JS菌 6.2k

🍒 图解 == 操作符规则和不同类型间转换规则

javascript 发布于 2019-03-11

图解 == 操作符规则和不同类型间转换规则

很多人包括我在内很抵触这种问题😡，因为很长一段时间我一直弄不明白 == 和 === 到底是怎么个规则。如果你也没闹明白 == 和 ===，读了这篇文章应该至少不会见到这俩操作符就觉得恶心了吧🤢。

> 另外需要注意的是，== 的英文名叫 Abstract Equality Comparison；=== 则是 Strict Equality Comparison。

废话不多说，我们开始搞起

== 操作符

== 操作符基本规则

首先需要注意的是

- 如果要比较的两个项是同种类型的，那么 == 就会返回 === 操作符的执行结果。举个例子🌰 `2 == 3` 最后会返回 `2 === 3` 的执行结果
- 如果要比较的两个项是不同类型的，== 就会对其中一个或两者都进行类型转换然后再比较。比如 `2 == '3'` 就会变成 `2 == 3` 最后会比较 `2 === 3`

这就是最基本的规则

== 操作符具体的转化规则

然后我们再来看看具体的转换规则⬇️：

整体流程概览

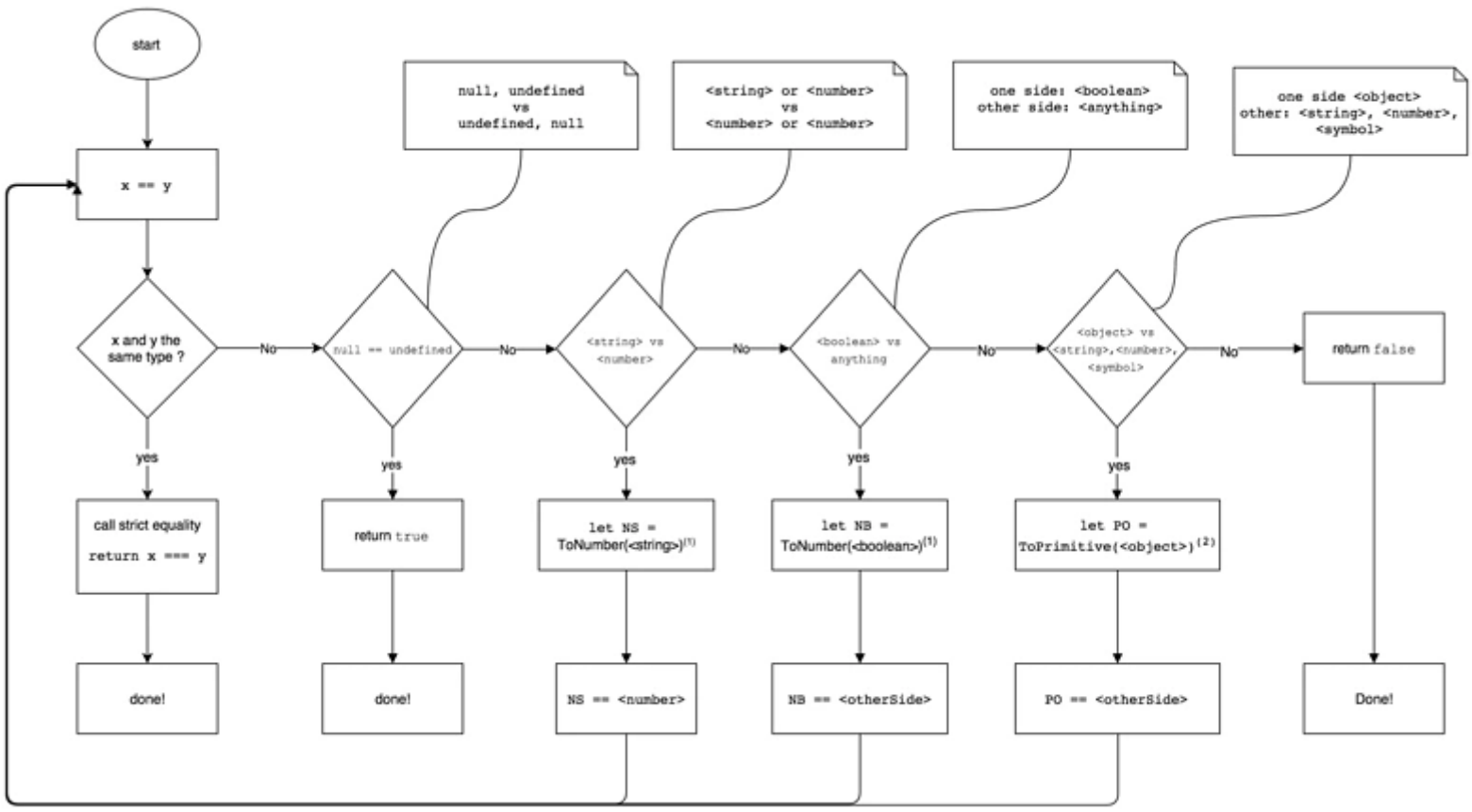
1. 如果类型相同，调用 `===` 操作符
2. 如果类型不同，尝试类型转换
    - 1. 查看是否是 `undefined` 和 `null` 比较
        - ✅ 返回 `true`
        - ⬇️ 如果不是继续下一条规则
    - 1. 是否在比较 `string` 和 `number`
        - ✅ 如果是，那么将 `string` 转为 `number` 并回到最初重新比较 ♻️
        - ⬇️ 如果不是继续下一条规则
    - 1. 查看我们比较的项中是否有 `boolean`
        - ✅ 如果有，那么将 `boolean` 转为 `number` 并回到最初重新比较 ♻️
        - ⬇️ 如果不是继续下一条规则
    - 1. 查看是否有一项是 `object`
        - ✅ 如果有，那么将 `object` 转为其原始值 `primitive` 并回到最初重新比较 ♻️
        - ❌ 如果还不是，只能返回 `false` 了💩

举几个🌰：

```
console.log(undefined == null);  true
// 匹配 2-1 规则
console.log('123' == 123);  true
// 匹配 2-2 规则：将转换为 123 == 123 然后再比较匹配 1 规则
console.log(false == '0');  true
// 匹配 2-3 规则：将转换为 0 == '0'；然后匹配 2-2 规则：将转换为 0 == 0；然后匹配 1 规则
console.log({name: 'oli'} == '[object Object]');  true
// 匹配 2-4 规则：将获取对象原始值并比较，转换为 '[object Object]' == '[object Object]'；然后匹配 1 规则
console.log(['0'] == 0);  true
// 匹配 2-4 规则：将获取数组对象原始值并比较，转换为 '0' == 0；然后匹配 2-2 规则，转为 0 == 0；最后匹配 1 规则
```

这么看来转换规则是不是很清晰明了😳

附上一张转换规则图，忘记了就看看，当然正常情况下应该用 === 代替 == 避免不必要的麻烦：



> ecma 的规范：http://www.ecma-international...

类型转换

上述在比较的过程中，涉及到类型的转换，如字符串转整数、布尔值转整数、以及获取对象原始值等等。了解一下这些不同类型之间是如何转换的：

获取对象原始值

接着我们再来研究一下对象怎么转换为原始值的：

> 我们需要知道转换类型的这个方法在 JS 源代码中是 `ToPrimitive` 这个方法，该方法有一个可选参数 `PreferredType`，这个参数的作用是指定期望类型；如果第一个参数对应的对象可以被转换为不止一种类型，那么后者可以作为一种暗示，表示该对象应该转换为那种类型

- 1. 默认情况下（期望类型默认为 `number`）
  - 1. 调用 `valueOf` 方法：
    - ✅ 如果返回的是原始值，那么就用这个
    - ⬇️ 如果返回的不是原始值，那么跳到下一步
  - 1. 调用 `toString` 方法：
    - ✅ 如果返回的是原始值，那么就用这个
    - ❌ 否则报错💩
- 1. 如果期望类型为 `string`：
  - 1. 调用 `toString` 方法：
    - ✅ 如果返回的是原始值，那么就用这个
    - ⬇️ 如果返回的不是原始值，那么跳到下一步
  - 1. 调用 `valueOf` 方法：
    - ✅ 如果返回的是原始值，那么就用这个

        - ❌ 否则报错💩
- 1. 如果对象是 Date 类型（期望类型为 `string`）：
    - 1. 调用 `toString` 方法：
        - ✅ 如果返回的是原始值，那么就用这个
        - ⬇️ 如果返回的不是原始值，那么跳到下一步
    - 1. 调用 `valueOf` 方法：
        - ✅ 如果返回的是原始值，那么就用这个
        - ❌ 否则报错💩

简单的说就是默认调用 `valueOf` 方法，然后是 `toString` 方法；如果对象是 `Date` 类型或对象的期望类型为 `string`，那么先调用 `toString` 方法😪

举几个🌰🌰🌰吧：

```
...
1   obj = {}
2   console.log(typeof obj.valueOf(), obj);  object {}
3   // 默认调用 valueOf 发现并不是原始值
4   console.log(typeof obj.toString(), obj);  string {}
5   // 然后调用 toString 方法
6
```

普通的对象，首先调用 valueOf 方法，返回的结果并非原始值，那么会调用 toString 方法

```
3
4   obj1 = {}
5   Object.prototype.valueOf = function() { return 'custom obj primitive' }
6   console.log(typeof obj1.valueOf(), obj1.valueOf())  string custom obj primitive
7   console.log(typeof obj1.toString(), obj1.toString())  string [object Object]
8   console.log(obj1 == 'custom obj primitive')  true
9
```

假设我们重写 valueOf 方法，valueOf 和 toString 同时返回 string 原始值。使用 == 操作符可以看出，对象还是优先使用了 valueOf 方法返回的值

```
3
4   obj1 = {}
5   Object.prototype.valueOf = function() { return 'custom obj primitive' }
6   console.log(typeof obj1.valueOf(), obj1.valueOf())  string custom obj primitive
7   console.log(typeof obj1.toString(), obj1.toString())  string [object Object]
8   console.log(obj1 == 'custom obj primitive')  true
9
```

上面的数组同理，首先默认调用 valueOf 方法，如不是原始值，则调用 toString 方法

```
3
4   arr = [1, '123', [1], {}]
5   console.log(typeof arr.valueOf(), arr.valueOf())  object [ 1, '123', [ 1 ], {} ]
6   console.log(typeof arr.toString(), arr.toString())  string 1,123,1,[object Object]
7   console.log(arr == '1,123,1,[object Object]')  true
8
```

这个包括众多类型的项的数组也是一样👀

```
4   date = new Date(1552213627379)
5   console.log(typeof date.valueOf(), date.valueOf())  number 1552213627379
6   console.log(typeof date.toString(), date.toString())  string Sun Mar 10 2019 18:27:07 GMT+0800 (CST)
7   console.log(date == 'Sun Mar 10 2019 18:27:07 GMT+0800 (CST)')  true
```

再看看 Date 类型，他的期望类型是 string 因此首先调用的是 toString 方法，该方法返回一个原始值，那么就是用这个原始值

转换为 number

下面我们来看看转换成 number 类型的规则：

1. `undefined` 👉 `NaN` 如果是 undefined 则直接转换成 NaN
2. `null` 👉 `0` 如果是 null 则转换成 0
3. `boolean` 👉 `0/1` 如果是 boolean 则转换成 0 或 1
4. `string` 👉 `0/NaN/(parse to number)` 如果是 string 则转换成对应的 number，空字符串转换为 0，无法转换的则为 NaN
5. `object` 👉 首先获取原始值然后再转为 number

看几个🌰：

```
1   console.log(undefined / 2);  NaN
2   // undefined 无法转换成具体 number 因此转为 NaN
3   console.log(null + 1);  1
4   // null 将转为 0，这里的结果则为 0 + 1
5   console.log(true / 2);  0.5
6   // true 转为 1
7   console.log('10' ** 2);  100
8   // '10' 字符串可以转为 number 这里变为 10
9   console.log('29x' / 2);  NaN
10  // 左边字符串无法转为 number 这里变为 NaN
11  console.log({} / 1);  NaN
12  // 对象调用 toString 方法转为字符串；字符串无法转为 number 因此为 NaN
13  console.log(new Date() * 1, new Date().getTime());  1552232222101 1552232222101
14  // Date 调用 getTime 方法转为 number
15
```

转换为 string

转为 string 的规则为：

1. `undefined` 👉 `'undefined'`
2. `null` 👉 `'null'`
3. `number` 👉 `'number`
4. `boolean` 👉 `'true'/'false'`
5. `object` 👉 首先获取原始值，然后转为 string

```
1   console.log(`res: ${undefined}`);  res: undefined
2   // undefined 转为 string 类型为 'undefined'
3   console.log(`res: ${null}`);  res: null
4   // null 转为 string 类型为 'null'
5   console.log(`res: ${12e200}`);  res: 1.2e+201
6   // number 转为 string 类型为 'number'
7   console.log(`res: ${{}}`);  res: [object Object]
8   // 对象转为 string 类型调用 toString 方法
9   console.log(`res: ${new Date()}`);  res: Sun Mar 10 2019 23:48:39 GMT+0800 (CST)
10  // Date 日期类型转为 string 调用 toString 方法
11
```

转为 boolean

常见的问题：哪些是 falsy 哪些是 truthy：

❌下面这些在 JS 中都为 falsy 除此之外的都是 truthy

1. `undefined` 👉 falsy
2. `null` 👉 falsy
3. `0` 👉 falsy
4. `""` 👉 falsy
5. `NaN` 👉 falsy

因此转换规则如下：

1. `undefined` 👉 `false`
2. `null` 👉 `false`
3. `number` 👉 当为 0 时 `false` 否则为 `true`

4. `string` 👉 当为空字符串时为 `false` 否则为 `true`
5. `object` 👉 `true`
6. `array` 👉 `true`
7. `Date` 👉 `true`

🦐🍜是几个例子：

```
1   console.log(undefined || 'done');  done
2   // undefined 为 false
3   console.log(null || 'done');  done
4   // null 为 false
5   console.log(0 || 123 || 'done');  123
6   // number 不为 0 则 true; 为 0 则 false
7   console.log('' || 'str' || 'done');  str
8   // string 不为空则 true 否则为 false
9   console.log({} || 'done');  {}
10  // 对象则为 true
11  console.log([] || 'done');  []
12  // 数组也为 true
13  console.log(new Date() || 'done');  Mon Mar 11 2019 00:00:35 GMT+0800 (CST)
14  // Date 为 true
```

附上一张不同类型间转换规则：

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	

就写到这里，基本上 == 和类型转换就是这个样子❗

EOF

参考：

- http://www.cnblogs.com/178mz/...
- http://www.ecma-international...
- https://www.codementor.io/jav...

扫码或搜索 JS菌 关注公众号

专注前端 共同成长

阅读 476 • 发布于 2019-03-11

举报

赞 7 | 收藏 4 | ¥ 赞赏 | 分享

JS菌

6.2k

关注作者

0 条评论

得票 • 时间

撰写评论 ...

提交评论

推荐阅读

使用JavaScript隐式类型转换输出"nb"

本文将介绍一段使用JavaScript隐式类型转换输出"nb"的代码，并讲解具体的转换过程。 预备知识 请先阅读文章ECMAScript7规...

luckness • 阅读 657 • 8 赞

为什么不要在 JavaScript 中使用位操作符?

如果你的第一门编程语言不是 JavaScript，而是 C++ 或 Java，那么一开始你大概会看不惯 JavaScript 的数字类型。在 JavaScript...

JerryZou • 阅读 3.7k • 6 赞 • 2 评论

细说 Javascript 类型篇（二）：typeof 操作符

typeof 操作符（还有 instanceof）可能是 Javascript 设计中最大缺陷，因为它几乎是完全破损的。由于 typeof 用法与调用函数的...

StephenLi • 阅读 3.2k • 3 赞 • 3 评论

JavaScript中new操作符的详细过程

理解new对象过程，需要提前了解原型及原型链的相关知识我们都知道，JS当中创建对象使用的是原型设计模式，即使用new操作...

有鱼是只猫 • 阅读 427 • 2 赞

JavaScript剩余操作符Rest Operator

定义函数的时候，如果函数的参数以... 为前缀，则改参数是剩余参数（rest parameter）。剩余参数表示参数个数不确定的参数列...

netcy • 阅读 406 • 2 赞

细说 Javascript 类型篇（三）：instanceof 操作符

Javascript 的 instanceof 操作符可以用来比较两个操作数的构造函数 constructor。但这个只有在比较自定义对象才有意义。当用...

StephenLi • 阅读 2.5k • 2 赞 • 1 评论

JavaScript的“&&”和“||”操作符总结

&&和||操作符链接的两个值最后取哪个值的问题，有点模糊和不好理解，比如下面的表达式输出什么？如果你能答对说明你对这个...

杜尼卜 • 阅读 1.9k • 1 赞

JavaScript漫谈之理解类型操作符typeof

在本文中，将简述JavaScript类型系统和数据类型，以及如何使用typeof操作符执行类型检查。还讲解了使用typeof操作符进行某...

sueRimn • 阅读 343 • 1 赞

前端和Node学习笔记

用户专栏

前端和Node学习笔记

1443 人关注　**191** 篇文章

关注专栏　专栏主页