举例一：

var s='this is a string';

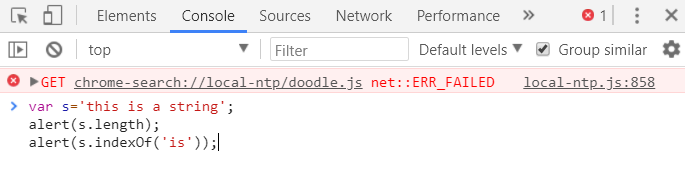
alert(s.length);

alert(s.indexOf('is'));

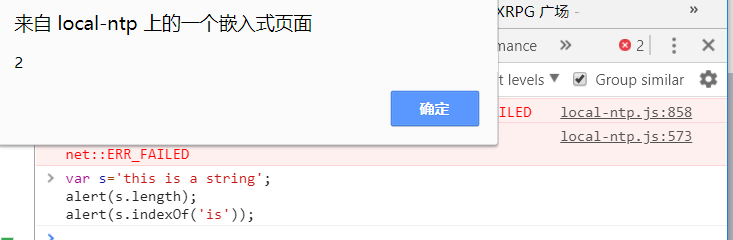
JavaScript是面向对象的语言，使用”.”操作符可以访问对象的属性和方法，

而对于基本类型（null, undefined, bool, number, string）应该是值类型，没有属性和方法，

然而上面的代码虽然结果简单，但是string明明是值类型却又有属性又有方法。







其实只要是引用了字符串的属性和方法，JavaScript就会将字符串值通过new String(s)的方式转为

内置对象String，一旦引用结束，这个对象就会销毁。所以上面代码在使用的实际上是String对象

的length属性和indexOf方法。

同样的道理，数字和布尔值的处理也类似。，null和undefined没有对应对象。既然有对象生成，那么：

var s='this is a string';

s.len=10;

alert(s.len);





结果并没有返回10，而是undefined！不是说好了是个对象吗！正如刚才提到第二行代码只是创建了

一个临时的String对象，随即销毁，第三行代码又会创建一个新的临时对象

（这就是低版本IE频繁处理字符串效率低的一个原因），自然没有len属性。

**类型转换练习代码**

var sColor = "red";

alert(sColor.length); //输出 "3"

var bFound = false;

alert(bFound.toString()); //输出 "false"

var iNum1 = 10;

var iNum2 = 10.0;

alert(iNum1.toString()); //输出 "10"

alert(iNum2.toString()) //输出 "10"

var iNum = 10;

alert(iNum.toString(2)); //输出 "1010"

alert(iNum.toString(8)); //输出 "12"

alert(iNum.toString(16)); //输出 "A"

### **parseInt()**

在判断字符串是否是数字值前，parseInt() 和 parseFloat() 都会仔细分析该字符串。

parseInt() 方法首先查看位置 0 处的字符，判断它是否是个有效数字；如果不是，该方法将返回 NaN，不再继续执行其他操作。但如果该字符是有效数字，该方法将查看位置 1 处的字符，进行同样的测试。这一过程将持续到发现非有效数字的字符为止，此时 parseInt() 将把该字符之前的字符串转换成数字。

var iNum1 = parseInt("12345red"); //返回 12345

var iNum1 = parseInt("0xA"); //返回 10

var iNum1 = parseInt("56.9"); //返回 56

var iNum1 = parseInt("red"); //返回 NaN

### **parseFloat()**

parseFloat() 方法与 parseInt() 方法的处理方式相似，从位置 0 开始查看每个字符，直到找到第一个非有效的字符为止，然后把该字符之前的字符串转换成整数。

var fNum1 = parseFloat("12345red"); //返回 12345

var fNum2 = parseFloat("0xA"); //返回 NaN

var fNum3 = parseFloat("11.2"); //返回 11.2

var fNum4 = parseFloat("11.22.33"); //返回 11.22

var fNum5 = parseFloat("0102"); //返回 102

var fNum1 = parseFloat("red"); //返回 NaN