正课:

1. ES5:

\*\*\*\*保护对象

数组API

Object.create

\*\*\*\*bind

严格模式

2. ES6:

模板字符串:

\*\*\*\*块作用域:

\*\*\*箭头函数

for of:

class

1. ES5:

\*\*\*\*保护对象:

保护属性:

命名属性: 可用.直接访问的属性

数据属性: 实际存储属性值的属性

如何保护: 每个属性其实由四大特性组成:

{

value: 值, //实际存储属性值

writable: true, //控制是否可修改

enumerable:true,//控制是否可用for in遍历到

//即使不能for in，用.也能访问

configurable:true,//控制是否可修改其他特性

//控制能否删除该属性

//改为false后不可逆

}

获取一个对象的属性的四大特性:

Object.getOwnPropertyDescriptor(obj,"属性名")

修改一个对象的属性的四大特性:

Object.defineProperty(obj,"属性名",{

特性名:特性值,

... : ... ,

});

问题: 一次只能修改一个属性

解决: 同时修改多个属性

Object.defineProperties(obj,{

属性名:{

特性名:特性值,

... : ...

},

属性名:{

特性名:特性值,

... : ...

},

})

强调: 必须启用严格模式才能报错！

问题: 只能进行简单的保护，无法自定义保护规则

解决:

访问器属性: 不实际存储属性值，仅提供对另一个属性值的保护。

何时: 只要使用自定义规则保护属性时

如何定义: 2步:

1. 必须先定义一个隐藏的属性来实际存储属性值

2. 定义访问器属性，提过对受保护的属性的读写操作和验证。

Object.defineProperty(obj,"属性名",{

get:function(){

return //受保护的属性值

},

set:function(val){

//如果val符合规则

受保护的属性=val

//否则

抛出自定义错误

},

enumerable:true, configurable:true,

})

强调:访问器属性中要使用对象自己的属性，也必须加this.

如何使用: 访问器属性的用法和普通属性的用法完全一样。

只不过: 获取访问器属性值时，自动调用get方法，实际获得的是受保护的隐藏属性的值

修改访问器属性值时，自动调用set方法，自动将新值传给val，经过验证才，实际保存到受保护的隐藏属性中。

内部属性: 不可用.访问的属性: \_\_proto\_\_

防篡改: 防止在对象创建后，修改对象的结构/内容

3个级别:

1. 防扩展: 禁止给对象添加任何新属性

Object.preventExtensions(obj);

2. 密封: seal 在防扩展基础上，再禁止删除现有属性

Object.seal(obj)

相当于将所有属性的configurable特性都改为false

3. 冻结: 禁止对对象做任何修改(包括结构和值)

Object.freeze(obj);

三件事: 1. 禁止扩展

2. 修改所有属性的configurable为false

3. 修改所有属性的writable为false

数组API: 3组

1. 判断: 判断数组中的元素是否符合要求:

every: 判断数组中的元素是否\*都\*符合要求:

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

//val, 当前元素值

//i, 当前位置

//arr, 当前数组

return 判断条件

})

some: 判断数组中是否\*包含\*符合要求的元素:

2. 遍历: 对每个元素执行相同操作

forEach: 对原数组中每个元素执行相同操作

直接修改原数组

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

arr[i]=新值;

})

map: 取出原数组中每个元素，执行相同操作后，放入一个新数组中返回

不修改原数组，仅返回新数组

var 新数组=arr.map(function(val,i,arr){

return 新值;

});

3. 过滤和汇总:

过滤: 筛选出原数组中符合条件的元素组成新数组。

原数组不变。

var subArr=arr.filter(function(val,i,arr){

return 判断条件

})

汇总: 将数组中每个元素的值，汇总成一个最终结果

返回值: 一个汇总结果

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

return prev+val;//累加

},base);

问题: 仅简化了代码，未提高程序的性能

所有遍历API中使用的都是for循环遍历每个元素

Object.create方法: 直接用父对象创建子对象，并扩展子对象的自有属性:

何时: 如果只有父对象，也想创建子对象时

如何: var child=Object.create(father,{

新属性:{四大特性},

... : ...

});

\*\*\*\*call/apply/bind: 替换函数中不想要的this

call/apply : 临时替换函数中的this

差别: 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入数组中整体传入。apply会将数组打散为单个参数值分别传入

强调: call/apply相当于调用函数，立刻执行。

bind: 永久绑定函数中的this

3件事: 1. 创建一个和原函数功能完全一样的新函数

2. 将新函数中的this永久绑定为指定对象

3. 将新函数中的部分固定参数提前永久绑定

强调: bind绑定在新函数中的this，无法被call/apply再替换。

总结: 如果临时调用一个函数，立刻执行时——call/apply

如果创建一个函数提前绑定this时，不一定立刻执行时——bind

严格模式:

1. 禁止给未声明的变量赋值

2. 将静默失败升级为错误

ES6:

模板字符串: 简化字符串拼接

何时: 只要一个字符串需要动态拼接而成时

如何: 反引号

\*\*\*\*let: 解决声明提前的问题

何时: 今后只要声明一个变量，首选let

特点: 2个:

1. let会将变量的作用域，限制在一个块内部

——增加了块级作用域

2. let之前不允许再出现未声明的同名变量

——避免了声明提前

\*\*\*箭头函数: 简化所有回调函数:

如何:

1. 去掉function，在()和{}之间加=>

2. 更简化:

如果只有一个参数，可省略()

如果没有参数，必须保留()

如果函数体只有一句话，可省略{}

如果函数体只有一句话，且是return，可省了{}和return

强调: 箭头函数简写后，函数中的this和外部this一致了

所以: 当回调函数内外的this不相同时，不能使用箭头函数简化。

反之: 如果希望函数内外公用一个this时，就可用箭头函数简化回调函数。

for...of: 简化普通for循环遍历:

for(var i=0;i<arr.length;i++){

arr[i] //当前元素值

}

for(var val of arr){

val //当前元素值

}

问题1: 仅适用于读取元素值的情况，不能修改原数组元素。

问题2: 只能遍历数字下标的索引数组和类数组对象。不能遍历关联数组中的元素值。

\*\*\*class: 简化: 封装,继承,多态

创建一个类型:

1. 用class结构包裹构造函数和原型对象方法, 将类型名定义在class后

2. 构造函数的function 函数名，改为constructor

3. 所有原型方法省略"类型.prototype"和"=function"

两种类型间的继承:

1. 不再需要Object.setPrototype

而是class 子类型 extends 父类型

2. 借用构造函数不再用call/apply，而用super(...)

其中super中省略this