**一、变量**

var

1. 可以重复声明
2. 无法限制修改
3. 没有块级作用域

let 不能重复声明，变量--可以修改 ，块级作用域

const 不能重复声明，常量--不可以修改 ，块级作用域

**二、箭头函数**

箭头函数

function(){

​

}

//相当于把function省略,()后加=>

()=>{}

​

let show=function(a,b){

alert(a+b);

}

​

let show=(a,b)=>{

alert(a+b);

}

show(1,2);

箭头函数只是一个简写

1. 如果只要一个参数，()可以省
2. 如果只有一个return,{}连带return可以省

let show=function(a){

return a;

}

​

let show=a=>a;

**三、函数的参数**

1. 参数扩展/展开
2. 默认参数

参数扩展：

1. 收集剩余的参数

function show(a,b,...args){};

\*Rest Parameter必须是最后一个

function show(a,b,...args){

alert(a); //1

alert(b);//2

alert(args); //args是剩余参数 4,5，6

}

//...args剩余参数必须是最后一个形参，...后可自定义剩余参数参数名

show(1,2,3,4,5,6);

1. 展开数组

展开后的效果，跟直接把数组的内容写在这一样

let arr1=[1,2,3];

let arr2=[5,6,7];

let arr=[...arr1,...arr2];//1,2,3,4,5,6

​

let arr=[1,2,3];

...arr

1,2,3,

show(...arr) 等价于 show(1,2,3)

​

​

funcction show(...args){

fn(...args);

}

function fn(a,b){

alert(a+b);

}

show(12,5);

默认参数

function show(a,b=5,c=12){

console.log(a,b,c);

}

show(99); //99,5,12

show(99,12); //99,12,12

show(99,12,6); //99,12,6

**四、解构赋值**

解构赋值：

1. 左右两边必须一样
2. 右边必须是个东西
3. 声明和赋值不能分开（必须一句话里完成）

let [a,b,c]=[1,2,3];

let {a,b,c}={a:12,b:5,c:8};

console.log(a,b,c);

let [{a,b},[n1,n2,n3],num,str]=[{a:12,b:5},[12,5,8],8,"feifei"];

//传粒度也可以。只有两边结构一致

let [json,arr,num,str]=[{a:12,b:5},[12,5,8],8,"feifei"];

console.log(json,arr,num,str);

let {a,b}={12,3}; //{12,3}既不是数组，又不是json

console.log(a,b);//报错 右边必须是个东西

​

let [a,b];

[a,b]=[1,2];

console.log(a,b);//报错 声明和赋值不能分开

**五、数组**

数组：

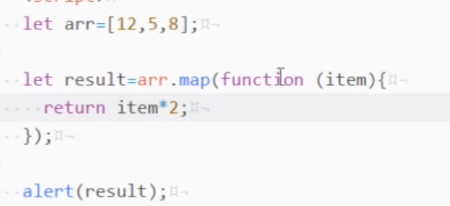
map 映射 一个对一个（给多少返回多少，处理函数）

let score=[19,8,94,60,57];

let result=score.map(item=>item>=60?'及格':'不及格')；

alert(score); //19,8,94,60,57

alert(result); // [不及格，不及格，及格，及格，不及格]



reduce 汇总 一堆出来一个

let arr=[12,69,180,8763];

let result=arr.reduce(function(tmp,item,index){

return tmp+item; //求和

})

alert(result); //9024

let result=arr.reduce(function(tmp,item,index){

if(index!=arr.length-1){ //不是最后一次

return tmp+item; //求和

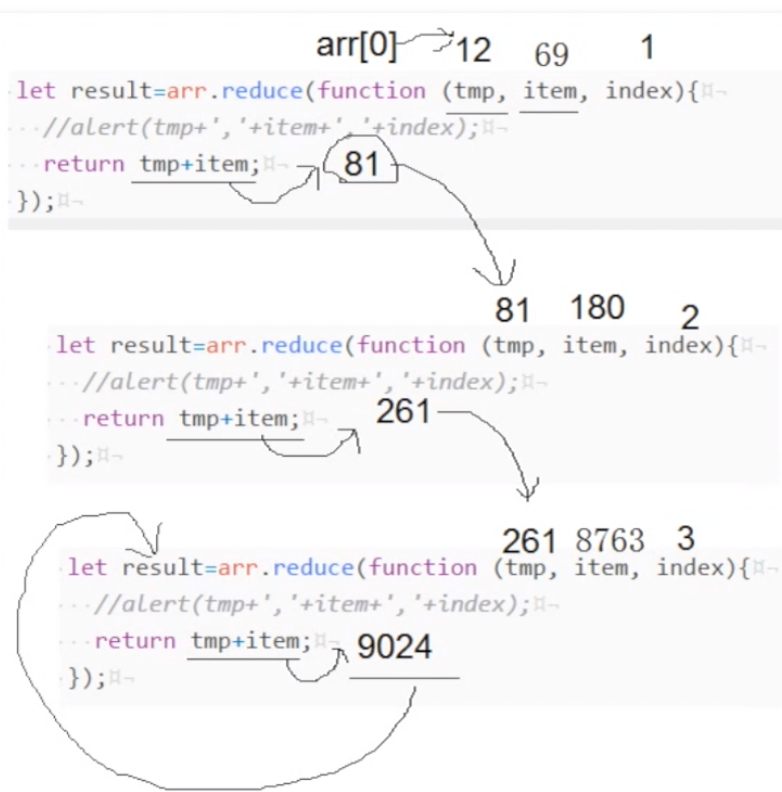
}else{ //最后一次

return (tmp+item)/arr.length; //this.length 中this此时指向window

}

})

alert(result); //2256



filter 过滤器

let arr=[12,5,8,99,27,36,75];

let result=arr.filter(item=>{

if(item%3==0){

return true;

}else{

return false;

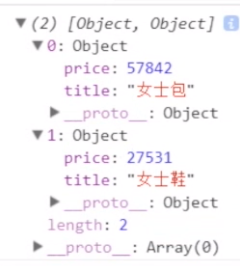
}

});

let result=arr.filter(item=>item%3==0);

alert(result); //留下能被3整除的





forEach 循环（迭代）

let arr=[12,5,13,4];

arr.forEach((item,index)=>{

alert(index+':'+item);

})

**六、字符串**

字符串：

1. 多了两个新方法

**startsWith**



**endsWith**



1. 字符串模板 （反单引号 ``)

let a=12;

let str=`a${a}bc`;

alert(str); //a12bc

字符串连接:

* 1. 直接把东西塞到字符串里面 ${东西}
  2. 可以折行



**七、面向对象**



//es6新方法

class User{

constructor(name,pass){ //创建实例时初始化

this.name=name;

this.pass=pass;

}

showName(){

alert(this.name);

}

showwPass(){

alert(this.pass);

}

}

var u1=new User('bule','123456');

u1.showName(); //bule

u1.showPass(); //123456

es6的面向对象：

1. class关键字、构造器和类分开了
2. class里面直接加方法

继承：

//es6新方法

class User{

constructor(name,pass){ //创建实例时初始化

this.name=name;

this.pass=pass;

}

showName(){

alert(this.name);

}

showwPass(){

alert(this.pass);

}

}

class VipUser extends User{

constructor(name,pass,level){ //创建实例时初始化

super(name,pass); //超类

this.level=level;

}

showLevel(){

alert(this.level);

}

}

var v1=new VipUser('bule','123456',3);

v1.showName(); //bule

v1.showPass(); //123456

v1.showLevel(); //3

**八、面向对象应用--React**

**九、JSON对象**

json：

1. JSON对象：

JSON.stringify() 将json对象转化为字符串

JSON.parse() 将字符串转化为json对象

1. 简写

名字跟值（key和value)一样的，留一个就行

方法 :function可以省略

show:function(){...}

show(){...}

json的标准写法：

1. key只能用双引号
2. 所有的名字都必须用引号包起来
3. {a:12,b:5} //false
4. {"a":12,"b":5} //true
5. {"a":"abc","b":5} //true
6. {"a":'abc',"b":5} //false

**十、Promise --承诺（一次读一堆）**

Promise --消除异步操作

用同步一样的方式来书写异步代码（本质还是异步）

let p1=new Promise(function(resolve,reject){

//异步代码

//resolve --成功了

//reject --失败了

$.ajax({

url:'arr.txt',

dataType:'json',

success(arr){

resolve(arr); //返回成功结果

},

error(err){

reject(err); //返回错误结果

}

})

});

let p2=new Promise(function(resolve,reject){

//异步代码

//resolve --成功了

//reject --失败了

$.ajax({

url:'json.txt',

dataType:'json',

success(arr){

resolve(arr); //返回成功结果

},

error(err){

reject(err); //返回错误结果

}

})

});

//当Promise调用有结果时调用then()方法 then()第一个参数为resolve，第二个参数为reject

//通过then()把回调函数里的结果取出来

p.then(function(arr){

alert('成功了'+arr);

},function(err){

alert('失败了'+err);

});

//封装后的

function createPromise(url){

return new Promise(function(resolve,reject){

$.ajax({

url:url,

dataType:'json',

success(arr){

resolve(arr); //返回成功结果

},

error(err){

reject(err); //返回错误结果

}

})

});

}

Promise.all([

createPromise('data/arr.txt'),

createPromise('data/json.txt')

]).then(function(arr){ //arr作为一个封装的返回结果

let [ret1,ret2]=arr; //解构赋值

alert('全部成功了');

},function(){

alert("至少有一个失败了");

})



**十一、generator --生成器（逻辑性）**

普通函数 --一路到底

generator --中间能停

function \*show(){

alert('a');

yield;

alert('b');

}

let genObj=show(); //generator创建了一个对象，接着这个对象才可以执行方法

genObj.next(); //a

genObj.next(); //b

yield会中间返回结果 最后结果需要return 返回

