正课:

1. ES5

2. ES6

1. ES5

严格模式:

1. 禁止给未声明的变量赋值

2. 静默失败升级为错误

3. 普通函数调用或匿名函数自调中的this不再指向window，而是undefined

4. 禁止使用arguments.callee

arguments.callee是在函数内自动获得当前函数自身

问题: 递归的效率极低

重复计算量极大！

解决: 绝大多数的递归，都可用循环代替！

保护对象:

问题: js中对象底层其实就是一个关联数组，随时可修改属性值，随时可篡改属性结构。

解决: 保护对象:

保护单个属性:

ES5将对象属性分为:

内部属性: 对象中有，但是不能随意用.访问的属性

比如: 每个对象内都藏着一个class属性，保存着对象的类型名，不能用.class直接访问的。

不用保护

命名属性: 所有能用.访问到的属性

最需要保护的:

ES5中将命名属性又细分为2种

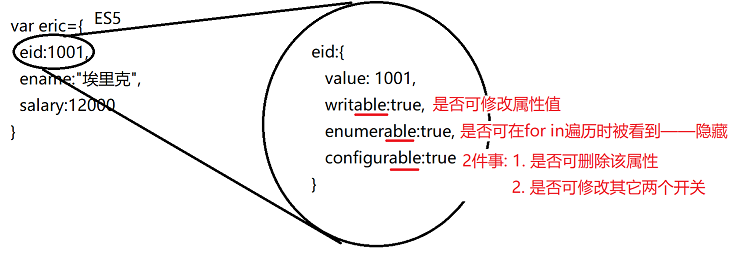
1. 数据属性:

什么是: 值直接保存在属性本地的属性

比如: var lilei={sname:"Li Lei"}

如何保护数据属性:

ES5中，一个属性，不再只是一个普通的变量，而是一个缩微的小对象。每个属性都有一个value特性存储属性值，还有三个开关，实现初步的自保能力:



修改开关: 不能用.，

必须使用defineProperty: HTML中: Attribute

定义 属性 属性

Object.defineProperty(对象,"属性名",{

开关: 开true/关false

})

比如: 让eric的eid属性只读:

Object.defineProperty(eric,"eid",{

writable: false

})

问题: 咱们能关，别人就能打开！

解决: 只要修改开关，都要双保险:

都要关上: configurable: false

配置 |可以

问题: 用enumerable:false隐藏的属性，其实用.依然可以访问。 枚举/遍历|可

问题: 一个defineProperty只能修改一个属性

解决: 用defineProperties

Object.defineProperties(对象,{

属性名:{

开关: true/false,

... : ...

},

属性名:{

开关: true/false,

... : ...

},

})

比如:

//员工编号只读:

//姓名禁止删除:

//salary禁止被遍历:

Object.defineProperties(eric,{

eid:{

writable:false,//管value

configurable:false//管writable

},

ename:{ configurable:false },

salary:{

enumerable:false,

configurable:false

}

});

问题: 只能用固定的规则保护属性，无法自定义规则

2. 访问器属性: 保镖

什么是: 不实际存储属性值，仅提供对其它数据属性保护业务的 特殊的属性

何时: 只要用自定义规则保护属性时，只能用访问器属性。

如何: 2步:

1. 先定义一个隐姓埋名的半隐藏的数据属性，实际存储属性值

Object.defineProperty(eric,"\_eage",{

value:25,

writable:true,

enumerable:false,

configurable:false,

})

2. 添加正式属性名称的访问器属性，来保护半隐藏的数据属性

必须用defineProperty添加，不能用{ . }

//定义保镖用正式的属性名代替受保护的属性抛头露面

Object.defineProperty(eric,"eage",{

//第一个保镖: 帮用户去属性中取值

get:function(){ return this.\_eage },

//第二个保镖: 帮用户将新值送入属性中，但是会先验证新值是否符合规定。如果新值不符合规定，则不保存直接报错。

set:function(value){//value自动接到新值

if(value>=18&&value<=65){

this.\_eage=value

}else{

throw Error("年龄超范围！")

}

},

enumerable:true,//代替受保护的属性抛头露面

configurable:false//不能删除保镖

})

保护对象结构: 防止别人对对象添加新属性或删除现有属性

3个层次:

1. 防扩展: 禁止向对象中添加新属性

Object.preventExtensions(obj);

阻止 扩展

禁止对obj添加新属性

原理: 其实每个对象中都有一个隐藏的内部属性:

extensible: true 所有对象默认都是可扩展的

Object.preventExtensions(obj); 将extensible改为false

2. 密封: 在兼具防扩展同时，进一步禁止删除所有属性

属性值依然可以改

Object.seal(obj) 禁止删除obj中任何属性

密封

原理: 1. preventExtensions

2. 自动将所有属性的configurable改为false！

3. 冻结: 在兼具密封的基础上，禁止修改所有属性值

Object.freeze(obj) 禁止修改obj中的所有属性的值

原理: 1. 自动执行seal()

2. 自动修改所有属性的writable特性为false

Object.create(): 如果没有构造函数，也想创建子对象继承父对象时

1. 创建新对象

2. 继承父对象

3. 为新对象添加新属性

如何:

var child=Object.create(father,{

//为child添加自有属性

属性:{value:值, 开关...},

... : ...

})

比如:

var hmm=Object.create(father,{

//defineProperties

//属性名:{ value:, 开关... }

bao:{

value:"LV",

writable:true,

enumerable:true,

configurable:false

},

phone:{

value:"iPhoneXXX",

writable:true,

enumerable:true,

configurable:false

}

});

vs new:

相同: 前三步做的事儿都是相同的

不同: new必须构造函数才能执行

create只要有father就能执行

call apply bind

替换函数中不想要的this！

何时: 只要函数执行时，其中的this不是想要的，都要用这三个去换

如何:

1. call/apply

调用 用

何时: 在本次调用函数时，临时替换一次this！

如何:

任意函数.call(任意对象,实参值列表...)

比如: calc.call(lilei,10000,2000,3000);

调用全局函数calc时，临时将其中的this替换为lilei

this.ename，就变为lilei.ename。

calc.call(hmm,10000,1000,2000);

调用全局函数calc时，临时将其中的this替换为hmm, this.ename就变为hmm.ename；