

Javascript基础部分

1.编程语言

1.1编程

编程:就是让计算机为解决某个问题而使用某种程序设计语言编写程序代码,并最终得到结果的过程。

计算机程序:就是计算机所执行的一系列的**指令集合**,而程序全部都是用我们所掌握的语言来编写的,所以人们要控制计算机一定要通过计算机语言向计算机发出命令。

1.2计算机语言

计算机语言指用于人与计算机之间通讯的语言,它是人与计算机之间传递信息的媒介。

计算机语言的种类非常的多,总的来说可以分成**机器语言、汇编语言和高级语言**三大类。

实际上,计算机最终所执行的都是**机器语言**,它是由"0"和"1"组成的二进制数,二进制是计算机语言的基础。

注意:上面所定义的计算机指的是**任何能够执行代码的设备**,可能是智能手机、ATM机、黑莓PI、服务器等等。

1.3编程语言

可以通过类似于人类语言的“语言”来控制计算机,让计算机为我们做事情,这样的语言就叫做编程语言(Programming Language)。

编程语言是用来控制计算机的一系列指令,它有固定的格式和词汇(不同编程语言的格式和词汇不一样),必须遵守。

如今通用的编程语言有两种形式:**汇编语言和高级语言**。

- 汇编语言**和机器语言实质是相同的,都是直接对硬件操作,只不过指令采用了英文缩写的标识符,容易识别和记忆。

- 高级语言**主要是相对于低级语言而言,它并不是特指某一种具体的语言,而是包括了很多编程语言,常用的有C语言、C++、Java、C#、Python、PHP、JavaScript、Go语言、Objective-C、Swift等。

1.4翻译器

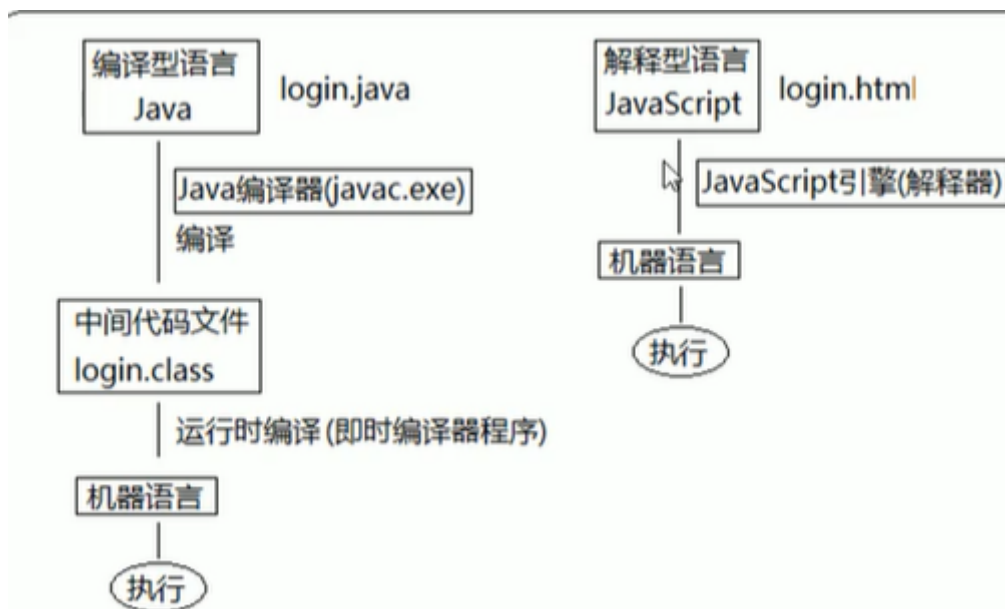
高级语言所编制的程序不能被计算机识别,必须经过转换才能被执行,为此,我们需要一个翻译器。

翻译器可以将我们所编写的源代码转换为机器语言,这也被称为二进制化。记住1和0。

1.4.1翻译的方式与区别

- 翻译器翻译的方式有两种:一个是编译,另外一个解释。两种方式之间的区别在于翻译的时间点不同
- 编译器是在代码执行之前进行编译,生成中间代码文件
- 解释器是在运行时进行及时解释,并立即执行(当编译器以解释方式运行的时候,也称之为解释器)

执行过程



类似于请客吃饭:

- 编译语言:首先把所有菜做好,才能上桌吃饭
- 解释语言:好比吃火锅,边吃边涮,同时进行

1.5编程语言和标记语言区别

- 编程语言**有很强的逻辑和行为能力。在编程语言里你会看到很多if else、for、while等具有逻辑性和行为能力的指令，这是主动的。
- 标记语言**(html)不用于向计算机发出指令,常用于格式化和链接。标记语言的存在是用来被读取的,他是被动的。

1.6标识符、关键字、保留字

1.6.1标识符

标识(zhi)符:就是指开发人员为变量、属性、函数、参数取的名字。

标识符不能是关键字或保留字。

1.6.2关键字

关键字:是指JS本身已经使用了的字,不能再用它们充当变量名、方法名。

包括: break、case、catch、continue、default、delete、do、else、finally、for、function、if、in、instanceof、new、return、switch、this、throw、try、typeof、var、void、while、with等。

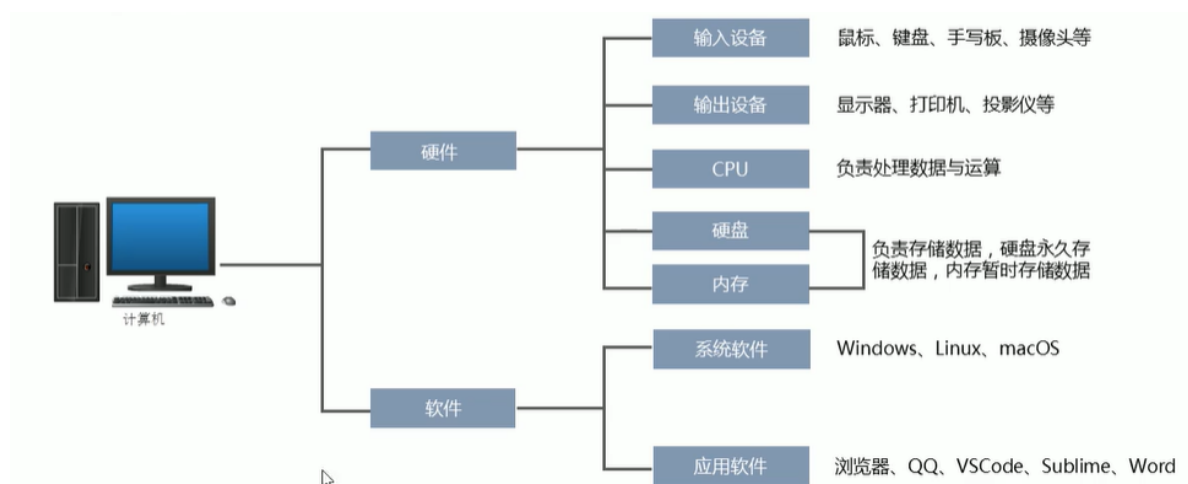
1.6.3保留字

保留字:实际上就是预留的"关键字",意思是现在虽然还不是关键字,但是未来可能会成为关键字,同样不能使用它们当变量名或方法名。

包括: boolean、byte、char、class、const、debugger、double、enum、export、extends、final、float、goto、implements、import、int、interface、long、native、package、private、protected、public、short、static、super、synchronized、throws、transient、volatile等。

2.计算机基础

2.1计算机组成



软件运行在硬件上。

2.2数据存储

- 1.计算机内部使用二进制0和1来表示数据。
- 2.所有数据,包括文件、图片等最终都是以二进制数据(0和1)的形式存放在硬盘中的。
- 3.所有程序,包括操作系统,本质都是各种数据,也以二进制数据的形式存放在硬盘中。平时我们所说的安装软件，其实就是把程序文件复制到硬盘中。
- 4.硬盘、内存都是保存的二进制数据。

2.3数据存储单位

bit< byte< kb<GB < TB<....

- 位(bit): 1bit可以保存一个0或者1 (最小的存储单位)
- 字节(Byte): 1B= 8b
- 千字节(KB): 1KB= 1024B
- 兆字节(MB): 1MB= 1024KB
- 吉字节(GB): 1GB = 1024MB
- 太字节(TB): 1TB= 1024GB

2.4程序运行



- 1.打开某个程序时,先从硬盘中把程序的代码加载到内存中
- 2.CPU执行内存中的代码

注意:之所以要内存的一个重要原因,是因为cpu运行太快了,如果只从硬盘中读数据,会浪费cpu性能,所以才使用存取速度更快的内存来保存运行时的数据。(内存是电,硬盘是机械)

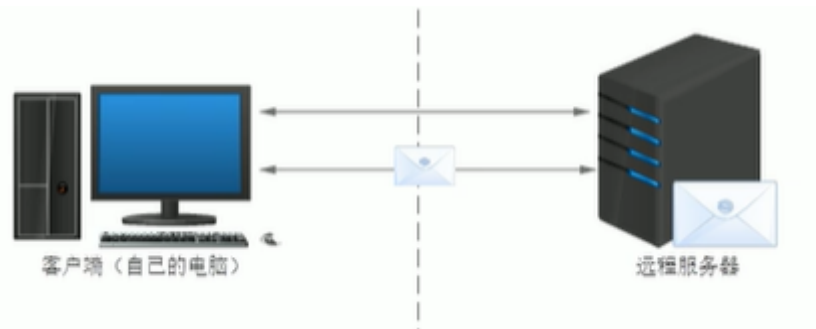
3.初识JavaScript

3.1 JavaScript历史

- 布兰登·艾奇(BrendanEich, 1961年~)。
- 神奇的大哥在1995年利用10天完成JavaScript设计。
- 网景公司最初命名为LiveScript ,后来在与Sun合作之后将其改名为JavaScript.

3.2 JavaScript是什么

- JavaScript 是世界上最流行的语言之一,是一种运行在**客户端的脚本语言**(Script是脚本的意思)
- 脚本语言:不需要编译,运行过程中由js解释器(js引擎)逐行来进行解释并执行
- 现在也可以基于Node.js技术进行服务器端编程



3.3 JavaScript的作用

- 表单动态校验 (密码强度检测) (JS 产生最初的目的)
- 网页特效 , 如轮播图特效, 下拉菜单
- 服务端开发(Node.js)
- 桌面程序(Electron)
- App(Cordova)
- 控制硬件-物联网(Ruff)
- 游戏开发(cocos2d-js)

3.4 HTML/CSS/JS的关系

HTML/CSS标记语言--描述类语言

- HTML决定网页结构和内容(决定看到什么) ,相当于人的身体
- CSS 决定网页呈现给用户的模样(决定好不好看) ,相当于给人穿衣服、化妆

JS脚本语言--编程类语言

- 实现业务逻辑和页面控制(决定功能) ,相当于人的各种动作

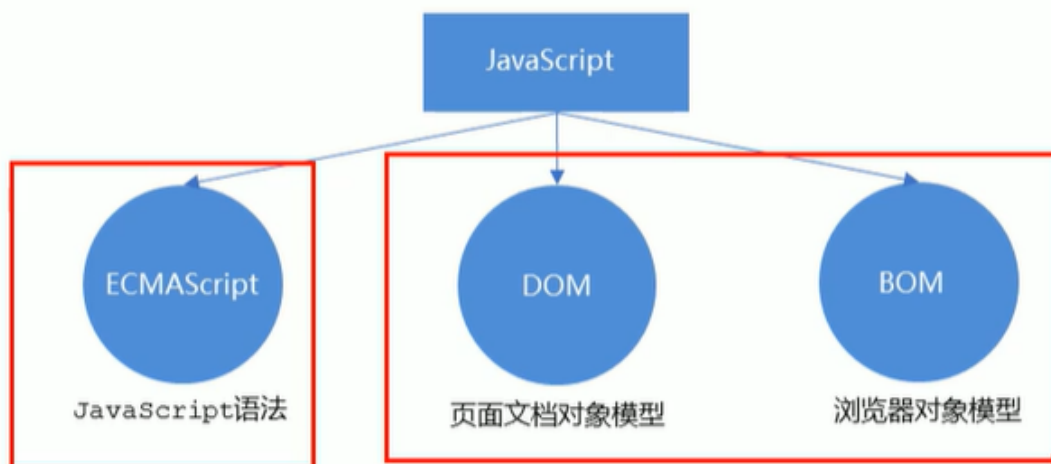
3.5浏览器执行JS简介

浏览器分成两部分:渲染引擎和JS引擎

- 渲染引擎**:用来解析HTML与CSS ,俗称内核,比如chrome浏览器的blink ,老版本的webkit
- JS引擎**:也称为JS解释器。 用来读取网页中的JavaScript代码, 对其处理后运行, 比如chrome浏览器的V8

浏览器本身并不会执行JS代码,而是通过内置JavaScript引擎(解释器)来执行JS代码。JS 引擎执行代码时逐行解释每一句源码(转换为机器语言),然后由计算机去执行,所以JavaScript语言归为脚本语言,会逐行解释执行。

3.6 JS的组成



3.6.1 ECMAScript

ECMAScript是由ECMA国际(原欧洲计算机制造商协会)进行标准化的一门编程语言,这种语言在万维网上应用广泛,它往往被称为JavaScript或JScript,但实际上后两者是ECMAScript语言的实现和扩展。

ECMAScript: ECMAScript规定了JS的编程语法和基础核心知识,是所有浏览器厂商共同遵守的一套JS语法工业标准。

3.6.2 DOM-文档对象模型

文档对象模型(Document Object Model ,简称DOM) ,是W3C组织推荐的处理可扩展标记语言的标准编程接口。通过DOM提供的接口可以对页面上的各种元素进行操作(大小、位置、颜色等)。

3.6.3BOM -浏览器对象模型

浏览器对象模型 (Browser Object Model, 简称BOM), 它提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。通过BOM可以操作浏览器窗口, 比如弹出框、控制浏览器跳转、获取分辨率等。

3.7 JS初体验

JS有3种书写位置, 分别为行内、内嵌和外部。(与css类似)

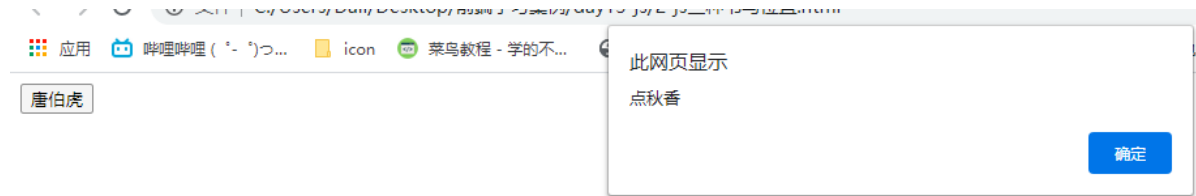
3.7.1行内式JS

```
<input type="button" value= "点我试试" onclick ="alert ('Hello world')"/> />
```

- 可以将单行或少量JS代码写在HTML标签的事件属性中(以on开头的属性), 如: onclick
- 注意单双引号的使用:在HTML中我们推荐使用双引号, JS中我们推荐使用单引号
- 可读性差,在html中编写JS大量代码时,不方便阅读;
- 引号易错, 引号多层嵌套匹配时,非常容易弄混;
- 特殊情况下使用

```
<!-- 1.行内式js, 直接写到元素的内部 -->
<input type="button" value="唐伯虎" onclick="alert('点秋香')">
```

点击按钮后, 弹出对话框



3.7.2内嵌JS

```
<script>
    alert( 'Hello world~!');
</script>
```

- 可以将多行JS代码写到<script>标签中
- 内嵌JS是学习时常用的方式

```
<!-- 2.内嵌式js -->
<script>
    alert('hello world')
</script>
```

此网页显示
hello world

确定

3.7.3外部JS文件

```
<script src= "my. js"></ script>
```

- 利于HTML页面代码结构化,把大段JS代码独立到HTML页面之外,既美观,也方便文件级别的复用
- 引用外部JS文件的script标签中间不可以写代码
- 适合于JS代码量比较大的情况

3.8 JS注释

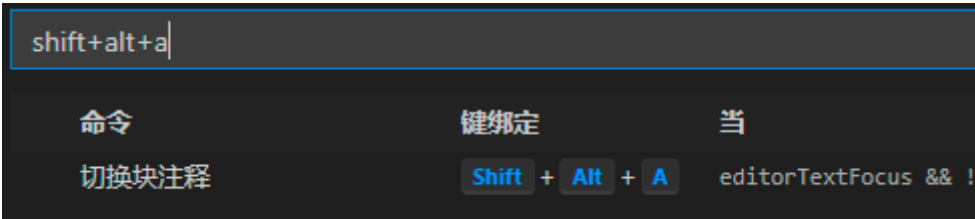
```
<script>
    // 1.单行注释 ctrl+/
    /* 2.多行注释
    默认快捷键 shift+alt+a
    可在vscode中修改多行注释的快捷键 */
</script>
```

vscode修改快捷键方法:

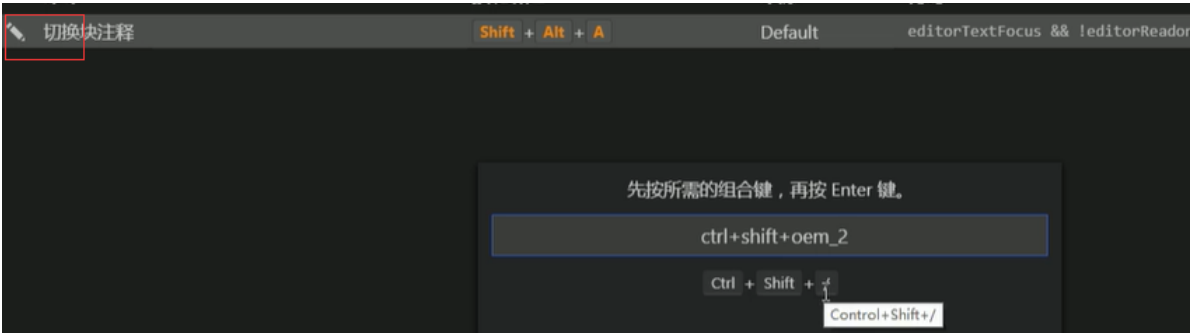
1.



2.里面搜索shift+alt+a



3.点击“笔”按弹窗说明进行修改



3.9 JS输入输出语句

为了方便信息的输入输出, JS中提供了一些输入输出语句 ,其常用的语句如下:

方法	说明	归属
alert(msg)	浏览器弹出警示框	浏览器
console.log(msg)	浏览器控制台打印输出信息	浏览器
prompt(info)	浏览器弹出输入框,用户可以输入	浏览器

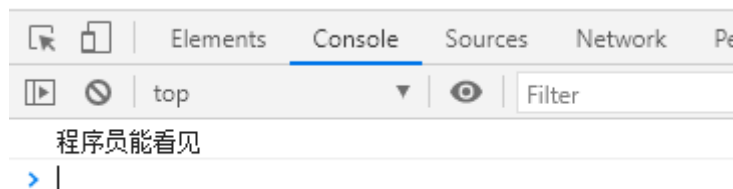
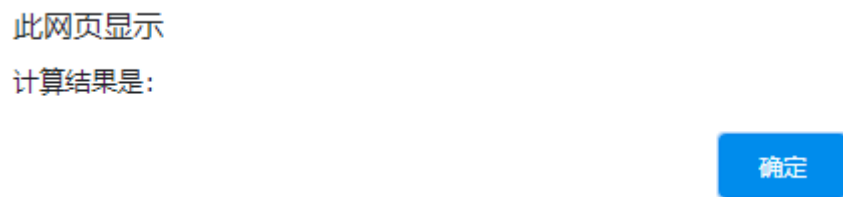
console意为控制台, log意为日志

案例代码:

```
<script>
  // 这是一个输入框
  prompt('请输入你的姓名: ');
  // alert弹出警示框, 输出框, 展示给用户
  alert('计算结果是: ');
  // console控制台输出, 给程序员测试用的
  console.log('程序员能看见');
</script>
```

注意: 不要把alert拼成alter.

实现结果:



4. 变量概述

4.1 什么是变量

通俗: 变量是用于存放数据的容器。我们通过变量名获取数据, 甚至数据可以修改。

4.2 变量在内存中的存储

本质: 变量是程序在内存中申请的一块用来存放数据的空间。

4.3 变量的使用

变量在使用时分为两步: 1. 声明变量 2. 赋值

4.3.1 声明变量

```
//声明变量
var age; // 声明一个名称为age的变量
```

●**var** 是一个JS关键字,用来声明变量(variable变量的意思)。使用该关键字声明变量后,计算机会自动为变量分配内存空间,不需要程序员管。

●age是程序员定义的变量名,我们要通过变量名来访问内存中分配的空间

●js(脚本语言)是弱类型语言, 变量的数据类型依赖它的值

●声明变量本质是去内存申请空间。

4.3.2 赋值

```
age = 10; //给age这个变量赋值为10
```

●=用来把右边的值赋给左边的变量空间中此处代表赋值的意思

●变量值是程序员保存到变量空间里的值

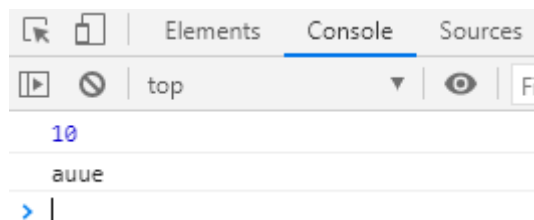
4.3.3 变量的初始化

```
var age = 18; // 声明变量同时赋值为18
```

声明一个变量并赋值,我们称之为**变量的初始化**。

```
<script>
    // 1.声明一个变量名为age
    var age;
    // 2.赋值
    age = 10;
    // 3.输出结果
    console.log(age);

    // 变量初始化
    var myname = 'auue';
    console.log(myname);
</script>
```

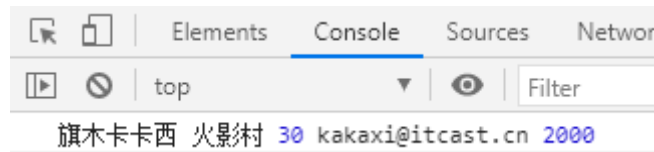


4.4 变量使用案例

1.使用变量存储以下信息并输出

我叫旗木卡卡西,我住在火影村,我今年30岁了,我的邮箱是kakaxi@itcast.cn,我的工资2000

```
<script>
    var myname = '旗木卡卡西', address = '火影村', age = 30, email =
    'kakaxi@itcast.cn', salary = 2000;
    console.log(myname, address, age, email, salary);
</script>
```



2.实现：弹出一个输入框,提示用户输入姓名；弹出一个对话框,输出用户刚才输入的姓名。

```
<script>
    // 1.用户输入姓名，并存储到myname的变量里
    var myname = prompt('请输入你的名字：');
    // 2.弹出提示框
    alert(myname)
</script>
```

此网页显示

请输入你的名字：

确定

取消

此网页显示

auue

确定

4.5变量语法扩展

4.5.1更新变量

一个变量被重新复赋值后,它原有的值就会被覆盖,变量值将以最后一次赋的值为准。

```
var age = 18;
age = 81;
//最后的结果就是81，因为18被覆盖掉了
```

4.5.2同时声明多个变量

同时声明多个变量时,只需要写一个var ,多个变量名之间使用英文逗号隔开。

```
var age = 10, name = 'zz', sex = 2;
```

4.5.3 声明变量特殊情况

情况	说明	结果
var age; console.log (age);	只声明不赋值	undefined
console.log(age)	不声明不赋值直接使用	报错
age = 10; console.log (age);	不声明只赋值	10

第三种虽然结果与预期一样，但不建议使用

4.6 变量命名规范

- 由字母(A-Za-z)、数字(0-9)、下划线(_)、美元符号(\$)组成,如: usrAge, num01
- 严格区分大小写。var app;和var App;是两个变量
- 不能以数字开头。18age 是错误的
- 不能是关键字、保留字。例如: var,for, while
- 变量名必须有意义。MMD BBD n | → age
- 遵守驼峰命名法。首字母小写,后面单词的首字母需要大写。myFirstName
- 推荐翻译网站:有道 爱词霸
- 不能使用'name'作为变量名，算关键字

5.数据类型简介

5.1 为什么需要数据类型

在计算机中，不同的数据所需占用的存储空间是不同的，为了便于把数据分成所需内存大小不同的数据，充分利用存储空间，于是定义了不同的数据类型。

5.2 变量的数据类型

变量是用来存储值的所在处,它们有名字和数据类型。变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。**JavaScript 是一种弱类型或者说动态语言**。这意味着不用提前声明变量的类型,只在程序运行过程中,类型会被自动确定。

```
<script>
  var num;//此时的num，我们不确定是属于什么数据类型的
  var num=10;//num属于数字型
  //js的变量数据只有在程序运行过程中，根据等号右侧的值来确定的
</script>
```

JavaScript拥有动态类型,同时也意味着相同的变量可用作不同的类型，可自由变化变量的类型:

```
var x=6;//x为数字类型
x = "Bill";//x为字符串类型
```

5.3 简单数据类型(基本数据类型)

JavaScript中的简单数据类型及其说明如下:

简单数据类型	说明	默认值
Number	数字型,包含整型值和浮点型值,如21、0.21	0
Boolean	布尔值类型, 如true、false, 等价于1和0	false
String	字符串类型, 如“张三”, 注意js里面, 字符串都带引号	""
Undefined	var a;声明了变量但没有给值, 此时a=undefined	undefined
Null	var a= null;声明了变量a为空值	null

5.3.1 数字型Number

JavaScript数字类型既可以用来**保存整数值,也可以保存小数**(浮点数)。

```
var age = 21; // 整数, 是数字型
var age = 21.3747; // 小数, 也是数字型
```

5.3.1.1 数字型进制

最常见的进制有二进制、八进制、十进制、十六进制。

```
// 1. 八进制数字序列范围: 0~7
var num1 = 07; // 对应十进制的7
var num2 = 019; // 对应十进制的19
var num3 = 08; // 对应十进制的8
// 2. 十六进制数字序列范围: 0~9以及A~F
var num = 0xA;
```

现阶段我们只需要记住,在JS中**八进制前面加0,十六进制前面加0x**

5.3.1.2 数字型范围

JavaScript中数值的最大和最小值

```
console.log(Number.MAX_VALUE); // 1. 7976931348623157e+308
console.log(Number.MIN_VALUE); // 5e-324
```

5.3.1.3 数字型三个特殊值

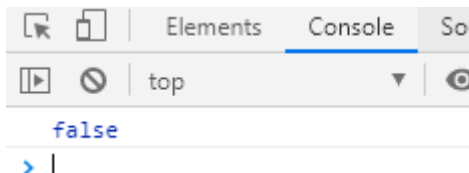
```
alert (Infinity); // Infinity, 无穷大
alert (-Infinity); // -Infinity, 无穷小
console.log(100-'na'); // NaN, 非数值
```

- Infinity, 代表无穷大,大于任何数值
- -Infinity, 代表无穷小,小于任何数值
- NaN, Not a number,代表一个非数值

5.3.1.4 isNaN()

用来判断一个变量是否为非数字的类型,返回true或者false

```
// 判断是否为数字, 是返回false, 不是则返回ture
console.log(isNaN(12));
```



5.3.2字符串类型String

字符串型可以是引号中的任意文本,其语法为双引号""和单引号''

```
//常见错误
var strMsg3 =我爱大肘子; //报错, 没使用引号, 会被认为是js代码, 但js没有这语法
```

因为HTML标签里面的属性使用的是双引号, JS这里我们更推荐使用单引号。

5.3.2.1字符串引号嵌套

JS可以用单引号嵌套双引号, 或者用双引号嵌套单引号(外双内单,外单内双)

```
var strMsg = '我是"高帅富"程序猿'; //可以用'包含'
var strMsg2 = "我是'高帅富'程序猿"; // 也可以用"包含'
var badquotes = 'what on earth?'; //报错, 不能单双引号搭配
```

5.3.2.2字符串转义符

类似HTML里面的特殊字符,字符串中也有特殊字符,我们称之为转义符。

注意: 转义字符需要写到引号内

转义符都是\开头的,常用的转义符及其说明如下:

转义符	解释说明
\n	换行符, n是newline的意思
\	斜杠\
斜杠+'	单引号
斜杠+"	双引号
\t	tab缩进
\b	空格, b是blank的意思

```
//字符串转义字符 都是用\开头但是这些转义字符写到引号里面
var str1 = "我是一个'高富帅'的\n程序员";
```

5.3.2.3字符串长度

字符串是由若干字符组成的,这些字符的数量就是字符串的长度。通过字符串的`length`属性可以获取整串的长度。

```
var str = "我是帅气多金的程序猿!";  
alert (str.length); //显示11
```

5.3.2.4字符串拼接

- 多个字符串之间可以使用 **+**进行拼接,其拼接方式为**字符串+任何类型=拼接之后的新字符串**
- 拼接前会把**与字符串相加的任何类型转成字符串**,再拼接成一个新的字符串
- 字符串可以和变量拼接

```
//1.1字符串"相加"  
alert( 'hello' + ' ' + 'world' ) ; // hello world  
//1.2数值字符串"相加"  
alert('100' + '100'); // 100100  
//1.3数值字符串+数值  
alert('11' + 12) ;// 1112  
//1.4字符串+变量  
var age =18;  
console.log('pink老师' + age + '岁啦'); // pink老师18岁啦
```

+号总结口诀:数值相加,字符相连, 变量不能添加引号,因为加引号的变量会变成字符串。

5.3.2.5小案例

实现: 弹出一个输入框,需要用户输入年龄,之后弹出一个警示框显示“您今年xx岁啦”(xx表示刚才输入的年龄)

```
var age = prompt('请输入您的年龄: ');  
var str='您今年已经'+ age + '岁了';  
alert(str);
```

5.3.3布尔型Boolean

布尔类型有两个值: true和false , 其中true表示真(对),而false表示假(错)。

布尔型和数字型相加的时候, true 的值为1 , false 的值为0。

```
console. log(true + 1); // 2  
console. log(false + 1); // 1
```

5.3.4 Undefined和Null

一个声明后没有被赋值的变量会有一个默认值 undefined (如果进行相连或者相加时,注意结果)

Undefined

```
var variable;
console.log (variable); // undefined
console.log('你好' + variable); // 你好undefined, 字符与任意相接仍为字符

console.log(11 + variable) ; // NaN, undefined与数字相加为NaN
console.log(true + variable) ; // NaN
```

Null

```
var vari = null;
console.log(vari + '你好'); // null你好, //字符与任意相接仍为字符

console.log(11 + vari) ;// 11
console.log(true + vari) ;// 1
```

注意区别两者与数字相加的结果

5.4获取变量数据类型

5.4.1获取检测变量的数据类型

typeof可用来求取检测变量的数据类型

语法: typeof空格+变量或typeof(变量)

```
var num = 10;
console.log(typeof num); //number
var str = 'auue';
console.log(typeof (str)); //string
var flag = true;
console.log(typeof flag); //Boolean
var vari = undefined;
console.log(typeof vari); //undefined
var timer = null;
console.log(typeof timer); //object
//输入框输入的值均为字符型
var age = prompt('请输入你的年龄');
console.log(typeof age); //string
```

注意:

1.null的数据类型是object对象

2.输入框输入的值都为字符型

3.也可以通过控制台输出结果字体颜色判断类型, 蓝色"10"为数字型, 红色为字符串型, 灰色为undefined或null型

```
10 "auue" true undefined null
```

5.4.2字面量

字面量是在源代码中一个固定值的表示法,通俗来说,就是字面量表示如何表达这个值。

- 数字字面量: 8,9,10
- 字符串字面量: '黑马程序员','大前端'
- 布尔字面量: true , false

5.5数据类型转换

5.5.1什么是数据类型转换

使用表单、prompt 获取过来的数据默认是字符串类型的,此时就不能直接简单的进行加法运算,而需要转换变量的数据类型。通俗来说,就是把一种数据类型的变量转换成另外一种数据类型。

5.5.2转换为字符串

方式	说明	案例
toString()	转成字符串	var num= 1; alert(num.toString());
String()强制转换	转成字符串	var num= 1; alert(String(num));
加号拼接字符串	和字符串拼接的结果都是字符串	var num = 1; alert(num+"我是字符串");

注意:

- toString() 和String()使用方式不一样。
- 三种转换方式,我们更喜欢用第三种加号拼接字符串转换方式,这一种方式也称之为隐式转换。

```
// 1.使用.toString(), 将数字型转换为字符型
var num = 10;
var str = num.toString();
console.log(str);
console.log(typeof str);
// 2.使用String()函数转换
console.log(String(num));
// 3.利用+拼接字符串（可使用空字符串），隐式转换。
console.log(num + '');
```

```
10
string
10
10
>
```

5.5.3转换为数字型

方式	说明	案例
parseInt(string)函数	将string类型转成整数数值型	parseInt('78')
parseFloat(string)函数	将string类型转成浮点数数值型	parseFloat('78.21')
Number()强制转换函数	将string类型转换为数值型	Number('12')
js隐式转换(- * /)	利用算术运算隐式转换为数值型	'12'-0

●注意parseInt和parseFloat单词的大小写,这2个是重点,最后一个也要注意

●隐式转换是我们在进行算数运算的时候,JS自动转换了数据类型

```
// var age = prompt('请输入你的年龄');
// 1.parseInt(变量), 将字符型转换为数字型, 且为整数
// console.log(parseInt(age));
console.log(parseInt('3.14'));// 3取整
console.log(parseInt('3.94'));//3取整, 而非四舍五入
console.log(parseInt('120px'));//120, 数字开头, 可自动去除数字后面的字符, 后续可用此去除px等单位
console.log(parseInt('120ss'));//120
console.log(parseInt('ww233ds'));//NaN, 字符开头直接pass
// 2.parseFloat(变量), 将字符型转换为数字型, 且为小数/浮点数
console.log(parseFloat('3.14'));// 3.14
console.log(parseFloat('3.94'));//3.94
console.log(parseFloat('120px'));//120
console.log(parseFloat('120ss'));//120
console.log(parseFloat('ww233ds'));//NaN
// 3.Number(变量)
console.log(Number('12'));//12
// 4.算数运算符- \ * 减除乘这三种, 隐式转换
console.log('123' - 0);//123
console.log('123' - '120');//3
```

5.5.3.1案例1 :计算年龄

此案例要求在页面中弹出一个输入框,我们输入出生年份后, 能计算出我们的年龄。

案例分析

- ①弹出一个输入框(prompt),让用户输入出生年份(用户输入)
- ②把用户输入的值用变量保存起来,然后用今年的年份减去变量值,结果就是现在的年龄(程序内部处理)
- ③弹出警示框(alert), 把计算的结果输出(输出结果)

```
var year = prompt('请输入您的出生年份');
var age = 2021 - year;//year字符型, 此处使用减号, 隐式转换
alert('您今年' + age + '岁了');
```

输入: 2000

此网页显示
您今年21岁了

确定

5.5.3.2 案例2 :简单加法器

计算两个数的值, 用户输入第一个值后, 继续弹出第二个输入框并输入第二个值, 最后通过弹出窗口显示出两次输入值相加的结果。

案例分析:

- ①先弹出第一个输入框,提示用户输入第一个值保存起来
- ②再弹出第二个框,提示用户输入第二个值保存起来
- ③把这两个值相加,并将结果赋给新的变量(注意数据类型转换)
- ④弹出警示框(alert) ,把计算的结果输出(输出结果)

```
var num1 = prompt('请输入第一个值: ');  
var num2 = prompt('请输入第二个值: ');  
var value = parseFloat(num1) + parseFloat(num2); //注意数字类型转换, 使用浮点  
数类型更好, 可能输入小数  
alert('两个值相加计算结果为: ' + value);
```

请输入第一个值:

2

确定

取消

请输入第二个值:

3.12

确定

取消

此网页显示

两个值相加计算结果为: 5.12

确定

5.5.4转换为布尔型

方式	说明	案例
Boolean()函数	其他类型转成布尔值	Boolean('true');

- 代表空、否定的值会被转换为false ,如"、0、NaN、null、undefined
- 其余值都会被转换为true

```
console.log (Boolean('')); // false
console.log (Boolean(0)); // false
console.log (Boolean(NaN)); // false
console.log (Boolean(null)) ; // false
console.log (Boolean(undefined)); // false

console.log (Boolean(12)); // true
console.log (Boolean('12')); // true
```

5.6课后作业

依次询问并获取用户的姓名、年龄、性别,并打印用户信息如图

```
var name = prompt('请输入你的姓名: ');
var age = prompt('请输入你的年龄: ');
var sex = prompt('请输入你的性别: ');
alert('姓名: ' + name + '\n年龄: ' + age + '\n性别: ' + sex)
```

注意: 换行符一定要写在引号内。

姓名: auue
年龄: 18
性别: 女

确定

6.运算符

运算符(operator)也被称为**操作符**,是用于实现赋值、比较和执行算数运算等功能的符号。

JavaScript中常用的运算符有:

- 算数运算符
- 递增和递减运算符
- 比较运算符
- 逻辑运算符
- 赋值运算符

6.1算数运算符

6.1.1算数运算符概述

概念:算术运算使用的符号, 用于执行两个变量或值的算术运算。

运算符	描述	实例
+	加	10+ 20= 30
-	减	10- 20=-10
/	除	10/ 20= 0.5
%	取余数(取模)	返回除法的余数9%2= 1
*	乘	10 * 10=100

注意:

1.运算符左右需要敲一个空格, 或者装插件保存时自动格式化。

2.除并非整除

3.如何判断一个数是否能够被整除: **使用%取余运算符, 余数是0就说明这个数能被整除**

4.算术运算符具有优先级,先乘除,后加减,有小括号先算小括号里面的

6.1.2浮点数的精度问题

浮点数值最高精度是17位小数,但在进行算术计算时其精确度远远不如整数。

```
//尽量避免浮点数运算
var result = 0.1 + 0.2;//结果不是0.3 ,而是: 0.30000000000000004
console.log(0.07*100) ;//结果不是7, 而是: 7.000000000000001
// 我们不能直接拿着浮点数来进行相比较是否相等
var num = 0.1 + 0.2;
console.log(num == 0.3); // false
```

6.1.3表达式和返回值

表达式:是由数字、运算符、变量等以能求得数值的有意义排列方法所得的组合

简单理解:是由数字、运算符、变量等组成的式子

表达式最终都会有一个结果,返回给我们,我们成为返回值

6.2递增和递减运算符

6.2.1递增和递减运算符概述

如果需要反复给数字变量添加或减去1 ,可以使用**递增(++)**和**递减(--)**运算符来完成。

在JavaScript中,递增(++)和递减(--)既可以放在变量前面,也可以放在变量后面。**放在变量前面时**,我们可以称为**前置递增(递减)运算符**; **放在变量后面时**,我们可以称为**后置递增(递减)运算符**。

注意:递增和递减运算符必须和变量配合使用。

6.2.2递增运算符

1.前置递增运算符

++num前置递增,就是自加1,类似于num=num+1。

使用口诀:先自加, 后返回值

2.后置递增运算符

num++后置递增，就是自加1 ,类似于num= num + 1

使用口诀:先返回原值,后自加

注意：

- 1.前置运算符和后置运算符的区别。
- 2.两个单独使用时效果一致。
- 3.开发时,大多使用后置递增/减,并且代码独占一行,例如: num++;或者num--;

6.3比较运算符

6.3.1比较运算符概述

概念:比较运算符(关系运算符)，是**两个数据进行比较时所使用的运算符**，比较运算后,会**返回一个布尔值**(true / false)作为比较运算的结果。

运算符名称	说明	案例	结果
<	小于号	1<2	true
>	大于号	1>2	false
>=	大于等于号(大于或者等于)	2>= 2	true
<=	小于等于号(小于或者等于)	3<= 2	false
==	判等号(会转型)	37 == 37	true
!=	不等号	37 != 37	false
===/!==	全等要求值和数据类型都一致	37 === '37'	false

6.3.2=小结

符号	作用	用法
=	赋值	把右边给左边
==	判断	判断两边值是否相等(注意此时有 隐式转换)
===	全等	判断两边的值和数据类型是否完全相同

注意区分==与===

```
// == 默认转换数据类型，会将字符串类型的数据自动转换成数字型，因此只要求值相等即可
console.log(18 == '18') ;//true
// === 全等要求值、数据类型都一模一样
console.log(18 === '18') ;//false
```

6.4逻辑运算符

6.4.1逻辑运算符概述

概念:逻辑运算符是用来进行布尔值运算的运算符,其返回值也是布尔值。后面开发中经常用于多个条件的判断。

逻辑运算符	说明	案例
&&	"逻辑与", 简称"与"and	true && false
	"逻辑或", 简称"或" or	true false
!	"逻辑非", 简称"非"not	! true

1.逻辑与&&

两边都是true才返回true ,否则返回false

2.逻辑或||

两边都为false才返回false ,否则都为true

3.逻辑非!

逻辑非(!)也叫作**取反符**,用来取一个布尔值相反的值,如true的相反值是false

6.4.2短路运算(逻辑中断)

短路运算的原理:当有多个表达式(值)时, **左边的表达式值可以确定结果时**,就不再继续运算右边的表达式的值。

1.逻辑与

- 语法:**表达式1 &&表达式2**
- 如果第一个表达式的值为真 ,则返回表达式2
- 如果第一个表达式的值为假 ,则返回表达式1

```
//注意: 如果值参与逻辑运算, 结果是值, 而不是布尔值
console.log(123 && 456); // 456
console.log(0 && 456); //0
```

2.逻辑或

- 语法:**表达式1 || 表达式2**
- 如果第一个表达式的值为真,则返回表达式1, **就不执行表达式2**
- 如果第一个表达式的值为假,则返回表达式2

```
console.log( 123 || 456 );// 123
console.log( 0 || 456 || 456 + 123) ;// 456
//逻辑中断很重要, 会影响我们程序运行结果
var num = 0;
console.log(123 || num++) ;
console.log (num) ;//0
```

6.5赋值运算符

概念:用来把数据赋值给变量的运算符。

赋值运算符	说明	案例
=	直接赋值	var usrName ='我是值';
+=、-=	加、减一个数后在赋值	var age= 10; age+=5; // 15
=、/=、%=	乘、除、取模后在赋值	var age= 2; age=5;// 10

```
var age=10;
age += 5; //相当于age=age+5;
age -= 5; //相当于age=age-5;
age *= 10; //相当于age=age*10;
```

6.7运算符优先级

优先级	运算符	顺序
1	小括号	()
2	一元运算符	++、--、!
3	算数运算符	先*/%后+ -
4	关系运算符	>、>=、<、<=
5	相等运算符	==、!=、===、!==
6	逻辑运算符	先&&后
7	赋值运算符	=
8	逗号运算符	,

一元运算符：只有一个操作数，如++num,只有num一个操作数

- 一元运算符里面的逻辑非优先级很高
- 逻辑与比逻辑或优先级高

7.流程控制

7.1流程控制概述

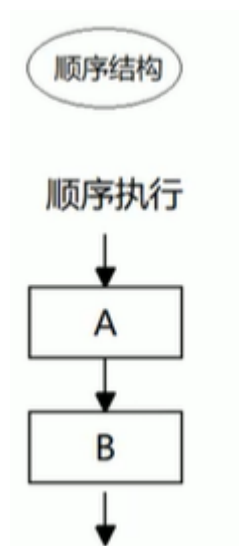
在一个程序执行的过程中,各条代码的执行页序对程序的结果是有直接影响的。很多时候我们要通过控制代码的执行顺序来实现我们要完成的功能。

简单理解:流程控制就是来控制我们的代码按照什么结构顺序来执行

流程控制主要有三种结构,分别是**顺序结构**、**分支结构**和**循环结构**,这三种结构代表三种代码执行的顺序。

7.2顺序流程控制

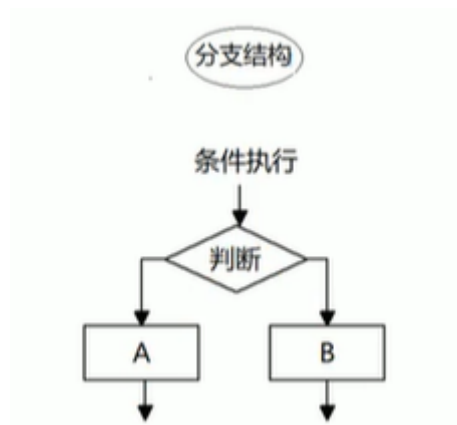
顺序结构是程序中最简单、最基本的流程控制,它没有特定的语法结构,程序会按照代码的先后顺序,依次执行。程序中大多数的代码都是这样执行的。



7.3分支流程控制if语句

7.3.1分支结构

由上到下执行代码的过程中,根据不同的条件,执行不同的路径代码(执行代码多选一的过程),从而得到不同的结果。



JS语言提供了两种分支结构语句

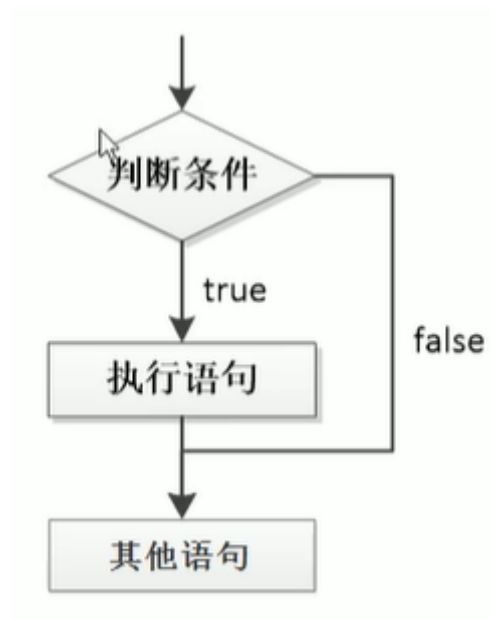
- if语句
- switch 语句

7.3.2 if语句

1. if语法结构

```
if (条件表达式) {  
    //条件成立执行语句  
}  
//执行思路: 如果if里面的条件表达式结果为真true, 则执行大括号里面的执行语句;  
//如果if条件表达式结果为假, 则不执行大括号里面的语句, 执行if语句后面的代码
```

2. 执行流程



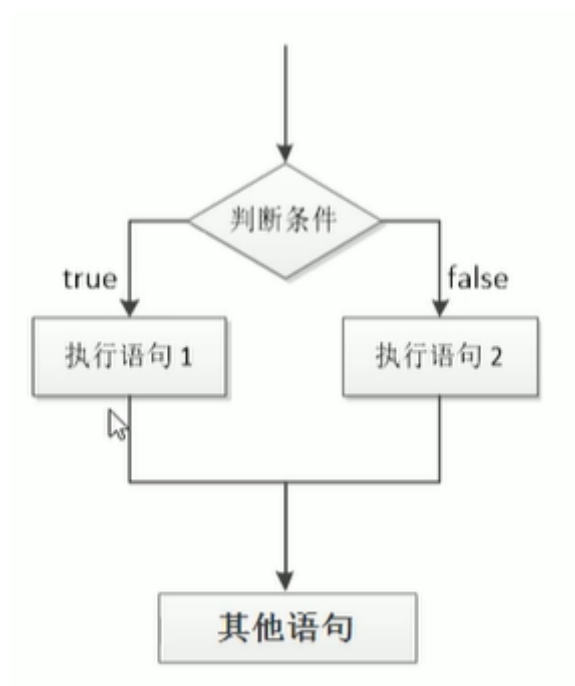
7.3.3 if else语句(双分支语句)

1.语法结构

```
//条件成立执行if里面代码,否则执行else里面的代码
if (条件表达式) {
    // [如果]条件成立执行的代码
} else {
    // [否则]执行的代码
}
```

else后面直接跟大括号

2.执行流程



案例1-判断是否是闰年

算法:能被4整除且不能整除100的为闰年(如2004年就是闰年, 1901年不是闰年)或者能够被400整除的就是闰年

```
var year = prompt("请输入年份: ");
// 运算符会自动将字符型转换为数字型, 不用进行转换
if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {
    alert(year + "年是闰年");
}
else {
    alert(year + "年是平年");
}
```

案例2-判断是否中奖

算法: 接收用户输入的姓名,来判断是否中奖, 如果输入的是刘德华,则提示中了5块钱,否则提示没有中奖。

```
var name = prompt("请输入你的名字: ");
if (name == "刘德华") {
    alert("恭喜您中奖了");
}
else {
    alert("很遗憾, 您没有中奖");
}
```

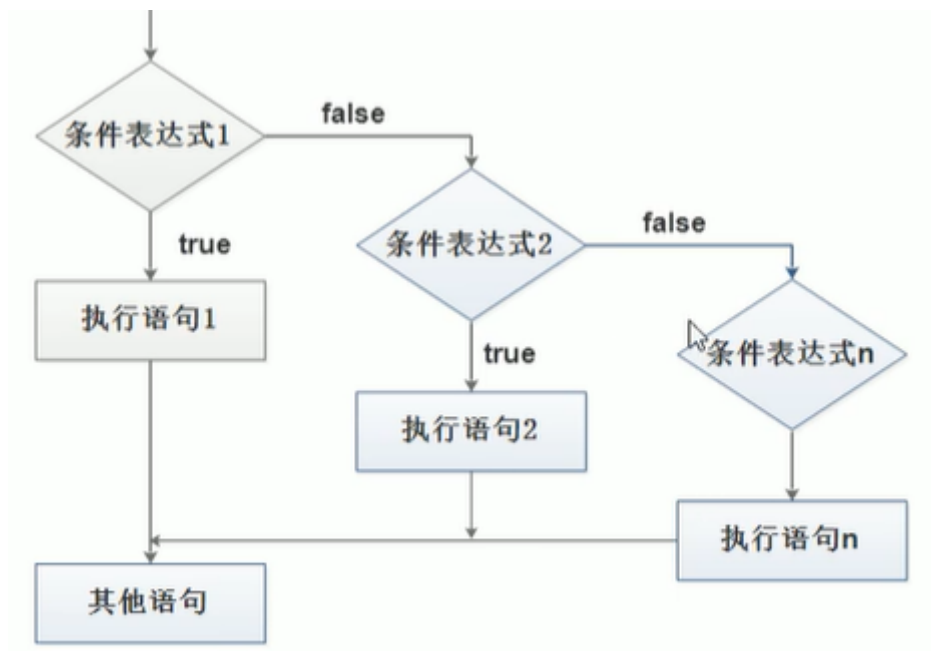
7.3.4 if else if (多分支语句)

1.语法结构

```
if (条件表达式1) {
    //语句1;
} else if (条件表达式2) {
    //语句2;
} else if (条件表达式3) {
    //语句3;
} else {
    //最后的语句;
}
```

注意: else if中间有个空格

2.执行流程



7.4 三元表达式

1.语法结构

条件表达式 ? 表达式1 : 表达式2

2.执行思路

如果条件表达式结果为真，则返回表达式1的值；如果条件表达式结果为假，则返回表达式2的值。

案例：数字补0

用户输入数字,如果数字小于10 ,则在前面补0,比如01 , 09 , 如果数字大于10 ,则不需要补， 比如20.

```
// 数字小于10，给前面补0，数字大于10则不做操作
var time = prompt("请输入0-59之间的一个数：");
//三元表达式更方便，简单
var result = time < 10 ? '0' + time : time;
alert(result);
// 补0，采取'0'拼接
```

注意：补0的方式采用字符0进行拼接

7.5分支流程控制switch语句

switch语句也是多分支语句,它用于基于不同的条件来执行不同的代码。当要针对变量设置一系列的**特定值**的选项时,就可以使用switch。

1.语法结构

```
// 2. 语法结构switch转换、开关,case例子或者选项的意思
switch(表达式) {
    case value1:
        执行语句1;//表达式等于value1 时要执行的代码
        break;
    case value2:
        执行语句2;// 表达式 等于value2 时要执行的代码
        break;
    ...
    default :
        执行最后的语句;//表达式不等于任何一个value时要执行的代码
}
```

注意:

- 1.case与value之间有个空格。
- 2.开发中,表达式我们经常写成变量,如num
- 3.变量的值与case里面的值相匹配时,是**全等匹配**,如果是字符型,写value时一定要加上引号。
- 4.尽量写break,如果当前的case里面没有break,则不会退出switch,是继续执行下一个case直到遇到下一个break。

7.6 switch语句和if else if语句的区别

- ①一般情况下,它们两个语句可以相互替换
- ②switch..case 语句通常处理case为比较确定值的情况,而if else if..语句更加灵活,常用于范围判断(大于、等于某个范围)
- ③switch 语句进行条件判断后直接执行到程序的条件语句,效率更高。而if..else 语句有几种条件,就得判断多少次。
- ④当分支比较少时,if... else语句的执行效率比switch语句高。
- ⑤当分支比较多时,switch语句的执行效率比较高,而且结构更清晰。

7.7作业

- ①1.判断时间阶段。比如用户输入12点弹出中午好,用户输入18点弹出傍晚好,用户输入23点弹出深夜好
- ②2. 比较两个数的最大值(用户依次输入2个值,最后弹出最大的那个值)——三元表达式
- ③3. 用户输入一个数,来判断是奇数还是偶数
- ④4.根据用户输入的数值(数字1到数字7),返回星期几——switch
- ⑤5. 接收班长口袋里的钱数?若大于等于2000,请大家吃西餐。若小于2000,大于等于1500,请大家吃快餐。若小于1500,大于等于1000,请大家喝饮料。若小于1000,大于等于500,请大家吃棒棒糖。否则提醒班长下次把钱带够
- ⑥6. 分数转换给一个分数,判定等级。大于等于90 A,大于等于80小于90 B,大于等于70小于80 C,大于等于60小于70 D,小于60 E

1.

```
// 5-10点为早上, 10-12为中午, 12-16为下午, 17-21为傍晚, 21-24为深夜
```

```
var time = prompt("请输入1-24时间点: ");  
if (time >= 5 && time <= 10) {  
    alert("早上好");  
} else if (time <= 12) {  
    alert("中午好");  
} else if (time <= 16) {  
    alert("下午好");  
} else if (time <= 21) {  
    alert("傍晚好");  
} else {  
    alert("深夜好");  
}
```

2.

```
var num1 = prompt("请输入第一个数: ");  
var num2 = prompt("请输入第二个数: ");  
var max = parseFloat(num1) > parseFloat(num2) ? num1 : num2;  
alert(max + "该值最大");  
//三元表达式比大小没有隐式转换, 需要我们强制转换
```

剩下代码在案例里。

8循环

流程控制之一

循环目的

●在实际问题中,有许多具有规律性的重复操作,因此在程序中要完成这类操作就需要重复执行某些语句

在js中,主要有三种类型的循环语句:

- for 循环, 主要
- while 循环
- do..while 循环

8.1 for循环

在程序中, 一组被重复执行的语句被称之为**循环体**,能否继续重复执行,取决于循环的**终止条件**。由循环体及循环的终止条件组成的语句,被称之为循环语句

1. for语法结构

```
for (初始化变量; 条件表达式; 操作表达式) {  
    //循环体  
}  
// 3. 初始化变量是用var声明的一个普通变量, 通常用于作为计数器使用
```

如:

```
for (var i = 1; i <= 100; i++){  
    console.log( '你好' );  
}
```

注意声明变量var

2.循环语句嵌套分支语句

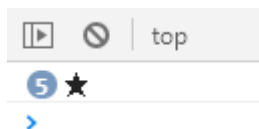
8.1.1案例：求学生成绩

要求用户输入班级人数,之后依次输入每个学生的成绩,最后打印出该班级总的成绩以及平均成绩。

```
var num = prompt("请输入班级人数: ");
var sum = 0, avg = 0;
for (var i = 1; i <= num; i++) {
    var score = prompt("请输入第" + i + "个人的成绩: ");
    sum += parseFloat(score); //记得数字转换
}
avg = sum / num
alert("班级总成绩: " + sum + "\n班级平均分: " + avg)
```

8.1.2案例：一行打印5个小星星

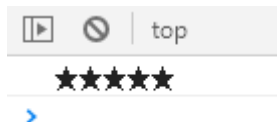
```
for (var i = 1; i <= 5; i++) {
    console.log('★');
}
```



结果：星星并非在一行上。

采取追加字符串的方式让星星字符一行显示。

```
var str = '';
for (var i = 1; i <= 5; i++) {
    str += '★';
}
console.log(str);
```



8.2断点调试

断点调试是指自己在程序的某一行设置一个断点,调试时,程序运行到这一行就会停住,然后你可以一步一步往下调试,调试过程中可以看各个变量当前的值,出错的话,调试到出错的代码行即显示错误,停下。

断点调试可以帮助我们观察程序的运行过程

浏览器中按F12--> sources -->找到需要调试的文件-->在程序的某一行设置断点

Watch:监视,通过watch可以监视变量的值的变化,非常的常用。

F11 (或右上角有个向下的箭头) :程序单步执行,让程序一行一行的执行,这个时候,观察watch中变量的值的变化。

代码调试的能力非常重要,只有学会了代码调试,才能学会自己解决bug的能力。

8.3双重for循环概述

很多情况下,单层for循环并不能满足我们的需求,比如我们要打印一个5行5列的图形、打印一个倒直角三角形等,此时就可以通过循环嵌套来实现。

循环嵌套是指在一个循环语句中再定义一个循环语句的语法结构,例如在for循环语句中,可以再嵌套一个for循环,这样的for循环语句我们称之为**双重for循环**。

8.3.1打印五行五列星星

核心:

1.内层循环负责一行打印五个星星

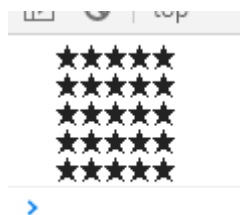
2.外层循环负责打印五行

注意:

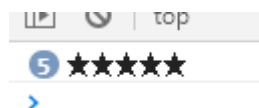
如何实现一行打印完5个星星然后换行打印?

解决方法: 外循环内添加换行符

```
var str = '';
for (var i = 1; i <= 5; i++) {
  for (var j = 1; j <= 5; j++) {
    str += '★';
  }
  // console.log(str);
  // str = '';
  str += '\n';
}
console.log(str);
```



若是采取注释内的方法, 结果: 与预期不太一样



8.3.2打印n行m列的星星

要求用户输入行数和列数,之后在控制台打印出用户输入行数和列数的星星。

```

var n = prompt("请输入打印的行数: ");
var m = prompt("请输入打印的列数: ");
var str = '';
for (var i = 1; i <= n; i++) {
    for (var j = 1; j <= m; j++) {
        str += '★';
    }
    str += '\n'
}
console.log(str);

```

行数: 3, 列数: 2

```

      ★★
      ★★
      ★★
      ★★

```

8.3.3打印倒三角形星星

```

var str = '';
//方法1
/* for (var i = 10; i >= 1; i--) {
    for (var j = 1; j <= i; j++) {
        str += '★';
    }
    str += '\n';
} */
//方法2
for (var i = 1; i <= 10; i++) {
    for (var j = i; j <= 10; j++) {
        str += '★';
    }
    str += '\n';
}
console.log(str);

```

```


★★★★★★★★★★
★★★★★★★★★★
★★★★★★★★★★
★★★★★★★★★★
★★★★★★★
★★★★★★
★★★★★
★★★★
★★★
★★
★

```

方法1: 通过控制i变小, 控制j的循环条件, 让内循环次数减少

方法2: 增大内循环的初始变量j, 让内循环次数减少

8.3.4打印九九乘法表



1x1=1									
1x2=2	2x2=4								
1x3=3	2x3=6	3x3=9							
1x4=4	2x4=8	3x4=12	4x4=16						
1x5=5	2x5=10	3x5=15	4x5=20	5x5=25					
1x6=6	2x6=12	3x6=18	4x6=24	5x6=30	6x6=36				
1x7=7	2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49			
1x8=8	2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64		
1x9=9	2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81	

案例分析:

- ①一共有9行,但是每行的个数不一样,因此需要用到双重for循环
- ②外层的for循环控制行数i,循环9次,可以打印9行
- ③内层的for循环控制每行公式j
- ④核心算法:每一行公式的个数正好和行数一致, $j \leq i$;
- ⑤每行打印完毕,都需要重新换一行

注意:

1. 让输出在一行内显示,采取追加字符串的方式,外层循环添加换行符。
- 2.为了美观,输出时把列写在前面 $j*i$
- 3.为了美观,可以在每个输出后添加一个tab间隔t

```
var str = '';
for (var i = 1; i < 10; i++) {
    for (var j = 1; j <= i; j++) {
        str += j + 'x' + i + '=' + i * j + '\t';
    }
    str += '\n';
}
console.log(str);
```

```
1x1=1
1x2=2  2x2=4
1x3=3  2x3=6  3x3=9
1x4=4  2x4=8  3x4=12  4x4=16
1x5=5  2x5=10  3x5=15  4x5=20  5x5=25
1x6=6  2x6=12  3x6=18  4x6=24  5x6=30  6x6=36
1x7=7  2x7=14  3x7=21  4x7=28  5x7=35  6x7=42  7x7=49
1x8=8  2x8=16  3x8=24  4x8=32  5x8=40  6x8=48  7x8=56  8x8=64
1x9=9  2x9=18  3x9=27  4x9=36  5x9=45  6x9=54  7x9=63  8x9=72  9x9=81
```

8.4 for循环小结

- for 循环可以重复执行某些相同代码
- for循环可以重复执行些不同的代码,因为我们有计数器
- for 循环可以重复执行某些操作,比如算术运算符加法操作
- 随着需求增加,双重for循环可以做更多、更好看的效果

- 双重for循环,外层循环一次,内层for循环全部执行
- for 循环是循环条件和数字直接相关的循环
- 分析要比写代码更重要

8.5 while循环

while语句可以在条件表达式为真的前提下, 循环执行指定的一段代码,直到表达式不为真时结束循环。
while语句的语法结构如下:

```
while (条件表达式) {  
    //循环体代码  
}
```

执行思路:

①先执行条件表达式,如果结果为true ,则执行循环体代码;如果为false ,则退出循环,执行后面代码

②再执行循环体代码

③循环体代码执行完毕后,程序会继续判断执行条件表达式,如条件仍为true ,则会继续执行循环体,直到循环条件为false时,整个循环过程才会结束

注意: 循环体代码里面应有操作表达式, 完成计数器的更新, 防止死循环

案例: 弹出一个提示框, '你爱我吗?', 如果输入我爱你, 就提示结束; 否则, 一直询问。

```
var message = prompt( '你爱我吗?');  
while (message !== '我爱你') {  
    message = prompt('你爱我吗?');//注意message再次赋值新输入的值  
    alert('我也爱你啊! ' );  
}
```

8.6 do while循环

do... while语句其实是while语句的一个变体。该循环会先执行次代码块,然后对条件表达式进行判断,如果条件为真,就会重复执行循环体,否则退出循环。

do... while语句的语法结构如下:

```
do {  
    //循环体代码-条件表达式为true时重复执行循环体代码  
} while (条件表达式) ;
```

执行思路:

①先执行一次循环体代码

②再执行条件表达式,如果结果为true ,则继续执行循环体代码,如果为false ,则退出循环,继续执行后面代码

注意:先再执行循环体,再判断,我们会发现d...while循环语句至少会执行一次循环体代码

案例: 弹出一个提示框, 你爱我吗? 如果输入我爱你, 就提示结束, 否则, 一直询问。

```
do {  
    var message = prompt( '你爱我吗?');  
} while (message !== '我爱你')
```

与while循环相比，少写一条提示框语句。有时候do..while更简单

8.7 循环小结

- JS中循环有for、while、do while
- 三个循环很多情况下都可以相互替代使用
- 如果是用来计次数,跟数字相关的,三者使用基本相同,但是我们更喜欢用for
- while 和do...while可以做更复杂的判断条件,比for循环灵活一些
- while 和do...while执行顺序不一样, while 先判断后执行, do..while先执行一次,再判断执行
- while 和do..while执行次数不一样, do...while至少会执行一次循环体,而while可能一次也不执行
- 实际工作中,我们更常用for循环语句, 它写法更简洁直观,所以这个要重点学习

8.8 continue break

8.8.1 continue关键字

continue关键字用于立即跳出本次循环，继续下一次循环(本次循环体中continue之后的代码就会少执行一次)

案例：求1-100之间除了能被7整除之外的整数和

```
var sum = 0;
for (var i = 1; i <= 100; i++){
    if (i % 7 == 0){
        continue;
    }
    sum += i;
}
console.log(sum);
```

8.8.2 break关键字

break关键字用于立即跳出整个循环(循环结束)。

8.9 作业

- 1.求1-100之间所有数的总和与平均值
- 2.求1-100之间所有偶数的和
- 3.求100以内7的倍数的总和
- 5.使用for循环打印直角三角形'★'
- 6.接收用户输入的用户名和密码,若用户名为"admin",密码为"123456",则提示用户登录成功!否则,让用户一直输入。
- 7.求整数1 ~ 100的累加值,但要求跳过所有个位为3的数[用continue实现]。
- 8.

小组项目：简易ATM

此网页显示

请输入您要的操作:

1. 存钱
2. 取钱
3. 显示余额
4. 退出

- 里面现存有 100 块钱。
- 如果存钱，就用输入钱数加上先存的钱数，之后弹出显示余额提示框
- 如果取钱，就减去取的钱数，之后弹出显示余额提示框
- 如果显示余额，就输出余额
- 如果退出，弹出退出信息提示框

9.需求：有一群人，如果3个人站一排多出一人，如果4个人站一排 多出来2个人；如果5人一排，多出3人。请问有多少人。（使用穷举法）

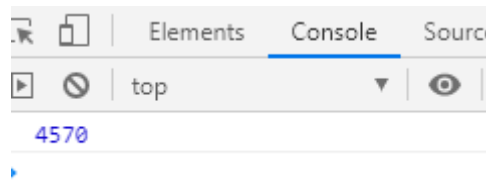
```
// 5使用for循环打印三角形
var str = '';
for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= i; j++) {
        str += '★';
    }
    str += '\n';
}
console.log(str);
```



// 6.接收用户输入的用户名和密码,若用户名为"admin" ,密码为"123456" , 则提示用户登录成功!否则,让用户一直输入。

```
var usrName = prompt('请输入用户名: ');
var psw = prompt('请输入密码: ');
// while (msg != 'admin' || pwd != '123') 或
while (!(usrName == 'admin' && psw == '123456')) {
    usrName = prompt('请输入用户名: ');
    psw = prompt('请输入密码: ');
}
alert('登录成功');
```

```
// 7.求整数1 ~ 100的累加值,但要求跳过所有个位为3的数[用continue实现]。
var sum = 0;
for (var i = 1; i <= 100; i++) {
    if (i % 10 == 3) {
        continue;
    }
    sum += i;
}
console.log(sum);
```



```
var money = 100, temp = 0;
var num = prompt('请输入您需要的操作: \n1.存钱\n2.取钱\n3.显示余额\n4.退出');
while (num != 4) {
    switch (num) {
        case '1':
            temp = prompt('请输入您存钱数: ');
            money += parseFloat(temp); //注意转换数字类型
            alert('您的余额: ' + money);
            break;
        case '2':
            temp = prompt('请输入您取钱数: ');
            money -= parseFloat(temp);
            alert('您的余额: ' + money);
            break;
        case '3':
            alert('您的余额: ' + money);
            break;
        case '4':
            alert('您已退出');
            break;
        default:
            alert('输入错误');
    }
    num = prompt('请输入您需要的操作: \n1.存钱\n2.取钱\n3.显示余额\n4.退出');
}
```

注意:

- 1.为了让取钱->显示余额后再次出现初始窗口,需要使用while语句,直至选择4退出。
- 2.转换数字类型,加法没有隐式转换

```
// 穷举: 从1遍历到无穷大,找出符合条件的
// 需求: 有一群人,如果3个人站一排多出一人,如果4个人站一排 多出来2个人;如果5人一
拍,多出3人。
// 请问: 这群人的数量
for (var i = 1; i <= Infinity; i++) {
    // 假设总的人数就是 i
    if (i % 3 == 1 && i % 4 == 2 && i % 5 == 3) {
        console.log("人数为" + i);
        break;
    }
}
```

9.命名规范以及语法规范

1.标识符命名规范

- 变量、函数的命名必须要有意义
- 变量的名称一般用名词
- 函数的名称一般用动词

2.操作符规范

```
//操作符的左右两侧各保留一个空格
for (var i = 1;i <= 5; i++) {
    if(i == 3){
        break; //直接退出整个for循环,跳到整个for循环下面的语句
    }
    console.log( '我正在吃第' + i + '个包子呢');
}
```

3.单行注释规范

```
for(var i = 1;i <= 5; i++){
    if( i == 3){
        break; // 单行注释前面注意有个空格
    }
    console.log( '我正在吃第' + i + '个包子呢');
}
```

4.其他规范

```
if(true){
}

for(var i = 0; i <= 100; i++){
}
```

2、3、4点规范，安装自动格式化文档插件后保存时自动规范。

10.数组

10.1数组的概念

数组是指**一组数据的集合** ,其中的每个数据被称作**元素**,在数组中可以**存放任意类型**的元素。数组是一种将**一组数据**存储在单个变量名下的优雅方式。

10.2数组的创建方式

JS中创建数组有两种方式:

- 利用new创建数组
- 利用数组字面量创建数组，更常用

10.2.1利用new创建数组

```
var 数组名 = new Array() ;  
var arr = new Array(); // 创建一个新的空数组
```

```
var arr1 = new Array(); // 创建一个空数组  
console.log(arr1);  
var arr2 = new Array(2); // 创建一个长度为2的数组，数组元素都为空  
console.log(arr2);  
var arr3 = new Array(2, 3); // 等价于[2, 3]，数组长度为2，数组元素是2， 3  
console.log(arr3);
```

```
▶ []  
▶ (2) [empty × 2]  
▶ (2) [2, 3]
```

- 注意Array(), A要大写

10.2.2利用数组字面量创建数组

```
// 1. 使用数组字面量方式创建空的数组  
var 数组名 = [];  
// 2. 使用数组字面量方式创建带初始值的数组  
var 数组名 = ['小白', '小黑', '大黄', '瑞奇'];
```

- 数组的字面量是方括号[]
- 声明数组并赋值称为数组的初始化
- 这种字面量方式也是我们以后最多使用的方式

10.3数组元素的类型

数组中可以存放任意类型的数据,例如字符串、数字、布尔值等。

```
var arrStus = ['小白', 12, true, 28.9] ; // 里面的数据，如12，我们称为数组元素
```

10.4获取数组元素

10.4.1数组的索引

索引(下标): 用来访问数组元素的序号(数组下标从0开始)。

数组可以通过**索引**来访问、设置、修改对应的数组元素,我们可以通过“**数组名[索引]**” (索引号从0开始)的形式来获取数组中的元素。

这里的**访问**就是获取得到的意思。

10.4.2遍历数组

遍历:就是把数组中的每个元素从头到尾都访问一次。

通过for循环遍历，**i是计数器,当索引号使用**，**arr[i] 是数组元素第i个数组元素**

数组长度: 使用“**数组名.length**”可以访问数组元素的数量(数组长度)，并动态检测元素个数。

注：数组长度是元素个数，与索引号不一样

```
var sum = 0;
for (var i = 0; i < arr.length; i++) {
    sum += arr[i]; //我们加的是数组元素arr[i]
}
```

10.4.3案例

案例1：求数组中的最大值。

```
var arr = [2, 6, 1, 77, 52, 34, 4];
var max = arr[0]; //默认最大值取数组中的第一个元素，而不是取一个固定值
for (var i = 0; i < arr.length; i++) {
    if (arr[i] > max) {
        max = arr[i];
    }
}
console.log('数组中最大值为' + max);
```

案例2: 数组转换为分割字符串

要求:将数组['red', 'green', 'blue', 'pink']转换为字符串,并且用|或其他符号分割

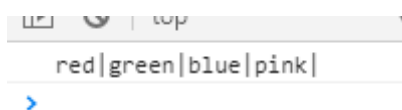
输出: 'red|green|blue|pink|'

案例分析

- ①需要一个新变量用于存放转换完的字符串str.
- ②遍历原来的数组,分别把里面数据取出来,加到字符串里面。
- ③同时在后面多加一个分隔符。

注意：定义一个‘sep’专门用来放置分隔符，若分割符变更，更改更方便。

```
var arr = ['red', 'green', 'blue', 'pink'];
var str = '';
var sep = '|';
for (var i = 0; i < arr.length; i++) {
    str += arr[i] + sep;
}
console.log(str);
```



```
red|green|blue|pink|
```

10.5数组中新增元素

可以通过**修改length长度**以及**索引号**增加数组元素

10.5.1通过修改length长度新增数组元素

- 可以通过修改length长度来实现数组扩容的目的
- length 属性是可读写的

```
var arr = ['red', 'green', 'blue', 'pink'];
arr.length = 6;
console.log(arr) ;
console.log(arr[4]);
console.log(arr[5]) ;
```

```
► (5) ["red", "green", "blue", empty × 2]
undefined
undefined
```

其中索引号是4, 5的空间没有给值,就是声明变量未给值,默认值就是undefined。

10.5.2通过修改数组索引新增数组元素

- 可以通过修改数组索引的方式追加或修改数组元素
- 不能直接给数组名赋值,否则会覆盖掉以前的数据 (也就是js的特点, 动态定义数组类型, 根据赋值来确定其数据类型)

```
var arr = ['red', 'green', 'blue', 'pink'];
arr[4] = 'hotpink' ;
console.log(arr) ;
```

这种方式也是我们最常用的一种方式。

10.6案例

案例1：筛选数组

要求:将数组[2, 0, 6, 1, 77, 0, 52, 0, 25, 7]中大于等于10的元素选出来,放入新数组。

案例分析

- ①声明一个新的数组用于存放新数据newArr。
- ②遍历原来的旧数组,找出大于等于10的元素。
- ③依次追加给新数组newArr。

```
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = [];
// 新数组索引号
// 方法1, 添加一个新变量j实现
/* var j = 0;
for (var i = 0; i < arr.length; i++) {
    if (arr[i] >= 10) {
        newArr[j] = arr[i];
        j++;
    }
} */
// 方法2, 不增加新变量, 通过newArr.length变更索引值, 初始是空数组, 数组长度也就为0
```

```

    for (var i = 0; i < arr.length; i++) {
        if (arr[i] >= 10) {
            newArr[newArr.length] = arr[i];
        }
    }
    console.log(newArr);

```

j替换成arr.length

注意第二种方法，积累好的思路

案例2:删除指定数组元素

要求:将数组[2,0,6, 1, 77, 0, 52, 0, 25, 7]中的0去掉后,形成一个不包含0的新数组。

案例分析:

先原数组上修改方法:

- ①遍历数组找到数组元素为0
- ②找到后，将其后面数组元素都前移一个位置，达到删除效果
- ③删除后，该数组长度-1

注意：数组长度-1，在**删除元素（前移完毕后）再长度-1**；若先长度-1后删除元素则不对，

```

var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
for (var i = 0; i < arr.length; i++) {
    if (arr[i] == 0) {
        for (var j = i; j < arr.length; j++) {
            arr[j] = arr[j + 1];
        }
        arr.length--;
    }
}
console.log(arr);

```

方法2: 新建一个数组，存放不为0的数组元素

- ①遍历数组找到数组元素不为0
- ②找到后，放入新数组中

```

var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];
var newArr = []
for (var i = 0; i < arr.length; i++) {
    if (arr[i] != 0) {
        newArr[newArr.length] = arr[i];
    }
}
console.log(arr);

```

案例3:翻转数组

要求:将数组['red', 'green', 'blue', 'pink', 'purple']的内容反过来存放。

输出: ['purple', 'pink', 'blue', 'green', 'red']

案例分析:

在原数组上更改:

- ①将前半数组元素与后半数组元素交换位置,如第一个元素与最后一个元素交换位置
- ②控制这个前半,即交换的元素所在索引号须小于数组的长度(不能等于,数组长度为偶数时,若等于会多交换一次)

```
var arr = ['red', 'green', 'blue', 'pink', 'purple'];
var temp ;
for (var i = 0; i < arr.length / 2; i++) {
    //第一个元素与最后一个元素交换位置,注意最后一个元素的索引号
    temp = arr[i];
    arr[i] = arr[arr.length - i - 1];
    arr[arr.length - i - 1] = temp;
}
console.log(arr);
```

▼ (5) ["purple", "pink", "blue", "green", "red"] ⓘ

注意:

- 1.后半元素的索引号, arr.length - i - 1, 不要忘了-1, 因为i是从0开始的
- 2.temp不需要定义为空字符, 后续赋值会自动确定其数据类型

案例4:数组排序(冒泡排序)

冒泡排序:是一种算法,把一系列的数据按照一定的顺序进行排列显示(从小到大或从大到小)。

案例分析

1.一共需要的趟数我们用外层for循环, 5个数据我们一共需要走4趟.长度就是数组长度arr.length-1

2.每一趟交换次数我们用里层for循环

第一趟交换4次

第二趟交换3次

第三趟交换2次

第四趟交换1次

长度就是数组长度减去次数, 每一趟摆好该趟的最大元素位置, 所以存在-i(趟数)-1 (次数是从0开始), 最终arr.length - i - 1

3.交换2个数组元素

n个元素, 只需跑n-1趟就排好序了, n-1趟, 若是从小到大排序, 后面n-1个元素已经从小到大排好了, 剩下的第一个元素已经是最小的元素了, 也就无需再跑一趟比较了。

```
var arr = [3, 2, 4, 1, 5];
var temp;
//外层循环决定趟数,从小到大排序,每一趟摆好该趟的最大元素位置
for (var i = 0; i < arr.length - 1; i++) {
    //内存循环进行比较
    for (var j = 0; j < arr.length - i - 1; j++) {
```

```

        //如果索引小的数大于索引号大的数，则互换位置
        if (arr[j] > arr[j + 1]) {
            temp = arr[j + 1];
            arr[j + 1] = arr[j];
            arr[j] = temp;
        }
    }
}
console.log(arr);

```

11.函数

11.1函数的概念

函数:就是封装了一段**可被重复调用执行的代码块**。通过此代码块可以实现大量代码的重复使用。

11.2函数的使用

函数在使用时分为两步:声明函数和调用函数。

11.2.1声明函数

```

//声明函数
function函数名() {
    // 函数体代码
}

```

- function 是声明函数的关键字,必须小写**

- 函数名一般是动词，如getSum

10.2.2调用函数

```

//调用函数
函数名(); // 通过调用函数名来执行函数体代码

```

- 调用的时候千万不要忘记添加小括号

- 口诀:函数不调用,自己不执行。

注意:声明函数本身并不会执行代码,只有调用函数时才会执行函数体代码。

10.2.3函数的封装

- 函数的封装是把一个或者多个功能**通过函数的方式封装起来**,对外只提供一个简单的函数接口

- 简单理解:封装类似于将电脑配件整合组装到机箱中(类似快递打包)

11.3函数的参数

11.3.1形参和实参

在**声明函数**时,可以在函数名称后面的小括号中添加一些参数,这些参数被称为**形参**,而在**调用该函数**时,同样也需要传递相应的参数,这些参数被称为**实参**。

参数	说明
形参	形式上的参数，函数定义的时候传递的参数，当前并不知道是什么
实参	实际上的参数，函数调用的时候传递的参数，实参是传递给形参的

参数的作用:在**函数内部**某些值不能固定,我们可以通过参数在**调用函数时传递**不同的值进去。

注意：

- 1.函数的执行过程。
- 2.形参与实参。
- 3.形参可以看作不用声明的变量。
- 4.参数之间用逗号隔开。

```
//声明函数
function 函数名(形参1,形参2...){//在声明函数小括号内的是形参，形式上的参数

}
//调用函数
函数名(实参1,实参2...);//函数调用小括号内的是实参（实际的参数）
```

```
// 2.形参和实参的执行过程
function cook(aru){//形参接受实参，即aru='酸辣土豆丝'
    console.log(aru);
}
cook('酸辣土豆丝');

//3.函数的参数可以有，可以没有，个数不限
```

11.3.2函数形参和实参个数不匹配问题

参数个数	说明
实参个等于形参个数	输出正确结果
实参个数多于形参个数	只取到形参的个数,不管后面多的参数
实参个数小于形参个数	多的形参定义为undefined,结果为NaN

第三种情况解释：形参可以看作是不用声明的变量，少的形参即声明了变量但未赋值，那么就是undefined。undefined与定义的参数相加结果为NaN。

```
function sum (num1, num2) {  
    console.log (num1 + num2) ;  
}  
sum(100,200) ;//形参和实参个数相等,输出正确结果  
sum(100,400, 500,700); // 实参个数多于形参,只取到形参的个数  
sum(200) ;//实参个数少于形参,多的形参定义为undefined ,结果为NaN
```

注意:在JavaScript中,形参的默认值是undefined.

11.3.4小结

- 函数可以带参数也可以不带参数
- 声明函数的时候,函数名括号里面的是形参,形参的默认值为undefined
- 调用函数的时候,函数名括号里面的是实参
- 多个参数中间用逗号分隔
- 形参的个数可以和实参个数不四配,但是结果不可预计,我们尽量要四配

11.4函数的返回值

11.4.1 return语句

使用return语句实现函数将值返回给调用者。

即 函数名()=return 后面的结果

语法：放在函数内

```
return num;
```

注：在实际开发里面，经常用一个变量来接受函数的返回结果

11.4.2 return 终止函数

return语句之后的代码不会被执行

11.4.3 return的返回值

return只能返回一个值。如果用逗号隔开多个值,以最后一个为准。

想要返回多个值，可借助数组、对象等实现。

如，借用数组：

```
//3.求任意两个数的加减乘数结果  
function getResult(num1, num2) {  
    return [ num1 + num2, num1 - num2, num1 * num2, num1 / num2] ;  
}
```

11.4.4函数没有return则返回undefined

函数都是有返回值的

1.如果有return则返回return后面的值

2.如果没有return 则返回undefined

11.4.5 break ,continue ,return的区别

- break :结束当前的循环体(如for、 while)
 - continue :跳出本次循环,继续执行下次循环(如for、 while)
 - return :不仅可以退出循环,还能够返回return语句中的值,同时还可以结束当前的函数体内的代码
- 前两个是针对循环来说的,

11.5通过榨汁机看透函数



榨汁机,放入不同的原料得到不同的果汁。

函数,放入不同的参数,得到不同的结果

11.6案例

11.6.1案例1-简单计算器

写一个函数,用户输入任意两个数字的任意算术运算(简单的计算器小功能),并能弹出运算后的结果。

```
var num1 = parseFloat(prompt('请输入第一个数: '));
var num2 = parseFloat(prompt('请输入第二个数: '));
var op = prompt('请输入操作符: \n1.加法运算+\n2.减法运算-\n3.乘法运算x\n4.除法运算÷');

// 让用户选择操作符比让用户输入操作符更好: 1.用户输入更简单2.操作符乘可能输入为x(小写x)或x或*,具有多样性,使用选择一定程度上进行了规范
function getResult(n, m, operation) {
    var result;
    switch (operation) {
        case '1':
            result = n + m;
            break;
        case '2':
            result = n - m;
            break;
        case '3':
```

```

        result = n * m;
        break;
    case '4':
        result = n / m;
        break;
    default:
        result = '输入错误';
    }
    return result;
}
var res = getResult(num1, num2, op);
alert(res);

```

11.6.2案例2-素数判断

写一个函数,用户输入一个数判断是否是素数,并返弹出回值(又叫质数,只能被1和自身整数的数)

//写一个函数，能判断传入的一个数值是否是质数，如果是质数返回true，如果不是质数返回false，（质数：从2开始只能被1和自身整数的数）

```

function isPrimeNum(num) {
    for (var i = 2; i < num; i++) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}
var n = prompt('请输入一个大于2的数: ');
var res = isPrimeNum(n);
if (res == true) {
    alert('该数是素数');
} else {
    alert('该数不是素数');
}

```

11.7 arguments的使用

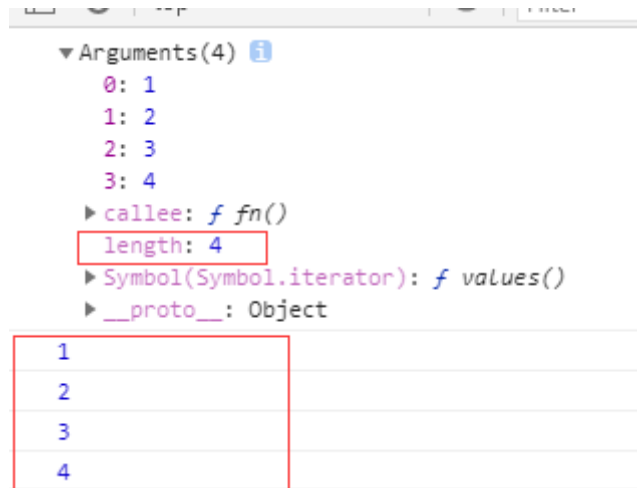
当我们不确定有多少个参数传递的时候,可以用arguments来获取。在JavaScript中, arguments实际上是它是当前函数的一个内置对象。所有函数都内置了一个 arguments对象, arguments对象中存储了传递的所有实参。

arguments展示形式是一个伪数组,因此可以进行遍历。伪数组具有以下特点:

- 具有length属性
- 按索引方式储存数据，可以按照数组的方式遍历
- 不具有数组的push , pop等方法

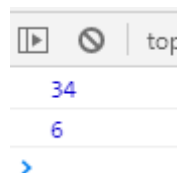
有了arguments就不用写形参了


```
//只有函数才有arguments对象，且每个函数都内置了这个arguments
function fn() {
    //arguments存储了所有传递过来的实参，并是个伪数组
    console.log(arguments);
    //可以按照数组的方式进行遍历
    for (var i = 0; i < arguments.length; i++) {
        console.log(arguments[i]);
    }
}
fn(1, 2, 3, 4);
```



案例：利用函数求任意个数的最大值

```
//任意个数，利用arguments
function getMax() {
    var max = arguments[0];
    for (var i = 0; i < arguments.length; i++) {
        if (arguments[i] > max) {
            max = arguments[i]
        }
    }
    return max;
}
console.log(getMax(2, 4, 5, 23, 34, 33));
console.log(getMax(2, 6, 4));
```



11.8函数可以调用另外一个函数

因为每个函数都是独立的代码块,用于完成特殊任务,因此经常会用到函数相互调用的情况。

```
function fn1() {
    console.log(111);
    fn2();
    console.log('fn1');
}
function fn2() {
    console.log(222);
    console.log('fn2');
}
fn1();
```

上图的结果：

111

222

fn2

fn1

案例：利用函数输出年份的2月份天数

```
//闰年，2月份28天；平年，2月份29天
function backDay() {
    var year = prompt('请输入年份：')
    if (isRunYear(year)) { //调用函数需要加小括号和参数
        alert('该年份2月份有28天');
    } else {
        alert('该年份2月份有29天');
    }
}
//判断是否是闰年
function isRunYear(year) {
    var flag = false;
    if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {
        flag = true;
    }
    return flag;
}
backDay();
```

11.9函数的两种声明方式

1.利用函数关键字自定义函数（命名函数）

```
function fn(){
}
fn();
```

2.函数表达式（匿名函数）

```
var fun = function (aru) {  
    console.log('函数表达式');  
    console.log(aru);  
}  
fun('11');
```

注意:

- 1) 两者的区别1: fun 是变量名, 不是函数名
- 2) 函数表达式声明方式与声明变量类似, 变量里存的是值, 函数表达式里存的是函数
- 3) 函数表达式也可以进行参数传递
- 4) 两者的区别2: 函数表达式没有函数名字, 因此又称为匿名函数

12.作用域

12.1作用域概述

通常来说, 一段程序代码中所用到的名字并不总是有效和可用的, 而限定这个**名字的可用性的代码范围**就是这个名字的**作用域**。作用域的使用提高了程序逻辑的局部性, 增强了程序的可靠性, **减少了名字冲突**。

//js作用域: 代码名字(变量)在某个范围内起作用 and 效果, 目的是提高程序的可靠性, 更重要的是**减少命名冲突**

//js的作用域分类(es6之前): 全局作用域、局部作用域

//全局作用域: 整个script标签或一个单独的js文件

```
var num = 10;  
console.log(num);
```

//局部作用域(函数作用域) 函数内部就是局部作用域

```
function fn() {  
    var num = 20;  
    console.log(num);  
}  
fn();
```

//num没有命名冲突, 因此会输出两个值



12.2变量的作用域

12.2.1变量作用域的分类

在JavaScript中, 根据作用域的不同, 变量可以分为两种:

- 全局变量
- 局部变量

12.2.2全局变量

在全局作用域下声明的变量叫做**全局变量**(在函数外部定义的变量).

- 全局变量在代码的任何位置都可以使用
- 在全局作用域下var声明的变量是全局变量
- 特殊情况下,在函数内不使用var声明的变量但赋值也是全局变量(不建议使用).

12.2.3局部变量

在局部作用域下声明的变量叫做**局部变量**(在函数内部定义的变量)

- 局部变量只能在该函数**内部**使用
- 在函数内部var声明的变量是局部变量
- 函数的**形参**实际上就是局部变量

12.2.4全局变量和局部变量的区别

- 全局变量:在任何一个地方都可以使用,只有在浏览器关闭时才会被销毁,因此比较占内存
- 局部变量:只在函数内部使用,当其所在的代码块被执行时,会被初始化;当代码块运行结束后,就会被销毁,因此更节省内存空间

12.3 js (es6之前) 没有块级作用域

块级作用域{}, 花括号内的

12.4作用域链

- 只要是代码,就至少有一个作用域
- 写在函数内部的局部作用域
- 如果函数中还有函数,那么在这个作用域中又可以诞生一个作用域
- 内部函数访问外部函数的变量,采取的是**链式查找**的方式来决定取那个值这种结构我们称为作用域链——**就近原则**

```
var num = 10;

function fn() { // 外部函数
  var num = 20;

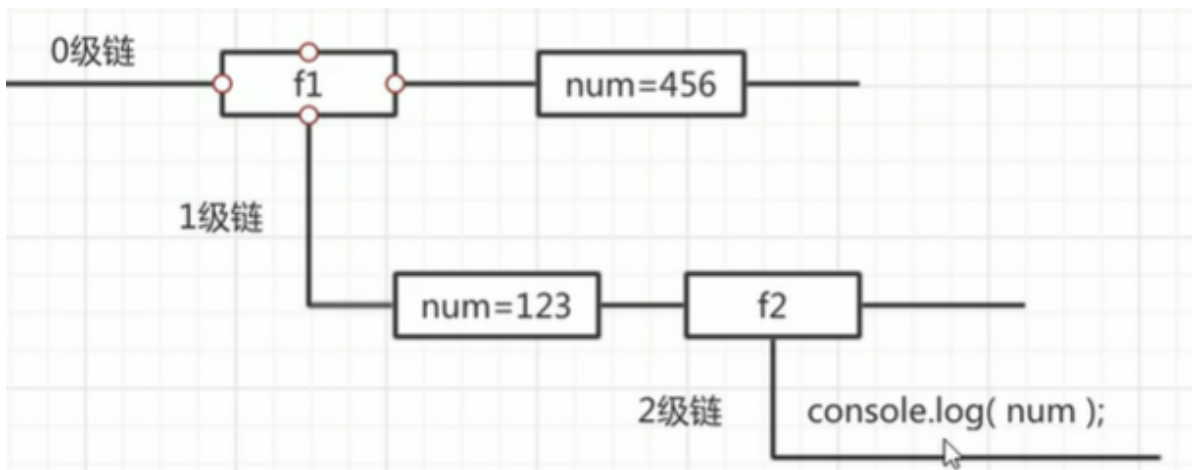
  function fun() { // 内部函数
    console.log(num);
  }
}
```

所以上图num=20

练习1:

```
function f1() {
  var num = 123;
  function f2() {
    console.log( num );
  }
  f2();
}
var num = 456;
f1();
```

结果：num=123 做法：站在目标出发，一层一层的往外查找，找到最近都即可



练习2:

```
var a = 1;
function fn1() {
  var a = 2;
  var b = '22';
  fn2();
  function fn2() {
    var a = 3;
    fn3();
    function fn3() {
      var a = 4;
      console.log(a); //a的值 ?
      console.log(b); //b的值 ?
    }
  }
}
fn1();
```

结果：a=4,b='22'

13.预解析

1.我们js引擎运行js分为两步: **预解析、代码执行**

(1). 预解析js引擎会把js里面所有的**var和function**提升到**当前作用域**的最前面

(2).代码执行按照代码书写的顺序从 上往下执行

2.预解析分为**变量预解析(变量提升)**和**函数预解析(函数提升)**

(1)变量提升就是把所有的**变量声明**提升到**当前作用域**最前面, **不提升赋值操作**

(2)函数提升就是把所有的**函数声明**提升到**当前作用域**的最前面, **不调用函数**

案例:

```
//1
console.log(num); // undefined
var num = 10;
//相当于执行了以下代码
var num;
console.log(num);
num = 10 ;
```

```
// 2
fun(); // not a function
var fun = function() {
    console.log(22);
}
//相当于执行了以下代码
var fun;
fun();
fun = function() {
    console.log(22);
}
```

因此, **函数表达式调用必须写在函数表达式的下面**

练习: 请判断结果

```
// 案例1
var num = 10;

fun();

function fun() {
    console.log(num);
    var num = 20;
}
```

相当于执行以下操作

```
var num;
function fun {
    var num;
    console.log(num);
    num = 20;
}
num = 10;
fun();
```

所以结果：undefined(根据作用域链)

```
// 案例2
var num = 10;
function fn(){
    console.log(num);
    var num = 20;
    console.log(num);
}
fn();
```

相当于执行以下操作

```
var num;
function fn(){
    var num;
    console.log(num);
    num = 20;
    console.log(num);
}
num = 10;
fn();
```

所以结果：undefined 20

```
// 案例3
var a = 18;
f1();
function f1() {
    var b = 9;
    console.log(a);
    console.log(b);
    var a = '123';
}
```

相当于执行以下操作

```

var a;
function f1(){
    var b;
    var a;
    b = 9;
    console.log(a);
    console.log(b);
    a = '123';
}
a = 10;
f1();

```

所以结果： undefined 9

```

// 案例4
f1();
console.log(c);
console.log(b);
console.log(a);
function f1() {
    var a = b = c = 9;
    console.log(a);
    console.log(b);
    console.log(c);
}

```

相当于执行以下操作

```

function f1(){
    var a;//注意
    a = b = c = 9;//注意
    console.log(a);
    console.log(b);
    console.log(c);
}
f1();
console.log(c);
console.log(b);
console.log(a);

```

输出结果： 9 9 9 9 9报错

注意：

```

var a = b = c = 9;
//相当于var a = 9;b = 9;c = 9;b和c直接赋值，没有var声明当全局变量看
//集体声明var a = 9,b = 9,c = 9;之间用逗号隔开
//预解析后应是
var a;
a = b = c = 9;

```


14.对象

14.1什么是对象？

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”。



红框内的为对象，其他不是

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)

14.2为什么需要对象

保存一个值时,可以使用变量;保存多个值(一组值)时,可以使用数组。如果要保存一个人的完整信息呢?

JS中的对象表达结构更清晰,更强大。张三疯的个人信息在对象中的表达结构如下:

```
person.name = '张三疯';
person.sex = '男';
person.age = 128;
person.height = 154;
```

14.3创建对象的三种方式

在JavaScript中,现阶段我们可以采用三种方式创建对象(object):

- 利用**字面量**创建对象
- 利用**new Object**创建对象
- 利用**构造函数**创建对象

14.3.1利用字面量创建对象

对象字面量:就是花括号{}里面包含了表达这个具体事物(对象)的属性和方法。

{ }里面采取**键值对**的形式表示

- 键:相当于属性名
- 值:相当于属性值,可以是任意类型的值(数字类型、字符串类型、布尔类型、函数类型等)

注意:

- 1.里面的属性或方法采用键值对的形式表达
- 2.多个属性或者方法之间**用逗号隔开**

3.方法冒号后面跟的是一个匿名函数

```
//创建了一个空的对象
// var obj={};
var obj = {
  name: '张三丰',
  sex: '男',
  age: 18,
  sayHi: function () {
    console.log('Hi');
  }
}
```

对象的调用

- 对象里面的属性调用:**对象.属性名**, 这个小点.就理解为"的"
- 对象里面属性的另一种调用方式:**对象['属性名']**,注意方括号里面的属性必须**引号**,我们后面会用两种都很重要, 都要会。
- 调用对象里的方法: **对象名.方法名()**, 注意**小括号**

```
console.log(obj.name);
console.log(obj['sex']);
obj.sayHi();
```

变量、属性、函数、方法总结

- 变量:单独声明赋值,**单独存在**
- 属性:对象里面的变量称为属性,**不需要声明**,用来描述该对象的特征
- 函数:**单独存在的**,通过“函数名()”的方式就可以调用
- 方法:对象里面的函数称为方法,方法**不需要声明**, 使用“对象.方法名()”的方式就可以调用,方法用来描述该对象的行为和功能。

14.3.2利用new object创建对象

语法:

```
var obj = new Object();
obj.name = '张三丰';
obj.age = 18;
obj.sayHi = function () {
  console.log('hi');
}
console.log(obj.name);
```

注意:

- 1.我们是利用 **等号=赋值** 的方法添加对象的属性和方法
- 2.每个属性和方法之间用**分号结束**
- 3.方法和属性调用与之前一致

14.3.3利用函数创建对象

为什么需要使用构造函数

因为前面两种创建对象的方式一次只能创建一个对象，而当多个对象里面的属性和方法是大量相同的时，我们只能复制。

因此利用函数的方法，重复这些相同的代码，我们就把这个函数称为**构造函数**。这个函数里面封装的不是普通代码，**而是对象**。

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运算符一起使用。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。

语法格式：

```
function 构造函数名(){
    this.属性=值;
    this.方法=function(){}
}
// 调用
new 构造函数名();
```

```
function Star(name, age, sex) {
    this.name = name;
    this.age = age;
    this.sex = sex;
    this.sing = function (sang) {
        console.log(sang);
    }
}
var ldh = new Star('刘德华', 18, '男');//调用函数返回的是一个对象
console.log(ldh.name);
console.log(ldh['age']);
ldh.sing('冰雨');// 方法的调用不用加log输出，因为方法内已有log输出
var zxy = new Star('张学友', 20, '男');
console.log(zxy.sex);
```



```
刘德华
18
冰雨
男
```

注意：

- 1.构造函数名字**首字母要大写**
- 2.我们构造函数不需要return 就可以返回结果
- 3.我们调用构造函数必须使用**new**
- 4.我们只要new Star() 调用函数就能创建一个对象ldh
- 5.我们的属性和方法前面必须添加**this**
- 6.方法的调用

14.3.4构造函数和对象

- 构造函数,如Stars(),抽象了对象的公共部分,封装到了函数里面,它泛指某一大类(class)
- 创建对象, 如new Stars(),特指某一个,通过new关键字创建对象的过程我们也称为**对象实例化**

14.4 new关键字

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).

14.5遍历对象属性

for..in语句用于对数组或者对象的属性进行循环操作。

语法格式:

```
for (var k in obj){  
  console.log(k); // k,变量,输出得到属性名  
  console.log(obj[k]); // obj[k] 得到是属性值  
}
```

常用变量名k、key作为循环, 注意使用是obj[k] 对象名+中括号+变量名

案例

```
var obj = {  
  name: '张三丰',  
  sex: '男',  
  age: 18,  
  sayHi: function () {  
    console.log('Hi');  
  }  
}  
for (var k in obj) {  
  console.log(k); //k变量 输出属性名  
  console.log(obj[k]); //obj[k] 输出属性值  
}
```

name
张三丰
sex
男
age
18
sayHi
f () { console.log(}

14.6小结

- 1.对象可以让代码结构更清晰
- 2.对象复杂数据类型object,
- 3.本质:对象就是一组无序的相关属性和方法的集合。
- 4.构造函数泛指某一大类,比如苹果,不管是红色苹果还是绿色苹果,都统称为苹果。
- 5.对象实例特指一个事物,比如这个苹果、正在给你们讲课的pink老师等。
- 6.for..in语句用于对对象的属性进行循环操作。

14.7作业

- 1.创建一个电脑对象,该对象要有颜色、重量、品牌、型号,可以看电影、听音乐打游戏和敲代码。
- 2.创建一个按钮对象,该对象中需要包含宽,高,背景颜色和点击行为。
- 3.创建一个车的对象,该对象要有重量颜色、牌子,可以载人、拉货和耕田。
- 4.预解析: 判断结果

```
var num = 1;
function demo() {
    console.log(num);
    function demoSon() {
        console.log(num);
        num = 3;
        console.log(num);
    }
    var num = 2;
    demoSon();
}
demo();
```

- 5.有以下两个数组,一个数组arr是班级里所有的学员的名称,一个数组currentArr是提交了每日反馈的学员名单,请创建一种算法,把未提交每日反馈的学员筛选出来

```
var arr = ["张瑞淑", "徐海涛", "谢岗岗", "薛鹏", "魏明杨", "党婷", "熊飞", "郑翠翠", "李航卫", "张大大", "屈涛", "汪孝双", "代攀飞", "武志钰"];
```

```
var currentArr = ["张瑞淑", "徐海涛", "谢岗岗", "魏明杨", "党婷", "熊飞", "郑翠翠"];
```

// 1. 创建一个电脑对象,该对象要有颜色、重量品牌、型号,可以看电影、听音乐打游戏和敲代码。

```
var computer = {
    color: 'black',
    weight: '800g',
    type: 'E470',
    watchMovie: function () {
        console.log('我在看电影');
    },
    listenMusic: function () {
        console.log('我在听音乐');
    },
    playGame: function () {
        console.log('我在打游戏');
    },
};
```

```

        work: function () {
            console.log('我在敲代码');
        }
    }
    for (var k in computer) {
        console.log(computer[k]);
    }

```

black
800g
E470
f () { console.log('我在看电影'); }
f () { console.log('我在听音乐'); }
f () { console.log('我在打游戏'); }
f () { console.log('我在敲代码'); }

// 2. 创建一个按钮对象,该对象中需要包含宽,高,背景颜色和点击行为。

```

var btn = new Object();
btn.width = '400px';
btn.height = '300px';
btn.backgroundColor = 'red';
btn.click = function () {
    console.log('点击');
}
for (var k in btn) {
    console.log(btn[k]);
}

```

400px
300px
red
f () { console.log('点击'); }

// 3. 创建一个车的对象,该对象要有重量颜色、牌子,可以载人、拉货和耕田。

```

var car = {
    weight: '700kg',
    color: 'white',
    type: '宝马',
    zairen: function () {
        console.log('可以载人');
    },
    lahuo: function () {
        console.log('可以拉货');
    },
    gengtian: function () {
        console.log('可以耕田');
    }
}

```

```

}
for (var k in car) {
    console.log(car[k]);
}

```

```

700kg
white
宝马
f () {
    console.log('可以载人');
}
f () {
    console.log('可以拉货');
}
f () {
    console.log('可以耕田');
}

```

//相当于执行以下代码:

```

var num;
function demo() {
    var num;
    function demoSon() {
        console.log(num);
        num = 3;
        console.log(num);
    }
    console.log(num);
    num = 2;
    demoSon();
}
num = 1;
demo();

```

结果: undefined 2 3

// 5. 有以下有两个数组, 一个数组arr是班级里所有的学员的名称, 一个数组currentArr是提交了每日反馈的学员名单, 请创建一种算法, 把未提交每日反馈的学员筛选出来

```

var arr = ["张瑞淑", "徐海涛", "谢岗岗", "薛鹏", "魏明杨", "党婷", "熊飞", "郑翠翠", "李航卫", "张大大", "屈涛", "汪孝双", "代攀飞", "武志钰"];
var currentArr = ["张瑞淑", "徐海涛", "谢岗岗", "魏明杨", "党婷", "熊飞", "郑翠翠"];

var newArr = [];
for (var i = 0; i < arr.length; i++) {
    for (var j = 0; j < currentArr.length; j++) {
        if (arr[i] == currentArr[j]) {
            break; //匹配到了就跳出该层循环, 开始外层循环的下一个
        } else if (j == currentArr.length - 1) {
            newArr[newArr.length] = arr[i]; //如果j一层都没匹配到, 就放入新数组
        }
    }
}
console.log(newArr);

```

作业小结:

1.对象内有多个方法时，用逗号隔开

```
type: 'E470',
watchMovie: function () {
    console.log('我在看电影');
},
listenMusic: function () {
    console.log('我在听音乐');
},
playGame: function () {
    console.log('我在打游戏');
},
```

2.属性的值为字符串型时，别忘记加"引号。

15.内置对象

15.1内置对象概述

- JavaScript 中的对象分为3种:自定义对象、内置对象、浏览器对象
- 前面两种对象是JS基础内容,属于ECMAScript;第三个浏览器对象属于我们S独有的，我们JS API讲解
- 内置对象**就是指JS语言自带的一些对象,这些对象供开发者使用,并提供了一些常用的或是最基本而必要的功能(属性和方法)。

简单理解:手机里的电话、短信功能，我们并不需要知道电话、短信功能具体是怎么实现的，会用就可以了。类似于库。

- 内置对象最大的优点就是帮助我们快速开发
- JavaScript 提供了多个内置对象: Math.Date、Array、String等

15.2查文档

15.2.1 MDN

学习一个内置对象的使用,只要学会其常用成员的使用即可,我们可以通过查文档学习,可以通过MDN/W3C来查询。

Mozilla开发者网络(MDN)提供了有关开放网络技术(OpenWeb)的信息,包括HTMLCSS 和万维网及HTML5应用的API。

MDN: <https://developer.mozilla.org/zh-CN/>

15.2.2如何学习对象中的方法

- 1.查阅该方法的功能
- 2.查看里面参数的意义和类型
- 3.查看返回值的意义和类型
- 4.通过demo进行测试，或者看示例，更建议自己动手敲代码试试

15.3封装自己的数学对象

Math数学对象，不是一个构造函数，不需要new来调用，直接使用里面的数学和方法即可。

Math.max使用：

语法：

```
Math.max(value1[,value2, ...])
```

参数：

value1, value2, ...一组数值

返回值：

返回给定的一组数字中的最大值。如果给定的参数中至少有一个参数无法被转换成数字，则会返回NaN。

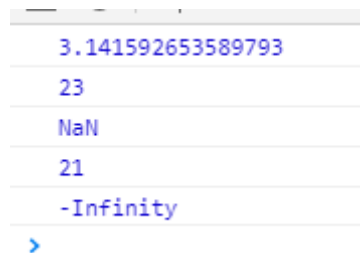
描述：

如果没有参数，则结果为 -[Infinity]。

如果有任一参数不能被转换为数值，会隐式转换，则结果为 [NaN]。

案例：

```
console.log(Math.PI);  
console.log(Math.max(1, 3, 23, 8));  
console.log(Math.max(1, 'pink'));  
console.log(Math.max(21, '1', '4')); // 会隐式转换  
console.log(Math.max());
```



```
3.141592653589793  
23  
NaN  
21  
-Infinity  
>
```

注意：使用时Math的M要大写

案例:封装自己的数学对象

利用对象封装自己的数学对象里面有PI最大值和最小值

```
var myMath = {  
  PI: 3.141592,  
  max: function () {  
    var max = arguments[0];  
    for (var i = 0; i < arguments.length; i++) {  
      if (max < arguments[i]) {  
        max = arguments[i]  
      }  
    }  
    return max;  
  },  
  min: function () {  
    var min = arguments[0];  
    for (var i = 0; i < arguments.length; i++) {  
      if (min > arguments[i]) {  
        min = arguments[i]  
      }  
    }  
    return min;  
  }  
};
```

```

    }
    }
    return min;
}
}
console.log(myMath.PI);
console.log(myMath.max(1, 3, 2));
console.log(myMath.min(1, 3, 2));

```

```

3.141592
3
1

```

15.4数学对象Math

15.4.1 Math概述

Math对象不是构造函数,它具有数学常数和函数的属性和方法。跟数学相关的运算(求绝对值,取整、最大值等),可以使用Math中的成员。

```

Math.PI//圆周率
Math.floor()//向下取整
Math.ceil()//向上取整
Math.round()//四舍五入就近取整注意-3.5结果是-3
Math.abs()//绝对值
Math.max()/Math.min() // 求最大和最小值

```

示例:

```

//.abs绝对值
console.log(Math.abs(-1));//1
console.log(Math.abs('-2'));//2,隐式转换
//取整
//.floor向下取整,抹掉小数
console.log(Math.floor(8.3));//8
console.log(Math.floor(6.9));//6
//.ceil向上取整
console.log(Math.ceil(6.2));//7
//.round四舍五入,注意.5时,往数值大的方向取
console.log(Math.round(4.5));//5
console.log(Math.round(2.4));//2
console.log(Math.round(-1.5));// -1

```

```

1
2
8
6
7
5
2
-1

```

注意: .round四舍五入, 注意.5时, 往数值大的方向取, 即特别小心负数的四舍五入

Math对于字符串型的数字有一定的隐式转换

15.4.2 Math.random()

Math.random() 函数返回一个浮点数, 伪随机数在 0 (包括0) 和 1 (不包括) 之间。**[0,1)**.注意小括号不要忘

扩展:

得到一个两数之间的随机整数, 包括两个数在内(应用很多)

```
//得到一个两数之间的随机整数, 包括两个数在内
function getRandomIntInclusive(min, max) {
    return Math.floor(Math.random() * (max - min + 1) + min);
    // 先是[0,max+1),再是[min,max+1),最后向下取整[min,max]里的整数
}
console.log(getRandomIntInclusive(1, 10));
```

随机点名:

```
//随机点名
var arr = ['张三', '张三丰', '王武', '李思', '徐三'];
console.log(arr[getRandomIntInclusive(0, arr.length - 1)]);
```

猜数字:

- ①随机生成一个1~10 的整数我们需要用到Math.random()方法。
- ②需要一直猜到正确为止,所以一直循环。
- ③使用while循环更简单。
- ④核心算法:使用if else if多分支语句来判断大于、小于、等于。

增加要求: 只有10次猜的机会

```
function getRandomIntInclusive(min, max) {
    return Math.floor(Math.random() * (max - min + 1) + min);
    // 先是[0,max+1),再是[min,max+1),最后向下取整[min,max]里的整数
}
var ran = getRandomIntInclusive(1, 10);
var n = 0;
while (n < 10) {
    var num = parseInt(prompt('请输入1-10之间的整数: '));
    if (num > ran) {
        n++;
        alert('猜大了,还剩' + (10 - n) + '次机会');
    } else if (num < ran) {
        n++;
        alert('猜小了,还剩' + (10 - n) + '次机会');
    } else {
        alert('猜对了');
        break; //猜对退出整个循环
    }
}
```

15.5 Date 日期对象

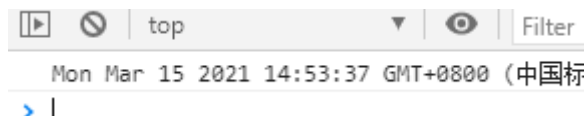
15.5.1 Date概述

- Date 对象和Math对象不一样,他是一个**构造函数**,所以我们需要**实例化**后才能使用
- Date 实例用来处理日期和时间

15.5.2 Date()方法的使用

1.获取当前时间必须实例化

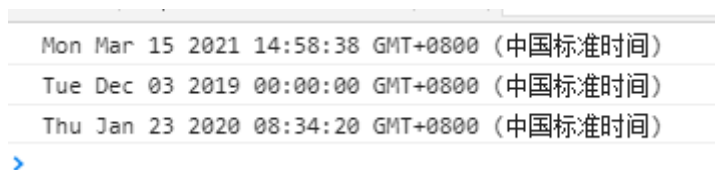
```
var now = new Date();//Date()是一个构造函数，必须实例化后才能使用
console.log(now);//返回当前系统时间
```



2.Date()构造函数的参数

如果括号里面有时间,就返回参数里面的时间、例如日期格式字符串为"2019-5-1",可以写成new Date('2019-5-1')或者new Date('2019/5/1')

```
//参数常用写法。数字型：2019,11,3 或字符串型 '2020-1-23 8:34:20'
var date1 = new Date(2019, 11, 3);//返回12月，坑！
console.log(date1);
var date2 = new Date('2020-1-23 8:34:20');
console.log(date2);
```



注意有参数情况下**日期的格式书写**，字符型更常用写，日期用-分隔，时间用：分隔

15.5.3日期格式化

方法名	说明	代码
getFullYear()	获取当年	dObj.getullYear()
getMonth()	获取当月(0-11)	dObj.getMonth()
getDate()	获取当天日期	dObj.getDate()
getDay()	获取星期几(周日0 到周六6)	dObj.getDay()
getHours()	获取当前小时	dObj.getHours()
getMinutes()	获取当前分钟	dObj.getMinutes()
getSeconds()	获取当前秒钟	dObj.getSeconds()

案例：

格式化日期年月日

```

var now = new Date();
console.log(now.getFullYear());
console.log(now.getMonth() + 1); //注意月份是0-11月，所以要+1
console.log(now.getDate()); //日期
console.log(now.getDay()); //星期，注意周日显示数字0，周一到周六1-6
//输入2021年3月2日 星期几格式
year = now.getFullYear();
month = now.getMonth() + 1; //注意月份+1
date = now.getDate();
//由于星期输出的是数字，可以利用数组索引号方式输出文字，注意将周日放在最前面，因为周日
数字为0
var week = ['星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六'];
day = now.getDay()
console.log('今天是: ' + year + '年' + month + '月' + date + '日' +
week[day]);

```

```

2021
3
15
1
今天是: 2021年3月15日 星期一

```

注意:

- 1.月份是0-11月，一般情况要+1
- 2.星期，注意周日显示数字0，周一到周六1-6
- 3.可以利用数组索引号方式输出星期几
- 4.方法名后记得加小括号

格式化日期时分秒:

```

var now = new Date();
console.log(now.getHours());
console.log(now.getMinutes());
console.log(now.getSeconds());
//要求封装一个函数返回当前的时分秒 11:11:02
function getTime() {
    var time = new Date();
    var h = time.getHours(); //时
    h = h < 10 ? '0' + h : h; //使用三元表达式补零
    var m = time.getMinutes(); //分
    m = m < 10 ? '0' + m : m;
    var s = time.getSeconds(); //秒
    s = s < 10 ? '0' + s : s;
    return h + ':' + m + ':' + s;
}
console.log(getTime());

```

```

15
48
2
15:48:02

```

注意：

1.在函数内也需要定义一个日期对象

2.使用三元表达式补零

15.5.4获取日期的总的毫秒形式

Date对象是基于1970年1月1日(世界标准时间)起的毫秒数

我们经常利用总的毫秒数来计算时间，因为它更精确

```
//获得Date总的毫秒数,不是当前时间的毫秒数而是距离1970年1月1号过了多少毫秒数
//1.通过valueOf() getTime()
var date = new Date();
console.log(date.valueOf());
console.log(date.getTime());
//2.简单的写法（最常用的写法）
var date1 = +new Date(); //+new Date()返回的就是总毫秒数
console.log(date1);
// 3.H5新增的方法，若不考虑兼容性问题，可以使用该方法
console.log(Date.now());
```

```
1615795019463
1615795019463
1615795019463
1615795019463
```

小结：

1.对象名.valueOf()

2.对象名.getTime()

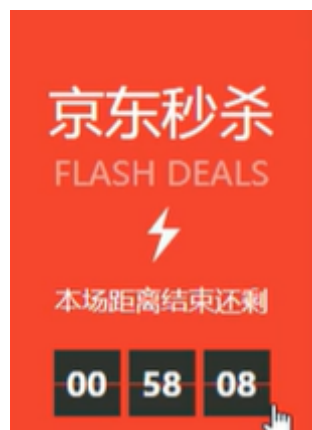
3.var date=+new Date(),

无参数获得当前时间总毫秒数；有参数获得参数时间的总毫秒数，最常用

4.Date.now() H5新增，D大写

6.总毫秒数是不可能重复的，每一个值都是唯一的，因此又称为时间戳

案例：实现倒计时（很经典，要记住）



案例分析：

①核心算法:输入的时间减去现在的时间就是剩余的时间,即倒计时,但是不能拿着时分秒相减,比如05分减去25分,结果会是负数的。

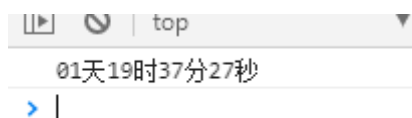
②用时间戳来做。用户输入时间总的毫秒数减去现在时间的总的毫秒数,得到的就是剩余时间的毫秒数。

③把剩余时间总的毫秒数转换为天、时、分、秒(时间戳转换为时分秒)

转换公式如下:

- `d= parseInt(总秒数/ 60/60/24); //计算天数`
- `h= parseInt(总秒数/ 60/60%24) ;//计算小时`
- `m=parseInt(总秒数/60 %60); // 计算分数`
- `s=parseInt(总秒数%60);//计算当前秒数`

```
function countdown(time) {  
    var nowTime = +new Date(); //返回当前时间的总毫秒数  
    var inputTime = +new Date(time); //返回用户输入时间的总毫秒数  
    var times = (inputTime - nowTime) / 1000; //time是剩余时间总秒数, 1秒  
    =1000毫秒  
  
    var d = parseInt(times / 60 / 60 / 24); //parseInt是取整用 /60 分 /60 时  
    /24 天  
  
    d = d < 10 ? '0' + d : d; //同样使用三元表达式补零  
    var h = parseInt(times / 60 / 60 % 24);  
    h = h < 10 ? '0' + h : h; //使用三元表达式补零  
    var m = parseInt(times / 60 % 60);  
    m = m < 10 ? '0' + m : m;  
    var s = parseInt(times % 60);  
    s = s < 10 ? '0' + s : s;  
    return d + '天' + h + '时' + m + '分' + s + '秒';  
}  
console.log(countdown('2021-3-17 12:00:00'));
```



注意:

- 1.同样需要使用三元表达式补零
- 2.使用函数时注意输入日期时间的格式, 字符型
- 3.要记住时间转换公式, 注意转换后都要取整

15.6数组对象

15.6.1检测是否为数组两种方法

1.instanceof, 运算符

语法:

```
变量 instanceof Array //是数组返回true, 否则返回false
```

```
//1. instanceof 运算符法
var arr = [2, 1, 4];
var obj = {};
console.log(arr instanceof Array);
console.log(obj instanceof Array);
```

2.Array.isArray(参数)，是则返回true，否则返回false

```
// 2.Array.isArray(参数); H5新增的方法，IE9以上版本支持
var arr = [2, 1, 4];
var obj = {};
console.log(Array.isArray(arr));
console.log(Array.isArray(obj));
```

可以借此完善之前的翻转数组函数，在翻转前判断传入的参数是否为数组

```
// 完善翻转数组函数
function reverse(arr) {
  if (arr instanceof Array) {
    for (var i = 0; i < arr.length / 2; i++) {
      //第一个元素与最后一个元素交换位置，注意最后一个元素的索引号
      var temp;
      temp = arr[i];
      arr[i] = arr[arr.length - i - 1];
      arr[arr.length - i - 1] = temp;
    }
    return arr;
  } else {
    return 'error 该函数参数必须为数组[1,3]';
  }
}
```

15.6.2添加或删除数组元素

方法名	说明	返回值
push(参数...)	末尾添加一个或多个元素，注意会修改原数组	并返回新的长度
pop()	删除数组最后一个元素，把数组长度减1，无参数、会修改原数组	返回它删除的元素的值
unshift(参数..)	向数组的开头添加一个或更多元素，注意会修改原数组	并返回新的长度
shift()	删除数组的第一个元素，数组长度减1，无参数、修改原数组	返回它删除的元素的值

添加数组元素的两个方法：


```
//添加数组元素
//1..push() 在数组末尾添加一个或多个元素
var arr = [2, 4, 2];
// arr.push('pink',6);
console.log(arr.push('pink', 6));//5
console.log(arr);

//2..unshift() 在数组开头添加一个或多个元素
console.log(arr.unshift('blue'));//6
console.log(arr);
```

```
5
▶ (5) [2, 4, 2, "pink", 6]
6
▶ (6) ["blue", 2, 4, 2, "pink", 6]
```

push()、unshift()两个方法的共同点:

- 1) push()、unshift() 参数即添加的数组元素
- 2) push()、unshift()完毕后, **返回的结果是新数组长度**
- 3) 原数组也都会发生变化

push()、unshift()两个方法的不同点:

- 1.push()是数组末尾添加元素
- 2.unshift()是数组开头添加元素

删除数组元素的两个方法:

```
//删除数组元素
//3.pop() 删除数组最后一个元素
console.log(arr.pop());
console.log(arr);
//1) pop() 无参数
//2) pop()完毕后, 返回的结果是 删除的元素
//3) 原数组也会发生变化
//4) 一次只能删除一个元素

//4.shift() 删除数组第一个元素
console.log(arr.shift());
console.log(arr);
//1) shift() 无参数
//2) shift()完毕后, 返回的结果是 删除的元素
//3) 原数组也会发生变化
//4) 一次只能删除一个元素
```

```
gray
▶ (5) ["blue", 2, 4, 2, "pink"]
blue
▶ (4) [2, 4, 2, "pink"]
```

pop()、shift()两个方法的共同点:

- 1) pop()、shift() 都无参数

2) pop()、shift()完毕后，返回的结果是删除的元素

3) 原数组也会发生变化

4) 一次只能删除一个元素

pop()、shift()两个方法的不同点：

1.pop()是删除数组最后一个元素

2.shift()是删除数组第一个元素

15.6.3数组排序

方法名	说明	是否修改原数组
reverse()	颠倒数组中元素的顺序,无参数	该方法会改变原来的数组，返回新数组
sort([compareFunction])	对数组的元素进行排序	该方法会改变原来的数组，返回新数组

注意：

sort()排序，默认排序顺序是在将元素转换为字符串，根据它们的UTF-16代码单元值进行排序。因此，我们需要改进：

```
arr1.sort(function (a, b) {  
    return a - b; //a-b>0时交换位置，即升序排列  
    //return b - a; //b-a>0时交换位置，即降序排列  
});
```

示例：

```
// 数组翻转  
var arr = [2, 4, 5, 2, 1];  
arr.reverse();  
console.log(arr);  
  
//数组排序（冒泡排序）  
var arr1 = [3, 42, 9, 13];  
arr1.sort(function (a, b) {  
    return a - b; //a-b>0时交换位置，即升序排列  
    //return b - a; //b-a>0时交换位置，即降序排列  
});  
console.log(arr1);
```

► (5) [1, 2, 5, 4, 2]

► (4) [3, 9, 13, 42]

>

15.6.4数组索引方法

方法名	说明	返回值
indexOf()	数组中查找给定元素的第一个索引	如果存在返回索引号;如果不存在, 则返回-1
lastIndexOf()	数组中查找给定元素的最后一个的索引	如果存在返回索引号;如果不存在, 则返回-1

```
//1.indexOf() 返回第一个满足条件的索引号, 从前往后查找
var arr = ['red', 'blue', 'green', 'pink', 'green'];
console.log(arr.indexOf('green'));
//找不到返回-1
console.log(arr.indexOf('gray'));

// 2.lastIndexOf() 返回最后一个满足条件的索引号, 从后往前查找
console.log(arr.lastIndexOf('green'));
//同样找不到返回-1
console.log(arr.lastIndexOf('gray'));
```

```
2
-1
4
-1
>
```

案例:数组去重(重点案例)

有一个数组['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'],要求去除数组中重复的元素。

案例分析:

- ①目标:把旧数组里面不重复的元素选取出来放到新数组中,重复的元素只保留一个,放到新数组中去重。
- ②核心算法:我们遍历旧数组,然后拿着旧数组元素去查询新数组,如果该元素在新数组里面没有出现过,我们就添加, 否则不添加。
- ③我们怎么知道该元素没有存在?利用新数组.indexOf(数组元素)如果返回时-1就说明新数组里面没有该元素

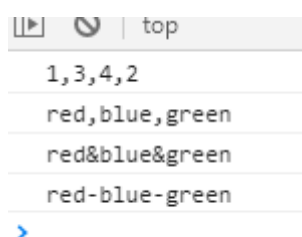
```
//封装一个去重的函数
function unique(arr) {
    var newArr = [];
    for (var i = 0; i < arr.length; i++) {
        if (newArr.indexOf(arr[i]) === -1) {
            // newArr[newArr.length]=arr[i];
            newArr.push(arr[i]); //上面语句的改进
        }
    }
    return newArr;
}
var arr = ['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'];
console.log(unique(arr));
```

注意案例分析过程

15.6.5数组转换为字符串

方法名	说明	返回值
toString()	把数组转换成字符串，逗号分隔每一项	返回一个字符串
join('分隔符')	把数组转换成字符串， 分隔符分隔每一项	返回一个字符串

```
// 1.toString()
var arr = [1, 3, 4, 2];
console.log(arr.toString());
//2.join('分隔符')
var arr1 = ['red', 'blue', 'green'];
console.log(arr1.join()); //无参数，默认用逗号分隔
console.log(arr1.join('&')); //有参数，用参数进行分隔
console.log(arr1.join('-')); //red-blue-green
```



输出结果字体颜色都为黑色，说明是字符串

15.7字符串对象

15.7.1基本包装类型

为了方便操作基本数据类型, JavaScript还提供了三个特殊的引用类型: String、Number和 Boolean。

基本包装类型就是把简单数据类型包装成为复杂数据类型,这样基本数据类型就有了属性和方法。

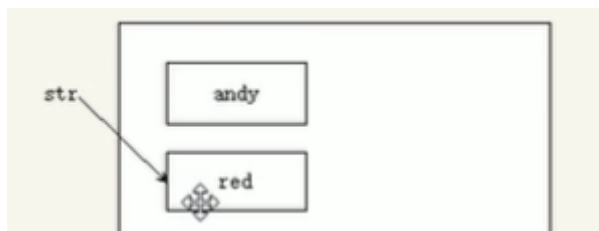
```
var str = 'auue';
console.log(str.length);
//按道理基本数据类型是没有属性和方法的，只有对象、复杂数据类型才有属性和方法
//这就是基本包装类型：将简单数据类型包装成复杂数据类型
//执行过程
// 1.生成临时变量，把简单类型包装为复杂数据类型
var temp = new String('andy');
// 2.赋值给我们声明的字符变量
str = temp;
// 3.销毁临时变量
temp = null;
```

15.7.2字符串的不可变

指的是里面的值不可变,虽然看上去可以改变内容,但其实是地址变了,内存中新开辟了一个内存空间。如下图

```
//字符串的不可变性
var str='andy';
console.log(str);
str='red';
console.log(str);
```

原先str指向'andy'，后来指向'red'，但'andy'仍在内存空间内，并没有消失。



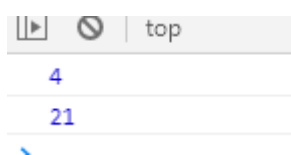
因为我们字符串的不可变，所以不要大量的拼接字符串，浪费内存

15.7.3根据字符返回位置

字符串所有的方法，都不会修改字符串本身(字符串是不可变的,操作完成会返回一个新的字符串。与数组方法类似

方法名	说明
indexOf('要查找的字符',开始的位置)	返回指定内容在原字符串中的位置，如果找不到就返回-1,开始的位置是index索引号
lastIndexOf()	从后往前找，只找第一个匹配的，即找最后一个匹配的索引号

```
//str.indexOf('要查找的字符',[起始点位置])
var str = '对于可控的事情，我们要保持谨慎；对于不可控的事情，我们要保持乐观';
console.log(str.indexOf('的'));
console.log(str.indexOf('的', 5));//这个5是指索引号5，从索引号5位置开始往后查找（包括5）
```



案例:返回字符位置

查找字符串"abcoefoxyozzopp"中所有o出现的位置以及次数

案例分析:

- ①核心算法:先查找第一个o出现的位置
- ②然后只要indexOf返回的结果不是-1就继续往后查找
- ③因为indexOf只能查找到第一个,所以后面的查找,利用第二个参数,当前索引加1 ,从而继续查找

```
var str = 'abcfoxyozzopp';
var index = str.indexOf('o');
var num = 0;
while (index !== -1) {
    num++;
    console.log(index);
    index = str.indexOf('o', index + 1);
}
console.log('o出现的次数:' + num);
```

```
3
6
9
12
o出现的次数: 4
```

15.7.4根据位置返回字符(重点)

方法名	说明	使用
charAt(index)	返回指定位置的字符(index字符串的索引号)	str.charAt(0)
charCodeAt(index)	获取指定位置处字符的ASCII码 (index索引号)	str.charCodeAt(0)
str[index]	获取指定位置处字符	HTML5, IE8+支持 和charAt()等效

```
var str = 'auue';
console.log(str.charAt(0));
//可以利用charAt遍历字符串
console.log(str.charCodeAt(0)); //可以用此明白用户按下了什么键
console.log(str[0]);
```

```
▶ ⌛ |
a
97
a
```

案例:返回字符位置

判断一个字符串'abcfoxyozzopp'中出现次数最多的字符,并统计其次数。

案例分析:

- ①核心算法:利用charAt()遍历这个字符串
- ②把每个字符都存储给对象, 如果对象没有该属性,就为1,如果存在了就+1
- ③遍历对象,得到最大值和该字符

利用对象的键值对存储每个字符及出现的次数, 最后遍历对象得到最大值及对应字符

```
// 判断一个字符串'abcfoxyozzopp'中出现次数最多的字符,并统计其次数。
//①核心算法:利用charAt()遍历这个字符串
```

```

    //②把每个字符都存储给对象，如果对象没有该属性，就为1，如果存在了就+1
    //③遍历对象，得到最大值和该字符
    var str = 'abcoefoxyozzopp';
    var o = {};

    for (var i = 0; i < str.length; i++) {
        var chars = str.charAt(i);
        if (o[chars]) {
            o[chars]++;
        } else {
            o[chars] = 1;
        }
    }
    console.log(o);
    //遍历对象得到最大值及字符
    var max = 0;
    var ch = '';
    for (var k in o) {
        //k得到的是属性名
        //o[k]得到的是属性值
        if (o[k] > max) {
            max = o[k];
            ch = k;
        }
    }
    console.log('出现最多的字符是：' + ch + ' 出现次数：' + max);

```

15.7.5字符串操作方法(重点)

方法名	说明
concat(str1,str2...)	concat()方法用于连接两个或多个字符串。拼接字符串,等效于+,+更常用
substr(start,length)	从start位置开始(索引号), length为截取的个数
slice(start, end)	从start位置开始，截取到end位置, end取不到(都是索引号)
substring(start, end)	从start位置开始，截取到end位置，end取不到，基本和slice相同，但是不接受负值
replace('被替换的字符','替换为的字符')	只能替换第一个字符
split('分隔符')	将字符串转换为数组，以字符串内的分隔符进行分隔转换

```

//concat() 连接字符串
var str = 'green';
console.log(str.concat('red'));

//substr('截取的起始位置','截取几个字符')
var str1 = '改革春风吹满地';
console.log(str1.substr(2, 2)); //从索引号2开始取2个字符

```

```
greenred
春风
> |
```

```
//3. 替换字符 replace('被替换的字符','替换为的字符'), 只能替换第一个字符
var str2 = 'auue';
console.log(str2.replace('u', 'r'));
//应用, 将字符串 'abcfoxyozzopp' 中所有的 'o' 替换为*
//思路: 配合indexOf实现
var str3 = 'abcfoxyozzopp';
while (str3.indexOf('o') !== -1) {
    str3 = str3.replace('o', '*');//注意替换完后要重新赋值给str3
}
console.log(str3);

//4. 字符串转换为数组split('分隔符'), 与join()类似
var str4 = 'green,red,pink';
console.log(str4.split(','));
var str5 = 'green&red&pink';
console.log(str5.split('&'));
```

```
arue
abc*ef*xy*zz*pp
▶ (3) ["green", "red", "pink"]
▶ (3) ["green", "red", "pink"]
```

方法名	说明
toUpperCase()	将字符串内字母全部转换为大写
toLowerCase()	将字符串内字母全部转换为小写

```
//5.toUpperCase() 转换为大写 toLowerCase() 转化为小写
var str6 = 'sDDsAdfKBBh';
console.log(str6.toUpperCase());
console.log(str6.toLowerCase());
```

```
SDDsAdfKBBh
sddsadfkbh
>
```

16.简单数据类型和复杂数据类型

16.1简单数据类型和复杂数据类型概述

简单类型又叫做基本数据类型或者**值类型**,复杂类型又叫做**引用类型**。

- 值类型**:简单数据类型/基本数据类型,在存储时变量中存储的是值本身,因此叫做值类型
string, number, boolean, undefined, null

- 引用类型**: 复杂数据类型,在存储时变量中存储的仅仅是地址(引用), 因此叫做引用数据类型

通过new关键字创建的对象(系统对象、自定义对象), 如Object、Array、Date等

注意：null返回的是一个空的对象。应用：若有一个变量我们以后打算存储为对象，但没有想好放什么，此时可设置为null

16.2堆和栈

堆栈空间分配区别：

1、栈(操作系统) :由操作系统自动分配释放存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈;**简单数据类型存放到了栈里面**

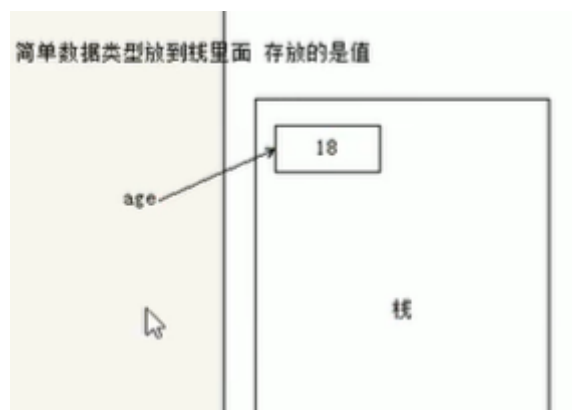
2、堆(操作系统) :存储复杂类型(对象)，一般由程序员分配释放,若程序员不释放,由垃圾回收机制回收。**复杂数据类型存放到了堆里面**



注意: JavaScript中没有堆栈的概念,通过堆栈的方式,可以让大家更容易理解代码的一些执行方式,便于将来学习其他语言

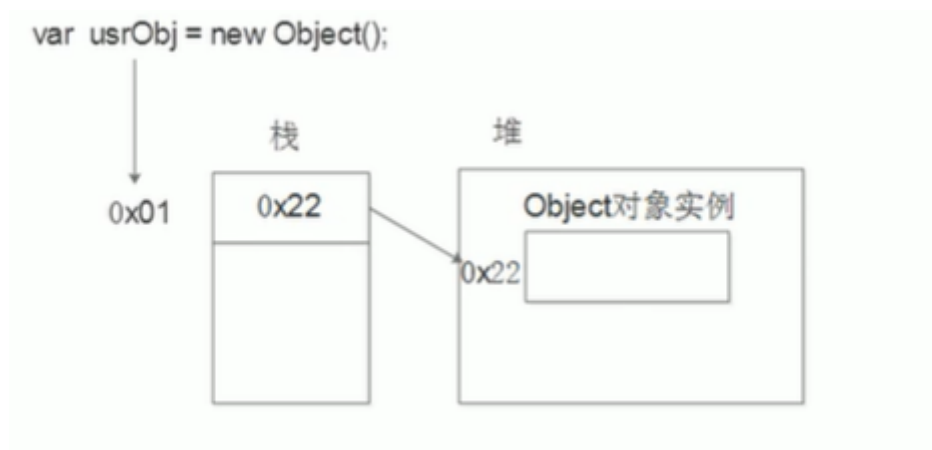
16.3简单类型的内存分配

- 值类型(简单数据类型): string , number , boolean , undefined , null
- 值类型变量的数据直接存放在变量(栈空间)中，里面直接开辟一个空间，存放的是值



16.4复杂数据类型的内存分配

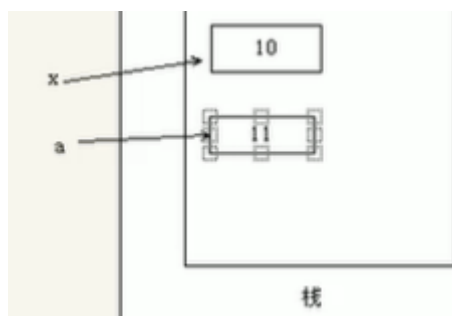
- 引用类型(复杂数据类型) :通过new关键字创建的对象(系统对象、自定义对象),如Object. Array. Date等
- 引用类型变量(栈空间)里存放的是地址,该地址为16进制，然后该地址指向堆里面的数据 (对象实例)



16.5简单类型传参

函数的形参也可以看做是一个变量,当我们把一个值类型变量作为参数传给函数的形参时,其实是把变量在栈空间里的值复制了一份给形参,那么在方法内部对形参做任何修改,都不会影响到的外部变量。

```
function fn(a) {
    a++;
    console.log(a) ;
}
var x = 10;
fn(x);
console.log (x);
```



传参时, 变量在栈空间里的值复制了一份给形参,即a指向栈空间内10 (值, 复制后的10, 而不是x所指的, 重新开辟一个空间), 因此后续对形参的修改不会影响原来x外部变量的值

输出结果:

```
11
10
```

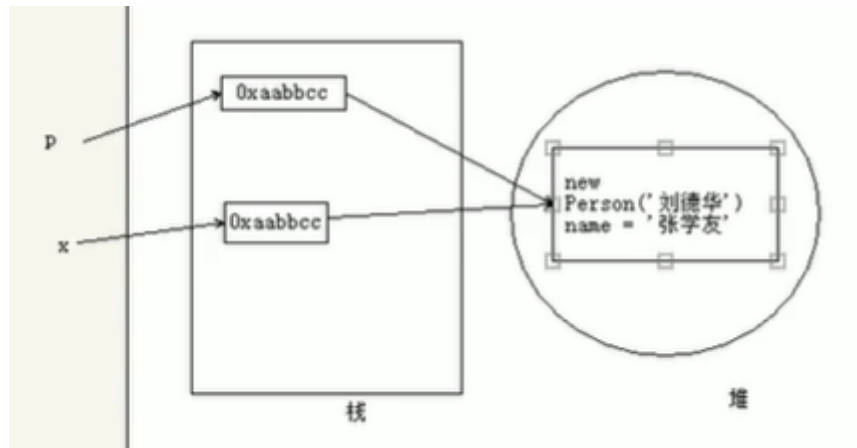
16.6复杂数据类型传参

函数的形参也可以看做是一个变量, 当我们把引用类型变量传给形参时,其实是把变量在栈空间里保存的堆地址复制给了形参,形参和实参其实保存的是同一个堆地址,所以操作的是同一个对象。

```

function Person (name){
    this.name = name ;
}
function f1(x) { //x=p
    console.log (x.name); // 2.这个输出什么？ 刘德华
    x.name = "张学友";
    console.log (x.name); // 3.这个输出什么？ 张学友
}
var p = new Person ("刘德华");
console.log (p.name); // 1.这个输出什么？ 刘德华
f1 (p);
console.log (p.name) ;// 4.这个输出什么？ 张学友

```



16.7传参小结

简单数据类型和复杂数据类型传参

相同点：

- 1.函数的形参都可以看作一个变量
- 2.传参的过程即赋值的过程， $x=p$ ，都需要在栈内重新开辟一个空间，让形参 x 指向栈空间

不同点：

- 1.简单数据类型，传参的过程复制的是**数值**；而复杂数据类型，传参的过程复制的是**地址**
- 2.简单数据类型，形参与外部变量互不影响；复杂数据类型，参数修改会**影响外部变量**，因为将地址所指向的堆里面的数据修改了。