正课:

1. ES5

1. ES5

严格模式:

要求:

1. 禁止给未声明的变量赋值

解决: 内存泄漏，全局污染

2. 静默失败升级为错误

3. 普通函数调用中的this不再指向window，而是undefined

解决: 内存泄漏，全局污染

4. 不再推荐使用arguments.callee arguments.caller

arguments.callee

保护对象:

保护属性:

ES5中对象属性分为:

内部属性: 仅在对象内保存，不能用.直接访问的属性

命名属性: 可用.随意访问的属性

细分为2类:

数据属性: 实际存储属性值的属性

访问器属性: 不实际存储属性值，仅提供对另一个数据属性的保护

保护数据属性:

每个数据属性，又包含四大特性: {

value: 保存属性值,

writable: 控制是否可修改,

enumerable: 控制是否可被for in遍历,

configurable: 控制是否可删除该属性

控制是否可修改前两个特性

}

强调:

1. 通常修改writable和enumerable时，都将configurable改为false，双保险。configurable改为false后，不可逆！

2. enumerable只能防止遍历，用.还可访问。

获取一个属性的四大特性:

var obj=Object.getOwnPropertyDescriptor(obj, "属性名")

修改属性的第四大特性:

只改一个属性的四大特性:

Object.defineProperty(obj, "属性名", {

特性: bool,

... : ...

})

同时修改多个属性的四大特性:

Object.defineProperties(obj,{

属性名:{

特性:值,

... : ...

},

属性名:{

特性:值,

... : ...

},

})

问题: 无法用自定义规则保护属性

解决: 访问器属性:

访问器属性:

什么是: 不存储属性值, 仅提供对其它数据属性的保护

何时: 用自定义规则保护属性时

如何:

前提: 还有一个被保护的数据属性

四大特性: {

get: function(){ return 从受保护的数据属性中取值}，

set: function(val){

if(满足条件)

将val的值保存给受保护的数据属性

else  
 报错！不赋值!

},

enumerable:true,

configurable:false

}

访问器属性只能用Object.defineProperty或defineProperties添加，不能用直接量添加。

如何发挥作用:

访问器属性的用法和普通属性完全一样:

试图获取属性值时，自动调用get方法

试图修改属性值时，自动调用set方法

其中val自动获得新值

保护结构: 3个级别:

1. 防扩展: 禁止添加新属性

Object.preventExtensions(obj)

原理:

每个对象中都隐藏着一个内部属性: extensible=true

preventExtensions() 将 extensible=false

2. 密封: 在禁止扩展同时禁止删除现有属性

Object.seal(obj)

原理:

1. extensible=false

2. 自动将所有属性的configurable都改为false

3. 冻结: 在密封基础上，禁止修改属性的值

Object.freeze(obj)

原理: 3件事:

1. extensible=false

2. configurable都改为false

3. writable=false

Object.create()

什么是: 仅基于一个现有父对象，创建子对象，并继承父对象。

何时: 如果没有构造函数，也想创建子对象时

如何: var child=Object.create(father,{

//同defineProperties中的格式

})

call, apply, bind

什么是: 替换函数中不想要的this

call,apply

什么是: 调用一个函数，并临时替换this为指定对象

如何: 函数.call(对象, 参数值, ....)

vs apply

apply要求传入的多个参数值，必须放在一个数组中整体传入.

bind

什么是: 基于原函数，创建一个新函数，并永久绑定this和部分参数

数组API: 3组:

1. 判断:

.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

返回值: bool

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

return 判断条件

})

2. 遍历:

.forEach(): 对原数组中每个元素执行相同操作

.map(): 取出原数组中每个元素，执行相同操作后，放入新数组返回。

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

//对原数组中每个元素执行相同操作

})

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

return //取出原数组中的值，执行操作后，返回

})