**正课:**

**1. RegExp对象**

**2. Function**

**1. RegExp对象:**

**API: 2个:**

**1. 验证字符串的格式:**

**var bool=reg.test(str)**

**检查str是否符合reg的格式要求**

**返回值: bool**

**问题: 正则表达式默认只要在str中找到匹配的内容就返回true。不要求完全匹配**

**解决: 今后凡是验证，必须前加^，后加$，表示从头到尾必须完整匹配**

**2. 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: 一次只返回一个关键词的内容和位置——同match不加g的情况一样**

**arr:[ "0": "关键词", "index": i ]**

**但是，每执行一次exec，就自动找下一个！**

**如果找不到，返回null**

**原理: reg对象中有一个lastIndex属性，表示下次开始查找的位置。exec每执行一次，就自动修改reg.lastIndex为本次的index+关键词的.length。**

**2. Function**

**什么是: 保存一段可重用的代码段的程序结构**

**何时: 只要一段代码可能被反复使用时。**

**如何: 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)**

**2. 用赋值方式:**

**var函数名=function (形参列表){**

**函数体**

**return 返回值**

**}**

**揭示了: 函数也是一个对象**

**函数名其实只是一个变量**

**函数名变量通过对象地址引用着函数对象**

**优点: 不会被声明提前——因为赋值留在原地**

**3. 用new:**

**var fun=new Function("形参","形参",...,"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，动态选择匹配的函数执行**

**为什么: 减少API的数量，减轻调用者的负担**

**何时: 1件事，可根据给的参数不同，执行不同的操作时**

**如何:**

**问题: js不支持重载语法！**

**因为: js不允许多个同名函数同时存在，最后一个同名函数会覆盖之前所有。**

**解决: 借助arguments对象**

**什么是arguments: 每个函数内自带的**

**专门接受所有传入函数的参数值的**

**类数组对象**

**何时: 将来不确定调用函数时会传入几个参数值**

**如何:**

**1. 不用创建，直接使用**

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

**不同: 类型不同，API不通用!**

**匿名函数:**

**什么是: 创建函数时不制定函数名的函数**

**为什么: 2个目的:**

**1. 节约内存**

**2. 临时使用划分临时作用域**

**何时: 只要一个函数，只使用一次时，都不起名**

**如何: 2种:**

**1. 回调函数:**

**什么是: 你定义的但是交给别的函数去调用**

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

**xhr.onreadystatechange=function(res){...}**

**str.replace(/正则/,function(kword){ ... })**

**2. 匿名函数自调:**

**什么是: 定义函数后，立刻调用自己**

**为什么: 全局变量极容易造成全局污染**

**全局污染: 一个变量不再使用，但空间不释放**

**何时: 划分临时作用域，避免使用全局变量**

**如何: (function(形参列表){**

**... ...**

**})(实参列表)**

**总结: 1. 今后禁止使用全局变量**

**2. 所有js代码都应该用匿名函数自调包裹起来**

**作用域和作用域链:**

**作用域(scope):**

**什么是: 一个变量的可用范围**

**其实, 每个作用域都是一个保存变量的对象**

**为什么: 防止内外变量之间互相干扰**

**包括: 2种:**

**1. 全局作用域:**

**什么是: 保存全局变量，从程序任何位置都可访问的作用域**

**全局变量: 随处可用，可反复使用**

**2. 函数作用域:**

**什么是: 保存局部变量，仅函数内可用的作用域**

**局部变量: 仅函数内可用，不可反复使用**

**强调: js中没有块级作用域:**

**比如: if...else for... 不是作用域**

**原理:**

**1. 开始执行前:**

**1. 创建执行环境栈数组: 用于记录将来调用的函数**

**2. 调用浏览器软件的主程序main()函数**

**3. main()函数会创建并引用window对象**

**此时window对象中已经包含了很对内置的函数和对象： alert() prompt() document console**

**2. 定义函数时:**

**1. 创建函数对象**

**2. 在window中添加函数名变量**

**3. 函数名变量引用函数对象**

**4. 函数对象用scope属性引用回window**

**3. 函数调用时:**

**1. 在执行环境栈数组中添加本次函数调用的记录**

**2. 为本次函数调用临时创建函数作用域对象**

**函数作用域对象的parent属性指向window**

**3. 在函数作用域对象中临时创建局部变量**

**4. 函数执行过程中:**

**优先在函数作用域对象中查找变量使用**

**如果找不到才被迫去window找**

**4. 函数调用后:**

**本次函数调用的记录出栈**

**导致函数作用域对象释放**

**导致局部变量一同释放——局部变量不可重用**

**作用域链scope chain**

**什么是: 由多级作用域对象，逐级引用形成的链式结构**

**保存着所有的变量: 全局变量和局部变量**

**控制着变量的使用顺序: 先局部，后全局**

**\*\*\*\*\*闭包:**

**什么是: 既重用变量，又保护变量不被污染的一种机制**

**为什么:**

**全局变量:**

**优: 可反复使用**

**缺: 随处可用**

**局部变量:**

**优: 仅函数内可用**

**缺: 不可重用**

**何时: 为一个函数永久绑定一个专属的可重用变量时**

**如何: 3步:**

**1. 用外层函数包裹要保护的变量和内层函数**

**2. 外层函数将内层函数对象返回到外部**

**3. 调用外层函数，获得内层函数对象，保存在变量中**

**闭包形成的原因: 外层函数的作用域对象在调用后无法释放**

**缺点: 闭包比普通函数多占用内存**

**鄙视: 画简图:**

**1. 找受保护的变量**

**2. 找外层函数共向外抛出了几个内层函数对象:**

**3种情况:**

**1. return function(){ ... }**

**2. 直接给全局变量赋值一个函数！**

**3. 将函数包裹在对象或数组中返回**