p的指向性问题

```
function setObj(p){
    p.name="小红";
    p={};
    p.name="西西"
    console.log(p)//西西
}
var p={
    name:"小明"
}
setObj(p);
console.log(p);//小红
```

解析

```
function serObj(p){
    p.name="小红"; //通过p找到小明,
    //--将原来的小明改名为小红
    p={};//创建新对象开辟了一个新的内存
    //--至此局部变量p和外面存"小明"的
    //--那块内存没关系了
    p.name="西西"
}
var p={
    name:"小明"
}
setObj(p); //小红
console.log(p);
```

call函数的应用

涉及到函数调用时，里面的this不是我们想要的怎么办

```
function calc(base,bonus1,bonus2){
    ${this.ename}的总工资是
    ${base+bonus1+bonsu2}
}
```

var lilei={ename:"Li Lei"}

> lilei 不可能直接调用call ,calc
> 就不能直接调用lilei的参数，这时候就可
> 以用call来
> `calc.call(hmm,4000,5000,6000);`

every回调函数的使用过程

```
 arr1=[1,2,3,4,5]
 arr2=[2,4,6,4,2]
var a=arr1.every(function(elem,i,arr){
  return elem%2==0;
}
)
console.log(a)//此every中elem为对应元素，i为元素对应索引值
 //此函数可以验证数组中的元素是否都为偶数
```

forEach

```
 arr1=[1,2,3,4,5]
 arr1.forEach(function(elem,i,arr){
     arr1[i]=arr[i]*2//这样子的话，原数组就会
     //被更改
     //elem=arr[i]*2//这样就会复制副本之后
     //在更改，不会影响原来的的数组
 }
 )
 var emps=[
     {eid:1,ename:"liang",salary:12000},
     {eid:2,enamel:"ran",salary:16000},
     {eid:3,ename:"dong",salary:1800}
 ];
emps.forEach(function(elem){
     //elem虽然发生了复制，但是复制的是地址，
     //用地址操作对象相当于操作了源对象，跟改了
     //原的对象
     elem.salary+=2000;
 })
 console.log(emps);
```

map 函数

> map()方法将调用的数组的每个元素传递给指定
> 的函数，并返回一个数组，它包含该函数的返回值。

```
arr =[1,2,3,4,5]
var lis= arr.map(function(elem){
    return `<li>${elem}</li>`
})
console.log(arr);
console.log(lis);
```

filter函数

```
var arr=[1,2,3,4,5];
var evens=arr.filter(
//先创建空数组等着接符合条件的元素
//var subArr=[]
//filter会拿着回调函数去每个元素上判断一次
  function(elem){
    //只要当前元素是偶数
    return elem%2==0//就放入新数组
  }
//如果当前元素判断结果为true，就追加到新数组
//if(return true) subArr.push(elem)
//遍历结束，返回subArr
)
console.log(arr);
console.log(evens);
```

reduce 函数

```
var class1=[1,3,5,7,9];
var result=class1.reduce(
 function(prev,elem){
   return prev+elem;//将当前值汇总到临时汇总值中
 },
 0//从0开始累加
);
var class2=[2,4,6,8,10];
result=class2.reduce(
 (prev,elem)=>prev+elem,
 result//从上一个班捐的总数继续开始累加,没有他
 //class2前面的result就是个新的
);
console.log(result);//25+30=55
```

let 函数

```
function fun(){
 var arr=[];//let将for变为块级作用域之后，
 //for内的变量在for外，不能使用。要想使用，
 //必须定义在for外部。
 //for(var i=0;i<3;i++){
 for(let i=0;i<3;i++){
   arr[i]=function(){console.log(i)}
 //(function(_i){
   //arr[_i]=function(){console.log(_i)}
 //})(i)
 }
 return arr;
}
var funs=fun();
funs[0]();//3  0
funs[1]();//3  1
funs[2]();//3  2
</script>
```

1.申明提前问题

```
function fun(){ console.log(1);}
fun();
function fun (){console.log(2);}
fun();
var fun=100;
fun();
```

分析

```
function fun(){ console.log(1);}//打酱油
fun(); //2
function fun (){console.log(2);}
//近似相当于var fun=function(){...}
fun();//2
var fun=100;
fun();  // 报错，fun现在由数字取代函数了，这么写报错
//遇到声明提前先提前 在分析，函数名fun只是个变量，
//函数就是个对象
```

数据类型

```
console.log(
  typeof 1,
  typeof "hello",
  typeof true,
  typeof null,
  typeof undefined,
  typeof function(){},
  typeof {sname:"Li Lei"},
  typeof [1,2,3],
  typeof NaN
);
```

如何正确判断**null**和**undefined***

```
var obj=undefined;//null
//undefined在底层会被自动转换成null
console.log(obj===null);
//屏蔽了类型转换的可能
var a=0;
console.log(a=="");//true
console.log(a==="");//false


var a=false;
console.log(a==="");//false
console.log(a==="");//false
//总结：今后判断null,undefined,0,"",false都要用===
//同理：!==（不全等于号）是屏蔽了自动顶
```

如何判断正确判断**NaN---isNaN()**

```
var a=10, b=NaN,c="",d=false;
console.log(isNaN(a))//false
console.log(isNaN(b))//true
console.log(b=NaN)//false,也是一个判断方法
console.log(isNaN(c))//false
console.log(isNaN(d))//false
```

如何判断数组和对象

```
var a={}, b=[],c=new Date();
console.log(//ES5
  Array.isArray(a),
  Array.isArray(b),
);
console.log(
  //toString()可输出一个对象内部属性class，
  //查看对象的类型名call()可强行让ab有这个查询方法
  Object.prototype.toString.call(a)===
  "[object object]",//true
  {}.toString.call(b)==="[object Array]"//更加高级
  object.prototype.toString.call(b)===
  "[object Array]"//true
);
```

字符串操作

统计一个字符串中字符出现最多的字符

```
var str="helloworld";
//先定义空对象
var dict={};
//遍历字符串中的米格字母
for(var i=0;i<str.length;i++){
   if(dict[str[i]]===undefined){
     dict[str[i]]=1;
   }else{dict[str[i]]+=1;

   }
}
console.log(dict)
var max=0,count=0;
for(var key in dict){

  if(dict[key]>count){
    max=key
    count=dict[key]

  }
}
console.log(max,count)
```

数组去重

```
var arr=[1,2,3,4,3,2];//不许用API
//用字典，不许出现 重复的下标,出现自动覆盖
var dict={};
for(var i=0;i<arr.length;i++){
  dict[arr[i]]=1;
}
//字典有个特点，同名同值zhi'neng'chu'xia
//遍历字典，值保留属性名只能出现一次
var result=[],i=0;
for (result[i++] in dict);//for...in遍历数组
//in 是依次取出属性名，并保存在result中
console.log(result);
```

this指向性问题

```
var a=10 ;
var obj ={
  a:20,
  intr:function(){
    var a=30;
    console.log(this.a)
  }
}
obj.intr();//20
var intr=obj.intr;//20
intr();//10,intr变成最普通的函数了
```

闭包问题

```
var funs=[

];
for (var i=0;i<3;i++){
  funs[i]=(function(i){
    return function(){
      console.log(i)
    }
  })(i);//实参传递给形参，拷副本按值传递123
}
funs[0]();//1，funs[0]位置被赋值之后立即，
//里面是个匿名函数自调用，立即被执行了
funs[1]();//2
funs[2]();//3
```

```
function fun(){
  for(var i=0,arr[];i<3;i++){
    arr[i]=function(){
      console.log(i)
    }
  }
  return arr;
}
var funs=fun();
funs[0]();//3 普通函数，完全循环完了之后再调用的
funs[1]();//3
funs[2]();//3
```

> 闭包：画简图，找两样东西
> 找外层函数和内层函数的局部变量
> 外层函数被调用了几次，内层函数被调用了几次
> 就反复生成了内层函数几次

垃圾回收机制

1. 监控每个对象被几个变量引用着
2. 若引用数为0时就回收

> 闭包既重用变量又不造成全局污染，只能手动释放，
> 占用内存空间，造成内存泄漏

面向对象

深拷贝和浅拷贝

```
var lilei={
  sname:"Li Lei",
  addres:{
    prov:"北京",
    city:"北京",
    area:"海淀",
    stress:"万寿路"
  }
  sage:11
}
function clone(obj){
  var newObj={};
  for (var key in obj){
    newObj[key]=obj[key]
  }
  return newObj;
}
var lilei2=lilei;//仅仅只是复制了指针地址

 //以下为浅克隆，克隆了对象平级的内容
 //若对象里面还有对象，如address,则指克隆address指针
 //内嵌对象中的内容如city改了后，lelei3的address里的city
 //更改
 var lelei3=clone(lilei);//创建新的地址储存对象的方法



 //以下为深克隆
 function clone2(obj){
   if(obj===null){
     return null;
   }
   if({}.tostring.call(obj)==="[object Array]"){
     var newArr=[];
     newArr=obj.slice();//对象素偶又内容切完丢进去
     return newArr;
   }
   var newObj={};
   for(var key in obj){

     //如果原对象中当前属性值时原始类型
     if (typeof obj[key]!=="object"){
       //在新对象中添加和原对象中同名的属性
       newObj[key]=obj[key];
     }else{//否则，当前属性不是原始类型的值，
     //再次调用clone函数，继续复制当前属性
     newObj[key]=clone2(obj[key])

     }
   }
```

```
    return newObj;
  }
```

函数调用和rguement问题

> 其实Javascript并没有重载函数的功能，但是Arguments对象能够模拟重载。Javascrip中每个函数都会有一个Arguments对象实例arguments，它引用着函数的实参，可以用数组下标的方式"[]"引用arguments的元素。arguments.length为函数实参个数，arguments.callee引用函数自身。

```
var length=10;
function fn(){
  console.log(this.length);//10,this->window
}
var obj={
  length:5,
  method:function(fn){
    fn();
    //arguments:[fn,1].length=2
//arguments[0]()等价于arguments.0()=>this->arguments
    arguments[0]();//argument.fn()得到的是2
  }
}
obj.method(fn,1)
```

var 和声明提前问题

```
function fn(a){
  console.log(a);
  var a=2;
  function a(){}
  console.log(a)

}
fn(1)//输出 f a(){}和2
```

声明提前问题和作用域问题

```
var f=true;
if(f===true){
  var a=10;
}
 function fn(){
   var b=20;
   c=30;//强行全局创建
 }
 fn();
 console.log(a);//10
 console.log(b);//undefine
 console.log(c);//30
```

声明和删除问题

```
//in:专门判断一个对象中，是否包含指
//定名称的属性，包含返回true否则false
var a=10;
console.log(window)
console.log(a)
console.log(window.a)
console.log(window["a"])

//3中创建的区别
var a1=10;//delete不可删除他
window.a2=20;//可被delete删除
window["a3"]=30;//可被delete删除


  if('a' in window){//js中没有块级作用域
    var a=10;
  }
  alert(a);
```

给基本数据类型添加属性，不报错

包装类型

```
//原始类型值太过简单,没有设计好的api
//那要用怎么办？
//str.toUpperCase()
//每次对原始类型的值调用方法或者访问属性，
//其实是对自动创建的包装类型对象调用方法和访
//问属性，且包装类型对象使用完自动释放
```

```
var a=10;//原始类型
a.pro=10;//typeof a->number
//new Number(10).pro=10;
console.log(a.pro+a);//NaN
//new Number(10).pro
      //undefinde    +10//NaN
var s="hello";//原始类型
s.pro="world";//typeof s string
console.log(s.pro+s);
//new String(s).pro
    //undefined     +s//字符串拼接
```

经典闭包问题

```
<button>1</button>
<button>2</button>
<button>3</button>
<button>4</button>
<button>5</button>
<script>
  var btns=document
  .getElementsByTagName("button");
  var i=0;
  for (var btn of btns){
    i++
    btn.onclick=(function(i){
      return function(){
        alert(i);
      }
    })(i)//原始类型：按值传递，复制副本
  }
</script>
```

面向对象和this结合问题

```
function JSClass(){
  //写入一个元素
  this.m_Text="division element";
  this.m_Element=document.createElement("div");
  this.m_Element.innerHTML=this.m_Text;
  //div上海绑定了一个单击事件
  this.m_Element.addEventListener(
    "click",this.func//在本句后面加上.bind(this)

  )//在事件处理函数等回调函数传递时，仅传递.后的函数对象
  //不会保留.前的对象，当回调函数在执行时，this和
  //原来的.前的对象无关this.m_Text会显示undefined,
  //加上.bind(this)后会解决这个问题
}
//额外定义两个方法
JSClass.prototype.Render=function(){
  document.body.appendChild(this.m_Element)

}
JSClass.prototype.func=function(){
  alert(this.m_Text);
}
var jc=new JSClass();
jc.Render();
jc.func();
```

split问题

> 请编写一个js函数parseQueryString,他的用途是吧url参数解析为一个对象，如：var url = "http://witmax.cn/index.php?key0=&key1=1&key2=2"

```
var search="?key0=0&key1=&key2=2"
//去问号
search=search.slice(1);
//按照&切开
var strs=search.split("&");
console.log(strs);
var params={}
//遍历strs中的每个字符串
for(var str of strs){
  var arr=str.split("=");
  params[arr[0]]=isNaN(arr[1])?
  arr[1]:Number(arr[1])
}
console.log(params)
```

去重,排序

```
//冒泡 排序
var arr=[2,6,6,5,3,8,9,12,3,1]
for(var i=0;i<arr.length-1;i++){
  for(var j=0;j<arr.length-1-i;j++){
   if(arr[j]>arr[j+1]){
     var temp=arr[j]
     arr[j]=arr[j+1];
     arr[j+1]=temp;
   }


  }

}
 for(i=0;i<arr.length;i++){
    var c=arr[i];
    for (var s=i+1;s<arr.length;s++){
      if(arr[s]==c){
        arr.splice(s,1);
        s--
      }
    }
  }
console.log(arr)
```

检测浏览器版本有哪些方式

> 1.根据navigator.userAgent//UA.toLowerCase().indexOf('chrome')
> 2.根据window对象成员//'ActiveXObject' in window

```
var browser=navigator.appName
var b_version=navigator.appVersion
var version=parseFloat(b_version)

document.write("Browser name: "+ browser)
document.write("<br />")
document.write("Browser version: "+ version)
```

js的内置对象有那些？

> ES中的内置对象共11个
> Number String BNoolean ---包装类型
> Array Date RegExp Math
> Error (try{...}catch{...}) throw Error("...")

> Function Object
> global(浏览器中被window代替)

如何提高性能的javascript？

> 1."use strick"
> 2.js放在页面底部，
> 3.js脚本组成打包
> 4.非阻塞下载js脚本
> 5.使用局部变量
> 6.减少闭包使用
> 7.缓存dom节点
> 8.避免是有eval()个Function()构造器
> 9.serTimeout()和setInterval()传递函
> 数而不是字符串作为参数
> 10.尽量使用直接量创建对象和数组
> 11.最小化重绘(repaint)和回流(reflow)(dom操作)

attr() vs prop()

html中

```
<input type="checkbox" id="chb" checked data-i="5" title="hello">
<script>
$().arrt()=elem.gerAttribute()/setAttribute()//这个只能访问自定义属性
btn{
 title:"hellow",
 checked:reue,//btn.属性名；这个可以访问核心属性
}
</script>
```

介绍javascript的原型，原型链？有什么特点？

1.js的所有对象都包含了一个[prop]内部属性，这个
属性所对应的就是该对象原型
2.js的函数对象，除了原型之外，还预制了prototype
属性
3.当函数对象作为构造对象创建实例时，就回去[proto]关联的前辈prototype对象去找
4.如果没有找到，就会去该prototype原型[proto]关联的前辈prototype去找，一次类推；直到找到属性/方法或者undefined
特点：

5.js对象是通过引用来传递的，当修改原型时，与之相关的对象也会继承这一改变