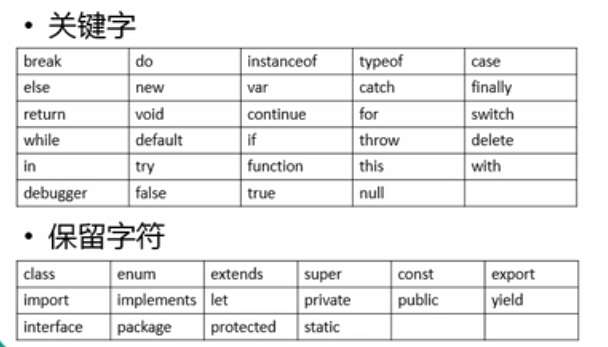
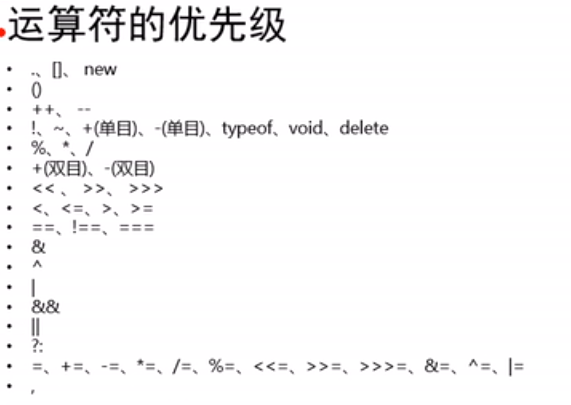
JS中变量没找着会报错 属性没找着会返回Underfind





<script type="text/javascript">

//控制浏览器弹出一个警告框

alert("哥你蒸煮爱！！");

//向网页输出一段文字

document.write("达达的网站!");

//向控制台输出一段文字

console.log("你猜我在哪显示啊！")

</script>

<body>

//可以将js代码写到标签中的onclick属性中当点击按钮时，js代码才会执行 //虽然可以写在标签中但是他们属于结构与行为的耦合，不方便维护，不推荐使用

<button type="button" onclick="alert('讨厌你点我干嘛!')">点我一下</button>

<a href="javascript:alert('让你点你就点!');">你也点我一下</a>

</body>

1.js中严格区分大小写

2.JS中每条指令都以分号结尾

-不写分号浏览器会自动添加，但是会影响性能，而且有可能加错分号

3.JS中会自动忽略空格和换行，所以可以使用换行来美化代码

<script type="text/javascript">

字面量，是一些不可改变的值

比如：1 2 3 4 5 6

字面量都是可以直接使用，但是我们一般都不会直接使用字面量

变量，变量可以用来保存字面量。而且变量的值是可以任意改变的

变量更加方便使用，使用在开发中通常使用变量来 保存一个字面量

可以通过变量对字面量进行描述

使用方法：首先要申明一个变量

var a;

a = 4564564561;

//申明和赋值一起完成

var b = 46486116848;

console.log(a);

</script>

**数据类型指的就是字面量的类型**

<script type="text/javascript">

数据类型指的就是字面量的类型

在JS中一共有六种数据类型

-String 字符串

-Number 数值

-Boolean 布尔值

-Null 空值

-Underfind 未定义

-Object 对象

其中String Number Boolean Null Undefined属于基本数据类型

而Object属于引用数据类型

String字符串

-在JS中字符串需要使用引导号引起来

-使用单引号或者双引号都可以

-引号不能嵌套 双引号里可以放单引号

-在字符串中我们可以使用\作为转义字符

当表示一些特殊字符时可以使用\进行转义

\" 表示"

\' 表示'

\n 表示换行

\t 表示制表符

\\ 表示\

var str = "hello";

console.log("hello");

console.log(str);

var = qqq;

qqq = "我说：\"今天\t天气真不错！\"";

var = sss;

sss = "\\\\\\";

console.log(qqq);

console.log(sss);

</script>

<script type="text/javascript">

在JS中所有的数值都是Number类型包括整数和浮点数（小数）

JS中可以表示数字的最大值

-Number.MAX\_VALUE

JS中可以表示数字的最小值

-Number.MIN\_VALUE

console.log(Nnmber.MAX\_VALUE)

如果使用的数大于最大数就返回infinity表示正无穷 也是数值

NaN是一个特殊的数字 属于Number类型，表示 Not a Number

如果使用js进行浮点数计算 可能得不到一个准确的值

例如： var c = 0.1 + 0.2;

console.log(c);

var a = 123;

var b = "123";

console.log(a);

console.log(b);

可以使用一个运算符typeof来检查变量类型

语法：typeof 变量

console.log(typeof a);

</script>

Null类型的值只有一个，就是null

-null这个值专门用来表示一个空的对象

-使用typeof检查一个null值时，会返回object

Underfind类型的值只有一个就是Underfind

-当申明一个变量但是不给她赋值时她的值就是Underfind

**强制类型转换**

<script type="text/javascript">

强制类型转换

-指将一个数值强制转换为其他数据类型

-类型转换主要指，将其他的类型转换为String Number Boolean

将其他的数据类型转换为String

方法一：

-调用被转换数据类型的toString()方法

-该方法不会影响到原变量，只是将转换的结果返回

-但是Null和Underfind没有toString()方法

方法二：

-调用String()函数，并将被转换的数据作为参数传递给函数

-使用String()函数做强制方法转换

对于Number和Boolean实际就是调用toString()方法

它会将Null直接转换为字符串“unll”

将Underfind直接转化为字符串"Underfind"

var a = 123; //a 为Number类型

//调用a的toString()方法

//调用xxx的yyy()方法,就是xxx.yyy()

var b = a.toString();

//a任然是Number类型 toString 方法是将a的值转换为String类型赋值给b

//即 b现在为String类型

a = 123;

//调用String()函数,将a转换为字符串

c = String(a);

</script>

<script type="text/javascript">

将它的数据类型转换为Number

转换方式一：

使用Number()函数

-字符串 -->数字

1.如果是纯数字的字符串，则转换为数字

2.如果字符串中有非数字内容，则转换为NaN

3.如果字符串是一个空串或者是一个全是空格的字符串则转换为0

Boolean -->数字

true 转为 1

false 转为 0

-null --> 数字 0

-Underfind --> 数字 0

转换方式二：

-专门用来对付字符串

-parseInt()函数 把一个字符串转换为一个整数 是将要转换的对象先转换为字符串然后在转换为Number

-parseFloat()把一个字符串转换为一个浮点数 是将要转换的对象先转换为字符串然后在转换为Number

var a = 123;

//调用Number()函数将a转换为Number类型

a = Number(a);

console.log(typeof a);

console.log(a);

var b = "123px";

//调用parseInt()函数将b转换为Number

//parseInt() 可以将一个字符串中的有效的整数内容取出来然后转换为Number

b = parseInt(b);

console.log(typeof b); 结果为number

console.log(b); 结果为123

var c = "123.456px";

//调用parseFloat()方法将字符串转换为Number

c = parseFloat(c);

console.log(c); 结果为123.456

console.log(typeof c); 结果为number

</script>

<script type="text/javascript">

在JS中，如果要表示16进制的数，则需要以0x开头

表示8进制的数字则需要以0开头

表示二进制数字需要以0b开头**不是所有浏览器都支持**

//16进制

var a = 0x456;

//8进制

var b = 070;

//二进制

// var c = 0b4564;

console.log(a);

console.log(typeof a);

console.log(b);

console.log(typeof b);

// console.log(c);

// console.log(typeof c);

var d = "070";

d = parseInt(d,10); //以十进制解析

var e = parseInt(d,8); //以八进制解析

console.log(d);

console.log(typeof d);

console.log(e);

console.log(typeof e);

</script>

将其它类型转换为Boolean

-数字 --> 布尔值

-除了0和NaN，其余都是true

-字符串 --> 布尔值

-除了空串，其余都是true

-null和Underfind都会转化为false

0对象也会转换为true

**14.运算符**

<script type="text/javascript">

运算符也叫操作符

通过运算符可以对一个或者多个值进行运算，并获取结果

比如：typeof就是运算符，可以获取一个值的类型

它会将该值的类型以字符串的形式返回

算数运算符

当对非Number类型的值进行运算时，会将这些值转换为Number然后进行运算

任何值和NaN做运算都得NaN

+

+可以把两个值进行加法运算，并将结果返回

如果对两个字符串进行加法计算则会进行拼串操作

任何的值和字符串做加法，都会先转换为字符串然后然后相加

-

-可以把两个值进行减法运算，并将结果返回

和任何值相减都会先转换为Number然后在相减

\*

\*可以把两个值进行乘法运算，并将结果返回

和任何值相乘都会先转换为Number然后在相乘

/

\*可以把两个值进行除法运算，并将结果返回

和任何值相除都会先转换为Number然后在相除

var a = 123;

var result = typeof a; //通过typeof的运算得到一个返回值赋给result

console.log(typeof result);//result的类型为String类型 因为typeof的返回值是String类型

var b = true + false;

console.log(b);

var c = "你好"+"大帅哥"+"！";

console.log(c);

//双引号只在同一行显示有效 当需要多行文字时需要多个双引号加上 加号(+)

var d = "锄禾日当午，"+

"汗滴禾下土。"+

"谁知盘中餐，"+

"粒粒皆辛苦！";

console.log(d);

var e = true + "false";

console.log(e); //结果为truefalse

//我们只需要为任意的数据类型 + 一个 "" 即可将其转换为String,这是一个隐式的类型转换由浏览器调用 String()方法实现

var c = 123;

c = c + ""; //将c转换为字符串

console.log(typeof c); //结果为string

console.log(c); //结果为123

任何值做- \* / 运算时都会自动转换为Number 利用这一特点做隐式类型转换

可以通过为一个值-0 \*1 /1 来将其转换为Number

原理和Number()函数一样使用起来更简单

</script>

<script type="text/javascript">

正号+ 负号- 可以将字符串转换为Number

var a = 1 + +"5" +9; //代码中字符串5起前面的加号就将字符串5转换为number5

console.log("a = "+a); //结果为11

</script>

**16.自增自减**

<script type="text/javascript">

自增

-通过自增可以是变量在自身的基础上增加1

-对于一个变量自增以后，原变量的值会立即增加1

-自增分成两种：后++（a++）和 前++（++a）

无论是a++ 还是 ++a，都会立即使原变量的值自增1

不同的是a++ 和 ++a 的值不同

a++的值等于原变量自增前的值

++a的值等于原变量自增后的值

自减

-通过自减可以是变量在自身的基础上减加1

-对于一个变量自减以后，原变量的值会立即减加1

-自减分成两种：后--（a--）和 前--（--a）

无论是a-- 还是 --a，都会立即使原变量的值自减1

不同的是a-- 和 --a 的值不同

a--的值等于原变量自减前的值

--a的值等于原变量自减后的值

//使a自增1

var a=8;

a++; //与 ++a; 结果一样

console.log("a = "+ a);

var b = 1;

console.log(b++);

console.log("b = " + b);

var c = 1;

console.log(++c);

console.log("c = " + c);

var d = 1;

d++; //d自增1

console.log(d); //结果为2

console.log(d++); //d++表示自增之前的值 使用结果为2

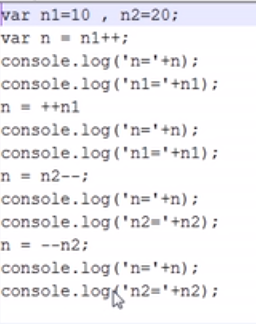
console.log(d); //d为d++ 和 console中d++ 自增之后的值所以结果为3

console.log(++d); //++d表示自增之后的值所以结果为4

console.log(d); //d为d++ console中d++ 和 console中++d 自增之后的值 结果为4

</script>

**小练习**



**18.与或非布尔值运算**

<script type="text/javascript">

/\*

JS中为我们提供了三种逻辑运算符

! 非

-!可以用来对一个值进行非运算

-非运算就是对布尔值取反操作 true变false false变true

-如果对一个非布尔值进行运算，则会先将其转换为布尔值，然后在取反

&& 与

-&&可以对符号两侧的值进行与运算并返回结果

-运算规则

-两个值只要有一个为false就返回false。

-只有两个值都为true时，才会返回true。

JS中的“与”属于短路的“与” 如果第一个值为false，则不看第二个值

|| 或

-||可以对符号两侧的值进行与运算并返回结果

-运算规则

-两个值只要有一个为true就返回true。

-只有两个值都为false时，才会返回false。

-JS中“或”属于短路的“或” 如果第一个值为true，则不看第二个值

\*/

var a = true;

//对a进行非运算

a = !a;

console.log("a = "+a);

//第一个值为true,会检查第二个值

//true && alert("看我出不出来！");

//第一个值为false时,不会检查第二个值

// false && alert("看我出不出来！");

//第一个值为false,会检查第二个值

//false && alert("看我出不出来！");

//第一个值为true时,不会检查第二个值

// true && alert("看我出不出来！");

</script>

**19.赋值运算符**

<script type="text/javascript">

&& || 非布尔值情况

&&运算

-对于非布尔值进行运算时会将其转换为布尔值，然后运算并返回原值

-如果第一个值为true则返回第二个值

-如果第一个值为false则返回第一个值

||运算

-对于非布尔值进行运算时会将其转换为布尔值，然后运算并返回原值

-如果第一个值为false则返回第二个值

-如果第一个值为true则返回第一个值

//true && true

//与运算:如果两个值都为true,则返回后面的值

var result = 5 & 6;

console.log("result = "+result); //结果为6

//与运算如果两个值中有false则返回靠前的false

</script>

=

可以将符号右侧的值赋值给符号左边的变量

+=（中间没有空格）

a += 5 等价于 a = a + 5

-=

a -= 5 等价于 a = a - 5

\*=

a \*= 5 等价于 a = a \* 5

/=

a /= 5 等价于 a = a / 5

%=

a %= 5 等价于 a = a % 5

**20.关系运算符**

<script type="text/javascript">

/\*

通过关系运算符可以比较两个值之间的大小关系

关系成立返回true 如果关系不成立返回false

number和非number值转换为number值然后比较返回true or false

任何值和NaN作比较都是false

符号两边边的值都为字符串时不会转换为number值进行比较而会比较字符串的字符编码

比较字符串编码时是第一位和第一位的比第二位和第二位比

如果两位一样则比较下一位，所以可以借用它来进行英文排序比较英文没有意义

所以在比较两个字符串型的数字时，一定要转换为number型

> 大于号

-判断符号左侧符值是否大于右侧的值

-如果关系成立则返回true，如果关系不成立则返回false

>= 大于等于号

-判断符号左侧符值是否大于或等于右侧的值

-如果关系成立则返回true，如果关系不成立则返回false

< 小于号

<= 小于等于号

\*/

var result = 5 > 10; //false

console.log("result = "+result); //结果为 result = false

console.log("a" < "b"); //结果为 true

/\*

== 判断是否相等

\*/

//Underfind 衍生自 null 所以两个值做相等判断时,会返回true

console.log(undefined == null); //结果为true

//NaN不和任何值相等,包括他本身

console.log(NaN == NaN); //结果为false

//可以通过isNaN()函数来判断一个值是否是NaN

var b = NaN;

console.log(isNaN(b));

/\*

不相等 != 与 == 相反语法类似

===

全等

-用来判断两个值是否全等，和相等类似，不同的是不会自动类型转换

类型不同直接返回false

!==

不全等

-用来判断两个值是否不全等，和不相等类似，不同的是不会自动类型转换

类型不同直接返回true

\*/

</script>

**21.条件运算符**

<script type="text/javascript">

/\*

条件运算符也叫三元运算符

语法：

条件表达式?语句1:语句2;

-执行流程：

条件运算符在执行时，首先对表达式进行求值，

如果该值为true，则执行语句1，并返回结果

如果该值为false，则执行语句2，并返回结果

\*/

//false?alert("语句1"):alert("语句2"); //改代码执行的是语句2

//true?alert("语句1"):alert("语句2"); //改代码执行的是语句1

var a = 10,b = 20,c = 30;

var max = a > b ? a : b;

max = max > c ? max : c;

console.log("最大值为："+max);

var d = 40,e = 50,f = 54;

var min = d < e ? d < f ? d : f : e < f ? e : f; //var min = d < e ? (d < f ? d : f) : (e < f ? e : f);

console.log("最小值为:"+min);

</script>

**22.运算符的优先级**

<script type="text/javascript">

/\*

, 运算符

使用 , 可以分割多个语句，一般可以用来声明多个变量使用

\*/

/\*

JS中加减乘除运算符的优先级和数学中的一样

JS中有运算符有优先级表 不确定的可以使用()来提高优先级

\*/

</script>

程序是由一条条语句构成的语句是自上而下顺序一条条执行的

在JS中可以使用{}来为语句进行分组

同在一个{}中的语句成为一组语句 他们要么都执行要么都不执行

一个{}中的语句也称代码块后面不要添加 ;

JS中代码块，只有分组的作用，没有其他作用

代码块内的内容在外部也能访问的到

**24.if'语句**

<script type="text/javascript">

/\*

流程控制语句

-JS中的程序是从上到下一行一行执行的

-通过流程控制语句可以控制执行程序

使程序可以根据一定条件来选择执行

-语句分类：

1.条件判断语句

2.条件分支语句

3.循环语句

条件判断语句：

-使用条件判断语句可以在执行某个语句之前进行判断条件成立执行不成立不执行

-if语句

-语法一：

if(条件表达式)

语句

if语句在执行时，会先对条件表达式进行求值判断，

如果条件表达式的值为true，则执行if语句，反正则不执行if语句

if语句只能控制紧随其后的语句 但是可以放在一个代码块中解决

if后的代码块尽量写上即使if后只有一条语句

\*/

// if(true)

// alert("你猜我出来不")

var a = 15;

if(a > 10 && a < 20){

alert("a大于10且a小于20！");

}

</script>

**If语句2**

<script type="text/javascript">

/\*

if语句

语法二：

if(条件语句){

语句....

}else{

语句....

}

if...else....语句

当语句执行时，会对if语句表达式进行求值判断

true执行if后面的语句

false执行else后面语句

语句三：

if(表达式){

语句1....

}else if(表达式){

语句2....

}else if(表达式){

语句3....

}else{

语句4....

}

该代码块只有一个语句被执行 一个被执行就结束

\*/

var age = 60;

if(age >= 60){

alert("你可以退休了！");

}else{

alert("您还不能退休！");

}

</script>

**If语句练习1**

<script type="text/javascript">

/\*

从键盘输入小明的成绩：

当成绩为100时奖励一辆宝马

当成绩为[80~99]时奖励一台iphonex

当成绩为[60~80]时奖励一本参考书

其他没奖励

\*/

/\*

prompt()可以弹出一个提示框，该提示框中会带有一个文本框，

用户可以在文本框中输入一段内容，该函数需要一个字符串作为参数

该字符串将作为提示框的提示文字

prompt返回值是String类型的

用户输入的内容将作为返回值返回，可以定义一个变量来接受这个返回值

\*/

var score = prompt("请输入小明的期末成绩！");

if(score > 100 || score < 0 || isNaN(score)){

alert("拉出去毙了！");

}else{

if(score == 100){

alert("奖励一辆宝马！");

}else if(score >= 80){

alert("奖励一台iPhoneX");

}else if(score >= 60){

alert("奖励一本参考书！");

}else{

alert("洗干净准备挨揍!");

}

}

</script>

**If语句练习2**

<script type="text/javascript">

/\*

男大当婚，女大当嫁，女方家长要嫁女儿，要求是：

身高：180cm以上； 存款：1000万以上； 帅气值：500以上；

如果同时满足三个，则：‘我一定嫁给他’

如果三个有一个满足，则‘嫁吧，比上不足，比下有余。’

如果第三个都不满足，则：‘我一定不嫁’

\*/

var stature = prompt("身高多少！");

var deposit = prompt("存款多少万！");

var handsome = prompt("帅气值多少!");

if(stature > 180 && deposit >1000 && handsome > 500){

alert("我一定嫁给他");

}else if(stature < 180 && deposit <1000 && handsome < 500){

alert("我一定不嫁!");

}else{

alert("嫁吧，比上不足，比下有余.");

}

</script>

**If语句练习3**

<script type="text/javascript">

/\*

编写程序，由键盘输入是三个整数num1、num2、num3

对他们进行排序，并且从大到小输出

\*/

var num1 = +prompt("输入第一个数！");

var num2 = +prompt("输入第二个数！");

var num3 = +prompt("输入第三个数！");

var a,b,c;

if(num1 > num2 && num1 > num3){

if(num2 > num3){

alert(num1+","+num2+","+num3);

}else{

alert(num1+","+num3+","+num2);

}

}else if(num2 > num1 && num2 > num3){

if(num1 > num3){

alert(num2+","+num1+","+num3);

}else{

alert(num2+","+num3+","+num1);

}

}else if(num3 > num1 && num3 > num2){

if(num1 > num2){

alert(num3+","+num1+","+num2);

}else{

alert(num3+","+num2+","+num1);

}

}

</script>

**条件语句分支**

<script type="text/javascript">

/\*

switch语句 语法：

switch(条件表达式){

case 表达式:

语句....

break;

case 表达式:

语句....

break;

case 表达式:

语句....

break;

default:

语句....

break;

}

执行流程：

switch....case...语句

在执行时会一次将case后面表达式的值与switch表达式后面的值进行比较

如果比较结果为true则执行下面的语句 反正继续往下执行

如果都满足则执行default下的语句

\*/

var num = +prompt("请输入一个数1~4")

switch (num){

case 1:console.log("壹");

break;

case 2:console.log("贰");

break;

case 3:console.log("叁");

break;

case 4:console.log("肆");

break;

default:console.log("输入错误！");

break;

}

</script>

**26.条件分支语句练习1**

<script type="text/javascript">

/\*

从键盘上接受一个1~7的数 并且打印对应的星期 否则打印非法数字

\*/

var day = +prompt("请输入数字1~7");

switch (day){

case 1:alert("星期一");

break;

case 2:alert("星期二");

break;

case 3:alert("星期三");

break;

case 4:alert("星期四");

break;

case 5:alert("星期五");

break;

case 6:alert("星期六");

break;

case 7:alert("星期日");

break;

default:alert("非法数字！");

break;

}

</script>

**26.条件分支语句练习2**

<script type="text/javascript">

/\*

对于成绩大于60的输出合格，小于60的输出不合格

\*/

var score1 = 90;

switch (parseInt(score1/10)){

case 10:

case 9:

case 8:

case 7:

case 6:

console.log("合格！");

break;

default:

console.log("不合格！");

break;

}

var score2 = 100;

switch (true){

case score2 >= 60:console.log("合格！");

break;

default:console.log("不合格！");

break;

}

</script>

**27.循环语句**

<script type="text/javascript">

/\*

向页面里连续输出数字

\*/

document.write("I LOVE YOU !");

/\*

循环语句：

通过循环语句可以反复的执行一段代码多次

while循环

-语法

while(条件表达式){

语句...

}

while语句执行时，首先对表达式进行求值判断。

如果值为true，则执行求值判断执行完毕后继续对表达式进行判断

如果任为true 则继续执行循环操作，以此类推

如果值为false，则终止循环

\*/

//创建一个循环,问问需要上步骤

//1.创建初始化变量

var i = 10;

//2.在循环中设置一个表达式

while(i > 0){

//3.定义一个更新表达式,每次更新初始化变量

alert("倒数十个数" + i--);

}

var h = 1;

while (h <= 5000){

document.write(h++ +"<br />")

}

</script>

**27.循环语句练习1**

<script type="text/javascript">

/\*

假如投资的年利率为5%。试求从1000块增长到5000块，需要多少年

\*/

var money = 1000;

var year = 0;

while (money <= 5000){

money = money + 0.05\*money;

year++;

}

console.log("需要" + year + "年");

</script>

**27.循环语句练习2**

<script type="text/javascript">

/\*

从键盘输入小明的成绩：

当成绩为100时奖励一辆宝马

当成绩为[80~99]时奖励一台iphonex

当成绩为[60~80]时奖励一本参考书

其他没奖励

\*/

while (true){

var score = prompt("请输入小明的成绩！");

if(score >=0 && score <= 100 ){

break;

}else{

alert("请输入有效值！");

}

}

if(score == 100){

alert("奖励一辆宝马！");

}else if(score >= 80){

alert("奖励一台iPhoneX");

}else if(score >= 60){

alert("奖励一本参考书！");

}else{

alert("洗干净准备挨揍!");

}

</script>

**28.for循环**

<script type="text/javascript">

/\*

for语句，也是一个循环，也称for循环

在for循环中，为我们提供了专门的上位置表达式

1.初始化表达式

2.条件表达式

3.更新表达式

for循环的语法

for(初始化表达式;条件表达式;更新表达式){

语句....

}

for循环的流程

1执行初始化表达式，初始化变量

2执行条件表达式，判断是否执行循环 true执行 false不执行

3执行更新表达式，更新表达式执行完毕继续重复

\*/

for(var i = 0;i < 10;i++){

alert(i);

}

</script>

**28.for循环练习1**

<script type="text/javascript">

/\*

打印1~100之间所有的奇数之和

\*/

var sum = 0;

for(var i = 1;i <= 100;i++){

if(i % 2 == 1){

var sum = sum + i;

}

}

console.log("1~100之间所有的奇数之和为：" + sum);

</script>

**28.for循环练习2**

<script type="text/javascript">

/\*

打印1~100之间所有7的倍数的个数及总和

\*/

var sum = 0;

for(var i = 1;i <= 100;i++){

if(i % 7 == 0){

sum = sum + i;

}

}

console.log("1~100之间所有7的倍数的个数及总和为:" + sum);

</script>

**28.for循环练习3**

<script type="text/javascript">

/\*

水仙花数

水仙花数是指一个三位数，它的每一位上的数字的3次幂之和等于它本身

（例如：13 + 53 + 33 = 153），请打印所有的水仙花数

\*/

var a,b,c;

for(var i = 100;i < 1000;i++){

a = parseInt(i / 100);

b = parseInt(i % 100 / 10);

c = i % 10;

if(a\*a\*a + b\*b\*b + c\*c\*c == i){

console.log("水仙花数为："+i);

}

}

//找出第二位数

// var b,i=457;

// b = parseInt(i % 100 / 10);

// console.log(b);

</script>

**28.for循环练习4**

<script type="text/javascript">

/\*

接收一个数判断是不是质数

只能被1或者本身整除的数为质数

\*/

var num = 14;

var flage = true;

for(var i=2 ; i<num ; i++){

if(num % i == 0){

//则num不是质数

flage = false;

}

}

if(flage){

console.log(num+"是质数");

}else{

console.log(num + "不是质数")

}

</script>

**29.嵌套的for循环**

<script type="text/javascript">

/\*

通过程序，在页面输出如下图形

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*/

for(var i=1 ; i<7 ; i++){

for(var j=0 ; j<i ;j++){

document.write("\*&nbsp&nbsp");

}

document.write("<br />");

}

document.write("<br />");

document.write("<br />");

document.write("<br />");

//方法一

for(var a=0 ; a<5 ; a++){

for(var b=a ; b<5 ; b++){

document.write("\*&nbsp&nbsp&nbsp")

}

document.write("<br />");

}

document.write("<br />");

document.write("<br />");

document.write("<br />");

//方法二

for(var a=0 ; a<5 ; a++){

for(var b=0 ; b<5-a ; b++){

document.write("\*&nbsp&nbsp&nbsp")

}

document.write("<br />");

}

</script>

**29.嵌套的for循环练习1**

<script type="text/javascript">

/\*

打印一个九九乘法表

1\*1=1

2\*1=2 2\*2=4

3\*1=3 3\*2=6 3\*3=9

4\*1=4 4\*2=8 4\*3=12 4\*4=16

.............................

\*/

for(var i=1 ; i<=9 ;i++){

for(var j=1 ; j<=i ; j++){

document.write(i+"\*"+j+"="+i\*j+"&nbsp&nbsp&nbsp&nbsp&nbsp");

}

document.write("<br /><br />");

}

</script>

**29.嵌套的for循环练习2**

<script type="text/javascript">

/\*

打印1~100之间的所有质数

\*/

for(var i=2 ; i<=100 ; i++){

var flage = true;

for(var j=2 ; j<i ;j++){

if(i % j == 0){

flage = false;

}

}

if(flage){

document.write(i+"是质数<br />");

}

}

</script>

**30.break和continue**

<script type="text/javascript">

/\*

break关键字可以用来退出switch语句

不能在if中使用break

break关键字会立即终止离他最近的循环语句

continue关键字可以跳过当次循环

同样continue只会对最近的循环起作用

\*/

/\*

可以给循环语句创建一个label，来标识当前的循环

label:循环语句

使用break语句时，可以在break后面跟着一个label

这样break将会结束指定的循环，而不是最近的

\*/

outer:

for(var i=0 ; i<6 ; i++){

console.log("我是外层循环！");

for(var j=0 ; j<5 ; j++){

break outer;

console.log("我是内层循环!");

}

}

for(var i=0 ; i<10 ; i++){

if(i == 5){

document.write("hello&nbsp");

continue;

}

if(i == 7){

document.write("word!!");

}

document.write(i+"&nbsp");

}

</script>

**31.计时器**

<script type="text/javascript">

/\*

开启一个计时器

语法 ；

--console.time("计时器的名字");

它需要一个字符串来作为参数，这个字符串将作为计时器的标识

--console.timeEnd("计时器的名字");

它用来停止一个计时器，也需要字符串作为计时器的标识

\*/

/\*

可以通过Math.sqrt()对一个数进行开放

\*/

var result = Math.sqrt(4);

console.log(result); //结果为2

</script>

**31.计时器**

<script type="text/javascript">

/\*

JS中数据类型

-String 字符串

-Unmber 数值

-Boolean 布尔值

-Null 空值

-Underfind 未定义

-以上五种类型属于基本数据类型，以后我们看到的值

只要不是上面五种就是对象

-object 对象

基本数据类型都是一个值"hello" 123 true，值与值之间没有任何联系

例如：一个人的身高 体重 性别 姓名 其数据类型分别为number型 number型 string型 string型

因为其不能成为一个整体 所以若要用其来表示一个人 显然行不通

所以这里我们用到对象来表示

对象属于一种复合的数据类型，在对象中可以保存多个不同的数据类型的属性

对象的分类

1.内建对象

-有js标准中定义的对象，在任何的js的实体中都可以使用

-比如：Math Syringe Number Boolean Function Object.....

2.宿主对象

-由js的运行环境提供的对象，目前来讲主要指由浏览器提供的对象

-比如：BOM DOM （document、console......）

3.自定义的对象

-由开发人员之间定义的对象

\*/

//创建对象

/\*

使用new关键词调用的函数是构造函数constructor

构造函数是专门用来创建对象的函数

\*/

var obj = new Object();

//console.log();

/\*

在对象中保存的值成为属性

向对象添加属性

语法：对象.属性名 = 属性值;

\*/

//向obj中添加一个name属性

obj.name = "孙悟空";

//向obj中添加一个gender属性

obj.gender = "男";

//向obj中添加age属性

obj,age = 18;

/\*

读取对象中的属性

语法： 对象.属性名

如果读取对象中没有的属性，不会报错而是会返回Underfind

\*/

//console.log(obj.gender); //男

//console.log(obj.hello); //显示Underfind

/\*

修改对象的属性值

语法： 对象.属性名 = 新值

就是覆盖

\*/

obj.name = "tom";

/\*

删除对象属性

语法； delete 对象.属性值;

\*/

delete obj.name;

console.log(obj.name); //结果为Underfind

</script>

**32.对象简介**

<script type="text/javascript">

/\*

JS中数据类型

-String 字符串

-Unmber 数值

-Boolean 布尔值

-Null 空值

-Underfind 未定义

-以上五种类型属于基本数据类型，以后我们看到的值

只要不是上面五种就是对象

-object 对象

基本数据类型都是一个值"hello" 123 true，值与值之间没有任何联系

例如：一个人的身高 体重 性别 姓名 其数据类型分别为number型 number型 string型 string型

因为其不能成为一个整体 所以若要用其来表示一个人 显然行不通

所以这里我们用到对象来表示

对象属于一种复合的数据类型，在对象中可以保存多个不同的数据类型的属性

对象的分类

1.内建对象

-有js标准中定义的对象，在任何的js的实体中都可以使用

-比如：Math Syringe Number Boolean Function Object.....

2.宿主对象

-由js的运行环境提供的对象，目前来讲主要指由浏览器提供的对象

-比如：BOM DOM （document、console......）

3.自定义的对象

-由开发人员之间定义的对象

\*/

//创建对象

/\*

使用new关键词调用的函数是构造函数constructor

构造函数是专门用来创建对象的函数

\*/

var obj = new Object();

//console.log();

/\*

在对象中保存的值成为属性

向对象添加属性

语法：对象.属性名 = 属性值;

\*/

//向obj中添加一个name属性

obj.name = "孙悟空";

//向obj中添加一个gender属性

obj.gender = "男";

//向obj中添加age属性

obj,age = 18;

/\*

读取对象中的属性

语法： 对象.属性名

如果读取对象中没有的属性，不会报错而是会返回Underfind

\*/

//console.log(obj.gender); //男

//console.log(obj.hello); //显示Underfind

/\*

修改对象的属性值

语法： 对象.属性名 = 新值

就是覆盖

\*/

obj.name = "tom";

/\*

删除对象属性

语法； delete 对象.属性值;

\*/

delete obj.name;

console.log(obj.name); //结果为Underfind

</script>

**33.属性值和属性名**

<script type="text/javascript">

/\*

对象的属性名可以是任何值

什么乱七八糟的东西都可以是属性名

\*/

// var obj = new Object();

// obj.var = "是是是";

// console.log(obj); //var: "是是是"

/\*

如果要使用特殊的属性名，不能采用一般的方式来操作

需要使用另一种方式

语法：对象["属性名"] = 属性值

读取时也需要采用这种方式

在[]中操作属性，更加灵活

在[]中可以直接传递一个变量，这样变量值是多少就会读取那个属性

\*/

var obj = new Object();

// obj["123"] = 789;

// obj["nihao"] = "你好";

// var n = "nihao"

//

// console.log(obj[n]); //就等于console.log(obj["nihao"]);

/\*

属性值

JS对象的属性值可以是任何数据类型

甚至也可以是一个对象

\*/

//创建一个新的对象

var obj2 = new Object();

obj2.name = "猪八戒！";

//将obj2设置为obj的属性

obj.test = obj2;

console.log(obj.test); // 结果为： {name: "猪八戒！"}

console.log(obj.test.name); // 结果为: 猪八戒!

/\*

in 运算符

-通过该运算符可以坚持一个对象中是否含有一个指定的属性

如果有则返回true，没有则返回false

-语法： "属性名" in 对象

\*/

console.log("test2" in obj); //结果为false (因为obj中没有test2这个属性名)

console.log("test" in obj); //结果为true

console.log("name" in obj); //结果为true

</script>

**34.基本数据类型和引用数据类型**

<script type="text/javascript">

/\*

基本数据类型

String Number Boolean Null Underfind

引用数据类型

Object

JS中的变量都是保存到栈内层中的

基本数据类型的值直接在栈内存中储存

值与值之间是独立存在的，修改一个变量不会影响其他的变量

对象是保存到堆内存中的，每创建一个新的对象就在堆内存中开辟一个内存空间

而变量保存的是对象的内存地址(对象的引用)，如果两个变量保存的是同一个对象引用

当通过一个变量修改时，另一个也受到影响

总结： 基本数据类型保存的是值

引用数据类型(对象)保存的是内存地址

\*/

</script>

**35.对象字面量**

<script type="text/javascript">

/\*

使用对象字面量，可以在创建对象时，直接指定对象属性

语法：{属性名:属性值,属性名:属性值.....}

对象字面量的属性名可以加引号建议不加

但是用特殊的名字是则必须要加引号

属性名和属性值是一组一组的名值结构

名和值之间使用:连接 多个值对之间用 , 隔开

，没有属性之后就不要用 ,

\*/

var obj = {

name:"猪八戒";

age:"28";

gender:"男";

};

</script>

**37.函数的简介**

<script type="text/javascript">

/\*

函数

-函数也是对象

-函数中可以封装一些功能(代码)，在需要时可以执行

\*/

var fun = new Function("console.log('这是以字符串方式创建的函数')"); //创建一个函数

//封装好的代码不会立即执行 将会在函数调用的时候执行

//调用函数语法: 函数对象();

//当调用函数时,函数中封装的代码会按顺序执行

fun();

fun(); //任意调用几次

/\*

使用函数声明来创建一个函数

语法：function 函数名([形参1，形参2，形参3......]){

语句.....

}

\*/

function fun2(){

console.log("这是以函数声明方式创建的函数");

}

//调用函数

fun2();

/\*

使用函数表达式创建一个函数

var 函数 = function([形参1，形参2.形参3........形参N]){

语句.....

}

\*/

var fun3 = function(){

console.log("这是以匿名函数赋值给变量");

}

fun3();

</script>

**38.函数的参数**

<script type="text/javascript">

/\*

定义一个用来求两个数和的函数

\*/

//声明函数时给函数添加两个形参 就相当于在函数内部 var a,b;

function sum(a,b){

console.log(a + b);

}

//调用函数时指定两个实参

sum(4,5);

/\*

调用函数时不会检查实参的类型

所以要注意是否可能会接受非法的参数

函数的实参可以是任意数据类型

\*/

sum(123,"hello"); //结果为 123hello

/\*

调用函数时，解析器也不会检查实参的数量

多余的参数不会被赋值

如果实参的数量少于形参的数量，则没有对应实参的形参是Underfind

\*/

sum(123,456,"hello",true,null); //结果为579

</script>

**39.函数的返回值**

<script type="text/javascript">

/\*

创建一个函数，用来计算三个数的和

可以使用return来设置函数的返回值

语法： return 值;

return后面的值会作为函数的执行结果返回

可以定义一个变量，来接受结果

return 后的语句就不会执行了

如果return语句后不加值就会返回一个Underfind

如果函数不写return也会返回一个Underfind

\*/

function sum(a,b,c){

//alert(a + b + c);

var d = a + b + c;

return d;

}

//调用函数

//sum(4,5,9);

var result = sum(4,5,7);

console.log(result);

</script>

**40.函数的实参可以是任何值**

<script type="text/javascript">

/\*

创建一个函数可以在控制台中数 出一个人的信息

可以输出一个人的nameage gender address

实参可以是任何数据类型 也可以是一个对象

当我们的参数过多时，可以将参数封装到一个对象中然后用过对象传递

\*/

function sayhello(o){

console.log("我是"+o.name+",我今年"+o.age+"岁了，"+"我是一个"+o.gender+"人,我住在"+o.address+"。");

}

var obj = {

name:"孙悟空",

age:18,

address:"花果山",

gender:"男"

}

sayhello(obj);

/\*

实参也可以是一个对象，也可以是一个函数

\*/

//以声明的方式创建一个函数 设一个形参a

function fun(a){

a(obj); //当sayhello以实参形式传给fun()时 a(obj) 就变为了 sayhello(obj)

}

fun(sayhello); //调用函数fun()并传函数sayhello给fun()

//得到的结果为:我是孙悟空,我今年18岁了，我是一个男人,我住在花果山。

/\*

fun(sayhello());

-调用函数

-相当于使用的是函数的返回值

fun(sayhello);

-函数对象

-相当于直接使用函数对象

\*/

</script>

1. **返回值的类型**

<script type="text/javascript">

/\*

\*/

// function fun(){

// alert("函数要执行了")

// for(var i=0 ; i<5 ; i++){

// if(i == 2){

// //使用break可以退出循环

// //break;

//

// //continue用于跳过当次循环

// //continue;

//

// //return可以结束整个函数 return所在的函数

// //return;

// }

// console.log(i);

// }

// alert("函数执行完毕！");

// }

// fun();

/\*

返回值可以是任意的数据类型

也可以是一个对象

\*/

// function fun2(){

// //返回一个对象

// return {name:"沙和尚"};

// }

// var a = fun2();

// console.log("a = "+a); //输出a是一个对象

// alert(a.name); //弹出沙和尚

function fun3(){

function fun4(){

alert("我是fun4");

}

//将fun4函数对象作为返回值返回

return fun4; //不加 () 表示返回的是 函数

//加上 () 表示返回的是 函数返回值

}

a = fun3();

console.log(a);

a();

fun3()();

</script>

**42.立即执行函数**

<script type="text/javascript">

/\*

立即执行函数

函数定义完，立即执行

立即执行函数只会执行一次

\*/

(function(){

alert("我是一个匿名函数~~~");

})();

(function(a,b){

alert(a + b);

})(4,5);

</script>

**43.方法**

<script type="text/javascript">

/\*

创建一个对象

\*/

var obj = new Object();

//向对象中添加属性

obj.name = "孙悟空";

obj.age = 18;

//对象的属性值可以是任何的数据类型,也可以是个函数

obj.sayName = function(){

console.log(obj.name);

};

function fun(){

console.log(obj.name);

};

//调方法

obj.sayName();

//调函数

fun();

/\*

函数也可以成为对象的属性

如果一个函数作为一个对象的属性保存

那么我们称这个函数是这个对象的方法

调用这个函数说调用对象的方法(method)

但是他是名称上的区别没有其他区别

\*/

var obj2 = {

name:"猪八戒",

age:18,

sayName:function () {

console.log(obj.name);

}

};

obj2.sayName();

</script>

**44.枚举对象中的属性**

<script type="text/javascript">

var obj = {

name:"孙悟空",

age:18,

gender:"男",

adress:"花果山"

};

/\*

枚举对象的属性

使用for...in 语句

语法：for(var 变量 in 对象){

}

for...in 语句 对象中有几个属性循环体就会执行几次

每次执行时，会将对象中的一个属性名字赋值给变量

\*/

for(var n in obj){

console.log("属性名："+n);

console.log("属性值："+obj[n]); //[] 用来放变量

}

</script>

**45.全局作用域(scope)**

<script type="text/javascript">

/\*

作用域

-作用域指定是一个变量的作用范围

-JS中有两种作用域

1.全局作用域

-直接编写在script标签中的js代码都在全局作用域

-全局作用域在页面打开时创建，关闭时销毁

-在全局作用域中有个全局对象window。

它代表的是浏览器的窗口，由浏览器创建文明可以直接使用

-在全局作用域中

创建的变量都会作为window对象的属性保存

创建的函数都会作为window对象的方法保存

-全局作用域中的变量都是全局变量

在页面的任意部分都可以访问的到

2.函数作用域

\*/

var a = 10;

var b = 20;

console.log(a); //在全局作用域中创建的变量都是保存在window对象中的

console.log(window.b); //使用变量就是调用window对象的方法

</script>

**46.变量的声明与提前**

<script type="text/javascript">

/\*

变量声明提前

-使用var关键字声明变量，会在所有的代码执行之前被声明(但不会赋值)

但是如果申明变量是不使用var关键字，则变了不会被声明提前

函数申明提前

-使用函数声明形式创建的函数 function 函数名(){}

它会在所有的代码执行之前被创建，所以可以在函数声明钱啊调用函数

-使用函数表达式创建的函数，不会被声明提前，所以不能在声明前调用

\*/

fun(); //因为使用的是函数声明形式创建的函数 所以会被提前创建 故而可以执行fun()函数

fun2(); //因为使用的是函数表达式形式创建的函数,所以不会被提前声明 故而函数不能执行

//使用函数声明形式创建函数

function fun(){

console.log("我是fun()函数");

}

//使用函数表达式创建函数

var fun2 = function(){

console.log("我是fun2()函数");

}

//在函数中也有声明提前

// 当在函数中用var申明一个变量时会在所有代码执行前创建这个对象 但是不会赋值

function fun3(){

console.log(a); //结果为Underfind

var a = 10;

}

fun3();

</script>

**47.函数作用域**

<script type="text/javascript">

/\*

函数作用域

-调用函数时创建函数作用域，函数执行完毕函数作用域销毁

-每调用一次函数就会创建一个新的函数作用域，他们之间是相互独立的

-在函数作用域中可以访问到全局作用域的变量

在全局作用域中访问不了函数作用域的变量

\*/

var a = 10,c = 5;

function fun(){

var a = 45;

var b = 87;

console.log(c); //结果为5 在函数作用域中可以访问到全局作用域的数

console.log(a); //结果为45 就近原则 先找函数内的变量 没有时再去全局作用域中找

console.log(window.a); //结果为10 加上widow就可以访问到全局作用域中a的值

}

fun();

//console.log(b); //报错 因为在全局作用域中访问不了函数作用域中的变量

console.log(a);

/\*

在函数中定义的变量 如果不使用 var 都将被判定为全局变量

\*/

var e = 10 , f = 50;

function fun1(){

console.log(e); //结果为Underfind

var e = 40; //在函数作用域中使用var申明变量将会在所有代码执行前创建 但是不会赋值

console.log(f); //结果为50 在函数作用域中定义的变量 如果不使用 var 都将被判定为全局变量

f = 40; //该行只会被认为是给f重新赋值

}

fun1();

//形参就相当于在函数中声明了变量

</script>

**48.this**

<script type="text/javascript">

/\*

解析器在调用函数每次都会想函数内部传递一个隐含的参数

这个隐含的参数就this，this执行的是一个对象

这个对象我们称为函数执行的上下文

根据函数的调用方式不同，this会指向不同的对象

1.以函数的形式调用时，this永远是window

2.以方法的形式调用时，this就是调用方法的对象

\*/

var name = "全局的name";

function fun(){

console.log(this.name);

}

var obj = {

name:"孙悟空",

age:18,

sayName:fun

};

obj.sayName(); //以方法的形式调用时,this就是调用方法的对象

fun(); //以函数的形式调用时,this就是 window的对象

</script>

**49.工厂方式创建对象**

<script type="text/javascript">

/\*

传统的创建对象

\*/

// var obj1 = {

// name:"孙悟空",

// age:18,

// gender:"男",

// sayName:function(){

// alert(this.name);

// }

// };

/\*

使用工厂方法创建对象

\*/

function fun(name,age,gender){ //给函数传三个形参

var obj = new Object(); //定义的方式创建一个对象

obj.name = name; //给obj设置属性和属性值

obj.age = age;

obj.gender = gender;

obj.sayName = function(){ //给obj设置属性值为sayName并赋值为一个函数

console.log(this.name); //当用方法调用时 输出的就方法对应的对象

};

return obj;

}

var obj2 = fun("猪八戒",21,"男");

console.log(obj2); //结果为：{name: "猪八戒", age: 21, gender: "男", sayName: ƒ}

console.log(obj2.gender); //结果为男

console.log(obj2.sayName()); //结果为猪八戒

obj2.sayName(); //结果为猪八戒

</script>

**50.构造函数**

<script type="text/javascript">

/\*

创建一个构造函数，专门用来创建Person对象

构造函数就是一个普通函数，创建方式和普通函数没区别

不同的是构造函数习惯上首字母大写

构造函数和普通函数的区别就是调用方式不同

普通函数直接调用，构造函数需要使用new关键字来调用

构造函数的执行流程

1.立刻创建一个新的对象

2.将新创建的对象设置为函数中的this，在构造函数中可以使用this来引用新建的对象

3.逐行执行函数中的代码

4.将新建的对象作为返回值返回

使用同一个构造函数创建的对象，我们称为一类对象，也将一个构造函数称为一个类

我们将通过一个构造函数创建的对象称为该类的实例

this的情况

1.当以函数的形式调用时，this是window

2.当以方法的形式调用时，谁调用方法 this就是谁

3.当以构造函数的形式调用时，this就是新创建的那个对象

\*/

function Person(name , age , gender){ //创建一个构造函数 并给它传三个形参

this.name = name; //this代表新创建的对象

this.age = age; //给新创建的对象赋值

this.gender = gender; //

this.sayName = function(){ //给新创建的对象赋值一个函数

console.log(this.name); //该函数的意思是输出调用该方法的对象的name属性值

//但是!!! 因为每执行一次就会创建一个方法 使得其很占用内存

// 创建对象时会立即开辟一个空间给对象，当调用构造函数时就会自动将sayName方法存入堆内存中 非常耗费内存

};

}

var per1 = new Person("猪八戒" , 18 , "男"); //创建对象并调用构造函数且为其传三个实参

console.log(per1); //结果为:Person {name: "猪八戒", age: 18, gender: "男", sayName: ƒ}

console.log(per1.age); //结果为:18

per1.sayName(); //结果为:猪八戒

//构造函数的修改 使其性能更高

function Dog(name , age ){

this.name = name;

this.age = age;

this.sayage = fun; //通过将sayage方法放在全局作用域中

} // 这样当创建对象时存入的不是sayAge方法而是fun()的地址 节省很大的空间

//在全局作用域中添加一个fun函数 这样导致全局作用域的环境被污染 可能会被无意覆盖掉

//解决方法: 详见 51.原型对象

function fun(){

console.log(this.age);

}

var dog1 = new Dog("旺财" , 5);

console.log(dog1);

dog1.sayage();

var dog2 = new Dog("李福" , 4);

console.log(dog2);

dog2.sayage();

console.log(dog1.sayage == dog2.sayage);

</script>

**51.原型对象**

<script type="text/javascript">

/\*

原型prototype

我们所创建的每一个函数，解析器都会想函数中添加一个属性prototype

这个属性对应着一个对象。这个对象就是我们所谓的原型对象

如果函数作为普通函数调用prototype没有任何作用

当函数以构造函数的形式调用时，它所创建的对象都会有一个隐含的属性

指向该构造函数的原型对象，我们可以通过\_\_prototype\_\_来访问该属性

原型对象就相当于一个公共区域，所有同一个类的实例都可以访问到该原型对象

我们可以将对象中共有的内容，统一设置到原型对象中。

当我们访问对象的一个属性或方法时，它会先在对象自身中寻找，如果有则直接使用

如果没有则会去原型对象中寻找，如果找到则直接使用

以后我们创建构造函数时，可以将这些对象共有的属性和方法统一添加到构造函数的原型对象中

这样不用分别为每个对象添加，也不会影响到全局作用域，就可以使每个对象都具有这些属性和方法了

\*/

function Person(name , age , gender){ //定义一个构造函数 并传三个参数

this.name = name; //给构造函数添加属性并传参数值

this.age = age; //将形参的赋给属性

this.gender = gender;

}

Person.prototype.sayName = function(){ //在构造函数的原型对象prototype中添加sayName方法 并且将匿名函数赋给sayName

console.log(this.name); //在控制台输出调用函数的对象的name属性的属性值

}

var per1 = new Person("孙悟空" , 18 , "男"); //调用构造函数 并给函数传三个实参

console.log(per1); //在控制台输出对象per1

per1.sayName(); //结果为:孙悟空

</script>

**52.原型对象补充**

<script type="text/javascript">

/\*

创建一个构造函数

\*/

function Country(){

this.power = "strong";

}

//向Country原型对象中添加一个city属性并赋值为"东京"

Country.prototype.city = "东京";

var china = new Country();

console.log(china.city);

//使用in检查对象中是否有某个属性是,如果对象中没有但是原型中有,也会返回true

console.log("city" in china); //结果为:true

//可以使用对象的hasOwnProperty()方法来检查对象自身中是否有该属性

//使用该属性只有当对象自身中含有属性时才会返回true

console.log(china.hasOwnProperty("city")); //结果为:false

console.log(china.hasOwnProperty("power")); //结果为:true

//补充 hasOwnProperty()方法 是对象的方法 然而China中没有定义该方法

// 则该方法一定在china的原型对象Country中或者在Country的原型对象中

</script>

**垃圾回收**

<script type="text/javascript">

/\*

垃圾回收

-程序运行过程中也会产生垃圾垃圾堆积多了就会导致程序运行速度过慢

所以我们要一个垃圾回收机制来处理程序运行中产生的垃圾

-当一个对象没有任何的变量或者属性对她引用或者操作时

此时这种对象就是一个垃圾，这种对象过多会占用大量内存，导致程序变慢

-在JS中拥有自动的垃圾回收机制，会自动将这些垃圾对象从内存中销毁

我们不需要也不能进行垃圾回收操作

-我们要做的就是将不在使用的对象设置为null 然后由系统自动回收

\*/

</script>

**55.数值**

<script type="text/javascript">

/\*

内建对象

宿主对象

自定义对象

数值（Array）

-数组也是对象

-它和普通对象功能类似，也是用来储存一些值的

-不同的是普通对象是使用字符串作为属性名的

而数组是使用数字作为索引操作元素

-索引：从0开始的整数

-数组的存储功能比普通的对象要好，所以在开发中我们常常使用数组来储存一些数据

\*/

//创建一个数组

var arr = new Array();

console.log(typeof arr); //结果为:object

/\*

想数组中添加元素

语法：数组[索引] = 值

\*/

arr[0] = 10;

arr[1] = 33;

console.log(arr);

/\*

读取数组中的元素

语法：数组【索引】

如果读取不存在的索引，不会报错而是返回Underfind

\*/

console.log(arr[3]); //结果为:Underfind

/\*

获取数组的长度

可以使用length属性来获取数组的长度（元素的个数）

语法：数组.length

对于连续的数组，使用length可以获取到的数组的长度

对于非连续的数组，使用length会获取到数组的最大索引+1

\*/

console.log(arr.length); //结果为:2

arr[10] = 45;

console.log(arr.length); //结果为:11 因为给数组加了一个索引为10的元素

//如果总想在数组最后一位添加一个元素ze

// 语法: arr[arr.length] = 数值;

arr[arr.length] = 55;

arr[arr.length] = 25;

arr[arr.length] = 15;

arr[arr.length] = 34;

arr[arr.length] = 215;

arr[arr.length] = 214;

console.log(arr);

</script>

**56.数值字面量**

<script type="text/javascript">

//创建一个数值

//var arr = new Array();

//使用字面量来创建数组

//语法:

//var arr = [];

//使用字面量创建数组时,可以在创建时就指定数组中的元素

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

console.log(arr); 结果为: [1, 2, 3, 4, 5, 6]

//使用构造函数创建数值是,也可以同时添加元素,将要添加的元素作为构造函数的参数传递

//元素之间使用 , 隔开

var arr2 = new Array(10,20,30);

console.log(arr2); //结果为: [10, 20, 30]

//表示创建一个只有一个元素为10 的数组

arr3 = [10];

console.log(arr3); //结果为:[10]

//表示创建一个长度为10的数组

arr4 = new Array(10);

console.log(arr4); //结果为: [empty × 10]

//数组中的元素可以是任何数据类型

arr5 = ["hello",1,true,false,null];

console.log(arr5); //结果为: ["hello", 1, true, false, null]

//元素也可以是对象

arr6 = [{name:"孙悟空",age:87},{age: 18},{gender:"男"}];

console.log(arr6); //结果为: [{…}, {…}, {…}]

//元素也可以是函数

arr7 = [function(){alert("hello");},function (){alert(word);}];

console.log(arr7);

//arr7[0](); //弹出对话框:hello

//元素也以是数组 二维数组

</script>

**57.数组的四种方法**

<script type="text/javascript">

//创建一个数组

var arr = ["孙悟空","猪八戒","沙和尚"];

/\*

push()

-该方法可以向数组的末尾添加一个或多个元素，并且返回数组新的长度

-可以将要添加的元素作为方法的参数传递

这样这些元素就会自动添加到数组的末尾

\*/

var result = arr.push("唐僧","蜘蛛精","白骨精"); //因为push带有返回值 所以我们设置一个变量接受这个返回值

console.log(arr); //结果为： ["孙悟空", "猪八戒", "沙和尚", "唐僧", "蜘蛛精", "白骨精"]

console.log("result = "+result); //结果为6(其为数组的长度)

/\*

pop()

-该方法可以删除末尾一个元素，并且返回值是删除的元素

\*/

result = arr.pop(); //该方法带有返回值 所以这一个变量接收其返回值

console.log(arr); //结果为:["孙悟空", "猪八戒", "沙和尚", "唐僧", "蜘蛛精"]

console.log("result = "+result); //结果为:result= 白骨精

/\*

unshift()

-向数组的开头添加一个或者多个元素，并返回新的数组的长度

\*/

console.log(arr); //先查看元素景为: ["孙悟空", "猪八戒", "沙和尚", "唐僧", "蜘蛛精"]

arr.unshift("牛魔王","二郎神");

console.log(arr); //结果为:["牛魔王", "二郎神", "孙悟空", "猪八戒", "沙和尚", "唐僧", "蜘蛛精"]

/\*

shift()

-删除数组开头的元素，并返回删除的元素

\*/

result = arr.shift(); //shift带有返回值 使用设置一个变量接收其返回值

console.log(arr); //结果为:["二郎神", "孙悟空", "猪八戒", "沙和尚", "唐僧", "蜘蛛精"]

console.log("result = "+result); //结果为:result = 牛魔王

</script>

**58.遍历数组**

<script type="text/javascript">

//创建一个数组

var arr = ["孙悟空","猪八戒","沙和尚","唐僧"]

//所谓的遍历就是将数组中的元素都取出来

for(var i=0 ; i<arr.length ; i++){

console.log(arr[i]);

}

</script>

**58.遍历数组练习1**

<script type="text/javascript">

function Person(name,age){

this.name = name;

this.age = age;

}

Person.prototype.sayName = function(){console.log(this.name);};

var per = new Person("帅呆呆" , 18);

console.log(per);

per.sayName();

per1 = new Person("孙悟空", 18);

per2= new Person("红孩儿", 8);

per3 = new Person("猪八戒", 28);

per4 = new Person("沙和尚", 38);

per5 = new Person("唐僧", 48);

var perArr = [per1,per2,per3,per4,per5];

console.log(perArr);

/\*

创建一个函数可以将perArr中的满18岁的person提出来

然后封装到一个新的数组中返回

\*/

function getAdult (arr){

//创建一个新的数组

var newArr = [];

//遍历数组取出perArr中的元素

for(var i=0 ; i<arr.length ; i++){

if(arr[i].age >= 18){

console.log(arr[i].age);

newArr.push(arr[i]);

}

}

//返回数组

return newArr;

}

var result = getAdult(perArr);

console.log(result);

</script>

**59.forEach遍历**

<script type="text/javascript">

//创建一个数组

var arr = ["孙悟空","白骨精","沙和尚","猪八戒","唐僧"]

/\*

forEach() 方法需要一个函数作为参数

-像这种函数，由我们创建但不由我们调用，我们称问回调函数

-数组中有几个函数就会执行几次，每次执行浏览器会将遍历到的元素

以实参的形式传递进来，我们可以来定义形参，来了读取这些内容

-浏览器会在回调函数中传递三个参数 分别为

第一个参数，当前正在变量的元素

第二个参数，就是当前正在遍历的元素的索引

第三个参数，就是正在遍历的数组 obj

！！！！IE8及一下浏览器不支持

\*/

arr.forEach(function(a){ //创建一个回调函数作为forEach的参数,并传一个形参

console.log("a = "+a); //打印形参 读取浏览器传进来的实参内容

}); // 传进来的结果为arr中的元素 且分五次依次传入

</script>

**60.slice和splice**

<script type="text/javascript">

var arr = ["孙悟空","猪八戒","沙和尚","唐僧","白骨精"];

/\*

slice()

-可以用来从数组中提取指定元素

-该方法不会改变数组，而是将截取的元素封装到一个新的数组中返回

-参数：

1.截取开始的位置索引

2.截取结束的位置的索引

包含开始不包含结束 可以获取到索引开始位置的元素，获取不到索引结束位置的元素

第二个值不写，则会返回第一个索引之后的所有元素

-索引也可以是负值。如果传递负值，则从前往后索引

-1.倒数第一个

-2.倒数第二个

\*/

var result = arr.slice(0,2); //slice有返回值所以定义一个变量接收 0和2表示索引的开始和结束的位置

console.log(arr); //输出结果为: ["孙悟空", "猪八戒", "沙和尚", "唐僧", "白骨精"] 不改变原始值

console.log(result); //结果为:["孙悟空", "猪八戒"] 只截取了索引为0,1的元素

/\*

splice()

-可以用于删除数组中的指定元素

-使用splice()会影响到原数组。会将指定元素从原数组中删除

并将删除的元素作为返回值返回

-参数

第一个，表示开始位置的索引

第二个，表示删除的数量

第三个，表示添加的新元素 添加位置为删除元素的位置

\*/

var arr2 = ["孙悟空","猪八戒","沙和尚","唐僧","白骨精","唐僧"];

var str = arr.splice(1,3,"牛魔王","玉漱公主","红孩儿","蜘蛛精");

console.log(str); //结果为:["猪八戒", "沙和尚", "唐僧"]

console.log(arr2); //结果为: ["孙悟空", "猪八戒", "沙和尚", "唐僧", "白骨精", "唐僧"]

</script>

**60.slice和splice练习1**

<script type="text/javascript">

//创建一个数组

var arr = [1,2,3,4,4,4,2,1,3,5,8];

//去除数组中重复的数字

for(var i=0 ; i<arr.length ; i++){ //遍历数组中的元素

for(var j=i+1 ; j<arr.length ; j++){ //遍历上一个for循环之后的所有元素 为比较做准备

if(arr[i] == arr[j]){ //将第i个元素与第i+1个元素及其之后的所有元素作比较 看其是否相等

arr.splice(j,1); //如果相等则删除相等元素 1表示删除的个数

j--; //当删除一个元素时 后面的元素替代上来 然而 循环还在继续往下一个进行

} // 所以要在比一次当前位置 j++之后再j-- 让其再比一次当前删除的位置元素

}

}

console.log(arr); //输出arr 结果为[1, 2, 3, 4, 5, 8]

</script>

**61.数组剩余的方法**

<script type="text/javascript">

/\*、

concat() 可以两个或者多个数组，并返回新的数组

-该方法不会对原数组产生影响

\*/

var arr1 = ["孙猴子","孙悟空","孙行者"];

var arr2 = ["猪猪侠","猪悟能","猪八戒"];

var arr3 = ["沙悟净","沙和尚","沙僧"];

var result = arr1.concat(arr2,arr3,"hello"); //调用arr1的concat方法 将数组1数组2数组3合并也可以合并字符串

console.log(arr1); //输出结果为: ["孙猴子", "孙悟空", "孙行者"] concat方法不会影响数组本身的元素

console.log(arr2); //结果为: ["猪猪侠", "猪悟能", "猪八戒"]

console.log("result = "+result); //结果为: result = 孙猴子,孙悟空,孙行者,猪猪侠,猪悟能,猪八戒,沙悟净,沙和尚,沙僧,hello

/\*

join();

-将数组转换为字符串

-该方法不会对原数组产生影响而是将转换后的字符串作为返回值返回

-join在后面的参数可以指定一个字符串 作为连接字符串的连接符

不指定就是默认逗号连接 指定空串时不用任何连接符

\*/

arr = ["孙猴子","孙悟空","孙行者"]; //创建一个数组

result = arr.join("——"); //调用数组的join方法 并将其返回值赋给变量result

console.log(arr); //打印arr数组 结果为: ["孙猴子", "孙悟空", "孙行者"]

console.log(result); //结果为:孙猴子——孙悟空——孙行者

console.log(typeof result); //结果为:String

/\*

reverse()

-该方法用来反转数组（前面的去后面后面的去前面）

-该方法会修改原数组

\*/

arr.reverse(); //调用数组的reverse方法

console.log(arr); //输出结果为:["孙行者", "孙悟空", "孙猴子"]

/\*

sort()

-可以用来对数组中的元素进行排序

-也是会影响数组，默认会按照Unicode编码进行排序

-即使是纯数字使用sort()排序也会按照Unicode排序所以对数字排序会出现错误

但是我们可以自己指定排序规则

我们可以在sort()添加一个回调函数,来指定规则

回调函数中需要两个参数

浏览器将会分别使用数组中的元素作为实参去调用回调函数

使用 哪个元素不确定，但是肯定的是数组中第一个参数一定第二个参数前面

浏览器会根据回调函数的返回值来决定元素的顺序

如果返回一个大于0的值，则会交换位置

如果返回一个小于或等于0的值，则元素位置不变

\*/

arr = ["a","s","e","f","q","f"];

arr.sort();

console.log(arr); //返回结果为:["a", "e", "f", "f", "q", "s"]

//创建一个纯数字的数组

arr = [5,4,8,6,77,55,11,23,6,7,5];

arr.sort(function(a,b){ //调用数组的sort方法 并使用回调函数 设置两个形参ab

return a-b; //

});

console.log(arr);

</script>

**62.call和apply**

<script type="text/javascript">

/\*

call()和apply()

-这两个都是函数对象的方法，需要通过函数对象来调用

-当对函数调用call()和appil()时 都会调用函数执行

-在调用call()和apply()时 可以将一个对象指定为第一个参数

此时这个对象将会成为函数执行时的this

-call()方法可以将实参在对象后依次传递

-apply()方法需要将参数封装到数组中统一传递

this的情况：

1.以函数的形式调用，this永远是window

2.以方法的形式调用，this是调用方法的对象

3.以构造函数的形式调用，this就是新创建的对象

4.使用call和apply调用，this就是参数指定的那个对象

\*/

function fun(a,b){

console.log(this.name);

}

//fun.call(); //与fun()一样可以调用fun函数

var obj = {

name:"孙悟空",

age:18

}

var obj1 = {

name:"猪八戒",

age:87

}

fun.call(obj1); //参数传哪个对象 函数中的this就代表哪个对象

var obj2 = {

name:"沙和尚",

age:23,

sayName:function(){ //给sayName属性添加一个匿名函数作为属性值

console.log(this.name);

}

}

obj2.sayName(); //结果为:沙和尚

obj2.sayName.call(obj1); //结果为:猪八戒 注: call和apply后面的参数是谁其所调用的函数中的this就是谁

</script>

63.arguments

<script type="text/javascript">

/\*

在调用函数时，浏览器每次都会传递两个隐含的参数

1.函数的上下文对象this

2.封装实参的对象arguments

-arguments是一个类数组的对象，他可以通过索引来操作数据，也可以获取长度

-在调用函数时，我们所传递的实参都会在arguments中保存

-arguments.length可以用来获取实参的长度

-我们即使不定义形参，也可以通过arguments来使用实参

arguments[0] 表示第一个实参

arguments[1] 表示第一二个实参

-它里面有一个属性叫做callee

这个属性对应一个函数对象，就是当前正在执行的函数对象

\*/

function fun(a,b){

console.log(arguments instanceof Array); //判断是否为数组

console.log(Array.isArray(arguments)); //判断是否为数组

console.log(arguments[1]); //结果为true 即为调用函数时所传的第二个参数

console.log(arguments.length); //结果为2 所传参数的个数

console.log(arguments.callee); //指的是当前执行的函数

}

fun("hello",true)

</script>

**64.Date对象**

<script type="text/javascript">

/\*

Date对象

-在JS中使用Date对象表示时间

\*/

//创建一个date对象

//如果使用构造函数创建一个Date对象,则会封装为当前代码执行的时间

var d = new Date();

console.log(d);

//创建一个指定的时间对象

//需要在构造函数中传一个表示时间段字符串作为参数

//日期格式 月份/日/年 时:分:秒

var d2 = new Date("8/29/2019 15:35:54");

console.log(d2);

/\*

getDate()

-获取当前日期是几日

\*/

var date = d2.getDate();

console.log(date);

/\*

getDay()

-获取当前日期是星期几

-0表示周日 1表示周一.........

\*/

var day = d2.getDay();

console.log(day);

/\*

getMonth()

-获取当前日期是几月份

-返回一个0~11的值

-0表示一月 1表示二月 2表示三月 .... 11表示十二月

\*/

var month = d2.getMonth();

console.log(month); //结果为:7

/\*

getFullYear

-获取当前日期的年份

-返回一个数值代表当前日期的年份

\*/

var year = d2.getFullYear();

console.log(year); //结果为:2019

/\*

getHours()获取小时 getMinutes()获取分钟 getSeconds()获取秒 getMilliseconds()获取毫秒

\*/

/\*

getTime()

-获取当前时间的时间戳

-时间戳，指从格林威治标准时间的1970年1月1日，0时0分0秒

到当前日期所花费的毫秒数（1秒 = 1000毫秒）

-计算机底层在保存时间时使用都是时间戳

\*/

var times = d2.getTime();

console.log(times); //获取的是毫秒数

time = Date.now();

console.log(time);

</script>

**65.Math**

<script type="text/javascript">

/\*

Math

-Math和其他对象不同，它不是构造函数

它不属于一个工具类不用创建，它里面封装好多数学相关的属性和方法

-比如

Math.IP表示圆周率

\*/

console.log(Math.PI); //结果为: 3.141592653589793

/\*

abs()可以用来计算一个数的绝对值

\*/

console.log(Math.abs(-45)); //结果为: 45

/\*

Math.ceil()

-可以对一个数进行向上取整

Math.floor()

-可以对一个数进行向下取整，小数部分会被舍掉

Math.round()

-可以对一个数四舍五入取整

\*/

console.log(Math.ceil(1.1)); //结果为:2

console.log(Math.floor(1.99)); //结果为:1

console.log(Math.round(1.4)); //结果为:1

/\*

Math.random()

-可以生成一个0~1的随机数

-生成一个0~x的随机数

Math.round(Math.random()\*X)

-生成一个X~Y之间的随机数

Math.round(Math.random()\*(Y-X))+X

\*/

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

//console.log(Math.random()); //生成0~1之间的数

//console.log(Math.round(Math.random()\*10)); //生成0~10之间的整数

//console.log(Math.round(Math.random()\*20)); //生成0~20之间的整数

//console.log(Math.round(Math.random()\*9)+1); //生成1~10之间的整数

}

/\*

max() 可以获取多个数的最大值

min() 可以获取多个数的最小值

\*/

var max = Math.max(10,54,87,65,78);

var min = Math.min(10,54,87,65,78);

console.log(min); //也可以传变量

console.log(max);

/\*

Math.pow(x,y)

-返回x的y次幂

\*/

console.log(Math.pow(4,3)); //结果为: 64

/\*

Math.sqrt()

-用于对一个数进行开方运算

\*/

console.log(Math.sqrt(2)); //结果为:1.4142135623730951

</script>

**66.包装类**

<script type="text/javascript">

/\*

基本数据类型

string Number Boolean Null

引用类

Object

在JS中提供三个包装类通过三个包装类可以将基本数据类型的数据转换为对象

String()

-可以将基本数据类型字符串转换为String对象

Number()

-可以将基本数据类型字符串转换为Number对象

Boolean()

-可以将基本数据类型字符串转换为Boolean对象

但是注意的是我们在实际应用中不会使用基本数据类型对象

如果使用基本数据类型对象，在做一些比较时可能会带来不可预期的结果

\*/

var num = 3;

console.log(num); //结果为3

console.log(typeof num); //结果为number

var num1 = new Number(3); //将num1这个基本数据类型包装为对象

// 基本数据类型没有方法可以调用 虽然系统可能会暂时自动转换基本数据类型为对象 但是调用完就自动转回金数据类型 (基本数据类型调用对象的方法只是不会报错没有实际的意义)

// 但是当包装为对象时,就可以使用对象的方法

console.log(num1); //结果为3

console.log(typeof num1); //结果为object

</script>

**67.字符串的相关方法 未完待续..........**

<script type="text/javascript">

/\*

创建一个字符串

\*/

var str = "hello word";

/\*

在底 层字符串是以字符数组的形式保存的

["h","e","i","i",.......]

\*/

console.log(str.length); //结果为：10 代表有是个字符

console.log(str[1]); //结果为: e 代表第二个字符为e

/\*

charAt( )

可以获取确定每个位置的数

\*/

</script>