变量：

var （问题）

1、可以重复声明

2、无法限制修改

3、没有块级作用域

{ } if( ){ } for( ){ }

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

let和const的特点

1、不能重复声明

2、let是变量，可以修改；const是常量，不能修改

3、都有块级作用域

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

箭头函数

1、如果只有一个参数，（）可以省

2、如果只有一个return，{}可以省

※3、箭头函数的this指向：谁调用就指向谁，即指向调用者

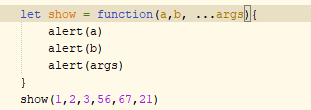
function(){} -------> () => {}

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

函数的参数

1、参数的展开和数组扩展

2、默认参数



参数扩展：

1、收集剩余的参数

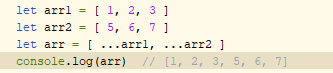
function( a, b, ...args ){ }

...args必须放在参数的最后一个，否则报错

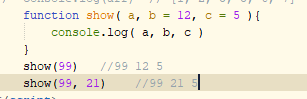
2.展开数组

let arr = [ 1, 2, 3 ]

...arr 等价于 1, 2, 3



默认参数：



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

解构赋值

1、左右两边结构必须一样

2、右边必须是个东西

3、声明和赋值不能分开（必须在一句话）

clipboard.png

clipboard.png

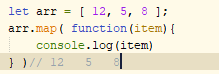
clipboard.png

clipboard.png

clipboard.png

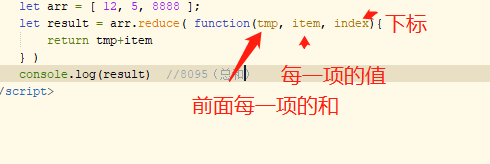
数组

map 映射 一个对一个

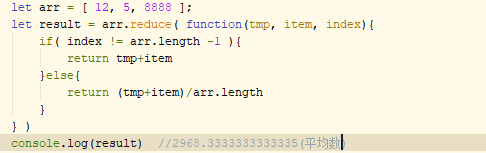


reduce 汇总 一堆出来一个

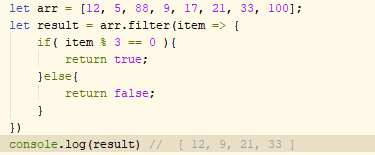
算个总数

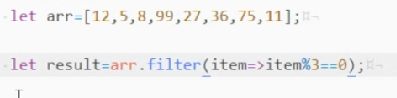


index从1开始



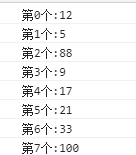
filter 过滤器





forEach 循环（迭代）

clipboard.png



async和await

async是说明这个function是异步的await这个关键字是阻塞（将函数挂起，等待返回结果），如果返回值是promise则等待promise的resolve或reject的结果，如果是函数返回的普通值直接往下执行

function call(){

console.log('call funtion');

return new Promise(resolve=>{

setTimeout(function(){

console.log('call funtion timeout');

resolve('dd');

},1000);

});

}

function normalFunction() {

console.log('normalFunction');

return 'data'

}

async function asynctest() {

console.log('start');

await call();

await normalFunction();

await new Promise(resolve=>{ console.log('wait result'); resolve()});

console.log('end');

}

asynctest();

执行结果：

start
call funtion
call funtion timeout
normalFunction
wait result
end

**yield使用**

函数执行每次都执行到yield这里返回一个yield阻塞的值{ value: Promise { }, done: false }，这个时候就可以处理值了，然后如果done为true时表示这个函数已经执行完成了

function call(){

console.log('call funtion');

return new Promise(resolve=>{

setTimeout(function(){

console.log('call funtion timeout');

resolve('dd');

},1000);

});

}

function normalFunction() {

console.log('normalFunction');

return 'data'

}

function\* yieldFunc() {

console.log('start');

yield call();

yield normalFunction();

console.log('end');

}

let yieldData=yieldFunc();

let firstData=yieldData.next();

console.log(firstData);

firstData.value.then(function(data){

console.log(data);

});

yieldData.next();

console.log(yieldData);

执行结果：

start
call funtion
{ value: Promise { <pending> }, done: false }
normalFunction
{}
call funtion timeout
dd