正课：

1. ES6

1. ES6

1. 面向对象上的简化：

1. 对单个对象提供了2处简化:

1. 如果对象的属性值来自于对象外的变量，且变量名刚好和属性名相同。则不用写两遍相同的名字，只要写一遍即可。

2. 所有对象的方法，不再需要写":function"

强调: 对象中的方法去掉:function，就不等效于箭头函数。突出的特点就是this保持原样不变！

所以，去掉:function，只是单纯的简写，没有任何原理的改变！

2. class:

什么是: 集中定义一种类型的所有对象统一属性结构和方法的程序结构

为什么: 每一种类型: Array Student Date都有两部分组成: 构造函数+原型对象

构造函数: 负责定义所有子对象统一的属性结构，并且负责创建子对象

原型对象: 负责保存所有子对象共有的属性值和方法

但是，在ES5中，构造函数和原型对象是分开定义的。不符合"封装"的要求

何时: 今后只要希望创建一种自定义类型时，都用class

如何: 3件事

1. 用class{}包裹构造函数和原型对象方法

2. 构造函数名提升为class名，所有构造函数，从此统一更名为constructor

3. 所有放在class中的函数，不需要加类型名.prototype前缀，自动就是保存在构造函数的原型对象中。

如何使用class: 用法和从前的构造函数完全一样！原型对象的原理依然保持不变。

两种类型间的继承:

问题: 两种类型之间包含部分相同的属性结构和方法定义。

解决: 定义一个上层的父类型，集中保存两种子类型相同的部分

如何:

1. 定义抽象父类型:

父类型构造函数中集中定义多个子类型都有的相同的属性结构

父类型class中集中保存多个子类型共用的方法

2. 让子类型继承父类型:

class 子类型 extends 父类型{

}

其中: extends等效于Object.setPrototypeOf(子类型.prototype, 父类型.prototype)。

但是，继承父类型不但要继承父类型的方法，还要继承父类型构造函数中的属性结构。所以，需要在子类型构造函数中先调用父类型构造函数。请父类型构造函数帮忙创建相同部分的属性结构。然后子类型构造函数，再添加自有的个性化的属性结构。最后，创建的子对象的属性结构来自于父类型构造函数和子类型构造函数共同执行的结果。

问题: 父类型构造函数没有名字！

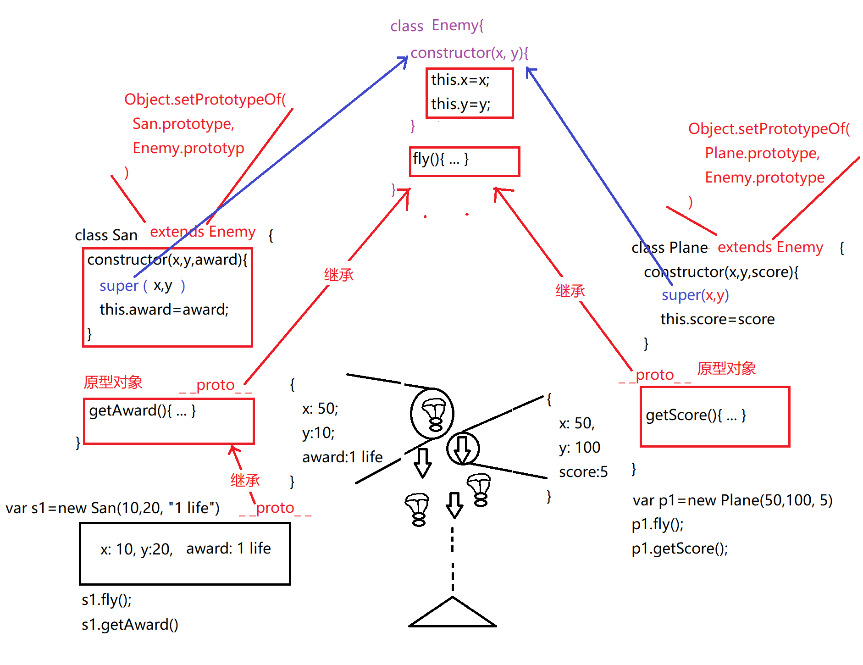
解决: 只要class 子类型 extends 父类型。就可以在子类型构造函数中调用super()关键词。

super是子类型中，专门指向父类型构造函数的方法。

强调:

1. 只要class 子类型 extends 父类型。则子类型构造函数中必须调用super()，否则会报错。

2. super()必须放在子类型构造函数开头，先调用。因为这样可以始终保证万一子类型构造函数中规定的属性和父类型构造函数中规定的属性重名时，子类型构造函数的属性可始终覆盖父类型构造函数中的属性。



2. Promise:

什么是: 专门保证多个异步函数，可以顺序执行的机制。而且还防止了回调地狱问题。

何时: 多个异步调用的函数，要求必须顺序执行！

为什么: 其实用回调函数，也可以实现多个异步函数，顺序执行。但是，使用回调函数，会有回调地狱问题！

问题1: 多个异步函数要求顺序执行

错误的解决: 仅按顺序调用

结果: 无法保证顺序执行

因为: 多个异步函数，相当于百米赛跑，每人各占一条跑道，互相之间不会等待。

传统的解决: 使用回调函数:

1. 在定义函数时，定义一个callback形参变量

在函数内部，最后一句话执行之后，自动调用callback()

function dong(callback){//异步!

console.log("东起跑...");

setTimeout(function(){//异步!

console.log("东到达终点!")

//等东的最后一句话执行完，自动调用提前托付的任务callback

callback();

},4000)

}

2. 在调用函数时，传入一个函数，函数中包含下一步要执行的操作。——提前托付

dong(

//callback

function(){

console.log("比赛结束!")

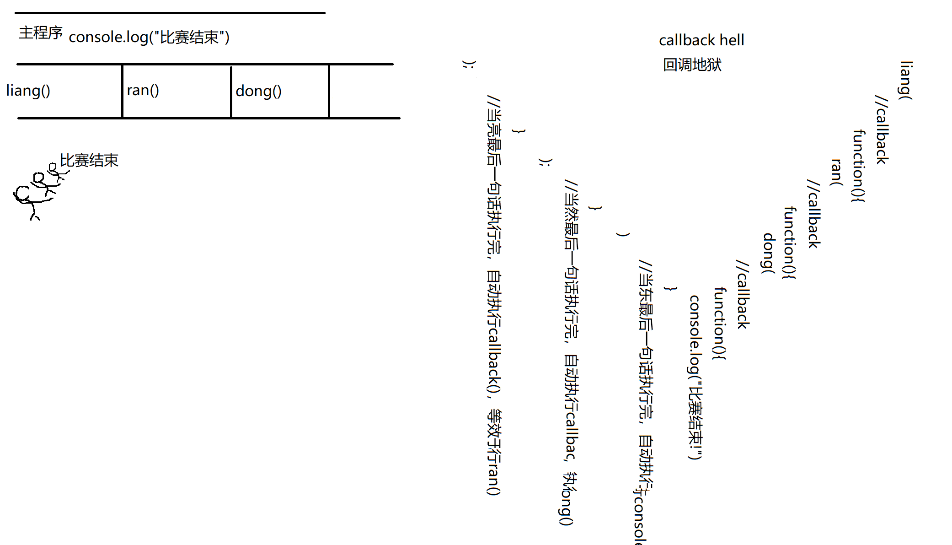
}

//当东最后一句话执行完，自动执行callback()，等效于执行console.log("比赛结束")

)

效果: 当前一个函数调用时，暂时不执行传入的小函数。只有前一个函数最后一句话执行完，才自动调用传入的小函数。一次保证小函数一定会在异步函数调用完才执行。

问题: 当连续执行的异步函数非常多时，就会形成回调地狱！



解决: ES6 Promise

前提: 不要用回调函数参数了！

因为回调函数嵌套传参的写法是造成回调地狱的根源

第一步: 在异步函数内部，用new Promise(function(door){ ... })来包裹原函数中所有代码。

其中door是用来打开通向下一项任务的大门的要是。

第二步: 在异步任务执行完最后一步之后，调用附赠的开关door()，开门，通知下一项任务可以开始执行。

问题: 不能在函数内写死.then(下一个函数)，应该让当前函数可以和之后任意函数自由组合。

第三步: 将整个new Promise()对象返回到函数外部，再用.then()接下一项任务函数

调用支持promise的函数：

前一个异步函数().then(下一个异步函数)

强调: .then()中"下一个异步函数"后不要加()，因为下一个异步函数不是立刻调用，而是在这里等待前一个函数开门。

信任: 前一个函数内调用door()后，就等于通知了.then()中的下一个函数可以开始执行。于是.then()就自动执行下一个函数。

多个异步函数顺序执行：

.then()能否接下一个.then()，取决于前一个.then()中的函数是否也支持promise。如果前一个.then()中的函数支持promise，则.then()可以继续.then()。如果前一个.then()中的函数不支持promise，则不能继续.then()

前后两个函数间传参:

2步:

1. 上一个函数中door(参数值)

2. 下一个函数定义时就要定义一个形参准备接

原理: 当上一个函数调用door(参数值)时，参数值，会顺着.then()交给.then()中的下一个函数的形参变量。在下一个中就可通过自己的形参变量获得上一步传下来的参数值。

局限: door()中只能传一个变量

如果必须穿多个值，则可以将多个值放在数组或对象中整体传入。

错误处理:

如何: 任何一个支持promise的函数中都有另一扇门。如果当前异步任务执行过程中发生错误，就可从另一扇门出来。一旦从报错这扇门出来后，后续.then()都不再执行。

其实new Promise()除了then外，还有另一个方法.catch()。凡是从出错的门出来的代码都进入.catch()中执行错误处理操作。

等待多个异步任务完成才执行:

Promise.all([

多个支持promise的函数调用(),

...,

...,

]).then(function(){ 后续操作...})

问题：如果每个异步任务都返回一个接力棒，则如何获得所有接力棒呢？

解决: .then(function(arr){ ... })

其中: arr数组中保存了Promise.all中所有异步函数通过door()返回的执行结果。

强调: arr中返回值存储的顺序和异步函数执行完成的顺序无关。只和调用的顺序有关！

Promise的问题: 并没有彻底消灭嵌套:

比如: .then(dong)

.then(function(){

console.log("比赛结束")

}).catch(function(err){

错误处理

})

解决: ES7: async await 可按照传统的同步指定的代码一样，编写异步代码顺序执行

只要多个异步任务需要顺序执行:

(async function(){

同步代码;

var 返回值=await 异步函数()

同步代码;

})();

其中: await可让整段匿名函数自调暂时挂起，等待当前异步函数执行完，在执行后续代码！

强调: es7的async和await仅仅简化的是promise函数调用的部分。而并没有简化Promise函数的定义。且，如果想用await，则异步函数必须定义为支持promise的样式。

错误处理: 如果await修饰的异步函数中调用了err()方法，打开了错误的门，则await会认为是程序错误。

应该用try{}catch(err){}来解决

强调: 只有async下的try catch才能捕获异步任务中的错误。没在async下的js基础中所学的try catch是不能捕获异步任务的！因为普通js基础中的try catch属于主程序，不会等待异步任务执行，就已经结束了。即使异步任务出错，try catch因为早就结束了，所以根本捕获不到。

为什么await配合try catch就可以捕获异步任务中的错误呢？

因为await可留住当前程序中的一切代码，等待异步函数执行完。try catch就有可能捕获到异步任务中的错误。

作业:

1. 使用class之间的继承实现Plane类型的定义