1、什么JavaScript？

是一门客户端的脚本语言   ，   用来提供与web页面进行交互。

js可以实现：

     表单验证

     轮播图

     放大镜 …

2、js的历史背景

   布莱登.艾奇

**JavaScript**诞生于1995年。起初它的主要目的是处理以前由服务器端负责的一些表单验证。在那个绝大多数用户都在使用调制解调器上网的时代，用户填写完一个表单点击提交，需要等待几十秒，完了服务器反馈给你说某个地方填错了......在当时如果能在客户端完成一些基本的验证绝对是令人兴奋的。当时走在技术革新最前沿的**Netscape**（网景）公司，决定着手开发一种客户端语言，用来处理这种装简单的验证。当时就职于**Netscape**公司的布兰登·艾奇开始着手计划将1995年2月发布的**LiveScript**同时在浏览器和服务器中使用。为了赶在发布日期前完成**LiveScript**的开发，**Netscape**与**Sun**公司成立了一个开发联盟。而此时，**Netscape**为了搭上媒体热炒**Java**的顺风车，临时把**LiveScript**改名为**JavaScript**，所以从本质上来说**JavaScript**和**Java**没什么关系。

3、javascript的组成？

 ECMAscript   欧洲计算机制造商协会   不是一门语言   是一个组织     用来规定js的语法规范

 BOM   （浏览器对象模型）

 DOM   （文档对象模型）

4、javascript  代码实现 位置注意

 1、所有的js代码都要写在  script 的标签对中

 2、 script 标签放到页面的任意位置（除了title中）

 3、可以将js的代码写到 .js  文件中  ，通过  <script src="xx.js"></script>,此标签对中的js代码就不会被识别了。

5、javascript的几个输出语句

 alert()    通过弹出对话框的方式输出信息 （项目中不建议使用）   一般用于调试

 document.write();  在浏览器中打印结果

 console.log(); 控制台输出   一般用于调试中

6、js中的注释

 //   单行

 /\*\*/  多行

7、js中的字面量（常量）

数值：

     整数：

          十进制整数   0---9 组成

          二进制整数   0   1

          八进制整数   0--7组成      一般以  0  开头      023

          十六进制整数  0--9  a--f    一般以  0x 开头

     小数：

          0.1

          1.1  +  0.1 = 1.2

字符：

     js中的字符 可以通过  ""   或  ''  括起来

     注意引号嵌套问题：

          "  '    '  "   或   '  "   "  '

8、js中标识符（变量）的命名规范

1、js中的标识符只能由 字母、数字、下划线、$  组成（不建议用中文）

2、标识符 不能用  数字 开头

3、不能使用系统指定的关键字命名

9、js中的变量问题

变量：就是在程序运行的过程中，其值有可能会发生改变的量。

10、变量的定义和赋值

显示定义：

     通过 var  定义

     var iNum = 90;

隐式定义：

     不同var直接就定义变量

     iNum = 90;

变量必须先定义  后使用

11、js中输出结果时注意几个问题：

1、如果输出是一个变量  该变量一定不能加引号   ""

2、如果输出是一个字符类型   一定要加引号

3、如果输出的是一个数 ，可以加引号 也可以不加引号

4、如果输出的是字符 和 变量的组合，需要使用 +   拼接

12、变量的数据类型

基本数据类型

     字符型 string  ：

          数据值有多个

          通过引号引起来

     数值型 number ：

          数据值有多个  （整数和小数）

     布尔型  boolean ：

          数据值 有两个  true    false

复合数据类型

     对象   object

特殊数据类型

     null  空

     undefined  未定义

13、（扩展） 测试变量的数据类型

 typeof

     用法：

          typeof  变量名

          typeof(变量名)

typeof的返回值有哪些？

     number     boolean   string   object  undefined   function

     typeof typeof 数据  --->　string

14、运算符和表达式

 什么是表达式：

     一个字面量或一个变量或通过运算符连接的式子

     23           iNum       5+8

 什么是运算符：进行运算的符号

运算符有哪些？    每一个运算符对应的表达式的值

15、算术运算符

一、运算符 ：

     （双目）+  -  \*   /  %(模运算符)

     （单目）  ++    --

二、表达式的值

     \* :

        1、如果符号两侧都是纯数值类型  正常计算     6\*8 = 48

        2、如果符号两侧有一侧是纯数字字符串  纯数字字符串自动转成数值类型          6\*"3" = 18         "5"\*"9" = 45

        3、如果符号两侧有一侧是非数字字符串  结果  NaN （not  a   number）     不能正常计算的结果都是NaN

     -   /    %  (对应的表达式的值 同 \*  )

     + ： 表示连接    /    计算的加号

     数字 和 字符串相加 表示连接

     单目运算符： ++  自增       --  自减      注意：  ++ -- 的操作数必须是变量

     var i = 0;

     表达式：

          i++  (后置)

          ++i  （前置）

          i--

          --i

     ++i  和  i++  的异同：

          相同： 变量i的结果都会自增1

          不同：

               表达式的结果不同

               如果++在后，也就是i++，  先将i的值作为整个表达式的值， i再自增  （先赋值，后自增）

               var i = 10;

                                   j = i++;  //  i =  11       j = 10  (用变量j来代替整个表达式的值)

               如果++在前，也就是++i，  先将i自增，再将i的值作为整个表达式的值  （先自增，后赋值）

16、关系运算符

 一、运算符：  >  >=  <   <=   ==等于      ===全等于     != 不等于       !== 全不等

 二、表达式的值  ：  true   false

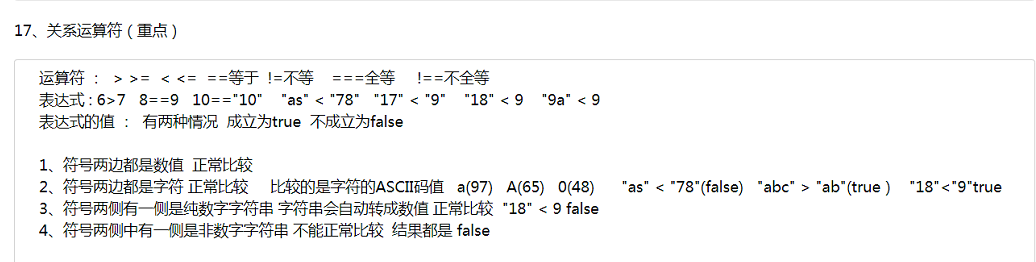
     如果关系表达式成立，结果为 true    否则为false

     1、如果运算符两侧都是数值型  正常比较

     2、如果运算符两侧都是字符型  正常比较 （通过键盘按键的ASCII码值进行比较的  0--48    a--97     A--65）

     3、如果运算符两侧有一侧是纯数字字符串  自动转换后正常比较

     4、如果运算符两侧有一侧是非数字字符串  不能正常比较  ， 结果是false



三、  ==   和  ===  的区别：

     == 等于  只比较结果值，不比较类型    100  == “100”    true       100!="100" false

     === 全等  先比较类型  在比较结果值   100  === “100”    false      100 !== "100"  true

17、逻辑运算符

一、运算符  ||或          && 与       ! 非

二、逻辑表达式的值   ： true   false  （一般参与逻辑运算的都是布尔值）

    &&（并且）：   如果&&  两侧都为真，结果为true               地球是方的 &&  姚天宇是男生  -- false           地球是圆的  &&  姚天宇是男生   ---  true

                             4>5 && 8==8     false

     || ( 或者 )     ：  如果 ||  两侧都为假 ，结果为false

                            4>5  ||  8==8    true

     ! 非 :　　　　　！false   --- true    !true ----false

     变量 num 既能被5整除  又能 被3整除     num%5==0 && num%3==0

18、 条件运算符   三目    ？  ：

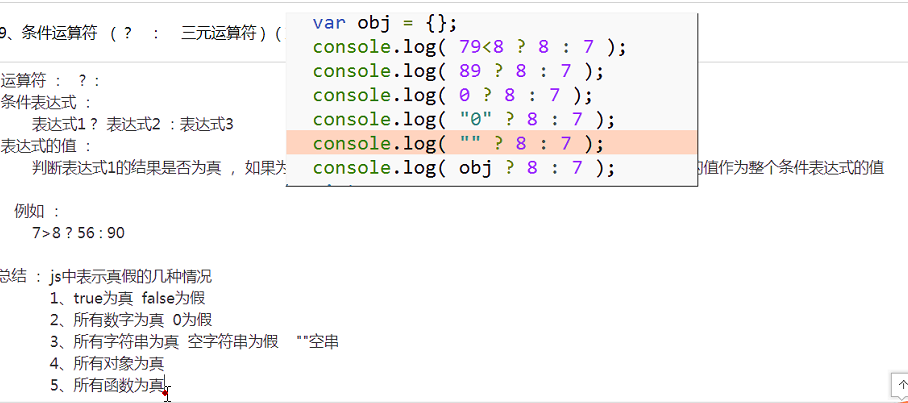
一、运算符   ？  ：

二、表达式的值

     表达式1 ？  表达式2  :　表达式3

   如果表达式1的结果为真 ，  整个条件表达式的值 为表达式2的值

   如果表达式1的结果为假，   整个条件表达式的值 为表达式3的值



      6、null和undeline

toFixed(保留位数)

19、赋值运算符

=  赋值

var i = 9 ;   将 9 赋值给 i

赋值符号要求  左侧必须是一个变量        90 = i;  错    i+89 = 90;  错      num = 9+8 对

复合赋值运算符    +=   -=   \*=    /=    %=

     var i = 1;

     表达式：　　i+=10;   等价   i = i + 10

                i+=1     等价   i = i + 1   等价  i++

20、运算符的优先级

() > 单目 > 算术  >  关系  >　逻辑　> 条件  >  赋值

          \*/%  +-           &&  ||

21、类型转换

自动类型转换

     在算术运算和关系运算中， 纯数字字符串会自动转换成数值进行计算    null 会自动转换成0     undefined 会自动转换成 NaN

     1 + null  = 1

     1 + undefined  = NaN

     "a" > undefined   false

强制类型转换  :  强制将某个类型转成另一个类型，如果不能正常转换，结果就是NaN

     parseInt()  将字符串转成整数

          parseInt("12")  --   12

          parseInt("12ab") --  12

          parseInt("ab12ddd") -- NaN

          parseInt("ab12") ---   NaN

          parseInt("12a34"); --12

     parseFloat()  将字符串转成小数

          parseFloat("12.34ab34"); --   12.34

          parseFloat("12ab12.34"); --   12

     Number()  尝试将某个类型转成 数值型  如果不能转换，值为NaN

          Number( true )   1

          Number( false )  0

          Number("12")      12

          Number( "12ab" );  NaN

          Number( "12ab34" )  NaN

      toString()  尝试将某个类型转成字符串

22、isNaN()  判断非数字   如果参数是非数字(不是数字)，结果是true    否则是false

isNaN("12")  false

isNaN(12)    false

isNaN("12ab")  true

isNaN("ab")    true

eval() 将一个字符串转成表达式  并返回表达式的值 (将引号去掉)

alert( "2+3" );

alert( eval("2+3") );

 eval("alert(2)");

23、程序的三大结构

顺序 ： 程序按照从上到下顺序来执行

选择 ： 程序按照某个条件执行 （出口只有一个）

循环 ： 反复执行

24、选择结构

选择结构：

     单分支选择结构（一个条件）

     多分支选择结构（多个条件）

单分支结构控制语句：

     if(条件){

          控制语句；

     }

     说明： 条件一般是 关系表达式 或  逻辑表达式

     执行逻辑：

           如果条件结果为 true  ， 就执行控制语句

     if(条件){

         控制语句;

     }else{

          控制语句;

     }

     执行逻辑：

          如果条件结果为true ，就执行if控制语句 ； 否则 就执行 else 控制语句

注意布尔环境下的if条件：

     1、 非0 表示真，0 表示 假

     if( 20 ){

          alert("ok");  // ok

     }else{

          alert("no");

     }

     2、 字符串表示真 ，空字符串表示假

     if( "" ){

          alert("ok");

     }else{

          alert("no"); // no

     }

     3、 null 和 undefined 表示 假

   if( null ){

          alert("ok");

     }else{

          alert("no"); // no

     }

     4、 所有对象表示真

25、多分支结构

控制语句：

if(条件1){

    控制语句；

}else if(条件2){

     控制语句

}....

else{

     else 控制语句

}

执行逻辑：

     首先判断条件1是否成立，如果成立就执行控制语句，然后跳出if结构

                         如果不成立，顺序向下判断，只要有满足条件的，就执行控制语句

                         如果上面的条件都不满足，执行else语句

26、if语句 的嵌套

if(){

     if(){

     }

}else{

     if(){

     }else{

     }

}

27、关于if语句需要注意

1、if语句后面如果只有一条语句，花括号可以省略       else 同理 （不建议省略）

2、else 可以省略

3、else 永远与离他最近的同级if语句配对

28、switch（开关语句）--多条件

控制语句：

switch(表达式){

     case 常量表达式 : 语句;

     case 常量表达式 : 语句;

       .....

      default : 语句;

}

说明：

     1、表达式 一般是一个变量  或 一个有具体值的式子

     2、 default 语句 可以省略

     3、如果case 后面没有 break ， 就会实现case穿透

执行逻辑：

     判断表达式的值和哪一个case后面的常量表达式相等，有满足条件的，就执行对应的case 语句

     如果case语句后面有break，则中断执行

     如果case语句后面没有break ，则实现case穿透

29、多分支 if  else if  和 switch 的使用场景

当条件是一个范围值时，使用 if  else...if

当条件是一个具体值时，使用  switch

循环结构

     循环控制语句

     当型循环

     while(表达式){

          循环体;

     }

     说明：

           1、表达式一般是条件表达式

           2、循环体如果有一条语句，花括号可以省略 （不建议省略）

     循环的五要素：

          1、循环变量

          2、循环变量初始值

          3、循环变量终值（循环条件）

          4、循环变量增值

          5、循环体 （反复执行的代码）

     while循环的执行过程：

          首先判断表达式是否成立，如果不成立，循环体一次也不执行，直接跳出while循环

                                如果成立，进入循环体执行，执行完循环体后，又继续判断表达式，直到表达式不成立，退出while循环

30、do...while(循环)

控制语句结构：

do{

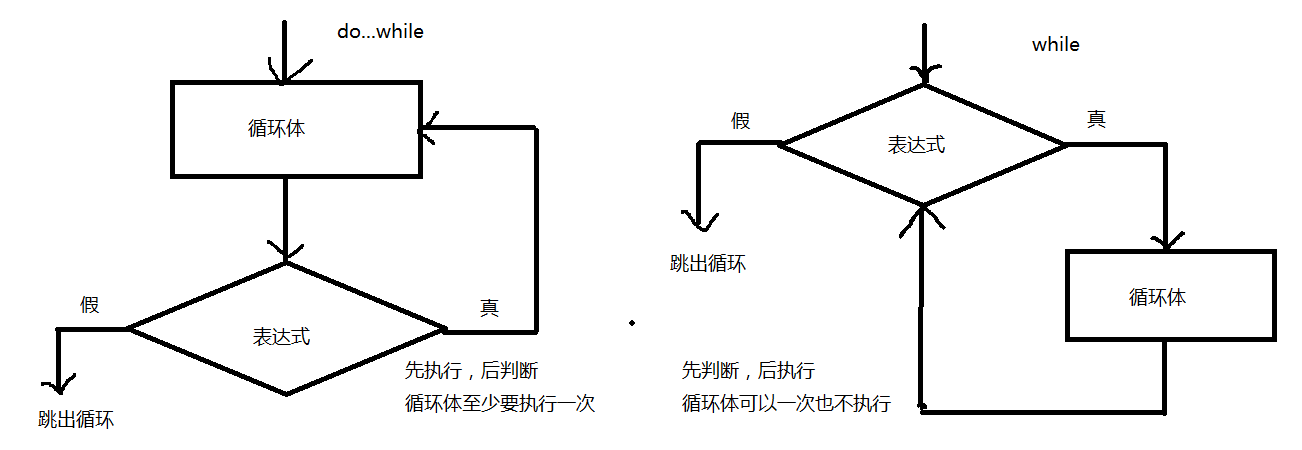
}while(表达式)

     说明：  表达式一般是条件表达式

       执行逻辑：先执行一次循环体，然后再判断表达式是否成立，如果不成立，退出循环体

      如果成立，继续执行循环体，直到表达式不成立退出循环体

while和do...while的区别：



31、死循环

循环条件永远成立

32、for循环  （for ...  in）

for循环控制语句：

for(表达式1;表达式2;表达式3){

     循环体;

}

说明：

     表达式1 一般是为循环变量赋初值

    表达式2 一般是循环变量的终值 （循环条件）

    表达式3  循环变量的增值

    表达式1 可以省略 ，提到for的上面

    表达式3 可以省略 ，提到循环体中

    表达式2 不可以省略，如果省略了表达式2，就会出现死循环。

执行逻辑：同 while循环

33、break 和 continue

continue：结束本次循环，继续进行下一次循环。

break: 结束本层循环 （结束break所在的那一层循环）

34、while 和 for的使用场景

明确了循环次数，一般使用for

没有明确循环次数的情况下，建议使用while

35、循环嵌套

36、函数的概念

函数的概念：函数就是一个工具，也就是具有某个特定功能的一小段代码块

37、函数定义

方式一：函数调用可以在任意位置

     function 函数名(){

          函数体;

     }

方式二：表达式定义---先定义，后调用，函数的调用一定是在定义后

var fn = function(){

     函数体;

}

38、函数的调用

函数名([参数]);

函数的调用还可以通过事件来调用

39、函数的参数

形式参数（形参）

     形参一般出现在函数定义中

     形参一般是一个变量

实际参数（实参）

     实参一般出现在函数调用中

     实参可以是变量，常量、表达式，但要求一定要有值

     参数的传递是 将实参传递给形参，传递时一一对应

     当实参的个数大于形参个数时，多余的实参自动舍去

     当形参的个数大于实参个数时，多余的形参值为undefined

40、函数的返回值

如果一个函数的结果 有其它用途时，可以将这个函数的结果返回

通过 return 返回函数的结果

一个函数可以没有返回值

如果函数有返回值，只能有一个

一个函数 遇到return，return后面的代码不会执行

**总结函数特性 ：**

**可以进行代码的封装**

**在合适的时机调用**

**有作用域**

**可以根据传入参数的不同获得不同的结果**

41、函数的递归调用

递归调用 ：函数自己调用自己

递归的本质　：实现了循环

递归实现 n！

function fac(n){     // 循环 初始值

     if( n == 1 ){  //循环 条件

          return 1;

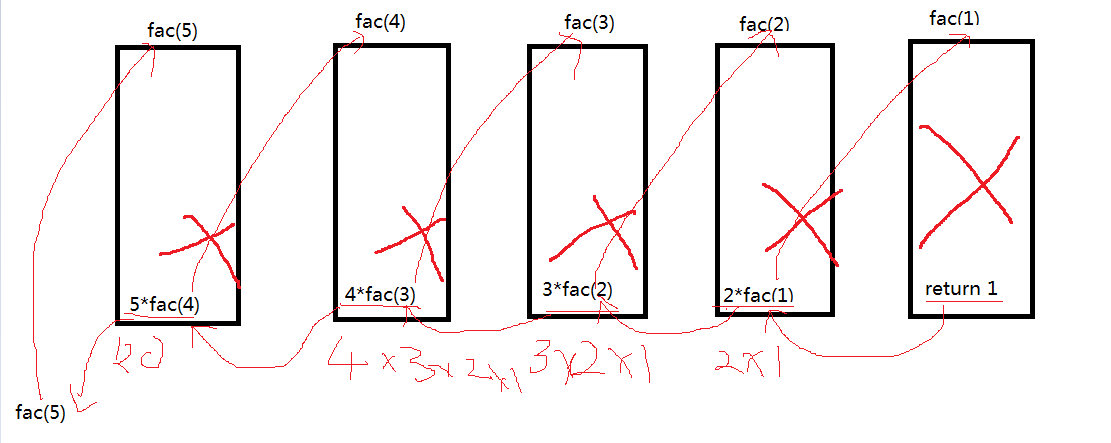
     }else{

          return n\*fac(n-1);  //步长　　　实现递归调用

     }

}

fac(5);



42、变量的作用域

作用域： 作用范围

全局变量：

     程序从开始到结束，变量都起作用

     在函数外面定义的变量，就是全局变量；

     隐式全局变量： 在函数体内部没有用var定义的变量 也是全局变量

局部变量：

     在函数体内部通过var定义的变量

     局部变量的作用范围  （函数体内部，函数调用完成后，局部变量自动销毁）

所有的形参都是局部变量

局部变量的优先权大于全局变量

43、变量的提升

在函数体内部定义的变量，会将变量提升到函数体的最顶端，只提升变量的声明，并不赋值

     function fun(){

          console.log(m);

          var m = 10;

          console.log(m);

     }

   等价

     function fun(){

          var m;

          console.log(m);

          m = 10;

          console.log(m);

     }

44、函数中的实参副本 arguments（扩展）

arguments使用说明：

     arguments只能用在函数体内部

    arguments代表的是实参副本   是一个对象

    可以通过arguments[下标] 得到所有的实参  下标从0开始  （索引、地址）

45、函数和事件的关系

所有的事件都要依赖与函数来执行

46、事件 三要素

事件源         名词

事件           动词

事件处理程序    要做的具体的事情

47、事件的种类

页面加载事件

     onload  页面元素全部加载完成才执行onload的内容  （所有内容加载后才执行onload的内容 --  图片  css   js   html等...）

     window.onload = function(){...}

     将onload事件写到body标签上  作为标签的一个属性出现    <body onload = "fun()"></body>

                                                       <script> function fun(){.....} </script>

鼠标事件

     onclick 单击

     ondblclick 双击

     onmouseover 鼠标划过（移入）

     onmouseout  鼠标移出 （移出）

     onmousedown 点击  （拖拽时用到）

     onmouseup  抬起

     onmouseenter  鼠标移入

     onmouseleave  鼠标移出

     onmousemove   鼠标移动

     oncontextmenu  右键单击

     onmousewheel   滚轮事件

键盘事件

     onkeyup  键盘抬起

     onkeydown 键盘按下

     onkeypress  按下 + 抬起

其他事件（针对于表单）

     onblur       失去焦点事件

     onfocus      获取焦点事件

     onchange     改变事件

     onsubmit     表单提交事件