正课：

1. ES6

1. ES6

1. for of

什么是: 专门代替普通for循环，来遍历一个索引数组

对比: 遍历索引数组：3种:

1. for循环

for(var i=0;i<arr.length;i++){

var elem=arr[i]

... ...

}

最灵活:

1. 可通过i从几开始，到几结束来控制i遍历的范围。

2. 可通过i++， i+=2, i+=5等控制遍历的步调！

3. 可通过i=10; i>0; i-- 从后往前遍历

2. arr.forEach()

arr.forEach((elem,i,arr)=>{

})

局限: 不便于灵活控制遍历的范围和步调和顺序

局限: 类数组对象不能用forEach，因为类数组对象不是数组家孩子！

3. for of

for(var elem of arr){

//of会自动获取arr中每个元素值

}

局限: 无法获得当前遍历的位置.

何时: 只关心元素值，不关心位置时

局限: 是ES6的功能，不是所有浏览器，所有数组和类数组对象都支持。

强调: for of只能用于遍历数字下标的索引数组和类数组对象

for in 只能遍历关联数组和对象！

2. 参数增强:

1. 参数默认值（default）:

什么是: 即使没有传入实参值，形参值也有一个备胎值可以使用！

何时: 只要希望一个形参，即使将来用户没有传入实参，也有默认值可以使用，不为undefined时

如何:

1. 定义函数时:

function 函数名(形参,...,形参=默认值)

强调:

1. 只有最后一个形参，才能有默认值。所以，定义函数时，通常都会把不确定的形参，放在形参列表的末尾。

2. 只能解决最后一个形参不确定的情况

2. 调用函数时: 如果没有传入最后一个实参值，则自动使用默认值代替。

如果不支持ES6: 形参=形参||默认值。

复习短路

2. 剩余参数(rest):

什么是: 专门代替arguments的语法，收集不确定个数的实参值

为什么: arguments有3大缺点:

1. 类数组对象，不是纯正的数组，不能使用数组家的函数

2. arguments只能全部获得实参值列表。不能有选择的获得部分实参值。

3. 单词太长！

何时: 今后只要使用arguments的地方，都可以用...rest。

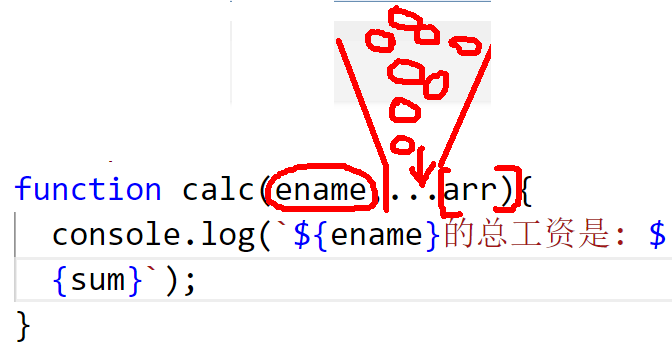
如何:

定义函数时:

function 函数名(其它形参,...数组名){

//...起到收集剩余参数的作用，将收集来的剩余的参数，都放入数组中，保存在右边的变量中。

//收集后的数组，不包含之前其它形参的值。且是一个纯正的数组，数组家的函数都可以使用！

}  


强调: ...arr必须放在形参列表结尾，因为将来传参时，只有结尾的实参才是不确定的。实参列表中间不可能出现不确定实参个数的情况。

3. 打散数组参数(spread):

问题: .apply()也能打散数组参数，再传给要调用的函数。但是，apply()的本职工作是修改this！然后顺便帮忙打散数组。如果今后我们只想单纯的打散数组，跟this无关时。apply()就不好用了

比如: Math.max(值1, 值2,...)

Math.min(值1, 值2,...)

但是，程序中通常保存多个值，都是保存在数组中的。而Math.max()和Math.min()都不支持查找数组中的最大值！

解决:

不好的解决: 请.apply()

比如: Math.max.apply( ? ,arr)

但是?处写啥？因为这个需求和this无关！所以?处写什么都对。因为apply的第一个参数，是专门修改this的。但是，最好?处写null，表示不替换this！

比如: Math.max.apply(null,arr)

好的解决办法: 用...

比如: Math.max(...arr)

其中, ...先将数组打散为单个值，再传入前边的函数。好处是，...和this无关，只是专门单纯的打散数组。

总结: 今后只要单纯想打散数组为单个值时，就用...。

vs 剩余参数rest:

问题： 都是...，何时是收集？何时是打散？

答: 定义函数时，形参列表中的...是收集不确定个数的实参值的意思

调用函数时，实参值列表中的...是打散数组实参为多个实参值，分别传入的意思。

3. 解构（destruct）:

什么是: 将一个大的对象中集中存储的成员，解构出来便于单独使用。

为什么: 因为将来从服务器端返回的对象可能很复杂，复杂:

比如: 学子商城详情页:

var output={

product:{ pid, pname, details, price,...},

specs:[

{lid:1, spec:i5 128g},

{lid:2, spec:i7 128g},

... ...

],

pics:[

{pic\_id:1, sm:"img/1\_sm.png", md:"img/1\_md.png", lg:"img/1\_lg.png"},

{pic\_id:1, sm:"img/1\_sm.png", md:"img/1\_md.png", lg:"img/1\_lg.png"},

... ...

]

}

何时: 今后凡是获得结构嵌套非常复杂的对象，都要先解构成多个小的成员，再分别单独使用。

比如: var {product, specs, pics}=output;

product

specs

pics

如何: 3种:

1. 数组解构:

什么是: 将大的数组中的元素，结构出来你，单独使用。

何时: 只要服务端返回的是一个巨大的数组，而我们只想单独使用数组中的某几个元素时

如何: 下标对下标

var [变量1, 变量2,...]=数组

1. 同时声明多个变量

2. 去数组中找到和当前变量所在下标一致的元素值，保存到对应位置的变量中

结果: 从此变量1和变量2就可代表数组中的单个元素，单独使用。

2. 对象解构:

什么是: 将结构复杂的较大的对象中的个别成员，单独提取出来。单独使用。

何时: 当对象结构很复杂，而我们希望分别使用对象中的成员时

如何: 属性名对属性名

var { 属性名1: 变量1, 属性名2: 变量2,...}=对象;

原理:

1. 同时声明多个变量

2. 去对象中查找相同属性名的属性值，赋值给对应位置的变量

结果: 每个变量中都获得了自己对应属性名的属性值。将来每个变量都可以单独使用！

3. 参数解构：

什么是: 所有的参数，都放在对象中传入函数，然后在函数中，再将对象的成员结构出来，单独使用。

何时: 当多个形参不确定将来有没有值，且传入的实参值，必须和形参变量一一对应！

问题: 参数默认值，...rest，arguments都不能很容易的解决这个问题。只能用参数解构:

如何:

1. 定义函数时:

将形参列表定义在一个对象结构中

比如: function order({

zx,tang,td,rc,sc

/\*配对 形参变量名

zx:zx,

tang:tang,

td:td,

rc:rc,

sc:sc

\*/

}){

zx=zx||"";

tang=tang||"";

td=td||"";

rc=rc||"";

sc=sc||"";

console.log(`

主食:${zx},

汤:${tang},

甜点:${td},

肉菜:${rc},

素菜:${sc}

`);

}

2. 调用函数时:

将实参列表也必须定义在一个对象中，整体传入！

优点: 无论哪个实参没有，函数执行都不会出错。大不了，如果解构不成功，某个实参值获得的是undefined。也不会报错。

order({

zx:"米饭",

tang:"嘎达汤",

td:"提拉米苏",

rc:"水煮肉片",

sc:"蒜蓉油麦"

});

order({

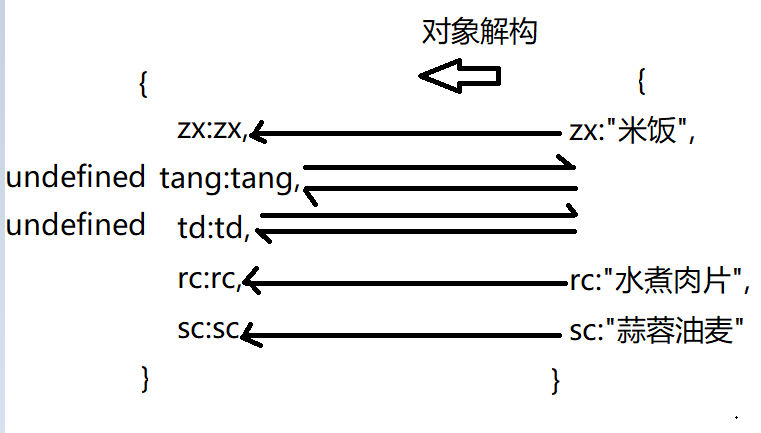
zx:"米饭",

rc:"水煮肉片",

sc:"蒜蓉油麦"

});

原理: 参数解构，其实就是对象解构在函数传参时的应用而已：



4. 面向对象:

1. 对于单个对象: 2种简写:

1. 如果一个属性的值来自于一个变量，且变量名刚好和属性名一致，则可只写一个名字，两用！

建议:今后在系统中，指一个事物时，都要用统一的名字，可以享受福利！

2. 因为对象中的方法不能用箭头函数简写（因为不希望方法内的this和外边的this一致），又想去掉function，可去掉":function"

比如:

var sname="Li Lei";

var sage=11;

var lilei={

sname, //sname:sname,

sage, //sage:sage

intr(){//:function(){

console.log(`I'm ${this.sname}, I'm ${this.sage}`)

}

}

2. class:

什么是: 对一个类型中构造函数和原型对象方法的集中定义的程序结构

为什么: 旧的js中，要定义一个类型，构造函数和原型对象方法是分着写的！不符合封装的要求

何时: 今后只要创建一种新的类型，不要先创建构造函数，也不要分着写原型对象方法，而是先创建一个class，然后将类型的所有成员集中定义在class中

如何: 3句话:

1. 定义class:

//1. 用class{}包裹构造函数和原型对象方法

class Student{

//2. 构造函数名提升为整个类型class名，所有构造函数统一更名为constructor

constructor(sname,sage){

this.sname=sname;

this.sage=sage;

}

//3. 所有原型对象方法可省略一切前缀和=function。定义在class中的普通函数，默认就是保存在原型对象中的。

intr(){

console.log(`我是 ${this.sname}, 我今年 ${this.sage} 岁`)

}

}

2. 使用class: 和使用构造函数完全一样！生成对象的结构和以前也是完全一样的——新瓶装旧酒!

3. class间的继承: 到ng和react框架再复习

问题: 两种类型间有部分相同的属性结构和方法定义，绝不能在程序中写两遍！不便于维护！

解决:

1. 创建一个父class，集中存储相同部分的属性结构和方法定义！

2. 创建子类型class，用extends关键词继承父类型。

强调: 子类型的构造函数中，要使用super()调用一次父类型的构造函数

强调: super()必须写在自己扩展的属性之前！

5. Promise:

什么是: 专门实现多个异步函数，必须先后顺序执行的技术。

为什么: 回调地狱 callback hell

何时: 今后只要多个异步函数，必须先后执行时，必须用Promise技术:

如何:

英文单词:

1. default: 默认

2. rest: 剩余

3. spread: 散播

4. destruct: 解构

5. extends: 扩展

6. inherits: 继承

7. promise: 承诺