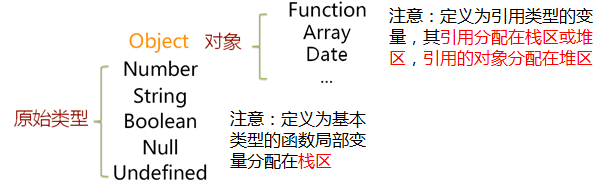
JS数据类型、值与类型转换

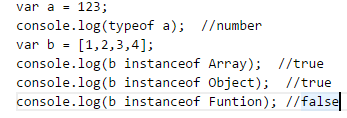
JS的数据类型

JS的数据类型可划分为基本（原始）数据（Undefined String Number Boolean Null）和对象（引用）数据(Object（Array、Function、Date、Error等）)



数据类型检测方法有：typeof instanceof

typeof检测基本数据类型 instanceof检测引用对象类型

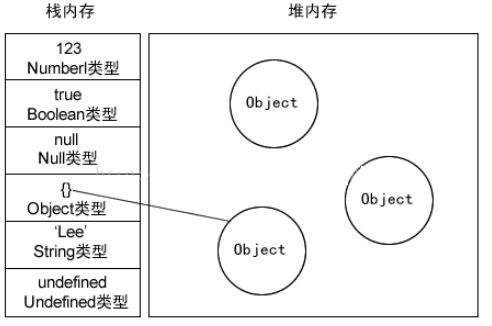


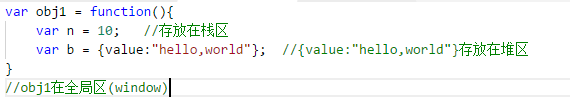
**基本类型与引用类型的区别**

1. **内存分配方式不同**

----堆区与栈区、存值与存地址、影响变量的生命周期（自动清除、垃圾回收）  
----函数内定义的基本数据类型的临时变量分配在栈区  
----引用数据类型的变量的引用（地址）存储在栈区或堆区，被引用（指向）的对象存储在堆区

（内存中没有任何引用时，清空堆区内容）





1. **赋值时不同**

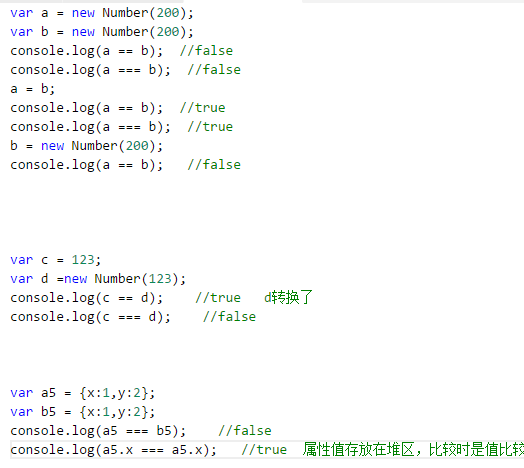
--- 赋值、赋引用（地址）、深拷贝与浅拷贝

引用和引用赋值时指向同一内存空间



1. **判等时不同**

--- 值类型是判断变量的值是否相等（值比较）  
---引用类型是判断所指向的内存空间是否相同（引用比较）



1. **函数传递参数时不同**

---按值传递  
---按引用传递

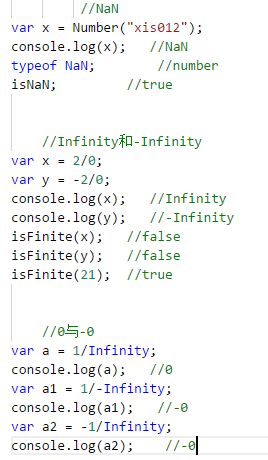


**不同类型的值**

**基本数据类型的值**

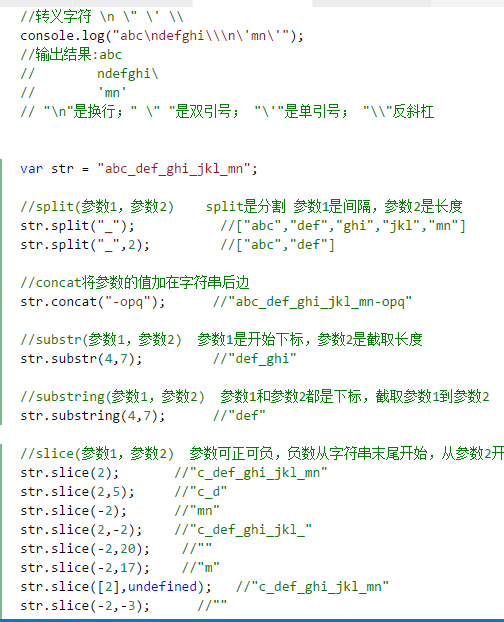
Number类型的值

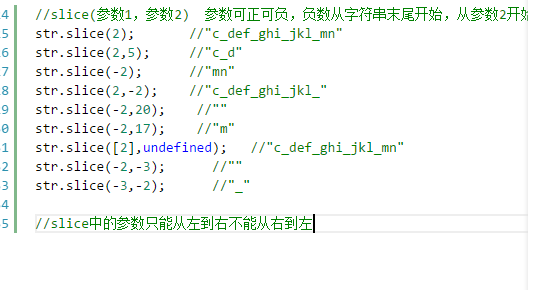
---整数与浮点数  
--- NaN、Infinity（正无穷）、-Infinity（负无穷）、+0、-0



String类型的值

---空字符、字符和字符串、转义字符





Boolean类型的值

- true、false

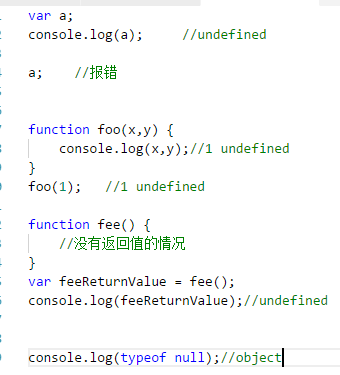
Null与Undefined

unfefined 表示“没有值”（既不是原始值也不是对象）。访问未初始化的变量、缺失的参数，以及缺失的属性会返回这个。函数中没有任何返回值时也会返回undefined

null表示“没有对象”。在用到对象的时候他表示空值。

null是原型链最顶端的元素，一切皆源于空。





**引用数据类型的值**

简单对象  
- 例：var obj = {name：“Jack”，age：20};

数组  
- 例：var arr = [1,2,true,“Hi”];

函数对象  
- 例：var foo = function(x,y){...};//函数也是对象（可执行的对象），也有属性和方法

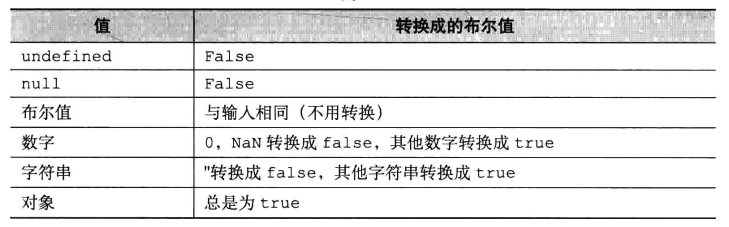
正则对象  
- 例：var reg = /^a+b+$/;

包装对象  
- 数字、布尔、字符串等基本数据类型都有对应的包装对象类型，可以将其包装成对象  
- 例：new Number(20)； new String('SomeStr');//装箱  
- 存储或读取基本类型（字符串、数字、布尔）值的属性时，会创建临时包装对象  
- 例: console.log('Hello，World'.length);  
- 基本类型其属性不能被改变、添加或删除（原始值不可变性）

临时对象在使用之后立即释放  
- 例：var str=”test”;  
 str.p = 4；//设置临时对象属性  
 var t = str.p; // 临时对象已释放，再输出t时为undefined

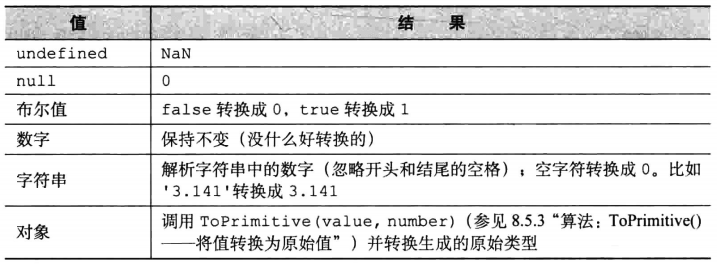
数据类型的转换

**其他类型转换为Boolean类型**



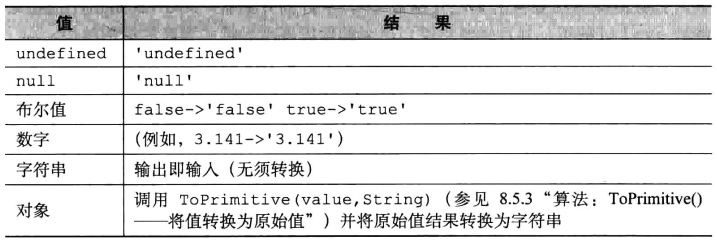
转换方式  
- Boolean（）、value？true：false、！！value

**其他类型转换为Number类型**



转换方式  
- Number（）、+value、parseFloat、parseInt

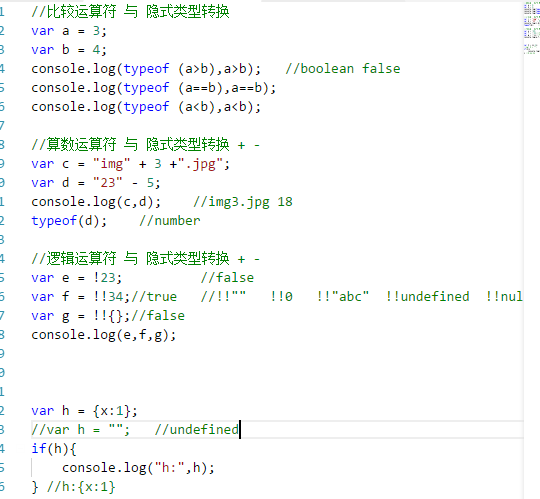
**其他类型转换为String类型**

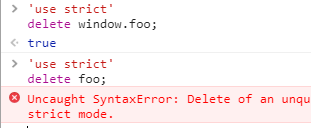


转换方式  
- String（）、‘’+value、 value.toString();

**隐式类型转换**- 使用关系运算符时的转换（==、>、<、引用类型和基本类型比较时）  
- 使用算数运算符时的转换（'img'+ 3 + '.jpg'; “25”-0;）  
- 使用逻辑运算符时的转换（ !!0; ）  
- 执行流程语句时的转换（if(obj){...}）

**显式类型转换（使代码更清晰）**  
- Boolean（）、Number（）、String（）、Object（）  
- 数转为字符串（toString()、toFixed()、toPrecision()、toExponential()）  
- 字符串转为数字（parseInt()、parseFloat()）  
- 对象转换为原始值（toString()、valueOf()）





基本（原始）数据类型进阶

1. Number

Number基本数据类型变量有与其对应的Number包装对象，当访问Number基本数据类型属性或方法时创建临时包装对象，访问的都是对象中的属性或方法（注：访问对象属性时，首先访问自身属性，访问不到时，则会在原型链上寻找对应的属性和方法）

**Number原型方法(Number对象继承的方法）**

---Number.prototype.toFixed(...)、Number.prototype.toPrecision(...)  
---Number.prototype.toString(...)、Number.prototype.toExponential(...)

**Number构造器属性（静态属性）**

--- Number.MAX\_VALUE、Number.MIN\_VALUE

--- Number.NaN、Number.NEGATIVE\_INFINITY、Number.POSITIVE\_INFINITY

**Number函数的使用**

---类型转换、实例化Number对象（原始值的包装对象）

**Math对象：**

|  |  |
| --- | --- |
| [ceil(x)](http://www.w3school.com.cn/jsref/jsref_ceil.asp) | 对数进行上舍入 |
| [floor(x)](http://www.w3school.com.cn/jsref/jsref_floor.asp) | 对数进行下舍入。 |
| [round(x)](http://www.w3school.com.cn/jsref/jsref_round.asp) | 把数四舍五入为最接近的整数 |

1. String

String基本数据类型变量有与其对应的String包装对象，当访问String基本数据类型属性或方法时创建临时包装对象，访问的都是对象中的属性或方法（注：访问对象属性时，首先访问自身属性，访问不到时，则会在原型链上寻找对应的属性和方法）

%VB~3(DEKSZ1@VET4Z4Y{`F

**字符串比较**

---"B" > "A"、"B" > "a"  
--- "B".localeCompare("A"); //考虑本地化的字符排序，返回0或非0

**字符串拼接**

---合并：加号运算符（+、+=）  
---合并：拼接字符串数组（通过数组方法push、join等）

**String函数的使用**

---类型转换、实例化字符串对象

**字符串构造器方法（静态方法）**

--- String.fromCharCode(97,98,99);  
 --- String.fromCharCode.apply(null,[97,98,99]);

**字符串原型方法（String对象继承的方法）之提取字符串**

--- String.prototype.charAt(pos);  
---String.prototype.charCodeAt(pos);  
--- String.prototype.slice(start,end?);  
---String.prototype.substr(start,length?)  
---String.prototype.substring(start,end?);  
---String.prototype.split(separator?,limit?);

**字符串原型方法（String对象继承的方法）之字符串变换**

---String.prototype.trim( );  
---String.prototype.concat(str1?,str2?);  
---String.prototype.toLowerCase( ); String.prototype.toLocaleLowerCase( );  
---String.prototype.toUpperCase( ); String.prototype.toLocaleUpperCase( );

**字符串原型方法（String对象继承的方法）之检索和比较**

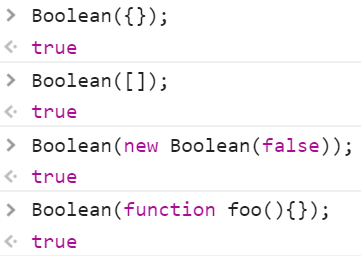
---String.prototype.indexOf(searchingString,position?);  
---String.prototype.lastIndexOf(searchingString,position?);  
---String.prototype.localeCompare(other);

**字符串原型方法（String对象继承的方法）之可正则方法**

---String.prototype.search(regexp);  
---String.prototype.match(regexp);  
---String.prototype.replace(regexp);

1. Boolean

**所有对象都是真值**



**Boolean函数的使用**

---类型转换、实例化布尔对象（原始值的包装对象）

JS语法、表达式及语句

JS语法、表达式及语句综述

**字面量**

- 对象字面量 var obj = {x:12，y:23};  
- 数组字面量 var arr = [1,2,true,'xyz'];

**标识符与保留字**

---标识符用来给变量或函数进行命名，以字母、下划线或$为开始  
---保留字：arguments、break、case、catch、class、const等（参见教材7.6）

**表达式（expression）与语句（statement）**

---表达式代码中基本的单位，它将产生一个值，用于需要值的地方，如：if(a>b){...}  
---语句表示了一种行为，如：var obj = {x:1,y:b}; //创建对象obj  
---JS期望语句的地方都可以写表达式（表达式语句），如：x+3；

**表达式及表达式分类**

---原始表达式、对象及数组初始化表达式、函数定义表达式、属性访问表达式

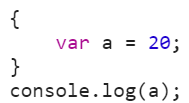
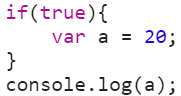
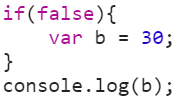
---函数调用表达式、对象创建表达式、算数表达式、关系表达式、逻辑表达式、赋

值表达式

**语句及语句分类**

---表达式语句、复合语句、条件语句(if-else、switch)、循环语句（for、for...in）

**ES5中的块（ES5中没有块作用域，所以带来了很多问题）**

JS严格模式

**ES5中的运行模式**

---严格模式和非严格模式（松散模式）

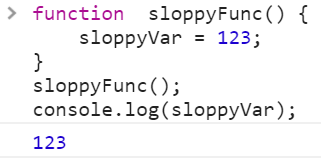
**严格模式的目的**

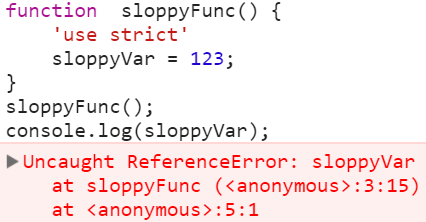
---消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为  
---消除代码运行的一些不安全之处，保证代码运行的安全  
---提高编译器效率，增加运行速度

**启用严格模式的方式**

---针对整个脚本文件使用 'use strict'  
---针对函数使用 'use strict'

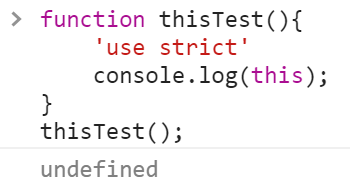
**严格模式下全局变量需显式声明**

****

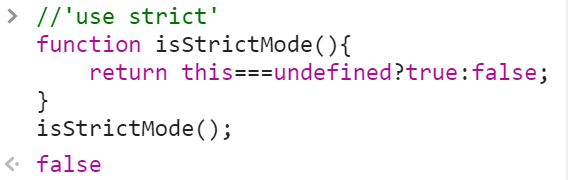
****

**JS严格模式下语法和行为的改变 二（函数中的this）**

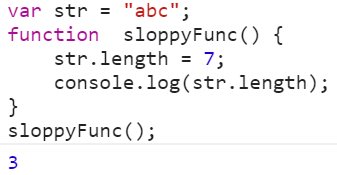
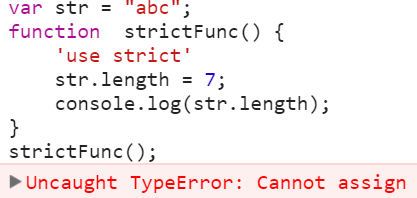
**一般函数中的this（严格模式）为undefined，非严格下为全局变量**

****

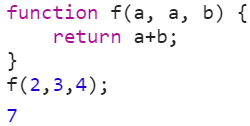
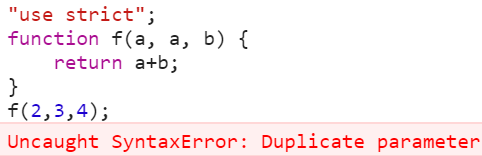
**可以用此特性来判断当前是否为严格模式**

****

**严格模式下禁止删除不可改变的属性和未定义的变量**

** **

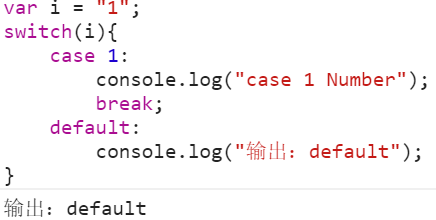
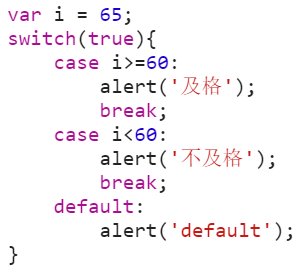
**严格模式下禁止函数参数重名**

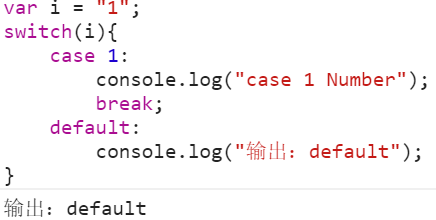
** **

switch详解、for...in

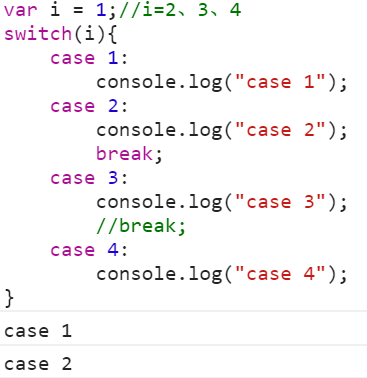
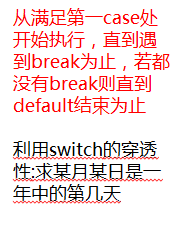
**switch语句中的case**

**--- case在比较时使用的是全等操作符比较,因此不会发生隐式类型转换  
--- case 后可以是一个表达式（如 i<60）**

** **

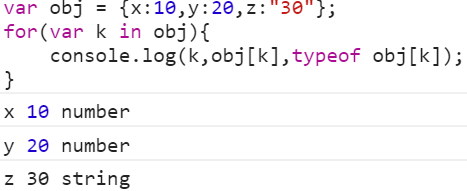
****

**switch语句中的穿透性及其应用**

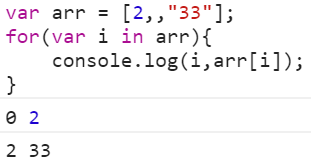
**** 

**for...in**

**for...in常用来遍历对象**

****

**for...in遍历数组（忽略空缺）**

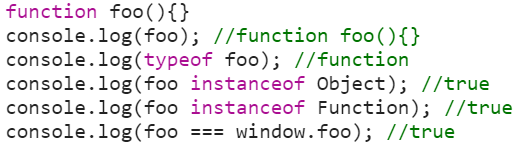
****

JS函数对象

函数对象

JS中的函数也是对象

--- JS中每个函数都是作为对象来维护和运行的，即函数对象（既有属性也有方法）  
---可以将函数（函数对象）赋值给一个变量，或将函数作为参数进行传递  
---函数对象对应的类型是Function（类似于数组对象对应于Array、日期对象对应于Date）  
---如果变量是函数（函数对象）时，typeof此对象，返回function，而非object   
---内置的函数对象（Array、Function、Date等），内置的非函数对象（Math、JSON）



函数对象的属性及方法

**函数对象的属性**

--- length、arguments（隐藏的局部变量）  
---caller、callee（arguments的属性，常用于递归调用）  
---prototype

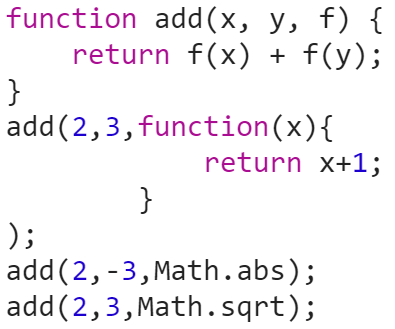
**函数对象的方法**

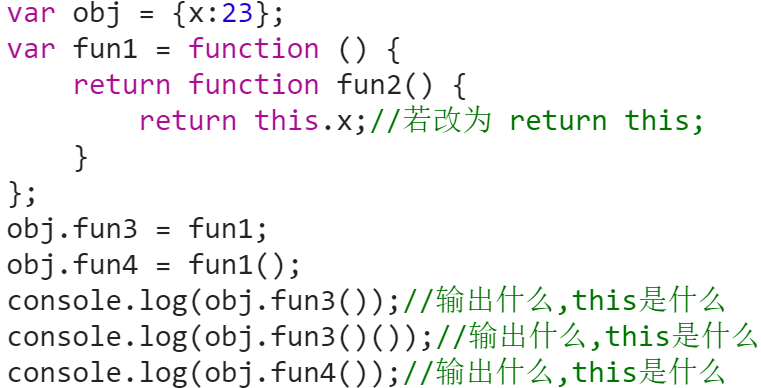
---call、apply  
---bind  
---toString、valueOf

高阶函数

**高阶函数是指至少满足下列条件之一的函数**

---函数作为参数被传递（最常见的形式：回调函数）  
---函数作为返回值输出（与闭包有紧密联系）



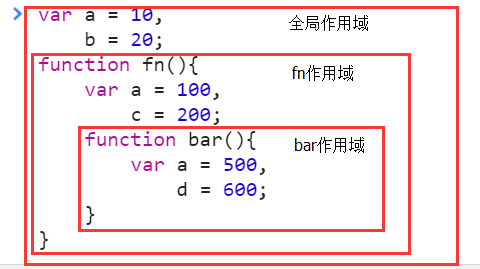


JS作用域及执行上下文

JS作用域及其特点

**什么是作用域**

- 作用域就是变量与函数的可访问范围（变量生效的区域范围，即在何处可以被访问到）  
- 作用域控制着变量与函数的可见性和生命周期，它也是根据名称查找变量的一套规则

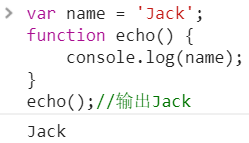
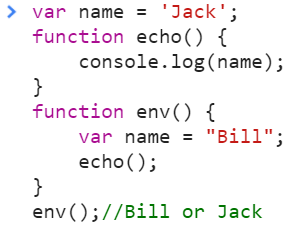
左侧实例（嵌套作用域）中：  
变量d只能在bar作用域中被访问到，  
变量c只能在fn和bar作用域中被访问到

在bar中访问a时为500（覆盖性）

在bar中访问c时为200（链式关系）

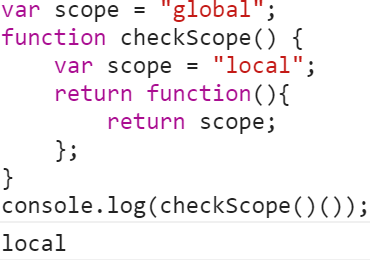
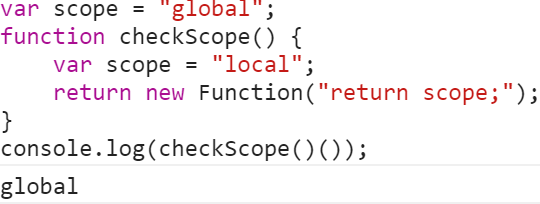
**JS作用域特点（词法作用域）**

---JS采用的是词法作用域（静态性），这种静态结构决定了一个变量的作用域  
---词法作用域不会被函数从哪里调用等因素影响，与调用形式无关（体现了静态性）

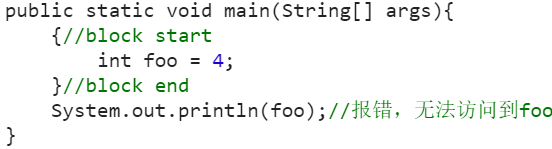
**JS作用域特点（静态词法作用域补充部分）**

---通过new Function创建的函数对象不一定遵从静态词法作用域  
---对比下边两个例子（通过不同形式定义的函数对象，访问到的scope的区别）

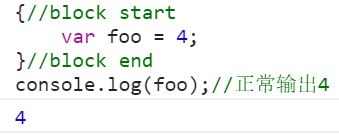
 

**大多数语言都有块级作用域**

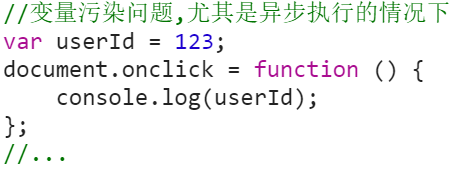
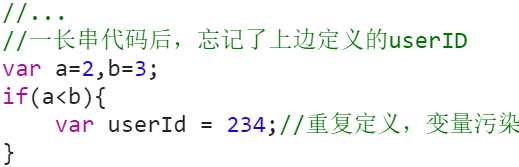
**---**变量“存活”在最近的代码块中，比如Java中



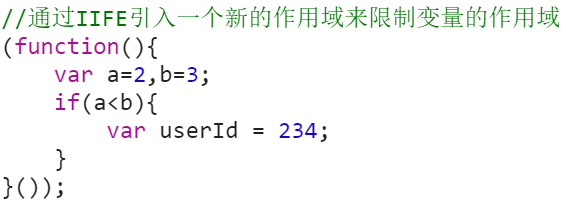
**JS（ES5）采用的是函数级作用域，没有块作用域**

****

**无块作用域的问题（变量污染、变量共享问题）**

** **

**解决方案IIFE（更多内容参见IIFE部分）**

****