目录

作业讲评

```
<script type="text/javascript">
    // 封装一个函数求这个数组var arr=[10,4,20,15,6,9]中最小值 并且在函数中输出这个最小值

    function getMin(){
        var arr=[10,4,20,15,6,9];

        // 假设数组的第一个元素为最小值
        var minValue = arr[0];

        for(var i = 1;i<arr.length;i++){
            if( minValue > arr[i] ){ // 如果数组元素比我们假设的那个值还要小
                minValue = arr[i];// 让它成为新的最小值
            }
        }

        document.write( minValue );
    }

    getMin();
</script>
```

```
<script type="text/javascript">
    // 封装一个函数，从函数外传递3个数求其和，并结果返回

    function getSum(a,b,c){
        return a+b+c;
    }
    console.log( getSum(10,20,30) );
</script>
```

```
<script type="text/javascript">
    // 封装一个函数,从函数外传递一个9个元素数组  求这个数组中中元素为奇数的平均值,并把结果返回

    function jishu(arr){
        var sum    = 0;// 奇数和
        var count  = 0;// 奇数个数
        // 遍历数组
        for( var i=0;i<arr.length;i++){
            if( arr[i]%2 == 1 ){// 如果是奇数
                sum += arr[i];
                count++;
            }
        }
        // 奇数平均值= 奇数和/奇数个数
        return sum/count ;
    }

    var arr1 = [5,3,7,10,2,4,20];
    console.log( jishu( arr1 ) );

    var arr2 = [4,8,7,9,10,24];
    console.log( jishu( arr2 ) );
</script>
```

```
<script type="text/javascript">
    /*
    4、封装一个函数,能够翻转传入的任意数组,并把翻转以后的数组通过返回值返回
    比如:数组[1, 3, 4, 6, 9]翻转以后变成了[9, 6, 4, 3, 1]
    比如:数组['red', 'pink', 'blue']翻转以后变成了["blue", "pink", "red"]
    */

    function reverseArr( arr ){
        // 定义一个新的空数组
        var newArr = [];
        // 遍历数组
        for(var i=arr.length-1; i>=0; i--){
            // 添加数组元素进新数组中
            newArr[newArr.length] = arr[i];
        }
        return newArr ;
    }

    var arr1 = [1,3,4,6,9];
    console.log( reverseArr(arr1) );
    console.log("");

    var arr2 = ['red', 'pink', 'blue'];
    console.log( reverseArr(arr2) );

</script>
```

```
<script type="text/javascript">
 // 使用for循环输出如下各个图案
 document.write("<h2>A:</h2>");
 for(var i=1;i<=4;i++){
 document.write("****<br/>");
 }



 document.write("<h2>B:</h2>");
 for(var i=1;i<=4;i++){
 for(var j=1;j<=i;j++){
 document.write("*");
 }
 document.write("<br/>");
 }

 document.write("<h2>C:</h2>");
 for(var i=1;i<=4;i++){
 for(var j=1;j<=2*i-1;j++){
 document.write("*");
 }
 document.write("<br/>");
```

```
}

document.write("<h2>D:</h2>");
for(var i=1;i<=4;i++){
for(var k = 4-i ;k>=1 ;k--){
document.write("a");
}

for(var j=1;j<=2*i-1;j++){
document.write("*");
}
document.write("<br/>");
}

document.write("<h2>E:</h2>");
for(var i=1;i<=4;i++){
for(var k = 4-i ;k>=1 ;k--){
document.write("a");
}

for(var j=1;j<=2*i-1;j++){
// 每一行的第一个跟每一个的最后一个才输出*,每一行的其他输出空格
if(j == 1 || j == 2*i-1){
document.write("*");
}else{
document.write("b");
}
}
document.write("<br/>");
}

document.write("<h2>F:</h2>");
for(var i=1;i<=4;i++){
for(var k = 4-i ;k>=1 ;k--){
document.write("a");
}

for(var j=1;j<=2*i-1;j++){
if(j == 1 || j == 2*i-1 || i == 4){
document.write("*");
```

```
}else{
document.write("b");
}
}
document.write("<br/>");
}


document.write("<h2>G:</h2>");
for(var i=1;i<=4;i++){
for(var k = 4-i ;k>=1 ;k--){
document.write(" ");
}

for(var j=1;j<=2*i-1;j++){
// 每一行的第一个跟每一个的最后一个才输出*,每一行的其他输出空格
if(j == 1 || j == 2*i-1){
document.write("*");
}else{
document.write(" ");
}
}
document.write("<br/>");
}
for(var i=3;i>=1;i--){
for(var k = 4-i ;k>=1 ;k--){
document.write(" ");
}

for(var j=1;j<=2*i-1;j++){
// 每一行的第一个跟每一个的最后一个才输出*,每一行的其他输出空格
if(j == 1 || j == 2*i-1){
document.write("*");
}else{
document.write(" ");
}
 }
 document.write("<br/>");
 }
 </script>
```

```
for(var gongji = 1;gongji<=16;gongji++){
    for(var muji = 1;muji<=29; muji++){
        // 小鸡的数量
        var xiaoji = 100 - gongji - muji;
        // 一共花费100块
        if(gongji*5 + muji*3 + xiaoji/3 == 100){
            document.write("公鸡数量为:"+gongji+",母鸡数量为:"+muji+",小鸡数量为:"+xiaoji);
            document.write("<br/>");
        }
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 7、进阶题,也要做 封装计算一个数字的n次幂(n次方)的函数,m是任何数字(小数,负整数,正整数)
 // 比如:pow(5,-2)返回0.04 5*5 = 25 1/25 = 0.04
 // 比如:pow(5,4)返回625 5*5*5*5 = 25*25 = 625
  // 比如:pow(3,0)返回1 3^0 = 1
  // 比如:pow(1,5)返回1 1*1*1*1*1 = 1

  function pow(m,n){
  // m:底数
  if(m == 0 && n>0 ){
  return 0;
  }
  // n:幂

  // n幂要分情况
  if(n==0){// 幂为0
  return 1;
  }else if(n>0){// 幂为正整数
  var sum = 1;
  for(var i=1;i<=n;i++){
  sum = sum * m;
  }
  return sum;
```

```
}else if(n<=0){// 幂为负整数
// 比如:pow(5,-2)返回0.04 5*5 = 25 1/25 = 0.04

var sum = 1;

for(var i=1;i<= -1 * n;i++){
sum = sum * m;
}
return 1/sum;
}
}

console.log( pow(5,-2) );
console.log( pow(5,4) );
console.log( pow(3,0) );
console.log( pow(1,5) );
</script>
</head>
<body>

</body>
</html>
```

第8题:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 8、进阶题,也要做 封装在指定n范围找到质数的函数, n>=1, n是整数,返回指定范围里面所有的质数

 /*function getPrime(n){
 // n:范围
 }
 比如:
```

```
getPrime(100)返回[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

getPrime(200)返回[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]*/


// 求2~n范围之间的质数,比如n如果是100,就代表找2~100之间的所有质数

// 质数是指在大于1的自然数中，除了1和它本身以外不再有其他因数的自然数

// 因数是指整数a除以整数b(b≠0) 的商正好是整数而没有余数，我们就说b是a的因数。

// 对于某个数, 最小的因数是1 最大的因数是本身
// 4这个数的因数有哪些? 1 2 4
// 42这个数的因数有哪些? 1 2 3 6 7 14 21 42
// 3这个数的因数有哪些? 1 3
// 可以得到另一个规律 因数只有1跟自身,也就是因素个数为2的自然数就是质数

// 第一步:先使用代码输出对应数的因数
/*var num = 42;
for( var i=1; i<=num; i++){// 假设i是因数
if( num%i == 0 ){// 当i满足num%i==0,就代表i是因数了
console.log( i );
}
}*/

// 第二步:统计因数的个数
/*var num = 42;
var count = 0;// 因数的个数
for( var i=1; i<=num; i++){// 假设i是因数
if( num%i == 0 ){// 当i满足num%i==0,就代表i是因数了
// console.log( i );
count++;
}
}
console.log( count );*/

// 第三步:判断因数的个数是否为2
/*var num = 42;
```

```
var count = 0;// 因数的个数
for( var i=1; i<=num; i++){// 假设i是因数
if( num%i == 0 ){// 当i满足num%i==0,就代表i是因数了
// console.log( i );
count++;
}
}

if( count == 2){
console.log( num+"是质数" );
}else{
console.log( num+"不是质数" );
}*/

// 第四步:先求2~10之间的所有质数
/*for(var j=2;j<=10;j++){
var num = j;
var count = 0;// 因数的个数
for( var i=1; i <= num; i++){// 假设i是因数
if( num%i == 0 ){// 当i满足num%i==0,就代表i是因数了
// console.log( i );
count++;
}
}

if( count == 2){
console.log( num+"是质数" );
}
}*/

// 第五步,2~10之间的所有质数放在一个数组里面
/*var newArr = [];
for(var j=2;j<=10;j++){
var num = j;
var count = 0;// 因数的个数
for( var i=1; i <= num; i++){// 假设i是因数
if( num%i == 0 ){// 当i满足num%i==0,就代表i是因数了
count++;
}
}
```

```
91
92  if( count == 2){
93  // console.log( num+"是质数" );
94  // 把质数添加到新数组中
95  newArr[newArr.length] = num;
96  }
97  }
98  console.log( newArr ); */
99
100
101  // 第六步:改成函数,求2~n之间所有质数,并把所有质数放在数组中,返回这个数组
102  function getPrime(n){
103  // n:范围
104  var newArr = [];
105  for(var j=2;j<=n;j++){
106  var num = j;
107  var count = 0;// 因数的个数
108  for( var i=1; i <= num; i++){// 假设i是因数
109  if( num%i == 0 ){// 当i满足num%i==0,就代表i是因数了
110  count++;
111  }
112  }
113
114  if( count == 2){
115  // console.log( num+"是质数" );
116  // 把质数添加到新数组中
117  newArr[newArr.length] = num;
118  }
119  }
120  // console.log( newArr );
121  return newArr;
122  }
123
124  console.log( getPrime(100) );
125  console.log( getPrime(200) );
126  </script>
127 </head>
128 <body>
129
130 </body>
131 </html>
```

```
// 参数是可以设置默认值的
function findElement(arr,element,startIndex = 0){
    // arr:原数组
    // element:需要查找的元素
    // startIndex:开始查找的下标,如果不传,代表从0开始查找

    // 怎么查询数组中的所有元素是否等于某个值
    for(var i=startIndex;i<arr.length;i++){
        if( arr[i] == element ){// 判断数组中的元素的值是否等于element
            // console.log( arr[i] );
            return i ;// 返回数组下标
        }
    }

    // 如果整个数组都遍历完了,还是找不到相等的,那就代表数组中没有这个element,返回-1
    return -1;
}

// 比如: findElement(["a","b","c","d","e","c"],"c") 返回2
console.log(  findElement(["a","b","c","d","e","c"],"c") );

// 比如: findElement(["a","b","c","d","e","c"],"c",3) 返回5
console.log(  findElement(["a","b","c","d","e","c"],"c",3) );

console.log(  findElement(["a","b","c","d","e","c"],"g") );
```

学习目标

- 能够了解变量的作用域
- 能够区别值传递与引用传递
- 能够了解什么是对象
- 能够了解对象的分类有哪些
- 能够创建自定义对象
- 能够遍历对象的属性
- 能够了解并使用JSON数据格式
- 能够遍历JSON对象的属性
- 能够掌握string对象的属性和方法
- 能够理解Math对象的属性和方法

作用域

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。

JavaScript（es6前）中的作用域有两种：

全局作用域局部作用域（函数作用域）

全局作用域

作用于所有代码执行的环境(整个 script 标签内部)或者一个独立的 js 文件。

局部作用域

作用于函数内的代码环境，就是局部作用域。 因为跟函数有关系，所以也称为函数作用域。

javascript没有块级作用域(在ES6之前)

块作用域由 { } 包括。

在其他编程语言中（如 java、c#等），在 if 语句、循环语句中创建的变量，仅仅只能在本 if 语句、本循环语句中使用，如下面的Java代码：

java有块级作用域：

```
if(true){
  int num = 123;
  system.out.print(num);  // 123
}
system.out.print(num);    // 报错
```

以上java代码会报错，是因为代码中 { } 即一块作用域，其中声明的变量num，在 “{ }” 之外不能使用；

而与之类似的JavaScript代码，则不会报错：

javascript中没有块级作用域（在ES6之前）

```
if(true){
 var num = 123;
 console.log(num);// 123
}
console.log(num);// 123
```

```
6
7  // 在es6中,可以使用let定义块作用域的变量
8  if(true){
9   let num2 = 456;
10   console.log(num2);// 456
11  }
12  console.log(num2);// 报错,报num2 is not defined
```

变量的作用域

什么是变量的作用域?

指变量在什么地方可以使用，什么地方不可以使用。

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

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 声明的变量也是全局变量（不建议使用）

举例:

```
1  // 什么是变量的作用域？指变量在什么地方可以使用，什么地方不可以使用。
2   // 我们js中的变量的作用域是通过"函数"来进行划分的
3
4   /*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"）。
5   全局变量在代码的"任何位置"都可以使用
6   在全局作用域下 var 声明的变量 是全局变量
7   特殊情况下，在函数内不使用 var 声明的变量也是全局变量（不建议使用）*/
8
9   // 在函数外定义了一个变量a,那么这个a就是全局变量
10   var a = 10;
11   // 定义函数
12   function fn1(){
13   console.log( a );// 10
14
```

```
// 特殊情况下，在函数内不使用 var 声明的变量也是全局变量（不建议使用）
b = 20;// 注意:没有加var
console.log( b );// 20
}
// 调用函数
fn1();
console.log("----");
console.log( a );// 10
console.log( b );// 20
```

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

举例:

```
/*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*/

function fn2(num1,num2){
console.log("形参1的值为:"+num1);
console.log("形参2的值为:"+num2);

 // 在函数内部 var 声明的变量是局部变量
 var num3 = 30;
 console.log( num3 );
 }
 fn2(10,20);

 // 访问函数内形参的值
 // console.log( num1 );// 报错
 // console.log( num2 );// 报错

 // 输出在函数内使用var定义的变量
 // console.log( num3 );// 报错
```

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

如何在函数外面 访问函数里面定义的变量 重点,掌握

方法一:只需要将函数里面定义的变量的**var关键字给去掉**就可以实现将**局部变量的作用域提升至全局作用域。**

```
// 如何在函数外面 访问函数里面定义的变量 重点,掌握
 // 方法一:只需要将函数里面定义的变量的var关键字给去掉就可以实现将局部变量的作用域提升至全局作用域。(不推荐大家使用)

function fn1(){
// 此时函数里面有var定义的变量是局部变量,在函数外是使用不了的
// var a = 10;

// 去掉var关键字,就可以将局部作用域提升成全局作用域
a = 10;
 console.log( a );
 }
 // 调用函数,执行函数体代码
 fn1();

 console.log( a );
```

方法二:我们并不建议直接就**var 关键字给省略，我们建议在函数的外面定义一个同名的全局变量。**

```
// 方法二:我们并不建议直接就var 关键字给省略，我们建议在"函数的外面定义一个同名"的全局变量。

// 在函数外定义一个同名全局变量,定义变量,但是不赋值
var b;

function fn2(){
// 给全局变量b赋值
```

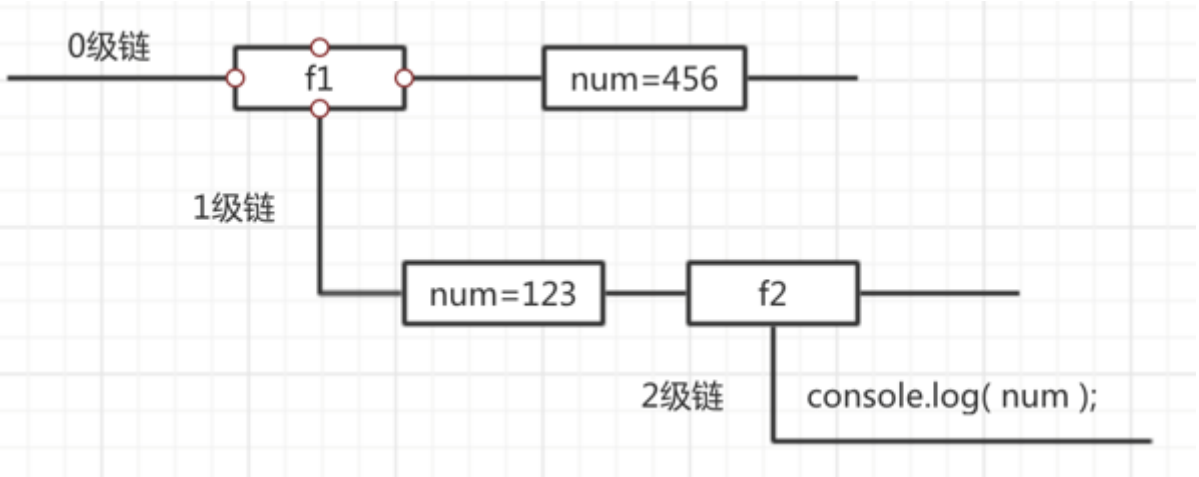
```
b = 20;
console.log( b );
 }
 // 调用函数,才会执行函数体中的代码
 fn2();
 console.log( b );
```

作用域链 难点 (变量的提升)

只要是代码都一个作用域中，写在函数内部的局部作用域，未写在任何函数内部即在全局作用域中；
如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域；
根据在**[内部函数可以访问外部函数变量]**的这种机制，用链式查找决定哪些数据能被内部函数访问，就称作作用域链

案例分析1：

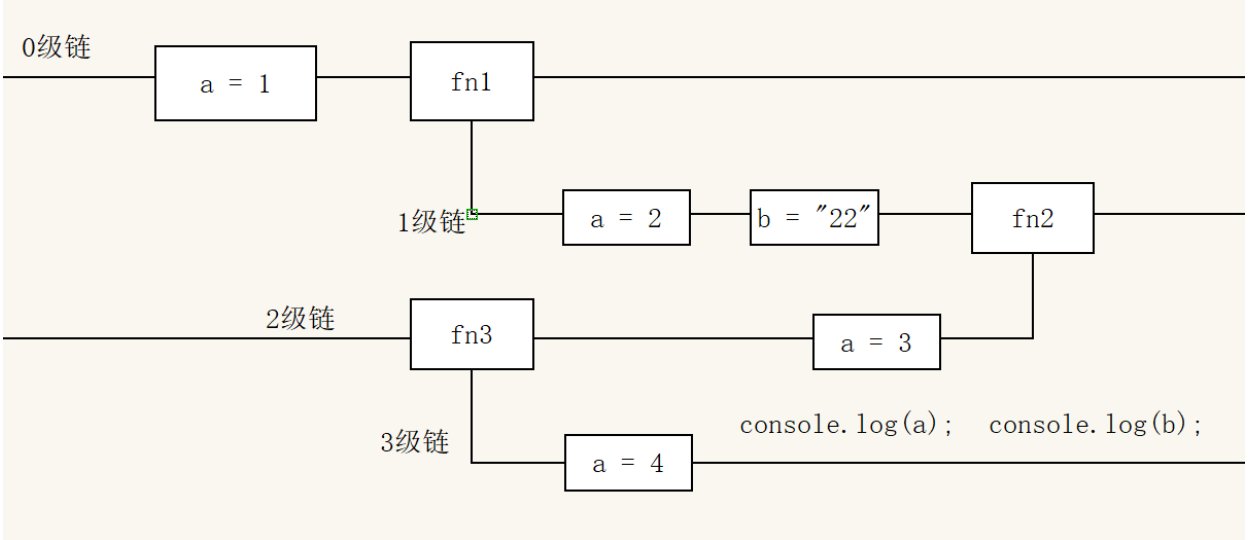
```
function f1() {
 var num = 123;
 function f2() {
 console.log( num );// 123
 }
 f2();
}

var num = 456;
f1();
console.log( num );// 456
```



作用域链：采取**就近原则**的方式来查找变量最终的值。

案例分析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 4
  console.log(b); //b的值 "22"
  }
  }

}

fn1();
```



预解析

预解析的相关概念

JavaScript 代码是由浏览器中的 JavaScript 解析器来执行的。JavaScript 解析器在运行 JavaScript 代码的时候分为两步：预解析和代码执行。

- 预解析：在当前作用域下, JS 代码执行之前，浏览器会默认把带有 var 和 function 声明的变量在内存中进行提前声明或者定义。
- 代码执行： 从上到下执行JS语句。

预解析会把变量和函数的声明在代码执行之前执行完成。

变量预解析

预解析也叫做变量、函数提升。

变量提升（变量预解析）： 变量的声明会被提升到当前作用域的最上面，变量的赋值不会提升。

```
1 console.log(num);  // 结果是多少?
2 var num = 10;
3 结果：undefined
4
5 注意：**变量提升只提升声明，不提升赋值**
```

```
// 变量预解析
// 预解析也叫做变量、函数提升。
// 变量提升（变量预解析）：
变量的声明会被提升到当前作用域的最上面，变量的赋值不会提升。

// 变量被提升了,所在变量赋值之前输出变量的值,不会报错,会输出undefind
console.log( num );// undefined
var num = 10;
console.log( num );// 10

console.log("");
```

函数预解析

函数提升： 函数的声明会被提升到当前作用域的最上面，但是不会调用函数。

```
1 fn();
2 function fn() {
3     console.log('打印');
4 }
```

结果：控制台打印字符串 ” 打印 “

```
// 函数预解析
// 函数提升： "函数的声明会被提升到当前作用域的最上面，但是不会调用函数"。
// 默认情况下,在函数定义之前调用函数是不会报错的,是可以正常调用的
// 不过我们还是推荐,先定义函数,再调用函数
// 注意：函数声明代表函数整体，所以函数提升后，函数名代表整个函数，但是函数并没有被调用！
fn();
function fn() {
    console.log("打印");
}
function fn(){
    console.log("复印")
}
fn();
```

注意：函数声明代表函数整体，所以函数提升后，函数名代表整个函数，但是函数并没有被调用！

函数表达式声明函数问题

函数表达式创建函数，会执行变量提升，此时接收函数的变量名无法正确的调用：

```
fn();
var  fn = function() {
    console.log('想不到吧');
}
```

结果：报错提示 " fn is not a function"

解释：该段代码执行之前，会做变量声明提升，fn在提升之后的值是undefined；而fn调用是在fn被赋值为函数体之前，此时fn的值是undefined，所以无法正确调用

```
// 函数表达式声明函数问题
// 函数表达式创建函数，会执行变量提升，此时接收函数的变量名无法正确的调用：

// fn2加括号调用匿名函数
console.log( fn2 );// undefined
// fn2();// 报错
// 匿名赋值给变量fn2
var fn2 = function() {
   console.log('想不到吧');
}
// 解释：该段代码执行之前，会做变量声明提升，fn2在提升之后的值是undefined;而fn2调用是在fn2被赋值为函数体之前，此时fn2的值是undefined，所以无法正确调用
```

值传递与引用传递

什么叫做传递呢?

就是变量与变量之间的赋值操作

js中的传递分为值传递与引用传递

- **基本数据类型之间的赋值操作都是值传递**
- **复合数据类型之间的赋值操作都是引用传递**

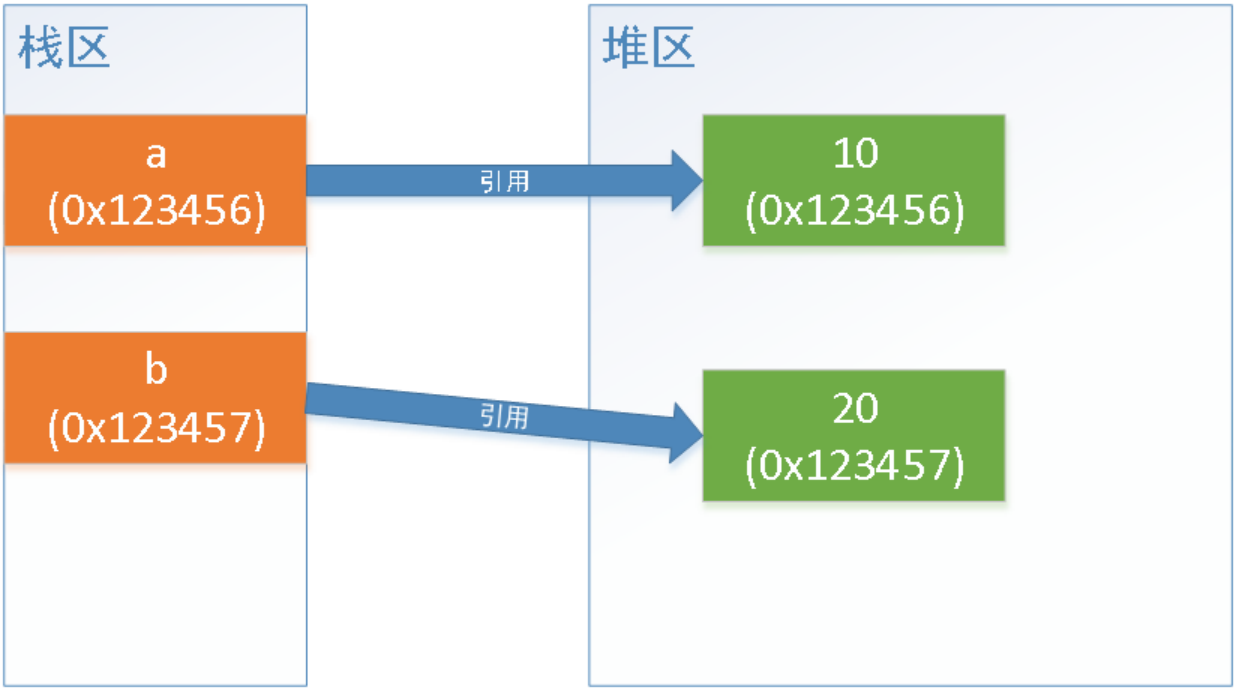
值传递

将一个变量的值赋值给另外一个变量，如果修改其中一个变量的值，不会影响到另外一个变量的值！这两个变量之间是没有任何的联系。

变量如何在内存中存储？内存它是分区域：**栈区和堆区**　　**栈区：**空间小，速度快　主要存储变量名以及值在堆区的地址　　**堆区：**空间大，速度慢　主要存储变量的值

变量名变量值之间的关系？　　**变量名引用变量值**，变量值被变量名所引用！

图解:



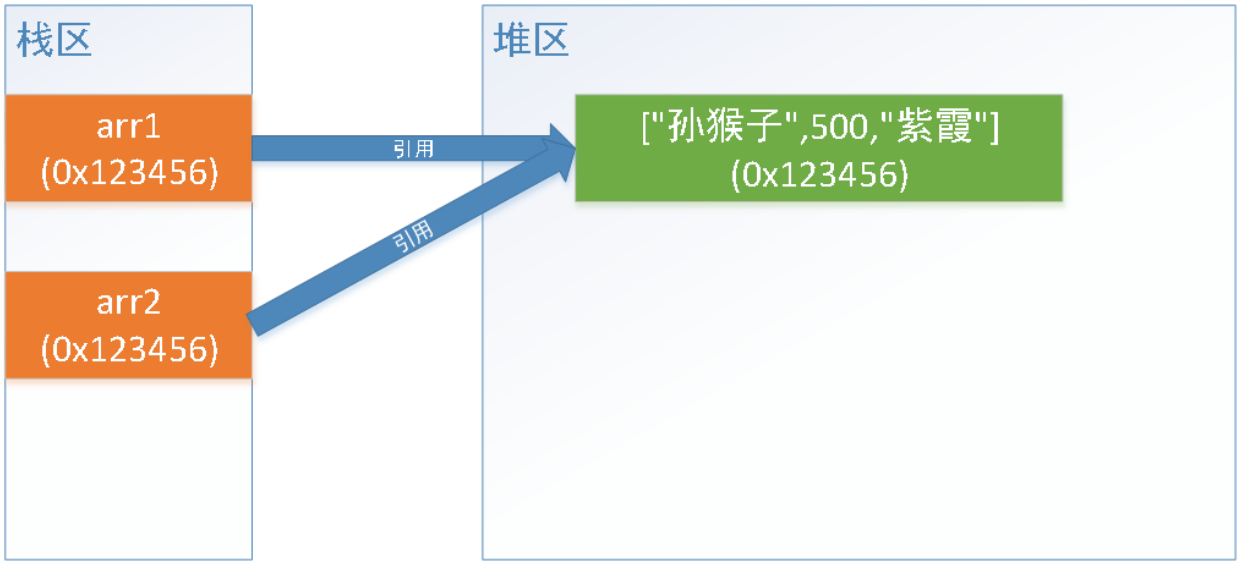
举例:

```
// 值传递与引用传递
// 什么叫做传递呢?就是"变量与变量之间"的"赋值"操作

// js中的传递分为值传递与引用传递
// "基本数据类型"之间的赋值操作都是"值传递"
// "复合数据类型"之间的赋值操作都是"引用传递"

// 值传递:将一个变量的值赋值给另外一个变量，如果修改其中一个变量的值，不会影响到另外一个变量的值！这两个变量之间是没有任何的联系。(互不相关)

var a = 10;
var b = a;// var b = 10
console.log( a,b );// 10 10
// 修改变量a的值
a = 100;
console.log( a,b );// 100 10
```

引用传递

将一个变量的值（内存地址）赋值给另外一个变量，如果修改其中一个变量的值，会影响到另外一个变量的值!这两个变量之间是有联系的。

图解:



举例:

```
// 引用传递:将一个变量的值（内存地址）赋值给另外一个变量，如果修改其中一个变量的值，会影响到另外一个变量的值!这两个变量之间是有联系的。(同生共死)

// 数组就是复合类型,所以数组之间的赋值操作是引用传递
var arr1 = [10,20,30];
var arr2 = arr1;
console.log( arr1, arr2 );// [10,20,30]  [10,20,30]
// 修改 数组中某个元素的值
arr2[0] = 100;
console.log( arr1, arr2 );// [100,20,30]  [100,20,30]
```

```
var arr1 = [10,20,30];
var arr2 = arr1;
console.log( arr1, arr2 );// [10,20,30] [10,20,30]
// 直接修改arr2变量的值
arr2 = 100;
console.log( arr1, arr2 );// [10,20,30]  100
```

对象(object)

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。 看得见摸得着实物。一本书、一辆汽车、一个人都可以是"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"。

- 泛指 **一大类** 比如 说 苹果 就是泛指 类似其他语言的 类(class)
- 特指 **某一个** 比如说 这个苹果 我们说对象就是特指的，某一个。

请说出下列哪个是 对象?

> 汽车 飞机 明星 周星驰(星爷) 女朋友 迪丽热巴 班主任
> 苹果 这个苹果 手机 老师现在用的这个手机 游戏 王者荣耀

这个手机有啥特性? --- **在对象里面 外在特性 我们称为(名词)** **对象的属性**

颜色是黑色 尺寸 3寸 形状 是 方形 重量 200g

这个手机有啥功能? -- **对象里面做 一些行为 (动词)** 称为 **对象的方法**

打电话 发短信 玩游戏

华仔有啥特性?

身高　体重　发型　肤色

华仔有啥行为?

演电影　唱歌　跳舞

为什么要有对象?

表达结构更清晰

我们保存一个值 我们用变量

我们保存多个值(一组值) 我们用 数组

但是我们如果要保存**一个人的完整信息**

刘德华信息如下:

> 英文名:Andy Lau
> 身高:174cm
> 出生地:香港新界大埔镇泰亨村
> 出生日期:1961年9月27日
> 体重: 63kg
> 生肖:牛
> 歌曲: 冰雨 忘情水 爱你一万年等
> 电影: 天下无贼 无间道等

如果你要用一个数组表示， 别人一般是看不懂的。

var arr = ["刘德华", "牛", "冰雨","天下无贼"];

因为信息非常的复杂了，就不要用变量去存储了，这里我们可以用 复杂数据类型object 来存储复杂数据。

应该这样表示:

```
刘德华.生日 = 1961;
刘德华.生肖 = 牛;
刘德华.唱歌();
刘德华.演戏();
```

这样来表达结构就很清楚了。

其中， 年龄 生肖 体重 身高这些 外在固定特性(一看就是名词) 我们称之为 **属性。**

唱歌、 演戏、 跳舞 这些 动作 (二看就是动词)我们称之为 **方法。**

属性和方法的区别是 **属性不加括号，方法必定加上括号**。**属性其实就是之前我们学的变量,方法就是我们之前学的函数**

可以大量复用

回忆:当业务功能很多，代码量大的时候，为了复用代码，我们用 **函数** 来封装代码

那如果 函数很多、变量很多呢？--- 我们可以使用 **对象** 来封装 函数 和 变量

- **用** 对象 **封装 相关的** 属性 **和** 方法

```
// 定义了一个Person类
function Person(username,age,sex){
//属性
this.username = username;
this.age = age;
this.sex = sex;

//方法
this.sayHello = function(){
 console.log("我的名字叫:"+this.username+",年龄:"+this.age+",性别:"+this.sex);
 };


 this.chifan = function(){
 console.log(this.username+"在吃饭");
```

```
}
}

// 通过new就可以从一个类中得到一个新的具体对象
var daBai = new Person("大白",23,'男');
daBai.sayHello();
daBai.chifan();

var xiaobai = new Person("小白",21,'男');
xiaobai.sayHello();
xiaobai.chifan();

var xiaoMei = new Person("小美",21,'女');
xiaoMei.sayHello();
xiaoMei.chifan();
```

对象的分类

自定义对象和内置对象

自定义对象由程序员自己创建的

内置对象一般就是js自带有的对象 ，比如 window对象

创建自定义对象

函数是封装了一组代码在里面

而对象是封装了一组变量(属性)和 函数(方法)

1)使用{}来创建自定义对象(重点 较为常用)

- 在花括号 {} 中的一个对象的0个或多个 键:值 无序列表
- **键:**相当于属性名
- **值:**相当于属性值，可以是任意类型的值(数值类型、字符串类型、布尔类型，甚至 函数类型)

格式:

```
var 变量名 = {
  属性名1:值1,
  属性名2:值2,
  方法名1:匿名函数1,
  方法名2:匿名函数2,
```

```
    ...
};

或者

var 变量名 = { };
变量名.属性名1 = 值1;
变量名.属性名2 = 值2;
变量名.方法名1 = 匿名函数1;
变量名.方法名2 = 匿名函数2;
...
```

对象调用:

- **对象里面的属性调用**： **对象.属性名**　　这个小点 就理解为　的
- 对象里面的属性另外调用方式： **对象['属性名']**　注意 方括号里面的属性 必须加 引号　我们后面会用
- 对象里面的方法调用: **对象.方法名()**　　注意这个方法名字后面一定加括号

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 创建自定义对象-使用{}创建自定义对象 推荐大家使用,简单容易理解

 /*
  语法一: 先定义一个空对象,再给这个对象设置属性和方法

  var 对象变量名 = {};
  // 给对象定义属性(特征),我们可以理解属性就是变量
  对象变量名.属性名1 = 属性值1;
  对象变量名.属性名2 = 属性值2;

  // 给对象定义方法(行为,功能,可以做什么),我们可以理解方法就是函数
```

```
对象变量名.方法名1 = function(){
// 具体方法1的代码
}

对象变量名.方法名2 = function(){
// 具体方法2的代码
}
*/

// 定义了一个猫对象
var cat = {};
// 猫的属性(猫的特性)
cat.color = "white";
cat.weight = "5kg";
// 猫的方法(猫的特性)
cat.skill = function(){
console.log("猫可以抓老鼠~");
}
cat.cry = function(){
console.log("喵~");
}

// 访问对象中的属性 语法: 对象名.属性名 或者 对象['属性名']
console.log( cat.color );
console.log( cat["weight"] );

// 访问对象中的方法 语法: 对象名.方法()
cat.skill();
cat.cry();


// 语法二: 也是使用{},定义的对象的时候,同时给这个对象设置属性和方法
// 有点像CSS的感觉
/*var 对象变量名 = {
属性名1:值1,
属性名2:值2,
方法名1:匿名函数1,
方法名2:匿名函数2,
...
};*/

```

```
59  console.log("");
60  console.log("");
61  console.log("");
62
63
64  // 定义一个女朋友对象
65  var girlFirend = {
66  // 属性
67  height:"170cm",
68  weight:"45kg",
69  // 方法
70  eating:function(){
71  console.log("女朋友可以陪你吃饭");
72  },
73  game:function(){
74  console.log("女朋友可以玩开心消消乐");
75  }
76  };
77
78  // 输出女朋友对象的属性
79  // 访问对象中的属性 语法: 对象名.属性名 或者 对象['属性名']
80  console.log("女朋友的身高为:"+girlFirend.height);
81  console.log("女朋友的体重为:"+girlFirend["weight"]);
82
83  // 调用女朋友对象的方法
84  // 访问对象中的方法 语法: 对象名.方法()
85  girlFirend.eating();
86  girlFirend.game();
87  </script>
88 </head>
89 <body>
90
91 </body>
92 </html>
```

函数和方法的区别:

- 函数是单独存在的， 函数调用是 **函数名()**
- 方法是在对象里面， 方法调用是 **对象.方法名()**

函数跟方法的使用基本是一致的。

练习: 请用{}形式创建 一个 王可可 的狗对象。 具体信息如下:

王可可 是一条 阿拉斯加犬(type)， 今年5岁了， 颜色是 红色。 它会 汪汪汪(cry) 它 会 演电影 《后会无期》(showFilm)

2) new Object 创建自定义对象

格式:

```
var 变量名 = new Object();
变量名.属性1 = 属性值1;
变量名.属性2 = 属性值2;
```

```
变量名.属性3 = 属性值3;
变量名.方法名1=匿名函数1;
变量名.方法名2=匿名函数2;
...
```

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // new Object 创建自定义对象
 /* 语法

  var 变量名 = new Object();
  变量名.属性1 = 属性值1;
  变量名.属性2 = 属性值2;
  变量名.属性3 = 属性值3;
  变量名.方法名1=匿名函数1;
  变量名.方法名2=匿名函数2;
  ...
 */

 // 定义了一个猫对象
 // var cat = {};
 var cat = new Object();

 // 猫的属性(猫的特性)
 cat.color = "white";
 cat.weight = "5kg";
 // 猫的方法(猫的特性)
 cat.skill = function(){
 console.log("猫可以抓老鼠~");
 }
 cat.cry = function(){
 console.log("喵~");
 }

 // 访问对象中的属性 语法: 对象名.属性名 或者 对象['属性名']
 console.log( cat.color );
```

```
36  console.log( cat["weight"] );
37
38  // 访问对象中的方法 语法: 对象名.方法()
39  cat.skill();
40  cat.cry();
41  </script>
42 </head>
43 <body>
44
45 </body>
46 </html>
```

小结:

- 跟我们前面学的 new Array() 一样。
- Object() 是构造函数 第一个字母大写
- new Object() 是调用构造函数 因为构造函数需要new 来调用 同时再内存中创建一个对象
- 注意里面使用的时候用点 不是 **冒号**

缺点: 每次都创建一个对象，还是存在大量的 重复代码， 并没有节约代码。

练习:使用new Object方式,创建一个 手机对象 型号(type) iphoneX 颜色是黑色 大小(size) 5.5寸 可以 发短信 (sendMessage) 输出 吃大肘子 可以打电话(call) 说 全聚德吃烤鸭

构造函数创建自定义对象 难点

我们可以和以前封装函数一样， 想创建多个对象，不用一个个的创造对象了。抽象可以将具有相同或相似功能的js代码独立出来封装成一个函数，这样可以提高代码的重复利用率，提高代码书写的效率，也可以有效的减少代码的冗余。**我们这个函数里面封装的对象， 为了和以前区别显示不同。 我们称为 构造函数。**

构造函数，是一种特殊的函数。主要用来在创建对象时初始化对象，即为对象成员变量赋初始值，总与new运算符一起使用在创建对象的语句中。

1. 构造函数用于创建某一大类对象，**类名建议首字母大写。**
2. 构造函数要和new一起使用才有意义。

语法:

```
//定义构造函数
function 类名(形参1,形参2,形参3...){
    this.属性1 = 形参1;
    this.属性2 = 形参2;
    this.属性3 = 形参3;
    this.方法名 = function(){
        方法体;
   }
}

//调用构造函数
var 变量名 = new 类名(实参1,实参2,实参3...);
//使用对象中某个属性
console.log(变量名.属性1);
console.log(变量名.属性2);
console.log(变量名.属性3);
//使用对象中某个方法
变量名.方法名();
```

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Document</title>
 <script type="text/javascript">
 // 我们之前学的{}跟new Object()创建对象的方法,每次只能创建一个对象,非常麻烦
 // 那么这个时候,如果想创建多个对象,我们就可以考虑使用构造函数来创建对象

  // 我们可以和以前封装函数一样， 想创建多个对象，不用一个个的创造对象了。

  // 抽象可以将具有相同或相似功能的js代码独立出来封装成一个函数，这样可以提高代码的重复利用率，提高代码书写的效率，也可以有效的减少代码的冗余。

```

```
// 我们这个函数里面"封装的对象"， 为了和以前区别显示不同。 我们称为 "构造函数"。

// 构造函数 ，是一种"特殊"的函数。主要用来"在创建对象时初始化对象"， 即为"对象成员变量赋初始值"，总与"new运算符"一起使用在创建对象的语句中。

// 构造函数用于创建某一大类对象，类名建议"首字母大写"。

// 构造函数要和new一起使用才有意义。

//定义构造函数
/*function 类名(形参1,形参2,形参3...){
this.属性1 = 形参1;
this.属性2 = 形参2;
this.属性3 = 形参3;
this.方法名 = function(){
方法体;
}
}*/

// 定义一个手机类
function Phone(brand,price,color){
// 属性
this.brand = brand;
this.price = price;
this.color = color;
// 方法
this.call = function(){
console.log( "手机可以打电话" );
}

this.send = function(){
console.log("手机可以发短信");
}

this.show = function(){
console.log("品牌为"+this.brand+"的手机,价格为"+this.price+",颜色为"+this.color);
}
}


```

```
53  //调用构造函数
54  // var 变量名 = new 类名(实参1,实参2,实参3...);
55
56  // 通过new从类中实例化出一个具体对象
57  var iphone = new Phone("iphoneX",5999,"黑色");
58  console.log( iphone );
59
60  var xiaomi = new Phone("小米10",3999,"白色");
61  console.log( xiaomi );
62
63  var huawei = new Phone("华为P30",3299,"天空蓝");
64  console.log( huawei );
65  console.log("");
66
67  //使用对象中某个属性
68  /*console.log(变量名.属性1);
69  console.log(变量名.属性2);
70  console.log(变量名.属性3);*/
71
72  console.log( iphone.brand );
73  console.log( iphone.price );
74  console.log( iphone.color );
75  console.log("");
76
77  //使用对象中某个方法
78  // 变量名.方法名();
79  iphone.call();
80  iphone.send();
81  iphone.show();
82  console.log("");
83
84  xiaomi.show();
85  console.log("");
86
87  huawei.show();
88  </script>
89 </head>
90 <body>
91
92 </body>
93 </html>
```

练习:

```
利用构造函数 创建一个英雄Hero 对象。
里面有 姓名属性(username),类型属性(type), 血量属性(blood), 具有 攻击方法(attack)

使用的英雄:
安琪拉       法师型       80血量       法术攻击
后羿         射手型     100血量       物理攻击
```

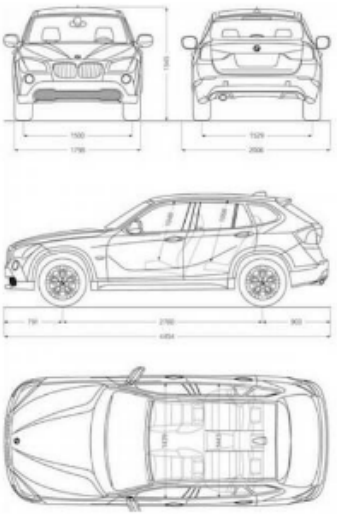
构造函数和对象(实例)

1. Person()是构造函数 主要是 初始化对象 用的。泛指某一大类 类似其他语言的 类 (class) 比如 苹果 一个 大类 类似 汽车图纸(包含 我们写好的 代码和数据 的模板)
2. new Person() 创建对象 用的。特指某一个 咱们前面指的 这个咬了一口的 苹果， 就是对象了。

是通过 new关键字 创建对象的过程 我们也称为 对象实例化。

宝马汽车就是 汽车对象 的一个 实例 。

汽车设计图纸 (构造函数)



靠，一辆真宝马！(对象实例)

new关键字

```
function Student(username, age) {
    this.username = username;
```

```
    this.age = age;
    this.sayHi = function() {
        console.log(this.username + ',你好');
    }
}
var lilei = new Student('李雷', 18);
lilei.sayHi();
console.log(lilei.username);

var hanmm = new Student('韩梅梅', 17);
hanmm.sayHi();
console.log(hanmm.username);
```

new在执行时会做四件事情

1. new会在内存中创建一个新的空对象
2. new 会让this指向这个新的对象
3. 执行构造函数里面的代码 目的:给这个新对象加属性和方法
4. new会返回这个新对象 (所以构造函数里面不需要return)

this详解

JavaScript中的this指向问题，有时候会让人难以捉摸，随着学习的深入，我们可以逐渐了解
现在我们需要掌握函数内部的this几个特点
1. 函数在定义的时候this是不确定的，只有在调用的时候才可以确定
2. 一般函数直接执行，内部this指向全局window
3. 函数作为一个对象的方法，被该对象所调用，那么this指向的是该对象(谁调用指向谁)
4. 构造函数中的this 指向这个对象的实例

举例:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script type="text/javascript">
  // JavaScript中的this指向问题，有时候会让人难以捉摸，随着学习的深入，我们可以逐渐了解
  // 现在我们需要掌握函数内部的this几个特点
  // 1. 函数在定义的时候this是不确定的，只有在调用的时候才可以确定
```

```
// 2. 一般函数直接执行，内部this指向全局window
// 3. 函数作为一个对象的方法，被该对象所调用，那么this指向的是该对象(谁调用指向谁)
// 4. 构造函数中的this 指向这个对象的实例

// 直接在代码中输出this,这个this代表window对象
console.log( this );

// 在函数中使用this,谁调用这个this就代表谁,默认情况下,函数是由window对象调用的
function fn(){
console.log( this );
}
fn();
window.fn();
console.log("");

// 在js代码中,定义的函数跟变量,都会被放在window对象下
var a = 10;
console.log( a );
console.log( window.a);
console.log("")

var person = {
"name":"张三",
"age" : 23,
"sex" : "男",
call : function(){
// 在哪个对象中的方法输出this,这个this就代表谁
console.log( this );
}
};

// 输出person对象
console.log( person );
// 调用person对象中的call方法
person.call();

console.log("");
console.log("");

```

```
49
50  // 构造函数
51  /*function 类名(形参1,形参2,形参3...){
52  this.属性名1 = 形参1;
53  this.属性名2 = 形参2;
54  this.属性名3 = 形参3;
55
56  this.方法名1 = 匿名函数;
57  this.方法名2 = 匿名函数;
58  }*/
59
60  function Student(number,name,age){
61  this.number = number;
62  this.name = name;
63  this.age = age;
64
65  this.code = function(){
66  console.log( this.name+"同学正在开心快乐的写代码~" );
67  };
68
69  this.eating = function(){
70  console.log( this.name+"同学正在吃饭" );
71  }
72
73  this.showInfo = function(){
74  // 这里的this代表当前对象
75  console.log( this );
76  console.log( "同学的个人信息,学号为"+this.number+",姓名为"+this.name+",年
龄为"+this.age );
77  }
78  }
79
80  // 想从类中得到具体对象,需要实例化,通过new
81  var student1 = new Student(1,"张三",23);
82  console.log( student1 );
83  console.log( student1.name );
84  student1.showInfo();
85
86  console.log("");
87  console.log("");
88  console.log("");
```

```
89
90  var student2 = new Student(2,"李四",24);
91  console.log( student2 );
92  student2.showInfo();
93
94  </script>
95 </head>
96 <body>
97
98 </body>
99 </html>
```

小结

对象 是 封装了 相关 属性 和 方法 的 复杂数据类型

本质:对象就是一组 无序的 **相关属性和方法** 的 **集合**

注意: 函数 用来按功能 **封装代码**，对象 用来按功能 **封装方法和属性**，都起到复用代码和数据的作用

- 构造函数 泛指 某一大类 比如 苹果 不管红色苹果还是 绿色苹果 都是统称为 苹果 类似其他语言的类
- 对象实例 特指 一个事务 比如 这个苹果

通俗:实例 就是按照汽车设计图纸 造一辆辆的车

概念:实例 是 某种对象类型 的一个具体存在

本质:实例 就是根据对象模板 创造 的一个对象实体

今日总结

xmind要做,今天新内容不多,主要是讲的创建对象的方法以及如何访问对象属性和如何对象的方法

今日作业

今天快下课布置的练习,做完也要交给助教老师,做完练习再做作业

课堂练习也要交

练习: 请用{}形式创建 一个 王可可 的狗对象。 具体信息如下:

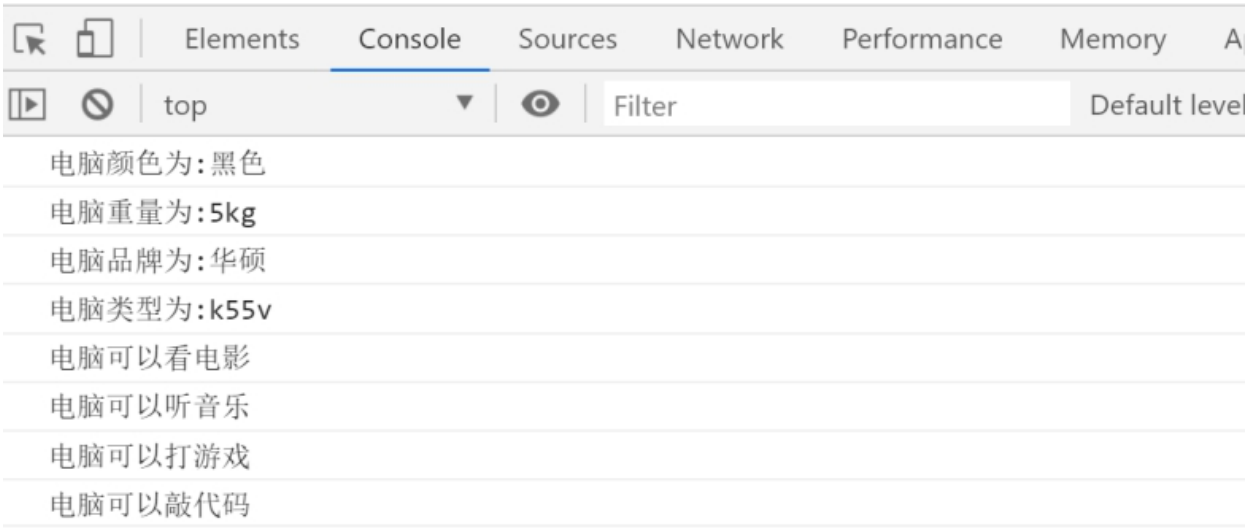
王可可 是一条 阿拉斯加犬(type), 今年5岁了, 颜色是 红色。 它会 汪汪汪(cry) 它 会 演电影 《后会无期》(showFilm)

练习:使用new Object方式,创建一个 手机对象 型号(type) iphoneX 颜色是黑色 大小(size) 5.5寸 可以 发短信 (sendMessage) 输出 吃大肘子 可以 打电话(call) 说 全聚德吃烤鸭

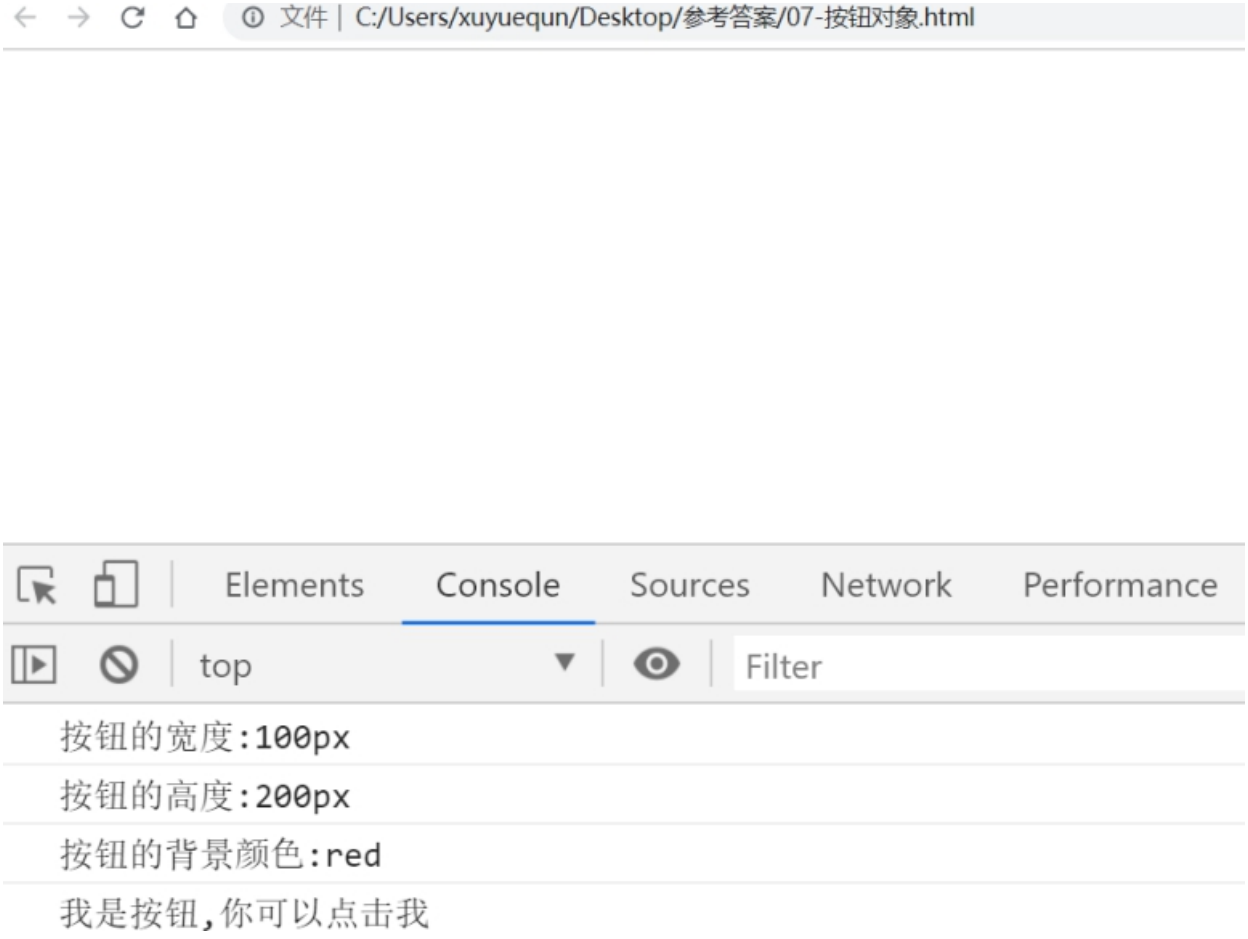
练习:

```
1 利用构造函数 创建一个英雄Hero 对象。
2 里面有 姓名属性(username),类型属性(type), 血量属性(blood), 具有 攻击方法(attack)
3
4 使用的英雄:
5 安琪拉      法师型      80血量      法术攻击
6 后羿        射手型     100血量      物理攻击
```

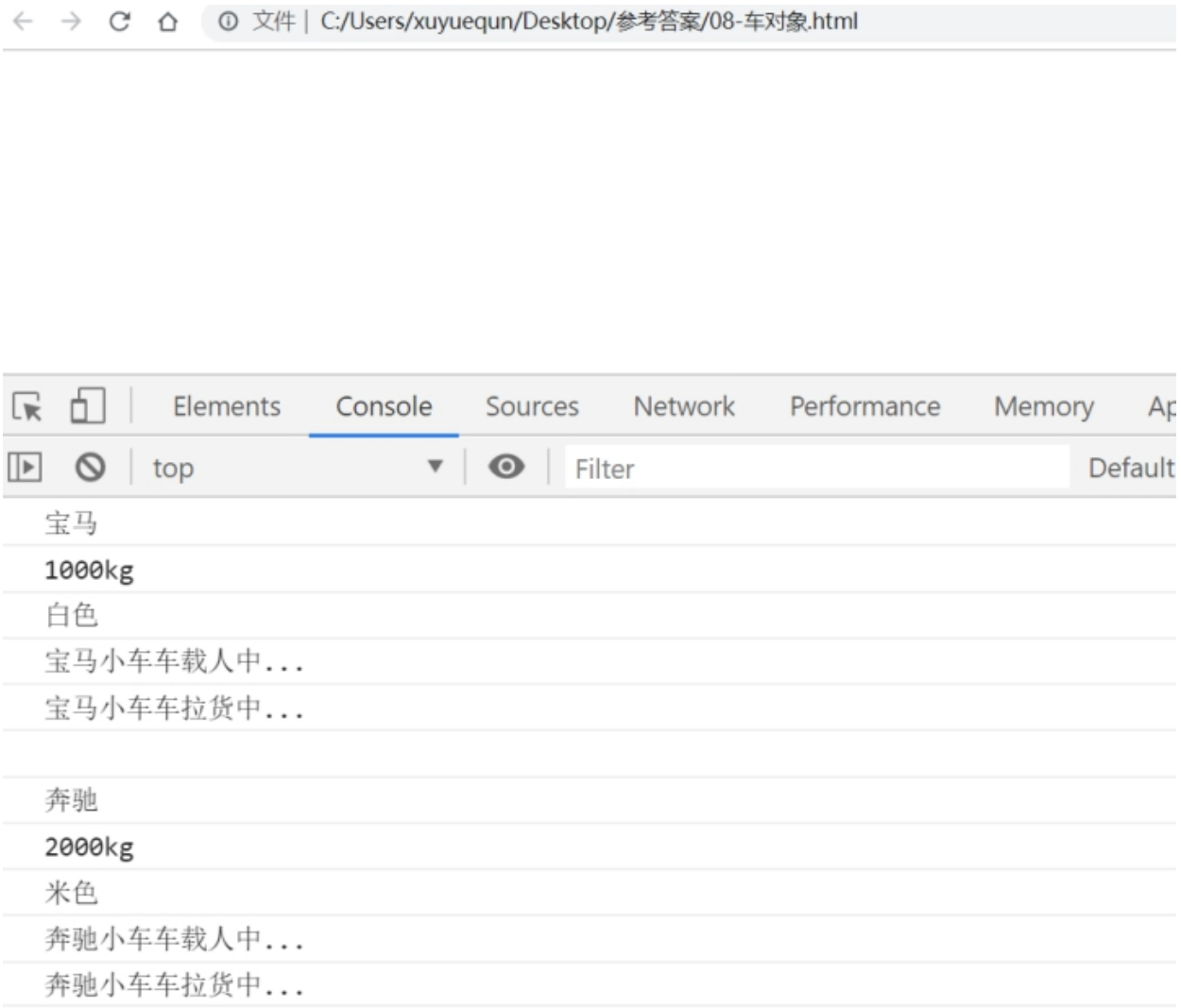
1、用{}方式,创建一个电脑对象, 有颜色,有重量,有品牌,有型号,可以看电影,可以听音乐,可以打游戏,可以敲代码



2、用new Object()方式,创建一个按钮对象,宽,高,背景颜色,可以鼠标点击



3、用构造函数方式,创建一个车的对象,有重量,颜色,牌子,可以载人,拉货


文件 | C:/Users/xuyuequn/Desktop/参考答案/08-车对象.html
Elements
Console
Sources
Network
Performance
Memory
top
Filter
Default
宝马
1000kg
白色
宝马小车车载人中...
宝马小车车拉货中...
奔驰
2000kg
米色
奔驰小车车载人中...
奔驰小车车拉货中...