**关注”千锋互联”微信公众号( ID: qianfengjiaoyu ),更多学习资源(电子书/面试题/简历模板/视频/公开课等等)与你分享**

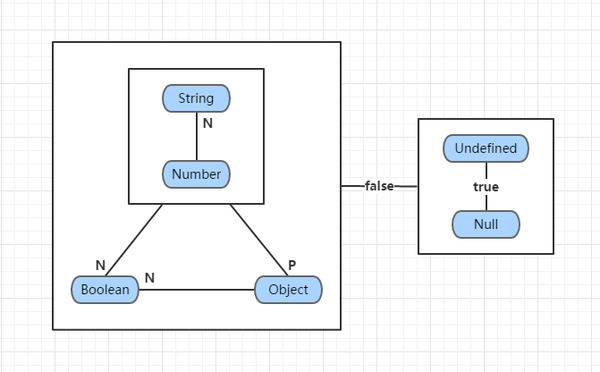


**JavaScript中的==,你真懂?**

大家知道，==是JavaScript中比较复杂的一个运算符。它的运算规则奇怪，容易让人犯错，从而成为JavaScript中“最糟糕的特性”之一。

在仔细阅读了ECMAScript规范的基础上，我画了一张图，我想通过它你会彻底地搞清楚关于==的一切。同时，我也试图通过此文向大家证明==并不是那么糟糕的东西，它很容易掌握，甚至看起来很合理。

**先上图：**



**图1** ==运算规则的图形化表示

==运算规则的精确描述在此：[The Abstract Equality Comparison Algorithm](https://link.zhihu.com/?target=http%3A//es5.github.io/%23x11.9.3)。但是，这么复杂的描述，你确定看完后脑子不晕？确定立马就能拿它指导实践？

肯定不行，规范毕竟是给JavaScript运行环境的开发人员看的(比如V8引擎的开发人员们)，而不是给语言的使用者看的。而上图正是将规范中复杂的描述翻译成了更容易看懂的形式。

在详细介绍图1中的每个部分前，我们来复习一下JS中关于类型的知识：

1. JS中的值有两种类型：原始类型(Primitive)、对象类型(Object)。
2. 原始类型包括：Undefined、Null、Boolean、Number和String等五种。
3. Undefined类型和Null类型的都只有一个值，即undefined和null；Boolean类型有两个值：true和false；Number类型的值有很多很多；String类型的值理论上有无数个。
4. 所有对象都有valueOf()和toString()方法，它们继承自Object，当然也可能被子类重写。

现在考虑表达式：

x == y

其中x和y是上述六种类型中某一种类型的值。

当x和y的类型相同时，x == y可以转化为x === y，而后者是很简单的(唯一需要注意的可能是NaN)，所以下面我们只考虑x和y的类型不同的情况。

**一. 有和无**

在图1中，JavaScript值的六种类型用蓝底色的矩形表示。它们首先被分成了两组：

* String、Number、Boolean和Object (对应左侧的大矩形框)
* Undefined和Null (对应右侧的矩形框)

分组的依据是什么？我们来看一下，右侧的Undefined和Null是用来表示**不确定**、**无**或者**空**的，而右侧的四种类型都是**确定的**、**有**和**非空**。我们可以这样说：

左侧是一个**存在**的世界，右侧是一个**空**的世界。

所以，左右两个世界中的任意值做==比较的结果都是false是很合理的。(见图1中连接两个矩形的水平线上标的false)

**二. 空和空**

JavaScript中的undefined和null是另一个经常让我们崩溃的地方。通常它被认为是一个设计缺陷，这一点我们不去深究。不过我曾听说，JavaScript的作者最初是这样想的：

假如你打算把一个变量赋予对象类型的值，但是现在还没有赋值，那么你可以用null表示此时的状态(证据之一就是typeof null 的结果是'object')；相反，假如你打算把一个变量赋予原始类型的值，但是现在还没有赋值，那么你可以用undefined表示此时的状态。

不管这个传闻是否可信，它们两者做==比较的结果是true是很合理的。(见图1中右侧垂直线上标的true)

在进行下一步之前，我们先来说一下图1中的两个符号：大写字母N和P。这两个符号并不是PN结中正和负的意思。而是：

* N表示ToNumber操作，即将操作数转为数字。它是规范中的抽象操作，但我们可以用JS中的Number()函数来等价替代。
* P表示ToPrimitive操作，即将操作数转为原始类型的值。它也是规范中的抽象操作，同样也可以翻译成等价的JS代码。不过稍微复杂一些，简单说来，对于一个对象obj：

ToPrimitive(obj)等价于：先计算obj.valueOf()，如果结果为原始值，则返回此结果；否则，计算obj.toString()，如果结果是原始值，则返回此结果；否则，抛出异常。

注：此处有个例外，即Date类型的对象，它会先调用toString()方法，后调用valueOf()方法。

在图1中，标有N或P的线表示：当它连接的两种类型的数据做==运算时，标有N或P的那一边的操作数要先执行ToNumber或ToPrimitive变换。

**三. 真与假**

从图1可以看出，当布尔值与其他类型的值作比较时，布尔值会转化为数字，具体来说

true -> 1

false -> 0

这一点也不需浪费过多口舌。想一下在C语言中，根本没有布尔类型，通常用来表示逻辑真假的正是整数1和0。

**四. 字符的序列**

在图1中，我们把String和Number类型分成了一组。为什么呢？在六种类型中，String和Number都是字符的序列(至少在字面上如此)。字符串是所有合法的字符的序列，而数字可以看成是符合特定条件的字符的序列。所以，数字可以看成字符串的一个子集。

根据图1，在字符串和数字做==运算时，需要使用ToNumber操作，把字符串转化为数字。假设x是字符串，y是数字，那么：

x == y -> Number(x) == y

那么字符串转化为数字的规则是怎样的呢？规范中描述得很复杂，但是大致说来，就是把字符串两边的空白字符去掉，然后把两边的引号去掉，看它能否组成一个合法的数字。如果是，转化结果就是这个数字；否则，结果是NaN。例如：

Number('123') // 结果123

Number('1.2e3') // 结果1200

Number('123abc') // 结果NaN

Number('\r\n\t123\v\f') // 结果123

当然也有例外，比如空白字符串转化为数字的结果是0。即

Number('') // 结果0

Number('\r\n\t \v\f') // 结果0

**五. 单纯与复杂**

原始类型是一种单纯的类型，它们直接了当、容易理解。然而缺点是表达能力有限，难以扩展，所以就有了对象。对象是属性的集合，而属性本身又可以是对象。所以对象可以被构造得任意复杂，足以表示各种各样的事物。

但是，有时候事情复杂了也不是好事。比如一篇冗长的论文，并不是每个人都有时间、有耐心或有必要从头到尾读一遍，通常只了解其中心思想就够了。于是论文就有了关键字、概述。JavaScript中的对象也一样，我们需要有一种手段了解它的主要特征，于是对象就有了toString()和valueOf()方法。

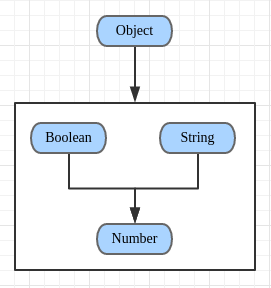
toString()方法用来得到对象的一段文字描述；而valueOf()方法用来得到对象的特征值。

当然，这只是我自己的理解。顾名思义，toString()方法倾向于返回一个字符串。那么valueOf()方法呢？根据[规范中的描述](https://link.zhihu.com/?target=http%3A//es5.github.io/%23x9.1)，它倾向于返回一个数字——尽管内置类型中，valueOf()方法返回数字的只有Number和Date。

根据图1，当一个对象与一个非对象比较时，需要将对象转化为原始类型(虽然与布尔类型比较时，需要先将布尔类型变成数字类型，但是接下来还是要将对象类型变成原始类型)。这也是合理的，毕竟==是不严格的相等比较，我们只需要取出对象的主要特征来参与运算，次要特征放在一边就行了。

**六. 万物皆数**

我们回过头来看一下图1。里面标有N或P的那几条连线是没有方向的。假如我们在这些线上标上箭头，使得连线从标有N或P的那一端指向另一端，那么会得到(不考虑undefined和null)：



**图2** ==运算过程中类型转化的趋势

发现什么了吗？对，在运算过程中，所有类型的值都有一种向数字类型转化的趋势。毕竟曾经有名言曰：

万物皆数。

**七. 举个栗子**

前面废话太多了，这里还是举个例子，来证明图1确实是方便有效可以指导实践的。

例，计算下面表达式的值：

[''] == false

首先，两个操作数分别是对象类型、布尔类型。根据图1，需要将布尔类型转为数字类型，而false转为数字的结果是0，所以表达式变为：

[''] == 0

两个操作数变成了对象类型、数字类型。根据图1，需要将对象类型转为原始类型：

* 首先调用[].valueOf()，由于数组的valueOf()方法返回自身，所以结果不是原始类型，继续调用[].toString()。
* 对于数组来说，toString()方法的算法，是将每个元素都转为字符串类型，然后用逗号','依次连接起来，所以最终结果是空字符串''，它是一个原始类型的值。

此时，表达式变为：

'' == 0

两个操作数变成了字符串类型、数字类型。根据图1，需要将字符串类型转为数字类型，前面说了空字符串变成数字是0。于是表达式变为：

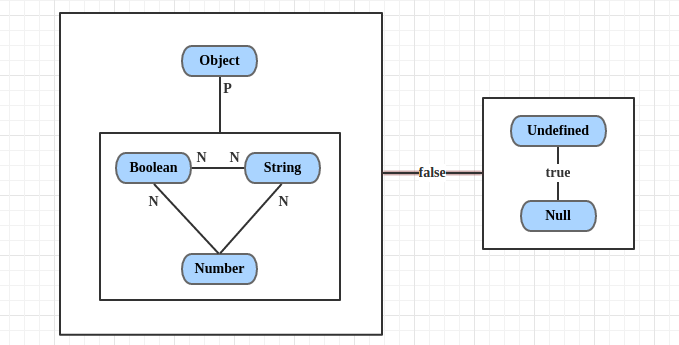
0 == 0

到此为止，两个操作数的类型终于相同了，结果明显是true。

从这个例子可以看出，要想掌握==运算的规则，除了牢记图1外，还需要记住那些内置对象的toString()和valueOf()方法的规则。包括Object、Array、Date、Number、String、Boolean等，幸好这没有什么难度。

**八. 再次变形**

其实，图一还不够完美。为什么呢？因为对象与字符串/数字比较时都由对象来转型，但是与同样是原始类型的布尔类型比较时却需要布尔类型转型。实际上，只要稍稍分析一下，全部让对象来转为原始类型也是等价的。所以我们得到了最终的更加完美的图形：



**图3** 更完美的==运算规则的图形化表示

有一个地方可能让你疑惑：为什么Boolean与String之间标了两个N？虽然按照规则应该是由Boolean转为数字，但是下一步String就要转为数字了，所以干脆不如两边同时转成数字。

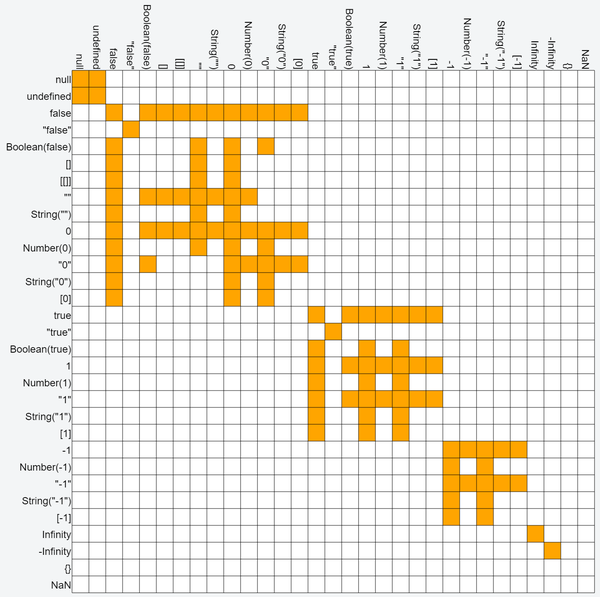
**九. 总结一下**

前面说得很乱，根据我们得到的最终的图3，我们总结一下==运算的规则：

* **undefined == null**，结果是**true**。且它俩与所有其他值比较的结果都是**false**。
* **String == Boolean**，需要两个操作数同时转为Number。
* **String/Boolean == Number**，需要String/Boolean转为Number。
* **Object == Primitive**，需要Object转为Primitive(具体通过**valueOf**和**toString**方法)。

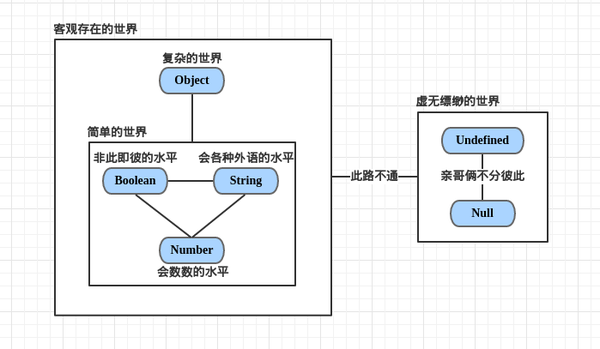
瞧见没有，一共**只有4条规则**！是不是很清晰、很简单。

最后，当时一位大神贴了一张图：

我看后觉得太复杂了，于是想能不能用一种更简单的方式来描述一下==运算，使大家更清晰更容易掌握。于是就有了此文，当然我不知道自己成功了没有。

OK，结束了。文章中的谬误，请不吝指出。

PS：最后，把图改了一下，仅供娱乐 : )



**关注”千锋互联”微信公众号( ID: qianfengjiaoyu ),更多学习资源(电子书/面试题/简历模板/视频/公开课等等)与你分享**

