JS中相等(==)运算符详解

原创 随风、逆风 最后发布于2018-05-17 16:09:41 阅读数 232 ☆ 收藏 展开

有时候我们会对某些语法的行为非常困惑，比如相等("==")运算符。由于JS是一种弱类型语言，在没有使用严格相等("===")运算符的情况下，相等运算符两侧的变量内部会发生隐式的类型转换，正是这个隐式类型转换导致相等运算符经常让人摸不着头脑。

刚好最近看到阮一峰老师ES6教程的规范文档这一部分，然后参考知乎一些大神的讲解，形成此文，同时也是让自己梳理一下，可能有不对的地方，还望指正。

一、规范文档中相等("==")运算符的解释

遇到实在是理解不了的问题，看规范文档最直接，《ECMAScript 2015规范文档》中讲解相等运算符的在7.2.12章节，这里直接贴出其算法的原文：

Abstract Equality Comparison

The comparison x == y, where x and y are values, produces true or false. Such a comparison is performed as follows:

1. ReturnIfAbrupt(*x*).
2. ReturnIfAbrupt(*y*).
3. If Type(*x*) is the same as Type(*y*), then
    1. Return the result of performing Strict Equality Comparison *x* === *y*.
4. If *x* is null and *y* is undefined, return true.
5. If *x* is undefined and *y* is null, return true.
6. If Type(*x*) is Number and Type(*y*) is String,
   return the result of the comparison *x* == ToNumber(*y*).
7. If Type(*x*) is String and Type(*y*) is Number,
   return the result of the comparison ToNumber(*x*) == *y*.
8. If Type(*x*) is Boolean, return the result of the comparison ToNumber(*x*) == *y*.

9. If Type(*y*) is Boolean, return the result of the comparison *x* == ToNumber(*y*).
10. If Type(*x*) is either String, Number, or Symbol and Type(*y*) is Object, then return the result of the comparison *x* == ToPrimitive(*y*).
11. If Type(*x*) is Object and Type(*y*) is either String, Number, or Symbol, then return the result of the comparison ToPrimitive(*x*) == *y*.
12. Return false.

算法流程翻译过来就是下面的：

1.如果x非正常值(比如x本身会抛出错误)，则中断执行

2.如果y非正常值(同上)，则中断执行

3.如果x的数据类型和y的数据类型相同，则返回以严格运算符执行判断的结果，即x===y的结果

4.如果x是null，y是undefined，返回true

5.如果x是undefined，y是null，返回true

6.如果x的数据类型是Number，y的数据类型是String，则将y转成Number，然后返回x==toNumber(y)的结果

7.如果x的数据类型是String，y的数据类型是Number，则将x转成Number，然后返回toNumber(x)==y的结果

8.如果x的数据类型是Boolean，则将x转成Number，然后返回toNumber(x)==y的结果

9.如果y的数据类型是Boolean，则将y转成Number，然后返回x==toNumber(y)的结果

10.如果x的数据类型是String、Number或者Symbol，y的数据类型是Object，则将y转成原始类型，然后返回x==toPrimitive(y)的结果

11.如果x的数据类型是Object，y的数据类型是String、Number或者Symbol，则将x转成原始类型，然后返回toPrimitive(x)==y的结果

12.返回false

二、规范文档中toNumber方法的解释

上面提到的toNumber又是个啥呢，是不是又云里雾里的？没关系，规范文档里面写了的！！我们接着看。

Argument Type	Result
Completion Record	If argument is an abrupt completion, return argument. Otherwise return ToNumber(argument.[[value]]).
Undefined	Return NaN.
Null	Return +0.
Boolean	Return 1 if argument is true. Return +0 if argument is false.
Number	Return argument (no conversion).
String	See grammar and conversion algorithm below.
Symbol	Throw a TypeError exception.
Object	Apply the following steps: 1. Let *primValue* be ToPrimitive(*argument*, hint Number). 2. Return ToNumber(*primValue*).

来来来，翻译走一波~

参数类型	结果
完成标志(例如return、break、throw等)	如果参数是一个异常中断，就返回这个参数，否则就返回该参数转换成Number之后的数值
Undefined	返回Nan
Null	返回+0
Boolean	如果参数是true，返回1；如果参数是false，返回+0
Number	返回参数(不做转换)
String	看本文第三节
Symbol	抛出一个TypeError异常

Object	采用下述的步骤： 1.利用ToPrimitive(argument,hint Number)的方式转成原始类型 2.将上述步骤的原始类型转成数值，即ToNumber(primValue)，并返回该数值

等等好像又冒出来了个ToPrimitive，What？？？别急别急，一步一来，我们第四节来介绍它，先介绍String转Number。

三、规范文档中String转Number方法的解释

还是按照惯例，规范文档的传送门送上。规范文档里面东西有点多，就不复制了，这里简要的介绍一下。

1.如果字符串中只包含数字（包括前面带加号或负号的情况），则将其转换为十进制数值，即"1"会变成1，"123"会变成123，而"011"会变成11（注意：前导的零被忽略了）；

2.如果字符串中包含有效的浮点格式，如"1.1"，则将其转换为对应的浮点数值（同样，也会忽略前导零）；

3.如果字符串中包含有效的十六进制格式，例如"0xf"，则将其转换为相同大小的十进制整数值；

4.如果字符串是空的（不包含任何字符），则将其转换为0；

5.如果字符串中包含除上述格式之外的字符，则将其转换为NaN。

四、规范文档中ToPrimitive方法的解释

原文传送门。

ToPrimitive (input [, PreferredType])

Input Type	Result
Completion Record	If input is an abrupt completion, return input. Otherwise return ToPrimitive(input.[[value]]) also passing the optional hint *PreferredType*.
Undefined	Return input.
Null	Return input.
Boolean	Return input.
Number	Return input.

String	Return input.
Symbol	Return input.
Object	Perform the steps following this table.

这个表格的东西不多，就不翻译了，说说Object的情况下的转换步骤。

原文是下面是这样的：

When Type(*input*) is Object, the following steps are taken:

1. If *PreferredType* was not passed, let *hint* be "default".
2. Else if *PreferredType* is hint String, let *hint* be "string".
3. Else *PreferredType* is hint Number, let *hint* be "number".
4. Let *exoticToPrim* be GetMethod(*input*, @@toPrimitive).
5. ReturnIfAbrupt(*exoticToPrim*).
6. If *exoticToPrim* is not undefined, then
    1. Let *result* be Call(*exoticToPrim*, *input*, «*hint*»).
    2. ReturnIfAbrupt(*result*).
    3. If Type(*result*) is not Object, return *result*.
    4. Throw a TypeError exception.
7. If *hint* is "default", let *hint* be "number".
8. Return OrdinaryToPrimitive(*input*,*hint*).

When the abstract operation OrdinaryToPrimitive is called with arguments O and hint, the following steps are taken:

1. Assert: Type(*O*) is Object
2. Assert: Type(*hint*) is String and its value is either "string" or "number".
3. If *hint* is "string", then

1. Let *methodNames* be «"toString", "valueOf"».

4. Else,
    1. Let *methodNames* be «"valueOf", "toString"».
5. For each *name* in *methodNames* in List order, do
    1. Let *method* be Get(*O*, *name*).
    2. ReturnIfAbrupt(*method*).
    3. If IsCallable(*method*) is true, then
        1. Let *result* be Call(*method*, *O*).
        2. ReturnIfAbrupt(*result*).
        3. If Type(*result*) is not Object, return *result*.
6. Throw a TypeError exception.

NOTEWhen ToPrimitive is called with no hint, then it generally behaves as if the hint were Number. However, objects may over-ride this behaviour by defining a @@toPrimitive method. Of the objects defined in this specification only Date objects (see 20.3.4.45) and Symbol objects (see 19.4.3.4) over-ride the default ToPrimitive behaviour. Date objects treat no hint as if the hint were String.

东西有点多，简而概之。

JS引擎内部转换为原始值ToPrimitive(obj,preferredType)函数接受两个参数，第一个obj为被转换的对象，第二个preferredType为希望转换成的类型（默认为空，接受的值为Number或String）

在执行ToPrimitive(obj,preferredType)时如果第二个参数为空并且obj为Date的实例时，此时preferredType会被设置为String，其他情况下preferredType都会被设置为Number

如果preferredType为Number，ToPrimitive执行过程如下：

1. 如果obj为原始值，直接返回；
2. 否则调用 obj.valueOf()，如果执行结果是原始值，返回之；
3. 否则调用obj.toString()，如果执行结果是原始值，返回之；

4. 否则抛异常。

如果preferredType为String，将上面的第2步和第3步调换，即：

1. 如果obj为原始值，直接返回；
2. 否则调用obj.toString()，如果执行结果是原始值，返回之；
3. 否则调用 obj.valueOf()，如果执行结果是原始值，返回之；
4. 否则抛异常。

五、举个例子

看上面的东西难免有点乏味，还是举几个例子会更形象。

例子一

'true'==true

先不给出答案，我们一步一步看。

在这个相等运算中，左侧'true'的数据类型是String，右侧true的数据类型是Boolean。

首先满足第9条，所以布尔值true转成数值1，返回'true'==1的值；

其次'true'==1又满足第7条，所以字符串true根据上面讲的规则，转换成Nan，故返回NaN==1；

然后NaN都不等于任何值，包括它本身，即NaN==NaN返回false；

所以最后'true'==true返回false。

例子二

0==null

在这个相等运算中，左侧0的数据类型是Number，右侧null的数据类型是Null（规范文档4.3.13节规定，内部Type运算的结果，与typeof运算符无关），所以根据上面的规则，前面11条都不满足，直到第12步才返回false。