张东

微信: 18301092802

准备:

1. 书: 犀牛书 —— JavaScript权威指南

2. 微信公众号: 前端大全

3. 上届笔记: 打印成纸质

不允许记笔记！

正课:

1. 正则表达式:

2. String的正则API:

1. 正则表达式:

正则表达式: Regular Expression

什么是: 规定一个字符串中内容出现规律的规则

何时: 2种:

1. 格式验证:

2. 模糊查找关键词:

如何:

1. 最简单的正则表达式，是一个关键词的原文

2. 字符集:

什么是: 规定一位字符上多种备选字列表的集合

何时: 只要一位字符上有多种备选字时

如何: [备选字列表]

强调: 一个字符集只能匹配一位字符

简写: 如果备选字符是连续的，可用-省略中间字符

比如: 1位数字: [0-9]

1位小写字母: [a-z]

1位大写字母: [A-Z]

1位字母: [A-Za-z]

1位字母或数字: [A-Za-z0-9]

1位汉字: [\u4e00-\u9fa5]

需求: 车牌号:

1位汉字 1位大写字母 中点 五位大写字母或数字

[\u4e00-\u9fa5][A-Z]•[A-Z0-9] [A-Z0-9] [A-Z0-9] [A-Z0-9] [A-Z0-9]

排除: [^47] 除了4和7之外都行

强调: 范围太大！

3. 预定义字符集: 4种:

1位数字: \d 0~9

1位字母数字或下划线: \w 等效于[A-Za-z0-9\_]

1位空字符: \s 包括: 空格，tab… …

通配符: .

问题: 单靠字符集，无法灵活定义字符的个数

解决: 用量词

4. 量词:

什么是: 规定一位字符出现次数的规则

何时: 只要定义一位字符出现的次数

如何: 字符集量词

放在字符集之后，修饰相邻的前一个字符集

包含: 2大类:

1. 有明确数量边界: 3种:

{n,m} 最少n个，最多m个

{n,} 最少n个，多了不限！

{n} 必须n个，不能多也不能少

2. 没有明确数量边界: 3种:

? 可有可无，最多一个

\* 可有可无，多了不限

+ 至少一个，多了不限

问题: 量词默认仅修饰相邻的前一个字符集

解决: 分组

5. 分组:

什么是: 用()将多个字符集包裹为一组

何时: 只要希望一个量词能修饰多个字符集时

身份证号:

15位数字 2位数字 1位数字或Xx

后三位整体可有可无，最多1次

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

6. 选择: 或

什么是: 在多个条件中任选其一匹配

何时: 只要在多个条件中任选其一匹配即可

如何: 规则1|规则2

作业: 编写正则，检查: 微 信 wei xin w x

7. 特殊位置: 3个:

字符串开头 ^

字符串结尾 $

何时: 仅匹配开头或结尾的关键词时

比如: 仅匹配开头的空字符: ^\s+

仅匹配结尾的空字符: \s+$

同时匹配开头或结尾空字符: ^\s+|\s+$

单词边界 \b

凡是能分割两个单词的字符

包括: 开头, 结尾，空格，标点…

比如: 匹配每个单词首字母: \b[a-z]

8. 匹配相同内容: 2步

1. 给第一个关键词加()

2. \1 代表和()中相同的内容

比如: 匹配连续的多个相同字母/字符

([a-z])\1\*

2. String 的正则API:

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",starti)

从str中starti位置开始，找下一个"关键词"的位置i

如果找到，返回关键词第一个字所在的位置i

如果找不到，返回-1

问题: 不支持正则，只能查找一个固定的关键词

2. 模糊查找一个关键词的位置:

var i=str.search(/正则/)

在str中从0位置开始找下一个符合"正则"要求的敏感词的位置i

如果找到，返回关键词第一个字的位置

如果没找到，返回-1

问题: 正则表达式默认区分大小写

解决: 在第二个/后加后缀i,表示ignore忽略大小写

更高要求: 既返回关键词内容，又返回关键词位置

3. 模糊查找关键词，并返回关键词内容: 2种:

1. 只返回一个关键词的内容:

var arr=str.match(/正则/i)

在str中查找符合/正则/要求的关键词，并同时返回关键词的内容和位置

返回值: arr[ "0": 关键词, "index": i ]

数组本质: 所有数组底层其实都是关联数组

获取关键词内容: arr[0] ~~arr.0~~

获取关键词位置: arr["index"] <==> arr.index

如果找不到，返回null

正则表达式第2个问题: 只找一个就结束了！

解决: 在第2个/后加g，global 全部

2. 查找所有关键词的内容(不关心位置):

var arr=str.match(/正则/g)

查找str中所有符合/正则/要求的关键词保存到arr中

返回值: 所有关键词组成的数组

找不到: 返回null

4. 既查找每个关键词的内容，又查找每个关键词的位置:

match()不能实现:

match()不加g时，只能找1个关键词

match()加g后，不能获得位置

解决: RegExp对象的exec()方法

替换关键词: 2种:

1. 简单替换: 所有关键词都替换为统一的新值

str=str.replace(/正则/ig,"新值")

回顾: 所有字符串API都无权修改原字符串，只能返回新字符串！所以必须用变量才能借助修改后的新值。

将str中所有符合/正则/要求的关键词都替换为统一的"新值"

2. 高级替换: 根据找到的每个关键词的不同，动态返回对应的新值替换。

str=str.replace(

/正则/ig, function(kword){ return 新值; }

)

原理: 1. replace查找str中符合条件的每个关键词

2. 每找到一个关键词，就自动调用回调函数

3. 每次调用函数都自动传入本次找到的关键词

4. 回调函数根据本次传入的关键词，加工后生成新替换值返回

5. replace将本地回调函数返回的值，替换到正在遍历的位置。

切割字符串:

正课:

1. String的正则API
2. RegExp
3. Function

\*\*\*\*\*闭包

作业:

(微|w(ei)?)\s\*(信|x(in)?)

1. String的正则API

替换: 衍生: 删除关键字: 替换为空字符串

切割: 将一个字符串，按照切割符，切割成多段子字符串

1. 简单切割: 切割符是固定的

比如: [zhangdong@tedu.cn](mailto:zhangdong@tedu.cn)

no zuo no die

var arr=str.split(“固定的切割符”)

返回多段子字符串组成的数组

固定套路: 打散一个字符串为字符数组

var chars=str.split(“”)

1. 复杂切割: 切割符不是固定的

var arr=str.split(/正则/);

2. RegExp对象:

什么是: 专门保存一条正则表达式，并提供用正则表达式执行验证和查找功能的API 的对象

何时: 只要在程序中保存一条正则表达式或使用正则表达式执行验证和查找时

如何:

创建: 2种:

1. 用//: var reg=/正则/ig

问题: 正则是写死的，无法再运行时动态生成正则表达式

1. 用new: var reg=new RegExp(“正则”,”ig”)

好处: 参数是一个字符串格式的正则表达式，所以可以用任何程序的手段先动态生成字符串，再传给new RegExp变为正则表达式对象——动态生成正则

API:

1. 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)

验证str是否符合reg的格式要求

问题: test其实是只要找到符合规则的，就返回true，不要求从头到尾完整匹配

比如: /\d{6}/.test(“123456a”) -> true

解决: 今后只要验证，必须前加^，后加$，表示必须从头到尾完全匹配才行

比如: /^\d{6}$/.test(“123456a”) -> false

其实: reg.test()也可判断一句话中是否包含敏感词，且更简单。

比如: if(reg.test(msg)) 如果msg中包含符合reg的敏感词

1. 查找: 既查找每个关键词的内容，又查找每个关键词的位置

var arr=reg.exec(str)

让reg去str中找下一个符合规则的关键词的内容和位置

返回值: 和str.match()不加g的情况是一样的

arr[“0”: 关键词, “index”: i ]

如果找不到了，返回null

vs str.match() :

str.match()始终只能找第一个关键词

reg.exec()反复调用时，可自动向后找下一个

总结: 查找关键词: 5种

1. 只判断是否包含: var bool=reg.test(str)
2. 查找一个固定关键词的位置: var i=str.indexOf(“关键词”)
3. 查找一个可变关键词的位置: var i=str.search(/正则/i)
4. 查找第一个关键词的内容和位置:

var arr=str.match(/正则/i)

arr[“0”:关键词, “index”: i ]

1. 查找所有关键词的内容:

var arr=str.match(/正则/ig)

arr[关键词1, 关键词2, … ]

1. 查找所有关键词的内容和位置:

do{

var arr=reg.exec(str);

if(arr!=null)

arr[“0”:关键词, “index”: i ]

else

break;

}while(true);

3. Function

什么是: 保存一段可重用的代码段的程序结构，再起一个名字

何时: 如果一段代码可能被反复使用

程序的重要设计原则之一: DRY Don’t Repeat Yourself

如何:

创建: 3种:

1. 用声明方式:

function 函数名(形参列表){ 函数体; return 返回值 }

问题: 会被声明提前(hoist)

回顾: 声明提前: 当程序开始执行前，先将var声明的变量和function声明的函数提前到当前作用域的顶部，集中创建。赋值留在原地。

习惯: 所有声明都集中放在当前作用域的顶部。

1. 用赋值方式:

var函数名=function (形参列表){ 函数体; return 返回值 }

揭露了js中函数的本质: 函数其实是一个保存代码片段的对象，函数名其实就是一个普通的变量。

不会被声明提前！

1. 用new:

var 函数名=

new Function(“形参1”,…,”函数体; return 返回值”)

重载(overload):

什么是: 多个相同名称不同参数列表的函数，调用时可自动根据传入参数的不同，选择对应的函数执行。

为什么: 减少API的个数，减轻调用者的负担

何时: 1件事，可能根据传入参数不同，执行不同的处理过程时

如何:

问题: js语法默认不支持重载效果

为什么: js中不允许多个同名函数同时存在

如果同时存在，最后一个同名函数会覆盖之前所有

解决: 变通实现

1. 只定义一个函数，不定义参数列表
2. 调用函数时，按需传入参数
3. 所有参数都会被函数中的内置对象arguments自动接收。

arguments: 每个函数中自动创建的可接受所有实参列表的类数组对象(长的像数组的对象)

vs 数组: 相同: 1. 下标, 2. .length, 3. 遍历

不同: 不是数组类型，API不通用！

1. 函数中根据参数列表的个数和内容，动态选择执行一种处理逻辑

匿名函数:

什么是: 定义函数时不指定函数名的函数

为什么: 节约内存

匿名函数用完后，不再被任何变量引用着，就被垃圾回收！

何时: 如果一个函数只用一次时

如何: 2个场景:

1. 回调函数:

什么是: 自己定义函数，却给别人去调用

1. 自调函数:

作业:

function add(){

}

console.log(

add(1,2,3), //6

add(1,2,3,4,5) //15

);

正课:

1. Function
2. 面向对象
3. Function

匿名函数:

如何:

1. 回调
2. 自调：函数定义后，立刻自己调用自己执行一次

为什么: 避免使用全局变量，避免全局污染

何时: 用匿名函数划分临时作用域，避免使用全局变量

如何: (function(){ … })()

//function是创建一个函数的意思——动作

//返回函数地址，比如0x1234

//0x1234() 调用匿名函数

总结: 今后，禁止使用全局变量！

今后，所有js代码一律放在一个大的匿名函数自调内！

作用域和作用域链: PPT

* JavaScript程序中，也包含两级作用域：
  + 1. 全局作用域，指从任何位置都可访问的范围
  + 2. 函数作用域，指仅限于函数内才可访问的范围
* 作用域链，就是由多级作用域组成的链式结构
* 当调用函数时，函数会将自己能用到的所有作用域都用“锁链”串联（引用）起来。
* 作用域链：
  + 1. 串连着当前函数可使用的所有作用域范围，保存着当前函数可用的所有变量
  + 2. 控制着变量的使用顺序：先局部，后全局；只要局部有，就不去全局找。

闭包: PPT

* 闭包，是一个特殊的作用域对象，也称“执行上下文” ：
  + 是外层函数使用封闭的方式留给内层函数的专属作用域对象
  + 专门用于为一个函数保护专属变量。
* 今后，只要为一个函数，保护专属的变量时，都用闭包
* 第一步: 用外层函数包裹内层函数和要保护的变量
* 第二步: 外层函数将内层函数返回到外部
* 第三步：调用外层函数，获得内层函数，保存到全局变量

2. 面向对象:

什么是: 程序中描述现实中一个具体事物的属性和功能的程序结构

为什么: 便于大量数据的管理和维护

何时: 今后，所有程序都要用面向对象思想实现

如何: 三大特点: 封装，继承，多态

1. 封装:

什么是: 用对象结构集中存储一个事物的所有属性和功能

为什么: 便于大量数据的管理和维护

何时: 今后，只要使用面向对象思想，都要先将零散的数据封装为对象

如何: 3种:

1. 用{}创建对象:

var obj={

属性名:值,

… : …,

方法名:function(){

… this.属性名 …

}

}

访问对象成员(属性+方法): 对象.成员名

问题: 对象自己的方法内，要使用当前对象自己的属性，不能直接写属性名——报错！未定义

原因: 对象不是作用域！所以不在方法的作用域链上，所以，即使自己的方法，也不能直接访问自己对象的属性

解决:

不好的解决: 写死“对象名.属性名”

问题: 如果对象名发生改变，被迫同时修改方法内的语句

好的解决: “this.属性名”

\*\*\*\*\* this: 自动引用正在调用当前方法的 .前 的对象

何时: 只要对象自己的方法，要访问自己的属性时，必须加this.

常犯错误: 以定义方法的位置来判断this的指向。

正确: this与定义的位置无关，只与调用时.前的对象有关。2种:

1. obj.fun() this->obj
2. fun() this默认->window

2. 用new:

var obj=new Object();

obj.属性名=值;

obj.方法名=function(){ … }

js中对象的本质: js中一切对象底层都是关联数组

访问对象的成员: 2种:

obj[“属性名”]=obj.属性名

如果属性名是动态变化的，必须用obj[表达式]

访问对象中不存在的属性: 不报错！返回undefined

随时可给对象添加任何新属性: 不报错！而是自动创建

前两种方式: 一次只能创建一个对象

如果反复创建多个相同结构的对象时，代码大量冗余——不便于维护

3. 用构造函数创建对象:

什么是构造函数: 描述一类对象相同结构的函数

为什么: 代码重用

何时: 反复创建多个相同结构的对象时

如何: 2步:

1. 定义构造函数:

function 类型名(){

this.属性名=值;

… = …;

this.方法名=function(){

… this.属性名 …

}

}

1. 用构造函数反复创建多个相同结构的对象

var obj=new 类型名();

new: 4件事:

1. 创建一个空对象
2. ？
3. 调用构造函数，并将构造函数中的所有this，自动替换为新的空对象

通过强行给新对象赋值的方式，向新对象中添加新属性

1. 返回新对象地址

正课

1. OOP
2. ES5
3. OOP

封装:

构造函数的问题：如果将方法定义放在构造函数中，会导致每次创建新对象时，都重复创建方法的副本——浪费内存

解决: 保存在一处，共用！

继承:

什么是: 父对象的成员，子对象无需重复创建，就可直接使用

为什么: 代码重用，节约内存

何时: 所有子对象，需要相同的方法或属性时，只要在父对象中添加一次，所有子对象就可共用

如何: js中继承，都是继承原型对象(prototype)

创建原型对象: 不用创建，买一增一:

每创建一个构造函数，都附赠一个空的原型对象

构造函数.prototype=自己的原型对象

何时继承:

new的第2步，自动设置继承关系

子对象.\_\_proto\_\_=构造函数.prototytpe

如何向原型对象中添加共有成员: 强行赋值:

构造函数.prototype.共有成员=值

11种内置对象，也都有原型对象:

将来只要能new的都是一种类型

只要是一种类型，都包含2部分组成:

1. 构造函数: 专门用于创建该类型的子对象
2. 原型对象:专门用于保存该类型的子对象共有的成员

比如: Array, Date, RegExp, Function, …

原型链:

什么是: 由多级父元素逐级继承形成的链式结构

保存着: 所有对象可访问的属性和方法

控制着属性和方法的使用顺序: 先用自己的，自己没有才延原型链向上找

多态:

什么是: 一个函数在不同情况下表现出不同的状态

包括: 2种:

1. 重载:
2. 重写(override)

什么是重写: 子对象觉得父对象的成员不好用，可在本地定义自有同名成员，覆盖父对象的成员

何时: 只要觉得父对象的成员不好用，都可重写一个新的

如何: 只要在子对象本地定义同名成员即可

静态方法与实例方法:

什么是实例方法: 给某个类型的子对象调用的方法

何时: 如果要求，方法执行时，必须先有一个具体的子对象，才能用。

如何: 要么直接定义在对象上，要么定义在对象的原型对象上

什么是静态方法: 不需要具体对象，也可直接执行的方法

何时: 如果不需要任何具体对象，也想调用函数时

如何: 将函数定义在构造函数对象上

构造函数.静态方法=function(){ … }

调用时: 构造函数.静态方法();

自定义继承关系:

1. 只修改一个对象的父对象:

child.\_\_proto\_\_=father

Object.setPrototypeOf(child,father)

设置set child 的of 爹prototype 为 father

问题: 一次只能改一个子对象的爹

1. 同时修改所有子对象的原型对象

其实就是修改构造函数的prototype属性

构造函数.prototype=father

时机: 必须在定义完构造函数后，开始创建子对象之前，就要更换。

1. 正式的两种类型间的继承:

问题: 两种类型间包含了部分相同的属性结构和方法定义

解决: 抽象父类型:

2步:

1. 定义抽象父类型:
2. 在父类型构造函数中包含相同的属性定义
3. 在父类型的原型对象中包含相同的方法定义
4. 删除子类型中相同的内容
5. 让子类型继承抽象父类型:
6. 让子类型构造函数继承父类型构造函数

Object.setPrototypeOf(

子类型.prototype, 父类型.prototype

)

1. 让子类型构造函数借用父类型构造函数

错误: 直接调用父类型构造:

父类型(参数值)

后果: 父类型构造函数中的this->window

解决: 函数内的this不是想要的，都可用.call()的方式，将正确的this注射进函数内，替换其中错误的this

正确的方法:

父类型构造.call(正确的this,参数值..)

结果: 执行时，父类型构造 中的this，被外部正确的this给替换了！

2. ES5:

严格模式: 比普通js运行机制要求跟严格的执行模式

何时: 今后，所有项目代码，都必须运行在严格模式下！

如何: 在当前作用域顶部: “use strict”;

如果加载<script>元素或js文件顶部，则整段js代码都启用严格模式

如果只加在函数内的顶部，则仅函数内启用严格模式

要求:

1. 禁止给未声明的变量赋值:

普通js中，可以给未声明的变量赋值

结果: 会自动在全局创建该变量

并且造成误会！

1. 静默失败升级为错误:

静默失败: 执行不成功，也不报错！

1. 普通函数调用/匿名函数自调中的this，不再指向window，而是undefined
2. 屏蔽了arguments.callee

什么是arguments.callee: 函数内自动创建的，引用当前函数本身的关键字。

何时: 在使用递归算法时，可在函数内避免写死当前函数名。

屏蔽arguments.callee是在屏蔽递归算法

为什么: 递归算法效率极低，重复计算量太大！

解决: 几乎所有递归都可用循环代替！

保护对象:

保护单个属性:

ES5将对象属性划分为两大类:

命名属性: 凡是可用.直接访问到的属性

又被分为两大类:

数据属性: 直接存储属性值的属性

访问器属性: 不直接存储属性值，专门提供对另一个数据属性的自定义保护

内部属性: 不能用.直接访问的属性

保护数据属性:

ES5将每个数据属性，都变成一个缩微的小对象：

{ //四大特性:

value: 实际存储属性值,

writable: true/false, //控制是否可修改

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

但是，用.依然可强行访问该属性

configurable: true/false, //控制是否可删除该属性

//控制是否可修改另外两个特性，一旦改为false，不可逆

设置前两个属性时，顺带设置configurable为false。

}

保护对象结构:

正课:

1. ES5
2. ES6
3. ES5

保护对象

保护属性

数据属性:

四大特性: value, writable, enumerable, configurable

获取: var 属性对象=

Object.getOwnPropertyDescriptor(obj,”属性名”)

修改四大特性:

Object.defineProperty(obj,”属性名”,{

开关: true/false,

… : …

})

问题: 一次只能修改一个属性的四大特性

Object.defineProperties(obj,{

属性名:{

开关: true/false,

… : …

},

属性名:{

开关: true/false,

… : …

}

})

问题: 无法使用自定义的规则保护属性

访问器属性: 访问器属性是不保存属性值，仅提供对另一个数据属性的保护——保镖

何时: 使用自定义规则保护属性时

如何:

前提: 先有一个实际存储数据的隐藏的数据属性

定义访问器:

Object.defineProperties(obj,{

\_数据属性:{ enumerable:false,configurable:false},

访问器属性:{

get:function(){ return this.\_数据属性},

set:function(value){

if(value符合规则)

this.\_数据属性=value;

else throw new Error(“自定义错误信息”)

},

enumerable:true,

configurable:false

}

})

保护对象的结构: 3个层次:

1. 防扩展: 禁止给对象添加新属性

Object.preventExtensions(obj)

阻止 扩展

设置obj对象，禁止扩展新属性

1. 密封: 在防扩展基础上，再禁止删除所有属性

本质: 在设置防扩展同时，自动设置所有属性的configurable为false

如何: Object.seal(obj)

1. 冻结: 在密封基础上禁止修改属性值

本质: 在密封基础上，自动将所有属性的writable设置为false！

如何: Object.freeze(obj)

Object.create(): 没有构造函数，也想创建子对象

如何: var child=Object.create(father,{

//defineProperties的语法

属性:{ 四大特性 }

})

创建新对象child，然后让child继承father，并为新对象添加自有属性

替换this:

.call/apply/bind

1. 调用一次函数，并临时替换函数中的this为指定对象：.call/apply

fun.call(obj,实参…)

2件事: 1. 调用一次函数执行

2. 在执行时临时替换其中的this为obj

.apply可打散数组为单个参数值后，分别传入函数:

如果函数需要多个参数值，但给的参数值却集中保存在一个数组中。

1. 基于原函数创建一个完全相同的新函数，并永久绑定this为指定对象

var newFun=fun.bind(obj,实参…)

创建一个和fun完全相同的新函数newFun

永久将newFun中的this绑定为obj

永久将newFun中的部分参数绑定为实参…

固定用法: 如果替换回调函数中的this，必须用.bind()

因为回调函数不是立刻执行！且不止执行一次！

数组API: 7个:

1. 查找: 查找一个指定的元素在数组中的位置

同字符串操作: var i=arr.indexOf(元素, starti)

1. 判断: 判断数组中的元素是否符合要求

2种:

1. 判断是否所有元素都符合要求：

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

//elem: 当前元素值

//i: 当前位置

//arr: 当前数组

return 判断条件;

})

every会拿着回调函数去每个元素上执行一次

每次执行时，都会将当前元素值，当前所在位置，当前数组对象，传入函数中

函数内用提供的参数值，判断当前元素是否符合条件，并返回判断结果

必须回调函数在每个元素上执行时，都返回true，整体判断结果才返回true！

1. 判断是否包含符合要求的元素:

var bool=arr.some(function(elem,i,arr){

//elem: 当前元素值

//i: 当前位置

//arr: 当前数组

return 判断条件;

})

遍历: 依次对数组中每个元素执行相同的操作

1. 对原7题喏23数组中的元素执行相同的操作：

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

操作

})

会自动拿着回调函数，在每个元素上执行一次！

2. 复制出原数组中的元素，执行相同操作后，放入新数组中，原数组保持不变。

var 新数组=arr.map(function(elem, i, arr){

return 新元素值;

})

map: 3件事:

1. 创建一个新的空数组

2. 自动在每个元素上调用一次回调函数，将返回的新值，放入新数组中相同位置

3. 返回新数组

4. 过滤和汇总:

1. 过滤: 复制出原数组中符合条件的元素，组成新数组返回

var 新数组=arr.filter(function(elem,i,arr){

return 判断条件;

})

执行过程: filter会自动拿着回调函数在每个元素上执行一次

每次执行时，如果当前元素符合“判断条件”要求，就复制到新数组中。

2. 汇总: 将数组中的所有元素，经过统计，得出一个最终结论

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

return 将当前元素的内容汇总到prev中

},start);

//start: 开始的基础值

//prev: 截止目前位置，之前的元素的临时汇总值

正课:

1. ES5
2. ES6
3. ES5

数组API:

遍历: 对每个元素都执行相同的操作

1. 直接修改原数组

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

elem…

arr[i]…

})

forEach会拿着回调函数去每个元素上执行一次

每次执行时，回调函数内都可用arr[i]修改原数组中的每个元素值。

1. 不直接修改原数组，而是返回新数组

var newArr=arr.map(function(elem,i,arr){

return 新值

})

map先创建一个新的空数组

然后拿着回调函数去每个元素上执行一次

每次执行时，都将回调函数返回的新值，放入新数组中相同位置

最后，将保存新值的新数组返回，原数组保持不变

过滤: 复制出原数组中符合条件的元素，组成新数组

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

return 条件

})

filter先创建一个新的空数组

会拿着回调函数去每个元素上执行一次

每次执行时，都获得当前元素的判断结果

只有返回结果为true的元素，才被放入新数组中返回

汇总: 对数组中的元素进行各种统计，得出最终结论

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

//prev: 截至到当前元素之前的临时汇总值

return prev+elem;

},base)

2. ES6: 不改变原理的基础上，尽量简化了API

let/const: 代替var/const来声明变量和常量的

为什么:

1. 因为传统的var和const会被声明提前
2. 没有块级作用域: for if while都不是作用域，其中的变量会被提前到外部，影响外部的代码。

使用后:

1. 不会被提前，不允许提前使用未声明的变量
2. 让for if while等程序块，也变为作用域

原理: let自动添加匿名函数自调来划分临时作用域，并将变量名前自动加\_，与其他变量区分。

参数增强:

1. 默认值(default): 即使用户不传入参数，参数也有备用的默认值代替

如何: 定义函数时: function fun(形参,…,形参=默认值)

强调: 有默认值得参数，必须在形参列表的末尾

1. 剩余参数(rest): 代替arguments接收所有不确定个数的参数

arguments: 2个问题:

1. 不是数组类型，不能使用数组API
2. 只能获得全部参数，无法有选择的获得部分

如何: 定义函数时: function fun(形参,…, …数组名)

其中: 数组可收集除前几个确定参数之外的剩余参数

1. 散播(spread): 代替apply专门用于打散数组为单个元素，再传入函数。

apply的问题: 主要功能是替换this，顺便可以打散数组为单个值。

如何: 调用函数时: fun(…数组)

箭头函数: 对一切回调函数或匿名函数自调的简写

如何: 回调函数: function (形参,…){ … }

可简化为: (形参,…)=>{ … }

如果只有一个形参，可省略()

如果函数体中只有一句话，可省略{}

如果这仅有的一句话还是return xxx，则必须省略return

特点: 箭头函数内外的this是同一个/共通的

总结: 如果希望内外this相同时，应该简化

如果反而希望内外this不同时，不能简化！

解构: 从一个大的对象中抽取想要的部分成员，单独使用

3种:

1. 数组解构: 从数组中抽取想要的元素出来，单独使用

如何: 下标对下标:

var arr=[1,2,3];

↓

var [x,y,z]=arr

var x,y,z

结果: x=1, y=2, z=3

arr[0]//麻烦，且没有意义

1. 对象解构: 从对象中抽取想要的成员出来，单独使用

如何: 属性名对属性名

var obj={x:1, y:2, z:3}

↓ ↓ ↓

var {x:a, y:b, z:c}=obj;

obj.x obj.y obj.z //麻烦

结果: a=1, b=2, c=3 简写: var {x:x, y:y, z:z}=obj;

可简写为var {x,y,z}=obj;

正课:

1. ES6
2. ES6

解构:

参数解构:

何时: 当参数列表中有多个参数值不确定时

问题: 形参列表中有多个参数不确定是否有值

解决: 参数解构:

如何: 2步:

1. 定义函数时: 形参列表使用对象语法

2. 调用函数时：也使用对象语法传参

for of: 最简化的遍历索引数组或类数组对象的方式：

总结: 遍历索引数组:

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

var elem=arr[i];

万能

1. arr.forEach((elem,i,arr)=>{

})

无法控制遍历顺序和步调

1. for(var elem of arr){ … }

of会依次取出arr中每个元素值，保存在of前的变量中

无法获得位置i

无法修改原数组中的值

for of vs for in

for of 专门遍历下标为数字的索引数组/集合/字符串

of取得是元素值

for in 专门遍历下标为自定义名称的关联数组/对象

in取得是属性名

class:

什么是: 集中描述一类对象统一属性结构和行为的程序结构

何时: 今后只要创建一种类型，必须用class

如何:

1. 用class{}包裹原来的构造函数和原型对象方法
2. 构造函数名提升为class名，构造函数要更名为constructor
3. 直接定义在class内的方法，默认保存在原型对象中，且不用加Xxx.prototype前缀和”=function”

继承: 2步

1. class child extends father{}

不再需要Object.setPrototypeOf(…)

1. 子类型构造函数中: super(参数值)//不需要传入this

super自动指向extends后的父类型

super()调用父类型构造函数的意思

访问器属性:

1. 依然要在构造函数内定义受保护的隐藏的数据属性
2. 在class内:

get 访问器属性名(){ return this. …}

set 访问器属性名(value){ 验证value并给this.xxx赋值}

静态方法:

class Xxx{

static 方法名(){ }

}

Promise: 代替回调函数，实现多个异步调用，顺序执行

问题: 使用回调函数方式实现多个异步调用顺序执行会导致回调地狱(callback hell)

根源: 所有回调函数规定，在调用函数前，就要提前传入到函数中

解决: 让回调函数在函数后传入

如何:

前提: 不要在参数列表里传递回调函数了！

1. 定义函数支持Promise

在原函数内，用new Promise()包裹所有原代码

再在源代码外层套一层function(){}

function()中必须接收Promise附赠的open开关

在当前函数异步任务调用后，自动打开开关open();

function fun(){

return new Promise(function(open){

异步任务

异步任务执行完:open()

})

}

1. 将多个任务串联起来:

第一个函数().then(第二个函数).then(…)

强调: 中间的then中的函数，不要加()，因为不是立刻执行，且中间的函数必须支持Promise

错误处理: 2步:

1. new Promise(function(open,err){

//如果出错:

err(“错误消息”) //通向最后的.catch()

})

1. 在函数1().then().then()…..catch(function(errMsg){ … })

无论中间哪个then出错，都会执行最后的catch，并将then中err(“错误消息”)传给errMsg。