### JS中‘hello’与new String(‘hello’)引出的问题详解

在工作中我们大概有3种方法去定义一个字符串：

|  |
| --- |
| 1. var str = 'hello';  2. var str1 = String('hello');  3. var str2 = new String('hello'); |

这三种方法定义出来的 'hello',都有自己的属性 例如lengh,有自己的方法例如: indexOf(),在日常工作定义中也没有感觉到任何的不同。

那是否深入过，

1.这三种方式定义出来的'hello',是否是一样的呢?

2.为什么基本类型可以直接调用其对应的方法呢?

console.log(str === str1) //true

console.log(str === str2) //false

console.log(str1 === str2) //false

我们可以发现 最后一种方式定义的 与上面两种方式定义的 不相等。

首先我们知道一个东西 就是:

new 出来的一定是对象。

所以分别 打出三个的类型：

console.log(typeof str) //string

console.log(typeof str1) // string

console.log(typeof str2) //object

引出数据类型 与 堆栈之间的关系

我们知道，String,Number,Boolean在JS中是基本类型，基本类型是存储在栈(stack)内存中的,数据大小确定，内存空间大小可以分配。

引用类型是存储在堆(heap)内存中的,例如对象, 栈中存在的仅仅是一个堆的指针，这也就是我们日常遇到 a = {num:1}, b=a, b.num1 = 2, 那么a.num1 也为2 的原因。因为a,b同时指向同一个地址。

前两种方式定义出来的是在栈中并且值相等，而第三种方法定义出来的仅仅是栈中的一个指针。

为什么基本类型可以直接调用其对应的方法呢?

|  |
| --- |
| console.log(str.length) // 5  str.say = 'world'  console.log(str.say) //undefined  console.log(str1.lengh) // 5  str1.say = 'world'  console.log(str1.say) //undefined  console.log(str2.lengh) // 5  str2.say = 'world'  console.log(str.say) //world |

我们发现第一种和第二种都可以访问length属性，但为什么并不能自定义一个属性进行访问呢？

数字、字符串、布尔三者，在JS中称为原始资料类型，而new String() new Number()就是包装对象。

包装对象也是对象。

所以我们打印 三种类型分别为 ： string(原始资料类型) , string(原始资料类型) , object(包装对象).

可以理解 new出来的str2对象有String的一系列方法：

console.log(str2.indexOf === String.prototype.indexOf) // true

但是：

console.log(str.indexOf === String.prototype.indexOf) //true

console.log(str1.indexOf === String.prototype.indexOf) //true

且

str instanceof String // false

str1 instanceof String // false

str str1 不属于String却拥有String的方法，为什么呢？

因为这是JS中的设计！！

原始资料类型的 方法和属性都是“借”来的。

原始资料类型并没有像对象那样的属性和方法，都是向包装对象借来的。所以原始资料类型是可以向new String() 或new Number() 借来所有的方法，但自己本身并没有属性和方法。

这就是为什么第一种第二种我们无法去自定义属性却可以使用对应类型的方法的原因。

总结：

1.第一种和第二种方法定义出来的是原始资料类型，并储存于栈中,并向包装对象(new ..())借来方法和属性.

2.第三种是 包装对象，栈中储存堆指针，堆中储存内容。