JavaScript的数据类型是非常弱的（不然不会叫它做弱类型语言了）！在使用算术运算符时，运算符两边的数据类型可以是任意的，比如，一个字符串可以和数字相加。之所以不同的数据类型之间可以做运算，是因为JavaScript引擎在运算之前会悄悄的把他们进行了隐式类型转换的，如下是数值类型和布尔类型的相加：

复制代码 代码如下:

3 + true; // 4

结果是一个数值型！如果是在C或者Java环境的话，上面的运算肯定会因为运算符两边的数据类型不一致而导致报错的！但是，在JavaScript中，只有少数情况下，错误类型才会导致出错，比如调用非函数，或者读取null或者undefined的属性时，如下：

复制代码 代码如下:

"hello"(1); // error: not a function  
null.x; // error: cannot read property 'x' of null

多数情况下，JavaScript都不会出错的，而是自动的进行相应的类型转换。比如-, \*, /,和%等算术运算符都会把操作数转换成数字的，但是“+”号就有点不一样了，有些情况下，它是算术加号，有些情况下，是字符串连接符号，具体的要看它的操作数，如下：

复制代码 代码如下:

2 + 3; // 5  
"hello" + " world"; // "hello world"

但是，如果字符串和数字相加，会是怎样的结果呢？JavaScript会自动把数字转换成字符的，不管数字在前还是字符串在前，如下：

复制代码 代码如下:

"2" + 3; // "23"  
2 + "3"; // "23"

字符串和数字相加结果是字符串，字符串和数字相加结果是字符串，字符串和数字相加结果是字符串，重要的事情说三遍！！！！！！

此外，需要注意的是，“+”的运算方向是从左到右的，如下：

复制代码 代码如下:

1 + 2 + "3"; // "33"

这与下面是等价的：

复制代码 代码如下:

(1 + 2) + "3"; // "33"

相比之下，下面的结果是不一样的：

复制代码 代码如下:

1 + "2" + 3; // "123"

但是，隐式类型转换，有时候，会隐藏一些错误的，比如，null会转换成0，undefined会转换成NaN。需要注意的是，NaN和NaN是不相等的（这是由于浮点数的精度决定的），如下：

复制代码 代码如下:

var x = NaN;  
x === NaN; // false

虽然，JavaScript提供了isNaN来检测某个值是否为NaN，但是，这也不太精确的，因为，在调用isNaN函数之前，本身就存在了一个隐式转换的过程，它会把那些原本不是NaN的值转换成NaN的，如下：

复制代码 代码如下:

isNaN("foo"); // true  
isNaN(undefined); // true  
isNaN({}); // true  
isNaN({ valueOf: "foo" }); // true

上面代码，我们使用isNaN来测试后，发现字符串，undefined，甚至对象，结果都返回真！！！但是，我们总不能说他们也是NaN吧？总而言之，得出的结论是：isNaN检测NaN并不可靠！！！

幸运的是，有一种可靠的并且准确的方法可以检测NaN。我们都知道，只有NaN是自己不等自己的，那么，我们就以使用不等于号（！==）来判断一个数是否等于自身，从而，可以检测到NaN了，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **var** a = NaN;  a !== a; // true  **var** b = "foo";  b !== b; // false  **var** c = undefined;  c !== c; // false  **var** d = {};  d !== d; // false  **var** e = { valueOf: "foo" };  e !== e; // false |

我们也可以把这种模式定义成一个函数，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2  3 | **function** isReallyNaN(x) {  **return** x !== x;  } |

OK,NaN的检测方法就是这么简单，我们下面继续讨论对象的隐式转换！

对象是可以转换成原始值的，最常见的方法就是把它转换成字符串，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2 | "the Math object: " + Math; // "the Math object: [object Math]"  "the JSON object: " + JSON; // "the JSON object: [object JSON]" |

对象转换成字符串是调用了他的toSting函数的，你可以手动的调用它来检测一下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2 | Math.toString(); // "[object Math]"  JSON.toString(); // "[object JSON]" |

类似的，对象也是可以转换成数字的，他是通过valueOf函数的，当然，你也是可以自定义这个valueOf函数的，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2 | "J" + { toString: **function**() { **return** "S"; } }; // "JS"  2 \* { valueOf: **function**() { **return** 3; } }; // 6 |

如果，一个对象同时存在valueOf方法和toString方法，那么，valueOf方法总是会被优先调用的，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **var** obj = {  toString: **function**() {  **return** "[object MyObject]";  },  valueOf: **function**() {  **return** 17;  }  };  "object: " + obj; // "object: 17" |

但是，多数情况下，这都不是我们想要的，一般的，尽可能使valueOf和toString表示的值相同（尽管类型可以不同）。

最后一种强制类型转换，我们常常称之为“真值运算”，比如，if, ||, &&，他们的操作数不一定是布尔型的额。JavaScript会通过简单的转换规则，将一些非布尔类型的值转换成布尔型的。大多数的值都会转换成true，只有少数的是false，他们分别是：false, 0, -0, ”", NaN, null, undefined，因为存在数字和字符串以及对象的值为false，所以，直接用真值转换来判断一个函数的参数是否传进来了，这是不不太安全的。比如，有一个可以具有默认值得可选参数的函数，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **function** point(x, y) {  **if** (!x) {  x = 320;  }  **if** (!y) {  y = 240;  }  **return** { x: x, y: y };  } |

这个函数会忽略任何的真值为假的参数的，包括0，-0；

复制代码 代码如下:

point(0, 0); // { x: 320, y: 240 }

检测undefined的更加准确的方法是用typeof操作：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **function** point(x, y) {  **if** (**typeof** x === "undefined") {  x = 320;  }  **if** (**typeof** y === "undefined") {  y = 240;  }  **return** { x: x, y: y };  } |

这种写法，可以区分开0和undefined的：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1  2 | point(); // { x: 320, y: 240 }  point(0, 0); // { x: 0, y: 0 } |

另外一种方法是利用参数跟undefined作比较，如下：

[?](http://www.52codes.net/article/2633.html)

|  |  |
| --- | --- |
| 1 | **if** (x === undefined) { ... } |

**总结:**

1. 类型错误有可能会被类型转换所隐藏。

2. “+”既可以表示字符串连接，又可以表示算术加，这取决于它的操作数，如果有一个为字符串的，那么，就是字符串连接了。

3. 对象通过valueOf方法，把自己转换成数字，通过toString方法，把自己转换成字符串。

4.具有valueOf方法的对象，应该定义一个相应的toString方法，用来返回相等的数字的字符串形式。

5.检测一些未定义的变量时，应该使用typeOf或者与undefined作比较，而不应该直接用真值运算。