**正课:**

**1. ES5**

**2. ES6**

**1. ES5**

**Object.create(): 仅基于一个现有父对象，就可创建子对象**

**何时: 如果只有对象，没有构造函数也想创建子对象时**

**如何: var child=Object.create(father,{**

**//definePropertites**

**属性名:{**

**特性:值,**

**... :...**

**},**

**属性名:{**

**... : ...**

**}**

**})**

**创建一个新对象child，并让child继承father，还可给child添加新自有属性**

**替换this: 3种方法:**

**1. call/apply: 立刻调用一次函数，并临时替换this**

**如果给定的参数值，是放在数组中的，可将call换成apply。apply可打散数组参数为单个值**

**2. bind: 基于原函数，创建一个一模一样的新函数，并永久绑定this。**

**特殊场景: 所有回调函数中的this要绑定，都用bind**

**数组API:**

**1. 查找元素:**

**var i=arr.indexOf(元素,starti)**

**用法同str.indexOf()完全一样**

**2. 判断:**

**判断数组中的元素是否符合要求: 2种:**

**every() 判断数组中是否所有元素都符合要求**

**some() 判断数组中是否包含符合要求的元素**

**如何:**

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

**//this->window，所以不能用**

**//elem: 当前元素值**

**//i: 当前位置**

**//arr: 当前数组**

**return 判断条件**

**})**

**every会拿着回调函数在arr中每个元素上执行一次，回调函数返回当前元素是否符合要求。如果所有元素都符合要求，整个every就返回true。只要有一个不符合要求，就返回false。**

**3. 遍历API:**

**对每个元素执行相同的操作: 2种:**

**1. forEach() 对原数组中每个元素执行相同的操作，执行后，再存入原数组中。**

**2. 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。原数组不变。**

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

**arr[i]=新值**

**})**

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

**return 新值**

**})**

**4. 过滤和汇总:**

**过滤: 复制出原数组中符合条件的元素组成新数组返回**

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

**return 判断条件**

**})**

**汇总: 对数组中所有元素进行统计，并返回统计结果**

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

**return 将elem和现有prev统计后，汇总回prev中**

**}, 起始值)**

**2. ES6 : 不改变原理的基础上，尽量简化代码**

**学ES6同时，要学替代的ES5方法**

**let: 代替var 用于声明变量**

**var的问题: 2种:**

**1. 声明提前**

**2. 没有块级作用域**

**let的原理: 匿名函数自调！**

**let的优点:**

**1. 避免声明提前**

**2. 添加块级作用域**

**参数增强:**

**1. 默认参数: function fun(形参, ..., 形参=默认值)**

**强调: 带默认值的形参，必须放在形参列表末尾**

**2. 剩余参数: 代替arguments**

**arguments的问题2:**

**1. 不是纯正的数组类型，无法使用数组的API**

**2. 只能获得全部参数，不能有选择的获得部分。**

**如何使用剩余参数: 2步:**

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

**2. 函数中使用剩余参数: 直接用"数组名"**

**优: 1. 纯正的数组！**

**2. 可有选择的获得部分参数**

**兼容: 将arguments转换为数组: 强行调用数组的slice**

**var arr=Array.prototype.slice.call(arguments) 复制全部参数**

**Array.prototype.slice.call(arguments,starti)**

**复制starti开始之后的剩余参数**

**其实可简写为: [].slice.call(arguments)**

**3. 散播(spread): 代替apply，专门打散数组参数为单个值**

**apply的问题: 本意是替换this，不是专门打散数组用的**

**如何: 在函数调用时: fun(...数组)**

**箭头函数: 专门简化回调函数或匿名函数自调**

**1. 去function改=>**

**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，就要继续省略return**

**强调: 箭头函数中的this，内外相同**

**何时: 如果希望回调函数中的this和外部的this保持一致时，就可改为箭头函数**

**反例: 如果不希望内外this通用时，就不能改成箭头函数**

**模板字符串: 代替+做字符串拼接**

**1. 用``包裹**

**2. 支持换行,"",''等**

**3. 支持动态生成内容: ${js表达式}**

**解构: 简化使用对象的成员**

**简化批量赋值**

**什么是: 将一个对象中的成员，提取出来，单独使用**

**何时: 将一个大的对象，拆成多个小成员单独使用时**

**如何: 3种:**

**1. 数组解构: 下标对下标**

**var [变量1,变量2,...]=[元素1,元素2,...]**

**0 1 0 1**

**变量1=元素1**

**变量2=元素2**

**... = ...**

**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, ... }**

**变量1=值1, 变量2=值2**

**简写: 如果等号左边{}中，属性名和变量名相同，可只写一个即可**

**var {属性1,属性2, ...}=对象**

**属性1 => 属性1:变量1**

**总结: 今后解构一个对象时，尽量使用对象的原有属性名作为变量名**

**3. 参数解构: 其实就是对象解构应用在函数中**

**何时: 如果函数的参数不确定顺序时**

**如何: 2步:**

**1. 定义函数时:**

**function fun({**

**属性名1:形参1,**

**属性名2:形参2,**

**}){**

**形参1=值1, 形参2=值2**

**... ...**

**}**

**2. 调用时: fun({属性名1:值1, 属性名2:值2,...})**

**优点: 与参数顺序和个数无关。**