**关于let、const**

提出这两个新东西的原因还是var的三个大问题

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-可修改、块级作用域(一个花开括号就是一个块级作用域)

Const 不能重复声明、常量-一旦声明不可以被修改、块级作用域

**关于箭头函数**

PS：箭头函数只是函数的一种缩写(目前认知)

原来： **function** Fn(a){**return** a\*2;}  
箭头： **let** Fn =(a) =>{**return** a\*2}

进一步的缩写有两点：

1. 如果函数的参数数量为1时，可以省略参数括号（）；
2. 如果参数中只有一句return语句，可以省略大括号连带return

**let** Fn = a => a\*2; //最终缩写版

**关于参数拓展、默认参数**

//参数拓展(关键词 … )

**function** S(a,b,...argts) //此处…argts会搜集传参时多余出的参数并,且可以将其打印出来(貌似无法对其进行操作)

-----------------------------------------------------------

**let** arr = [1,5,9,3,0,6]; //参数展开  
**function** B(...argts){console.log(...argts);}  
B(...arr); //还可以将数组展开 ...arr的效果就和直接传入1,5,9,3,0,6一样

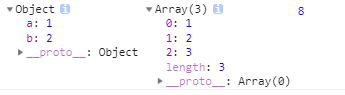
//默认参数

**function** T(a,b=1,c=3){console.log(a, b, c);}  
T(10,9);//如果此处再传入b和c的参数就覆盖，没有传入就用默认参数

**关于解构赋值**

解构赋值三个注意点：  
 1.两个必须对应  
 2.右边不得出现语法错误  
 3.声明赋值不能分开

**let** [a,b,c] = [{a:1,b:2},[1,2,3],8];



**关于map、reduce、filter数组操作(并非ES6新内容)**

//map 映射 == 传进去多少，吐出来多少（返回值）  
**var** a = [1, 2, 3, 4, 5];  
a = a.map(num => num \* 2); //箭头函数简写  
console.log(a);//返回一个在原基础上数值\*2的新数组

**----------------------------------------------**

//reduce 汇总 == 传进去多少，吐出来一个（返回值）  
**var** b = [1, 2, 3, 4, 5];  
b.reduce(**function** (big, num, index) { //三个参数分别为返回的最终结果（默认拿第0个数值当总和，所以一开始的下标为1）、当前的数值、当前的下标  
 **return** big+num; //一直返回，直到没有num可以操作  
})

**----------------------------------------------**

//filter 过滤 == 留下你想要的，不想要的丢掉（返回值）  
**var** c = [1,2,3,4,5];  
c = c.filter(val=>val%2===0);//只留下可以整除2的数值  
console.log(c);

**----------------------------------------------**

//forEach 迭代 == 我懂，不哆嗦  
**var** d = [1,2,3,4,5];  
d.forEach(**function**(value,index){ //前面是值，后面是下标  
 console.log(index, value);  
})

**关于字符串新增方法和字符串模板**

//新增startsWith()和endsWith()用于判断字符串前后的内容

**var** a = 12;//抄python的  
**var** b = `  
aa  
  
aaa  
aa${a}`;//反引号可任意换行 ${}在字符串中插入内容，  
console.log(b);

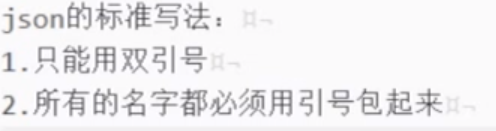
**关于ES6新的OOP创建方式**

**class** a {//抄JAVA的  
 **constructor**() {  
 **this**.name='aaa'; //属性  
 }  
 say(){ //方法  
 alert(**this**.name);  
 }  
}  
**var** A = **new** a();  
A.say();

-----------------------------------------------------------

**class** b **extends** a{ 新的集成方法(继承者 extends 被继承)  
 **constructor**(){  
 **super**(); //java中没有父类，只有超类  
 }  
}  
**var** B = **new** b();  
B.say();

关于JSON(非ES6新内容)



JSON的三种常用方法

.stringify()将一个json串转化为字符串

.parse()将json转化为对象 ps:只接受上面标准的JSON

Eval(‘(‘+JSON+’)’)这种写法可以转化不标准和标准的JSON

----------------------------------------------------------

关于JSON的简写格式

**var** a = 'aaa'; //简写1  
**var** b = 'bbb';  
**var** c = {a,b};   
console.log(c); 打印为a:’aaa’,b:’bbb’

-----------------------------------------------------------

**var** c = { //简写2   
 b:**function**(){  
 alert(**this**.a);  
 },  
 c(){  
 alert(**this**.a);  
 }  
}  
c.b(); //b可以简写为c  
c.c();

**关于promise承诺**(不哆嗦，没理解，以后懂了记得补上)

[**Promise.html**](file:///D:\PHPTutorial\WWW\ES6\promise.html)**(服务器)**

**关于generator生成器** [**(generator.html)**](file:///C:\Users\123\AppData\Roaming\Microsoft\小实例\ECMAscript6.0\generator生成器.html)

//generator和普通function的区别就是\*，可以贴着a也可以贴着function，或者都不贴着，但是不可以两个都贴着

**PS:yield前面为传参，后面为返回值**

**function** \*a(){

alert('a');  
 **yield**; //分割点  
 alert('b');  
}  
**var** A = a();//generator对象无法通过函数本身直接调用，需要利用他创建一个新对象  
A.next(); //利用对象.next()执行一步  
// A.next();//想再走一步须再调用一次此方法

**----------------------------------------------**

//关于yield  
/\*  
1.一个.next()对应一个节点的yield  
2..next()可以传参，参数会传给下一个继承yield的变量  
3.yield可以返回，返回值会传给对应的.next()  
4.第一个.next()不可以传参，只能通过向本体generator函数中传参；最后一个yield无法返回值，只能通过本体generator函数return返回数值  
 \*/  
  
**var** num = 3;  
**function** \*a(){ //generator对象不可以使用箭头函数  
 **var** b = num+3;  
 **var** c = **yield** b; //这里的yield b是将上面b的数值传给对应的.next()，var c接收过来的数据时下方的.next()传递进来的参数  
 alert(c);//打印5  
}  
**var** A = a();  
console.log(A.next());//返回一个对象：{value: 6, done: false}  
console.log(A.next(5));//返回{value: undefined, done: true}

**Async和await（升级版的generator）**

或者说是官方版的runner