### 一．js高阶 + es6

Day1 ················································1

Day2 ················································1

Day3 ················································1

Day4 ················································1

Day5 ················································1

Day6 ················（es6开始）······················1

Day7 ················································1

Day8 ················································1

Day1

张东

zhangdong@tedu.cn

正课:

1. String

2. 内置对象和包装类型

3. String API

4. 正则表达式

1. String字符串

什么是String: 一串字符组成的只读字符数组

vs 数组:

相同: 1. 下标,

2. length

固定套路:

1. 最后一个元素: arr[arr.length-1]

倒数第n个元素: arr[arr.length-n]

2. 追加一个新元素: arr[arr.length]=值;

3. 删除最后一个元素: arr.length--

清空数组: arr.length=0

3. for遍历

4. slice:

arr.slice(starti,endi+1) 含头不含尾

只知道要获取的个数:

arr.slice(starti,starti+n)

简写:

1. 负数参数: 表示倒数第n个位置

2. 省略第二个参数: 一直选取到结尾

3. 两个参数都省略: 完整复制数组

不同: 类型不同！API不通用！

2. 内置对象和包装类型:

鄙视: 请列举js中所有内置对象: 11个

什么是内置对象: ES标准中规定的，浏览器厂商已经实现的，程序员可直接使用的对象

包含: String Number Boolean ——包装类型

Array Date RegExp Math

Error

Function Object

Global(在浏览器中被window代替)

鄙视: 对包装类型的理解

什么是包装类型: 专门封装原始类型的值，并提供操作原始类型值的API 的类型

为什么: 原始类型的值，本身没有任何功能！

何时: 试图用原始类型的值调用API时，程序会自动创建包装类型对象

特点: 调用后，自动释放！

3. String API

强调: 所有String API都无权修改原字符串，只能返回新字符串！

大小写转换:

str.toUpperCase() 转为大写

str.toLowerCase() 转为小写

何时: 只要不区分大小写时，都要先转为一致的大小写，再判断和比较

比如: 验证码，用户名和邮箱地址

获得指定位置的字符:

str.charAt(i) <==> str[i]

str.charCodeAt(i) 获得i位置的字符的unicode号

String.fromCharCode(unicode) 将unicode转回字

选取子字符串:

str.substring(starti,endi+1) <==> str.slice(starti,endi+1)

问题:substring不支持负数参数！

解决: str.substring(starti,str.length-n+1)

如果只知道个数:

str.substr(starti,n) <==> str.slice(starti,starti+n)

强调: 只有两个参数都是下标的函数，才考虑含头不含尾。所以，substr不用考虑含头不含尾，n不用+1。

查找关键词: 4种:

1. 查找一个固定的关键词出现的位置

var i=str.indexOf("关键词",fromi)

在str中，从fromi位置开始，找下一个"关键词"的位置。

返回值: 本次找到的一个关键词的下标！

如果找不到，返回-1

如果直接找最后一个关键词:

var i=str.lastIndexOf("关键词");

问题: 只能查找一种固定的关键词

解决: 用正则表达式模糊查找多种关键词

2. 判断是否包含符合规则的敏感词:

var i=str.search(/正则表达式/i);

问题1: 所有正则表达式，默认区分大小写！

解决: 在第二个/后加i (ignore忽略)

问题2: 只能找第一个，无法找所有！

3. 查找所有关键词的内容:

var kwords=str.match(/正则/ig)

返回值: 包含所有关键词的数组

如果找不到，返回null！

问题1: 默认只找第一个

解决: 在第二个/后加g (global全部)

问题2: 只能返回内容，无法返回每个关键词的位置

4. 即找每个关键词的内容，又找每个关键词的位置:

4. 正则表达式（Regular Expression）:

什么是: 规定字符串中，字符出现规律的规则

何时: 2种:

1. 用规则模糊查找多种关键词

2. 验证字符串的格式

如何:

1. 最简单的规则，是关键词原文

2. 字符集:

什么是: 规定1位字符，备选字符列表的集合

何时: 只要字符串中1位字符，有多种备选时

如何: [备选字符列表]

强调: 1. 不是数组，不加逗号！

2. 无论备选字符列表，有多少字，一个字符集只能匹配1位字符！

当备选字符列表部分连续时，可用-省略中间的字符

常见: 一位数字 [0-9]

一位小写字母[a-z]

一位大写字母[A-Z]

一位字母[A-Za-z]

一位字母或数字[0-9A-Za-z]

一位汉字[\u4e00-\u9fa5]

3. 预定义字符集: 4个

其实，就是对常用字符集的简写

\d 一位数字 [0-9]

\w 一位字母，数字或下划线 [0-9A-Za-z\_]

\s 一位空字符: 空格 换行 Tab

. 除换行外的任意字符

问题: 仅使用字符集，无法灵活定义字符的个数

4. 量词:

什么是: 专门规定一个字符集，出现次数的规则

何时: 只要固定一个字符集出现的次数，就用量词！

如何: 字符集量词

包括: 2大类:

1. 有明确数量限制:

字符集{m,n} 至少m个，最多n个

字符集{m, } 至少m个，多了不限！

字符集{m} 必须m个，不能多也不能少

比如: 手机号: 1[34578]\d{9}

QQ号: \d{4,11}

2. 没有明确数量限制:

字符集? 可有可无，最多1个

字符集\* 可有可无，数量不限

字符集+ 至少一个，多了不限

5. 选择和分组:

选择: 或 在多个规则中，匹配任意一个即可！

如何: 规则1|规则2

分组: () 将多个规则分为一组

为什么: 量词，默认只修饰相邻的前一个字符集

何时: 如果希望量词修饰多个字符集时，就用分组

如何: (多个字符集)

比如: 身份证号:

15位数字 2位数字 数字或Xx

\d{15} ( \d{2} [0-9Xx] ) ?

后三位整体可有可无，最多一次

比如: 完整手机号:

+86或0086 整体可有可无，最多一次

空字符 可有可无，数量不限

(\+86|0086)?\s\*1[34578]\d{9}

比如: 微信 weixin wx

(微|w(ei)?)\s\*(信|x(in)?)

6. 匹配特殊位置: 3个位置

何时: 只要匹配字符串特殊位置上的关键词时

包括:

开头 ^，结尾 $，

比如: ^\s+ 开头的空字符

\s+$ 结尾的空字符

^\s+|\s+$ 开头的和结尾的空字符

单词边界 \b: 包括 空格，标点...

何时: 只要匹配一个专门的单词时

Day2

正课:

String:

1. 替换

2. 切割

RegExp:

1. 验证:

2. 查找:

Math:

1. 取整

2. 乘方开方

3. 最大值/最小值

4. 随机数

String

1. 替换: 2种:

1. 简单替换: 将所有关键词替换为一个统一的值

如何: str=str.replace(/正则/i,"替换值")

2. 高级替换: 根据本次找到的关键词不同，动态选择不同的关键词替换

如何: str=str.replace(/正则/i,function(kword){

return 根据kword，动态返回不同的替换值

})

衍生: 删除: 将关键词替换为""

2. 切割:

什么是: 将字符串，按指定分隔符，切割为多段子字符串

如何: 2种:

1. 简单切割: 切割符是固定的字符串

var subs=str.split("切割符");

切割后的结果中，不包含切割符

固定套路: 将字符串打散为字符数组

var chars=str.split("");

2. 复杂切割: 切割符不是固定的

var subs=str.split(/正则/i);

RegExp:

什么是: 专门封装一条正则表达式，并使用正则表达式，执行验证和查找功能的对象

何时: 1. 查找, 2. 验证

如何:

1. 创建: 2种:

1. 直接量: var reg=/正则/ig

何时: 只要正则表达式是固定不变的！

2. 用new: var reg=new RegExp("正则","ig");

何时: 如果正则表达式需要动态生成！

2. 验证: 验证一个字符串的格式，是否符合正则表达式的要求

var bool=reg.test(str);

问题: 默认，正则表达式只要部分匹配，就返回true

解决: 只要验证，必须前加^，后加$，要求，从头到尾完整匹配

3. 查找: 即找每个关键词的内容，又找每个关键词的位置

var arr=reg.exec(str);

在str中，找下一个符合reg要求的关键词

返回值: 数组:[

0:"关键词内容", index:本次找到关键词的位置

]

如果找不到，返回null

原理: 每次查找3件事:

1. 将本次找到的关键词，放入arr中0位置

2. 将本次关键词的位置，放入arr中index位置

3. 自动修改reg的lastIndex属性为当前位置+关键词长度

Math

什么是: 专门封装数学计算的常量，并提供数学计算的API的对象

何时: 只要执行数学计算时

如何:

创建: 不能new！必须用Math直接调用API

1. 取整: 3种:

上取整: 只要超过，就取下一个整数！

Math.ceil(num)

下取整: 舍弃小数部分

Math.floor(num) 只能处理纯数字内容

vs parseInt(str) 可去掉数字结尾的非数字字符

四舍五入取整:

Math.round(num) 缺: 只能取整！优: 返回number类型

vs num.toFixed(d)

优: 按任意小数位数四舍五入

缺: 返回字符串类型

自定义:

2. 乘方和开平方:

Math.pow(底数,幂)

Math.sqrt(num)

3. 最大值、最小值

Math.max(值1,值2,...)

Math.min(值1,值2,...)

问题: 不支持获得数组中的最大值，最小值

解决: Math.max/min.apply(null,arr)

其中: apply，负责将arr打散为单个元素，再传给max/min

4. 随机数:

Math.random() 在0~1之间生成一个随机小数

在任意min~max之间，取一个随机整数的公式

parseInt(Math.random()\*(max-min+1)+min)

简写: 从0~max之间取随机:

parseInt(Math.random()\*(max +1))

Day3

正课:

1. Date

2. Error

3. \*\*\*\*Function

创建，重载，匿名函数，

\*\*\*\*作用域和作用域链

\*\*\*\*\*闭包

1. Date:

什么是: 封装日期和时间, 提供操作日期和时间的API

何时: 只要存储日期和时间，都用Date类型的对象

如何:

创建: 4种:

1. 获得当前系统时间: var now=new Date();

强调: 只能获得客户端当前操作系统的时间

2. 保存自定义时间:

var date=new Date("yyyy/MM/dd hh:mm:ss");

3. 复制一个日期对象:

为什么: 今后，日期对象的计算，都直接修改原日期对象。计算后，原日期无法保留。

何时: 只要用户希望同时保留计算前后的开始和结束时间时，都要先复制副本。再用副本计算结束时间。

如何: var date2=new Date(date1);

4. 将毫秒数转化为当地时间:

为什么: 系统中存储时间，都是使用毫秒数

何时: 只要将毫秒数转为当地时间显示

如何: var date=new Date(ms);

补: 如何获取毫秒数: var ms=date.getTime();

原理: 日期对象中，保存的是1970年1月1日0点0分0秒至今的毫秒数

为什么: 文字会受时区的影响，而毫秒数不受时区的影响

API:

1. 8个单位:

FullYear Month Date Day 没有s结尾

Hours Minutes Seconds Milliseconds 都有s结尾

2. 每个单位都有一对儿getXXX/setXXX方法

其中: getXXX() 负责获取指定单位的数值

setXXX() 负责修改指定单位的数值

特例: Day没有set方法

3. 取值范围:

Month: 0~11 计算机中每个月份都比现实少1

读取时，+1修正; 修改时，-1修正

Date: 1~31 Day: 0~6

Hours: 0~23 Minutes/Seconds: 0~59

日期计算: 2种:

1. 两日期对象可相减，结果是毫秒差

2. 对任意单位的数值做加减: 3步

1. 取出单位的值: date.getXXX()

2. 做加减

3. 设置回去: date.setXXX(n)

强调: 1. setXXX()可自动根据n，调整时间进制

2. 如果希望同时保留计算前后的新旧时间，则必须先复制副本，再用副本计算。

其实可简写:

date.setXXX(date.getXXX()+n)

日期转字符串:

date.toString() 转为当地时间的完整格式

date.toLocaleString() 转为当地时间的简化版格式

date.toLocaleDateString() 转为当地时间的简化版格式，仅保留日期部分

date.toLocaleTimeString() 转为当地时间的简化版格式，仅保留时间部分

date.toGMTString() 转为国际标准时间(0时区)

自定义:

Error

什么是错误: 程序运行过程中，导致程序无法正常执行的问题。

程序发生错误后，都会强行中断退出

什么是错误处理: 当程序发生错误时，依然保证程序不中断退出的机制

何时: 只要有可能发生错误的代码，都要提前使用错误处理

如何:

try{

可能出错的正常逻辑代码

}catch(err){//只有出错才执行，不出错不执行！

一旦出错，才执行的错误处理代码:

提示用户, 记录日志, 保存数据

}

其中: err: 错误对象: 专门封装错误信息的对象

发生错误时，自动创建！

主动抛出错误: throw new Error("错误信息")

何时: 函数的作者向函数的调用者抛出错误，提示调用者错误的使用了函数。

鄙视: 错误的类型: 6种

SyntaxError: 语法错误

ReferenceError: 引用错误: 要用的变量/对象，没找到！

TypeError: 类型错误

错误的使用了类型: 1. 错误的使用了()或[]

2. 错误的调用了对象的方法

RangeError: 范围错误: 参数超范围

EvalError; URIError;

Function

创建: 3种:

1. 声明: function 函数名(参数列表){

函数体;

return 返回值

}

问题: 被声明提前

解决:

2. 直接量: var函数名=function (参数列表){...}

优: 不会被声明提前!

揭示: 函数其实也是一个引用类型的对象

function是创建一个新函数的意思

函数名其实就是一个引用函数对象的简单变量

3. 用new:

var 函数名=new Function("参数变量",...,"函数体")

鄙视: 以下创建函数正确的是:

function cmp(a,b){return a-b;}√

var cmp=function(a,b){return a-b;};√

var cmp=new Function(a,b,"return a-b"); X

var cmp=new Function("a","b","return a-b"); √

重载(overload):

什么是: 相同函数名不同参数列表的多个函数，在调用时，可根据传入参数的不同，自动选择对应函数执行。

为什么: 减少函数的个数，减轻调用者的负担

何时: 只要一项任务，根据不同的参数，有不同实现逻辑时

问题: js语法默认不支持重载！

原因: js中不允许多个同名函数同时存在！

解决: 用arguments对象

什么是: 函数中，自动创建的

自动接收所有传入函数的参数值的

类数组对象(object like array)

相同: 1. 下标, 2. .length, 3. for遍历

不同: 类型不同! API不通用！

何时: 只要函数的参数个数不确定时，都可用arguments接住所有参数值

问题: 今后，还需不需要参数变量！

参数的作用:提醒调用者，如何正确使用函数！

一般自定义的参数名，都是见名知意，且都非常简单

所以，函数都要优先使用参数变量接收参数值

Day4

微信公众号: 前端大全

书: 犀牛书

上届笔记

正课:

\*\*\*\*Function

1. 匿名函数

2. \*\*\*\*作用域和作用域链

3. \*\*\*\*\*闭包

\*\*\*OOP

封装，继承，多态

\*\*\*\*Function

1. 匿名函数:

什么是: 创建函数时，不指定函数名的函数

为什么: 节约内存，划分临时作用域

何时: 2种:

1. 如果一个函数，只使用一次时

2. 如果避免使用全局变量，划分临时作用域时

如何: 2种:

1. 回调: 将一个函数，交给另一个函数去调用

3种: 1. sort比较器函数

2. str.replace高级替换

3. 事件处理函数

2. 自调: 创建函数后，立刻调用自己

为什么: 全局变量最大的问题：全局污染

何时: 避免使用全局变量，造成全局污染时

今后，几乎所有的js程序，都应该放在匿名函数自调中

如何自调: 2种

(function (参数列表){ 函数体 }) ();

!function(参数列表){ 函数体 }();

2. \*\*\*\*作用域(scope): //练习10分钟，下课休息10分钟

什么是: 一个变量的可用范围——用途

一个存储变量的对象——存储/本质

为什么: 避免不同范围的变量间互相干扰

包括: 2种:

1. 全局作用域: 不属于任何函数的，外部的范围

其实: window

全局变量: 反复使用，随处可用！

2. 函数作用域: 函数内的范围

其实: 活动对象(AO)

局部变量: 仅函数内可用，不可反复使用！

函数的生命周期:

1. 开始执行程序前

创建执行环境栈(ECS): 保存正在调用的函数记录

首先自动调用浏览器主程序main()

主程序创建全局作用域对象window

2. 定义函数时:

在window中用函数名创建全局变量

window外创建函数对象，保存函数定义

函数名变量引用函数对象

函数对象的score属性指回函数来自的作用域对象

3. 调用函数时:

在ECS中压入当前函数的调用记录

为本次函数调用创建活动对象AO

AO中保存局部变量

让AO指向函数的scope属性指向的相同对象window

变量的使用顺序:

优先用AO中的局部变量，AO中没有，才去window找

4. 函数调用后:

函数调用的记录从ECS中出栈

导致: AO释放->导致: AO中的局部变量一同释放！

所以，局部变量仅在函数调用时可用，不可重用！

作用域链(scope chain): 由多级作用域连续引用形成的链式结构

存储: 所有变量

控制: 变量的使用顺序: 先局部，后全局

3. \*\*\*\*\*闭包(closure):

什么是: 即重用变量，又保护变量不被污染的一种机制

为什么:

全局变量: 优: 可重用！ 缺: 污染，随处可用!

局部变量: 优: 仅函数内可用，不会污染全局

缺: 不可重用！

何时: 只要即重用一个变量，又保护变量不被污染时

如何: 3步:

1. 用外层函数包裹要保护的变量和操作变量的函数

2. 外层函数将内层函数的对象返回到外部

3. 使用者调用外层函数，获得内层函数对象

闭包的形成原因: 外层函数的作用域对象(AO)无法释放

被内层函数对象引用着

简图: 2步:

1. 找受保护的变量，并确定其最终值

2. 找使用变量的内层函数对象，只有内层函数才能使用受保护的变量

闭包缺点: 比普通函数占用更多内存空间！

解决: 闭包不再使用，要及时释放

如何释放: 将引用内层函数对象的变量赋值为null

\*\*\*OOP

什么是面向对象: 程序中都是用对象结构描述现实中一个具体事物

什么是对象: 程序中专门描述现实中一个具体事物的程序结构

为什么: 现实中，一个数据，必须属于某个具体对象中，才有意义。

何时: 今后，只要描述一个事物，都要将事物的属性和功能集中定义在一个对象中

如何: 面向对象三大特点:

封装，继承，多态

封装:

什么是: 创建一个对象，存储一个事物的属性和功能

其中: 事物的属性，会成为对象的属性

属性其实就是存在对象中的变量

事物的功能，会成为对象的方法

方法其实就是存在对象中的函数

功能和属性，统称为成员

为什么: 便于大量维护数据

何时: 只要使用面向对象的方式编程，都必须先将事物的属性和功能封装在对象中，再按需使用对象的属性和方法。

如何: 3种:

1. 对象直接量:

var obj={

属性名:值,

... : ... ,

方法名 (){//ES6

...

}

}

如何访问对象的成员：

obj.属性名 -> 每个属性的用法和普通变量完全一样！

obj.方法名() -> 每个方法的用法和普通函数完全一样！

Day5

正课:

\*\*\*OOP

1. 封装

2. 继承

3. 多态

4. 自定义继承

1. 封装: 3种:

1. 直接量:

var obj={

属性名:值,

... : ... ,

方法名(){

... this.属性名 ...

}

}

问题: 对象自己的方法，要使用自己的属性值，缺找不到！

原因: 不加.访问的变量，只能在作用域链中查找，无法自动进入对象中查找

解决: 错误: 在属性前加"对象."

在对象内，禁止写死对象名

因为对象名其实仅是一个普通变量名，很有可能发生变化

正确: 用this.属性名:

this: 在函数执行时，自动创建的一个关键词

自动指向正在调用当前函数的.前的对象

为什么: 不受对象名的影响

总结: 今后，只要对象自己的方法，想访问自己的属性，必须用this.属性名

2. 用new: 2步:

1. 先创建一个空对象

var obj=new Object(); //new可省略,()也可省略,但不能同时省

2. 向空对象中添加新成员:

obj.属性名=值;

obj.方法名=function(){ ... this.属性名 ... }

说明: js中的对象，可随时，通过强行赋值的方式，添加新成员

何时: 如果在创建对象时，暂时不知道对象的成员

揭示: js中的对象，底层其实就是关联数组

相同: 1. 都可用.或[]随时添加/访问成员

2. 访问不存在的成员，不报错，返回undefined

3. 都可用for in遍历每个成员

不同: 对象比关联数组用法简单！

今后，都是用对象来代替关联数组使用！

问题: 一次只能创建一个对象

如果反复创建相同结构的多个对象，重复代码会很多

解决:构造函数

3. 用构造函数(constructor):

什么是: 描述同一类型的多个对象，相同成员结构的函数

第二个作用: 将一个空对象构造为拥有属性和功能的完整对象

何时: 只要反复创建同一类型的多个对象时，都要先用构造函数描述统一的结构，再用构造函数创建对象

如何: 2步

1. 定义构造函数:

function 类型名(属性参数,....){

this.属性名=属性参数;

... = ... ;

this.方法名=function(){

... this.属性名 ...

}

}

2. 使用构造函数反复创建对象:

var obj= new类型名(属性值,...);

调用时的参数值，应和定义构造函数时的属性参数保持一致。

new: 4件事:

1. 创建一个新的空对象

2. 自动让新的子对象继承构造函数的原型对象

3. 调用构造函数，将构造函数中的this执行正在创建的新对象。向新的空对象中强行添加新成员

4. 将新对象地址返回给变量保存

优: 重用结构定义

缺: 浪费内存

2. 继承:

什么是: 父对象的成员，子对象无需创建即可直接使用！

为什么: 代码重用！节约内存！

何时: 所有子对象都拥有相同的属性值和方法定义时，都要用继承来实现

如何: js中所有继承，都是继承原型对象

什么是原型对象: 集中存储所有子对象共有成员的父对象

为什么: 实现继承

何时: 只要实现继承，都要继承原型对象

如何:

创建: 不用手动创建，买一赠一

其实创建构造函数同时，都附赠一个空的原型对象

继承: 不用手动设置

用new创建新的子对象时，会自动设置新对象继承构造函数的原型对象

添加共有成员:

构造函数.prototype.成员=值;

总结: 只要所有子对象共用的成员，都必须集中存储在原型对象中

自有属性和共有属性:

自有属性: 直接保存在对象本地的属性

共有属性: 保存在原型对象中，所有子对象共有的属性

读取: 两者完全一样: 对象.属性名

修改: 自有属性: 只能用子对象改: 子对象.自有属性名=值

共有属性: 只能用原型对象修改:

构造函数.prototype.共有属性名=值

原型链: prototype chain

什么是: 由多级父对象，逐级继承，形成的链式结构

为什么: 为了更高级,更大范围的重用

如何:

所有对象，都有\_\_proto\_\_属性

原型对象的\_\_proto\_\_指向更上级的父对象

所有对象最终都继承自Object.prototype——顶级父对象

内置对象的原型链:

其实每种内置类型都有对应的构造函数和原型对象，也最终都继承自Object.prototype

其中: 内置类型的构造函数负责创建该类型的子对象

内置类型的原型对象负责保存该类型所有子对象共有的API

问题: 旧浏览器不支持新的API

解决: 向旧浏览器中的原型对象中手动添加一个函数

鄙视: 判断一个对象是不是数组类型，有几种方法:

typeof不行！

1. 判断原型对象:

Object.getPrototypeOf(obj)==Array.prototype

判断obj是数组类型的子对象

问题: \_\_proto\_\_是内部属性，本不应该被访问到

解决: 用Object.getPrototypeOf(obj) 代替\_\_proto\_\_

2. 判断构造函数:

实例

obj instanceof Array

判断obj是不是被构造函数Array创造出来的

instanceof 不仅判断直接父类型，而是所有在原型链上的类型，都返回true！

3. 判断对象的内部class属性

每个对象内部，都有一个隐藏的class属性，记录该对象创建时的数据类型

class属性不会随继承关系的改变而改变

问题1: class是内部属性

解决: 只有最顶层的toString()才能输出对象的class属性值

[object class名]

问题2: 内置类型的原型对象中几乎都重写了新的toString()

解决: 用call强行调用:

call: 让一个对象，调用一个本来无法调用到的函数

何时: 只要希望调用一个本无法调用到的函数

如何: 要调用的函数.call(对象)

Object.prototype.toString.call(obj)=="[object Array]"

说明obj的内部属性class的值为"Array"

3. 多态:

什么是: 同一个函数在不同情况下表现出不同的状态

重写: 如果子对象觉得父对象的成员不好用，可在本地定义同名成员，覆盖父对象中继承来的成员

为什么: 体现子对象和父对象之间的差异

何时: 只要子对象觉得父对象的成员不好用，就可以重写！

Day6

正课:

\*\*\*\*ES5

严格模式

\*\*\*\*保护对象: 保护属性和防篡改

Object.create

\*\*\*call,apply,bind

数组API

严格模式:

什么是: 比普通js运行机制更严格的模式

为什么: js中存在很多广受诟病的缺陷

1. 允许给未声明的变量赋值

2. 静默失败

何时: 今后所有项目必须在严格模式下开发

如何:

在当前代码段的顶部添加"use strict";

在两个范围启用严格模式:

在全局启用严格模式: 在全局添加"use strict"

何时: 新项目，必须在全局启用

仅在函数内启用严格模式: 只在函数内顶部添加"use strict"

何时: 旧项目，向严格模式迁移

要求:

1. 禁止给未声明的变量赋值！

2. 将所有静默失败升级为错误！

\*\*\*\*保护对象:

为什么: 传统js中，对象的属性值和结构，可随意更改，毫无自保和验证能力

何时: 今后几乎所有的对象，都要不给严格规定保护

ES5中的变化:

对象的属性也是一个小的对象:

对象的属性，包含四大特性:

{

value: 属性值,

writable:true/false, //控制是否可修改属性

enumerable:true/false,//控制是否可用for in遍历到该属性

//仅控制for in遍历，用.依然可以访问

configurable:true/false //控制:

//1. 禁止删除该属性

//2. 禁止修改其它特性的值

//一旦改为false，不可逆！

//今后，只要修改其它特性，都要伴随修改configurable为false——双保险

}

如何:

获取属性的四大特性:

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

修改属性的四大特性:

Object.defineProperty(obj,"属性名",{

特性名:值,

... : ... ,

})

问题: defineProperty每次只能修改一个属性

解决: defineProperties 一次可同时修改多个属性

Object.defineProperties(obj,{

属性名:{

特性名:值,

... : ...

},

属性名:{

特性名:值,

... : ...

},

})

问题: 普通属性的四大特性，只能提供基本的保护

无法使用自定义规则保护属性

解决: 用访问器属性:

访问器属性:

ES5将对象的属性分为: 数据属性和访问器属性

数据属性: 直接存储属性值的属性

访问器属性: 不直接存储数据，仅提供对其它数据属性的保护

为什么: 数据属性只能对自己提供基本的保护，无法使用自定义逻辑保护自己。

何时: 只要用自定义属性保护数据属性时，就只能用访问器属性

如何: 2步:

1. 首先定义一个隐藏的实际存储数据的数据属性

2. 然后定义访问器属性保护数据属性

Object.defineProperties(obj,{

\_age:{ //数据属性

value:25,

writable:true,

enumerable:false, //隐藏

configurable:false

},

age:{ //访问器属性

get(){ //在试图用访问器属性取值时，自动调用

return this.\_age;

},

set(val){ //在试图给访问器属性赋值时，自动调用

//val可自动获得要赋的新值

if(val在范围内)

this.\_age=val;

else 报错！

},

enumerable:true,

configurable:false

}

})

使用访问器属性: 用法和普通数据属性完全一样！

防篡改：禁止修改对象的结构

三个级别:

1. 防扩展: 禁止为对象强行添加新属性:

Object.preventExtensions(obj)

原理: 对象都有一个内部属性: extensible, 默认true

preventExtensions其实是将extensible改为false！

2. 密封: 在防扩展的同时，再禁止删除所有属性

Object.seal(obj)

原理: 将所有属性的configurable特性都改为false

3. 冻结: 在密封的同时，再禁止修改所有属性值

何时: 如果一个对象，连属性值都不让随便改，就要冻结

如何: Object.freeze(obj)

原理: 将所有属性的writable都自动改为false

Object.create:

三件事: 基于现有父对象，创建一个子对象，再扩展新属性

何时: 如果只有父对象，没有构造函数时，也想创建子对象

如何:

var child=Object.create(father,{

//defineProperties

属性名:{

特性名:值,

... : ...

},

属性名:{

特性名:值,

... : ...

},

})

Day7

正课:

\*\*\*\*ES5

1. call/apply/bind

2. 数组API

ES6

\*\*\*\*ES5:

1. call/apply/bind

替换this！

何时: 只要this不是想要的，就用call/apply/bind替换

鄙视: call vs apply vs bind

call/apply : 立刻调用函数，并临时替换函数中的this为指定对象

何时: 如果希望立刻调用函数时

如何: fun.call(obj,参数值,......)

立刻调用fun函数，并临时替换fun中this为obj

fun.apply(obj,数组)

立刻调用fun函数，并临时替换fun中this为obj

同时，打散数组参数为单个参数值，再传给fun

bind: 基于原函数，创建一个新函数，永久绑定this为指定对象

何时: 如果希望基于原函数创建一个新函数时

或者说，不是立刻调用时

如何: fun.bind(obj,参数值,...)

创建一个和fun完全一样的新函数

永久绑定fun中的this为obj

永久绑定部分参数值

2. 数组API:

3组:

1. 判断:

every(): 判断数组中是否所有元素都符合条件

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

//val自动获得当前元素值

//i获得当前位置

//arr 指当前数组对象本身

return 判断条件

})

原理: every用function去每个元素上执行一次

只有所有元素都返回true时，整体才返回true

只要有一个返回false，则整体返回false

some(): 判断数组中是否包含符合条件的元素

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

同every

return 判断条件;

})

原理: some用function去每个元素上执行一次

只要有一个返回true，则整体返回true

除非所有元素都返回false时，整体才返回false

2. 遍历API:

forEach(): 对原数组中每个元素执行相同的操作

如何: arr.forEach(function(val,i,arr){

arr[i]=新值;

})

map(): 依次取出原数组中的值，执行相同操作后，放入新数组返回。

如何: var newArr=arr.map(function(val,i,arr){

return新值;

})

3. 过滤和汇总:

filter(): 选取数组中符合条件的元素，组成新数组。原数组不变

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

return判断条件;

})

reduce(): 遍历数组中每个元素值，汇总出一个最终结果

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

//prev 截止目前的临时汇总值

return prev+val;

},prev)

ES6:

1. let和块作用域

问题1: hoist，打乱了程序的正常执行顺序

解决: 今后用let代替var声明变量

原理: let通过禁止在当前作用域内提前使用未声明的变量，来避免声明提前

强调: 如果已经用let声明的变量，不能再用var重复声明

问题2: js没有块级作用域, 块内的变量即使不执行，也会影响块外的变量:

块级作用域: 程序中的一对儿{}之间，称为代码块

js vs java: Java 三级作用域: 全局，函数，块

js 两级作用域: 全局，函数，没有块级作用

解决: 用let代替var

规定: let声明的变量仅在当前块内有效！

原理: 自动在块作用域位置创建匿名函数自调，划分临时作用域。

问题3: 原来在for/while/if...内声明的变量，出了结构，就不能使用

解决: 今后，凡是出了结构，还要使用的变量，必须在结构外提前声明。

总结: 今后所有变量都要用let声明，而不var

2. 箭头函数: 简化回调函数

何时: 今后所有回调函数都可用箭头函数简化！

如何:

1. 去function改箭头

更简化:

2. 如果只有一个参数变量，可省略()

3. 如果函数体只有一句话，可省略{}

如果仅有一句return，则可省略return

箭头函数特点: 内容this通用！

3.参数增强:

Default:(默认值)

function fun(参数1,参数2,参数3=默认值,参数4=默认值){}

调用时，如果传给参数的值有效，就用传入的值

如果没有传入参数，或传入的参数无效，则用提前指定的默认值

REST:(剩余参数列表) 为了代替arguments

arguments的问题:

1. 不是真正的数组类型，无法使用数组类型的API

2. 默认只能获得所有参数值，无法仅选取部分参数值

解决: REST

何时: 今后，都是用REST语法代替arguments

如何:

定义函数: function fun(参数1,参数2,...剩余参数的统称)

调用时: ...剩余参数的统称 会获得一个数组

SPREAD:(散播) 为了代替apply打散数组类型的参数

apply的问题: 1. 必须传入第一个对象参数，代替this

2. 必须将所有参数都放入数组中

解决: spread

何时: 今后只要打散数组类型参数时，首选spread

如何: 定义函数时，参数变量是分开定义的

调用函数时，可用fun(值1,...arr)，打散arr数组

4. 模板字符串: 简化复杂格式的字符串拼接

何时: 今后所有字符串拼接，都用模板字符串代替+

如何:

1. 用反引号`包裹整个字符串

2. 在模板字符串内，自动支持换行

3. 在模板字符串内，支持动态生成表达式的值

要执行的表达式，必须用${}包裹

5. 解构: 简化复杂对象的打散和批量赋值

1. 数组解构: 等号左右两边都是数组格式

解构时，用下标对应值和变量

2. 对象解构: 等号左右两边都是对象格式

解构时，用属性名匹配值和变量

3. 参数解构: 其实，在调用函数时，如果传入的参数值是数组或对象，也可被解构为单个参数值

用法和数组结构和对象结构完全一样

6: class: 为了简化面向对象:

封装，继承，多态，访问器属性，静态方法

class: 集中定义构造函数和原型对象方法的程序结构

为什么: 更像封装

何时: 只要创建一种新类型，必须先定义class

如何: class 类型名{

constructor(...){

this.xxx=xxx;

}

原型对象方法(){//类型名.prototype.xxx()

... this.xxx ...

}

}

继承: 2步:

1. 子类型构造函数中借用父类型构造函数:

super(参数值)

super自动指代父类型的构造函数

2. 让子类型继承父类型

class 子类型 extends 父类型

不用再Object.setPrototypeOf

访问器属性:

class 类型名{

constructor(...){

this.\_age=age

}

get age(){return this.\_age}

set age(val){

... ...

}

}

Day8

正课:

ES6

Set 集合:

什么是: 值不允许重复的集合

何时: 快速去重复

通过元素值，快速判断是否包含元素时

如何: var set=new Set(凡是可遍历的，都可打散转为Set)

将set转回数组: var arr=[...set];

Map 集合:

什么是: 一组键值对儿的集合

何时: 代替关联数组使用

如何: var hash=new Map(); //{}

hash.set(key,value); //hash[key]=value

var bool=hash.has(key); //hash[key]!==undefined

for of : 代替for循环遍历

什么是: 依次获得数组/类数组对象/集合中每个值

何时: 只要不关心位置，只关心每个元素值时

如何: for(var value of 数组|类数组对象|集合){

value: 自动获得当前元素值

}

问题: 1. 无法获得位置

2. 必须从头到尾，逐个执行，不能跳跃或遍历部分

3. 只能从头到尾，顺序遍历，不能反向遍历

\*\*\*\*Promises

什么是: 解决"callback hell"

为什么: 回调函数，层级很可能很深，可读性极差

何时: 今后，只要多个回调函数，必须保证顺序执行时

如何: 2步:

1. 在前一个函数内

return new Promise((callback)=>{

... callback() ...

})

2. 在调用前一个函数时,

前一个函数.then(()=>{ 后一个函数(); })

错误处理:

1. 在前一个函数内

return new Promise(resolve,reject)=>{

如果没出错，执行resolve

如果出错，执行reject("错误提示")

})

2. 在调用前一个函数时,

前一个函数.then(()=>{ 下一个函数(); })

.catch((err)=>{ 输出err })