渔人码头

非淡泊无以明志，非宁静无以致远

JS中 toString() & valueOf()

数据的转换

所有对象继承了两个转换方法：

第一个是toString(),它的作用是返回一个反映这个对象的字符串

第二个是valueOf(),它的作用是返回它相应的原始值

toString()

toString()可以看做是把一个数据转换成了相应字符串的形式，安照这个转换规则中

表3-2：JavaScript类型转换

值	转换为：			
	字符串	数字	布尔值	对象
undefined	"undefined"	NaN	false	throws TypeError
null	"null"	0	false	throws TypeError
true	"true"	1		new Boolean(true)
false	"false"	0		new Boolean(false)
""(空字符串)		0	false	new String("")
"1.2"(非空,数字)		1.2	true	new String("1.2")
"one"(非空,非数字)		NaN	true	new String("one")
0	"0"		false	new Number(0)
-0	"0"		false	new Number(-0)
NaN	"NaN"		false	new Number(NaN)
Infinity	"Infinity"		true	new Number(Infinity)
-Infinity	"-Infinity"		true	new Number(-Infinity)
1(无穷大,非零)	"1"		true	new Number(1)
{}(任意对象)	参考3.8.3节	参考3.8.3节	true	
[](任意数组)	""	0	true	
[9](1个数字元素)	"9"	9	true	
['a'](其他数组)	使用join()方法	NaN	true	
function(){}(任意函数)	参考3.8.3节	NaN	true	

使用样例：

```
//返回相应的字符串
console.log(
  ({x:1,
    y:1
   }).toString()

  );  // [object Object]
console.log([1,2,3].toString()); // 1,2,3
console.log((function(x){f(x); }).toString()); //function (x){f(x); }
console.log(/\d+/g.toString()); // /\d+/g
console.log(new Date(2015,4,4).toString()); // Mon May 04 2015 00:00:00 GMT+0800

console.log(new Date(2015,4,4).valueOf()); //  1430668800000
```

valueOf()

每个JavaScript固有对象的 valueOf 方法定义不同。

对象	返回值
Array	数组的元素被转换为字符串，这些字符串由逗号分隔，连接在一起。其操作与 Array.toString 和 Array.join 方法相同。
Boolean	Boolean 值。
Date	存储的时间是从 1970 年 1 月 1 日午夜开始计的毫秒数 UTC。
Function	函数本身。
Number	数字值。
Object	对象本身。这是默认情况。
String	字符串值。

Math 和 Error 对象没有 valueOf 方法。

一般来说，对象到字符串的转换经过了如下步骤：

1.如果对象具有toString()方法，则调用这个方法。如果它返回一个原始值，js将这个值转换成字符串，并返还这个字符串结果。

2.如果对象没有toString()方法，或者这个方法并不返回一个原始值，那么js将调用valueOf()方法。

3.否则，js无法从toString()或者valueOf()获得一个原始值，因此这时它将抛出一个类型错误异常。

一般来说，对象到数字的转换过程中，js做了同样类似的事情，但这里它会首先尝试使用valueOf()方法：

1.如果对象具有valueOf()方法，后者返回一个原始值，则js将这个原始值转换成数字，并返回这个数字。

2.否则，如果对象具有toString()方法，后者返回一个原始值，则js将转换并返回。

（首先js转换成相应的字符串原始值，再继续将这个原始值转换成相应的数字类型，再返回数字）

3.否则，js抛出一个类型错误异常。

对象通过toString或valueOf方法转换为原始值，JS语言核心的内置类首先尝试使用valueOf()，再尝试使用toString()

一个小李子

“1” == true;

将返回true，转换形式是：true首先转换为1，然后再执行比较。接下来字符串“1”也转换成了数字1，相等，所以返回true

另外如：

```
var str = new String('hello,world');
console.log(typeof str); //'object'
console.log(typeof str.valueOf()); //'string'
```

对于所有非日期对象来说，对象到原始值的转换基本上是对象到数字的转换

（首先调用valueOf,但日期对象则使用对象到字符串的转换模式，但这种转换只执行一次就立即被使用，不会像上面所说的那般 先转成字符串再转成相应的数字类型）

比如说，js中“+"运算符可以进行数学加法和字符串连接操作。

如果他它的其中一个操作数是对象，则js将使用特殊的方法将对象转换成原始值，而不是使用其他算术运算符的方法执行对象到数字的转换，”==“运算符类似

和”==“一样，”<"与其他运算符也会做对象到原始值的转换，但要出去日期对象的特殊情形

“-“减号运算符把两个操作数都转换成数字

比如：

```
var now = new Date();
console.log(now);  // Date {Sat Apr 04 2015 13:21:08 GMT+0800}
console.log(typeof (now+1));  //string
console.log((now+1));  //Sat Apr 04 2015 13:21:08 GMT+08001
console.log(typeof (now-1));  //number
console.log((now-1));  // 1428124868474
console.log(now == now.toString());  //true
console.log(now > now -1); //true
```

```
var date = new Date();
var date_string = date.toString();
var date_value = date.valueOf();
alert(date == date_string); //true
alert(date == date_value); //false
```

更详细了使用例子：

（摘自：http://www.jb51.net/article/32327.htm）

```
var aaa = {
i: 10,
valueOf: function() { return this.i+30; },
toString: function() { return this.valueOf()+10; }
}
alert(aaa > 20); // true
alert(+aaa); // 40
alert(aaa); // 50
```

之所以有这样的结果，因为它们偷偷地调用valueOf或toString方法。更进一步测试

```
var bbb = {
i: 10,
toString: function() {
console.log('toString');
return this.i;
},
valueOf: function() {
console.log('valueOf');
return this.i;
}
}
alert(bbb);// 10 toString
alert(+bbb); // 10 valueOf
alert(''+bbb); // 10 valueOf
alert(String(bbb)); // 10 toString
alert(Number(bbb)); // 10 valueOf
alert(bbb == '10'); // true valueOf
alert(bbb === '10'); // false
```

乍一看结果，大抵给人的感觉是，如果转换为字符串时调用toString方法，如果是转换为数值时则调用valueOf方法，但其中有两个很不和谐。

一个是alert(''+bbb)，字符串合拼应该是调用toString方法……另一个我们暂时可以理解为===操作符不进行隐式转换，因此不调用它们。

为了追究真相，我们需要更严谨的实验。

```
var aa = {
i: 10,
toString: function() {
console.log('toString');
return this.i;
}
}
alert(aa);// 10 toString
alert(+aa); // 10 toString
alert(''+aa); // 10 toString
alert(String(aa)); // 10 toString
alert(Number(aa)); // 10 toString
alert(aa == '10'); // true toString
再看valueOf。
var bb = {
i: 10,
valueOf: function() {
```

```
console.log('valueOf');
return this.i;
}
}
alert(bb);// [object Object]
alert(+bb); // 10 valueOf
alert(''+bb); // 10 valueOf
alert(String(bb)); // [object Object]
alert(Number(bb)); // 10 valueOf
alert(bb == '10'); // true valueOf
```

发现有点不同吧？！ 它没有像上面toString那样统一规整。对于那个[object Object]，我估计是从Objec

```
Object.prototype.toString = null;
var cc = {
i: 10,
valueOf: function() {
console.log('valueOf');
return this.i;
}
}
alert(cc);// 10 valueOf
alert(+cc); // 10 valueOf
alert(''+cc); // 10 valueOf
alert(String(cc)); // 10 valueOf
alert(Number(cc)); // 10 valueOf
alert(cc == '10'); // true valueOf
```

总结起来就是 如果只重写了toString，对象转换时会无视valueOf的存在来进行转换。

但是，如果只重写了valueOf方法，在要转换为字符串的时候会优先考虑valueOf方法。在不能调用toString的情况下，只能让valueOf上阵了

[-_-]眼睛累了吧，注意劳逸结合呀[-_-]

posted @ 2015-04-04 13:25 -渔人码头- 阅读(18395) 评论(4) 编辑 收藏 举报