# 解构赋值

## 扩展运算符

//…代表将数据扩展到另一个数组中，在作为参数时必须放置到最后一个参数

let a = [1,2,3,4];

let b = [5,6,7,8];

b = [...a,...b];

console.log(b);//输出结果 [1, 2, 3, 4, 5, 6, 7, 8]

## 别名赋值

let {name:n,age:g,sex:s} = {name:'张三',age:17,sex:'男'};//通常用于对象

console.log(n,g,s);//打印结果：张三 17 男

## 默认值赋值

let [a,b,c='暂无数据'] = ['aa','bb'];

console.log(a,b,c);//打印结果：aa bb 暂无数据

# 时间

|  |  |  |
| --- | --- | --- |
|  | 解释 | 备注 |
| new Date() | 定义时间 | 括号中无内容默认取系统时间 |
| .getFullYear() | 获取当前年 |  |
| .getMonth() | 获取当前月 | 月份需加一天，系统月份从0开始算的 |
| .getDate() | 获取当前日期（1-31） |  |
| .getDay() | 获取当前日是星期几 |  |
| .getHours() | 获取当前小时（24小时制） |  |
| .getMinutes() | 获取当前分钟 |  |
| .getSeconds() | 获取当前秒 |  |
| .getMilliseconds() | 获取当前毫秒（1s = 1000ms） |  |
| .setFullYear(year) | 设置当前年 |  |
| .setMonth() | 设置当前月 | 范围0-11 |
| .setDate() | 设置当前日 | 范围0-31或0-30 |
| .setHours() | 设置当前小时 | 范围0-23 |
| .setMinutes() | 设置当前分钟 | 范围0-59 |
| .setSeconds() | 设置当前秒 | 范围0-59 |
| <、>、==、!= | 比较日期 |  |
| Js目前没有提供时间格式化输出的方法，需自己写 | | |

# 数组

## 定义

1.let arr = new Array('aa','bb','cc','dd');

2.let arr = ['aa','bb','cc','dd'];

## pop/push/shift/unshift/delete

pop():弹出最后一个并返回被弹出的值

push(obj):尾部压入

shift():弹出首部一个并返回被弹出的值

unshift(obj):头部压入

delete arr[index]:清除数组中该元素的值，但并不删除该元素，返回Boolean值，成功为true，失败false。

## 数组 替换/删除/增加 元素

//第一个参数：操作位置，

第二个参数：操作数量，

第三个~第n个参数：增加的值。

如果后面不加插入的参数，该的意义是从数组某个地方开始向后删除指定数量的元素。

该会操作原数组并返回被截取的数组。

let fruits = ["Banana", "Orange", "Apple", "Mango"];

let fruit = fruits**.splice(**2, 2, "Lemon", "Kiwi"**)**;

console.log(fruits);// ["Banana", "Orange", "Lemon", "Kiwi"]

//切割字符串slice(start,end);也可以达到相同的删除功效,并返回被切下来的数组

## 数组转字符串

//join可见数组转字符串

let arr = new Array(1,2,3,4,5,6,7,8,9,0);

let str = arr.join('-');//连接符

console.log(str);//结果：1-2-3-4-5-6-7-8-9-0

## 数组连接

//concat()连接多个数组为一个数组

let str1 = [1,2,3],str2 = [4,5],str3 =[6,7,8],str4=[9,0]

str1 = str1.concat(str2,str3,str4);

console.log(str1);

## 排序【不推荐用于数字排序】

//对数组升序排序，这种排序仅可排按字符排序

let num = [0,7,2,5,4,6,8,9,6,1,3];

console.log(num**.sort()**);//结果[0, 1, 2, 3, 4, 5, 6, 6, 7, 8, 9]

## 循环

//对数组循环，并执行指定方法；实例用的是箭头,item当前项，index：下标，arr：被循环的数组

let arr = [11,22,33,44,55];

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

console.log(item,index,arr)//打印结果：11，22，33，44，55

});

## 循环映射

/\*对数组不但循环而且进行相应的操作，并且需要返回值的的情况下使用，如将数组中所有11的倍数除以11并返回\*/

let arr = [11,1,2,22,33,44,55];

let newarr = arr**.map**((value) => {

if ( value % 11==0) {

return value / 11;

}else{

return value;

}

});

console.log(newarr,arr);//打印结果：[1, 1, 2, 2, 3, 4, 5]

## 循环过滤

/\*回调返true则留下元素，返false去除元素。注：回调仅支持返Boolean值，结果为数组\*/

let num = [1,2,3,4,5,6,7,8,9,10,11,12,13];

let newnum = num**.filter**((item,index,arry)=>{

if ( item % 2 == 0 ) {

return true;

} else {

return false;

}

});

console.log(newnum);//打印结果：[2, 4, 6, 8, 10, 12]

## 循环查找

//数组中是否有需查找的元素或计算结果是否符合需求，返回Boolean值，结果是Boolean值

let num = [1,2,3,4,5,6,7,8,9,10,11,12,13];

let num2 = num**.some**((item,index,arry)=>{

return item == 121;

})

console.log(num2);//打印结果：false

## 查找

//查找数组是第一个符合条件的数组成员，返回数据值，若没有则返回undefined

let num = [1,2,3,4,5,16,7,8,9,10];

let res = num**.find**(item => {

return item > 6;

});

console.log(res);//打印结果：16

//查找数组是第一个符合条件的数组成员位置，返回位置值，若没有返回-1

res = num**.findIndex**(item => {

return item > 6;

});

console.log(res);//打印结果：5

//查找数组中是否存在符合条件的数组成员，返回Boolean值

res = num.includes(16);

console.log(res);//打印结果：true

## 查询

//indexOf():从头开始查询数组中存在的某个值的下标，lastIndexOf():从尾部查询第一个下标

let arr = new Array(1,2,3,4,5,6,7,8,9)

let res = arr.indexOf(5)

console.log(res)

## 循环检查

//every检查数组中每一项是否符合规则，返回Boolean值，全部符合规范则返回true。

Some检查数组中每一项是否符合规则，任意一个符合则返回true，否则返回false

let num = [2,4,6,8,10];

let num3 = num.**every**((item,index,arry)=>{

return item%2 == 0;

});

console.log(num3);//返回结果：true

num3 = num.**some**((item,index,arry)=> item>10)

console.log(num3);//返回结果：false

## 循环运算

/\*从左往右将数组中每一个值纳入计算，最终返回计算总值；第一个参数值暂存当前运算结果，per不需要定义，只是作为形参作为中间变量，最后返回结果\*/

let num = [2,4,6,8,10];

let num4 = num.**reduce**((**pre**,cur,index,array)=>{

return **pre**\*cur;

})

console.log("乘：",num4);//打印结果：乘： 3840

/\*从右往左将数组中每一个值纳入计算，最终返回计算总值；第一个参数值暂存当前运算结果\*/

let num = [1,2,3];

let num4 = num.**reduceRight**((pre,cur,index,array)=>{

return pre\*\*cur;

})

console.log("乘：",num4);//打印结果：乘： 9

## ES6 for…of…循环

//循环遍历数组，数组某一项.entries(),序号.keys()，建议 of 对象即可。

let num = [1,2,3];

for (const iterator of num) {

console.log(iterator);//打印结果：1，2，3

}

/\*循环便利对象时可以结合业务逻辑使用**Object.entries**(…)、**Object.keys**(…)、**Object.values**(…)分别对当前项、key、value循环取值\*/

## Array.from()

/\*将字符串，对象【具有length属性】转换为数组；字符串拆分是按照每一个字母占用一个数组元素进行拆分的\*/

let json = {

0:'orange',

1:'apple',

2:'peach',

3:'banana',

4 :'watermalen',

length:5

}

let str = "abcdefghijklmn";

console.log(Array.from(json),Array.from(str));//打印结果：["orange", "apple", "peach", "banana", "watermalen"] (14) ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n"]

## Array.of()

/\*将传入的数据转换成数组，\*/

let str = "abcdefghijklmn";

console.log(Array.of(...str));//打印结果：["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n"]

# 数字

## 转字符型

num.**toString()**

## 字符串转整型

**parseInt("**10.33**")**;//返回10，**此方法不支持四舍五入**

## 字符串转浮点型

**parseFloat("**10.33**")**;// 返回 10.33

## 四舍五入num.toFixed(精度)

//toFixed() 支持四舍五入，返回**字符串**值，它包含了指定位数小数的数字,最后需Number()转为数字型

let p1 = 3.1415926;

let p2 = **Number(**p1**.toFixed(3))** ;

console.log(p2,typeof p2);// 3.142 "number"

|  |  |  |
| --- | --- | --- |
| Math.trunc(str/num) | 返回整数 |  |
| Math.sign(str/num) | 判断正负数，正数返回1，负数返回-1 |  |
|  |  |  |

# 对象

## 定义对象

//1.通过new Object()来定义；2.通过普通格式{obj:val,obj2:val2……}定义

let bill = new Object();

let bill2 = {firstName: 'Bill', lastName: 'Gate'}

## 访问对象属性

//1.通过 . 访问，2.通过[]访问

bill.age或bill['age']

//若对象有严格要求：做get

let bill = {

firstName:'Bill',

lastName:'',

**get** LastName(){

if (this.lastName === '') {

return 'lastName未定义'

}

return this.lastName;

}

}

//也可这样定义：这种定义可以复用Person类，定义为 let person = new Person();

class Person {

constructor(name = '', sex = false, age = 0) {

this.name = name;

this.sex = sex;

this.age = age;

}

get fullInfo(){

return `${this.name}-${this.sex}-${this.age}`

}

/\*\*

\* 修改名称

\*/

set setName(name=''){

this.name = name;

}

}

console.log(bill.LastName);

## 设置对象属性

//1.通过 . 设置，2.通过[]设置

bill.age=43 或bill['age']=43

//对对象有严格要求：做get

let bill = {

firstName:'Bill',

lastName:'',

**set** LastName(lastName){

this.lastName = lastName;

}

}

bill.LastName = 'Gate';

console.log(bill.lastName);

## 添加对象属性

//如访问对象属性相同，不过需添加值，如：bill.age = 43或bill['age'] = 43

## 删除对象属性

//如：delete bill.age或delete bill['age']，返回Boolean值

## Call()/apply()

//可以调用其他对象的属性作为己身属性的方法;

//call(被调用对象,...[数组]);apply(被调用的对象,[数组])

class Person {

constructor(name = '', sex = false, age = 0) {

this.name = name;

this.sex = sex;

this.age = age;

}

buyCar(car){

this.cartype = car.cartype

this.price = car.price

}

getCar(){

return this.cartype+'-'+this.price//没有调用buyCar(car)方法之前，cartype和price均不存在，这个时候调用call，将Car对象中的属性拿过来使用

}

}

class Car {

constructor(cartype='',price=0){

this.cartype = cartype

this.price = price

}

}

let p = new Person('舒黎',true,22);

let car1 = new Car('吉利缤瑞',98000);

let car2 = new Car('大众朗逸',149000);

console.log(p)

console.log(p.getCar.call(car2));

console.log(p);

## 原型prototype

原型为prototype，类中定义的公用内存；可存储方法和数据，所有该类定义的对象均使用该类中的这块公用内存中的数据。

对象1

对象2

对象3

类class

prototype:school【这公共内存】

age、sex、name

定义原型

class **Person** {

constructor(name = '', age = 0, sex = '男') {

this.name = name;

this.age = age;

this.sex = sex;

**Person.prototype.sex = 0;//定义原型**

}

}

或者 let per = new Peron();per.prototype.school = '华夏大学'

//如果原型和属性中的key值一样就会用属性中的key值，删除原型不可以引用对象删除，必须引用类删除。例如：delete Person.prototype.sex【返回Boolean值】

ES6中类允许被继承，使用关键字extends，但要注意在constructor中必须super(…)父类的方法

class Student **extends Person**{

**constructor**({name='Jack',age=10,sex='男',grade = 0, school = ``}) {

**super(name,age,sex)**

this.grade = grade;//年级

this.school = school;//学校

}

}//删除子类的prototype不会影响父类的prototype

|  |  |  |
| --- | --- | --- |
| 重置原型 | Object.setPrototypeOf(obj, {原型对象}) |  |
| 获取原型 | Object.getPrototypeOf(obj) |  |
| 获取对象值 | Object.values(obj) | 返回数组 |
| 获取对象键 | Object.keys(obj) | 返回数组 |
| 获取键值对 | Object.entries(obj) | 返回数组 |

# 字符串

## 字符串转数组

//str.split(''), ''表示按字母分割，若以其他值分割需在''中写分割值

let str = `abcdefg`;

console.log(str.split(''));

## 字符串模板

//ES6提供的字符串模板，支持无转义字符进行换行，使用${…}包裹其他值

let n = '张三',g = 18;

let introduce = `大家好，我叫**${**n**}**，我今年**${**g**}**岁了`;

console.log(introduce);

## 字符串长度

//返回number类型的字符串长度

let str = `abcdefghijklmnopqrstuvwxyz`

let len = Str**.length**;//打印值： 26

## 转大写，转小写

toUpperCase()

toLowerCase()

## 字符串查找

### 查找位置

//查找字符串第一次出现目标字符串的位置，未找到文本则返回-1

let introduce = `大家好，我叫${n}，我今年${g}岁了`;

console.log(introduce,introduce**.search**('我叫'));//返回值4；老方法有**indexOf(str)**，建议使用新方法

//查找字符串最后一次出现目标字符串的位置，未找到文本则返回-1

let introduce = `大家好，我叫张三，我今年19岁了`;

console.log(introduce,introduce**.** **lastIndexOf** ('我'));//返回值9

### 是否存在

//查找字符串是否存在于目标字符串中是否存在，存在为true，不存在为false

let introduce = `大家好，我叫${n}，我今年${g}岁了`;

console.log(introduce,introduce**.includes**('大家好'));//返回值true false

## 截取字符串

### 按位置截取

//slice(截取开始位置，截取结束位置)，slice(开始位置)：从开始位置一直截取到末尾，slice(负数，负数)，从尾部开始截取。也可以用subString()方法，唯一的区别是subString()不接受负索引。或subStr(),不同之处在于，第二个参数为被提取字符串的长度。【注意：该不改变原字符串，仅返回结果】

let obj = `abcdefghijklmnopqrstuvwxyz`;

let str = obj.slice(3,13);

console.log(obj,str);//打印结果abcdefghijklmnopqrstuvwxyz defghijklm

## 清空首、尾空格

//str.trim()将首尾空格清空

//str.trimStart()清空首部空格

//str.trimEnd()清空尾部空格

## 连接两个字符串

//将多个字符串连接到目标字符串中str.concat(str1,str2,……);

let strArry = ['aaaaa','bbbbb','ccccc','ddddd']

let str = ''.concat(...strArry);

console.log(str)// aaaaabbbbbcccccddddd

## 替换字符串

//替换第一个符合条件的字符串

let s = 'micorsoft,micorsoft,micorsoft'

s = s.replace('micorsoft','apple')

console.log(s);//打印结果apple,micorsoft,micorsoft

## 检测字符串开头

//验证字符串是否以xxx开头

let introduce = `大家好，我叫${n}，我今年${g}岁了`;

console.log(introduce**.startsWith(**'家好'));//返回值true false

## 检测字符串结尾

//验证字符串是否以xxx结尾

let introduce = '大家好，我叫张三，我今年18岁了';

console.log(introduce**.endsWith**('岁了'));//返回值true false

## 字符串重复

//将目标字符串重复n次再赋值回去【n值必须>1】

let i = 'good';

console.log(i**.repeat**(5));//打印结果：goodgoodgoodgoodgood

## 字符串头部填充

/\*将截取字符串填入目标字符串头部，padStart(length,str),第一个参数是结果总长度，如果结果总长度不等于i.length+j.length，i的数据不会改变，只会根据总长度-i.length的长度截取j，如果长度小于i则不会截取，也不会改变i。\*/

let i = 'efghijklm',j = 'abcd';

console.log(i**.padStart**(i.length+j.length, j));//打印结果**abcd**efghijklm

## 字符串尾部填充

/\*将截取字符串填入目标字符串尾部，padStart(length,str),第一个参数是结果总长度，如果结果总长度不等于i.length+j.length，i的数据不会改变，只会根据总长度-i.length的长度截取j，如果长度小于i则不会截取，也不会改变i。\*/

let i = 'abcd',j = 'efghijklm';

console.log(i**.padEnd**(i.length+j.length,j));//打印结果abcd**efghijklm**

# JOSN

string转对象

JSON.parse(str,fun)/JSON.parse(str)

str:为必填项，待转换的string字符串，

fun:选填项，对每组数据做出的改变。

JSON.parse(obj,(key,value)=>{

     if (key == "name")

          return value.toUpperCase();

     else

return value;

     });

对象转String

JSON.stringify(obj)

## 参数允许设定默认值

//默认值用等号连接，注意：第一个参数必须传值，不然报错！故而第一个参数没必要设定默认值

function show( a = 10, b = 10 ){

console.log( a/b );

}

show(100,10);//结果10

show();//结果1

show(20,);//结果2

//show(,5);//错误-第一个参数必须传值

/\*对象做参数仍然通用\*/

function show2( { a = 'Game' , b = 'over' }={} ){

console.log( a,b );

}

show2();//输出Game over

show2({a:'Junk',});//输出Junk over

show2({a:'Junk',b:'food'});//输出Junk food

## 参数数量

//使用.length属性得到number型的参数的数量

function **show**( a = 10, b = 10 ){

console.log( a/b );

}

console.log(**show.length**);

## 箭头

箭头格式：let 名=(参数)=>{内容}

注意：

1. **this问题**，定义所在的对象，不是在运行时所在的对象
2. 箭头里面没有arguments，用’…’
3. 箭头不能当构造

## 比较

//比较两个值的大小，仅比较基础数据如string，number，Boolean，不能比较数组和对象

let obj = {name:'张三',age:18},obj2 = {name:'张三',age:18};

let arr = [1,2,3,4,5,16,7,8,9,10],arr2 =[1,2,3,4,5,16,7,8,9,10];

let str = '张三',str2 = '张三';

let num = 10,num2 = 10;

console.log(**Object.is**(obj,obj2),obj,obj2);//打印结果：false

拼接对象

//将不同的对象或数组拼接到第一个对象或数组中，若存在相同属性，则前者被覆盖，取后者值

let obj = {age:17},obj2 = {name:'李四',age:18};

let json = **Object.assign**({},obj,obj2,{sex:"女"});

console.log(json);//打印结果：{age: 18, name: "李四", sex: "女"}

## 暂时不清楚什么情况下用promise

let a = 11;

let promise = new Promise((resovlve,reject)=>{

if(a == 10){

resovlve('success');

}else{

reject('faild');

}

});

promise.then(res=>{ console.log(res);},err=>{console.log(err);})

# Set

一般用于存储数字，字符串，Boolean值的数组；

|  |  |  |
| --- | --- | --- |
| 定义 | new Set()/new Set([array]) |  |
| 添加 | .add(obj) | 将对象添加进去，返回set本身 |
| 删除 | .delete(obj) | 删除对象，返回Boolean |
| 清空 | .clear() | 清空set |
| 检查 | .has(obj) | 查找set是否包含某项，返回Boolean |
| 长度 | .size | 返回set元素个数 |

# Map

用于相当于键值对，键和值不仅允许存Boolean、String、Integer而且允许存储Object。遍历途径有keys()、values()、entries()

|  |  |  |
| --- | --- | --- |
| 定义 | new Map() |  |
| 添加 | .set(obj) | 将对象添加进去，返回set本身 |
| 删除 | .delete(key) | 删除键值对，返回Boolean |
| 检查 | .has(key) | 检查set是否包含某项，返回Boolean |
| 获取 | .get(key) | 依据键获取值 |
| 长度 | .size | 返回键值对个数 |

存储set(keyObj,valueObj);拿取get(key);检验是否存在has(key);删除delete(key)例子如下

class Person{

constructor({name = '',sex = '',age = ''}={}){

this.name = name;

this.sex = sex;

this.age = age;

}

}

class Student extends Person{

constructor({name,sex,age,school = '',grade= ''}={}){

super({name,sex,age});

this.school = school;

this.grade = grade;

}

}

let stu1 = new Student({name:'张三',sex:'男',age:19});

let stu2 = new Student({name:'李四',sex:'女',age:20});

let map = new Map();

map**.set("stu1",stu1);**

map**.set("stu2",stu2)**

console.info(**map.get("stu1")**)

# WeakMap

WeakMap与Map类似，都用于生成键值对的集合；不过WeakMap只接受对象作为键名，不接受其它类型的值做为键名。

# ES6模块化

建议此章节在CLI中实验

## 定义模块

//在普通js文件中定义

const a = '10',b = 12,c = true;

//定义了数据，还不能使用，需要导出；使用export关键字

export {

a as aaaa,//**as**是命名别名的关键字

b as bbb,

c as cc,

}

## 使用模块

//导入模块：**import 模块名 from 被引入模块的相对路径**

import {aaaa,bbb,cc} from './assets/test'//导出需要的Js中的数据

import \* as tst from './assets/test'//导出test.js并以别名tst引用

花括号{}与export default

如果JavaScript文件中不是全部数据都需要引出，那么可以使用花括号，与此对应的，在引用的时候，也同样需要{}进行引入，如果是全部引出，建议使用export default，对应的在引入的时候就不需要使用花括号了，而全部导出的js数据格式请遵循：

let js\_name = {

introduce : 'I am js import file',

age:13,

bool:true,

arr:[1,2,3,4,5,6,7,8,9],

fun:()=>{

console.log("I am running here.");

}

}

export default js\_name;

//js引入

import test from './assets/test'//export default模式不能使用别名

动态引入:这种方式的引入支持根据需求写在if…else…中根据需求引入

import(js路径).then(res=>{ });