对象

Js中所有变量都是对象，除了null和undefined

一个常见的误解是数字的字面量不是对象，这是因为js解析器的一个错误，它试图将点操作符解析为浮点数字面值的一部分。

2.toString();//出错：SyntaxError

2..toString();

2 .toString();

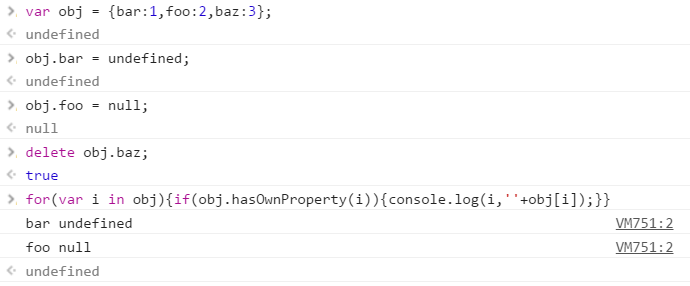
(2).toString();//2先被计算

访问属性



删除属性：

删除属性的唯一方法是使用delete操作符，设置属性为undefined或者null并不能真正的删除属性，而仅仅是移除了属性和值的关联

、

当使用for in循环遍历对象的属性时，原型链上的所有和属性都将被访问，因此如果一个对象的继承层次太深的话会影响性能。



hasOwnProperty是js中唯一一个处理属性但是不查找原型链的函数

如果hasOwnProperty被非法占用，比如是某个对象的属性，那么再调用来检测属性时，就要使用外部的hasOwnProperty函数：

//使用其他对象的hasOwnProrperty，并将其上下文设置为foo

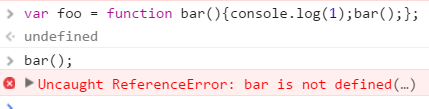
{}.hasOwnProperty.call(foo,’bar’); 有报错？



注意：for in 循环不会遍历那些enumerable设置为false的属性；比如数组的length属性

函数

函数是一等对象，可以把 像其他值一样传递。一个常见的用法是把匿名函数作为回调函数传递到异步函数中。



Bar函数声明外事不可见的，这是因为我们已经把函数赋值给foo，然而在Bar内部依然可见。这是由于js的命名处理所致，函数名在函数内总是可见的。

This指向：

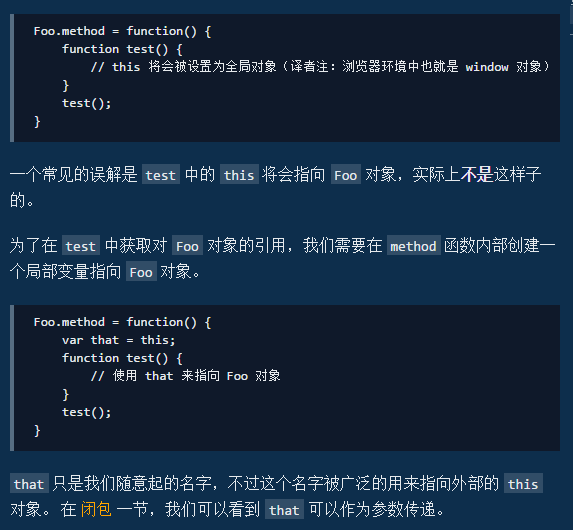
全局范围内，函数调用foo()；指向全局对象

方法调用test.foo();指向test对象

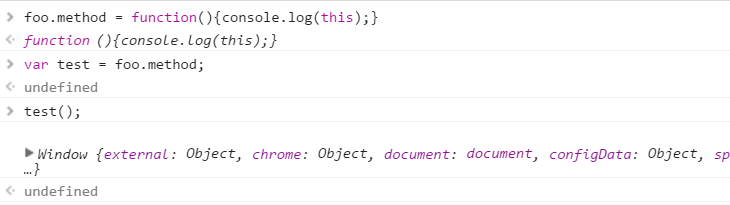
调用构造函数new foo();指向新创建的对象

Foo.call(bar,1,2,3);函数内的this指向bar

在对象字面声明的语法中，this不能用来指向对象本身，而是window对象

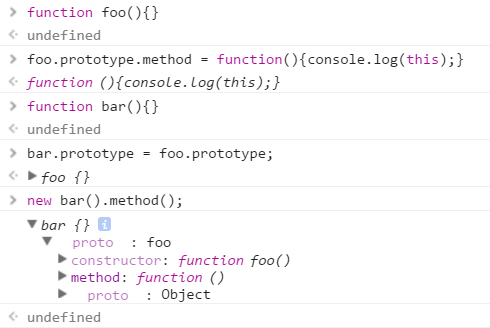


两个返回的都是method方法？？？



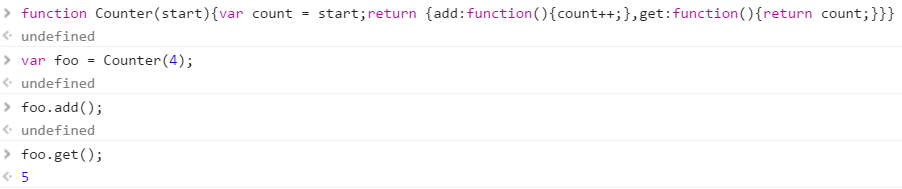
方法赋值给一个变量后，函数内的this将不再指向对象

虽然this 的晚绑定特性似乎不太友好，但这确实基于原型链继承赖以生存的土壤。

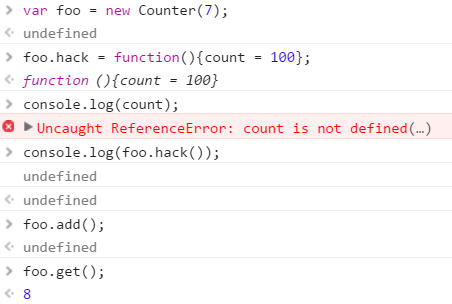


当method被调用时，this将会指向bar的实例对象

模拟私有变量：



Return的两个函数都维持着对外部作用域Counter的引用，因此总可以访问此作用域内定义的变量count。



上面的代码不会改变定义在Counter作用域中的count变量，因为foo.hack没有定义在哪个作用域内。

它将会创建或者覆盖全局变量count？？？？

循环中的闭包

1为了得到想要的结果，需要每次循环中创建变量i的拷贝

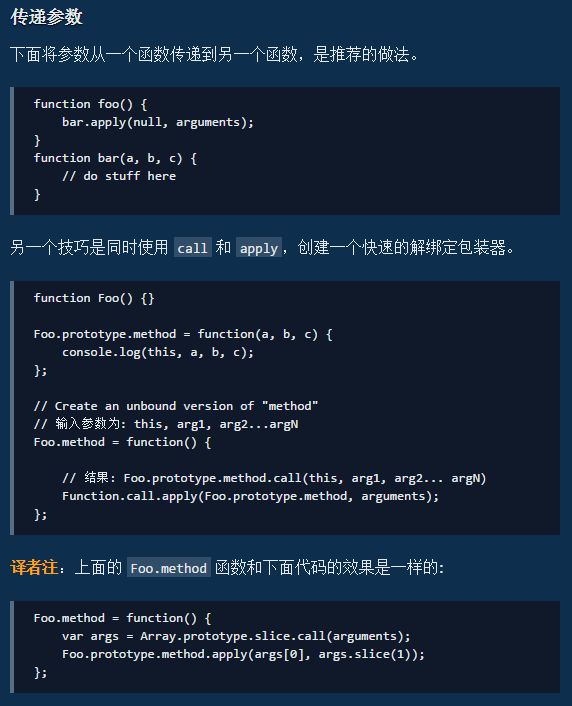
2为了正确的获得循环序号，最好使用自执行匿名函数，函数内变量就拥有了i的一个拷贝

3从自执行匿名函数中返回一个函数function(e){return function(){console.log(e);}}(i)

Arguments对象

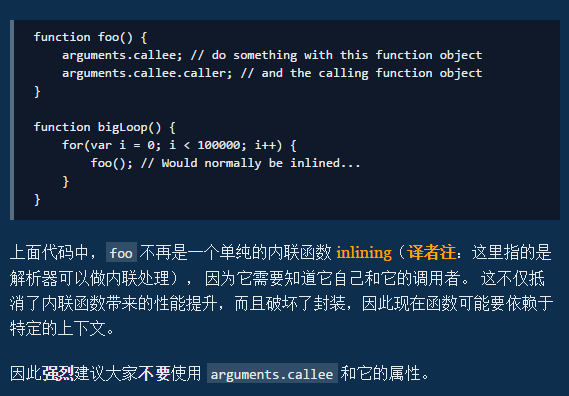
其实是一个对象

转换为数组：Array.prototype.slice.call(arguments);转化比较慢，在性能不好的代码中不推荐使用



Arguments对象总会被创建，除了两个特殊情况——作为局部变量声明和作为形式参数。而不管它是否被使用

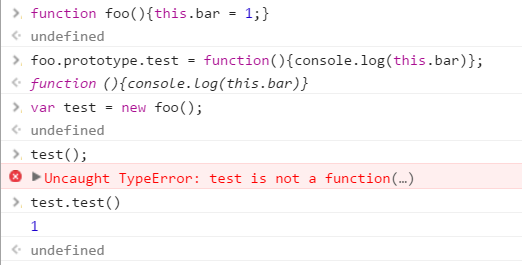
使用arguments.callee会显著的影响现代js引擎的性能



构造函数

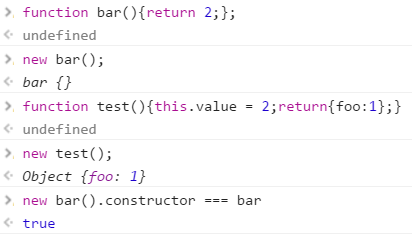
这个新创建的对象的prototype被指向到构造函数的prototype

如果被调用的构造函数没有显式的return表达式，则隐式的会返回this对象，也就是新创建的对象。

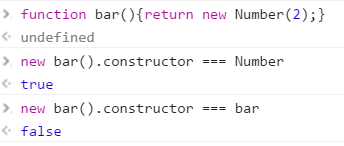


显式的return表达式将会影响返回结果，但仅限于返回的是一个对象

New test(); 是函数返回的对象，而不是新创建的对象

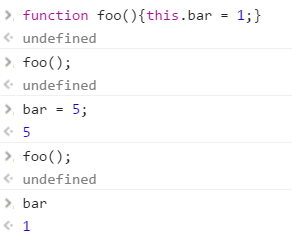


但如果返回的是数字对象，结果就不同了

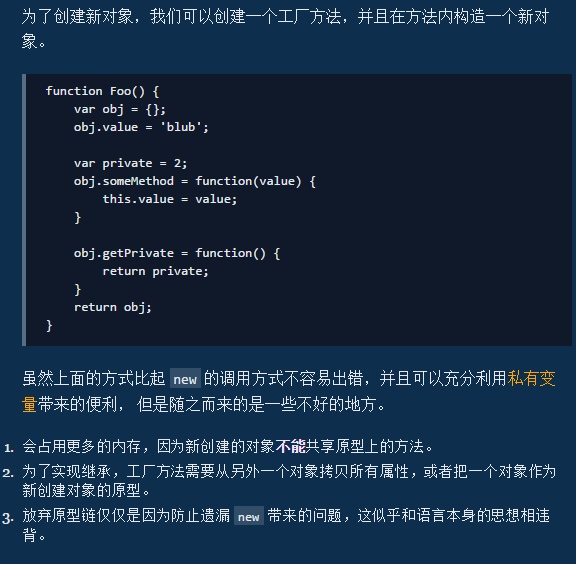


New 被遗漏，函数则不会返回新创建的对象

这里的this指向全局对象



不要使用new关键字来调用函数，因为如果忘记使用就会导致错误。



那到底是用还是不用？？？

最终使用哪种方式取决于应用程序的需求，选择一种代码书写风格并坚持下去才是最重要的。

作用域与命名空间

Js中没有显式的命名空间定义，这就意味着所有对象都定义在一个全局共享的命名空间下面。

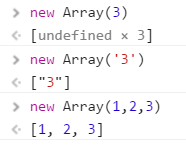
Js中局部变量只能通过两种方式声明，一个是作为函数参数，另一个是通过var关键字声明

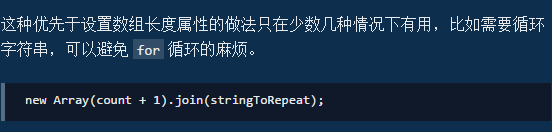
推荐使用匿名包装器来创建命名空间。这样不仅可以防止命名冲突，而且有利于程序的模块化。

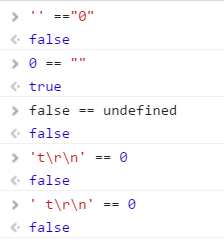
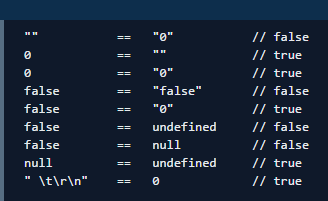
另外，使用全局变量被认为是不好的习惯。这样的代码倾向于产生错误和带来高的维护成本。

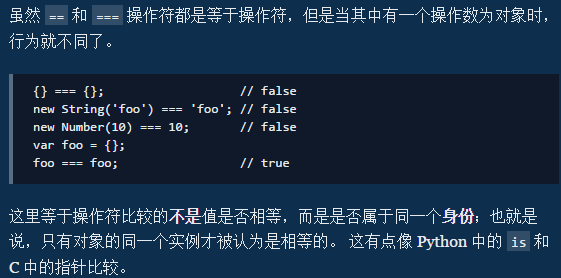
数组

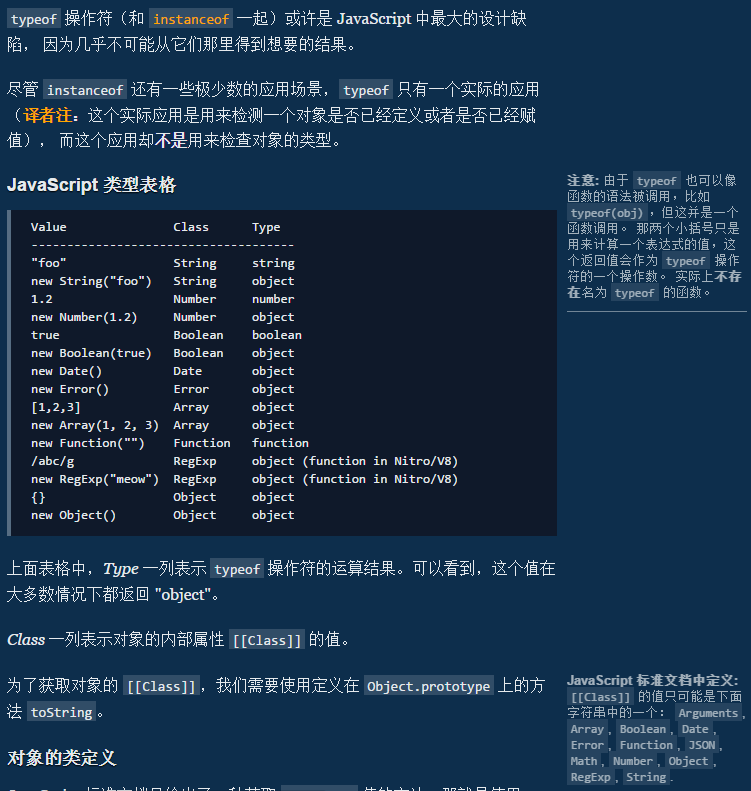
为了更好的性能，推荐使用普通的for循环并缓存数组length属性。使用for in遍历数组被认为是不好的代码习惯并倾向于产生错误和导致性能问题。

Array的构造函数在如何处理参数时有点模棱两可。

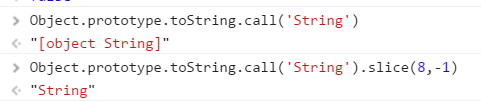
？？？？？

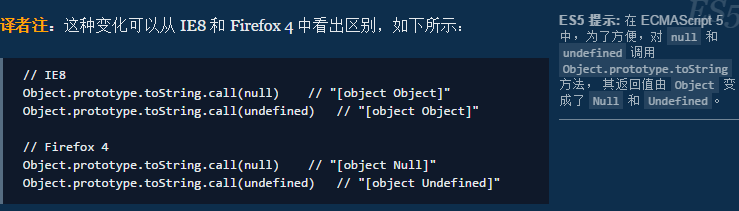


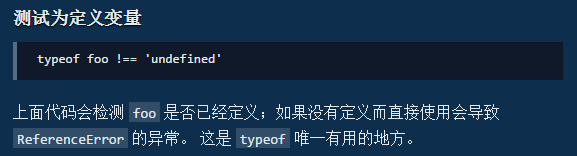




获取[[class]]值的方法







为了检测一个对象的类型，强烈推荐使用Object.prototype.toString方法；因为这是唯一一个可依赖的方式。Typeof的一些返回值在标准文档中并未定义，因此不同的引擎实现可能不同。

Instanceof操作符应该仅仅用来比较来自同一个js上下文的自定义对象。正如typeof操作符一样，任何其他的用法都应该是避免的。

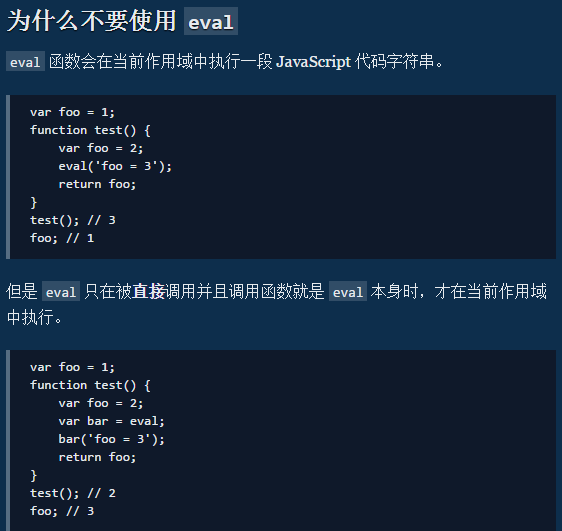
10 == 010 //false 010被认为是八进制

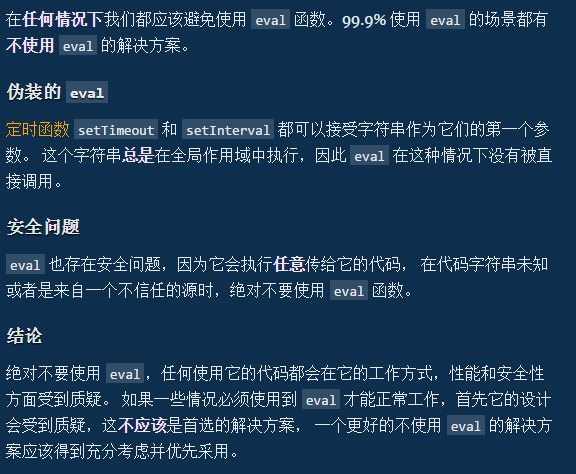
New Number(10) === 10 // false 对象和数字的比较

Number(10) === 10 //true 数字和数字的比较

New Number(10) + 0 === 10 // true 由于隐式的类型转换

核心





建议绝对不要省略分号，同时也提倡将花括号和相应的表达式放在一行， 对于只有一行代码的 if 或者 else 表达式，也不应该省略花括号。 这些良好的编程习惯不仅可以提到代码的一致性，而且可以防止解析器改变代码行为的错误处理。

其他

