## 当心隐式的强制转换

### 一、加减乘除和NaN

Javascript对类型错误初期宽容。许多语言都认为表达式而抛出错误。但是js却能继续执行，甚至能得到一个看上去和去合理的结果。

*console*.log(3 + true); // 4



在Js中有一些极少数的情况，提供错误类型会产生一个即时错误，例如，调用一个非函数对象，或者是调用null的属性或者方法时。

*console*.log("hello"(0)); // hello is not a function

*console*.log(null.length); // cann't read property 'length' of null

大多数情况下，Js不会跑出一个错误，（我使用的谷歌浏览器就抛出错误了。而且在我的印象中似乎都报错了。）而是按照多种多样的自动转换协议将值强制转换成期望的类型。例如-、\*、/、%在计算的时候会用把参数转换为数字。而+则比较特殊，他会重载自己的方法，有时实现数字运算，而有时则会进行字符串的拼接。

我认为只要按照左结合律进行一步一步的运算，没进行一步就得出一个结果和下一步进行运算的想法，就可以很容易的判断最后的结果是什么。

比如：

*console*.log((3 + 2) + "1"); // 51 是string类型

首先执行括号中的内容，3+2得5，然后后面的参数是字符串”1”，那么就会变成拼接字符串的预算，得到字符串结果”51”。

判断+得到的结果类型很简单，就看+运算中有没有出现字符串，如果出现了，那么+得到的结果就是字符串。

但是强制类型转换也会隐藏错误，null则会被转换为0,而没有被定义的变量则会被转换为NaN。而NaN这个东西又比较特殊，它自身代表着非数值，而它确实是一个浮点数类型的值。

*console*.log(typeof NaN); // number



而且判断是不是NaN还比较的麻烦，因为他还有一个特殊的地方，就是他是js世界中唯一一个不等于其自身的值（注意，这与new两个同类型的对象不同，两个同类型的对象本身就是两个不同的对象，因此不相等也是正常的）。

总体上判断一个值是不是NaN有以下几个方法;

1 、知道了这个值是数值型（可能是NaN），但是想知道他是不是NaN。

可以使用标准库的函数 isNaN();

Tip：因为这个方法可以精准的从一大堆数值型挑出来NaN。

如果是NaN，必然返回的结果是true，如果是数字就返回false

但是，这种方法也仅限于清楚需要判断的东西一定是数值型的情况。

isNaN在判断是否为非数字的时候会经历一个对参数强制转换类型的过程。如果带入的参数是一个可以转换为数值类型的东西，他会自动的把他Number转换一下，然后判断转换后的结果是不是数值型的。之后判断的原理同上。因此我们必须保证判断的东西一定是数值型的才行。

var a = "1";

var b = "NaN"

*console*.log(isNaN(a)); // false,字符串"1"本不是数字，但是却当成数字处理了

2 、不知道判断的东西是什么，但是要判断出结果到底是不是NaN。

在这可以利用NAN的特点，就是他自己都不等于他自己，而且是唯一一个不等于自身的。

function isReallyNaN(*x*){

// 用于判断参数x到底是不是NaN

return x !== x; // 是的话返回true，否则返回false;

}

### “+”对对象之间的操作

对于对象而言，大多数都具备toString()和valueOf();

在对象只有toString()方法的时候，+默认调用toString()方法。只有valueOf()的时候则调用valueOf();

var obj = {

a:"0",

b:"c",

toString:function(){

return 1;

},

// valueOf:function(){

// return "你好";

// }

};

*console*.log(1 + obj); //"2"

var obj = {

a:"0",

b:"c",

// toString:function(){

// return 1;

// },

valueOf:function(){

return "你好";

}

};

*console*.log(1 + obj); //"1你好"

但是两者都有的时候会调用那个呢？ js会盲目的调用valueOf();

var obj = {

a:"0",

b:"c",

toString:function(){

return 1;

},

valueOf:function(){

return "你好";

}

};

*console*.log(1 + obj); //"1你好"

### 对于真值运算

最后一种强制转换有时成为真值运算（truthiness）。If 、||、&&在逻辑上需要布尔值作为操作数，但是实际上存在强制转换，却能接受任何值。大部分的值都是真值，对于字符串和数字以外的值，js不会进行任何隐式的类型转换（我认为这里特指Boolea()方法,在if中所有值都要经过Boolean转换一下，只不过boolean()在判断字符串和数字需要特殊看待，其他的值就不看了，直接就会判定为true），直接判定为真值。只有特殊的数字和字符串才会被转换为假值。

在js中只有7个假值。分别是

false、0、-0、""、undefined、null、NaN

其他的值都是真值！

因此常犯的错误也就出现了，我们不能通过判断未知变量是否存在的时候用类似这样的方法：

var a = 0; // a明显存在

if(a){

alert("存在");

}

else{

alert("不存在"); // 却说不存在

}

正确的操作为：

var a = "";

if(typeof a == 'undefined'){

// do something ...

} else {

// do something ...

}

### 我自己的SB操作

有的时候大脑潜意识的出错，而且感觉很自然，还有的时候看着半天感觉是对的。那就是我脑残的时候。

比如，我想判断输入框中是否为空值。我曾经这样做过：

<input *type*="number" *name*="" *id*="number" />

var txt = document.getElementById("number");

if(txt){ // 经常认为，txt如果是空的，就警告是空的

// 但是事实上空的是false，就直接提示有东西了

alert("空的");

} else { // 如果有东西，应该提示有东西

// 如果有东西，则直接走if不走else，说有是空的了。。。

alert("有东西");

}

调转一下if和else的逻辑就对了。。。。希望以后不要犯这种SB错误，愿神保佑。。。。。。