正课:

ES5:

1. 严格模式:

2. \*\*\*对对象的保护

数据属性, 访问器属性, 防篡改

3. Object.create()

4. \*\*call/apply/bind

5. 数组API

1. 严格模式:

什么是: 比传统js运行机制要求更严格的模式

为什么: js本身有很多广受诟病的缺陷

何时: 今后所有项目的开发，都必须在严格模式下进行

如何: 2个范围:

1. 在整个js文件或script标签内启用严格模式:

在js文件或script标签内第一行: "use strict";

何时: 新项目，都要启用严格模式

2. 仅在单个函数内启用严格模式:

在函数内的第一行:"use strict";

何时: 旧项目改造时，要逐个功能向严格模式迁移

要求:

1. 禁止给未声明的变量赋值:

2. 将静默失败升级为错误:

静默失败: 即修改不成功，又不报错！

3. 不建议使用arguments, arguments.callee

4. 普通函数调用和匿名函数自调中的this不再指向window！而是undefined

2. 对对象的保护:

为什么: js中普通的对象对自己的属性和结构，没有任何验证和保护的办法。

何时: 今后所有对象都要对自己的属性提供一定保护

如何:

1. 保护属性

ES5规定: 对象的属性分为2大类:

命名属性: 可用.直接访问到的属性

又细分为2类:

1. 数据属性: 直接存储属性值的属性

如何保护: 每个数据属性不再是一个简单的变量

而是一个拥有四大特定的小对象。

查看一个属性的四大特性:

Object.getOwnPropertyDescriptor(obj,"属性名")

返回值: {

value: 实际存储属性值,

writable: true/false, 控制只读

enumerable: true/false, 控制能否被for in遍历到

但是，用.依然可以访问到！

configurable: true/false, 控制:

1. 控制能否删除该属性

2. 控制能否修改其它特性

一旦该为false，不可逆！

总是伴随其它特性的修改，充当双保险

}

如何修改四大特性保护数据属性:

Object.defineProperty(obj,"属性名",{

要修改的特性:值,

... : ...,

})

问题: 一次只能修改一个属性的四大特性

解决: 同时修改多个属性的四大特性:

Object.defineProperties(obj, {

属性名:{

四大特性:值,

... : ...

},

属性名:{

四大特性:值,

... : ...

},

...

})

说明: 如果要修改的属性不存在，则自动创建

但是, 自动创建的属性，四大特性默认值都为false

问题: 只能用固定的三种特性保护属性，无法用自定义规则灵活保护属性

2. 访问器属性:

什么是: 不直接存储属性值，仅提供对其它数据属性的保护。

何时: 只要用自定义规则保护属性时

如何定义: 2步:

1. 定义隐藏的数据属性,起别名，实际存储数据

问题: 使用enumerable隐藏的属性，防for in，但防不住. 别人可用eric.\_age绕过访问器属性，直接篡改受保护的属性。

解决: 使用局部变量，代替隐藏的数据属性!

2. 定义访问器属性，从受保护的数据属性中读取或修改数据

Object.defineProperties(eric,{

age:{//访问器属性

get(){return \_age},

set(val){

if(val>=18&&val<=65)

\_age=val;

else

throw new RangeError("年龄必须介于18~65之间");

},

enumberable:true,

configurable:true

},

})

如何使用: 访问器属性的用法和普通属性完全一样！

只不过: 在试图获取属性值时，自动调用get

在试图修改属性值时，自动调用set

参数val自动获得等号右边的新值

内部属性: 不能用.访问的属性: class \_ \_proto\_ \_

2. 防篡改:(保护结构)

3个级别:

1. 防扩展: 禁止向对象中添加新属性:

如何: Object.preventExtensions(obj)

原理: 每个对象内都有一个隐藏的内部属性extensible

默认true

preventExtensions将extensible改为false

2. 密封: 即禁止扩展，又禁止删除！

如何: Object.seal(obj)

原理: 即修改对象的extensible为false

又自动修改每个属性的configurable为false！

3. Object.create()

什么是: 仅基于一个现有父对象，创建他的子对象，并为子对象扩展新的自有属性

何时: 今后，如果只基于父对象，也想创建子对象时

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

属性名:{四大特性}

... : ...

});

4. call/apply/bind:

相同: 如果函数中的this不是想要的，都可替换

不同:

call/apply: 强行\*调用\*一个函数，并\*临时\*替换函数中的this为指定对象

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必须放在数组中整体传入

apply可先打散数组参数为单个元素，再传参

bind: 基于一个现有函数，\*创建\*一个一模一样的新函数，并\*永久\*绑定this为指定对象

总结: 如果是临时\*调用\*一个函数,用call/apply

如果"创建"一个新函数，永久绑定this时

所有回调函数中的this，要想替换都用bind

因为回调函数不是立刻执行，且不止执行一次！

5. 数组API:

indexOf()已学

1. 判断: ——了解

every(): 判断数组中是否所有元素\*都\*符合要求

some(): 判断数组中是否包含符合要求的元素

如何: 仅以every()

var bool=arr.every(function(elem,i,arr){

//elem自动获得当前元素值

//i 自动获得当前位置

//arr 自动获得当前数组->this

return 针对当前元素执行的判断条件

})

执行过程: every会遍历每个元素，反复调用回调函数，只有所有函数调用都返回true，结果才返回true

2. 遍历API: 对数组中每个元素执行相同的操作

forEach(): 对原数组中每个元素执行相同操作

map(): 依次取出原数组中每个元素，执行相同操作后，放入新数组中返回。

如何:

arr.forEach(function(elem,i,arr){

arr[i]=新值;

});

arr.map(function(elem,i,arr){

return 新值;

})//return的新值会放入新数组中相同位置

3. 过滤和汇总:

过滤: 选取出原数组中符合条件的元素，组成新数组返回

var subs=arr.filter(function(elem,i,arr){

return 判断条件

}); //如果当前判断为true，则放入新数组返回

汇总: 将数组中每个元素统计汇总出一个最终的结果

var result=arr.reduce(function(prev,elem,i,arr){

//prev: 截止目前的临时汇总值

return prev+elem;

},base)

以base值为基数，累加arr中每个元素值