** <= 5; ***i***++) {  
 **console**.log(100);  
 }

*//打印输出1-100这些数字（包括100）***for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **console**.log(***i***);  
 }

*//打印输出1-100的和、平均值***var *sum*** = 0;  
 **for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 *//sum = sum + i;* ***sum*** += ***i***;  
 }  
 **console**.log(***sum***);  
 **console**.log(***sum*** / 100);

*//打印输出1到100之间的数的乘积（包括100）***var *pro*** = 1;  
 **for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 *//pro = pro \* i;* ***pro*** \*= ***i***;  
 }  
 **console**.log(***pro***);

*//计算1到100之间所有不能被3整除的整数的和***var *sum*** = 0;  
 **for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 3 !== 0) {*//不能被3整除* ***sum*** += ***i***;  
 }  
 }  
 **console**.log(***sum***);

*//计算1到1000之间所有不能被7整除的整数的和***var *sum*** = 0;  
 **for** (**var *i*** = 1; ***i*** <= 1000; ***i***++) {  
 **if** (***i*** % 7 !== 0) {  
 ***sum*** += ***i***;  
 }  
 }  
 **console**.log(***sum***);

*//求1到100之间所有偶数(even)的和***var *sum*** = 0;  
 **for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 2 === 0) {*//能被2整除就是偶数* ***sum*** += ***i***;  
 }  
 }  
 **console**.log(***sum***);

*//求1到100之间所有奇数(odd)的和***var *sum*** = 0;  
 **for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 2 !== 0) {  
 ***sum*** += ***i***;  
 }  
 }  
 **console**.log(***sum***);

*//同时将1到100之间所有奇数和和偶数和分别求出***var *sumEven*** = 0;*//偶数和***var *sumOdd*** = 0;*//奇数和***for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 2 === 0) {  
 ***sumEven*** += ***i***;*//记录偶数的和* } **else** {  
 ***sumOdd*** += ***i***;*//记录奇数和* }  
}  
**console**.log(***sumEven***);  
**console**.log(***sumOdd***);

04for循环的中等练习.avi

打印正方形

打印直角三角形

打印 9 \* 9 乘法表

*//打印正方形***for** (**var *i*** = 1; ***i*** <= 10; ***i***++) { *//外层循环一次表示打印一行* **for** (**var *j*** = 1; ***j*** <= 10; ***j***++) { *//内存循环一次表示打印一颗星* **document**.write(**"★"**);  
 }  
 **document**.write(**"<br/>"**);  
}  
*//打印直角三角形***for** (**var *i*** = 1; ***i*** <= 10; ***i***++) {  
 **for** (**var *j*** = 1; ***j*** <= ***i***; ***j***++) { *//每行打印的星星数和外层有密切的关系,第一行打印一颗,第二行打印两颗,以此类推,所以判断条件是j<=i* **document**.write(**"★"**);  
 }  
 **document**.write(**"<br/>"**);  
}  
*//打印 9 \* 9 乘法表***for** (**var *i*** = 1; ***i*** < 10; ***i***++) {  
 **for** (**var *j*** = 1; ***j*** <= ***i***; ***j***++) { *//将直角三角形换成乘法运算即可* **document**.write(***j*** + **"\*"** + ***i*** + **"="** + ***i*** \* ***j***);  
 **document**.write(**" "**);  
 }  
 **document**.write(**"<br/>"**);  
}

05for循环的困难练习.avi

本金10000元存入银行，年利率是千分之三，每过1年，将本金和利息相加作为新的本金。计算5年后，获得的本金是多少？

*//本金10000元存入银行，年利率是千分之三，每过1年，将本金和利息相加作为新的本金。计算5年后，获得的本金是多少？***var *money*** = 10000;  
**var *rate*** = 0.003;  
*//每年利息就是本金乘利率,最后加上本金作为新的本金,五年就执行五次****money*** = ***money*** + ***money*** \* ***rate***;  
***money*** = ***money*** + ***money*** \* ***rate***;  
***money*** = ***money*** + ***money*** \* ***rate***;  
***money*** = ***money*** + ***money*** \* ***rate***;  
***money*** = ***money*** + ***money*** \* ***rate***;  
  
*//多次执行可以使用for循环***for** (**var *i*** = 1; ***i*** <= 5; ***i***++) {  
 ***money*** = ***money*** + ***money*** \* ***rate***;  
}  
**console**.log(***money***);

**斐波那契数列**

兔子数列将来有更好理解的方法

有个人想知道一年之内一对兔子能繁殖多少对。于是就筑了一道围墙把一对兔子关在里面。已知一对兔子每个月可以生一对小兔子，而一对兔子从出生后第3个月起每月生一对小兔子。假如一年内没有发生死亡现象，那么，一对兔子一年内（12个月）能繁殖成多少对？

兔子数列规律为：1，1，2，3，5，8，13，21

**var *n1*** = 1;  
**var *n2*** = 1;  
**var *n3***;  
**for** (**var *i*** = 3; ***i*** <= 12; ***i***++) {  
 ***n3*** = ***n1*** + ***n2***;  
 ***n1*** = ***n2***;  
 ***n2*** = ***n3***;  
}  
**console**.log(***n3***);

06break和continue.avi

break

立即退出**整个**循环，执行循环后的语句

continue

立即退出**本次**循环，但退出循环后会从循环的顶部继续执行该循环

练习: 了解即可,实际应用时知道break和continue的作用即可

1. 求1到100之间不能被7整除的整数的和（continue）
2. 求200-300之间所有的奇数的和(continue)
3. 求200-300之间第一个能被7整除的数(break)
4. 求1到100之间所有不能被3整除的整数的第一个大于2000的和(continue和break)

*//求1到100之间不能被7整除的整数的和（continue）***var *sum*** = 0;  
 **for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 7 === 0) {*//能被7整除* **continue**;  
 }  
 ***sum*** += ***i***;  
 }  
 **console**.log(***sum***);  
*//求200-300之间所有的奇数的和(continue)***var *sum*** = 0;  
 **for** (**var *i*** = 200; ***i*** <= 300; ***i***++) {  
 **if** (***i*** % 2 === 0) {  
 **continue**;  
 }  
 ***sum*** += ***i***;  
 }  
 **console**.log(***sum***);  
*//求200-300之间第一个能被7整除的数(break)***for** (**var *i*** = 200; ***i*** <= 300; ***i***++) {  
 **if** (***i*** % 7 === 0) {  
 **console**.log(***i***);  
 **break**;  
 }  
 }  
*//求1到100之间所有不能被3整除的整数的第一个大于2000的和(continue和break)***var *sum*** = 0;  
**for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 3 === 0) {  
 **continue**;  
 }  
 ***sum*** += ***i***;  
 **if** (***sum*** > 2000) {  
 **console**.log(***sum***);  
 **break**;  
 }  
}

07循环语句while.avi

语法:

while(条件表达式) {

循环执行的代码;

}

while执行顺序

1. 先执行小括号中条件表达式

2. 再执行循环体中的代码

do {

循环执行的代码;

} while(条件表达式);

dowhile执行顺序

1. 先执行循环体中的代码一次
2. 再判断小括号中的条件表达式是否满足下一次的执行条件,如果满足继续执行循环体中的代码

while和dowhile的区别: while必须先判断,当条件表达式返回结果为true时再执行循环体的内容,但是dowhile不论怎样都会执行一次循环体的内容再进行条件表达式的判断决定是否再次循环执行

*//1到10之间整数的和（while）***var *sum*** = 0;  
 **var *i*** = 1;  
 **while** (***i*** <= 10) {  
 ***sum*** += ***i***;  
 ***i***++;  
 }  
 **console**.log(***sum***);  
*//1到10之间整数的和（dowhile）***var *sum*** = 0;  
 **var *i*** = 1;  
 **do** {  
 ***sum*** += ***i***;  
 ***i***++;  
 } **while** (***i*** <= 10);  
 **console**.log(***sum***);  
*//区别 在于 dowhile 先执行再判断***do** {  
 **console**.log(**"dowhile执行了"**);  
} **while** (1 > 2);  
*//即使条件是false do中的语句也至少会执行一次***while** (1 > 2) {  
 **console**.log(**"while循环执行了"**);  
}

08循环语句while练习.avi

1. 1到100之间整数的和（while）
2. 1到100之间整数的和（dowhile）
3. 循环输出询问"你爱我吗？"选择"y/n"若输入为y则打印"我也是"若输入为n则继续询问

prompt方法可以提示用户输入内容,该方法有返回值,可以获取用户输入的内容

*//1到100之间整数的和（while）***var *sum*** = 0;  
**var *i*** = 1;  
**while** (***i*** <= 100) {  
 ***sum*** += ***i***;  
 ***i***++;  
}  
**console**.log(***sum***);  
*//1到100之间整数的和（dowhile）***var *sum*** = 0;  
**var *i*** = 1;  
**do** {  
 ***sum*** += ***i***;  
 ***i***++;  
} **while** (***i*** <= 100);  
**console**.log(***sum***);  
*//循环输出询问"你爱我吗？"选择"y/n"若输入为y则打印"我也是"若输入为n则继续询问  
//prompt提示***while** (**true**) {  
 **var *msg*** = prompt(**"你爱我么?y/n"**);  
 **if** (***msg*** === **"y"**) {  
 **console**.log(**"我也是，我也很爱我自己"**);  
 **break**;  
 }  
}

09数组的声明.avi

学习目标:

数组的声明赋值和使用

使用循环遍历数组

运用数组的length属性解决问题

为什么要学数组呢?

之前学习的数据类型，只能存储一个值

我们想存储多个值的时候可以使用数组

比如：存储班级中所有学生的分数

使用数组不但批量存储数据方便，批量处理数据更方便（有很多方法可以使用）

数组的声明方式:

构造函数:**var *arr*** = **new** Array();*//不传参数打印出来的是空数组***var *arr*** = **new** Array(1, 2, 3, 4, 5);*//传入的东西表示数组中的每一项***var *arr*** = **new** Array(5);*//如果只传入一个数字 这个数字表示数组的长度***var *arr*** = **new** Array(**"a"**, **"b"**, **"c"**);*//JS是弱类型语言 数组中可以放任何类型的数据***var *arr*** = **new** Array(**"a"**, 2, **true**, **null**, **undefined**);

字面值:

**var *arr*** = [];*//空数组***var *arr*** = [1, 2, 3, 4, 5];*//***var *arr*** = [5];*//直接量的形式 即使只有一个数字也表示数组中的一项***var *arr*** = [**"a"**, **"b"**, **"c"**];*//***var *arr*** = [**"a"**, 2, **true**, **null**, **undefined**];  
**var *arr*** = [1, 2, 3];*//最常用的*

10数组的基本属性.avi

**数组有索引,索引是从0开始的**

**数组名.length可以获取数组的长度(有多少个元素)**

给数添加元素: 数组名[索引] = 要赋的值;

**var *arr*** = [];  
**console**.log(***arr***);  
**console**.log(***arr***.**length**);*//0  
//给数组添加元素  
//数组名[索引号] = 要赋的值;****arr***[0] = 1;*//数组的索引是从0开始的****arr***[1] = 1;  
***arr***[2] = 1;  
**console**.log(***arr***);

使用(获取)数组的元素: 数组名[索引]

*//从数组中获取元素  
//数组名[索引号]***console**.log(***arr***[0]);  
**console**.log(***arr***[1]);  
***arr***.**length**;*//表示数组中元素的个数***console**.log(***arr***);  
**console**.log(***arr***.**length**);*//3*

11数组的遍历.avi

之前学习了获取数组元素的方法:

**var** arr = [0, 1, 2];

**console**.log(arr[0]);  
 **console**.log(arr[1]);  
 **console**.log(arr[2]);

但是通过上面的方法只能一个一个的获取元素,如果数组太长,获取数据太麻烦了

所以需要利用循环来获取元素,也被称为数组的遍历

*//循环 arr.length是3 最后一项的索引号是2,也就是arr.length-1***for** (**var *i*** = 0; ***i*** <= ***arr***.**length** - 1; ***i***++) {  
 **console**.log(***arr***[***i***]);  
 }  
*//数组遍历的基本格式***for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **console**.log(***arr***[***i***]);  
}  
*//反向遍历***for** (**var *i*** = ***arr***.**length** - 1; ***i*** >= 0; ***i***--) {  
 **console**.log(***arr***[***i***]);  
}

12length属性的应用.avi

现在如果需要循环的将数据放入数组

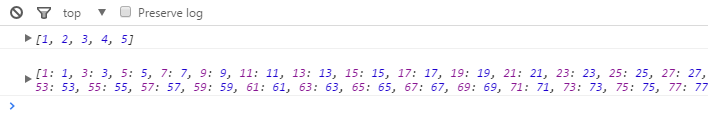
比如将0-100所有的奇数放入数组

**var *arr*** = [];  
**for** (**var** i = 1; i <= 100; i++) {  
 **if** (i % 2 !== 0) {  
 ***arr***[i] = i;  
 }  
}  
**console**.log(***arr***);

可以看到和普通的数组有些区别(上面一行是普通声明的数组)

数组前面多了索引

这是因为循环的同时i是在自增的,所以导致数组的所有元素都存入到奇数索引了,也就是说数组的索引是不连续的,这种被称为”稀疏数组”,**极力不推荐这样做!**



为了解决这个问题

可以定义一个变量来做数组的索引,添加一个元素就自增一

**var *arr*** = [];  
**var *j*** = 0;*//用来表示数组的索引***for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 2 !== 0) {  
 ***arr***[***j***] = ***i***;  
 ***j***++;  
 }  
}  
**console**.log(***arr***);



虽然可以解决这个问题,但是多声明了一个j变量也是很多余的

可以发现一些规律

当循环一次,往数组中加一个元素,数组长度也会自增一

通过这个规律可以发现,j变量可以替换为数组的长度,也就是arr.length

**var *arr*** = [];  
*//var length = 0;//根本不需要单独声明一个变量  
//空数组的length属性就是0***for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 2 !== 0) {  
 ***arr***[***arr***.**length**] = ***i***;  
 *//length++;//使用了length属性后 就不需要手动++了  
 //因为length属性是动态改变的* }  
}  
**console**.log(***arr***);

13length属性详解.avi

数组的length属性是随着数组元素增加或减少而动态变化的

**手动修改length值可以直接影响到数组的长短**

arr.**length** = 2;*//可以影响数组中元素的个数*

可以利用length属性结合循环,一次给数组追加多个元素

**for** (**var *i*** = 1; ***i*** <= 50; ***i***++) {  
 ***arr***[***arr***.**length**] = 1;*//length属性是动态改变的*}  
**console**.log(***arr***);

14数组练习1 .avi

1. 把0-100之间的所有数字，放到数组中
2. 把1-100之间的所有数字，放到数组中
3. 把1-100之间所有的奇数，放到数组中
4. 1-100之间能被3整数的数字，存到数组中

*//把0-100之间的所有数字，放到数组中***var *arr*** = [];  
**for** (**var *i*** = 0; ***i*** <= 100; ***i***++) {  
 *//arr[i] = i;* ***arr***[***arr***.**length**] = ***i***;  
}  
**console**.log(***arr***);

*//把1-100之间的所有数字，放到数组中***var *arr*** = [];  
**for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 *//arr[i] = i;* ***arr***[***arr***.**length**] = ***i***;*//保证从0开始连续追加*}

**console**.log(***arr***);  
*//把1-100之间所有的奇数，放到数组中***var *arr*** = [];  
**for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 2 !== 0) {  
 ***arr***[***arr***.**length**] = ***i***;  
 }  
}  
**console**.log(***arr***);

*//1-100之间能被3整数的数字，存到数组中***var *arr*** = [];  
**for** (**var *i*** = 1; ***i*** <= 100; ***i***++) {  
 **if** (***i*** % 3 === 0) {  
 ***arr***[***arr***.**length**] = ***i***;  
 }  
}  
**console**.log(***arr***);

15数组练习2 .avi

*var arr = [55, 3, 6, 7, 8, 3, 1];*

求一组数中的所有数的和 和 平均值

求一组数中的最大值

求一组数中的最小值及其索引

求一组数中的最大值和最小值以及所在位置

*//求一组数中的所有数的和 和 平均值***var *arr*** = [55, 3, 6, 7, 8, 3, 1];  
**var *sum*** = 0;  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 ***sum*** += ***arr***[***i***];  
}  
**console**.log(***sum***);  
**console**.log(***sum*** / ***arr***.**length**);  
*//求一组数中的最大值***var *arr*** = [55, 3, 6, 7, 8, 3, 1];  
**var *max*** = ***arr***[0];  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **if** (***max*** < ***arr***[***i***]) {  
 ***max*** = ***arr***[***i***];  
 }  
}  
**console**.log(***max***);  
*//求一组数中的最小值及其索引***var *arr*** = [55, 3, 6, 7, 8, 3, 1];  
**var *min*** = ***arr***[0];  
**var *minIndex*** = 0;  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **if** (***min*** > ***arr***[***i***]) {  
 ***min*** = ***arr***[***i***];  
 ***minIndex*** = ***i***;  
 }  
}  
**console**.log(***min***);  
**console**.log(***minIndex***);  
*//求一组数中的最大值和最小值以及所在位置***var *arr*** = [55, 3, 6, 7, 8, 3, 1];  
**var *max*** = ***arr***[0];  
**var *maxIndex*** = 0;  
**var *min*** = ***arr***[0];  
**var *minIndex*** = 0;  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **if** (***max*** < ***arr***[***i***]) {  
 ***max*** = ***arr***[***i***];  
 ***maxIndex*** = ***i***;  
 }  
 **if** (***min*** > ***arr***[***i***]) {  
 ***min*** = ***arr***[***i***];  
 ***minIndex*** = ***i***;  
 }  
}  
**console**.log(***max***);  
**console**.log(***maxIndex***);  
**console**.log(***min***);  
**console**.log(***minIndex***);

16数组练习3 .avi

将字符串数组用|或其他符号分割

求一个字符串数组中每项的长度，并把长度的数值存储到新的数组中

将数组中值为0的项去掉，将不为0的值存入一个新的数组

翻转数组（可以用一个新的数组也可以翻转原数组）

将一个数组中的值从小到大排列

*//将字符串数组用|或其他符号分割***var *arr*** = [**"a"**, **"bb"**, **"ccc"**, **"dddd"**];  
**var *str*** = ***arr***[0];  
**var *spe*** = **"-"**;  
**for** (**var *i*** = 1; ***i*** < ***arr***.**length**; ***i***++) {  
 ***str*** = ***str*** + ***spe*** + ***arr***[***i***];  
}  
**console**.log(***str***);  
*//求一个字符串数组中每项的长度，并把长度的数值存储到新的数组中***var *arr*** = [**"a"**, **"bb"**, **"ccc"**, **"dddd"**];  
**var *newArr*** = [];  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 ***newArr***[***newArr***.**length**] = ***arr***[***i***].**length**;  
}  
**console**.log(***newArr***);  
*//将数组中值为0的项去掉，将不为0的值存入一个新的数组***var *arr*** = [1, 2, 3, 4, 0, 5, 67, 58, 0, 9, 90, 0];  
**var *newArr*** = [];  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **if** (***arr***[***i***] !== 0) {  
 ***newArr***[***newArr***.**length**] = ***arr***[***i***];  
 }  
}  
**console**.log(***newArr***);  
*//翻转数组（可以用一个新的数组也可以翻转原数组）***var *arr*** = [**"a"**, **"bb"**, **"ccc"**, **"dddd"**];  
**console**.log(***arr***);  
**var *newArr*** = [];  
**for** (**var *i*** = ***arr***.**length** - 1; ***i*** >= 0; ***i***--) {  
 ***newArr***[***newArr***.**length**] = ***arr***[***i***];  
}  
**console**.log(***newArr***);

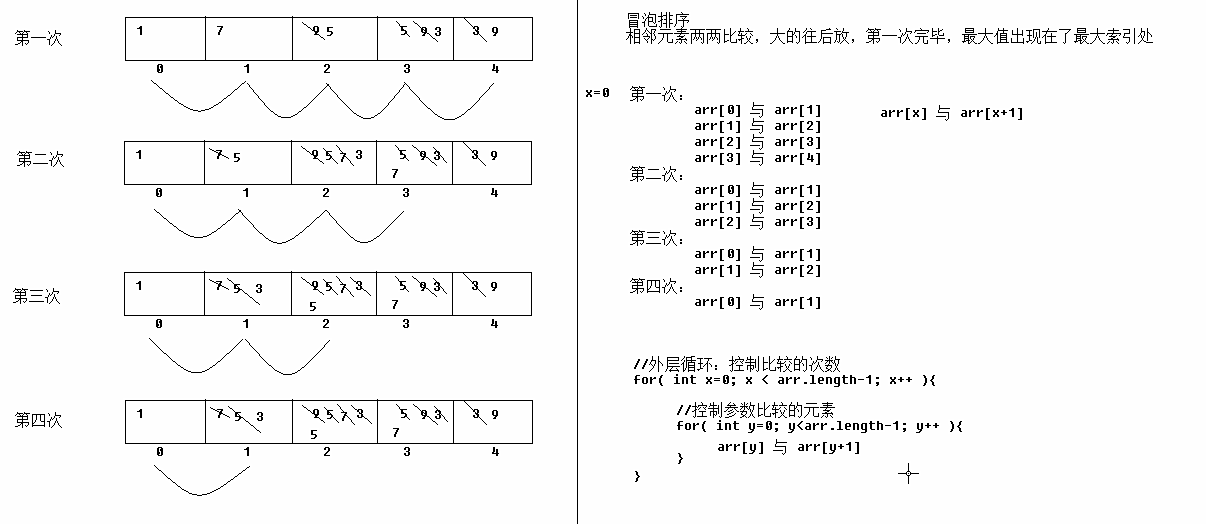
17冒泡排序1排序原理 .avi

在软件开发中,程序员必然会接触到的一个东西就是算法

这里学习排序算法中最简单的算法之一: 冒泡排序

作用:顾名思义,将数组中的元素按照从小到大的顺序排列(当然也可以从大到小)

原理:



代码如下:

**var *arr*** = [65, 97, 76, 13, 27, 49, 58];  
*//外层循环控制的是比较的趟数  
//5个元素比较4趟 arr.length个元素比较arr.length-1趟***for** (**var *i*** = 0; ***i*** < ***arr***.**length** - 1; ***i***++) {*//arr.length-1趟  
 //内层循环 控制的是两两比较的次数  
 //5个元素比较4次 arr.length个元素 比较arr.length-1次* **for** (**var *j*** = 0; ***j*** < ***arr***.**length** - 1; ***j***++) {*//arr.length-1趟* **if** (***arr***[***j***] > ***arr***[***j*** + 1]) {  
 **var *temp*** = ***arr***[***j***];  
 ***arr***[***j***] = ***arr***[***j*** + 1];  
 ***arr***[***j*** + 1] = ***temp***;  
 }  
 }  
}  
**console**.log(***arr***);

18冒泡排序2优化内层 .avi

看了上面的原理图就会发现,在之前写的代码中有一些多余的循环操作



五个元素的排序其实内层循环并不是每次都需要执行4次比较的

第一趟需要执行四次比较

第二趟需要执行三次比较

第三趟需要执行两次比较

第四趟仅需执行一次比较

所以每一趟外层循环,都会让内层循环的次数自减一

而外层循环的i是递增一的,所以可以在内存循环的条件表达式中再减去一个外层循环的i即可

***arr***.**length** - 1 - ***i***

这样一来:

第一次减0得4

第二次减1得3

第三次减2得2

第四次减3得1

perfect!

代码如下:

*//将一个数组中的值从小到大排列***var *arr*** = [65, 97, 76, 13, 27, 49, 58];  
**var *outer*** = 0*//记录外层循环的次数***var *inner*** = 0;*//记录内层循环的次数  
//外层循环控制的是比较的趟数  
//5个元素比较4趟 arr.length个元素比较arr.length-1趟***for** (**var *i*** = 0; ***i*** < ***arr***.**length** - 1; ***i***++) {*//arr.length-1趟  
 //内层循环 控制的是两两比较的次数  
 //5个元素比较4次 arr.length个元素 比较arr.length-1次* **for** (**var *j*** = 0; ***j*** < ***arr***.**length** - 1 - ***i***; ***j***++) {*//arr.length-1趟* **if** (***arr***[***j***] > ***arr***[***j*** + 1]) {  
 **var *temp*** = ***arr***[***j***];  
 ***arr***[***j***] = ***arr***[***j*** + 1];  
 ***arr***[***j*** + 1] = ***temp***;  
 }  
 ***inner***++;  
 }  
 ***outer***++;  
}  
**console**.log(***arr***);  
**console**.log(***inner***);*//36 -> 21***console**.log(***outer***);*//6*

19冒泡排序3外层内层 .avi

做到上面的优化已经让内层循环次数大大减少了,但是外层循环依旧还要执行很多次

其实有些外层循环也是多余的

比如: 数组为var arr=[1,2,3,4,5];

对该数组进行冒泡排序,外层循环必须执行length-1次,也就是4次

但其实这个数组第一次外层循环完毕后,就已经排列好了,所以后面三次都是多余的

这时候可以对外层循环也做一个优化

在进入外层循环时定义一个标记,默认值为true 表示这个数组已经排好了

在进入内层循环后,如果发现前面的元素比后面元素大,说明需要换位置,那就表示这个数组并没有排列好,所以把标记改为false

在外层循环结束前对该标记进行判断,一旦为true表示整个内层循环都没有发生元素换位,那说明这个数组已经排序完毕了,就可以直接break掉了

**大大提高运行效率!**

但是这个知识点并非重点.

代码如下:

**var *arr*** = [1, 2, 3, 4, 5];  
**var *outer*** = 0  
**var *inner*** = 0;  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length** - 1; ***i***++) {  
 *//var wocai = "我猜这次一排列的是一个已经排好了的数组";* **var *flag*** = **true**;*//假设这次排列的是一个已经排好了的数组* **for** (**var *j*** = 0; ***j*** < ***arr***.**length** - 1 - ***i***; ***j***++) {  
 **if** (***arr***[***j***] > ***arr***[***j*** + 1]) {  
 *//wocai = "猜错了";* ***flag*** = **false**;  
 **var *temp*** = ***arr***[***j***];  
 ***arr***[***j***] = ***arr***[***j*** + 1];  
 ***arr***[***j*** + 1] = ***temp***;  
 }  
 ***inner***++;  
 }  
 *//if (wocai !== "猜错了") {* **if** (***flag***) {  
 **break**;  
 }  
 ***outer***++;  
}  
**console**.log(***arr***);  
**console**.log(***inner***); *//4***console**.log(***outer***); *//1*

day03

01函数介绍.avi

学习目标:

函数的基本语法

会使用函数解决一些问题

为啥要学函数?

函数可以封装一段代码，只需定义一次，即可多次调用和执行

一般可以将一个功能封装为函数

函数的使用:

1. 声明
2. 调用

声明函数的关键字: **function**

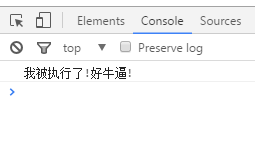
声明:

**function** *函数名*() {  
 *//函数体 要执行的代码*}

**声明的函数并不会执行**,需要手动调用才可执行

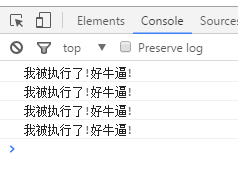
调用: 使用方法名();就可以调用了

**function** *test*() {  
 *//函数体 要执行的代码* **console**.log(**"我被执行了!好牛逼!"**);  
}  
  
*test*();



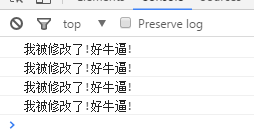
也可以多次调用执行:

**function** *test*() {  
 *//函数体 要执行的代码* **console**.log(**"我被执行了!好牛逼!"**);  
}  
  
*test*();  
*test*();  
*test*();  
*test*();



如果需要修改这个功能,只需要修改函数体即可,所有调用函数的地方执行结果都会变,非常方便:

**function** *test*() {  
 *//函数体 要执行的代码* **console**.log(**"我被修改了!好牛逼!"**);  
}  
  
*test*();  
*test*();  
*test*();  
*test*();

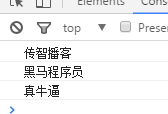


02函数的参数.avi

之前定义的方法每次调用所做的事情都是固定的,每次执行fn()方法都是答应固定的内容

这样封装函数并不灵活,比如现在想封装一个打印文字的函数,但是打印什么内容由调用函数的人来决定,可以给函数定义参数来实现

*//参数 形参 形式参数***function** *cl*(msg) {*//声明了一个参数 告诉解析器我是一个占位符* **console**.log(msg);  
}  
*//参数 实参 实际参数  
cl*(**"传智播客"**);  
*cl*(**"黑马程序员"**);  
*cl*(**"真牛逼"**);



**在函数上定义的参数只是一个占位符,被称为形式参数,简称形参**

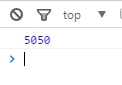
**在调用时传入的参数是具体的内容,被称为实际参数,简称实参**

小练习:

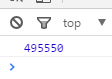
求X到Y之间所有数的和

**function** *getSum*(x, y) {  
 **var** sum = 0;  
 **for** (**var** i = x; i <= y; i++) {  
 sum += i;  
 }  
 **console**.log(sum);  
}  
  
*getSum*(1, 100);

定义方法之后要记得调用哦



*getSum*(100,1000);



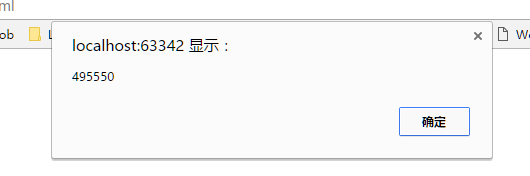
03函数的返回值.avi

刚刚的求和案例只能在控制台输出打印,因为在函数中求出和了立马就打印了,也就意味着这个函数的功能仅仅只是求出和并打印

如果想获取到和的值,继续进行后续的运算或其他操作(例如弹窗显示),那应该怎么办呢?

可以使用函数的返回值来完成:

**function** *getSum*(x, y) {  
 **var** sum = 0;  
 **for** (**var** i = x; i <= y; i++) {  
 sum += i;  
 }  
 **return** sum;  
}  
  
**var *sum*** = *getSum*(100, 1000);  
alert(***sum***);

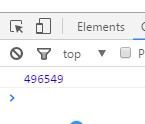


**return是返回的意思,return后的变量就是返回的值**

可以获取返回值之后继续使用该值

比如继续运算然后打印:

**var *sum*** = *getSum*(100, 1000);  
*// alert(sum);* **var *num*** = ***sum*** + 999;  
 **console**.log(***num***);



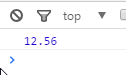
04函数练习1.avi

求任意半径的圆的面积

求任意半径的圆的周长

求任意2个数中的最大值

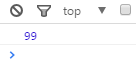
*// 求任意半径的圆的面积***function** *getArea*(r) {  
 **return** 3.14 \* r \* r;  
}  
  
**console**.log(*getArea*(2));



*// 求任意半径的圆的周长***function** *getPer*(r) {  
 **return** 3.14 \* r \*2;  
}  
  
**console**.log(*getPer*(3));



*// 求任意2个数中的最大值***function** *getMax*(a,b) {  
 **return** a > b ? a : b;  
}  
  
**console**.log(*getMax*(99, 69));



05函数练习2.avi

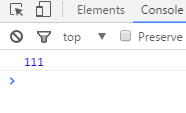
求任意3个数中的最大值

**function** *getMax3*(a, b, c) {  
 **return** (a > b ? a : b) > c ? (a > b ? a : b) : c;  
}  
  
**console**.log(*getMax3*(111, 31, 99));

这样去求三个数的最大值也可以,但是不太方便,之前已经写过一个求两个数最大值的函数,所以可以利用之前的函数来完成

里面的getMax获取a和b之间的最大值,外面的getMax获取最终的最大值

**function** *getMax3*(a, b, c) {  
*// return (a > b ? a : b) > c ? (a > b ? a : b) : c;* **return** *getMax*(*getMax*(a,b), c);  
}  
  
 **console**.log(*getMax3*(111, 31, 99));



**函数中可以调用函数**

**在JS中函数没有”重载”的概念**

**什么是重载?**

**函数名相同,函数的参数不同就是重载,在很多语言中都有重载的概念,这里作为了解内容即可**

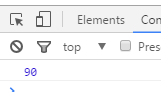
**记住,JS中函数如果重名,后面的会覆盖前面的函数**

06函数练习3.avi

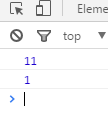
求任意数组中的最大值

求任意数组中的最小值

*//求任意数组中的最大值***function** *getMax*(arr) {  
 **var** max = arr[0];  
 **for** (**var** i = 0; i < arr.**length**; i++) {  
 **if** (max < arr[i]) {  
 max = arr[i];  
 }  
 }  
 **return** max;  
}  
  
**var *arr*** = [11, 22, 33, 85, 12, 90];  
**console**.log(*getMax*(***arr***));



*//求任意数组中的最小值***function** *getMin*(arr) {  
 **var** min = arr[0];  
 **for** (**var** i = 0; i< arr.**length**; i++) {  
 **if** (min > arr[i]) {  
 min = arr[i];  
 }  
 }  
 **return** min;  
}  
**var *arr*** = [11, 22, 33, 85, 12, 90];  
**console**.log(*getMin*(***arr***));  
**console**.log(*getMin*([1, 2, 3, 4, 5, 6]));

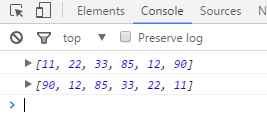


07函数练习4.avi

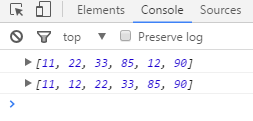
翻转任意数组，返回一个新的数组

对任意数组从小到大排序

*//翻转任意数组，返回一个新的数组***function** *reverse*(arr) {  
 **var** newArr = [];  
 **for** (**var** i = arr.**length** - 1; i >= 0; i--) {  
 newArr[newArr.**length**] = arr[i];  
 }  
 **return** newArr;  
}  
  
**var *arr*** = [11, 22, 33, 85, 12, 90];  
**console**.log(***arr***);  
**console**.log(*reverse*(***arr***));



*//对任意数组从小到大排序***function** *sort*(arr) {  
 **for**(**var** i = 0; i < arr.**length** - 1; i ++) {  
 **for**(**var** j = 0; j < arr.**length** - 1 - i; j++) {  
 **if**(arr[j] > arr[j+1]) {  
 **var** temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = temp;  
 }  
 }  
 }  
 **return** arr;  
}  
  
**var *arr*** = [11, 22, 33, 85, 12, 90];  
**console**.log(***arr***);  
**console**.log(*sort*(***arr***));



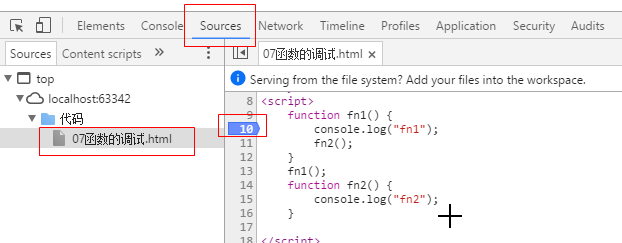
08函数的调试.avi

打开F12调试工具

找到sources选项卡

双击打开当前的html页面

在行号的地方设置断点,不要设置在方法的申明上,可以设置在方法中或者调用方法的地方



F10是执行这行代码,不钻进去



F11是钻进这个方法



09参数详解.avi

1.在JS中 形参和实参的个数可以不一致

如果多传入了实参 多传入的实参因为没人接收所以就废了

如果少传入了实参 有的形参就会因为接收不到实参 值就变成了undefined

2.形参只是把实参的值复制了一份 这两个参数之间相互不会产生影响(**暂时这么理解,将来学习了引用数据类型就不同了**)

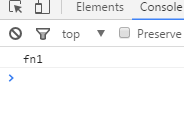
10函数的返回值.avi

1. 代码执行到**return**关键字后 会跳出当前函数 后续代码不再执行
2. 函数的返回值是什么调用这个函数就相当于调用了什么
3. 没有返回值的函数相当于返回了undefined

*//1.代码执行到return关键字后 会跳出当前函数 后续代码不再执行***function** *fn1*() {  
 **console**.log(**"fn1"**);  
 **return** 1;  
 **console**.log(**"return后面的代码"**);  
}

*fn1*();

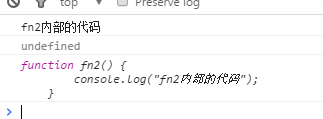
并没有打印return后面的代码



*//2.函数的返回值是什么调用这个函数就相当于调用了什么*

**function** *fn1*() {  
 **return** 1;   
}  
**var *num*** = *fn1*();*//var num = 1;***var *num*** = *fn1*() + 1;*//var num = 1 + 1;***console**.log(*fn1*() + 1);*//console.log(1 + 1);*

*//3.没有返回值的函数相当于返回了undefined***function** *fn2*() {  
 **console**.log(**"fn2内部的代码"**);  
}  
**console**.log(*fn2*());  
**console**.log(*fn2*);



11函数的两种定义方式.avi

1. 函数声明: 直接使用function关键字声明函数

例如: function fn(a,b){} //函数名在function关键字后

1. 函数表达式: 同样使用function关键字声明函数,然后赋值给一个变量

例如: var fn = function(a,b){}; //函数名就是变量名

如果写代码的顺序都是先声明后调用,那两者使用起来没有什么区别,但是**如果声明函数写在调用函数后面,那么函数表达式将无法调用**

**console**.log(*fn1*(1, 2));  
*//正常使用  
//函数声明提升 解析器会把函数的声明提升到最顶端***console**.log(*fn2*(1, 2));  
*//变量声明提升 解析器会把变量的声明提升到最顶端  
//注意 变量声明提升 只提升声明 不提升赋值 所以现在fn2里面保存的是undefined  
//报错 fn2 is not a function  
//函数声明***function** *fn1*(a, b) {  
 **return** a + b;  
}  
*//函数表达式***var** *fn2* = **function** (a, b) {  
 **return** a + b;  
};

12全局变量和局部变量.avi

全局变量: 哪儿都能使用(访问)的变量

局部变量: 只有在一定范围内才能使用,比如说在函数内部声明的变量就只能在函数内部使用

隐式全局变量: 在函数中声明变量不使用var关键字,**强烈不推荐使用**!

*//全局变量 在最外层声明的变量是全局变量 在哪儿都能用***var *a*** = **"a"**;  
*//console.log(a);***function** *fn1*() {  
 **console**.log(***a***);  
}  
*//fn1();  
//局部变量 在函数内部声明的变量是局部变量 只能在当前函数内部使用***function** *fn2*() {  
 **var** b = **"b"**;  
 *//console.log(b);* **c** = **"c"**;*//隐式全局变量 在函数内部但是没有加var的变量* **console**.log(**c**);*//不建议大家这样写*}  
*fn2*();  
*//console.log(b);//会报错***console**.log(**c**);

13递归.avi

递归: 就是在函数中调用函数自身,形成自己调用自己的一个现象

**如果一直自己调用自己,没有定义一个”出口”,那么将会出现栈溢出的错误,所以在使用递归时,一定要注意!定义一个结束递归的条件,当达到某个条件时结束不再调用自己**

**条件变量一定要是全局变量,否则每次递归调用时都会重新声明,等于没用**

**var *i*** = 0;  
**function** *story*() {  
 **console**.log(**"从前有座山，山里有座庙，庙里有个老和尚在给小和尚讲故事，故事是："**);  
 ***i***++;  
 **if** (***i*** < 10) {  
 *story*();  
 }  
}  
*story*();

14回调.avi

将一个函数当做参数传递到另一个函数中,在另一个函数中进行调用,这就叫回调函数(**暂时简单的了解一下这个概念**)

**function** *fn* () {  
 **console**.log(**"我是fn"**);  
 };  
*// console.log(typeof fn());//fn undefined  
// console.log(typeof fn);//function  
 //函数也是一种数据 所以也可以被当作参数传递* **function** *fn1*(a) {  
 *//console.log(a);* a();  
 }  
 *//fn1(num);  
 fn1*(*fn*);  
 *//像这种 被当做参数传递的函数叫做 回调函数*

15键值对.avi

**键值对就是一种对应关系，通过键能够方便地找到值**

数组就是键值对: 索引是键,元素的内容就是值

**键值对的目的就是以空间换时间,提高查询效率**

这也正是数组的特点: 查询速度很快

16英雄对象.avi

从宏观的角度讲，对象是对客观事物的抽象，事物的特征(基本信息)可以用**属性**表示，事物的行为可以用**方法(函数)**表示

从微观的角度讲，对象就是一种**数据类型**，通过对象可以方便地对**变量和函数**进行管理

初期我们甚至可以把他简单地理解为一个工具箱

比如现在需要创建一个英雄对象,这个对象具备一些基本信息(姓名,等级)和一些行为动作(攻击,移动),那这个对象的姓名和等级可以用属性来表示,对象的动作可以用方法(函数)来表示

**var *hero*** = **new** Object(); *//创建对象赋值给hero变量****hero***.**name** = **"黑马人"**; *//给对象的名字属性赋值****hero***.**level** = 99; *//给对象的等级属性赋值****hero***.move = **function** () { *//给对象的移动行为赋值 给对象加了个移动方法* **console**.log(**"我们正在走向世界"**);  
}  
***hero***.attack = **function**(name) { *//给对象的攻击行为赋值 给对象加了个攻击方法* **console**.log(***hero***.name + **"正在攻击"** + name);  
}  
  
**console**.log(***hero***.name); *//查看对象的名字 对象名.属性名***console**.log(***hero***.**level**); *//查看对象的等级****hero***.move(); *//调用对象的移动方法 对象名.方法名()****hero***.attack(**"某鸟人"**); *//调用对象的攻击方法*

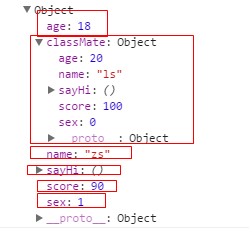
17学生对象.avi

为什么要使用对象?

因为在开发中有很多的变量和很多的函数,可以把**描述同一个事务的变量和函数放入一个对象中**,这样管理和使用起来就会方便很多很多

对象的属性可以保存所有类型的数据,值类型,函数,对象都可以保存

*//学生对象（要有姓名 年龄 性别 分数 有同学 会打招呼）***var *stu*** = **new** Object();  
***stu***.**name** = **"zs"**;  
***stu***.**age** = 18;  
***stu***.**sex** = 1;  
***stu***.**score** = 90;  
***stu***.sayHi = **function** () {  
 **console**.log(**"大家好，我是"** + **this**.**name**);  
 *//this指的是当前函数所属的对象*};  
*//stu.sayHi();***var *stu2*** = **new** Object();  
***stu2***.**name** = **"ls"**;  
***stu2***.**age** = 20;  
***stu2***.**sex** = 0;  
***stu2***.**score** = 100;  
***stu2***.sayHi = **function** () {  
 **console**.log(**"我只告诉你我今年"** + **this**.**age**);  
};  
*//stu2.sayHi();****stu***.**classMate** = ***stu2***;  
**console**.log(***stu***);



18对象字面量.avi

之前创建对象的方式比较麻烦,可以稍微简单一些,不需要new Object();只要用{}就表示创建对象

*//var stu = new Object();***var *stu*** = {};  
***stu***.**name** = **"zs"**;  
***stu***.**age** = 18;  
***stu***.sayHi = **function** () {  
 **console**.log(**"大家好"**);  
};

其实这样也没简单多少,对象和数组一样,也有字面量的创建方式

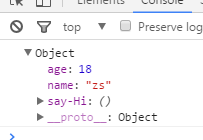
比如数组的创建方式就是var arr = [1,2,3,4,5];

对象的字面量创建方式就是var stu = {...};

**区别在于中括号是数组,大括号是对象**

*//对象字面量***var *stu*** = {  
 **name**: **"zs"**,  
 **age**: 18,  
 sayHi: **function** () {  
 **console**.log(**"大家好"**);  
 }  
};  
**var *stu*** = {  
 **"name"**: **"zs"**,  
 **"age"**: 18,  
 **"say-Hi"**: **function** () {  
 **console**.log(**"大家好"**);  
 }  
};  
  
**console**.log(***stu***);

**属性名加不加双引号都可以,标准写法应该是要加双引号的,不过一般习惯都不加**



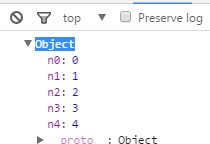
19访问属性的两种方式.avi

直接使用属性名去访问属性很好理解很直接,但是并不灵活

对象也可以像数组访问元素一样的方式去访问属性,通过键访问值,但是需要注意,数组是有索引的,但对象没有索引,只能通过属性名去访问

也被称为**无序属性的集合**

*//对象是无序属性的集合，我们可以把他看成是键值对***var *arr*** = [1, 2, 3, 4, 5];  
***arr***[0] = 20;  
**var *obj*** = {};  
*//obj.age = 18;  
//obj["age"] = 18;  
//console.log(obj);***for** (**var *i*** = 0; ***i*** < 5; ***i***++) {  
 ***obj***[**"n"** + ***i***] = ***i***;}  
**console**.log(***obj***);



20for循环的两种方式.avi

之前for循环遍历数组是没问题的,因为数组的索引是数字

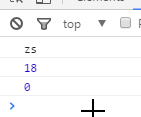
但是如果想遍历获取对象的所有属性,那使用传统的for循环就没办法了

可以使用for...in循环,语法:

for(var 键名 in 要遍历的对象) {...}

在循环体中可以使用键名这个变量,通过对象[键名]的形式获取到对象的所有值

**var *arr*** = [1, 2, 3, 4, 5];  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 *//console.log(arr[i]);*}  
  
*//for(var 键 in 要遍历的对象){}***for** (**var *key* in *arr***) {  
 *//console.log(key);  
 //console.log(arr[key]);*}  
  
**var *obj*** = {  
 **name**: **"zs"**,  
 **age**: 18,  
 **sex**: 0  
};  
  
**for** (**var** k **in *obj***) {  
 *//console.log(k);* **console**.log(***obj***[k]);  
}



day04

01批量创建对象.avi

创建一个学生对象之前已经学习过了

**var** stu = **new** Object();  
stu.**name** = **"张三"**;  
stu.**age** = 18;  
stu.**sex** = 0;  
stu.**score** = 90;  
stu.sayHi = **function**() {  
 **console**.log(**"大家好,我是"** + **this**.**name**);  
}

那如果需要创建多个学生对象,只能多次拷贝粘贴上面的代码,使用起来非常不方便

如果有需求要同时创建多个学生对象,但每个学生的基本信息都不相同怎么办呢?

这时候就可以利用**封装函数**来解决了,函数正是用于封装一个功能的,比如现在的创建对象功能

*//自己创建一个学生对象（要有姓名 年龄 性别 分数 会打招呼）***function** *createStu*() {  
 **var** stu = **new** Object();  
 stu.**name** = **"张三"**;  
 stu.**age** = 18;  
 stu.**sex** = 0;  
 stu.**score** = 90;  
 stu.sayHi = **function**() {  
 **console**.log(**"大家好,我是"** + **this**.**name**);  
 }  
 **return** stu;  
}  
  
**var *stu1*** = *createStu*();  
**var *stu2*** = *createStu*();  
**var *stu3*** = *createStu*();  
**console**.log(***stu1***);  
**console**.log(***stu2***);  
**console**.log(***stu3***);

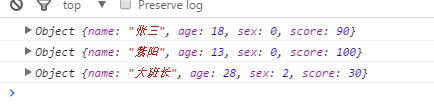


这样封装好函数后,多次调用即可创建多个对象了

但是这些对象都是相同的属性,并不能满足现在的需求

**对象的属性应该由调用该函数的使用者来决定,所以可以将对象的属性值作为函数的参数传递进来**

*//自己创建一个学生对象（要有姓名 年龄 性别 分数 会打招呼）***function** *createStu*(name,age,sex,score) {  
 **var** stu = **new** Object();  
 stu.**name** = name;  
 stu.**age** = age;  
 stu.**sex** = sex;  
 stu.**score** = score;  
 stu.sayHi = **function**() {  
 **console**.log(**"大家好,我是"** + **this**.**name**);  
 }  
 **return** stu;  
}  
  
**var *stu1*** = *createStu*(**"张三"**,18,0,90);  
**var *stu2*** = *createStu*(**"紫阳"**,13,0,100);  
**var *stu3*** = *createStu*(**"大班长"**,28,2,30);  
**console**.log(***stu1***);  
**console**.log(***stu2***);  
**console**.log(***stu3***);



这样一来就完成需求了,将来就算要创建多个学生对象,都只需要调用createStu()函数即可完成,如果需要定义创建对象的属性,传入参数就行了

02this关键字.avi

**this关键字非常重要**

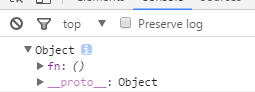
它在不同的情况所表示的意义不同

分为3种情况:

*//1.全局上下文（在所有函数的外部） this指的是全局对象window***console**.log(**this**);



*//2.函数上下文（在某个函数的内部） this指的是当前函数所属的对象***var *obj*** = {  
 fn: **function** () {  
 **console**.log(**this**);  
 }  
};  
***obj***.fn();

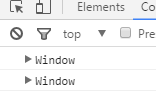


函数如果在对象中,那么这个函数就是该对象的方法,所以this是该对象

但如果直接声明一个函数,没有给任何对象,那么这个函数就属于window对象,所以this表示的是window对象

可以直接调用该函数,也可以使用window.fn2()去调用该函数

**function** *fn2*() {  
 **console**.log(**this**);  
}  
*fn2*();  
**window**.*fn2*();



*//3.当一个函数被作为构造函数使用时（前面加new关键字）*

**new关键字会让this的指向改变，并让其成为当前函数的返回值**

03构造函数模式.avi

**之前创建的对象打印在控制台都是Object类型**,其实使用该对象时并不能准确的知道它到底是一个什么东西,可能想描述学生对象,可能是想描述狗狗对象,所以需要给之前**创建的对象指定一种自定义的类型**,这时候就需要用到**构造函数**了

构造函数: 是一种比较特殊的函数,在JS中可以用于初始化对象,它和普通函数的区别在于它默认返回值为this,**这个函数不是直接调用的,而是使用new关键字创建对象的**

1.构造函数首字母大写

2.构造函数并没有显示返回任何东西,new 关键字会自动创建给定的类型并返回他们,当调用构造函数时,new会自动创建this对象,且**类型就是构造函数类型**

3.也可以在构造函数中显示调用return.如果返回的值是一个对象，它会代替新创建的对象返回。如果返回的值是一个原始类型，它会被忽略，新创建的对象会被返回.

**function** *Student*(name, age, sex, score) {  
 *//var stu = new Object(); //不需要在函数中创建对象了,构造函数就是用来创建对象的模板* **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**sex** = sex;  
 **this**.**score** = score;  
 **this**.sayHi = **function** () {  
 **console**.log(**"大家好，我是"** + **this**.**name**);  
 };  
 *//return stu; //不需要手动返回创建的对象了,默认返回this*}

**var *s2*** = **new** *Student*(**"ls"**, 20, 0, 100);



可以看到现在返回的对象不再是Object类型了,而是Student(构造函数名)类型

可以通过instanceof关键字来判断对象是否由某个构造函数创建出来的

*//A instanceof B 表示A是不是一个通过B创建出来的对象***console**.log(***s2* instanceof** *Student*);*//true***console**.log([] **instanceof** *Student*);*//false*

04值类型和引用类型.avi

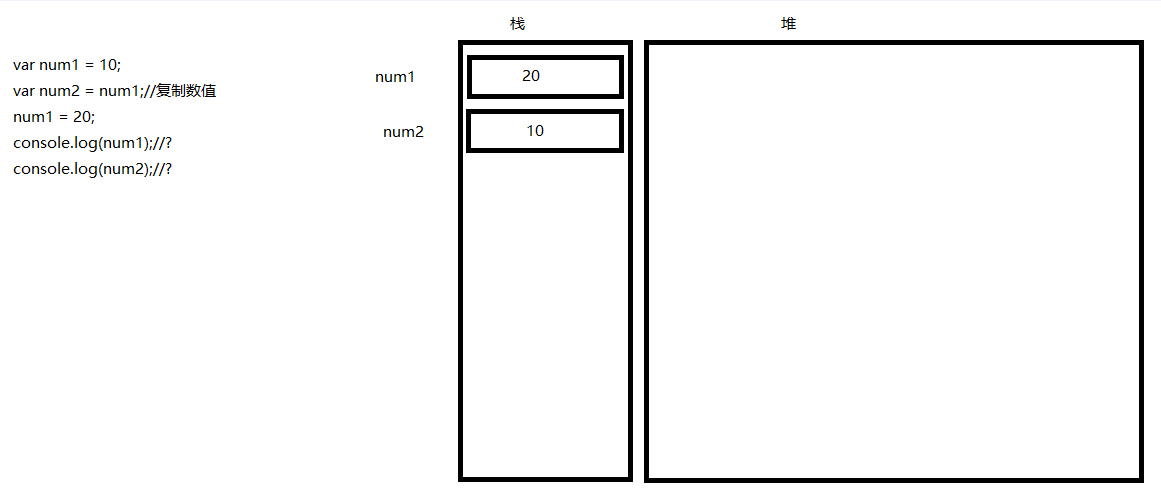
JS中数据类型分为两大类:

基本数据类型

引用数据类型

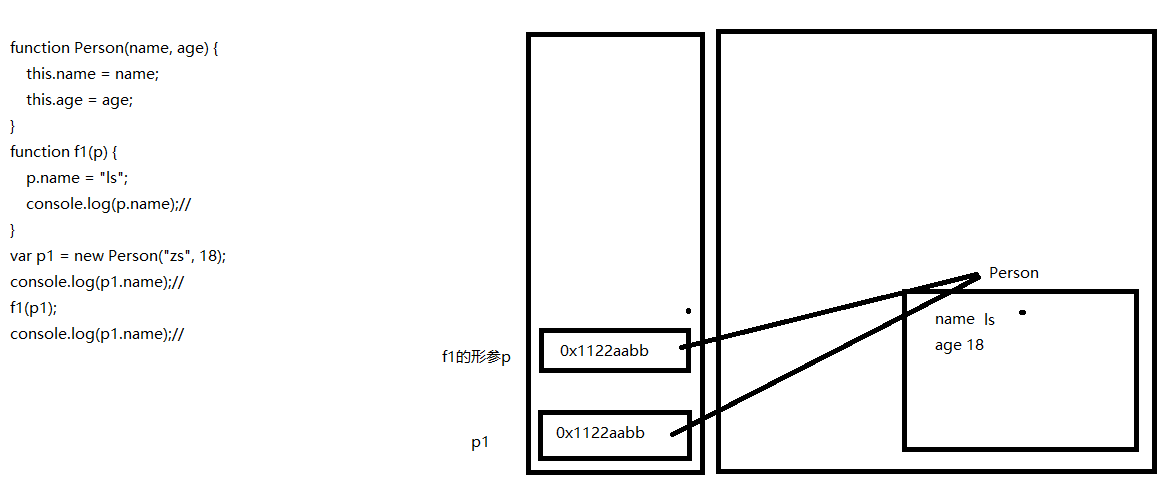
声明基本数据类型的变量时,会在栈内存区域开辟空间存储数值,然后赋值给变量

*//基本类型的复制***var *num1*** = 10;  
**var *num2*** = ***num1***;*//复制数值****num1*** = 20;  
**console**.log(***num1***);*//20***console**.log(***num2***);*//10*



而引用数据类型则不同,它会在**堆内存中开辟一块空间存储对象的具体内容**,并且**这块空间具有一个地址值**,然后在**栈内存中开辟一块区域存储堆内存的地址值,表示引用该对象**,最后将栈内存的内容赋值给变量

*//引用类型的复制***function** *Person*(name, age) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
}  
**var *p1*** = **new** *Person*(**"zs"**, 18);  
**var *p2*** = ***p1***;*//复制引用****p2***.**name** = **"ls"**;  
**console**.log(***p1***.**name**);*//ls***console**.log(***p2***.**name**);*//ls*

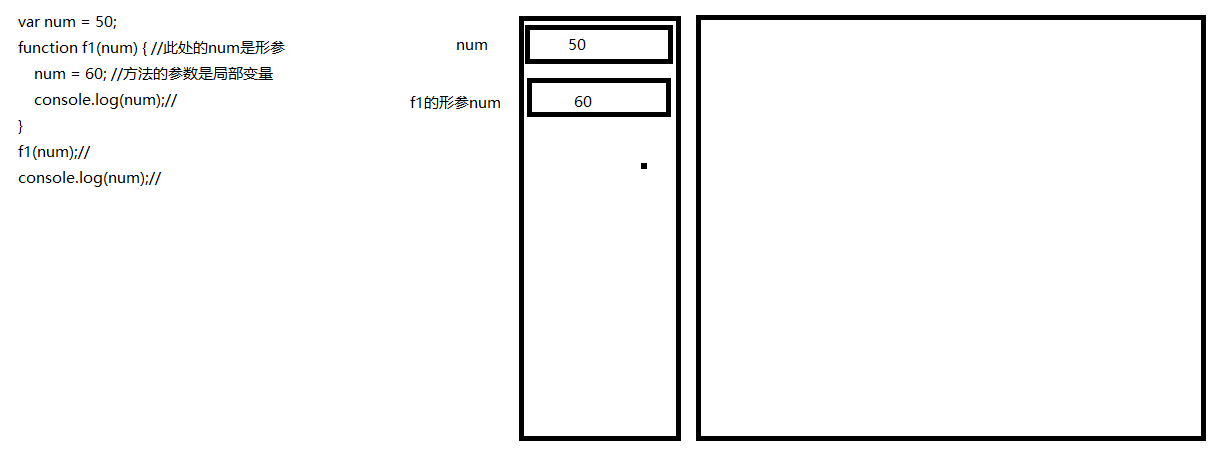


05值类型和引用类型作参数.avi

这里基本数据类型和引用数据类型作为参数传递其实指的就是,调用函数时传入基本数据类型和引用数据类型的不同之处

基本数据类型: 就是将**实参赋值给了形参**,其实就是**一个变量赋值给另一个变量**,**在函数中修改形参的值不会影响到外面的实参**

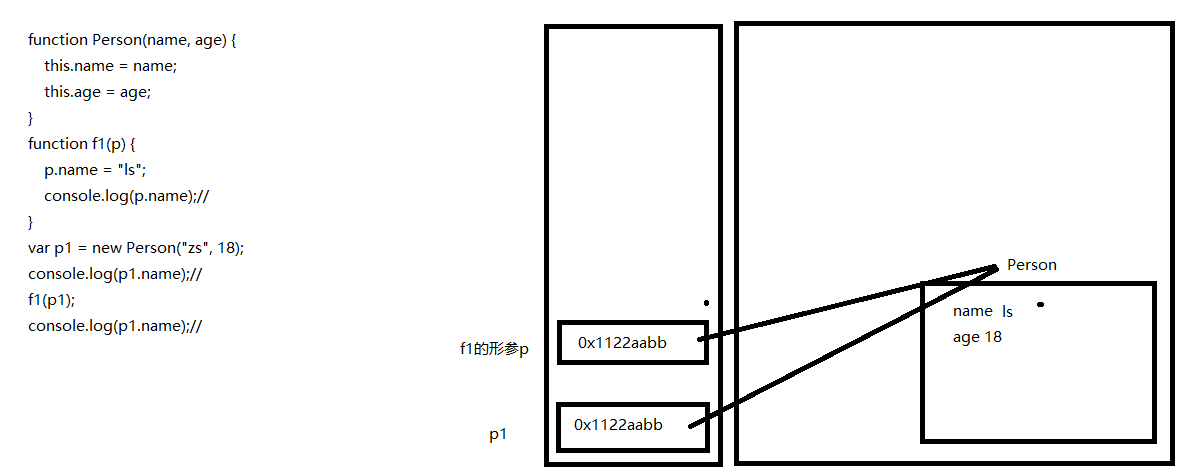
*//基本类型做参数***var *num*** = 50;  
**function** *f1*(num) { *//此处的num是形参* num = 60; *//方法的参数是局部变量* **console**.log(num);*//60*}  
*f1*(***num***);*//此处的num是实参 会把num的值赋值一份传递给函数内部***console**.log(***num***);*//50*



引用数据类型: 也是将**实参赋值给形参**,但是**由于实参是引用数据类型**,所以就是将**保存的堆内存的地址值赋值给形参**,**通过形参可以找到堆内存中的对象对其进行属性的修改**,同时**实参也是引用的堆内存中的对象,所以属性也等于是被修改了**

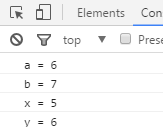
*//引用类型做参数***function** *Person*(name, age) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
}  
**function** *f1*(p) {

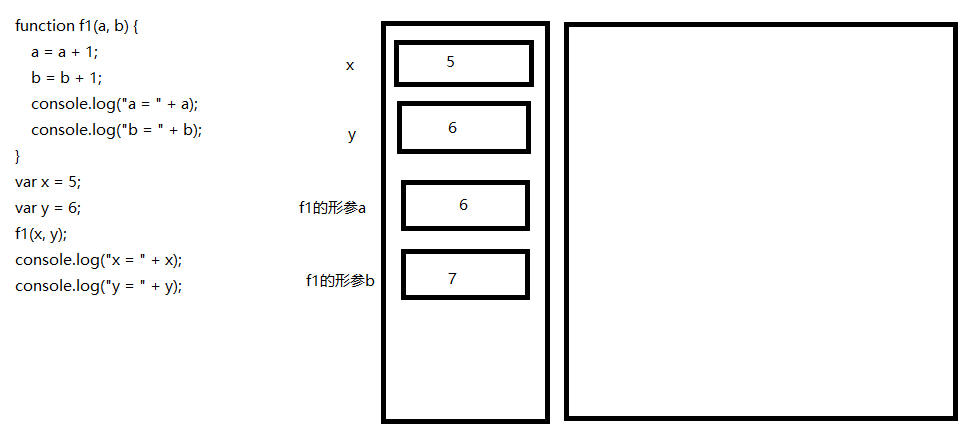
p = new Person(“王五”,19);  
 //p.**name** = **"ls"**;  
 **console**.log(p.**name**);*//ls*}  
**var *p1*** = **new** *Person*(**"zs"**, 18);  
**console**.log(***p1***.**name**);*//zs  
f1*(***p1***);  
**console**.log(***p1***.**name**);*//ls*



06面试题.avi

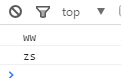
**function** *f1*(a, b) {  
 a = a + 1;  
 b = b + 1;  
 **console**.log(**"a = "** + a);*//6* **console**.log(**"b = "** + b);*//7*}  
**var *x*** = 5;  
**var *y*** = 6;  
*f1*(***x***, ***y***);  
**console**.log(**"x = "** + ***x***);*//5***console**.log(**"y = "** + ***y***);*//6*

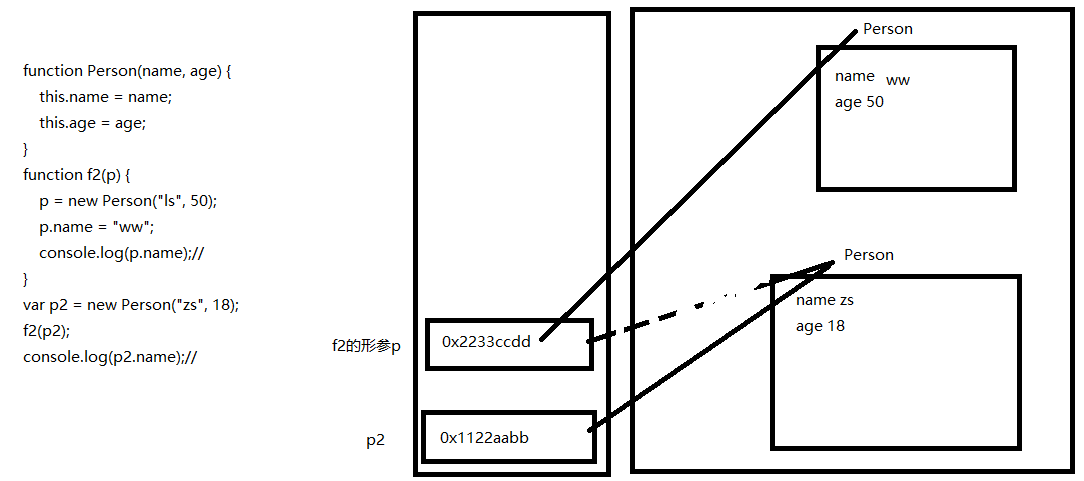




**注意: 由于在f2函数中又创建了一个对象并赋值给了f2的形参p,所以又会在堆内存中开辟一块新空间,并将地址值赋值给形参p,所以在此之后形参p和外面的实参p2就不再指向同一片区域了,这时候在f2函数中修改形参p对象的属性值不会再影响到外面的实参p2了**

**function** *Person*(name, age) {  
 **this**.**name** = name;  
 **this**.**age** = age;  
}  
**function** *f2*(p) {  
 p = **new** *Person*(**"ls"**, 50);  
 p.**name** = **"ww"**;  
 **console**.log(p.**name**);*//ww*}  
**var *p2*** = **new** *Person*(**"zs"**, 18);  
*f2*(***p2***);  
**console**.log(***p2***.**name**);*//zs*





07数组.avi

内置对象: 是**JS内部的一些对象**,自带一些功能和属性,暂时可以理解成一个工具箱

先了解第一个内置对象: **Array**

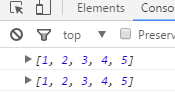
这个内置对象中提供了一些对数组操作的工具

转换数组(将数组转换成一些其他形式):

valueOf() //返回数组对象本身,和直接输出数组对象是一样的

**var *arr*** = [1, 2, 3, 4, 5];  
**console**.log(***arr***);  
**console**.log(***arr***.valueOf());

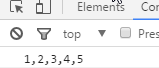
以上两种输出是相同的,其实直接输出数组对象就等于自动调用了valueOf()方法



toString() //把数组转换成字符串

**var *arr*** = [1, 2, 3, 4, 5];  
**console**.log(***arr***.toString());

将数组中的元素全部取出并以逗号相连转为字符串

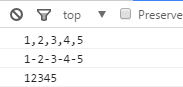


join() //把字符串中的每一项进行拼接

**var *arr*** = [1, 2, 3, 4, 5];  
**console**.log(***arr***.join());  
**console**.log(***arr***.join(**"-"**));  
**console**.log(***arr***.join(**""**));

不传入参数即表示默认用逗号连接转为字符串

传入参数表示用传入的参数连接并转为字符串



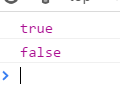
08数组的检测.avi

数组检测: 用于判断是否是数组类型

有以下两种方式

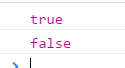
1. instanceof关键字

**var *arr*** = [1,2,3,4,5];  
**console**.log(***arr* instanceof** Array); *//true***console**.log(**"a" instanceof** Array); *//false*



1. Array.isArray();方法

**var *arr*** = [1,2,3,4,5];  
**console**.log(Array.isArray(***arr***));*//true***console**.log(Array.isArray(**"a"**));*//false*

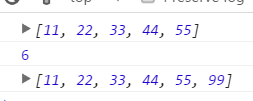


09数组的增删方法.avi

数组一共有四种增删方法:

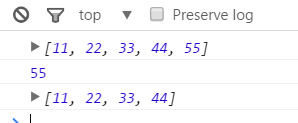
1. push

**var *arr*** = [11,22,33,44,55];  
**console**.log(***arr***);  
**console**.log(***arr***.push(99));*//从后面添加元素 返回新数组的长度***console**.log(***arr***);



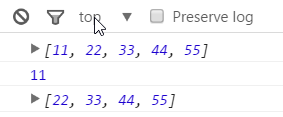
1. pop

**var *arr*** = [11,22,33,44,55];  
**console**.log(***arr***);  
**console**.log(***arr***.pop());*//从后面删除元素 返回被删除的元素***console**.log(***arr***);



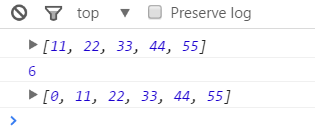
1. shift

**var *arr*** = [11,22,33,44,55];  
**console**.log(***arr***);  
**console**.log(***arr***.shift());*//从前面删除元素 返回被删除的元素***console**.log(***arr***);



1. unshift

**var *arr*** = [11,22,33,44,55];  
**console**.log(***arr***);  
**console**.log(***arr***.unshift(0));*//从前面添加元素 返回新数组的长度***console**.log(***arr***);

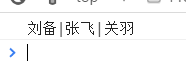


10数组的join方法.avi

练习:

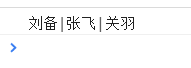
将一个字符串数组输出为用|分割的字符串，如 ["刘备", "张飞", "关羽"] 变为 "刘备|张飞|关羽"。（两种方式）

**var *arr*** = [**"刘备"**, **"张飞"**, **"关羽"**];  
**function** *join*(arr, sep) {  
 **var** str = arr[0];  
 **for** (**var** i = 1; i < arr.**length**; i++) {  
 str = str + sep + arr[i];  
 }  
 **return** str;  
}  
**console**.log(*join*(***arr***, **"|"**));



以上是自己封装join函数,不过JS已经提供了一个数组的join方法进行切割数组,直接调用即可

**var *arr*** = [**"刘备"**, **"张飞"**, **"关羽"**];  
**console**.log(***arr***.join(**"|"**));



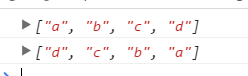
11数组的reverse方法.avi

练习:

将字符串数组的顺序进行反转。["a","b","c","d"]变为["d","c","b","a"]。（三种方式）

方式一:

**var *arr*** = [**"a"**, **"b"**, **"c"**, **"d"**];  
**console**.log(***arr***);  
**function** *reverse*(arr) {  
 **var** newArr = [];  
 **for** (**var** i = arr.**length** - 1; i >= 0; i--) {  
 newArr.push(arr[i]);  
 }  
 *//console.log(newArr);* **return** newArr;  
}  
**console**.log(*reverse*(***arr***));



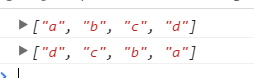
方式二:

**var *arr*** = [**"a"**, **"b"**, **"c"**, **"d"**];  
**console**.log(***arr***);  
**function** *reverse*(arr) {  
 **for** (**var** i = 0; i < arr.**length** / 2; i++) {  
 *//0 arr.length-1-0  
 //1 arr.length-1-1  
 //2 arr.length-1-2  
 //i arr.length-1-i* **var** temp = arr[i];  
 arr[i] = arr[arr.**length** - 1 - i];  
 arr[arr.**length** - 1 - i] = temp;  
 }  
 *//console.log(arr);* **return** arr;  
}  
**console**.log(*reverse*(***arr***));



方式三:

**var *arr*** = [**"a"**, **"b"**, **"c"**, **"d"**];  
console.log(***arr***);  
console.log(***arr***.reverse());

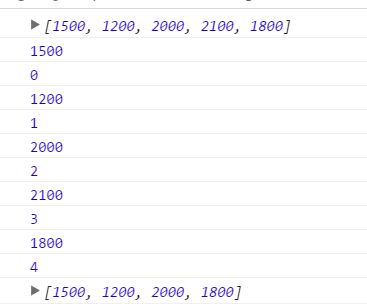


12数组的迭代方法.avi

练习:

工资的数组[1500,1200,2000,2100,1800]把工资大于等于2000的删除

**var *arr*** = [1500, 1200, 2000, 2100, 1800];  
 **console**.log(***arr***);  
 **var *newArr*** = ***arr***.filter(**function** (element, index, array) {  
 **console**.log(element);  
 **console**.log(index);  
*// console.log(array);* **if** (element > 2000) {  
 **return false**;  
 }  
 **return true**;  
 });  
 **console**.log(***newArr***);



filter方法:

需要**传入一个回调函数**,该回调函数需具有**三个参数**,这个回调函数由js内部调用

**filter方法会遍历数组,每次遍历到一个元素就会执行回调函数一次,第一个参数为当前元素,第二个参数为当前元素的索引,第三个参数为当前数组**

**回调函数返回值为true则表示当前需要保留当前元素,返回false则表示不保留当前元素**,如果没写返回值默认为undefined也就是false

当数组迭代完成时,会返回一个新数组,**这个新数组中的元素取决于在回调函数中,有多少个元素返回了true得以保留**

13数组的位置方法.avi

练习:

["c", "a", "z", "a", "x", "a"]找到数组中第一个a出现的位置

["c", "a", "z", "a", "x", "a"]找到数组中最后一个a出现的位置

找到数组中第一个a出现的位置:

**var *arr*** = [**"c"**, **"a"**, **"z"**, **"a"**, **"x"**, **"a"**];  
**console**.log(***arr***.indexOf(**"a"**));*//1*



找到数组中最后一个a出现的位置

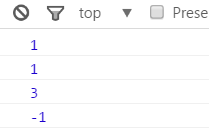
**var *arr*** = [**"c"**, **"a"**, **"z"**, **"a"**, **"x"**, **"a"**];  
**console**.log(***arr***.lastIndexOf(**"a"**));*//5*



indexOf的第二个参数:

从哪个索引开始找,包含该索引

**var *arr*** = [**"c"**, **"a"**, **"z"**, **"a"**, **"x"**, **"a"**];  
**console**.log(***arr***.indexOf(**"a"**, 0));  
**console**.log(***arr***.indexOf(**"a"**, 1));*//从1开始 包括1***console**.log(***arr***.indexOf(**"a"**, 2));*//从2开始***console**.log(***arr***.indexOf(**"b"**));*//找不到 返回-1*



14寻找每个a出现的位置.avi

因为要找数组中每一个a的出现的位置,所以肯定需要用到循环

具体循环多少次也不确定,所以可以采取while循环

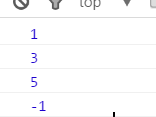
每次循环记录一下index并打印

循环条件为index不等于-1就继续找,因为如果等于-1了就表示没有了

**var *arr*** = [**"c"**, **"a"**, **"z"**, **"a"**, **"x"**, **"a"**];  
**var *index*** = -1;  
**do** {  
 ***index*** = ***arr***.indexOf(**"a"**, 0);  
 **console**.log(***index***);  
} **while** (***index*** !== -1);

但如果这样去循环就会造成死循环,因为每次循环都是从0找,那永远找的都是第一个a,所以indexOf的第二个参数应该是动态变化的,也就是index+1而非index

**var *arr*** = [**"c"**, **"a"**, **"z"**, **"a"**, **"x"**, **"a"**];  
**var *index*** = -1;  
**do** {  
 ***index*** = ***arr***.indexOf(**"a"**, ***index***+1);  
 **console**.log(***index***);  
} **while** (***index*** !== -1);



15统计元素的出现次数.avi

练习:

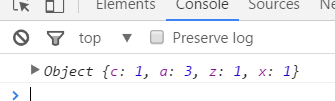
["c", "a", "z", "a", "x", "a"]获取数组中每个元素出现的次数

要统计每个元素出现的次数,可以使用键值对(对象)的形式来存储数据

元素作为键(对象的属性),出现的次数作为值

元素每出现一次,就让键对应的值自增一即可

**var *arr*** = [**"c"**, **"a"**, **"z"**, **"a"**, **"x"**, **"a"**];  
**var *count*** = {}; *//计数对象***for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **var *item*** = ***arr***[***i***];  
 **if** (***count***[***item***]) { *//循环过程中判断,如果这个元素在对象中有键,那取出值进行自增* ***count***[***item***]++;  
 } **else** { *//如果这个元素在对象中没有键,那就加一个键进去,初始值为1,表示出现了一次* ***count***[***item***] = 1;  
 }  
}  
**console**.log(***count***);



16数组的其他方法1.avi

练习:

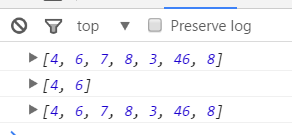
从数组中截取一个新的数组,返回新的数组

删除和替换数组中的一部分

截取数组: slice,从一个索引开始截取到另一个索引结束

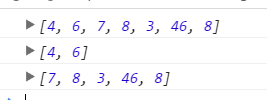
**包头不包尾**

**var *arr*** = [4, 6, 7, 8, 3, 46, 8];  
**console**.log(***arr***);  
*//1开始 2结束 [start,end) 开始能取到 结束取不到***console**.log(***arr***.slice(0, 2));*//slice 返回一个新数组不会改变原数组***console**.log(***arr***);



删除数组中的元素: splice,从一个索引开始,删除多少个元素

**var *arr*** = [4, 6, 7, 8, 3, 46, 8];  
**console**.log(***arr***);  
*//参数1:开始的索引 参数2:删除多少个元素 参数3:要加入的元素***console**.log(***arr***.splice(0, 2));*//会对原数组进行修改***console**.log(***arr***);



截取数组和删除数组中元素的区别:

slice

参数1:从哪个索引开始

参数2:到哪个索引结束(不包含该索引)

不会影响原数组,返回值为新数组

splice

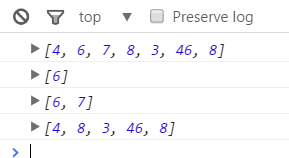
参数1:从哪个索引开始

参数2:删除多少个元素

会影响原数组,直接在原数组中删除,返回值为删掉的元素

可以往删除元素的地方添加元素

**var *arr*** = [4, 6, 7, 8, 3, 46, 8];  
console.log(***arr***);  
console.log(***arr***.slice(1, 2));*//6 开始和结束*console.log(***arr***.splice(1, 2));*//6,7 开始和个数*console.log(***arr***);

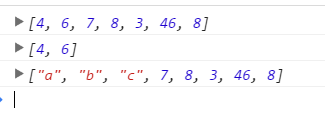


往删除元素的地方添加元素

将以前的4,6删掉了,添加了后面的元素

可以添加多个,使用逗号隔开即可

**var *arr*** = [4, 6, 7, 8, 3, 46, 8];  
**console**.log(***arr***);  
**console**.log(***arr***.splice(0, 2, **"a"**, **"b"**, **"c"**));  
**console**.log(***arr***);



17数组的其他方法2.avi

练习:

对一个数组进行循环遍历

foreach方式进行迭代: ES5的新方法

调用数组上的forEach()方法

和filter()方法一样传入一个回调函数,有JS内部调用进行数组遍历,循环遍历数组时,每循环到一个元素就会调用一次

回调函数同样有3个参数

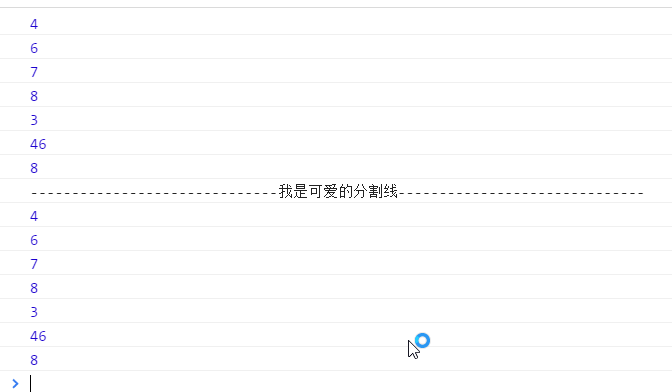
参数1: 遍历到的元素

参数2: 便利到的元素索引

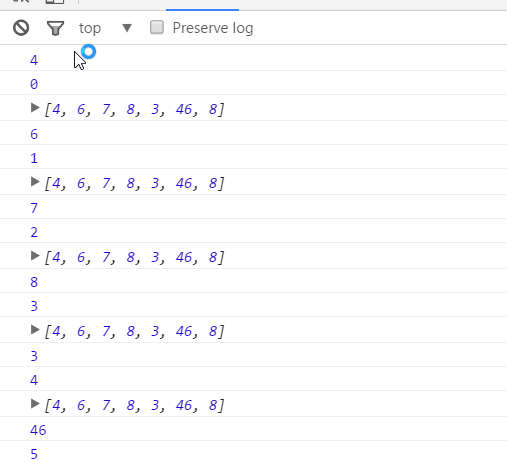
参数3: 原数组

如果只是最简单的想获取每一个元素,就可以只写一个参数即可

**var *arr*** = [4, 6, 7, 8, 3, 46, 8];  
**for** (**var *i*** = 0; ***i*** < ***arr***.**length**; ***i***++) {  
 **console**.log(***arr***[***i***]);  
}  
**console**.log(**"------------------------------我是可爱的分割线------------------------------"**);  
***arr***.forEach(**function**(e){  
 **console**.log(e);  
});



**var** arr = [4, 6, 7, 8, 3, 46, 8];  
arr.forEach(**function**(element, index, arr){  
 **console**.log(element);  
 **console**.log(index);  
 **console**.log(arr);  
});



清空数组:

var array = [1,2,3,4,5,6];

方式1

array.splice(0,array.length); //删除数组中所有项目

方式2

array.length = 0; //length属性可以赋值

方式3

array = []; //推荐