JavaScript进阶

- JavaScript基本数据类型总结
  - String
    - String构造器方法（静态方法）
      - String.fromCharCode()：
        - 参数为一系列UTF-16代码单元的数字
        - 返回值为一个长度为n的字符串
      - String.fromCharCode.apply()：
        - 返回值为一系列UTF-16代码单元的字符串
    - String原型方法（提取字符串）
      - String.prototype.charAt()：从一个字符串中返回指定的字符
      - String.prototype.slice(begin,end)：截取字符串，开始和结束的字符串
      - String.prototype.substr(start,length)：返回一个字符串中从指定位置开始到指定字符数的字符串
      - String.prototype.substring(indexstart, indexend)：返回一个字符串在开始索引和结束索引之间的一个子集，或从开始索引到字符串末尾的一个子集
      - String.prototype.split()：用指定的分隔符分割成字符串数组，如果空字符串被用作分隔符，则会在每个字符之间分割
    - String原型方法
      - 字符串的变换
        - String.prototype.trim()：从字符串的两端删除空白字符，返回一个新的字符串
        - String.prototype.concat()：将一个或多个字符串与原字符串连接合并，形成新的字符串并返回
        - String.prototype.toLowerCase()：将调用该方法的字符串值转为小写形式
      - 检查和比较
        - String.prototype.indexOf()：返回第一次出现指定值的索引，若未找到则返回-1
        - String.prototype.lastIndexOf()：返回指定值在调用该方法的字符串中最后出现的位置，未找到则返回-1
          从这个字符串的后面向前查找，从fromIndex处开始
        - String.prototype.localeCompare()：返回一个数字指示给定字符串与参考字符串的对比顺序。如果引用字符串在于比较字符之前则为负数，否则为正数，相等则为0
    - Number
      - Number的原型方法
        - Number.isFinite()：用来检查传入的参数是否为有穷数
        - Number.prototype.toFixed()：用定点表示法格式化一个数值
        - Number.prototype.toString()：返回字符串
        - Number.prototype.toPrecision()：以指定精度返回该数值对象的字符串
        - Number.prototype.toExponential()：以指数表示法返回该数值字符串
      - Number的构造器属性（静态属性）
        - Number.NaN
        - Number.prototype
- JavaScript概念性总结
  - 什么是闭包？
    - 闭包是函数和与其相关的引用环境组合而成的实体
    - 闭包是词法作用域中的函数和其相关变量的检查体
      - function createInc(startValue){
        return function(step){
        startValue+=step;
        return startValue;
        }
        }
        //即两次输出中，startValue常驻内存？
        var inc = createInc(5);
        console.log(inc(1));//6
        console.log(inc(2));//8
        //第二次调用函数，新建了一个闭包，startValue重新创建
        inc = createInc(5);
        console.log(inc(1));//6
    - 闭包的常见形式
      - 函数bar和其相关词法上下文中的变量，构成了一个闭包
        - function foo(){
        var i = 0;
        function bar(){
        console.log(++i);
        }
        return bar;
        }
        var a = foo();
        var b = foo();
        a();//1
        a();//2
        b();//1
      - 以函数对象形式返回
        - var tmp = 100;
        function foo(x){
        var tmp = 3;
        return function(y){
        console.log(x+y++(++tmp));
        }
        }
        var fee = foo(2);
        fee(10);//16
        fee(10);//17
        fee(10);//18
      - 作为对象的方法返回
        - function counter(){
        var n = 0;
        return{
        count:function(){return ++n},
        reset:function(){n =0;return n;}
        }
        }
        var c = counter();console.log(c.count());//1
        console.log(c.count());//2
        console.log(c.reset());//0
        console.log(c.count());//1
        console.log(c.count());//2
    - 闭包的作用及常用场景
      - 可通过闭包来访问隐藏在函数作用域内的局部变量
      - 使函数中的变量被保存在内存中不被释放
        - function f1(){
        var n = 999;
        function f2(){
        console.log(++n);
        }
        return f2;
        }
        var f = f1();
        f();//1000
        f();//1001
  - 什么是IIFE？
    - Immediately- Invoked Function Expression：立即执行的函数表达式
    - 作用：建立函数作用域，解决ES5作用域缺乏带来的问题，如：变量污染，变量共享等
    - IIFE的写法
      - 使用小括号的写法
      - 与运算符结合的写法
    - 通过IIFE3的作用域的改变（限制变量生命周期）
      - JS中没有块作用域，首要造成x文件内或文件间的同名变量相互污染
        - var userId = 12;
        document.cell(id + function () {
        console.log("userId = ",userId);
        });
        // 一个文件引入，定义了id变量userID
        var p=[1,3];
        if(p[0]
        var userId = 3;//重定义，变量污染
        }
        - function f() { // IIFE不用
        var a = 10;
        function g() {
        console.log(a * a);
        }
        g();
        }
        f();// IIFE用法
        - (function () { // IIFE不用
        var a = 10;
        function g() {
        console.log(a * a);
        }
        g();
        }
)();// IIFE用法
      - 通过IIFE对变量存储的改变（避免变量共享问题）
        - 页面导航问题
        - 定时器问题
  - 什么是事件的冒泡与捕获？
    - 事件类型从页面中接收事件的顺序，包括冒泡法和捕获法
    - 冒泡：从最具体的节点到最不具体的节点
      - e.stopPropagation()阻止事件冒泡
      - e.preventDefault()阻止默认响应
    - 捕获：从最不具体的节点到最具体的节点
  - let与var定义变量方式的区别？
    - 声明时未赋值，输出结果为undefined
    - 使用未声明的变量：
      var直接报错
      let输出undefined
    - 重复声明同一个变量：
      var 在重复第一次声明的语句
      let重复声明会报错
    - 变量的作用域不同：
      var 声明的是全局变量
      let声明的变量只能在当前作用域使用，与其他作用域中的let声明互不影响
  - 事件冒泡从页面中接收事件的顺序，包括冒泡法和捕获法
- JavaScript内置对象
  - Array
    - push()
    - shift()
    - pop()
    - splice()
      - 参数：start ,deleteCount ,item
      从第几个元素开始，删除几个元素，字符串
    - reverse()
    - sort()
    - concat()
    - slice()
      - arr.slice(), arr.slice(begin), arr.slice(begin,end)
    - join()
    - forEach()
    - indexOf()
    - map()
  - Date
    - now()
    - getDate()
    - getDay()
    - getFullYear()
    - setDate()
    - toJSON()
    - toISOString()
    - toString()
  - RegExp
    - 创建
      - 通过字面量直接创建
      var reg1 = /[a-z]+[0]/g;
      - 通过RegExp构造函数实例化出时对象
      var reg = new RegExp("[a-z]+", "g");
    - g全局，i忽略大小写，m包含换行
    - 语法
      - 以数字字符类，如[abc]代表abc中的任意一个字符，可以配合范围符号
        - [^]代表字符类取反，如[^abc]代表非abc中的任意一个字符
        - -：代表范围，如[a-z]
        - .：代表一个除了换行和换行符之外的所有字符，等效于[^\r\n]
        - *出现0次或以上（最多1次）
        +出现1次或多次（至少1次）
        ?出现0次或多次（任意次）
        {n}出现n次
        {n,m}出现n-m次
        {n,}出现至少n次
      - 定义属性和方法
        - test()
        - exec()
        - String.prototype.search()
        - String.prototype.match()
  - Error
    - name()
    - message()
  - Math
    - 属性
      - Math.PI
      - Math.E
      - Math.abs(x)
    - 方法
      - Math.ceil(x)
      - Math.max()
      - Math.random()
  - JSON
    - JSON.parse()
    - JSON.stringify()
- jQuery相关方法总结
  - jQuery选择器
    - 基本选择器
      - $("*")：所有元素
      - $("#id")：根据id选取
      - $(".class")：根据class选取
      - $("element")：根据标签选取
      - $(".class.class")：选取两个class
    - 过滤选择器
      - $("p:first")：第一个元素
      - $("p:last")：最后一个元素
      - $("tr:even")：所有偶数元素
      - $("tr:odd")：所有奇数元素
      - $("ul li:eq(index)")：列表中第index元素
      - $("ul li:gt(3)")：列出index大于3的元素
      - $("ul li:lt(3)")：列出index小于3的元素
      - $("input:not(:empty)")：所有不为空的input元素
    - 表单选择器
      - $("input")：所有input元素
      - $("text")：所有type = "text"的input元素
  - jQuery事件
    - 事件名称去掉on即可
  - jQuery DOM操作
- Object相关静态方法及原型方法总结
  - JS对象是若干无序属性的集合，可以直接通过属性名来访问对象的属性
  - 函数作为第一对象的属性时，称为该对象的方法
  - 对象的分类
    - 内置对象
    - 宿主对象
    - 自定义对象
    -

  - JS对象的相关操作
    - 通过对象字面量的方式直接创建
    - 通过Object的create静态方法创建
    - 通过构造函数的方式创建
  - 对象的属性
    - Object.prototype.setTable：
    - Object.prototype.constructor
  - Object的静态方法
    - Object.getOwnPropertyNames()
    - Object.getPrototypeOf()
    - Object.defineProperties()
    - Object.getOwnPropertyDescriptor()
    - Object.prototype.hasOwnProperty()
    - Object.keys()
- JavaScript中this指向总结
  - 动态绑定（运行期绑定），可以是全局对象，当前对象或任意对象
  - 对所在环境是（this指向的都是调用此函数的主体）
    - 一般函数下的this
      - 非严格松散模式下指代全局对象
      - 严格模式下为undefined
    - 对象方法中的this
      - 指代调用该对象（尤其重）
    - 构造函数中的this
      - 指代通过new调用的对象
    - 间接调用中的this
      - 通过call/apply间接调用的函数中的this
      - JS中函数也是对象，也有属性和方法
      - 动态的原型方法非常调用此函数
  - 通过this在函数内添加、删除、修改全局对象属性
  - this不进行作用域传递，函数嵌套中的this存在缺陷
    -
    - this指向解决方法
      - 1.使用变量（that或self）锁死，使this指向正确
        -
      - 2.使用call/apply间接调用
        -
      - 3.使用Function.prototype.bind
        -
    - 构造函数中的this（构造函数嵌套的情况下）
- JavaScript继承方式总结
  - 每个对象都有一个原型对象，最原始的原型是null
  - 对象-对象的原型继承，任何方式创建的对象都有原型对象，可以通过对象的__proto__属性来访问原型对象
  - js对象的原型链
    -

  - 基于构造函数实现的原型继承
    - 当一个函数与new结合，该函数将作为构造函数来使用，用来创建Js对象
      - js中通过构造函数来实现类的功能
        - function Person(age,name){
        this.name = name;
        this.age = age;
        }
        Person.prototype.say4li = function(){
        console.log("hi,I'm" + this.name);
        }
        var p1 = new Person(20,'shangsan');
      - 构造函数的prototype属性就是实例化出来的对象的原型，实例化的对象通过prototype实现属性的继承
      - 实例化对象的__proto__属性指向原型
      - 实例化对象的__proto__属性指向构造函数的prototype属性

XMind:ZEN 试用模式