正课:

1. \*\*\*\*\*ES5

\*\*\*\*\*保护对象

Object.create();

\*\*\*\*call/apply/bind();

数组API

严格模式

2. ES6

模板字符串

let

箭头函数

for...of

class

1. ES5:

\*\*\*\*\*保护对象:

什么是: 1. 保护对象的属性和属性值始终有意义

2. 防止篡改对象的结构

为什么: 1. 对象的属性，默认可随意赋值

2. 对象可随意添加删除属性

何时: 严格来说，今后所有对象，都要有自我保护的抵抗力

如何:

保护属性:

分为:

命名属性: 可通过.直接访问的属性

分为:

数据属性: 直接保存属性值的属性

包括: 四大特性:{

value: 实际存储属性值,

writable: true/false, 控制是否可修改属性值,

enumerable: true/false, 控制是否可被for in遍历，

强调: 仅限制无法for in遍历到，依然可用.访问到。

configurable: true/false, 控制着禁止删除该属性

控制禁止修改其它特性

说明: configurable一旦改为false，不可逆

所以，在修改其它特性是，通常都用configurable为false，作为双保险

}

获取四大特性:

Object.getOwnPropertyDescriptor(obj,"属性名")

设置四大特性:

Object.defineProperty(obj,"属性名",{

要修改的特性: 特性值,

... : ...

})

问题: 一次只能修改一个属性

解决: 用defineProperties代替defineProperty

Object.defineProperties(obj,{

属性名:{

要修改的特性

},

... : { ... ... }

})

添加属性: 其实defineProperty也可添加新属性:

只要defineProperty要修改的属性不存在，就会自动添加。

强调: 用defineProperty添加的新属性，四大特性默认值都为false

而用.添加的新属性，四大特性默认值都为true。

访问器属性: 不直接存储属性值，仅提供对其它数据属性的保护

何时: 只要使用自定义的规则保护属性值

为什么: 数据属性的四大特性，保护规则是固定的，无法自定义。

如何定义: 2步:

1. 定义一个隐藏的数据属性: 用于实际存储属性值

2. 再定义一个访问器属性，来保护数据属性:

只能通过defineProperty或defineProperties

不能通过直接量或.创建

如何使用:

只要定义了访问器属性，就优先使用访问器属性，而不用受保护的数据属性.

访问器属性的用法和普通数据属性完全一样:

只不过: 在取值时，自动调用get()

在赋值时，自动调用set(val), val会自动获得要赋的新值

内部属性: 无法通过.直接访问的属性

比如: \_\_proto\_\_

防篡改

什么是: 阻止对对象结构的修改

如何: 3个级别:

1. 防扩展: 禁止添加新属性

Object.preventExtensions(obj)

2. 密封: 在防扩展的基础上，进一步禁止删除现有属性

Object.seal(obj);

原理: 将每个属性的configurable都自动改为false

3. 冻结: 在密封基础上禁止修改一切属性值

原理: 将每个属性的writable都自动设为false

Object.create()

什么是: 基于一个现有父对象，创建一个新的子对象继承当前父对象。并扩展新属性。

何时: 今后如果没有构造函数的情况下，也想创建子对象

如何: var child=Object.create(father);

创建一个新的子对象child，继承father

其实还可为新对象扩展新的自有属性:

var child=Object.create(father,{

新属性:{四大特性},

... :

});

call/apply/bind:

共同: 为了替换函数中不想要的this

何时: 只要函数中的this不是想要的

如何:

call/apply:

什么是: 1.\*调用\*一个函数

2. 将函数中的this\*临时\*替换为指定的新对象

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集中传入

其中: apply可打散数组类型参数，单个传入

何时: 只要调用函数时，函数中的this不是想要的，就用call/apply临时替换

bind: 基于现有函数\*创建\*一个新函数，并\*永久\*绑定this

强调: 被bind永久绑定的this，不能再用call/apply临时替换。

何时: 创建一个新函数，并且希望永久绑定this

固定套路: 替换回调函数中的this时，都用bind

数组API: 了解

判断:

1. every: 用于判断数组中所有元素是否都满足要求

arr.every(function(val,i,arr){

val//获得当前元素值

i//获得当前元素位置

arr//获得当前数组对象

return 判断条件;

});

2. some: 用于判断数组中是否包含满足条件的元素

arr.some(function(val,i,arr){

return 判断条件;

});

遍历API:

什么是: 对数组中每个元素执行相同的操作

包括:

1. forEach: 对原数组中每个元素执行相同操作

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

arr[i]=新值;

})——直接修改原数组

2. map: 取出原数组中每个元素，执行相同操作后，再放入新数组返回。

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

return 新值;

})——不修改原数组，而是将新值放入新数组中返回

过滤和汇总:

过滤: 选取原数组中符合条件的元素，组成新数组

arr.filter(function(val,i,arr){

return 判断条件

})——将原数组中符合判断条件的元素，复制到新数组中返回

汇总: 将数组中每个元素，统计出一个最终结果

arr.reduce(function(prev,val,i,arr){

prev //截止到当前元素之前的临时汇总值

return prev+val;

}/\*,start\*/) //start表示汇总开始的基数

严格模式:

什么是: 比普通js运行模式要求更严格的运行机制

为什么: 普通js运行模式中有广受诟病的缺陷

何时: 今后都要在严格模式下开发

1. 全新的项目: 必须全部启用严格模式

2. 旧项目: 逐个函数的向严格模式迁移

如何:

可在两个范围启用严格模式:

1. <script>或js文件的开头加入"use strict";

整个script或js文件内都采用严格模式运行

2. 仅在function内顶部添加"use strict";

只在函数内启用严格模式

规则:

1. 仅为给未声明的变量赋值

2. 静默失败升级为错误

3. 不推荐使用arguments.callee来实现递归

2. ES6

模板字符串: 对字符串拼接的简化

何时: 如果字符串中包含需要执行的表达式或回车换行时

如何: 3件事:

1. 用``反引号包裹字符串

2. 字符串中的表达式要放在${...}中

3. 模板字符串中支持换行

\*\*\*let:

1. 专门声明仅在当前块中有效的局部变量

块: js中只要一个{}就是一个代码块

let声明的变量，仅在{}内有效，不会被提前到{}外

2. 放置声明提前现象

何时: 今后建议用let代替var

\*\*\*箭头函数: 对所有回调函数的终极简写

何时: 今后只要回调函数，都要用箭头函数简写

如何:

1. 去function改=>

2. 更简化: 如果函数体只有一句话，可省略{}

如果仅有的一句话还是return，可省略return

3. 更简化: 如果只有一个参数，可省略()

特点: 简化了bind

箭头函数可让内外this共用同一个对象，不再需要用bind替换

问题: 有时，我们反而希望内外this不共用！

比如: addEventListener("click",function(){

this->当前单击的元素对象

})

解决: addEventListener("click",e=>{

e.target->当前单击的元素对象

})