正课:

1. ES6:

1. ES6:

解构: 将一个对象中的属性或数组中的元素，拆解给多个变量

何时: 简化批量赋值!

如何: 3种:

1. 数组解构: 将数组中的元素值，拆解后，赋值给多个变量

/\*var\*/ [变量1,变量2,...]=[元素1,元素2,...]

执行结果:

变量1=元素1;

变量2=元素2;

匹配: 下标对下标

2. 对象解构: 将对象中的属性值，拆解后，赋值给多个变量

{属性1:变量1,属性2:变量2,...}={ 属性1:值1,属性2:值2,...}

3. 参数结构:

定义函数时:

function fun(...,[参数1,参数2,...])

{属性1:变量1,属性2:变量2,...}

调用函数时:

fun(...,arr)

obj

for of: 简化for循环

for(var elem of arr)

其中of 会依次取出arr中每个元素值，自动保存到变量elem中

缺点: 1. 只能获得元素值，无法获得当前位置i

2. 只能从头到尾逐个完整遍历，不能有选择的遍历

3. 如果数组中是原始类型的值，则elem是按值传递得到的副本。修改副本，无法影响原数组元素

强调: for...of，只能遍历数字下标的数组或类数组对象

for...in，专门遍历自定义下标的关联数组或对象

class: 对OOP

1. class定义:

class 类型名{

constructor(属性参数列表){

this.属性名=属性参数;

}

方法名(){ ... this.属性名 ... }

... ...  
 }

如何: 3步:

1. 用class封装构造函数和原型对象方法定义

2. 构造函数名提升为class名,构造函数更名为固定的关键词constructor

3. 直接写在class中的方法，默认都保存在原型对象中

2. 继承:

1. setPrototypeOf -> extends father

2. 借用构造函数: super(参数值列表)

不再需要.call(this,...)

强调: class Flyer已经不允许直接调用

3. 访问器属性: 验证, 虚拟/计算属性

class内:

get 访问器属性(){}

set 访问器属性(val){ ... ... }

4. 静态方法:

static 方法名(){}

\*\*\*\*\*Promise:

什么是: 对传统回调函数的规范写法

为什么: callback hell

何时: 只要定义带回调函数参数的函数时，都要用promise

如何:

1. 定义接收回调函数的主函数

function fun(){

return new Promies(function(callback){

...

callback();

...

})

}

1. 去掉参数中的callback

2. 用return new Promise(function(){

原函数逻辑

})

3. 将callback写在Promise对象的参数函数中的参数列表中

2. 调用主函数执行，并传入下一个要执行的回调函数

fun()//return Promise

.then(callback)

如何:

1. 主函数fun不再接收任何回调函数参数

2. 在主函数调用后用.then(callback)，将callback传给fun中的Promies对象的callback参数。

异常处理: .then() .catch(fun)

return new Promise(function(resolve,reject){

//函数体

//如果执行成功

//调用resolve()

//否则

//调用reject()

})

Promise.all

什么是: 等待多个任务都完成后，才开始后续操作

何时: 如果需要等待多个任务时

如何: Promise.all([

支持Promise的函数(),

... ... ,

]).then(function(){ 后续任务 })