正课:

1. \*\*\*OOP

自定义继承:

2. \*\*\*ES5

1. \*\*\*OOP

自定义继承:

\*\*\*\*\*3. 两种类型间的继承:

问题: 两种类型间拥有部分相同的属性和方法定义

解决: 抽象一个父类型

1. 定义抽象父类型: 2部分

父类型构造函数: 保存相同的属性结构

父类型原型对象: 保存相同的方法定义

2. 让子类型继承抽象父类型: 2步:

让子类型原型对象继承父类型原型对象

在子类型构造函数中借用父类型构造函数

错误1: 直接调用父类型构造

Flyer(...); //this->默认window

this.fname=fname;

window.fname=fname;

原因: 一个函数,不用new,不用.调用，其中的this，默认指window

错误2: new Flyer(...) 创建一个新对象，另起炉灶，和原对象无关了

错误3: this.Flyer(...) 构造函数根本不在原型链上

正确: 抢!

Flyer.call(this,fname,speed)

=>this.Flyer(fname,speed)

call相当于用外部正确的this，代替了Flyer中原本错误的this！

2. \*\*\*ES5:

严格模式:

什么是: 比普通js运行机制要求更严格的模式

为什么: 普通js中有很多广受诟病的缺陷

何时: 今后所有项目的开发必须在严格模式下进行

如何启用严格模式: 2种:

1. 为整个js文件或script标签启用严格模式

在js文件或script标签的开头:

"use strict";

何时: 从0开始的新项目，都要整个js文件或script标签启用严格模式

2. 为单个函数启用严格模式

在函数内顶部: "use strict";

何时: 旧项目向严格模式迁移时

要求:

1. 禁止给未声明变量赋值!

解决: 给未声明的变量赋值，会自动创建在全局，造成全局污染

2. 将静默失败升级为错误!

静默失败: 即执行不成功，又不会报错!

3. 不再推荐使用arguments.callee

arguments.callee 专用于在函数内部递归调用当前函数本身。

禁用arguments.callee，就等效于不推荐使用递归

今后所有递归都可用循环代替

4. 匿名函数自调中的this不再指向window

对对象的保护:

保护属性: 将属性分为2大类

命名属性: 可用.直接访问的属性

又分为2小类:

数据属性: 直接存储属性值

如何保护: 4大特性:

属性:{

value: 值, 实际存储属性值

writable: true/false, 控制是否可修改属性值

enumerable:true/fales, 控制是否可用for in遍历到

问题: 仅控制for in，不控制.直接访问

configurable:true/false, 控制2件事:

1. 是否可删除当前属性

2. 是否可修改其它特性

特点: 一旦该为false，不可逆！

}

API:

获取一个属性的四大特性:

var attrs=

Object.getOwnPropertyDescriptor(obj,"属性名")

设置一个属性的四大特性:

Object.defineProperty(obj,"属性名",{

要修改的特性: 值,

... : ...

})//问题: 每次只能修改一个属性的四大特性

解决: Object.defineProperties(obj,{

属性名:{

特性名:值,

... : ... ,

},  
 属性名:{

... : ...,

}

})

强调: 如果要修改的属性不存在，则自动添加

坑: 如果用defineProperty添加的新属性，四大特性默认值都为false!

问题: 数据属性的特性无法用自定义规则保护自己

解决:

访问器属性: 不直接存储属性值，仅提供对另一个数据属性的保护

何时: 只要用自定义规则保护属性时

如何定义:

前提: 先定义一个隐藏的数据属性被保护，实际存储属性值.

4大特性:{

get(){ return this.受保护的数据属性},

set(val){

val: 会自动获得要赋的新值

如果val符合自定义规则

this.受保护的数据属性=val

否则

报错! 且不赋值!

},

enumerable:true,

configurable: false

}

如何发挥作用:

使用时不要用隐藏的数据属性，要用访问器属性

试图用访问器属性获取属性值时，自动调用get()

试图给访问器属性赋值时，自动调用set(新值)

问题: 隐藏并不彻底，依然可通过.直接给\_xxx赋值，就绕过了访问器属性的保护。

解决: 闭包: 为访问器属性封装专门的变量

2步: 1. 在构造函数中定义局部变量\_xxx实际存储属性值

2. 去掉访问器属性中的this.

内部属性: 不能用.访问到的属性

除此之外，访问器属性和普通属性的用法，完全一样!

比如: class

问题: 仅保护住单个属性，还没有保护住对象的结构

保护结构: 3等级:

1. 防扩展: 禁止对当前对象添加任何新属性

Object.preventExtensions(obj)

原理: 每个对象都有一个内部属性:extensible:true

preventExtensions() 将对象的extensible:false

2. 密封: 即防扩展，又禁止删除现有属性

Object.seal(obj)

原理: 2件事:

1. 将对象的extensible:false

2. 自动将所有属性的configurable特性统一改为false

3. 冻结: 禁止对对象的结构和属性值做任何修改。

Object.freeze(obj)

原理: 2件事:

1. 将对象的extensible:false

2. 自动将所有属性的configurable特性统一改为false

3. 自动将所有属性的writable改为false！

2. Object.create:

什么是: 基于一个现有父对象，创建一个子对象，并扩展自有属性

何时: 没有构造函数，也想创建子对象，继承父对象时

如何: var child=Object.create(father/\*,{

//defineProperties一样

属性名:{

特性:值,

... : ... ,

},

属性名:{

特性:值,

... : ... ,

}

}\*/)

3. .call() .apply() .bind()

共同的作用: 替换函数中不正确的this为指定对象

何时: 只要函数中的this不是想要的，都可用三个函数换

差别:

1. 在调用函数时，临时替换函数中的this为指定对象

.call() .apply()

.call() 要求传入函数的参数必须单独传入

.apply() 要求传入函数的参数列表必须放在一个数组中

何时: 只要调用函数时，临时替换

2. 基于现有函数，创建一个新函数，并永久绑定this为指定对象

.bind()

何时: 需要创建一个新函数，并永久绑定this时