**JSCore**

Day01

张东

zhangdong@tedu.cn

准备:

1. 犀牛书: JavaScript权威指南

2. 微信公众号: 前端大全

3. 上届笔记/思维导图

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: Regular Expression

什么是: 规定字符串中字符出现规律的规则

比如: 手机号: 1 [345678] 9位数字

电子邮件:

任意字符任意个@任意字符任意个.com|.cn|.net,

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就是关键词本身

2. 字符集:

什么是: 一位字符上，备选字符列表的集合

为什么: 有些字符可能发生变化！

何时: 只要一个字符有多种备选情况时

如何: [备选字列表]

强调: 一个[]只能匹配一位字符！

简写: 如果备选字符列表是连续的，可简写为[x-x]

比如: 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]

车牌号: 1位汉字 1位大写字母 · 五位大写字母或数字

3. 预定义字符集:

什么是: 对常用字符集的简化

包括: \d 一位数字

\w 一位字母数字或下划线 [0-9A-Za-z\_]

\s 一位空字符: 空格 tab ...

. 通配符，可匹配所有字符

何时: 如果规则完全匹配时，才能使用预定义字符集,

如果规则不完全匹配，则退一步用自定义[字符集]

问题: 一个字符集只能规定一位字符的变化

无法灵活定义出现的次数

4. 量词:

什么是: 专门规定一个字符集出现次数的规则

何时: 只要定义一位字符集出现的次数

如何:

必须放在一个字符集之后，修饰相邻的前一个字符集

包括2大类:

1. 有明确数量边界:

{n,m} 至少n次，最多m次

{n,} 至少n次，多了不限

{n} 必须n次

比如: qq号 \d{5,11}

2. 没有明确数量边界的

\* 可有可无，多了不限

? 可有可无，最多一次

+ 至少一次，多了不限

5. 选择和分组:

选择: 其实就是"或"

何时: 如果在多个规则之间选其一匹配即可

如何: 规则1|规则2

分组: 将多个规则合并为一组

为什么: 默认一个量词只能修饰相邻的前一个字符集

何时: 希望一个量词可以修饰之前多个字符集时

如何: (多个规则)

比如: 电子邮件地址:

任意字符任意个@任意字符任意个.com|cn|net .cn

.\*@.\*\.(com|cn|net)(\.cn)?

问题: .\* 或 .+ 容易发生贪婪模式

什么是: 正则表达式默认会匹配最长的符合条件的字符串

解决1: 懒惰模式: 仅匹配最短的符合条件的字符串

如何: (.\*?)

[^.@]+@[^.@]+\.(com|cn|net)(\.cn)?

解决2: 限制字符: [^xxx] ^称为"除了"

[^.@]+@[^.@]+\.(com|cn|net)(\.cn)?

比如: 手机号:

+86或0086 空格 1 [3-8] 9位数字

(\+86|0086)?\s\*1[3-8]\d{9}

比如: 我草 卧槽 我操 wo cao

我([艹草操槽]|cao)

比如: 身份证号:

15位数字 2位数字 1位数字或x

后三位整体可有可无，最多一次

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

作业: 微 信 wei xin w x

6. 匹配特殊位置:

3种:

字符串开头: ^

字符串结尾: $

何时: 只要仅匹配开头的或结尾的字符时

比如: 开头的空字符 ^\s+

结尾的空字符 \s+$

同时匹配开头或结尾的空字符: ^\s+|\s+$

单词边界: \b

包括: ^ $ 空格 标点符号

比如: 每个单词首字母: \b[a-z]

选择一个单词no : \bno\b

2. String的正则API:

1. 查找关键词: 4种:

1. 查找一个固定的关键词的位置:

var i=str.indexOf("关键词"[,fromi])

var i=str.lastIndexOf("关键词")

问题: 不支持正则，只能查找一个固定的关键词

2. 用正则查找一个关键词的位置:

var i=str.search(/正则/) /xxx/是正则表达式的标识

返回关键词的位置i,

如果找不到，返回-1

问题: 默认正则表达式是区分大小的

解决: 在第二个/后加后缀i ignore忽略

问题: 只能返回位置，不能返回关键词内容

3. 查找关键词的内容:

2种:

1. 只查找一个关键词的内容和位置

var arr=str.match(/正则/i)

返回值: 返回一个数组: [ 0: "关键词", index: i ]

找不到: null

强调: 只查找第一个关键词

问题: 只能找一个关键词

2. 查找所有关键词的内容:

var arr=str.match(/正则/ig) g global 全部

返回值: [ "关键词1","关键词2",...]

arr.length 代表关键词个数！

找不到，返回null

小心: 先判断不是null，再使用！

强调: 只包含关键词内容，不包含下标

问题: 只能获得内容，无法获得每个关键词的位置

4. 查找每个关键词的内容和位置:

RegExp.exec(str)

2. 替换:

2种:

1. 简单替换: 将所有关键词都替换为一个统一的新值

str=str.replace(/正则/ig,"替换值")

2. 高级替换: 根据不同的关键词，选择替换不同的新值

str=str.replace(/正则/ig,function(kw){

//function(kw)每找到一个关键词，就自动调用一次

//每次调用时，kw: 自动获得本次找到的关键词

return 新值

})

衍生: 删除: 其实就是替换为空字符串

3. 切割:

2种:

1. 简单切割: 分隔符是固定的

var arr=str.split("分隔符")

固定用法: 将字符串大散为字符数组:

var chars=str.split("")

2. 高级切割: 分隔符是变化的

var arr=str.split(/正则/i)

Day02

正课:

1. RegExp

2. Function

1. RegExp:

什么是: 封装一条正则表达式，并提供使用正则表达式执行验证和查找功能的API

何时: 只要在程序中存储正则表达式，或使用正则执行验证和查找时

如何:

创建: 2种:

1. 直接量: var reg=/正则/ig

问题: //中的正则是写死的，不能动态拼接生成

何时: 如果正则表达式可以写死，不需要动态生成时

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！

API:

1. 字符串格式验证:

验证一个字符串的格式是否符合正则表达式要求

如何: var bool=reg.test(str)

用reg检查str的格式是否正确

结果为布尔值

问题: 默认在str中只要能找到符合reg要求的子内容，就算通过！

解决: 今后，凡是验证，必须前加^，后加$

2. 查找关键词:

查找4: 即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)

在str中查找下一个符合reg要求的关键词的内容和位置

返回值: 只返回本次找到的一个关键词的内容和位置

arr:[ 0: "关键词", index: i ]

如果找不到，返回null！

聪明: 可用循环反复调用exec，不断找下一个！

exec()每次可自动从下一个开始找！

强调: reg必须加g，才能先后找！

2. Function

什么是: 封装一段可重用的代码段的对象

何时: 重用一段代码

如何:

创建: 3种:

1. 声明:

function 函数名(形参,...){

函数体;

return 返回值

}

何时定义参数: 函数本身缺少必须的值

何时定义返回值: 调用者需要获得执行结果时

本质: 函数其实也是一个引用类型的对象

函数名其实仅是一个普通的变量

函数名变量使用内存地址指向函数对象

问题: 声明提前——广受诟病的缺陷！

声明提前: 在程序开始执行前，引擎会将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

解决: 改为赋值方式创建函数:

2. 赋值方式:

var 函数名=function(...){ ... }

3. 用new: ——了解:

var 函数名=new Function("形参1","形参2",...,"函数体")

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，自动选择匹配的函数执行

为什么: 减少API的个数，减轻调用者的负担

何时: 一件事，可根据传入参数的不同，执行不同的逻辑时

如何:

问题: js中不支持重载写法:

因为: js中不允许多个同名函数同时存在

结果: 只有最后一个函数会覆盖之前所有同名函数

解决: 靠arguments对象

什么是: 函数内专门接收所有传入函数的参数值的类数组对象

类数组对象: 长的像数组的对象:

vs 数组: 相同: 1. 下标, 2. .length, 3. for遍历

不同: 类型不同——API不通用！

如何:

创建: 不用创建！每个函数中自带！

实现重载: 根据arguments中的参数个数和参数值的不同，动态执行不同的逻辑

匿名函数:

什么是: 定义函数时，不指定函数名

为什么: 2个原因:

1. 节约内存: 因为没有函数名变量引用着，用完立刻释放！

2. 避免函数重名

何时: 只要一个函数仅使用一次，不会被重用

如何: 2种:

1. 回调函数:

自己定义函数后不调用，而是交给别的函数去调用

arr.sort(function(a,b){return a-b})

//sort()执行完，内部匿名函数自动释放！

str.replace(/正则/,function(kw){ return 新值; })

2. 匿名函数自调:

定义匿名函数后，立刻调用自己执行。调用后立刻释放。

(function(...){ ... })() //调用后，匿名函数立刻释放！

作用域和作用域链:

作用域(scope):

什么是: 一个变量的可用范围——使用角度

其实是一个保存变量的对象——存储角度

为什么: 避免作用域内外的变量间互相干扰

包括: 2个作用域

1. 全局作用域:

什么是: 保存全局变量/函数的对象

nodejs: global

浏览器: window 代替了 global

2. 函数作用域:

什么是: 保存函数内局部变量的对象

函数的生命周期:

1. 程序开始执行时:

调用浏览器主程序main()启动浏览器

创建执行环境栈, 将main()的调用保存其中

main()创建window对象——全局作用域对象:

保存了所有内置对象和方法  
 准备保存将来声明的所有全局变量

2. 定义函数时:

在window中添加函数名变量

在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scope属性指回自己诞生自的对象

3. 调用函数时:

在执行环境栈中添加本次函数调用的记录

为本次函数调用创建临时作用域对象——函数作用域对象

在函数对象中添加函数中声明的所有局部变量

调用时变量使用顺序:

只要自己有，就用自己的

局部没有，才去全局

4. 函数调用后:

执行环境栈中本次函数调用的记录出栈

->导致函数作用域被释放

->导致局部变量一同释放！——局部变量不可重用

作用域链(scope chain):

由多级作用域逐级引用，形成的链式结构

两大作用: 1. 保存所有全局和局部变量

2. 控制着变量的使用顺序: 先局部，后全局

\*\*\*\*\*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: 全局变量和局部变量都有不可兼得的优缺点:

全局变量: 优: 可重复使用

缺: 随处可用——极容易被污染

今后尽量少用或不主动使用全局变量

局部变量: 优: 仅函数内可用，不会被污染

缺: 不可重用

何时: 即重用变量，又保护变量不被污染

如何:

1. 用外层函数包裹要保护的变量和使用变量的内层函数

2. 外层函数将内层函数返回到外部

3. 使用者调用外层函数，获得内层函数对象，保存在变量中反复使用

闭包形成的原因: 外层函数的作用域对象无法释放

主动使用闭包:

为一个函数，绑定一个专属的变量

课后任务:

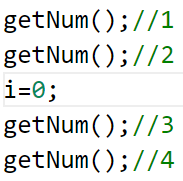
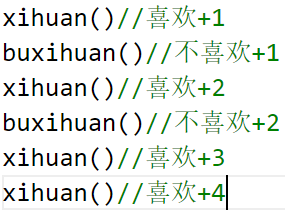
(1)复习: 闭包的使用方法

(2)作业: 完成课后练习:  
 题目要求:

1. 使用闭包实现"喜欢+1"和"不喜欢+1"效果

2. 使用闭包实现取号机程序

运行效果:

提示: 无

(3)项目:

题目要求:

运行效果:

Day03

正课:

1. OOP:

1. OOP:

什么是对象: 使用: 描述现实中一个具体事物的程序结构

存储: 集中存储一个事物的属性和功能的存储空间

什么是面向对象: 程序都是先用对象描述出现实中的事物，再使用对象去操作现实中的事物。

为什么: 便于大量数据和功能的管理

何时: 今后所有程序都是用面向对象实现的

如何: 三大特点: 封装，继承，多态

1. 封装:

什么是: 创建一个对象，集中保存一个事物的属性和功能

为什么: 便于大量数据和功能的管理

何时: 只要使用面向对象思想编程，都要先创建对象

如何: 3种:

1. 直接量:

var obj={

属性名:值,

... : ...,

方法名:function(){

... ...

}

}

访问对象成员: 成员=属性+方法

访问属性: obj.属性名 ——属性其实就是保存在对象中的变量，用法和变量完全一样

访问方法: obj.方法名() ——方法其实就是保存在对对象中的功能函数，用法和函数完全一样

问题: 对象自己的方法内要使用对象自己的属性

错误: 直接使用属性名

为什么: 不加任何前缀的变量，默认只能在作用域链中查找，不能擅自进入对象中获得对象的成员。

正确: this.属性名

什么是this: 自动指向正在调用该函数的.前的对象

.前是谁，this就自动指谁

本质: 保存在函数作用域中的一个关键词

何时: 今后，只要对象自己的方法，要访问自己的成员，都必须加this.

2. 用new: 2步:

1. 创建空对象: var obj=new Object()

new可省略, ()也可省略，但必须保留其一

2. 为对象添加新成员: obj.成员=值

揭示本质: js中一切对象底层，都是关联数组！

vs 关联数组:

1. 即可用.也可用[""]访问成员

何时用. : 如果属性名是写死的

何时用[""] : 如果属性名是动态获得的

2. 访问不存在的成员不报错，返回undefiend

3. 给不存在的成员赋值不报错，强行添加该成员

4. 都能用for in遍历每个属性

问题: 一次只能创建一个对象

反复创建多个相同结构的对象时:

代码繁琐，不便于维护

解决: 构造函数！

3. 用构造函数:

何时: 反复创建多个同一类型相同结构的对象时

优点: 代码/结构重用

如何: 2步:

1. 用构造函数定义同一类型所有对象的统一结构

function 类型名(形参,...){

this.属性名=形参;

... ...

this.方法名=function(){

... this.属性 ...

}

}

2. 调用构造函数，反复创建多个相同结构的对象:

var obj=new 类型名();

new: 4件事:

1. 创建一个新的空对象:

2. ?

3. 调用构造函数:

new会让this自动指向新对象

执行构造函数过程中: 通过强行给新对象赋值新属性的方式，添加新属性

4. 返回新对象地址给变量

问题: 构造函数中的方法定义，每new一次，就会反复创建副本——浪费内存！也不便于维护

解决: 1. 构造函数中不应再包含方法定义！

2. 方法定义放哪儿？

2. 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: 只要发现多个子对象拥有相同的成员时，都要用继承实现——只定义一次

如何: 共有的成员，只要定义在父对象上一份即可！

所有子对象，自动继承使用

如何使用父对象: ——原型对象

什么是: 集中存储所有子对象共有成员的父对象

创建: 不用创建，买一赠一

每创建一个构造函数，就附赠一个空的原型对象

确定继承关系: new的第2步: 自动让新对象继承构造函数的原型对象

利用继承关系: 只要放在原型对象中的成员，所有子对象自动就可继承使用

总结: 定义一种类型: 2部分:

1. 构造函数:

定义属性结构

用于创建子对象

2. 原型对象:

定义所有子对象共有的属性和方法

保存共有成员

内置类型的原型对象:

什么是内置类型: ES标准规定的，浏览器/平台已经实现的对象

包括: 11个:

Number String Boolean

Array Date RegExp Math

Error

Function Object

global (在浏览器中被window代替)

如何为内置类型扩展共有方法:

其实: 每种内置类型，同样也包含2部分:

1. 构造函数: 用于创建该类型的子对象

2. 原型对象: 保存该类型所有子对象共有的成员！

将共有方法添加到内置类型的原型对象中

this->将来调用该方法的.前的子对象

作业: 为String添加 ltrim() 和 rtrim()

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

保存着所有对象的属性

控制着对象属性的使用顺序和共享范围

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

包括2种:

1. 重载(overload)

2. 重写(override)

什么是重写: 子对象觉得父对象的成员不好用，在本地定义同名成员来覆盖父对象成员

何时: 子对象觉得父对象的成员不好用

如何: 只要在子对象本地定义同名成员

作业: 如何判断一个对象是不是数组类型，有几种方法

typeof?

课后任务:

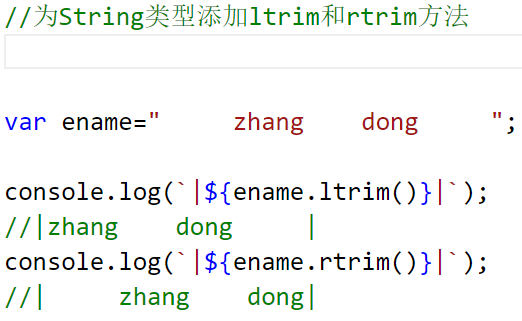
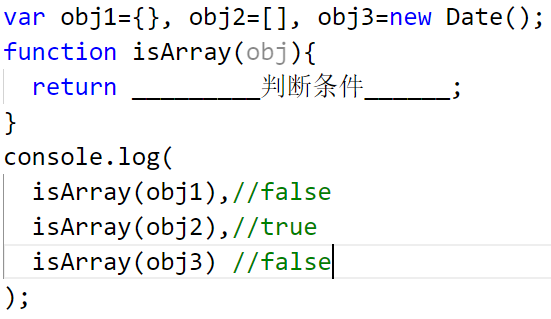
(1)复习: 面向对象三大特点

(2)作业: 完成课后练习:  
 题目要求:

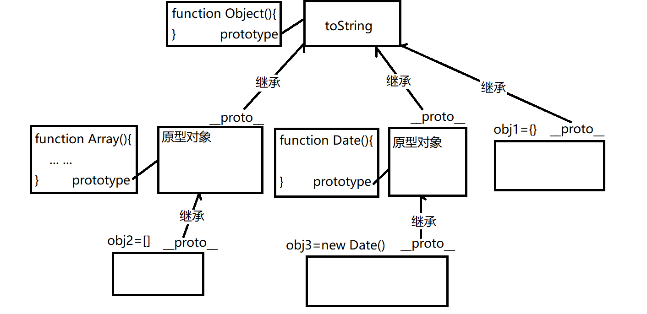
1. 为String类型添加共有方法ltrim和rtrim

2. 判断一个对象是不是数组，共有几种方法

运行效果:

提示:



(3)项目:

题目要求:

运行效果:

Day04

正课:

1. OOP

2. ES5

1. OOP

静态成员:

什么是: 不需要创建子对象，就可用构造函数直接调用的方法。

何时: 有些函数调用时暂时不确定传入的对象的类型

如何: 2步:

1. 定义时: 将函数直接定义在构造函数对象上

2. 调用时: 无需创建子对象，直接用构造函数.静态方法()调用

总结:

1. 必须某个类型的子对象，才能使用的方法，放在原型对象中——实例方法

2. 不需要具体子对象，就可直接调用的方法，放在构造寒素上——静态方法

包装类型:

什么是: 专门保存一个原始类型的值，并提供操作原始类型值的API

为什么: 原始类型的值，只是一个值而已，本身不带任何功能

何时: 只要试图对原始类型的值，用.访问属性或调用方法时，自动使用包装类型

如何:

var str="Hello";

str.toUpperCase();

typeof str string

new String(str).toUpperCase()

new String()释放

自定义继承:

1. 仅修改一个对象的父对象:

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

Object.setPrototypeOf(child,father)

//设置child的prototype为father

问题: 只能修改一个子对象的父对象

2. 同时修改多个子对象的父对象:

构造函数.prototype=father

强调: 时机: 定义完构造函数后，创建子对象之前换

问题: 随意！

3. 两种类型间的继承:

问题: 两种类型间包含部分相同的属性结构和方法定义

解决: 定义抽象父类型

什么是抽象父类型: 集中存储多个子类型共有属性结构和成员方法的父类型

如何: 2步:

1. 定义父类型: 2步:

1. 定义父类型构造函数，包含相同部分的属性结构

2. 定义父类型原型对象，包含相同部分的方法定义

2. 让子类型继承抽象父类型: 2步:

1. 让子类型的原型对象继承父类型的原型对象

2. 让子类型的构造函数借用父类型构造函数

错误: 直接调用父类型构造函数:

为什么: 直接调用: this->window, 属性都泄露到全局

正确: .call() 为函数打针！

何时: 函数中的this不是想要的！

如何: 函数.call(替换this的对象, 实参1, ... ... )

结果: 用指定的对象，替换函数中不想要的this

2. ES5

ECMAScript : JavaScript的核心语法标准

简称ES

版本: 3.1 5 6 ... ...

改进:

1. 严格模式:

什么是: 比普通js运行机制要求更严格的运行模式

为什么: js有广受诟病的缺陷

比如: 1. 可以给未声明的变量赋值，且自动在全局创建

2. 静默失败

3. 普通函数调用的this，默认指向window

4. 递归调用:

问题: 效率极低，重复计算量太大

何时: 今后所有项目都应该在严格模式下运行

如何: 2个范围:

1. 整个js文件/<script>标签内启用严格模式:

在顶部添加: "use strict";

2. 仅在函数内启用严格模式:

在函数内顶部加: "use strict";

要求: 1. 禁止给未声明的变量赋值

2. 将静默失败，升级为错误

3. 普通函数调用中的this不再指向window，而是值为undefined。

4. 禁止使用arguments.callee，其实就是禁止使用递归

解决: 绝大部分递归都可用循环代替

Day05

正课:

1. ES5

2. ES6

1. ES5

保护对象:

什么是: 控制对对象结构和属性值的修改

为什么: js的对象毫无自保能力

何时: 如果限制对一个对象的操作时

如何: 2个层面:

1. 保护单个属性:

ES5规定: 属性分2大类：

1. 内部属性: 保存在对象内部，但不能用.直接访问的属性。比如: class ——不需要保护，因为根本改不了

2. 命名属性: 凡是用.可直接访问到的属性

如何保护命名属性: 又分为两大类:

1. 数据属性: 实际存储属性值的属性

2. 访问器属性: 不实际存储属性值，仅提供对另一个数据属性得保护

数据属性: 如何保护:

每个数据属性中，都包含有四大特性:

sname:"Li Lei"

↓ES5

sname{

value: "Li Lei", //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-隐藏

强调: 仅控制for in遍历，不能控制.直接访问

configurable:true/false //控制是否可删除该属性

//控制是否可修改前两个特性

//一旦改为false，不可逆！

}

获得一个属性的四大特性(描述信息):

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

修改一个属性的四大特性:

Object.defineProperty(obj,"属性名",{

特性: true/false,

... : ...

})

问题: 一次只能修改一个属性的四大特性

解决: 批量修改多个属性的四大特性:

Object.defineProperties(obj,{

属性名:{

特性名:true/false

... : ...

},

属性名:{

特性名:true/false

... : ...

},

})

问题: 无法使用自定义规则保护属性

访问器属性:

何时: 只要用自定义规则保护属性时

如何:

前提: 先有一个隐藏的数据属性实际存储属性值

定义访问器属性: 不能用直接量，只能用API

Object.defineProperty(obj,"访问器属性名",{

//四大特性:

get:function(){ //每次试图取值时自动调用

return this.隐藏数据属性

},

set:function(val){ //每次试图赋值时自动调用

//val 自动获得要赋的新值

if(val符合要求)

this.隐藏数据属性=val

else

报错！

},

enumerable:true,

configurable:false

})

使用访问器属性: 用法同普通数据属性完全一样！

obj.访问器属性=新值

2. 保护对象结构: 3个级别:

1. 防扩展: 禁止给对象添加新属性

Object.preventExtensions(obj)

原理: 每个对象内部都隐藏着一个内部属性

extensible:true

preventExtensions(obj)将obj的extensible:false

2. 密封: 在兼具防扩展的同时，又禁止删除现有属性

Object.seal(obj)

原理: 2件事: 将obj的extensible:false

将obj的每个属性的configurable:false

3. 冻结: 在兼具密封的同时，又禁止修改所有属性值

何时: 一些公用的配置信息对象

Object.freeze(obj)

原理: 3件事: 将obj的extensible:false

将obj的每个属性的configurable:false

将obj的每个属性的writable:false

Object.create()

什么是: 基于一个现有父对象，直接创建子对象，并为子对象扩展自有属性

3件事: 1. 创建一个新对象

2. 让新对象继承父对象

3. 为新对象扩展自有属性

何时: 如果没有构造函数，也想创建子对象时

如何: var child=Object.create(father)

替换this: .call() .apply() .bind()

何时: 如果一个函数执行时，其中的this不是想要的

如何: 3种方法:

1. 在调用函数时，临时替换this: .call()

函数.call(替换this的对象, 实参1, ... ... )

问题: 临时绑定，只调用一次

2. 基于原函数，为某个对象，创建一个专属的新函数，并永久绑定this为指定对象: .bind()

var 新函数=函数.bind(替换this的对象)

.bind()2件事:

1. 创建一个和原函数一模一样的新函数

2. 永久替换新函数中的this为指定对象

固定场景: 专门用于绑定回调函数中的this

3. 打散数组参数: .apply

何时: 如果函数需要传入多个参数值，但参数是放在数组中给定的。

如何: 函数.apply(替换this的对象,数组)

apply会打散数组为单个值↓

//(替换this的对象,元素1,元素2,...)

数组API:

1. 查找元素在数组中的位置:

var i=arr.indexOf(elem[, fromi])

返回值: elem的位置i, 如果找不到，返回-1

2. 判断: 判断数组的元素是否符合要求

.every(): 判断数组中每个元素是否都符合要求

.some(): 判断数组中是否包含符合要求的元素

都返回bool值

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

//function会在每个元素上自动执行一次

//在每个元素上执行时:

//val: 自动获得当前元素值

//i: 自动获得当前位置

//arr: 自动获得当前正在遍历的数组对象

//不能用this！只能用arr->当前数组

return 判断条件

})

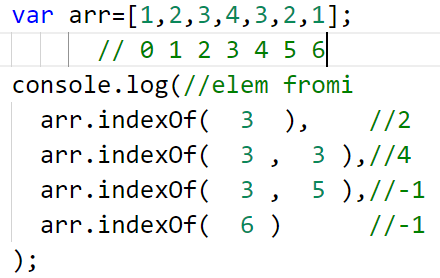
课后任务:

(1)复习: 如何保护对象属性和结构

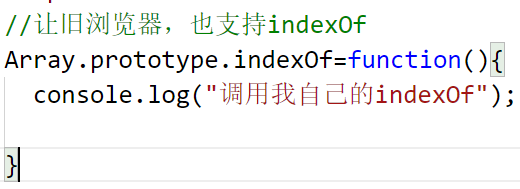
(2)作业: 完成课后练习:  
 题目要求:

1. 让旧浏览器也支持新的数组API

运行效果:



提示:



(3)项目:

题目要求:

运行效果:

Day06

正课:

1. ES5

2. ES6

1. ES5

数组API:

遍历API:

.forEach(): 对原数组中每个元素执行相同的操作

何时: 直接修改原数组时

如何:

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，放入新数组返回。

何时: 保护原数组不被修改，而返回新数组时

如何:

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

return 新值

})

过滤和汇总:

过滤: 选择出数组中符合条件的元素组成新数组

强调: 原数组不变

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

return 判断条件

})

汇总: 将数组中所有元素统计得出一个最终结果

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

//prev: 截止当前元素之前的临时汇总值

return prev+val;

}, base ) //base为初始值

2. ES6:

不改变原理的基础上，极简化了代码

学ES6，除了学简化外，必须会替代方法

包括:

let: 代替var声明变量

var的问题: 2个:

1. 声明提前, 2. 没有块级作用域

let的改进: 1. 阻止声明提前，2. 添加了块级作用域

原理: 匿名函数自调！

const: 常量, 一旦创建，值不能改变的变量

ES6中的const也具有let的特征

参数增强: 3项:

1. 默认值:

function fun(形参1,...,形参n=默认值)

调用时，如果没有给形参n传参，则用默认值代替

兼容: if(形参n===undefined)

形参n=默认值

强调: 带默认值的参数，必须放在参数列表的末尾

2. 剩余参数: rest

代替arguments！

arguments问题:

1. 不是数组类型，数组家api不能用！

2. 只能获取全部参数值，不能选择部分

如何使用rest: 2步

1. 定义函数时: function fun(形参1,..., ...数组名)

2. 函数内: 数组名 只接到除前几个参数外，剩余的参数值

兼容: 用回arguments

将类数组对象转为数组:

原理: 用slice复制整个类数组对象中所有元素

[].slice.call(arguments[,starti])

3. 散播:spread

代替apply，打散数组参数

apply的问题: 其实apply不是专职打散数组参数

是专职替换函数中的this，顺带能打散数组

如何: 调用时: fun(实参1,..., ...数组)

Math.max (...arr)

兼容: fun.apply(obj,数组)

箭头函数:

代替回调函数

如何: 口诀三句话:

1. 去掉function，换=>

2. 如果只有一个参数，可省略()

3. 如果函数体只有一句话，可省略{}

如果仅有的一句话还是return，则必须省略return

特例: 内外this通用！

事件处理函数不能改为箭头函数！

兼容: 就是用回function

模板字符串:

代替字符串拼接

ESLint中规定不允许使用+拼接字符串，必须用模板字符串

如何:

1. ``包裹字符串模板

2. 内部支持换行

3. 支持动态生成内容: ${js表达式}

解构:

简化变量的批量赋值

3种情况:

1. 数组解构: 将数组中的元素，提取出来，分别保存到多个变量中

如何: 下标对下标

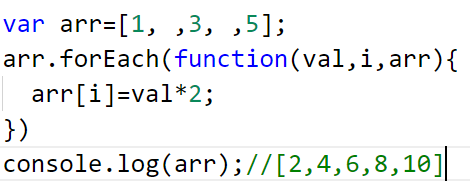
课后任务:

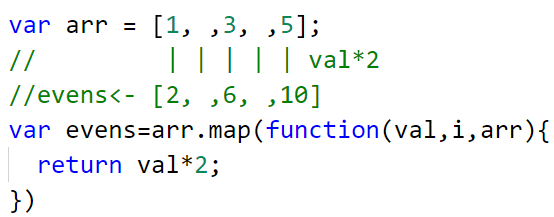
(1)复习: 收集整理js笔试题

(2)作业: 完成课后练习:  
 题目要求:

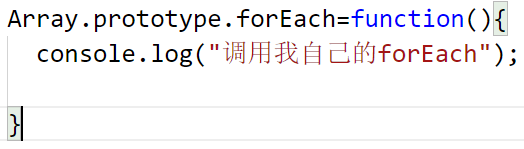
1. 让旧浏览器也支持forEach()和map()

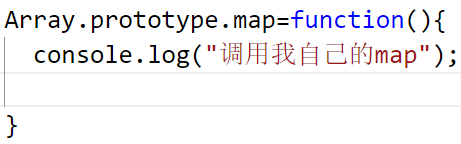
运行效果:





提示:





(3)项目:

题目要求:

运行效果:

Day07

正课:

1. ES6

1. ES6:

解构:

1. 数组解构: 下标对下标

2. 对象解构: 属性对属性

var { 属性1: 变量1, 属性2: 变量2, ...}

={ 属性1: 值1, 属性2: 值2, ...}

结果: 变量1=值1, 变量2=值2

3. 参数解构: 其实是特殊的对象解构:

何时: 定义函数时，函数的参数个数和顺序都不确定时

如何:

1. 定义函数时，将所有参数定义在一个对象结构中

function fun({参数1, 参数2, ... ...})

2. 调用时:

fun({ 变量1, 变量2,...})

其中, 变量名要和参数名保持一致，

但是，顺序和个数不必一致！

优点: 调用函数时，参数的个数和顺序与函数定义时无关！

for...of...

简化for循环:

总结: 遍历数组: 3种方式

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

var elem=arr[i]

...

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

for(var elem of arr){ ... elem ...}

for of vs for in

for of专门遍历索引数组或类数组对象——下标都是数字

for in 专门遍历关联数组或对象——下标是无规则的自定义名称

class:

1. 封装: class

什么是: 包裹构造函数和原型对象方法的一个程序结构

为什么: 传统构造函数和原型对象方法是分别独立定义的，不符合封装的要求

何时: 今后，只要创建一种类型都要包在class中

如何:

1. 用class{}包裹构造函数和原型对象方法

2. 构造函数名提升为class名，构造函数统一更名为constructor

3. 放入class中的原型对象方法，不再需要加prototype前缀，默认就已经保存在原型对象

简写: 放在class中的原型对象方法，不需要加function

2. 继承: extends

什么是: 让一个子类型继承父类型

如何:

1. class 子类型 extends 父类型

代替了setPrototypeOf()——不再需要！

2. 用super代替父类型.call(this,...)

super是专门指代extends后的父类型的关键词

super(参数值1, 参数值2,...)——不必再考虑this

3. 静态方法: static

定义时

class 类型{

...

static 方法名(){ ... }

}

调用时: 类型.方法名()

4. 访问器属性:

class 类型{

constructor( ... ){

... this.\_xxx 定义隐藏受保护的数据属性...

}

get 访问器属性名(){ return this.\_xxx }

set 访问器属性名(val){

只有验证val通过，才this.\_xxx=val

否则报错

}

}

Promise:

专门解决回调地狱: callback hell

什么是回调地狱: 因为回调函数嵌套层级太深导致的代码可读性差

何时: 只要要求两个异步调用的函数，必须先后执行时

如何:

1. 定义函数时:

function ajax(参数){ //去掉回调函数参数

return new Promise(function(open){

...办完自己的事儿后

open(要传给下一步的参数)

})

}

2. 调用时:

ajax(...) //return Promise

.then(function(){

return ajax(...) //return Promise

})

... ...

.then(function(){

...最后一步...

})

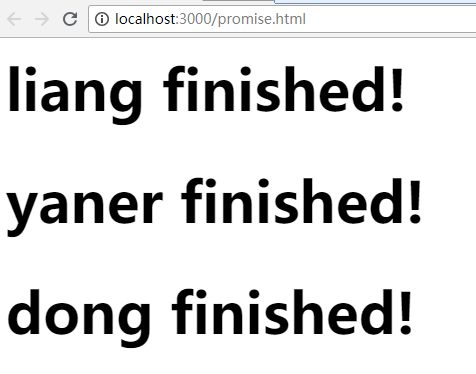
课后任务:

(1)复习: 复习ES6新特性

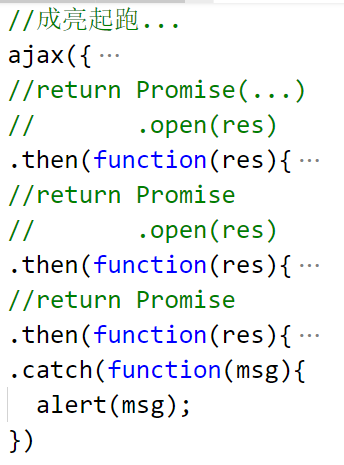
(2)作业: 完成课后练习:  
 题目要求:

使用Promise技术控制异步方法顺序执行，并支持错误处理

运行效果:



提示:



(3)项目:

题目要求: … 运行效果:…

**DOM**

Day01

正课:

1. 什么是DOM

2. DOM Tree

3. 查找

4. 修改

1. 什么是DOM:

js: ES(核心语法)+DOM(操作网页)+BOM(操作浏览器)

DOM: Document Object Model

专门操作网页内容的API标准——W3C

为什么: 统一操作网页内容的API标准

只要用DOM API操作网页100%兼容所有浏览器

能做什么: 查找，修改，添加，删除，事件

2. DOM Tree:

什么是: 内存中保存所有网页内容的树形结构

为什么: 树形结构最适合保存上下级包含关系

何时: 只要浏览器读到一个.html网页时，会自动创建DOM树

如何:

1. 先创建唯一的根节点对象document，代表整个网页

2. 依次创建每个后代节点对象Node

网页中每项内容(元素, 属性, 文本)，都是一个节点对象

所有节点对象都是document的后代节点

节点对象:

每个节点对象都有三大属性:

node.nodeType: 判断节点的类型

值: 数字

包括: 9-document 1-element 2-attribute 3-text

node.nodeName: 判断节点的名称\*\*\*

值: 字符串

包括:

document #document

element 全大写的标签名

attribute 属性名

text #text

node.nodeValue: 判断节点的值

document null

element null

attribute 属性值

text 文本内容

3. 查找:

什么是:找到要修改的元素对象

何时: 几乎所有的功能第一步都是查找！

如何: 4种:

1. 不需要查找就可直接获得的元素:

document.documentElement <html>

document.head <head>

document.body <body>

document.forms[i] <form>

2. 按节点间关系查找:

何时: 前提: 已经获得一个节点，要找周围附近的节点时

节点树: 包含所有节点对象的完整树型结构

2大类:

1. 父子关系:

node.parentNode 获得node的父节点

node.childNodes 获得node的所有直接子节点

返回所有直接子节点的集合

node.firstChild 获得node的第一个直接子节点

node.lastChild 获得node的最后一个直接子节点

2. 兄弟关系:

node.previousSibling 获得node的前一个兄弟

node.nextSibling 获得node的后一个兄弟

问题: 包含看不见的回车和空字符，干扰查找！

解决:

元素树: 只包含元素节点的树结构

优: 不包含回车和看不见的空字符，查找不受干扰

2大类:

1. 父子关系:

elem.parentElement 获得elem的父元素

elem.children 获得elem的所有直接子元素

返回所有直接子元素的集合

elem.firstElementChild 获得elem的第一个直接子元素

elem.lastElementChild获得elem的最后一个直接子元素

2. 兄弟关系:

elem.previousElementSibling 获得elem的前一个兄弟元素

elem.nextElementSibling 获得elem的后一个兄弟元素

问题: 兼容性 IE9+

遍历一个父元素下所有后代节点: 2步

1. 定义函数仅遍历一级直接子元素

2. 对每个子元素，调用和父元素完全相同的函数——递归

Vue框架中，就是利用遍历后代节点的方法，找到需要实时更新内容个元素，并不断监视。

3. 按HTML特征查找: 4个API:

1. 按id查找:

var elem=document.getElementById("id")

强调: 1. 只能用document调用

2. 总是返回一个元素对象，找不到返回null

2. 按标签名查找:

var elems=parent.getElementsByTagName("标签名")

强调: 1. 可在任意父元素上调用

2. 返回一个类数组对象，找不到返回空类数组对象

3. 不仅查找直接子元素，而且查找所有后代

3. 按name属性查找:

var elems=document.getElementsByName("name")

强调: 1. 只能用document调用

2. 返回一个类数组对象

4. 按class属性查找:

var elems=parent.getElementsByClassName("class")

强调: 1. 可在任意父元素上调用

2. 返回一个类数组对象

3. 不仅查找直接子元素，而且查找所有后代

4. 不必使用所有class名找，只要用其中一个查找就能获得当前元素

问题: 按HTML查找，一次只能用一个条件查找

如果查找条件复杂，代码很繁琐！

解决:

4. 按选择器查找:

何时: 当查找条件复杂时，首选按选择器查找

如何: 2个API:

1. 只查找一个元素:

var elem=parent.querySelector("selector")

返回值: 一个元素对象，找不到返回null

2. 查找多个元素:

var elems=parent.querySelectorAll("selector")

返回值: 包含多个元素对象的类数组对象，如果找不到返回空类数组对象

强调: 1. 都能在任意父元素上调用

2. 兼容性受制于浏览器

鄙视: 按HTML查找 vs 按选择器查找

1. 返回值:

按HTML查找，返回动态集合

动态集合: 不实际存储所有数据，每次访问集合，都要重新查找DOM树

按选择器查找，返回非动态集合

非动态集合: 实际存储完整数据，及时反复访问集合，也不会导致重新查找DOM树

2. 效率:

首次查找: 按HTML查找，快

按选择器查找，慢

https://jsperf.com/queryselectorall-vs-getelementsbytagname

3. 易用性:

按选择器查找，好用, 按HTML查找，繁琐

总结:

只靠一个条件就可找到想要的元素时，首选按html查找

查找条件复杂时，首选按选择器查找

4. 修改:

3种:

1. 内容: 3种情况:

1. 获取或修改原始HTML代码片段:

elem.innerHTML

2. 获取或修改纯文本内容:

elem.textContent

比innerHTML多做两件事:

1. 将转义字符翻译为正文

2. 去掉所有内嵌的标签

兼容性问题: IE9+

3. 获取或修改表单元素的值:

elem.value

2. 属性:

3. 样式:

修改内联样式:js: elem.style.css属性="值"

html: <elem style="css属性:值"

强调: css属性名必须去横线变驼峰

比如: fontSize, backgroundColor, listStyleType

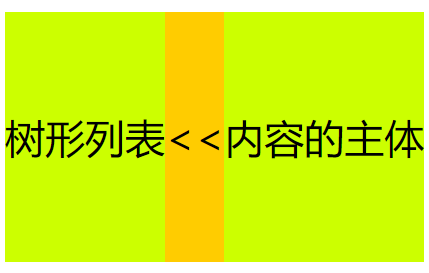
课后任务:

(1)复习: DOM查找API

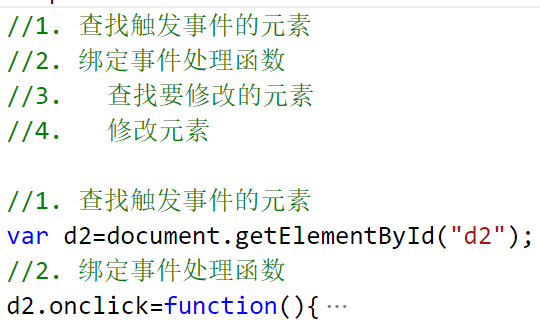
(2)作业: 完成课后练习:  
 题目要求:

使用DOM API实现开关门效果，要求支持过渡效果

运行效果:



提示:



(3)项目:

题目要求:

运行效果:

Day02

正课:

1. 修改

2. 添加/删除

3. HTML DOM常用对象

1. 修改:

3种:

内容: 3种:

1. HTML片段: .innerHTML

2. 纯文本: .textContent

3. 表单元素的值: .value

属性: 3种:

1. html标准属性: 值都是字符串

2种:

1. 核心DOM API: 最早指定的操作所有结构化文档的API

4个API:

1. 获取属性值: elem.getAttribute("属性名")

2. 修改属性值: elem.setAttribute("属性名","值")

3. 判断是否包含: elem.hasAttribute("属性名")

4. 移除属性: elem.removeAttribute("属性名")

2. HTML DOM API: 对常用DOM API进行的简化

HTML DOM已经将HTML标准属性提前定义在元素对象的内存中。只不过默认值暂时为""。

如何: elem.属性名

特例: class属性=> className

因为ES标准中已经有了内部属性class，用于标识一个对象的类型名。

1. 获取属性值: elem.属性名

2. 修改属性值: elem.属性名="值"

3. 判断是否包含: elem.属性名!==""

4. 移除属性: elem.属性名=""

2. 状态属性: enabled disabled checked selected

值都是bool类型

不能用核心DOM四个API访问

只能用HTML DOM打.访问

3. 自定义扩展属性:

什么是: 由程序员自己定义在开始标签中的属性

何时: 2种情况:

1. 在HTML中临时缓存业务数据

2. 代替其它选择器，用于选择触发事件的元素，绑定事件处理函数。——boot

如何: 2步:

1. html中添加自定义扩展属性

html5: <ANY data-自定义属性="值"

2. js中:

不能用HTML DOM打.访问

因为自定义扩展属性不可预知，就无法提前保存在DOM元素对象上。

只能用核心DOM:

获取: elem.getAttribute("自定义属性")

修改: elem.setAttribute("自定义属性","值")

HTML5: elem.dataset.自定义属性

dataset只能收集data-开头的自定义扩展属性

查询: [data-自定义属性名="xxx"]

样式:

修改内联样式: elem.style.css属性="值"

css属性名必须去横线变驼峰

style表示内联样式: 优先级最高！仅当前元素使用！

问题: elem.style.css属性一次只能修改一个css属性

如果一次要同时修改多个css属性，代码会繁琐

解决: 用class属性批量修改样式

如何: 2步:

1. css中: 提前定义几组需要的class

2. js中: elem.className="class" 批量应用一套样式

获取样式:

问题: .style仅代表内联样式，所以用.style也只能获得内联样式——丢样式

解决: 获得计算后的完整样式:

什么是: 最终应用到该元素上的所有样式属性的总和

何时: 只要获取样式，都必须获得计算后的样式

如何: 2步:

1. 获得计算后的完整样式表:

var style=getComputedStyle(elem)

2. style.css属性

问题: getComputedStyle获得的计算后的样式是只读的

为什么: css属性的来源和共享范围不确定，可能牵一发而动全身。

2. 添加/删除

添加: 3步:

1. 创建空元素:

var a=document.createElement("a")

<a></a>

2. 设置关键属性

a.href="http://tmooc.cn"

a.innerHTML="go to tmooc"

<a href="http://tmooc.cn">go to tmooc</a>

问题: 只有挂到DOM Tree上的元素，才能被浏览器发现，并绘制在界面上。

3. 将新元素添加到DOM树

3种:

parent.appendChild(a) 末尾追加

parent.insertBefore(a, child) 将a插入到child元素之前

parent.replaceChild(a, child) 用a代替child元素

\*\*\*优化: 尽量减少操作DOM树的次数

为什么: 每操作一次DOM树，都会导致浏览器重排重绘

原理:

html->DOM Tree

↓

Render Tree-> layout -> paint

↑ 重排 重绘

css->COM

如何: 2种:

1. 如果同时添加父元素和子元素: 现在内存中将子元素都添加到父元素上，最后将父元素一次性添加到DOM Tree

2. 如果父元素已经在页面上，要添加多个平级子元素时:

用文档片段:

什么是: 内存中临时保存多个平级子元素的虚拟父元素

何时: 要添加多个平级子元素时

如何: 3步:

1. 创建文档片段对象

var frag=document.createDocumentFragment();

2. 将多个子元素添加到文档片段中

frag.appendChild(child)

3. 将文档片段一次性添加到DOM树

parent.appendChild(frag)

结果: frag将子元素挂到DOM树上后，自动释放！

课后任务:

(1)复习: DOM 增删改查操作

(2)作业: 完成课后练习:  
 题目要求:

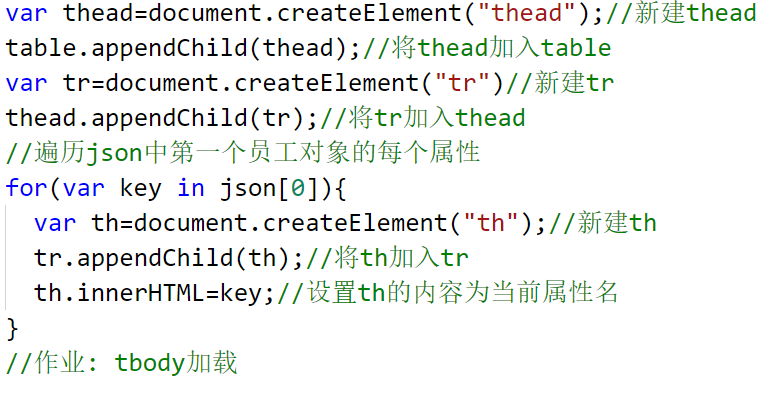
使用DOM API动态加载表格内容和select选项

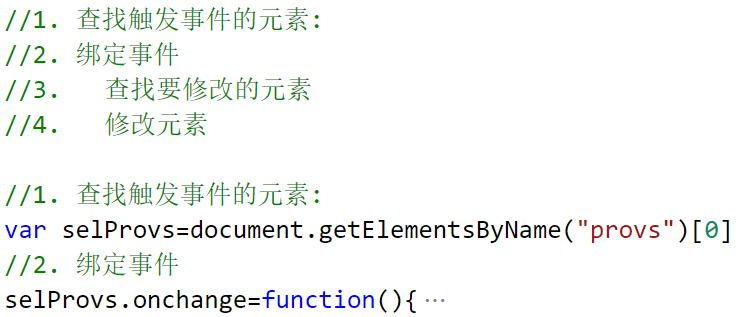
运行效果:





提示:





(3)项目:

题目要求:

运行效果:

Day03

正课:

1. HTML DOM 常用对象

2. 什么是BOM

3. window对象

4. 打开和关闭窗口

5. history

6. location

7. navigator

8. 定时器

1. HTML DOM常用对象

HTML DOM对一些常用的复杂HTML对象提供了简化版API

Image: 代表页面上一个<img>

唯一的简化: var img=new Image();

Select/Option

Select: 代表页面上一个select元素

属性: .selectedIndex 获得选中项的下标

.value 本来是获得选中项的value

用内容代替

.options 获得select下所有option的集合

.options.length 获得select下所有option的个数

.length => .options.length

固定用法: 清空所有option: sel.length=0

sel.innerHTML=""

方法: .add(opt) 将一个option对象添加到select下

问题: .add不支持文档片段！

.remove(i) 移除select下i位置的option

事件: .onchange=function(){} 当选中项改变时触发

Option: 代表select下一个option元素

创建: var opt=new Option(text,value)

Table/...:

管着行分组

添加行分组:

var tXX=table.createTHead()|.createTBody().creatTFoot()

删除行分组

table.deleteTHead()|deleteTFoot()

获取行分组

table.thead

table.tbodies[i]

table.tfoot

行分组管着行

添加: var tr=行分组.insertRow(i)

在行分组中i位置添加一个新行

固定套路: 1. 末尾追加: .insertRow()

2. 开头插入: .insertRow(0)

删除: 行分组.deleteRow(i)

强调: i要求是行在行分组内的相对下标位置

问题: tr.rowIndex 只能自动获得相对于整个表的下标位置，不能用于行分组删除行

解决: 今后删除行一律用table.deleteRow(tr.rowIndex)

获取: 行分组.rows[i]

行管着格:

添加: var td=tr.insertCell(i)

在tr中i位置插入一个新格

固定用法: tr.insertCell() 末尾追加一个新格

强调: 只能创建td,不能创建th

删除: tr.deleteCell(i)

获取: tr.cells[i]

Form/...

Form: 代表一个<form>

获取: var form=document.forms[i/id]

属性: .elements 获得表单中所有的表单元素

强调: 只能获得表单元素（input select textarea button）

.elements.length 获得表单中表单元素的个数

.length => .elements.length

方法: .submit() 代替提交按钮，实现手动提交表单

何时: 如果希望手动控制是否提交时，就要用:

普通按钮 + form.submit() -> 提交表单

事件: onsubmit()

表单元素:

获取: var 表单元素=form.elements[i/id/name]

简化: 如果表单元素有name属性，可简写为:

form.name

方法: .focus() 让当前元素获得焦点

.blur() 让当前元素失去焦点

2. 什么是BOM:

BOM: Browser Object Model

专门操作浏览器窗口的API

问题: 没有标准——兼容性问题

3. window对象:

window对象: 2个角色:

1. 代替global充当全局作用域对象

2. 保存所有ES，DOM和BOM的内置对象

属性:

窗口大小: 2组:

1. 完整大小: window.outerWidth, window.outerHeight

2. 文档显示区大小:

window.innerWidth, window.innerHeight

4. 打开和关闭窗口:

4种:

1. 在当前窗口打开新连接，可后退:

html: <a href="url" target="\_self"

js: window.open("url","\_self")

2. 在当前窗口打开新连接，禁止后退:

js: location.replace("新url")

用新url，代替history中现在的url，以此禁止后退

3. 在新窗口打开新连接，可打开多个:

html: <a href="url" target="\_blank"

js: window.open("url","\_blank")

4. 在新窗口打开新连接，只能打开一个:

html: <a href="url" target="自定义窗口名"

js: open("url","自定义窗口名")

原理: 每个窗口在内存中都有一个唯一的name属性

浏览器规定相同name属性的窗口只能打开一个

后打开的同名窗口会覆盖先打开的

其实<a ... target="新窗口的name"

open("url","新窗口的name")

两个保留的特殊窗口名:

\_self: 获取当前窗口自己的名字作为新窗口的名字

结果: 新窗口会覆盖当前窗口

\_blank: 不指定窗口名，浏览器会随机分配窗口名

窗口名永不冲突，所以可反复打开多个！

5. history:

什么是: 保存当前窗口打开后，成功访问过的url的历史记录栈

API: history.go(n)

前进一步: history.go(1)

刷新: history.go(0)

后退一步: history.go(-1)

后退2步: history.go(-2)

6. location: 代表地址栏

什么是: 保存当前窗口正在浏览器的url地址的对象

如何:

属性:

location.href 完整的url

.protocol 协议

.host 主机名+端口号

.hostname 主机名

.port 端口号

.pathname 相对路径

.hash #锚点地址

.search ?查询字符串

方法:

1. 在当前窗口打开，可后退:

location.assign("新url")

原理: location.assign("新url") => location.href="新url"

2. 在当前窗口打开，禁止后退:

location.replace("新url") 用新url代替history中现在url

3. 刷新: 2种:

1. 普通刷新: 优先从浏览器本地缓存中获得资源文件

本地缓存没有或过期，才去服务器重新下载

location.reload()

history.go(0)

F5

2. 强制刷新: 总是跳过本地缓存，强制从浏览器下载最新的资源文件

location.reload(true)

7. navigator:

什么是: 保存浏览器配置信息的对象

包括:

navigator.cookieEnabled 获得当前浏览器是否启用了cookie

什么是cookie: 在客户端本地硬盘持久存储用户私密信息的小文件

为什么: 程序中保存的数据都是在内存中临时存储的

程序关闭，内存释放，数据丢失。

何时: 只要希望在客户端永久保存用户私密信息时

比如: uid, keywords, config...

运行原理:

查看和设置cookie: chrome://settings/content/cookies

navigator.plugins: 保存浏览器安装的插件信息的集合

什么是插件: 为浏览器添加新功能的小软件

如何判断浏览器是否安装某个插件:

if(navigator.plugins["插件名"]!==undefined)

已安装

else

未安装

navigator.userAgent: 保存浏览器名称和版本号的字符串

何时: 只要判断浏览器的名称和版本号

如何:

1. 判断浏览器名称: 只能猜:

检查当前浏览器的userAgent字符串中是否包含某个浏览器的名称

2. 判断版本号: 截取从浏览器名称所在位置跳过浏览器名字的长度+1位置开始后的3位

8. 定时器:

1. 周期性

什么是: 让程序每隔一段时间，反复执行一项任务

何时: 让程序每隔一段时间，反复执行一项任务

如何: 3件事:

1. 定义任务函数

2. 启动定时器: var timer=setInterval(任务函数, 间隔ms)

3. 停止定时器: clearInterval(timer)

2. 一次性

什么是: 让程序先等待一段时间，再延迟执行一项任务

执行后自动停止定时器

何时: 让程序延迟执行一项任务时

如何: 3件事:

1. 定义任务函数

2. 启动定时器: var timer=setTimeout(任务函数, 延迟ms)

3. 停止定时器: clearTimeout (timer)

课后任务:

(1)复习: 定时器的原理和使用方法

(2)作业: 完成课后练习:  
 题目要求:

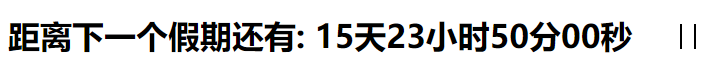
1. 定义函数将查询字符串转化为对象

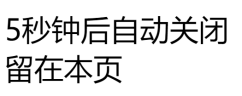
2. 使用定时器实现倒计时效果

3. 使用定时器实现5秒后自动关闭页面

运行效果:







Day04

正课:

1. event

1. event:

什么是事件: 浏览器自己触发的或用户手动触发的页面内容状态的改变

什么是事件处理函数: 事件发生时自动调用的函数

何时: 只要希望发生事件时，能自动执行一个响应操作

如何:

1. 绑定事件处理函数: 3种:

1. html中: <ANY on事件名="js语句"

问题: 不符合内容与行为分离的原则，不便于维护和重用

2. js中, 以赋值方式: elem.on事件名=function(){ ... }

定义一个函数，交给elem的事件属性保管

当触发事件时，元素会自动调用该事件属性保管的处理函数

问题: 一个元素的一个事件，只能绑定一个处理函数

3. js中，添加事件监听对象方式:

elem.addEventListener("事件名",处理函数)

原理:

创建一个事件监听对象，包含事件名和处理函数的对应关系。

将事件监听对象添加到浏览器的事件监听队列中

elem.removeEventListener("事件名",原处理函数)

强调: 移除事件监听，只能用函数名从才能获得原函数对象，所以，添加监听时，不能用匿名函数！必须用有名的函数

\*\*\*DOM事件模型:

从事件触发，到执行完处理函数的过程

3个阶段:

1. 捕获: 由外向内依次记录各级父元素行绑定的事件处理函数——只记录不触发

2. 目标触发: 优先触发目标元素上的处理函数

目标元素: 最初，实际触发事件的元素

3. 冒泡: 由内向外依次触发捕获阶段记录的各级父元素上的处理函数

事件对象:

什么是: 保存事件信息，并提供操作事件行为的API

何时: 获取事件信息，以及修改事件的行为

如何:

1. 创建: 不用手动创建，而是在事件发生时，自动创建

2. 获取事件对象:

默认事件对象总是作为处理函数的第1个参数传入

3. 事件对象的API:

1. 取消冒泡: e.stopPropagation()

2. \*\*\*利用冒泡:

优化: 尽量减少事件监听对象的个数

为什么: 浏览器是通过遍历方式查找要触发的事件监听对象。如果事件监听对象多，遍历慢，事件响应就慢

如何:

如果多个平级子元素，需要绑定相同的事件处理函数时，其实，只要在父元素上绑定一次，所有子元素就可共用

2个难题:

1. 获得目标元素:

错误: this, this->父元素

正确: e.target -> 始终保存着最初的目标元素，不随冒泡而改变。

2. 只限于想要的元素触发事件:

不利用冒泡 vs 利用冒泡

1. 绑定位置: 不利用冒泡绑在子元素上

利用冒泡绑在父元素上

2. 监听个数: 不利用冒泡——多！

利用冒泡——只有一个

3. 动态添加的元素:

不利用冒泡, 无法自动获得事件处理函数

利用冒泡, 可自动获得事件处理函数

3. 阻止默认行为: e.preventDefault();

何时: 如果一个事件的浏览器默认行为不是想要的，就可取消默认，写自己的行为

3种:

1. <a>作为按钮时:

默认: 会修改url地址栏,添加#锚点地址

解决: e.preventDefault()

2. 实现自定义表单提交:

2种:

<input type="button" + btn.onclick + form.submit()

<input type="submit" + form.onsubmit() 验证不通过，就阻止提交

3. HTML5拖拽API——下个老师讲

4. 获得鼠标坐标:

3组:

1. 相对于整个屏幕左上角: e.screenX, e.screenY

2. 相对于文档显示区左上角: e.clientX, e.clientY

3. 相对于当前元素左上角: e.offsetX, e.offsetY

**jQuery**

Day01

正课:

1. 什么是jQuery

2. 如何使用jQuery

3. 查询

1. 什么是jQuery

第三方的执行DOM操作的极简化的函数库

第三方: 下载

执行DOM操作: 学jQuery还是在学DOM

极简化: jQuery是DOM操作的终极简化

比如: document.querySelector("选择器") => $("选择器")

函数库: jQuery中是用函数解决一切问题

比如: div.innerHTML=html => div.html(html)

为什么: 2大原因:

1. 是DOM操作的终极简化: 4大方面:

1. DOM操作(增删改查)

2. 事件绑定

3. 动画

4. ajax

2. 解决了绝大多数兼容性问题

凡是jQuery允许使用的，都没有兼容性问题

比如: document.querySelector("ul>li:last-child") IE9+

=>$("ul>li:last-child") 所有浏览器都可用

何时: 今后所有项目，和框架的开发都用jQuery实现

2. 如何使用jQuery:

学习一项新技术:

1. 上官网看是什么

2. 上官网看快速入门案例

3. 浏览手册

下载: jquery.com

版本:

1.x 兼容旧浏览器

jquery.1.11.3.js 未压缩版本

保留了完整的注释

具有完备的代码格式

具有见名知意的变量名

特点: 优: 可读性好

缺: 体积大不便于下载

何时: 学习时或开发时

jquery.1.11.3.min.js 压缩版本

去掉所有的注释

去掉代码格式(空格，换行，缩进)

极简化变量名

特点: 优: 体积小，便于快速下载

缺: 可读性差

何时: 生产环境

2.x 不再兼容旧浏览器

3.x 也不再兼容旧浏览器

新特性: 1. jQuery所有代码都运行在严格模式下

2. 引入for...of循环，代替$.each

3. 新动画API: RequestAnimationFrame

setInterval(移动一小步, 10)——可能浪费

RequestAnimationFrame(移动一小步)

可自动根据显示器刷新频率，来控制移动

4. 支持Promise技术

如何引入jquery.js: 2种:

1. 引入项目本地的jquery.js文件:

js/jquery.1.11.3.js

2. 引入CDN网络中在线的jquery.js文件

CDN: 内容分发网络，可根据用户到服务器的网络状况优劣，自动选择最优的服务器下载。

jQuery原理:

引入jQuery: 其实是向全局添加一种新的类型jQuery

1. 构造函数: 创建该类型的子对象

jQuery = function (参数){ —— 工厂函数

return new XXX(参数)

}

window.$=window.jQuery=jQuery

var obj=$(参数)

//new XXX(参数)

何时: 只要想用jQuery的简化版API，都必须先创建jQuery对象。

为什么: 只有jQuery类型的子对象才能使用简化版API

如何: 2种:

1. 先查找DOM元素，再封装进jQuery对象中:

var $elem=$("选择器");

2. 直接将到手的DOM元素，封装进jQuery对象中:

var $elem=$(DOM元素)

2. 原型对象: 保存只有该类型子对象才能使用的API

学习jQuery还是在学DOM: 增删改查，事件绑定

3. 查找:

按选择器查找: jQuery支持所有css选择器，并少量添加了部分新选择器

基本选择器: 5个: 同CSS

#id, element, .class, \*,

selector1,selector2,...

层级选择器: 4个: 同CSS

父子: 父>子, 父 子

兄弟: 兄+弟, 兄~弟

过滤选择器:

子元素过滤: 4个: 同CSS

:first-child, :last-child, :nth-child(n), :only-child

基本过滤: jQuery新增: 7个

:first/last :eq/gt/lt(i) :even/odd

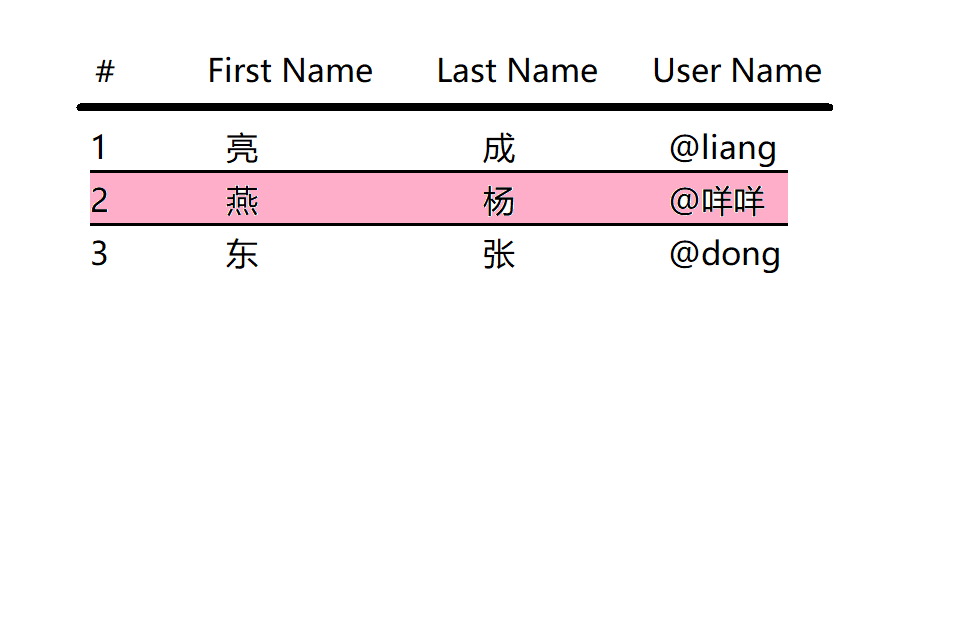
课后任务:

(1)复习: CSS选择器

(2)作业: 完成课后练习:  
 题目要求:

实现规定表格效果

运行效果:

提示: 使用css和jQuery两种方式实现

(3)项目: 仿学子商城商品详情页，完成个人项目中详情页内容加载

题目要求:

运行效果:

Day02

正课:

1. 查询:

2. 修改:

3. 按节点间关系查找

4. 添加/删除/替换/克隆

5. 事件绑定

1. 查询:

内容过滤:

1. :contains(文本) 用元素的内容文本作为条件

2. :has(selector) 选择包含符合条件的子元素的父元素

3. :parent和:empty: 选择非空的或空的元素

可见性过滤: :visible :hidden

:hidden只能选择: display:none的和input type=hidden的

属性过滤: 同CSS

表单过滤:

表单元素过滤:

:input 查询所有表单元素: input select button textarea

每种type都有专门的选择器:

:text :password :checkbox :radio :file :submit ...

表单状态过滤:

:enabled :disabled :checked :selected

2. 修改:

内容: 3种

1. html片段: .html()

2. 纯文本: .text()

3. 表单元素的值: .val()

属性: 3种

1. HTML标准属性:

$elem.attr("属性名"[,"新值"])

//getAttribute/setAttribute

可以获得开始标签中的一切字符串类型的属性

2. 状态属性:

$elem.prop("状态"[,bool])

可以获得内存中可用.访问的所有属性

3. 自定义扩展属性:

$elem.attr()

问题: .attr()和.prop()一次只能修改一个属性

解决: .attr/prop({

属性:值,

... : ...

})

样式:

修改css属性: $elem.css("css属性"[,新值])

.style.css属性

$elem.css({

css属性:新值,

... : ...

})

福利: 数值类型的css属性，不再需要加px单位

获取css属性: $elem.css("css属性")

getComputedStyle()

用class批量管理样式:

.addClass("class")

.removeClass("class")

.hasClass("class")

.toggleClass("class")

相当于: if($btn.hasClass("down"))

$btn.removeClass("down")

else

$btn.addClass("down")

3. 按节点间关系查找:

1. 父子关系:

.parent()

.children(["selector"]) 仅选择符合条件的直接子元素

.find("selector") 可在所有后代中查找符合条件的直接子元素

.children(":first-child") 仅选择直接子元素中第一个元素

.children(":last-child") 仅选择直接子元素中最后一个元素

2. 兄弟关系:

.prev() 前一个

.prevAll() 之前所有

.next() 后一个

.nextAll() 之后所有

.siblings() 除自己之外所有

4. 添加/删除/替换/克隆

添加: 2步:

1. 用$创建一个新元素: var $elem=$("html片段")

2. 将新元素追加到DOM树

$parent.append($elem) 末尾追加

$elem.appendTo($parent) return $elem

$parent.prepend($elem) 开头插入

$elem.prependTo($parent) return $elem

$child.before($elem) 将$elem插入到$child之前

$elem.insertBefore($child)

$child.after($elem) 将$elem插入到$child之后

$elem.insertAfter($child)

删除: $elem.remove()

替换:

$child.replaceWith($elem)

$elem.replaceAll("child")

克隆: var $newElem=$elem.clone()

默认: .clone()是浅克隆，仅复制样式和属性，不复制行为

改为: .clone(true) 深克隆，即复制样式和属性，又复制行为

5. 事件绑定:

1. 普通绑定:

.on("事件名",function(){ ... }) <==> addEventListener()

简写: .事件名(function(){ ... })

2. 利用冒泡:

$parent.on("事件名","selector",function(){

//this->e.target

//执行操作

})

3. 移除事件绑定:

.off("事件名",原处理函数) <==> removeEventListener()

Day03

正课:

1. 事件

2. 动画

3. 类数组对象

4. 添加自定义API

5. 封装自定义插件

1. 事件:

加载后执行: 2种

1. DOMContentLoaded: 仅DOM内容加载完就提前执行

只等html和js加载完就立刻执行，不等css和图片

何时: 不依赖于css和图片的代码都应该放在DOMContentLoaded中提前执行

比如: 事件绑定

如何: $(document).ready(function(){...})

简写: $().ready(function(){...})

更简写为: $(function(){...})

总结: 今后所有jq代码，都要包含在$(function(){ ... })

2. window.onload: 整个HTML页面加载完才执行

等所有网页内容(html,css,js,图片)加载完才执行

何时: 依赖于css和图片的代码，只能放在window.onload中稍后执行。

如何: $(window).load(function(){ ... })

鄙视: $的原理: 4种:

1. 查找并封装DOM元素: $("selector")

如果"selector"是#id，$自动调用getElementById

如果"selector"是elem，$自动调用getElementsByTagName

如果"selector"是.class，$自动调用getElementsByClassName

否则，其余都调querySelectorAll

2. 直接封装DOM元素: $(DOM元素)

3. 创建新元素: $("html片段")

4. 绑定DOMContentLoaded事件: $(function(){ ... })

鼠标事件:

mouseover mouseout

问题: 反复进出子元素，也会反复触发父元素上的事件

解决: mouseenter和mouseleave代替

简化: 如果同时绑定mouseenter和mouseleave

可简写为: .hover(

function(){ ... },//给enter的

function(){ ... } //给leave的

)

更简化: 如果两个函数刚巧可以合并为一个函数，也可以只写一个处理函数

模拟触发:

$elem.trigger("事件名")

可简写为: $elem.事件名()

2. 动画:

1. 简单动画: 效果固定的动画API, 3组:

1. 显示隐藏: .show() .hide() .toggle()

默认: 用display:none/block瞬间显示隐藏

加动画: 必须指定动画持续时间:

.show(ms) .hide(ms)

2. 上滑下滑: .slideUp() .slideDown() .slideToggle()

3. 淡入淡出: .fadeIn() .fadeOut() .fadeToggle()

问题:

1. 效果是用js写死的，无法维护。

2. 用js定时器实现的动画效果，效率低！

总结: 简单动画应首选css+transition实现——效率高，可维护

特例: .show() .hide() 推荐使用

2. 万能动画: 可对多数css属性指定动画效果的API

如何: $elem.animate({

css属性: 目标值,

... : ...

},动画持续时间ms)

问题: 只支持单个数值的css属性: width, height, margin, padding, borderWidth, fontSize, opacity, borderRadius

不支持: color, backgroundColor, fontFamily ...

排队和并发:

并发: 多个css属性同时变化

如何: 放在一个animate内的多个css属性，默认并发变化

排队: 多个css属性先后依次变化

如何: 先后对一个元素调用多次animate，则每个animate内的变化是排队依次执行

动画结束后执行:

每个动画API都有最后一个回调函数参数，会自动在动画结束后执行。

停止动画: .stop()

问题: .stop()只能停止队列中当前正在播放的动画，队列中后续动画依然正常执行。

解决: .stop(true) 清空队列

选择器: :animated 用来选择正在播放动画的元素

和判断元素是否正在播放动画

总结: 实现动画效果共几种手段:

css: class+transition animation+keyframes

优点: 效率高

缺点: 无法添加交互行为

js定时器: .animate()

优点: 随意添加交互行为

缺点: 效率低

RequestAnimationFrame——自学

3. 类数组对象操作:

1. 获得jQuery对象中DOM元素的个数:

$elem.length 或 $elem.size()——新版本已废弃

2. 将jQuery对象转换回DOM元素对象:

var li=$lis[i] 或 $lis.get(i)

本意: 取出jQuery类数组对象中i位置的DOM元素

3. 查询一个元素在结果集合中的位置:

var i=$lis.index(li)

简写: 如果是在一个父元素内，找子元素的位置:

var i=$child.index()

4. 遍历:

$(...).each(function(i,elem){

//i 获得当前位置

//elem 获得当前DOM元素

//this-> elem

})

其实，在3.x版本中被for of代替

for(var elem of $(...)){

//elem当前DOM元素

}

4. 添加自定义API:

1. 定义独立的js文件封装自定义的扩展API

jQuery.fn.自定义API=function(){

//this->将来调用自定义API的.前的$(...)对象

~~$(this)~~ 而是直接用this

}

2. 引入:

先引入jquery.js, 再引入自定义的js文件

3. 调用自定义API

$(...).自定义API()

5. 封装自定义插件:

jQuery官方插件: jquery-ui

手风琴: accordion

标签页: tabs

如何:

1. 引入插件的css

2. 按插件要求编写HTML内容

3. 引入js: 先引jquery.js, 再引jquery-ui.js

4. 编写自定义脚本:

找到插件的父元素，调用插件API

原理: 侵入式:

插件函数根据自身需要，自动添加全部的class和行为！

优点: 简单！

缺点: 行为和样式都是写死的，不便于维护！

vs bootstrap DIY:

需要开发人员手工设置样式的class和自定义扩展属性

缺点: 麻烦！

优点: 可维护！

封装自定义插件时，如何选择:

如果给自己人用，小范围使用，优先选择jQuery-ui侵入方式

如果公开使用，大范围使用，优先选择bootstrap DIY方式

封装插件:

前提: 必须已经用普通的html ,css,js,jq实现了效果

封装jQuery UI风格插件:

1. 提取并整理css到一个独立的css文件中

要求: 一个插件内的css选择器，必须以一致的父选择器开头

为什么: 避免插件的样式和其它插件样式冲突

2. 定义独立的js文件

为jQuery类型添加自定义插件API:

侵入class和自定义扩展属性:

以父元素为起点，遍历子元素，并添加class和自定义扩展属性

查找触发事件的元素，绑定事件:

其实是，将原来的事件绑定拷贝到js中，修改主语为$parent即可

如何使用: 同jquery ui插件的用法。

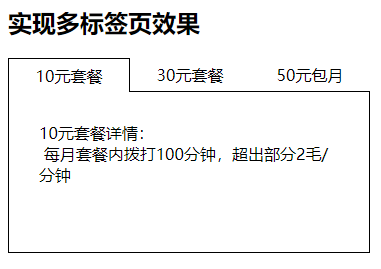
课后任务:

(1)复习: jQuery和Bootstrap插件定义和使用方式

(2)作业: 完成课后练习:  
 题目要求:

用jQuery-ui侵入式封装自定义标签页插件

运行效果:

提示:

(3)项目: 仿学子商城商品详情页，完成个人项目中详情页内容加载

题目要求:

运行效果:

Day04

正课:

1. 封装自定义插件

2. ajax

3. \*\*\*\*\*跨域

1. 封装自定义插件:

jQueryUI侵入式:

bootstrap DIY式:

封装插件:

1. css完全一样

2. js文件：

1. 不用添加自定义API

2. 不用侵入

3. 只需要查找自定义扩展属性的元素，绑定事件即可！

使用插件:

1. 引入插件css

2. 按插件要求编写HTML内容

并手动添加class和data-\*扩展属性

3. 引入jquery.js和插件.js

不需要再调用任何函数

因为: 在引入插件.js时，就自动绑定了事件！

2. ajax

$.ajax({

url: "url",

type:"get|post",

data:"变量=值&变量=值&..."|{变量:值,变量:值,...},

dataType:"json",

success:function(res){

//收到响应，且响应成功时

},

error:function(){

//收到响应，但出错时——用于调试

},

complete:function(){

//无论成功还是失败，只要响应结束就触发——用于调试

}

})

//3.x

.then(function(res){

...

})

**VUE**

Day01

正课:

1. 原生API vs 函数库 vs 框架

2. Vue概述

3. MVVM框架

4. 数据绑定语法

5. 指令(directive)

1. 原生API vs 函数库 vs 框架

原生API: 浏览器/平台已经实现的，咱们可直接使用的原生对象和函数

比如: ES, DOM, BOM

问题: 代码繁琐

函数库(library): 基于原生的API基础上，进一步封装的，更简化的一组函数的集合。

优: 每步操作的API比原生API简单

缺: 并没有改变做事的方法，如jQuery，还是DOM，依然是增删改查，事件绑定。

框架(framework): 前人将多次成功项目的经验总结起来，形成的半成品项目

优: 后人继续开发即可，不再需要大量重复的手工劳动。

缺: 需要改变原有的做事方法。

比如: vue中根本没有查找方法！数据一样能自动出现在想出现的位置。

2. Vue概述

官网: http://cn.vuejs.org

什么是Vue.js: 一个渐进式(Progressive)的，基于MVVM设计模式的JavaScript框架。

什么是渐进式: 虽然有很多组件，但不强迫必须全部使用。

vs AngularJS和React: 必须打包全套使用

什么是MVVM: 稍后讲

何时: 适用于以数据操作(增删改查)为主的项目(Web, APP)。

安装:

兼容性: 不支持IE8及以下版本

当前版本: 2.5

如何: 2种:

1. 直接下载vue.js文件，引入网页中——前三天

2个版本: 开发版 —— 有错误提示

压缩版 —— 没有错误提示

强调: vue是一个纯前端框架，所以不需要nodejs，apache等服务端软件，就可使用。

原理: 添加一种新类型——Vue:

包括: 2部分:

1. 构造函数——创建Vue类型的实例对象

2. 原型对象——保存所有Vue类型子对象共有的API

2. 使用Vue-CLI脚手架工具——后两天

3. MVVM设计模式:

传统网页组成: 3部分:

HTML: 定义网页的内容

CSS: 定义网页的样式

JS: 为网页添加交互行为

问题: JS DOM操作，既要加工数据，又要承担大量的反复的增删改查操作。

重新划分上述3部分：

View视图: 指网页中的元素和样式，包括HTML和css

Model模型: 指程序中创建的或从远程获取的数据

ViewModel视图模型/控制器:

代替之前的DOM/JQUERY操作，自动把模型中的数据和视图中的HTML元素"绑定"在一起.

什么是绑定: 不需要写一行代码，就可让页面元素内容和数据保持联动变化。

Vue类型，提供的就是一种强大的ViewModel对象，可自动同步数据和页面元素。

ViewModel的核心原理: 2部分:

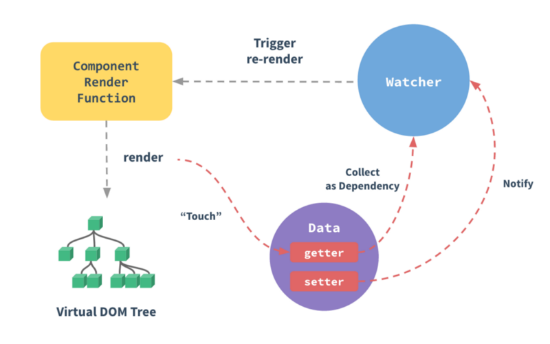
1. 响应系统(Reactivity System):

内存中数据变，页面自动变

原理:

每个内存中的数据，都被Vue自动添加了访问器属性。

每次修改内存中数据时，都会自动调用set访问器方法。set访问器方法会自动通知Vue框架修改页面对应的元素。

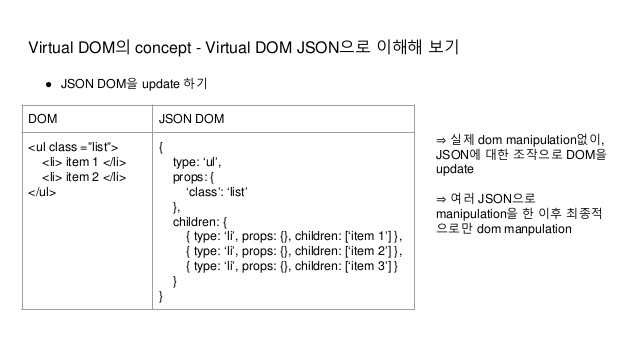


2. 虚拟DOM(Virtual DOM):

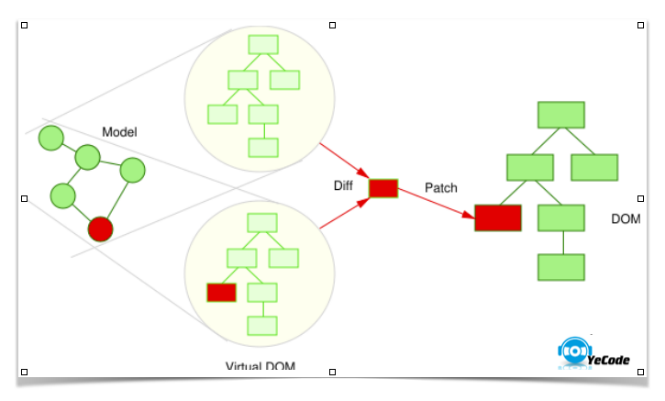
用js面向对象方式，模拟的DOM树。

保存了当前界面中所有DOM节点的描述信息，以及节点间关系。

优点1: 仅保存关注的少部分节点属性和内容，遍历快！



当响应系统通知Vue框架某个属性发生变化，要修改页面时，Vue框架会快速遍历虚拟DOM树，找出发生变化的DOM元素节点，仅修改发生变化的真实DOM树上的节点。没有发生变化的节点，保持不变。



优点2: 仅修改受影响的少量元素，效率高。——无需人工

如何使用Vue框架:

1. 创建View视图: HTML,CSS静态页面

2. 创建Vue对象实例: 包含2部分:

1. el: 用选择器指向希望Vue监视的页面元素及其子元素

强调: 即使选择器找到多个父元素，vue也只能监视第一个找到的父元素及其子元素。

2. data对象: 包含所有要显示在页面上的数据——模型变量

执行时: vue对象实例，会监视页面和数据的变化，并随时同步变化。

4. 绑定语法:

{{}}: 双括号语法，也叫大胡子语法(Mustache)

官方名字: interpolation 补缺

什么是: 可选择特定的Model中的数据，绑定到View视图中

何时: 只要希望将Model中的一个数据，显示到页面上，就在页面上用{{模型变量}}引入

如何: {{}}中可包含多种表达式:

1. 直接写模型变量名: {{uname}}

2. 运算: 算术/比较/逻辑/三目运算: {{price\*count}}

3. 方法调用: {{uname.toUpperCase()}}

4. 属性访问: {{address.city}}

5. 数组下标: {{list[2]}}

注意: 不能写if,else,for等程序结构

问题: 只能绑定元素的内容(innerHTML)，不能绑定属性和事件

解决:

5. 指令(directive):

什么是: Vue.js提供的，专门用于增强HTML功能的特殊属性

为什么: HTML本身是静态的，写死的，没有任何动态生成内容的能力。

何时: 只要希望根据内存中的数据，动态绑定HTML元素的属性和样式时，都用指令。

包括: 看手册: 官网/学习/API

如何: v-\*, 凡是v-\*开头的自定义扩展属性，都是vue为增强HTML功能而提供的指令。

1. v-bind: 专门绑定属性值:

如何: <any v-bind:属性="模型变量"

强调: 绑定属性，不用{{}}！

简写: v-bind可省略，只保留:

<any :属性="模型变量"

2. v-for: 根据数组或集合内容，反复生成多个相同的HTML元素。

何时: 只要反复生成多个相同的HTML元素时

如何: <any v-for="(元素,下标) in 数组/对象/字符串"

根据in后的集合中的元素个数，反复生成<any元素

每次in都取出元素的值和下标，保存到in前的变量中

在<any元素及其子元素中，可使用元素和下标，动态绑定内容

强调: v-for必须写在想要反复生成的元素上

ul

li v-for

简写: 如果只关心内容，不关心下标可省略()

v-for="元素 in ...

变化: 其实, in可用of代替。

3. v-if: 根据一个bool数据作为条件，控制是否创建该元素到DOM树（控制显示隐藏）

何时: 根据条件，控制一个元素的显示隐藏时

如何: <any v-if="bool值的模型变量"

<any v-if="bool"

<any v-else-if="bool"

<any v-else

强调: v-if, v-else-if, v-else之间不允许插入任何其他元素

4. v-show: 用法和v-if几乎完全一样！

v-if vs v-show:

v-if... 都是用添加删除DOM元素的方式，来控制显示隐藏

v-show: 不改变DOM的内容，通过display:none来控制显示隐藏

1. 如果只控制一个元素的显示隐藏时，首选v-show

2. 如果控制多个元素选其一显示时，首选v-if

3. 如果频繁显示隐藏切换时，首选v-show，效率高

5. v-on: 为元素绑定事件处理函数

问题: 通常vue实例对象都是没名的，为了避免冲突和全局污染。所以，在普通事件处理函数中，不可能访问vue中的模型变量。

解决: v-on

何时: 今后只要在vue中绑定事件一律用v-on:

如何: 2步:

1. 添加事件处理函数:

new Vue({

el:...,

data:{ ... },

methods:{

处理函数(){

//this->当前vue实例对象

... this.模型变量 ...

}

}

})

2. 定义绑定: <any v-on:事件名="处理函数()"

简写:

1. v-on:事件名 => @事件名

2. 如果没有参数，可省略() ——不建议

问题: 如果a当按钮, 会擅自修改url

解决: 2种:

1. e.preventDefault()

获取事件对象: 2步:

1. 在绑定时: @click="处理函数($event)"

其中: $event是Vue内置的对象，可直接使用

$event（传入的参数）名字不能变

2. 在定义处理函数时: 处理函数(e){ ... }

2. 用事件修饰符代替事件函数:

.preventDefault() => .prevent

.stopPropagation() => .stop

如何: @事件名.修饰符="处理函数()" @click.stop.prevent=..可连用且与顺序无关

提示: 如果想获得鼠标位置，就必须用$event

简写: 键盘事件绑定:

@keyup.键盘号="处理函数()"

比如: @keyup.13="处理函数()"

课后任务:   
 题目要求:

1. 实现学子商城商品详情页信息加载

2. 实现学子商城页头查找功能事件绑定

运行效果:

**VUE**

Day02

正课:

1. 指令:

2. 双向绑定:

3. 绑定class和style

4. 计算属性:

5. 自定义指令——了解

6. 过滤器:

1. 指令:

1. v-html: 使用模型数据替换当前元素的innerHTML

何时: 要用HTML代码片段替换内容

2. v-text: 使用模型数据替换当前元素的innerText

何时: 要用纯文本内容代替元素内容

{{}} vs v-text vs v-html

1. {{}}可和其它内容拼接形成新的内容

v-text和v-html只能完整替换内容

2. 如果绑定纯文本内容，则首选v-text，原样显示，不转换

如果绑定HTML片段，则首选v-html，会被浏览器转换为正文，不保持原样。

3. v-cloak: 可让未加载完的元素，暂时不显示

问题: 当vue加载慢时，页面会暂时显示{{}}内容

解决: 如果vue没有加载完，则元素不显示！

如何: 2步:

1. 为vue监视的元素指定v-cloak属性

2. 手动在css中为v-cloak指定隐藏样式:

[v-cloak]{ display:none }

原理: 当vue加载完成后，会自动移除v-cloak属性！

总结: 今后被vue监视的父元素都要用v-cloak默认隐藏，等vue加载好后，才显示出来。

4. v-pre: 保留当前元素中的{{}}，不进行绑定渲染

何时: 如果正文中有{{}}，不想被vue渲染，可用v-pre保护

5. v-once: 设置一个{{}}，仅在首次加载时渲染一次，之后不再被监视。

何时: 如果希望一个元素，只在首次加载时绑定一次

原理:

其实在Vue中有一个巨大的数组，保存着所有绑定的元素

每次模型数据有修改时，都通过遍历数组找到绑定的元素，修改元素。

v-once可在第一次加载后，将元素从监视数组中移除！之后遍历，无法遍历到该元素，则值不会被改变！

2. 双向绑定:

什么是: 即可把Model数据绑定到页面的元素中——M => V

又可把元素的内容修改，反向更新到Model中——V =>M

为什么: {{}}和v-bind，仅是单向绑定，只能将Model中的数据绑定到页面显示。无法将页面上的更改，反向更新到Model中

何时: 专门用于绑定表单元素

如何: <表单元素 v-model:value="模型变量"

可简写为: <表单元素 v-model="模型变量"

总结: 表单元素的双向绑定:

1. text, textarea: v-model="模型变量"

2. radio: name="分组名" value="写死值" v-model="模型变量"

当用户单击一个radio时，用选中的value赋值给v-model指定的模型变量。

3. select: <select v-model="模型变量"

<option value="写死的值"

...

当用户选择一个option时，option的value会给select，select将选中的value反向更新到模型变量上

4. checkbox: <input type="checkbox" v-model="模型变量"

默认: v-model绑定的是checked属性，值为bool类型

自定义监视函数:

什么是: 模型变量发生变化时，自动执行的函数

何时: 希望在每次模型变量发生变化时，都执行一项操作

如何:

new Vue({

el:"#app",

data:{ ... },

methods:{ ... },

watch:{

模型变量名(){ ... this.模型变量 ...}

}

})

3. 绑定class和style属性:

2种:

方式1: 把class和style作为普通字符串绑定

方式2: 把class和style作为对象绑定:

data:{

xxxclass:{

class1:true,

class2:false,

... : ...

}

}

绑定时: <any class="固定不变的class" :class="data中的xxxclass变量"

style的绑定:

data:{

xxxStyle:{

css属性名:值,

... : ...

}

}

绑定时: <any style="固定不变的属性" :style="data中的xxxStyle变量"

4. 计算属性:

什么是: 不实际存储属性值，而是根据其他数据属性的值，动态计算获得。

为什么: 有些属性的值，不能直接获取到，需要经过其他属性计算后，才能获得。

比如: 订单总价=每个商品的数量\*每个商品的单价，再累加求和。

FullName=FirstName+ · +LastName

何时: 今后，只要一个属性的值，无法直接获得，依赖于其他属性的值，就用计算属性临时计算获得。

如何:

1. 定义计算属性:

new Vue({

el: "selector",

data:{ ... },

methods:{ ... },

watch:{ ... },

computed:{

属性名(){ return 属性值 }

}

})

2. 绑定: 同普通属性绑定完全一样！

{{属性名}} 强调: 不加()!

vs methods:

计算属性的值可被vue缓存！只要以来的其他属性不变，则始终使用缓存中的值，避免重复计算

methods不会被缓存。每调用一次，就重复执行一次。

5. 自定义指令——了解:

何时: 如果除13种Vue指令外，还想为HTML元素添加更多的功能

如何:

1. 创建自定义指令:

Vue.directive("自定义指令名",{

inserted(el){ //当元素被加载到DOM树时触发

//el: 写有当前指令的当前DOM元素

//对el执行DOM操作

}

})

强调: 定义指令名时, "自定义指令名"不加v-前缀

在HTML中使用时，要加v-前缀！

2. 在HTML元素上使用自定义指令:

<any v-指令名

强调: 必须加v-

6. 过滤器(filter):

什么是: 在接收到原始数据后，执行再加工

强调: vue官方没有提供任何预定义的过滤器，都要自己定义。

何时: 有些数据，需要经过加工后才能给用户看

比如: ms->日期, 1,0->"男","女", 钱->¥xx.00

10,20,30->订单状态名

如何: 2步:

1. 创建自定义过滤器:

Vue.filter("过滤器名", function(val){

//val收到传入的原始值

return 修改后的新值

})

2. 在绑定时使用过滤器:

2种:

1. {{ 模型变量 | 过滤器名 | ... }}

2. <any :属性="模型变量 | 过滤器名 | ..."

多个过滤器可连续使用，但后一个过滤器时基于前一个过滤器的加工结果继续加工。

带参过滤器:

1. 定义时:

Vue.filter("过滤器名", function(val, 其它参数...){

return ...

})

2. 使用时:

{{模型变量|过滤器(参数值)}}

**VUE**

Day03

正课:

1. 组件:

2. Axios

3. Vue.js实例/组件生命周期——重点原理

4. 组件化开发——重点,难点

1. 组件(Component):

什么是: 用有专属HTML，CSS，数据的，可重用的独立页面区域

为什么: 重用！松耦合！便于团队协作和分工

何时: 今后，所有页面都是由多个组件组合而成。

凡是网页中，重要的，可重用的区域都要先封装为组件，再反复使用组件。

如何:

1. 创建一个组件:

Vue.component("组件名",{

template:`<div> html+css

html代码片段  
 </div>`,//强调: 组件模板必须有一个唯一的父元素包裹

data:function(){ //为每个组件实例创建专属的data

return {

模型变量

}

},

methods:{},

watch:{},

computed:{}

})

2. 在html视图中反复使用组件

组件其实就是一个可重用的标签: <组件名></组件名>

3. 也需要new Vue() 来监视页面

组件只能用在被Vue监视的父元素中

问题: HTML和js又掺杂在一起，不便于维护

解决: 用html5: <template>

<template>是默认不显示的元素

专门给框架使用，用于临时隐藏一段代码

何时: 今后组件的HTML片段都要先放在<template>中隐藏

如何: 2步:

1. 在<body>中用<template>封装组件的HTML片段

<template id="tplXXX">

<div>

html片段  
 </div>

</template>

强调: 1. <template>不能放在被Vue监视的父元素内

2. <template>中也必须使用唯一父元素包裹

2. 在Vue.component中用template:"选择器"找到<template>

Vue.component("组件名",{

template:"#tplXXX",

... ...

})

2. Axios:

什么是: 一个基于promise的HTTP函数库

可在任意平台发起http请求

何时: 今后在没有jquery的情况下，想要发起http/ajax请求，都用axios

为什么:

发送ajax请求: 4种:

1. 自己封装ajax函数——麻烦

2. jQuery的$.ajax——必须引入jQuery，大材小用

3. 官方曾经提供了VueResource插件——官方废弃

4. 使用第三方专门的axios库——本身与jQuery或Vue没有关系

如何:

1. 在页面中引入axios.min.js

2. 调用axios的API，发送请求:

①get请求:

axios.get("url",{

params:{

//随url发送的参数 lid:5

}

}).then(function(res){

//你要的数据: res.data

})

②post请求:

axios.post(

"url","查询字符串" //uname=dingding&upwd=123456

//"url", Qs.stringify({ 参数: 值, 参数: 值, ... })

).then(function(res){

//你要的数据: res.data

})

3. Vue.js实例/组件生命周期

生命周期: 从创建到销毁的过程

Vue.js实例/组件生命周期: 四个阶段:

1. 创建(create)阶段:

new Vue实例或组件对象创建完成

但尚未挂载DOM树

$el: undefined 没有虚拟DOM树

data: { ... }

2. 挂载(mount)阶段:

组件加载DOM树完成

$el: DOM元素

data: { ... }

3. 更新(update)阶段: 组件所绑定的model数据发生变化

4. 销毁(destroy)阶段: 调用$destroy()方法，组件被销毁。

为了在每个阶段，触发执行一项任务，Vue为每个阶段定义了生命周期钩子函数

8个钩子函数:

1. beforeCreate(){}

2. created(){} ——可ajax请求获取数据，不可DOM操作

3. beforeMount(){}

4. mounted(){} ——可ajax请求，又可DOM操作

5. beforeUpdate(){}

6. updated(){}

7. beforeDestroy(){}

8. destroyed(){}

new Vue({

el:"#main>div:nth-child(2)>h3:first-child",

data:{

res

},

created(){ //将axios写在created而不是写在mounted,是将返回数据在DOM树挂载前就填充数据

//this->当前vue

(async function(self){

var res=await axios.get(

"http://localhost:3000/index/"

);

self.res=res.data;

})(this)

}

})

4. 组件化开发:

全局组件: 可在页面中任何位置使用的组件

Vue.component()

局部组件: 只能用在一个特定的父组件内

如何:

拿到网页后，先划分区域

每个区域，再包含子区域

每个区域都可做成一个组件:

1. 将该区域的HTML代码放入一个单独的template中

2. 先定义子组件:

var 子组件名={ //子组件本质是一个普通对象

//只是属性结构和Vue组件内容一致

template:"#tplXXX",

data(){ return { ... } }

}

3. 再定义全局组件:

Vue.component("组件名",{

template:"#tplXXX",

data(){ return { ... } }

})

4. 在全局组件或new Vue({})中，用components属性，引入子组件:

Vue.component("父组件",{

template:

data(){ return { ... } }

components:{

子组件名: 子组件对象

//子组件名和子组件对象名都要用驼峰命名

//如果子组件名和子组件对象名相同，可只写一个

//Vue会自动将驼峰命名的子组件名，转化为横线分割: todoAdd自动转为<todo-add>; todoList 自动转为 <todo-list>

}

})

一个页面，都是由多块区域，多个组件

以后尽量不要用花括号给值，用v-text，这样数据没回来时，页面也不会出现花括号，只是不显示内容而已

<h5 class="d-inline-block mb-2" v-text="res[0].title"></h5>

<span class="text-primary" v-text=" '¥'+res[0].price.toFixed(2)"></span>

**VUE**

Day04

正课:

1. 组件间通信——重点&难点

2. SPA应用

3. Vue-router组件——路由

1. 组件间通信:

问题: 父组件的数据，子组件无法自动使用

子组件无法修改父组件的数据

兄弟组件之间的数据无法直接使用

2大类关系:

1. 父子间

2种:

1. 父给子: 属性下行

如何: 2步:

1. 子: 先定义自定义属性:

var 子={

template:

data(){ return {... } },

props:[ "自定义属性名",... ],

... ...

}

props中的属性: 2用:

1. 同data中的属性，可在组件内自己使用

2. 可在组件外，被别人赋值

2. 父: <template>中

<子 :子的自定义属性="父组件模型变量"

2. 子给父: 事件上行

如何:

1. 父:

html: <template id="父"

<子 @自定义事件="父组件处理函数"></子>

js: Vue.component("父",{

template:

data:

methods:{

处理函数(参数列表){

修改父组件自己的模型变量

}

}

})

2. 子:

var 子={

template:

data:

methods:{

//触发绑定在自己身上的自定义事件

this.$emit("自定义事件名",参数值列表)

}

}

2. 兄弟间

2种:

1. 先给父再给子——强烈不推荐

2. 直接给兄弟:

用"总线"机制

3步:

1. 创建空的vue实例，作为数据总线

var bus=new Vue()

2. 接收者，先在vue实例上绑定事件，并定义处理函数

接受者组件{

mounted(){

bus.$on("自定义事件名",this.处理函数)

},

methods:{

处理函数(参数列表){ 修改自己的数据 }

}

}

3. 发送者，触发vue实例上绑定的事件，并传参

可在任何时候任何位置:

bus.$emit("bus上的事件名", 参数值列表 )

2. SPA应用: (Single Page Application) 单页面应用

什么是SPA应用: 整个程序只有一个完整的HTML页面

每次"跳转"，只是用一个html片段，代替页面中另一个HTML判断

vs 多页面应用:

|  |  |
| --- | --- |
| 多页面应用 | 单页面 |
| 项目中有多个独立的HTML文件  缺: 反复请求 | 每个项目中只有一个完整的HTML页面，其它"页面"只是HTML片段而已  优: 请求少  缺: 不利于搜索优化 |
| 页面跳转的本质: 在浏览器中删除旧的DOM树，重新下载并创建新的DOM树  缺: 效率低 | 页面跳转的本质:  只是删除个别div内容，替换为新的div  优: 效率高 |
| 如果两个页面都需要某个资源，多页面应用必须重复下载。 | 如果两个页面都需要某个资源，不需要重复下载 |
| 多页面不可能有页面切换动效 | 单页面应用容易实现页面切换动效 |

SPA应用的实现原理:

1. 定义多个模板

2. 定义一个完整的HTML页面

页面中提供一个可被找到，并替换的div容器

3. 创建一个路由与模板的对应字典:

路由: 一个路径和一个组件对象的配对儿

为什么: 帮助框架在路径发生变化时，找到要替换的新组件

如何: [

{path:"路径1", component: 组件对象1}，

{path:"路径2", component: 组件对象2}，

{path:"路径3", component: 组件对象3}

]

4. 每当地址发生变化时，都获得路径，根据路径获得对应的组件。用组件内容代替页面中占位的div的位置。

4. Vue-router组件

什么是: Vue中专门实现单页面应用的组件

专门识别路径，并加载不同的路由组件模块

何时: 只要用Vue实现单页面应用，都用Vue-router

如何: 2部分：

1. 配置路由和组件模块

1.1 引入vue.js和vue-router.js

1.2 封装组件: 2步:

1. 定义组件<template>

2. 在js中将<template>包装为组件对象

1.3 定义路由列表:

var routes=[

{path:"/路径名",component: 组件对象},

...

{path:"/\*",component: NotFound}

]

1.4 将路由列表封装为路由器对象

var router=new VueRouter({ routes })

1.5 在HTML的body中定义:

<div id="app">

....

<router-view></router-view>

1.6 在js中:

new Vue({

el:"#app",

router

})

强调: Vue-router默认用#/xxx方式作为路由地址

2. 跳转

html中:

<a href="#/xxx"> ——强烈不推荐

<router-link to="/xxx">代替<a>

原理: router-link 会被自动翻译为<a>

js中: vue中:

this.$router.push("/xxx")

**VUE**

Day05

正课:

1. Vue-CLI脚手架

1. Vue-CLI脚手架:

什么是脚手架: 具有核心功能的半成品代码

如何:

创建脚手架项目:

1. 下载Vue-CLI工具: npm i -g @vue/cli

2. 用vue命令创建项目脚手架代码:

vue create xz\_vue

选项:

1. 选Manually select features 手动选择功能

2. 选择组件: 空格: 选中/取消选中

Babel: 脚手架中包含浏览器不认识的ES6代码

要靠Babel将ES6代码转为ES5代码

比如: Vue脚手架中: ES6

引入模块: import 模块名 from 路径

抛出模块: export default{ ... }

Router: 路由器组件

Vuex: 状态管理

3. Use history mode for router? 使用不带#的方式执行跳转

选Y

4. 选In package.json，所有组件的配置信息，集中放在一个package.json中

5. Save this as a preset for future projects? (y/N) 是否将当前项目的配置，作为今后项目的模板, 选N

6. Your connection to the default npm registry seems to be slow.

Use https://registry.npm.taobao.org for faster installation? (Y/n) 选Y

脚手架项目结构:

public文件夹: 不会被打包编译的可直接访问的静态资源

包括: img文件夹: 访问路径: http://域名/

公用的css文件: 比如: bootstrap.css, base.css

公用的js文件: 比如: jquery-3.2.1.js, bootstrap.js

唯一完整的index.html页面: 引入公用的css文件和公用的js文件

src文件夹: 包含了项目的所有源代码

assets文件夹: 保存需要打包的静态资源

css: 放置公用的需要被打包的css文件

js: 放置公用的需要被打包的js文件

views文件夹: 路由组件，所有出现在路由表中的组件

普通网站有几个.html页面, views下就应该对应有几个vue组件

每个组件包含三部分：

<template>

<div>

组件的html内容  
 <div>

</template>

<script>

import 子组件 from "子组件路径"

export default{

data(){

return {

}

},

components:{

子组件对象

},

... ...

}

</script>

<style scoped>

仅当前组件可用

</style>

components文件夹: 专门保存子组件的文件夹

至少: header和footer

子组件内容和视图组件内容完全相同: 也是三部分组成

如何在views下的视图组件中引入components下的子组件:

1. <script>

import XzHeader from "@/components/XzHeader.vue"

2. export default{

...

components:{

XzHeader

}

}

3. <template>

<div>

<xz-header></xz-header>

... ...

App.vue: 项目中唯一的根组件:

其中包含<router-view></router-view>

运行时，会根据路由，选择Views中的某一个.vue内容替换app.vue中的<router-view>

main.js: 对单页面应用中唯一new Vue实例的配置

通常: 配置axios:

import axios from "axios"

Vue.prototype.$http=axios

将来在其它.vue中，就可this.$http.get() => axios.get()

router.js: 路由器

1. import views文件夹下的视图组件

比如: import Index from "./views/Index.vue"

2. 在export default{

... ,

routes:[

{

path:"/",

name:"路径别名",

component: import引入的组件对象

}

]

}